POLYTECHNIC UNIVERSITY OF THE PHILIPPINES

Computer Engineering College of Engineering Mabini Campus, Sta. Mesa

CMPE 30274

EMBEDDED SYSTEMS

Engr. Julius S. Cansino
Engr. Adelino T. Racusa

Embedded Systems

CMPE 30274

Engr. Julius S. Cansino
Engr. Adelino Racusa

Published and distributed by:

Polytechnic University of the Philippines

Address : 1016 Anonas St., Brgy. 630, Sampaloc, Sta. Mesa,
Manila Website : https://www.pup.edu.ph
Email Address : inquire@pup.edu.ph
Tel. No. **:** (+63 2) 5335-1PUP (5335-1787) or 5335-1777

The VMPGO

VISION

PUP: The National Polytechnic University

MISSION

Ensuring inclusive and equitable quality education and promoting lifelong learning opportunities through a re-engineered polytechnic university by committing to:

- provide democratized access to educational opportunities for the holistic development of individuals with a global perspective.
- offer industry-oriented curricula that produce highly skilled professionals with managerial and technical capabilities and a strong sense of public service for nation-building.
- embed a culture of research and innovation.
- continuously develop faculty and employees with the highest level of professionalism.
- engage public and private institutions and other stakeholders for the attainment of social development goals.
- establish a strong presence and impact in the international academic community.

PHILOSOPHY

As a state university, the Polytechnic University of the Philippines believes that:

- Education is an instrument for the development of the citizenry and for the enhancement of nation-building; and,
- That meaningful growth and transformation of the country are best achieved in an atmosphere of brotherhood, peace, freedom, justice and nationalist-oriented education imbued with the spirit of humanist internationalism.

SHARED VALUES AND PRINCIPLES

1. Integrity and Accountability
2. Nationalism
3. Sense of Service
4. Passion for Learning and Innovation
5. Inclusivity
6. Respect for Human Rights and the Environment
7. Excellence
8. Democracy

GOALS OF THE COLLEGE/CAMPUS

1. Provide quality education through instruction, advance research and extension services

2. Produce world-class professionals as potential industry leaders and job providers

3. Develop and produce facilities using adapted technology and indigenous materials

4. Maintain, upgrade or improve facilities through the applications of engineering technology

PROGRAM DESCRIPTION

Computer Engineering is a four-year degree program that deals with the study of computer systems. The curriculum covers both software and hardware and develops the student's ability to analyze computer systems, designs, construction of electronic equipment and its peripherals. Since computer science is directed to the theory and technology of computation, the curriculum does not specialize along traditional lines that divide hardware and software, systems and applications, or theory and experiment. Rather, a unified approach to the design and analysis of computers and of computing structures is employed. This background prepares the student for placements as computer engineers in the government industry. It also qualifies them for related jobs with computer manufacturers and consulting firms as systems programmers as well as application programmers with scientific, research, and business organizations. Ethical considerations with respect to the profession is an important component of the program of study.

COURSE DESCRIPTION

This course provides a comprehensive introduction to embedded systems, focusing on the design, implementation, and application of these systems in various industries. Students will explore the fundamental concepts of embedded systems, including hardware and software components, real-time operating systems, and interfacing techniques.

INSTITUTIONAL LEARNING OUTCOMES (ILOS)

A PUP graduate imbibed:

1. Creative and Critical Thinking
2. Adeptness in the Responsible Use of Technology
3. Strong Service Orientation
4. High Level of Leadership and Organizational Skills
5. Community Engagement
6. Effective Communication
7. Sense of Personal and Professional Ethics
8. Passion for Life-long Learning
9. Sense of National and Global Responsiveness

PROGRAM LEARNING OUTCOMES (PLOS)

1. Employed in the field of Network Engineering or Big Data or Machine Learning or System Development or other related fields

2. Committed members of professional and other allied organizations engaged in community development and nation building

3. Involved in continuous training and development in current or emerging trends, and advancement in their chosen field of specialization.

COURSE LEARNING OUTCOMES (CLOS)

By the end of this course, students will be able to:

- Understand Embedded System Fundamentals: Explain the core concepts, architecture, and significance of embedded systems in various applications.

- Design Embedded Systems: Develop and implement embedded systems using microcontrollers, microprocessors, and real-time operating systems (RTOS).

- Interface with Peripherals: Integrate embedded systems with external peripherals such as sensors, actuators, and communication modules.

- Develop Software for Embedded Systems: Write and optimize software for embedded systems, ensuring efficient and reliable operation.

- Debug and Test Embedded Systems: Apply debugging and testing techniques to identify and resolve issues in embedded systems.

- Apply Embedded Systems in Real-World Scenarios: Utilize embedded system concepts to solve practical problems in industries such as automotive, healthcare, consumer electronics, and more.

- Evaluate System Performance: Assess the performance and reliability of embedded systems, making necessary improvements to meet design specifications.

Preface

Welcome to the world of Embedded Systems! This lecture manual is designed to guide you through the fascinating and rapidly evolving field of embedded systems, which are integral to countless modern technologies. Whether you are a student, an engineer, or an enthusiast, this manual aims to provide you with a solid foundation and practical insights into the design, implementation, and application of embedded systems.

Embedded systems are specialized computing systems that perform dedicated functions within larger systems. They are found in a wide range of applications, from consumer electronics and automotive systems to industrial automation and medical devices. Understanding embedded systems is crucial for anyone looking to innovate and excel in today's technology-driven world.

This instructional manual is structured to offer a comprehensive learning experience, combining theoretical knowledge with hands-on practice. Each chapter is carefully crafted to build upon the previous one, ensuring a cohesive and progressive learning journey. You will explore key topics such as microcontrollers, real-time operating systems, peripheral interfacing, and system design, all of which are essential for mastering embedded systems.

Throughout this instructional manual, you will find detailed explanations, illustrative examples, and practical exercises designed to reinforce your understanding and skills. By the end of this course, you will be equipped with the knowledge and confidence to design, develop, and deploy embedded systems in various real-world scenarios.

We hope this instructional manual serves as a valuable resource in your educational and professional endeavors. Embrace the challenges and opportunities that come with learning about embedded systems, and let your curiosity and creativity drive your success.

TABLE OF CONTENTS

COURSE SYLLABUS

UNIT I

Introduction to Embedded Systems

INTRODUCTION

This module provides an overview of Embedded Systems, including its definition, characteristics, applications in different industries, key components, and design challenges. By the end of this module, students will be able to clearly define and discuss Embedded Systems.

LEARNING OBJECTIVES / OUTCOMES

At the end of this module, students are expected to:

- Define Embedded Systems and Describe their Characteristics
- Identify and Explain the Applications of Embedded Systems in Different Industries
- Recognize and Explain the Key Components of an Embedded Systems
- Analyze and Elaborate the Challenges and Considerations involved in Designing Embedded Systems

DISCUSSION OF THE LESSON

What is Embedded Systems?

Embedded systems are more than just a combination of hardware and software components. They are specialized computer systems designed to perform specific functions within larger systems or devices. Unlike general-purpose computers, embedded systems operate autonomously, require little human intervention, and are optimized for efficiency. These systems are found in various everyday devices such as washing machines, microwave ovens, ATMs, and toys, where they enhance functionality and provide seamless automation. In this introduction, we will explore the definition and characteristics of embedded systems, their applications in different industries, and the key components that make them work efficiently. Additionally, we will discuss the challenges and considerations involved in the design of embedded systems. By understanding the fundamentals of embedded systems, we can appreciate their pervasive presence and their vital role in shaping modern technology.

Efficiency is a paramount consideration in the design of embedded systems. These systems are purpose-built to accomplish specific tasks in the most optimized and effective way possible. They are engineered with a keen focus on resource utilization, power consumption, and performance to ensure maximum efficiency in their operation. By tailoring the hardware and software components to the specific requirements of the task at hand, embedded systems can achieve high levels of efficiency and effectiveness. Embedded systems are ubiquitous in our daily lives, integrated into various devices and appliances. Common examples include washing machines, microwave ovens, and ATMs, where embedded systems control and manage their respective functionalities. Even seemingly simple devices like toys often incorporate embedded systems to provide interactive features and enhanced functionality.

The presence of embedded systems in these devices is made possible by the integration of computing capabilities. These systems typically consist of microcontrollers or microprocessors that act as the core processing units, along with other essential components like memory, input/output interfaces, and sensors. The combination of these components enables the embedded system to acquire data, process information, and control external devices or systems as required by the specific application.

We can agree that embedded systems are specialized computer systems designed to fulfill specific functions within larger systems or devices. They operate autonomously, optimized for efficiency, and distinct from general-purpose computers. Their presence is widespread, powering a wide range of devices we encounter in our daily lives, and their integration is made possible through purpose-built hardware and software components.

Elaborating the Characteristics of Embedded Systems

Embedded systems possess several distinct characteristics that set them apart from general-purpose computers. Understanding these characteristics is crucial for comprehending their design and functionality. The key characteristics of embedded systems are as follows:

1. Task-Specific

   Embedded systems are designed to perform specific tasks repeatedly and continuously throughout their lifespan. For example, an MP3 player is dedicated solely to playing music.

2. Time-Constrained

   Embedded systems operate within predefined time limits. They must complete their tasks in a timely manner to ensure proper functionality and avoid potential consequences. For instance, a car brake system must respond within a specified time frame to prevent accidents.

3. Minimal User Interface

   Many embedded systems have minimal or no user interface (UI). Once programmed or configured, they operate autonomously without direct user interaction. An example is a fully automatic washing machine that performs its functions without constant user intervention.

4. Reactive to External Stimuli

   Some embedded systems are designed to respond and react to external stimuli. Devices like thermometers or GPS tracking devices gather data from their surroundings and adjust their behavior accordingly.

5. Efficiency-Oriented

   Embedded systems are built to achieve specific efficiency levels. They are typically compact, consume less power, and are cost-effective. These characteristics enable them to function optimally within the constraints of their intended applications.

6. Reliability and Stability

   Embedded systems are expected to be highly reliable and stable since users cannot easily change or upgrade their components. They are designed to operate for extended periods without experiencing issues or failures.

7. Use of Microcontrollers or Microprocessors

   Embedded systems are typically built using microcontrollers or microprocessors as their central processing units. These components provide the necessary computational power for executing tasks and managing system operations.

8. Connected Peripherals

   Embedded systems require peripherals to connect input and output devices. These peripherals facilitate communication with external components, enabling data acquisition, control, and interaction with the surrounding environment.

9. Hardware-Software Combination

   The hardware of an embedded system focuses on security and performance, while the software enables specific features and functionality. This synergy between hardware and software ensures the overall effectiveness of the embedded system.

Understanding these characteristics helps engineers and developers design embedded systems that meet the specific requirements of their intended applications while delivering reliable and efficient performance.

Determining the Applications of Embedded Systems in Various Industries

Embedded systems have revolutionized multiple industries, enabling enhanced functionality and efficiency. From healthcare and automotive to construction and manufacturing, embedded systems play a crucial role in various applications. This section provides a glimpse into the diverse industries that leverage embedded systems to address specific challenges and achieve their goals.

Healthcare and Medical Industry

- Biomedical Sensors

  Biomedical sensors are embedded systems used in remote health monitoring and diagnosis. They belong to the healthcare and medical industry. These sensors enable doctors to monitor patients' health and diagnose conditions remotely, without the need for in-person visits or surgical procedures. Patients can easily track their health progress at home, and devices like glucose monitors allow individuals with diabetes to measure their blood sugar levels. These smart devices also enable users to monitor heart rates, glucose levels, and blood pressure, providing valuable health information.

- Pacemakers

  Pacemakers are medical devices that utilize embedded systems to monitor and regulate heartbeats. They belong to the healthcare and medical industry. These embedded systems include sensors that can record various activities of the heart, allowing doctors to analyze the data and adjust treatment plans accordingly. Pacemakers play a crucial role in managing heart problems, especially in developing countries where such conditions are prevalent. By continuously monitoring the heart's activity, pacemakers assist in providing tailored care and improving patients' health.

- CPAP Machines

  Continuous Positive Airway Pressure (CPAP) machines, used in the treatment of sleep apnea, often include embedded systems. They belong to the healthcare and medical industry. These systems can monitor sleep patterns and automatically communicate with doctors if patients are experiencing sleep disturbances. By analyzing the data collected by the CPAP machine, doctors can make necessary changes to the treatment plan, ensuring better sleep quality and improved health outcomes for individuals with sleep apnea.

Health and Wellness Industry

- Fitness Trackers

  Fitness trackers, such as Fitbits, incorporate embedded systems and belong to the health and wellness industry. These devices can track weight, activity levels, and body composition, aiding individuals in their fitness journeys. Fitness trackers collect data related to body temperature, weight, and step count, which is then transmitted via wireless networks (e.g., GPRS or LTE) to servers for analysis. This data helps users, and their doctors monitor progress, set fitness goals, and make informed decisions about treatment and lifestyle adjustments.

Automotive Industry

- Automobiles

Embedded systems in automobiles ensure the safety and functionality of vehicles. They belong to the automotive industry. Advanced features like adaptive speed control, pedestrian recognition, airbags, and automated parking rely on embedded systems to provide a seamless and safe driving experience. These systems integrate various sensors, actuators, and control units to monitor the vehicle's surroundings, assist with driving tasks, and prevent accidents. The integration of embedded systems in automobiles enhances user safety, improves vehicle performance, and enables advanced functionalities.

Construction and Building Management Industry

- Central Heating Systems

Central heating systems found in buildings such as hospitals, schools, and homes utilize embedded systems. They belong to the construction and building management industry. Embedded systems regulate temperature control and ensure comfort. They include thermostat controls, allowing users to adjust the temperature as needed. By maintaining optimal temperature conditions, central heating systems enhance comfort and create a pleasant environment in various structures.

Manufacturing and Industrial Automation Industry

- Factory Robots

Factory robots belong to the manufacturing and industrial automation industry. They rely on embedded systems to perform complex tasks efficiently and safely. These embedded systems incorporate sensors and actuators to assess the environment, enabling robots to interact with objects and perform precise movements. By integrating artificial intelligence and machine learning algorithms, factory robots become more intelligent, effective, and adaptive to changing manufacturing requirements. Embedded systems in factory robots minimize errors, enhance productivity, and contribute to increased automation in industrial processes.

Technology and Navigation Industry

- GPS Systems

  GPS technology belongs to the technology and navigation industry. Embedded systems are used to provide precise time, location, and velocity synchronization. GPS systems are commonly found in devices such as cars, watches, and smartphones. These systems enable individuals to easily find their locations or navigate to specific destinations. By utilizing satellite signals and triangulation methods, embedded GPS systems provide accurate positioning information, facilitating navigation, tracking, and location-based services.

Understanding the Key Components of Embedded Systems

This section focuses on the hardware components of embedded systems. It covers essential components that contribute to the functionality and operation of embedded systems.

1. Power Supply

   The power supply is a critical component that provides the necessary electrical power to the embedded system. It can be sourced from a wall adapter or a battery, ensuring smooth and efficient operation.

2. Microcontroller

   The microcontroller serves as the computing power of the embedded system. It is an integrated circuit that acts as the "brain" of the system, processing data and executing instructions. Different microcontrollers with varied sizes, such as 8-bit, 16-bit, or 32- bit, are available to meet the requirements of different applications.

3. ROM/RAM

   Memory is essential for storing program code and data in the embedded system. Read-Only Memory (ROM) is non-volatile memory that stores the program code, while Random Access Memory (RAM) is volatile memory used for temporary data storage during system operation.

4. Timers / Counters

   Timers and counters are used in embedded systems to create delays or count events. Timers help in controlling the timing of specific functions, while counters keep track of the number of times a particular event occurs. These components are implemented using register-type circuits like flip-flops.

5. Communication Ports

   Communication ports enable the embedded system to establish communication with other devices or systems. Various communication ports, including USB, UART, I2C, SPI, and

RS-485, facilitates data exchange and interaction. The microcontroller may have built-in communication ports, or external ports can be added based on the complexity of the application.

6. Input and Output

Input and output components allow interaction with the embedded system. Sensors can be used as input devices to provide data to the system, while the microcontroller can be configured to function as input or output ports. The number of available input and output ports depends on the specific microcontroller used in the system.

These hardware components work together to provide the necessary processing power, storage, communication, and user interaction capabilities in embedded systems.

Identifying the Challenges and Considerations in Designing Embedded Systems

Developers working with embedded systems face various challenges despite the significant technological advancements in recent years. Some of these challenges include:

1. Lack of Flexibility

   As the demand for connected devices increases, embedded systems need to adapt to heterogeneous devices and different networking structures to accommodate new functionalities. This poses challenges in ensuring smooth integration of new services, adapting to new environments, handling frequent changes in hardware and software, and packaging and integrating small-sized chips with low weight and power consumption.

2. Security Issues

   Embedded systems operate in resource-constrained and often insecure environments, making security a significant concern. Designers face challenges in ensuring the security of embedded components, necessitating the implementation of robust security measures, cryptographic algorithms, and security protocols.

3. High Power Dissipation

   With an increasing number of transistors in embedded systems, managing power consumption becomes a constant challenge. Engineers need to find ways to optimize power usage beyond relying solely on process technology. Design choices, system architecture, and efficient utilization of resources are crucial in achieving better performance with low power consumption.

4. Increased Cost

   The development and deployment cycle of embedded systems require cost optimization to manage expenses associated with electronic components and production quantities. Design time and time-to-market constraints also pose challenges to ensure products reach the market within specified timelines.

5. Dependability

   Embedded systems play critical roles in various aspects of modern society, such as utility grids, transportation infrastructure, and communication networks. The reliability of these systems is paramount, as any disruptions or cyberattacks can have catastrophic consequences.

6. Testing Challenges

   Embedded systems often operate with software that may not receive regular security updates. Once deployed, these systems may continue running on the same software for years or even decades. Providing a mechanism for remote software updates becomes crucial to address security vulnerabilities that may arise. Ensuring embedded security requires careful planning and consideration for firmware updates, as the embedded operating system may not have automated capabilities to facilitate easy updates.

Addressing these challenges requires continuous innovation, research, and collaboration among embedded systems engineers, designers, and industry stakeholders. By overcoming these obstacles, embedded systems can continue to advance and contribute to various industries and applications.

SUMMARY

Embedded systems have become an essential part of our daily lives, powering devices like washing machines, microwaves, and ATMs, making our tasks easier and more efficient. These systems operate seamlessly in the background, allowing us to enjoy the convenience and functionality of these devices with minimal human intervention.

SUMMATIVE ASSESSMENT (SA)

Embedded systems are integral to our daily lives, powering devices and systems that enhance our convenience and efficiency. This summative assessment aims to test your understanding of embedded systems, their applications in various industries, key components, and the challenges faced in their design. Answer the following questions to assess your knowledge in this field.

Instructions: Read the following statements and select the correct answer from the list provided.

1. Embedded systems are specialized computer systems designed to perform specific functions within larger systems or devices. (True/False)

2. Which industry do fitness trackers belong to?
   a. Healthcare and Medical Industry
   b. Automotive Industry
   c. Construction and Building Management Industry
   d. Technology and Navigation Industry

3. What is the purpose of ROM in an embedded system?
   a. To provide electrical power
   b. To store program code
   c. To create delays and count events
   d. To establish communication with other devices

4. Timers are used in embedded systems for:
   a. Monitoring heartbeats
   b. Regulating temperature control
   c. Counting the number of times a particular event occurs
   d. Storing program code

5. Embedded systems are built to achieve efficiency in terms of:
   a. Power consumption
   b. Processing speed
   c. Memory capacity
   d. Communication speed

6. Which industry utilizes embedded systems in central heating systems?
   a. Healthcare and Medical Industry
   b. Automotive Industry
   c. Construction and Building Management Industry
   d. Manufacturing and Industrial Automation Industry

7. The hardware component that serves as the computing power of an embedded system is:
   a. Power Supply
   b. Microcontroller
   c. ROM
   d. Communication Port

8. One of the challenges faced in designing embedded systems is:
    a. Lack of flexibility
    b. High power dissipation
    c. Increased cost
    d. All of the above

9. Embedded systems are widely used in:
    a. Computers and laptops
    b. Household appliances
    c. Heavy machinery
    d. Telecommunication networks

10. Which industry relies on embedded systems for factory robots?

a. Healthcare and Medical Industry
b. Automotive Industry
c. Construction and Building Management Industry
d. Manufacturing and Industrial Automation Industry

REFERENCE / BIBLIOGRAPHY

Bose, A. (2018, September 04). *Embedded System – Characteristics, Types, Advantages & Disadvantages*. Retrieved from Electrical Fund Blog: https://electricalfundablog.com/embedded-system-characteristics-types-advantages-disadvantages/

MosChip Institute of Silicon Systems. (2022, June 20). *Industrial Applications of Embedded Systems – A Few Examples*. Retrieved from MOS Chip Institute of Silicon Systems: https://m-iss.in/embedded-systems/industrial-applications-of-embedded-systems-a-few-examples/

Sarkar, A. (2020, May 16). *Challenges Of Embedded Systems*. Retrieved from WHAT AFTER COLLEGE: https://whataftercollege.com/robotics-embedded-system/challenges-of-embedded systems/

Shaw, A. (2021, March 07). *Components of Embedded Systems*. Retrieved from The Engineering Projects: https://www.theengineeringprojects.com/2021/03/ components-of- embedded-systems.html

Hardware Fundamentals for Embedded Systems

INTRODUCTION

This module provides an essential foundation for understanding the underlying hardware components and principles involved in developing embedded systems. This covers the key concepts such as microcontrollers, microprocessors, sensors, actuators, and memory devices, equipping learners with the necessary knowledge to build efficient and reliable embedded systems.

LEARNING OBJECTIVES / OUTCOMES

At the end of the module, students should be able to:

- to understand the role and importance of microcontrollers and microprocessors, including their architecture, functionalities, and programming methodologies
- to explore the various types of sensors and actuators commonly used in embedded systems, and learn how to interface and control them effectively
- to gain knowledge about the characteristics, advantages, and limitations of different memory devices such as ROM, RAM, and flash memory

DISCUSSION OF THE LESSON

A. Microcontrollers and Microprocessors

I. Microcontroller

A microcontroller is essentially a small computer, resembling a chip with memory, capable of performing calculations through programming. It can receive input and produce outputs. Microcontrollers are found in various devices that require a certain level of control. For example, they enable the display of characters, types, and integers on the LCD screen of appliances. Devices that require measurement, control, information display, and value calculation are also equipped with a microcontroller chip.

The architecture of the microcontroller is its internal hardware design. Understanding their difference is crucial for assessing their suitability for different purposes. The architecture provides clear and distinct definitions for each section of the microcontroller.

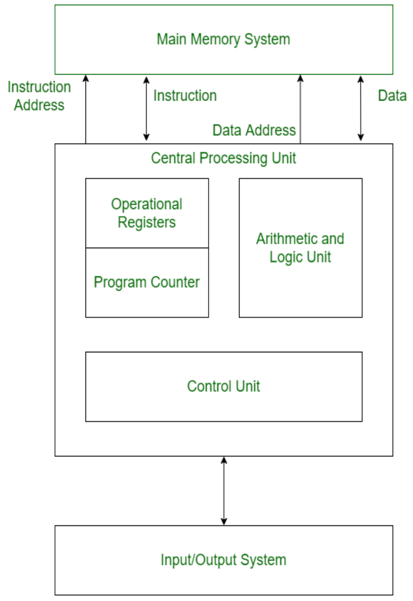
a) *Harvard Architecture*

This architecture utilizes a distinct approach by dividing its memory into two separate parts, enabling independent storage of data and instructions. Additionally, it incorporates separate buses for transferring data and fetching instructions. This unique design allows the CPU to simultaneously retrieve data and instructions, enhancing overall efficiency and performance.

In the Harvard architecture, buses serve as pathways for transmitting signals. This type of computer architecture employs separate buses for handling instructions and data. This ensures that instructions and data can be transferred independently, allowing for efficient and simultaneous processing of both types of information.

Types of Buses:

- Data Bus – carries data in the main memory, processor, and I/O devices.
- Data Address Bus – carries the address of data from the processor to the main memory system.
- Instruction Bus – carries instructions in the main memory, processor, and I/O devices.
- Instruction Address Bus – carries the address of instructions from the processor to the main memory system.

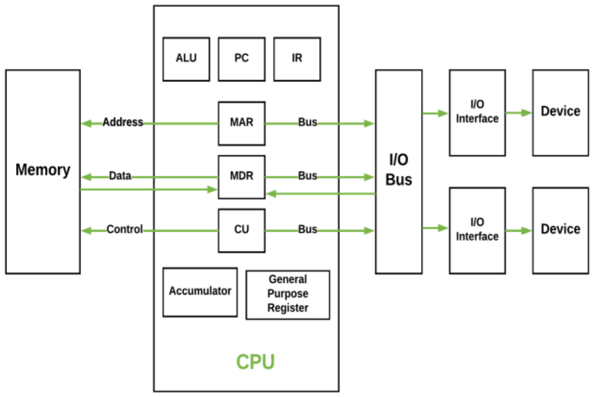


Various types of registers are utilized to store addresses corresponding to different types of instructions. These registers play a crucial role in holding the memory addresses required for accessing and executing specific instructions within the architecture.

Types of Registers:

- Program Counter – it contains the location of the following instruction to be carried out. The memory address register receives the next address from the program counter.

- Arithmetic and Logic Unit – all necessary calculations are carried out by the CPU's arithmetic logic unit. It can perform several arithmetic operations, including addition, subtraction, comparison, logical operations, bit shifting, and more.
- Control Unit – All processor control signals are operated by the Control Unit. It manages the system's input and output devices as well as how data and instructions are moved around the system.
- Input/Output System – data is read into the main memory using input devices and CPU input instructions. Through output devices, information from the computer is given as output.



Features:

- Separate Memory Spaces – data and instruction are kept in separate memory spaces. This separation makes it possible for the processor to access both the instruction and data memories at the same time, facilitating quicker and more effective data retrieval.
- Fixed Instruction Length – instructions have a fixed length, which makes the instruction fetching easier and speeds up instruction processing.
- Suitable for Embedded Systems – embedded systems often adopt Harvard architecture due to its ability to deliver rapid and effective accessibility to both instructions and data, which is crucial in real-time applications.
- Limited Flexibility – the Harvard architecture's separate memory spaces restrict the processor's ability to carry out certain tasks, like changing instructions in-place. This is because altering instructions requires access to the instruction memory, which is separate from the data memory.

b) *Von Neumann Architecture*

Von Neumann architecture was named after John Von Neumann, a Hungarian-American physicist and mathematician, who gave the description of this architecture in 1945. It consists of a single shared memory (primary memory), a single bus for memory access, an input, and output unit, an arithmetic unit, and a program control unit. This architecture is important in developing operating systems.

In a multiple-register configuration system, information is shared between the registers via buses. A bus structure is made up of a number of common lines, one for each bit in a register, which are used to transfer binary data one bit at a time. For each specific register transfer, the bus chooses a register based on control signals. Three main bus systems are included in the Von-Neumann Architecture for data transfer.

Types of Buses:

- Address Bus – transports data addresses between the processor and memory but not the actual data.
- Data Bus – carries information between the processor, the memory, and the input/output devices.
- Control Bus – carries control commands from the CPU to regulate and coordinate all the computer's operations.

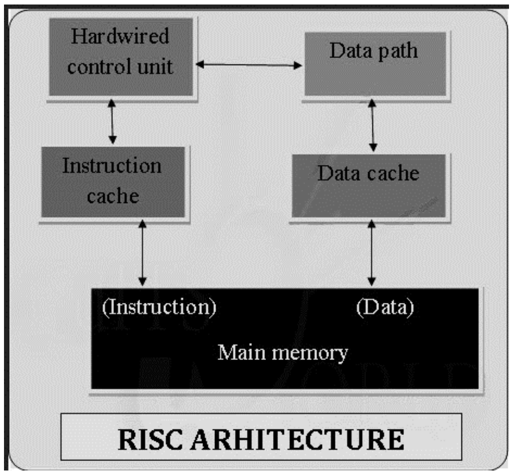
Registers are high-speed storage locations in the CPU. The registers are queried for the data that the CPU will process. Different register types are employed in architecture.

Types of Registers:

- Memory Address Register (MAR) – it saves the memory locations where instructions are stored or fetched from.
- Memory Data Register (MDR) – it stores any information that needs to be transferred to and stored in memory, including instructions that are fetched from memory.
- Accumulator (AC) – stores the outcomes of ALU calculations. It stores the intermediate of logical and arithmetic operations. It serves as a device or temporary storage location.
- Program Counter (PC) – keeps track of the memory location of the next instructions. The Memory Address Register (MAR) receives this next address from the PC.
- Current Instruction Register (CIR) – contains the current instruction during processing.

Features:

- Fetch-Decode-Execute – The processor follows a cyclic process called the fetch-decode-execute cycle. It fetches the next instruction from memory, decodes it to know the operation to be performed, and executes the operation.
- Single Memory – The Von Neumann architecture uses a single memory system to store both data and program instructions. Data and instructions are stored in the same memory area, allowing for flexibility in programming.
- Von Neumann Bottleneck – this architecture has a limitation known as the Von Neumann bottleneck, where the sequential execution of instructions causes potential performance limitations due to the shared use of the same memory for both data and instructions.

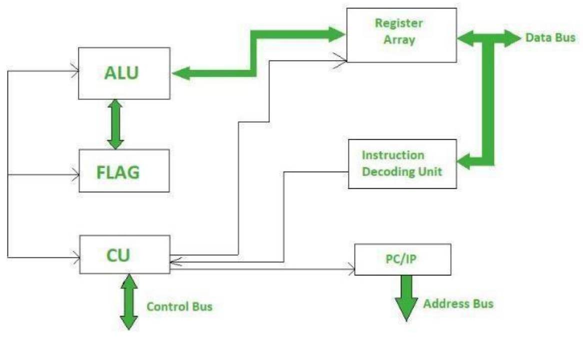


II. **Microprocessor**

Most modern personal computers, smartphones, and other electronic gadgets use microprocessors. Most processing tasks are carried out by the computer system's central processing unit (CPU). A computer's microprocessor is a vital component because it manages the retrieval, decoding, and execution of instructions that are stored in memory. It could be assumed that the microprocessor serves as the computer's "brain," controlling all overall operations and execution. The advancement of microprocessors has been crucial to the advancement of computers, allowing them to get smaller, faster, and more powerful over time.

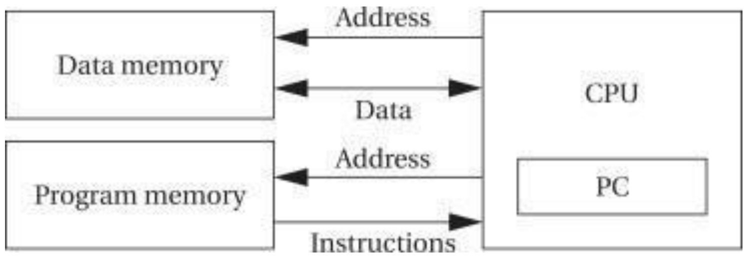
a) *Reduced Instruction Set Computing (RISC)*

In contrast to the highly specialized set of instructions typically used in other architectures, a reduced instruction set computer uses a condensed, highly optimized set of instructions. Complex Instruction Set Computing (CISC) is an



alternative to RISC, which is frequently regarded as the most effective CPU architecture currently in use.

A central processing unit (CPU) that uses RISC implements the processor design principle of simplified instructions with faster execution. This results to improved performance. The ability to increase the register set and internal parallelism by increasing the number of parallel threads executed by the CPU and the speed at which the CPU executes instructions is a key feature of RISC. The "Advanced RISC Machine" (ARM) created by Arm Ltd., is a family of instruction set architecture that is RISC-based. Processors based on this architecture are widely used in laptops, desktop computers, gaming consoles, smartphones, tablets, and an increasing number of other smart devices.



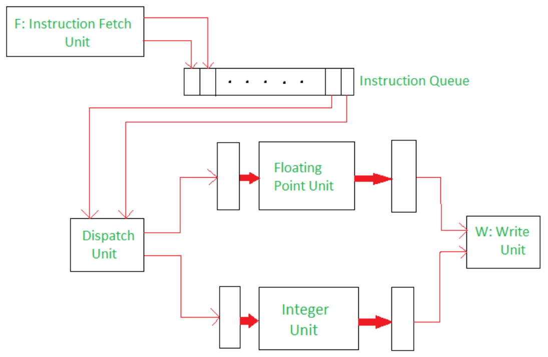
Features:

- Simple Instructions – employ a limited set of fixed instructions that are simple and atomic in nature. These instructions are made to be quick to execute; they usually complete in one clock cycle.
- Pipelining – this architecture commonly utilizes pipelining, in which instructions are broken up into stages and carried out concurrently. Thus, performance is enhanced and instruction throughput is increased.
- Load/Store Architecture – commonly utilize a load/store architecture, where data is loaded from memory into registers for manipulation, and results are stored back into memory. Arithmetic and logical operations are performed on registers rather than directly on memory.
- Compiler Friendly – RISC is designed to work efficiently with compilers. The simplicity of the instruction set and uniform instruction format allows compilers to optimize code generation and utilize registers effectively.

b) *Complex Instruction Set Computing (CISC)*

It has a complex set of instructions and uses a format for variable-length instructions. Only two bytes can be used for instructions that need register operands. Instructions that need two memory addresses may require up to five bytes for the entire instruction code. As a result, CISC has a variable-length instruction encoding, and the time it takes to execute an instruction can vary. Operands that are in memory can be directly manipulated by the CISC processor.

Instead of using a register file, many CISC architectures read their inputs and write their outputs to the memory system. More hardware logic is needed to implement the numerous addressing modes that the CISC architecture accepts. It also makes an effort to convert each statement written in a high-level language into a single machine instruction. As a result, computation speed is slower.



Features:

- Rich Instruction Set – offers a wide range of complex instructions that can perform multiple operations in a single instruction. These instructions often involve memory access, arithmetic operations, and complex data manipulation.
- Memory-to-Memory Operations – often supports memory-to-memory operations, where instructions can directly access memory operands without involving intermediate registers. This can simplify code development and reduce the number of instructions required for certain operations.
- Complex Addressing Modes – typically provides a variety of addressing modes, allowing instructions to access memory operands in various ways. This includes indexed addressing, indirect addressing, and base+offset addressing, among others.
- Instruction Decoding – due to the complexity of CISC instructions, the decoding process can be more intricate compared to simpler instruction sets like RISC. Decoding these instructions often requires more hardware resources and can result in longer decoding times.

c) *Superscalar Architecture*

Many processors use the parallel computing technique known as superscalar architecture. The central processing unit (CPU) of a superscalar computer controls multiple instruction pipelines to carry out several operations simultaneously during a clock cycle. This is accomplished by feeding the various pipelines through a number of the processor's execution units. The CPU's instruction fetching mechanism must be able to delegate and retrieve instructions intelligently in order to implement a superscalar architecture successfully. Otherwise, pipeline stalls could happen, leaving execution units often idle.

Features:

- Instruction-Level Parallelism (ILP) – superscalar processors exploit ILP by identifying and executing instructions that can be processed concurrently. This is achieved through techniques such as instruction reordering, speculation, and dependency analysis.
- Multiple Execution Units – superscalar processors include multiple execution units, such as arithmetic logic units (ALUs), floating-point units (FPUs), and memory units. These units can perform different types of operations simultaneously, enhancing parallel processing capabilities.
- Register Renaming – this is a technique used in superscalar architectures to eliminate false data dependencies. It allows multiple instructions to use the same physical register simultaneously by mapping them to different architectural registers.
- Branch Prediction –branch prediction mechanisms is used to mitigate the impact of branch instructions on performance. Predicting the outcome of branches helps in minimizing pipeline stalls and ensuring the continuous flow of instructions.

III. Comparison Between Different Microcontrollers

	Arduino Uno	Raspberry Pi 4	ESP32	PIC16F877A	STM32F407	MSP430 G2553
Architecture	AVR (RISC)	AVR (CISC)	Xtensa (RISC)	PIC (RISC)	ARM (Cortex-M4)	MSP430 (RISC)
Clock Speed	16 MHz	1.5 GHz	240 MHz	20 MHz	168 MHz	16 MHz
Flash Memory	32 KB	-	4 MB	14 KB	1 MB	16 KB
RAM	2 KB	2 GB–8 GB	520 KB	368 B	192 KB	512 KB
GPIO Pins	14	40	17	34	33	82

ADC Channels	6	-	1	18	8	12
Communication Protocols	UART, SPI, I2C	UART, SPI, I2C, Ethernet, Wi-Fi, Bluetooth	UART, SPI, I2C, Wi-Fi, Bluetooth	UART, SPI, I2C	UART, SPI, I2C, Ethernet, USB	UART, SPI, I2C
Additional Features	-	HDMI, USB, GPIO	2x CPU cores, Touch Sensor, CAN Bus, Camera Interface	-	ADC, DAC, DMA, PWM	-
Power Consumption	20 mA (Active)	3.5 A (Typical)	80 mA (Active)	20 mA (Active)	100 mA (Typical)	280 µA (Active)

B. Sensors and Actuators

Sensors and actuators are essential components of many technological systems, enabling the measurement and control of various physical quantities. In this topic, we will delve into the characteristics of sensors and actuators, explore their different types, and discuss the key differences between them. Understanding these fundamental aspects will provide a solid foundation for comprehending their applications and integration into modern technologies.

Characteristics of Sensors and Actuators:

Characteristics	**Sensors**	**Actuators**
Sensing Principles	Resistive, capacitive, optical, magnetic, etc.	Electrical, hydraulic, pneumatic, thermal, etc.
Sensitivity and Range	Varies based on the sensing principle	Varies based on actuation principle
Accuracy and Precision	Determines measurement reliability and repeatability	Influences actuation precision
Response Time	Time taken to detect and respond to changes	Time taken to initiate mechanical action
Environmental Considerations	Affected by temperature, humidity, pressure, etc.	Endurance in challenging environmental conditions
Force and Torque	N/A	Actuators generate force and torque for mechanical action
Speed and Accuracy	N/A	Actuators exhibit varying speed and accuracy levels
Power Consumption	N/A	Energy consumption for mechanical motion
Reliability and Durability	N/A	Endurance and longevity in continuous operation

Differences of Sensors and Actuators:

Differences	**Sensors**	**Actuators**
Function and Purpose	Detect and measure physical quantities	Convert energy into mechanical motion or action
Signal Flow	Generate electrical signals as a result of sensing	Receive electrical signals to produce mechanical motion
Design and Construction	Diverse designs and materials for sensing physical phenomena	Varied designs and materials for generating mechanical action

Applications	Monitoring, measurement, and control systems	Physical response, movement, and control systems
Integration	Sensor-actuator interfacing, signal conditioning, feedback control	Interaction within closed-loop systems, integration with control mechanisms

I. Types and functionalities

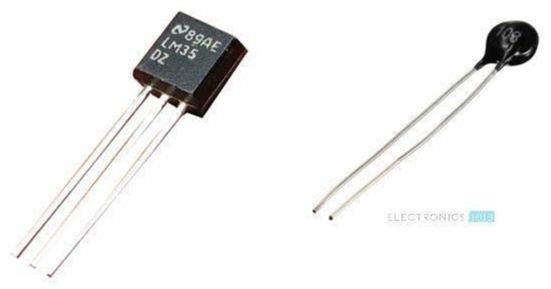
Types of Sensors:

1. Temperature Sensor

The Temperature Sensor is a widely used and highly favored sensor due to its versatility. As its name implies, this sensor is designed to detect and measure temperature variations in its surroundings.

There are several variations of Temperature Sensors available, including Temperature Sensor ICs such as LM35 and DS18B20, Thermistors, Thermocouples, and RTD (Resistive Temperature Devices).

Temperature Sensors can be classified into two main types: analog and digital. Analog Temperature Sensors operate by correlating temperature changes with alterations in their physical properties, such as resistance or voltage. LM35, for example, is a well-known analog Temperature Sensor that follows this principle.


LM35 - Temperature Sensor IC 10KΩ NTC Thermistor

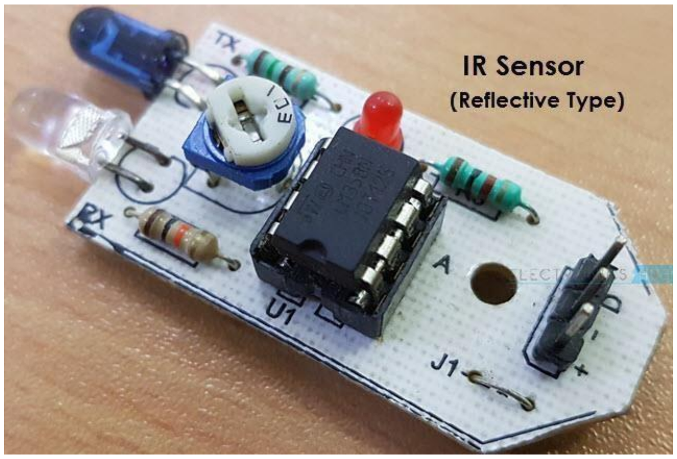
2. Proximity Sensors

A Proximity Sensor is a sensor that is designed to detect the presence of an object without making physical contact. There are various techniques employed to implement Proximity Sensors, including Optical methods such as Infrared or Laser sensing, Sound-based approaches using Ultrasonic technology, Magnetic sensing utilizing the Hall Effect, Capacitive sensing, and more.

3. **Infrared Sensor (IR Sensor)**

IR Sensors, also known as Infrared Sensors, utilize light-based technology and find applications in various fields such as Proximity Sensing and Object Detection. They are commonly integrated as proximity sensors in almost all mobile phones.

There are two primary types of Infrared or IR Sensors: Transmissive Type and Reflective Type. In the Transmissive Type IR Sensor, an IR Transmitter (usually an IR LED) and an IR Detector (typically a Photo Diode) are positioned facing each other. When an object passes



between them, the sensor detects the object by sensing the interruption of the infrared light transmission.

On the other hand, the Reflective Type IR Sensor involves positioning the transmitter and the detector adjacent to each other, both facing the object. When an object comes in front of the sensor, the infrared light emitted by the IR Transmitter is reflected off the object and detected by the IR Receiver. This reflection allows the sensor to detect the presence of the object.

IR Sensors are implemented in various applications, including Mobile Phones, Robots, Industrial assembly processes, and automobiles, among others.

4. **Ultrasonic Sensor**

An Ultrasonic Sensor is a non-contact device designed to measure both distance and velocity of an object. It operates on the principles of sound waves with frequencies beyond the range of human hearing.

By utilizing the time of flight of the sound wave, an Ultrasonic Sensor can determine the distance between the sensor and an object, similar to how SONAR works. The sensor emits ultrasonic waves, and by measuring the time it takes for the waves to bounce back after hitting an object, the distance can be calculated.

Additionally, the Doppler Shift property of the sound wave is employed to measure the velocity of an object using an Ultrasonic Sensor. Changes in frequency occur when the sound waves are reflected off a moving object, allowing for the calculation of the object's velocity.

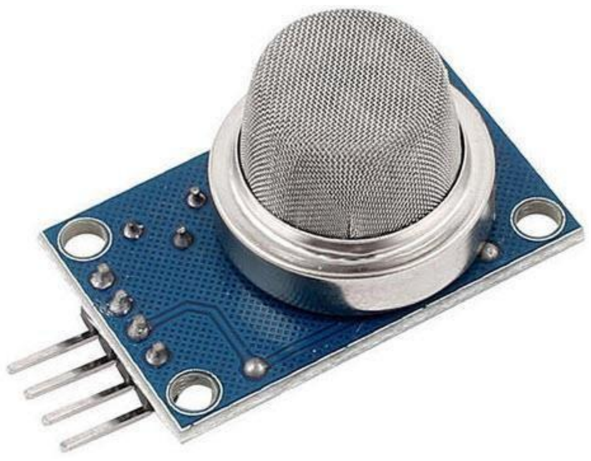
5. **Light Sensor**

Light Sensors, also referred to as Photo Sensors, play a significant role in sensing light levels. A commonly used light sensor is the Light Dependent Resistor (LDR). The LDR exhibits a characteristic where its resistance varies inversely with the intensity of ambient light. As light intensity increases, the resistance of the LDR decreases, and vice versa.

By incorporating an LDR into a circuit, changes in its resistance can be calibrated to measure the intensity of light. In more intricate electronic system designs, two other light sensors, namely the Photo Diode and Photo Transistor, are frequently employed. These sensors, like the LDR, are analog sensors capable of detecting light variations.

6. Smoke and Gas Sensors

Smoke and Gas Sensors are highly valuable sensors utilized in safety-related applications. Offices and industries typically have multiple smoke detectors that are capable of detecting smoke, typically associated with fires, and triggering an alarm.



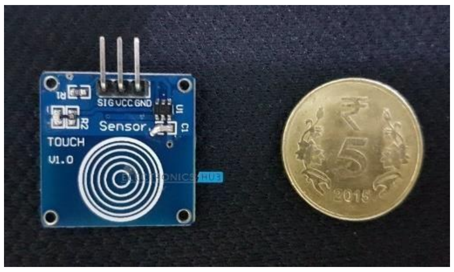
Gas Sensors, on the other hand, are commonly found in laboratories, large-scale kitchens, and industrial settings. These sensors are designed to detect various gases such as LPG, Propane, Butane, Methane (CH4), and more.

In recent times, smoke sensors that can also detect gases have become increasingly common in residential homes as an additional safety measure. This helps to ensure early detection and timely response to potential fire or gas leak incidents.

7. **Alcohol Sensor**

An Alcohol Sensor, as the name implies, is a sensor designed to detect the presence of alcohol. These sensors are commonly utilized in breathalyzer devices, which are used to determine whether a person is under the influence of alcohol or not. Law enforcement personnel employ breathalyzers to identify individuals who are driving under the influence of alcohol, aiding in the prevention of drunk driving incidents.

8. **Touch Sensor**



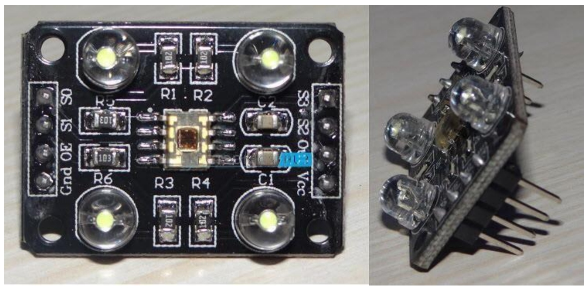
Although touch sensors may not receive much attention, they have become an indispensable part of our daily lives. Most touch screen devices, including mobile phones, tablets, and laptops, rely on touch sensors for user interaction. Additionally, trackpads in laptops also utilize touch sensors.

As the name implies, touch sensors detect the touch of a finger or stylus on a surface. They are commonly classified as Resistive and Capacitive types. Capacitive touch sensors have gained prominence in modern devices due to their superior accuracy and higher signal-to-noise ratio compared to Resistive touch sensors.

The prevalence of touch sensors in our technological devices highlights their significance in enabling intuitive and interactive user experiences.

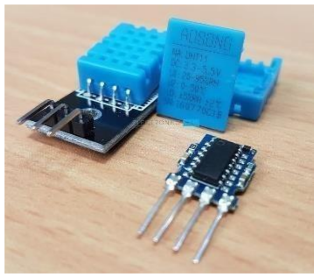
9. **Color Sensor**

A Color Sensor is a valuable tool for developing applications related to color sensing, image processing, color identification, and industrial object tracking. It enables the detection and analysis of colors in various contexts. One commonly used Color Sensor is the TCS3200, which is a straightforward device capable of detecting any color and producing a square wave signal proportional to the wavelength of the detected color. This sensor facilitates precise color measurement and analysis for a range of applications.



10. **Humidity Sensor**

In weather monitoring systems and various other applications, the measurement of humidity plays a crucial role, and Humidity Sensors are instrumental in achieving accurate humidity readings.

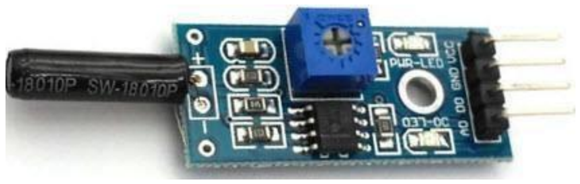


Most humidity sensors primarily measure relative humidity, which is the ratio of water content in the air to the maximum water vapor the air can hold at a specific temperature. Since relative humidity is temperature-dependent, the majority of Humidity Sensors are also capable of measuring temperature.

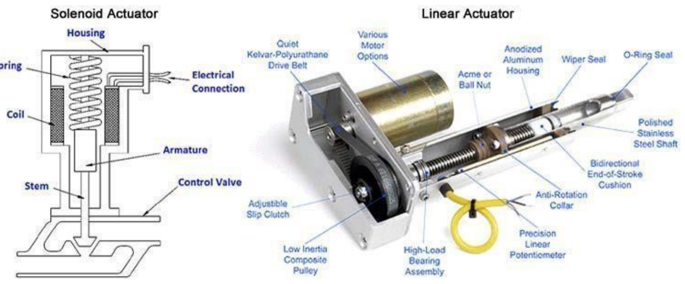
Humidity Sensors are typically categorized into three types: Capacitive, Resistive, and Thermal Conductive. Capacitive and resistive types, such as the popular DHT11 and DHT22 sensors frequently used in DIY projects, are widely employed. These sensors offer reliable humidity and temperature measurements for a variety of applications.

11. **Tilt Sensor**

Tilt Sensors are commonly utilized for detecting inclination or orientation and are known for their simplicity and cost-effectiveness. In the past, tilt sensors were typically constructed with



Mercury, earning them the alternate name of "Mercury Switches." However, modern tilt sensors predominantly incorporate a roller ball mechanism, offering enhanced reliability and safety compared to Mercury-based designs.


Solenoid Actuator
Housing
Spring
Electrical Connection
Coil
Armature
Stem
Control Valve
Linear Actuator
Quiet Kelvar-Polyurathane Drive Belt
Various Motor Options
Anodized Aluminum Housing
Wiper Seal
O-Ring Seal
Acme or Ball Nut
Polished Stainless Steel Shaft
Bidirectional End-of-Stroke Cushion
Anti-Rotation Collar
Adjustible Slip Clutch
Precision Linear Potentiometer
Low Inertia Composite Pulley
High-Load Bearing Assembly

Types of Actuators:

1. **Electrical Actuator**

Electric actuators are devices that convert electrical energy, whether AC or DC, into mechanical torque or linear motion. Electric motors have gained significant popularity as actuators due to their ease of control, long lifespan, and high efficiency. Servo motors, as well as other types of DC and AC motors, are commonly used to provide rotational output.

Rotational actuators, such as the one depicted in the provided image, offer efficient performance with a higher power-to-weight ratio and fewer moving parts compared to most linear actuators. In the image, a rotary actuator is showcased, specifically employed as a valve within a cross-sectional area of a pipeline.

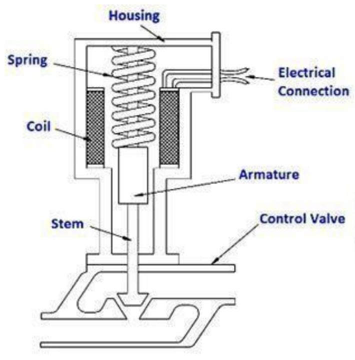
2. **Electric Linear Actuators**

Linear actuators are devices that transform electrical energy into linear motion. There are two primary types of linear actuators: solenoid actuators and rotary-to-linear motion actuators.

Solenoid actuators provide linear motion directly through the action of a solenoid, which converts electrical energy into a magnetic field to generate linear movement.

The second type of linear actuator employs a mechanical system to convert rotary motion into linear motion. These actuators consist of a motor that provides rotational motion, with the output shaft connected to a set of gears and a drive mechanism. This arrangement converts the rotary motion into linear motion. The output shaft of the linear actuator is connected to linear guides that ensure smooth and controlled movement.

Among these two types, the linear actuators with rotary-to-linear motion conversion are more commonly used, as they typically offer higher load capacity and can handle heavier loads.



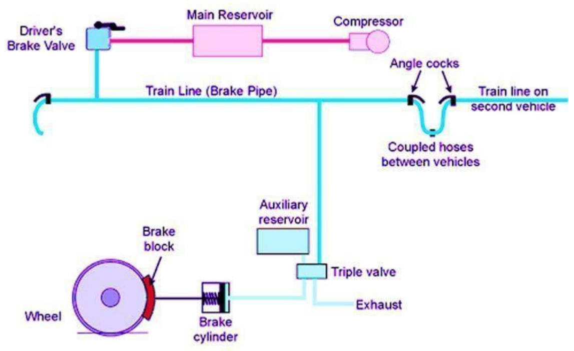
3. **Hydraulic Actuator**

Hydraulic actuators are devices that employ hydraulic power to generate mechanical movement. They consist of a cylinder or fluid-based motor capable of providing both linear and rotary motion. In hydraulic actuators, incompressible fluids such as oil are used from a pump to fill the cylinder and apply power to the pistons on one or both sides.

The speed and force of hydraulic actuators can be adjusted by increasing the pressure of the fluid inside the cylinder. These actuators have a long history and are among the earliest types of actuators known.

The provided image displays a cross-section of a hydraulic actuator. In the image of a JCB machine, we can observe three hydraulic actuators utilized to control its arm, showcasing the practical application of hydraulic actuators in industrial machinery.

4. **Pneumatic Actuator**



Pneumatic actuators operate on the same principle as hydraulic actuators, but they utilize a gaseous fluid instead of a liquid. The driving force for the pistons in pneumatic actuators comes from high-pressure compressed air or vacuum. This compressed air or vacuum is employed to generate linear or rotary mechanical motion.

Like hydraulic actuators, pneumatic actuators convert pressure into force to achieve the desired mechanical action. They are commonly used in various applications where compressed air is readily available.

The provided diagram showcases the application of pneumatic actuators in air brakes used in trains, highlighting their importance in ensuring safe and efficient braking systems in railway transportation.

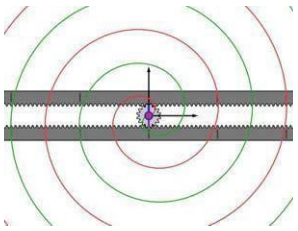
5. **Magnetic Actuators**

Magnetic actuation relies on the principle of Lorentz Forces, which states that when a current-carrying conductor is placed in a static magnetic field, the interaction between the two produces a force. This force can be harnessed to induce the displacement of a mechanical structure.

Magnetic actuators find applications in small-scale systems such as nano-robots and hold great potential in the field of biomedicals. One key advantage of magnetic actuators is that they operate at low voltages and do not involve contact operations, making them suitable for delicate and sensitive applications.

Additionally, there is another type of magnetic actuator known as a Magneto Rheological Fluid Actuator. These actuators employ fluids that solidify in the presence of a magnetic field. They are used in various applications such as hydrodynamic bearings and semi-active clutches, where the ability to change the fluid's rheological properties allows for precise control and adjustment.

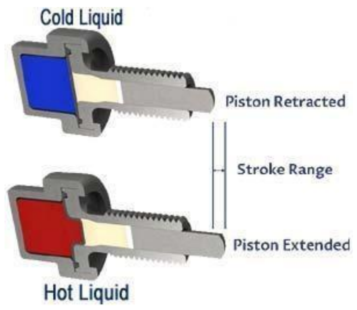
Overall, magnetic actuators offer unique advantages and have diverse applications in fields ranging from small-scale robotics to advanced engineering systems.



6. **Mechanical Actuators**

Mechanical actuators are devices that convert one form of motion into another using mechanisms such as gears, chains, pulleys, rails, and various other contraptions. These actuators are often combined with other actuators and driving mechanisms to enhance torque or power output, as well as to convert between linear and rotary motion, or vice versa.

Mechanical actuators play a crucial role in various applications where precise control and efficient transmission of motion are required. They can be found in systems ranging from industrial machinery to robotics, automotive engineering to aerospace technology. By utilizing mechanical principles and mechanisms, these actuators enable the amplification and transformation of motion to meet specific requirements and achieve desired functionality.



7. **Thermal Actuators**

Thermal actuators are typically composed of metals or shape memory alloys that can maintain a specific shape when exposed to heat energy. These materials possess thermal sensitivity and are capable of producing linear motion in response to changes in heat energy. Thermal actuators offer advantages such as compactness, lightweight design, and ease of use.

The provided image depicts a thermostatic valve, a common application of thermal actuators in shower systems. Thermostatic valves passively regulate the temperature of water, allowing for a comfortable and consistent showering experience by automatically adjusting the flow of hot and cold water based on the temperature set by the user.

Thermal actuators find application in various industries, including automotive, aerospace, and consumer electronics, where precise control of temperature-induced motion is required.

II. Interfacing with microcontrollers

The Internet of Things (IoT) is a concept wherein objects are equipped with sensors, actuators, and processors to enable communication and meaningful interactions. It can be understood as the convergence of the physical and digital realms. With the integration of sensors, actuators, processors, and transceivers, devices that were once standalone can now connect to networks.

IoT comprises various technology layers that facilitate the sharing of data among everyday objects. These objects collect information over the Internet to provide intelligence, autonomous actions, and value. The effectiveness of IoT depends largely on the quality of the collected data.

An IoT device typically consists of a physical object (the "thing") combined with a controller (the "brain") and connected through networks (the "Internet"). Sensors and actuators play a critical role in enabling interactions with the physical world in IoT technology. They allow for the collection of data from the environment and the ability to act upon that data.

The provided diagram illustrates the collaboration between sensors and actuators in an IoT system, showcasing their combined functionality in collecting data and carrying out actions.

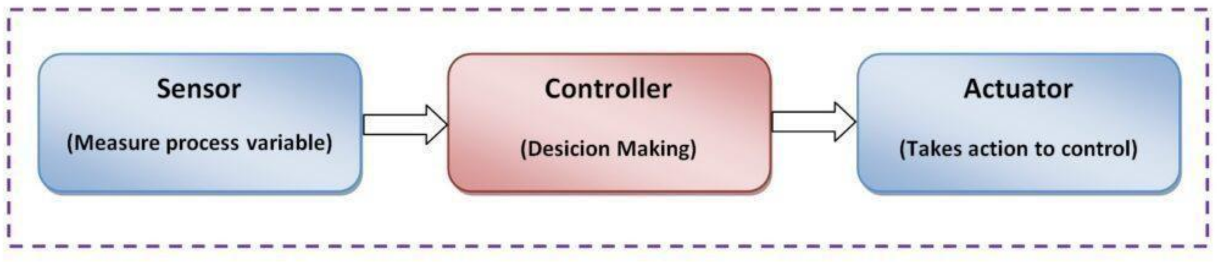
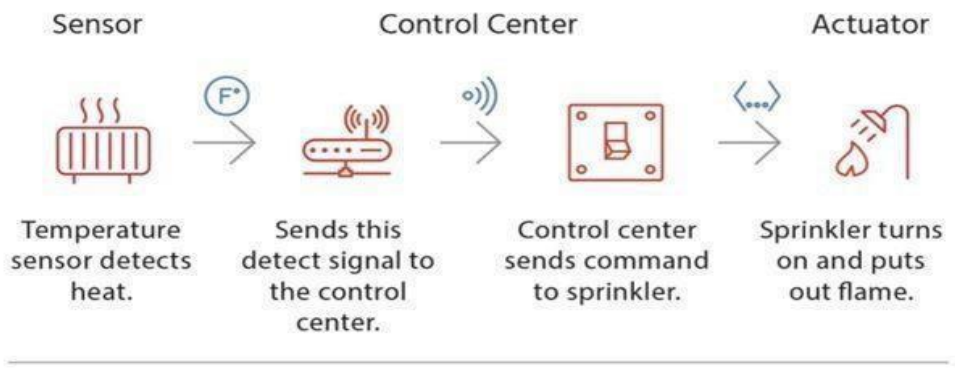


Figure- Sensor and actuator in a system

In a basic sense, the functioning of a device in an IoT system involves sensors sensing the environment and sending the collected data to the controller. The controller then uses this data to determine the appropriate control signal required to maintain a specific set value or desired condition. This control signal is then sent to the actuator, which is responsible for performing the necessary actions to achieve the desired outcome.

It's important to note that an actuator requires external energy or power source to carry out its actions. This energy can come from various sources, such as electricity, pneumatic pressure, hydraulic power, or other forms of energy, depending on the specific actuator and the system requirements.

Overall, in an IoT system, sensors gather environmental data, the controller processes the data to generate control signals, and actuators utilize external energy to perform the required


Sensor
Control Center
Actuator
F°
Temperature sensor detects heat.
Sends this detect signal to the control center.
Control center sends command to sprinkler.
Sprinkler turns on and puts out flame.

From Sensor to Actuator in IoT

actions, all working together to achieve the desired functionality and maintain the desired set value.

The diagram showcases the flow from sensors to actuators in an IoT system. The core infrastructure of an IoT framework consists of sensors, actuators, computer servers, and the communication network. Within this framework, data collection, handling, communication, and processing take place.

The IoT device collects a significant amount of information from various sensors, capturing data from the environment or system it is monitoring. Through decision-making processes, the device determines which data is relevant for the current conditions and decides where it should be processed or stored. Additionally, the device establishes the desired level of communication required for transmitting the data.

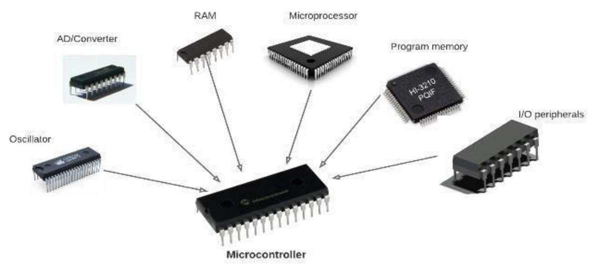
Actuators play a crucial role in the automation of the system. They enable the execution of actions or control mechanisms based on the relevant information processed by the IoT device. Actuators respond to the decisions made by the system and carry out the necessary actions to maintain desired conditions or implement specific functionalities.

Overall, this diagram illustrates the flow of data from sensors to actuators within the IoT framework, emphasizing the importance of data collection, decision-making, and automation in enabling intelligent and responsive IoT systems.

Microcontrollers play a crucial role in numerous electronic systems, ranging from simple embedded devices to complex automation systems. Interfacing sensors and actuators with microcontrollers is essential for enabling these systems to interact with the physical world. Sensors provide input to the microcontroller by detecting and measuring physical quantities, while actuators receive output signals from the microcontroller to initiate mechanical action or control processes.

When interfacing sensors with microcontrollers, the primary objective is to acquire accurate and reliable data from the physical environment. Sensors generate electrical signals in response to physical phenomena, and these signals need to be conditioned and processed before being fed into the microcontroller. Signal conditioning techniques, such as amplification, filtering, and calibration, are employed to enhance the accuracy, resolution, and stability of sensor measurements. Once the sensor signals are conditioned, analog-to-digital converters (ADCs) are often used to convert the analog signals into digital data that can be easily processed and interpreted by the microcontroller.

The integration of sensors and actuators with microcontrollers enables closed-loop control systems, where sensor data is continuously monitored, processed, and used to regulate the actuator's behavior. This feedback mechanism allows the microcontroller to maintain desired system parameters or respond to changing environmental conditions in real-time.



In conclusion, the interfacing of sensors and actuators with microcontrollers is a vital aspect of electronic systems. By connecting sensors to microcontrollers, accurate and real-time data acquisition is made possible, allowing for intelligent decision-making. Simultaneously, interfacing actuators with microcontrollers enables precise control and manipulation of physical processes. This integration facilitates the creation of smart devices, automation systems, and Internet of Things (IoT) applications that can sense, analyze, and respond to their environment effectively.

C. Memory Devices

I. ROM, RAM, and Flash Memory

There are different type of memory devices used in embedded system. The core parts of embedded systems are RAM (Random Access Memory), ROM (Read-Only Memory), and Flash memory. Each type of memory has a specific function and is essential to the system's overall performance.

a.) ROM (Read-Only Memory)

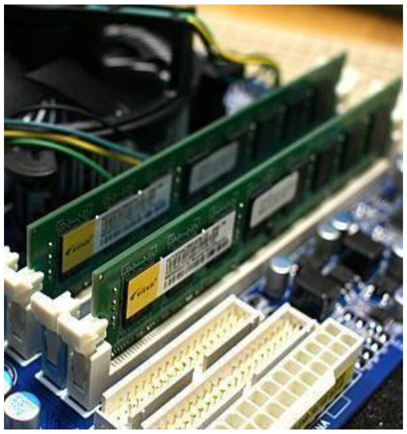
The non-volatile memory type memory that used to store data or instructions that are meant to remain unchanged throughout typical system operation. It includes bootloaders, firmware, and other essential programs that initialize the system at startup. ROM, as opposed to RAM, keeps its data even if power is lost.

There are various types of Read-only Memory (ROM):

- PROM (Programmable Read-Only Memory) - It belongs to the non-volatile class of computer memory. As a result, data that has been written to PROM cannot be removed. Programs or firmware for microcontrollers and other digital devices are stored in PROM.
- EPROM (Erasable and Programmable Read-Only Memory) - Is a memory that maintains its data even when the power is turned off. The data in EPROM is removed using UV (ultraviolet) rays. The operating system and the program for the access control panel are both stored on EPROMs by manufacturers of security systems.
- EEPROM (Electrically Erasable and Programmable Read-Only Memory) - It refers to a specific kind of rewritable storage chip or memory package that can maintain its stored data even in the absence of power. Another non-volatile memory example is EEPROM.
- MROM (Mask Read-Only Memory) - Another kind of read-only memory (ROM) whose data is programmed by the manufacturer of integrated circuits rather than

the user. The term "mask" originates from the fabrication of integrated circuits, in which areas of the chip are blocked off during the photolithography process. Photolithography, a general term for methods that use light to create finely patterned thin films of suitable materials over a substrate, such as a silicon wafer, to shield particular regions of it from etching, deposition, or implantation operations later on.

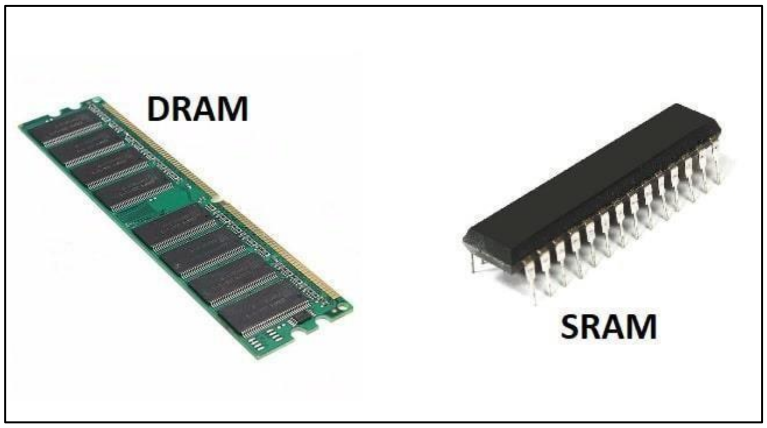
b.) RAM (Random Access Memory)



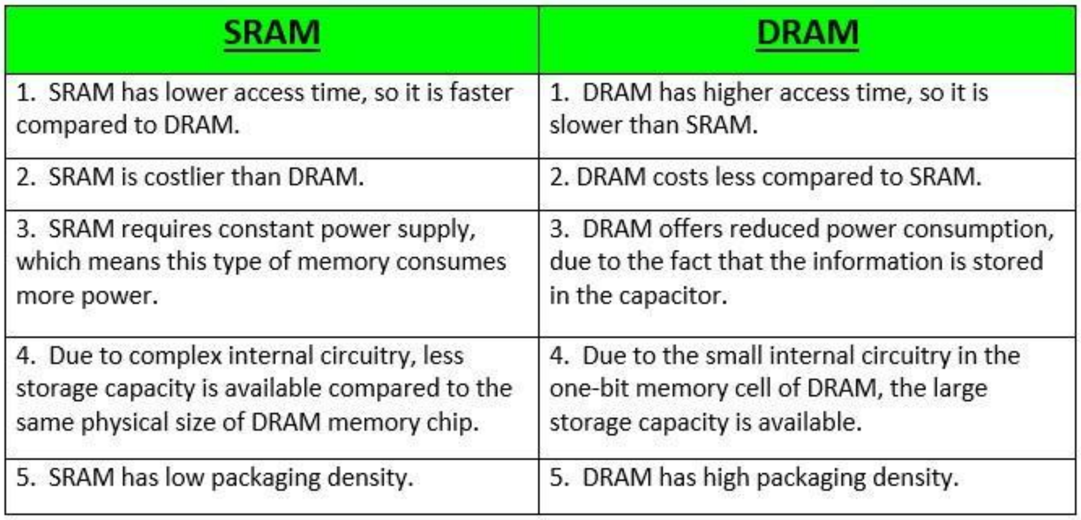
RAM is a type of volatile memory that gives program execution temporary storage for data and instructions. It is perfect for storing variables, stacks, and piles of data because it enables quick read and write operations. Since RAM is volatile, the contents will be no longer saved and will be lost when the power is turned off or the computer is restarted. The system's capacity to handle complex programs and multitask is directly influenced by the RAM size. Larger data storage and more efficient task execution are made possible by larger size of RAM.

Two known types of Random Access Memory (RAM):

- SRAM (Static Random Access Memory)
- DRAM (Dynamic Random Access Memory)



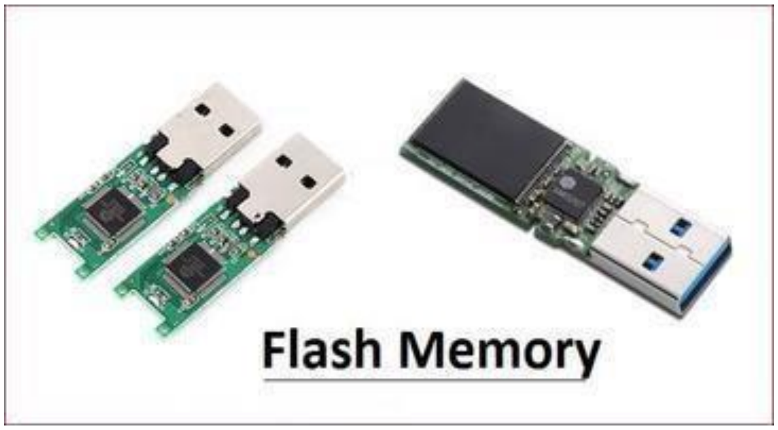
Difference between SRAM and DRAM.



SRAM	DRAM
1. SRAM has lower access time, so it is faster compared to DRAM.	1. DRAM has higher access time, so it is slower than SRAM.
2. SRAM is costlier than DRAM.	2. DRAM costs less compared to SRAM.
3. SRAM requires constant power supply, which means this type of memory consumes more power.	3. DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
4. Due to complex internal circuitry, less storage capacity is available compared to the same physical size of DRAM memory chip.	4. Due to the small internal circuitry in the one-bit memory cell of DRAM, the large storage capacity is available.
5. SRAM has low packaging density.	5. DRAM has high packaging density.

c.) Flash Memory

Flash memory is a type of non-volatile memory that combines RAM and ROM features. It is suitable for storing and updating firmware, configuration data, and user data because it supports both read and write operations. As flash memory can be rewritable, a large amount of storage, and relatively quick access times, it is frequently used in embedded systems. The operating system, program code, data files, and settings are typically stored on it. The lifespan of flash memory must be extended using methods like wear leveling and error correction because frequent write operations can cause wear and tear. “Wear and Tear” is a process in flash memory where it can degrade over time due to the number of times it is erased and rewritten. One best example of flash memory is the USB drives.



These memory types cooperate in an embedded system to maintain proper operation. Flash memory offers non-volatile storage for firmware updates and user data, ROM stores permanent software and initialization code, and RAM serves as temporary storage for data and program execution. For a system to operate effectively, these memory types must be managed properly, and the sizes and access times of these memory types must be carefully considered in order to satisfy the requirements of the embedded application.

II. Storage Considerations and Trade-offs

Comparison Table

Parameters	**RAM**	**ROM**	**Flash Memory**
Volatility	Volatile	Non-volatile	Non-volatile
Physical chip size	Bigger	Smaller	Smaller
Storage capacity	Higher	Lower	Higher
Speed	Faster	Slower	Faster
Cost	Expensive	Cheaper	Cheaper

Storage Capacity: ROM and Flash memory generally offer larger storage capacities compared to RAM. ROM stores permanent data and instructions, while Flash memory provides non-volatile storage for firmware and user data. The available storage capacity of these memories determines the amount of code, data, and settings that can be stored in the system.

Read/Write Access Speed: RAM provides the fastest read/write access speed among the three memory types. It allows for rapid data manipulation and program execution. ROM and Flash memory, on the other hand, have slower access times. While reading from ROM and Flash memory is relatively fast, writing to these memories is typically slower due to the programming and erasing operations involved.

Volatility and Persistence: RAM is volatile, meaning its contents are lost when power is removed, or the system is reset. ROM and Flash memory are non-volatile, ensuring that data and instructions are retained even without power. Flash memory, however, may require wear-level algorithms and error correction mechanisms to maintain data integrity over repeated write cycles.

Write Endurance: Flash memory has a limited number of write cycles before it becomes unreliable. Therefore, it is crucial to manage write operations carefully and employ techniques like wear leveling to distribute writes evenly across the memory cells. RAM and ROM do not have such limitations as they are not subject to wear and tear due to writing cycles.

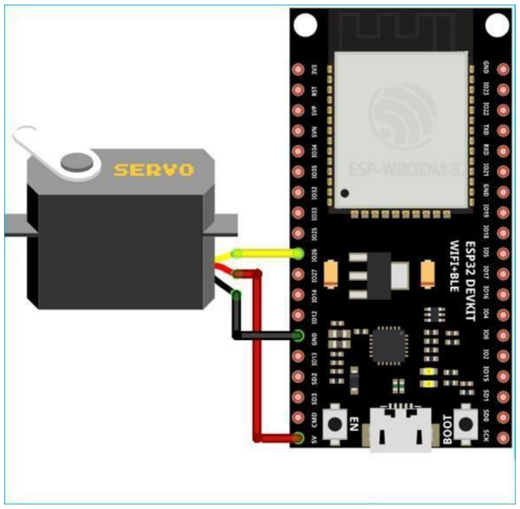
Flexibility and Upgradability: RAM is highly flexible and can be dynamically allocated and deallocated during program execution. It allows for runtime data manipulation and efficient memory management. Flash memory enables firmware updates and allows user data storage, providing upgradability and customization options. ROM, however, typically contains fixed and unchangeable data.

Cost: Cost considerations can vary among these memory types. RAM is typically more expensive than ROM and Flash memory due to its faster access times and volatile nature. ROM and Flash memory are more cost-effective for storing large amounts of permanent or non-volatile data.

ACTIVITIES / EXAMPLES

You are now introduced to microcontrollers and various sensors. Here are a few examples of activities you could try.

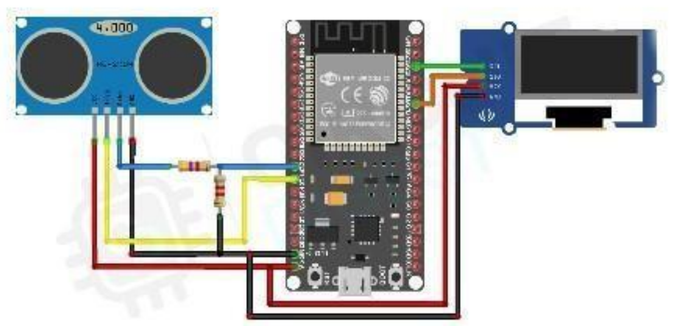
1.) Interfacing Servo Motor using ESP32: Users can control the servo motor's position and speed by connecting and programming the ESP32.



Sample source code for Interfacing Servo Motor using ESP32.

```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
// twelve servo objects can be
created on most boards int pos = 0;

void setup() {
 myservo.attach(26); // attaches the servo on pin 13 to the servo object
}

void loop() {
 for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
  // in steps of 1 degree
  myservo.write(pos);       // tell servo to go to
  position in variable 'pos' delay(15); // waits
  15ms for the servo to reach the position
 }
 for (pos = 180; pos >= 0; pos -= 1) { // goes
  from 180 degrees to 0 degrees
  myservo.write(pos);       // tell servo to go to
  position in variable 'pos' delay(15); // waits
  15ms for the servo to reach the position
 }
}
```

2.) Ultrasonic Sensor with ESP32: Ultrasonic sensors can be used with the ESP32 microcontroller to measure distance or detect presence in a variety of applications.

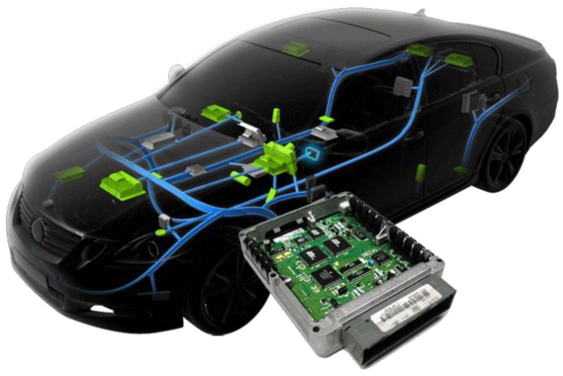


Sample source code for Interfacing Ultrasonic Sensor using ESP32.

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "NewPing.h"

#define TRIGGER_PIN 14
#define ECHO_PIN 27
// Maximum distance we want to ping for (in centimeters).
#define MAX_DISTANCE 400

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET -1

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Serial.begin(115200);
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
  Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }
}

void loop() {
  Serial.print("Distance = ");
  Serial.print(sonar.ping_cm())
  ; Serial.println(" cm");
  delay(300);
  update_display();
}

void update_display(){
  display.clearDisplay();
  display.setTextSize(2);
  delay(1);
  display.setTextColor(WHITE);
  delay(1);
  display.println(F("Distance"));
  delay(1);
  display.println(sonar.ping_cm());
  delay(1);
  display.println("cm");
  delay(1);
  display.display();
  delay(500);
}
```

SUMMARY

This module provides a comprehensive overview of crucial components in embedded systems. It covers microcontrollers and microprocessors, which serve as the brains of these systems, enabling control and computation. The module also delves into sensors and actuators, which facilitate the interaction between the embedded system and its environment. Lastly, it explores memory devices, crucial for storing and retrieving data in embedded systems.



Real-life applications of the topics can be found in various fields. For instance, in the automotive industry, embedded systems with microcontrollers and microprocessors are utilized in engine control units (ECUs), providing precise control over fuel injection, ignition timing, and emissions. Sensors and actuators play a crucial role in monitoring and adjusting various parameters in automobiles, such as temperature, pressure, and position.

In the field of home automation, embedded systems with microcontrollers and sensors enable the control of lighting, temperature, and security systems. They can detect motion, measure ambient light, and adjust settings, accordingly, providing convenience and energy efficiency.

Medical devices heavily rely on embedded systems. Microcontrollers and microprocessors are used in pacemakers to regulate heart rhythm, while sensors monitor vital signs such as blood pressure and oxygen levels. Actuators control drug delivery systems and prosthetic limbs, enhancing patient care and quality of life.

Overall, understanding the hardware fundamentals for embedded systems is crucial for designing and developing innovative solutions across industries, improving efficiency, automation, and enhancing the user experience. Embedded systems play a vital role in various industries as they are responsible for managing and controlling complex tasks within devices.

SUMMATIVE ASSESSMENT (SA)

I. Multiple Choice

_________1. Which of the following best describes a microcontroller?

a) A type of display device used in small computers for programming and calculations.

b) A chip with memory but lacking the ability to perform calculations or programming.

c) A small computer with memory and the ability to perform calculations through programming.

d) A large computer with advanced memory and processing capabilities.

_________2. This component is used to manage the retrieval, decoding, and execution of instructions that are stored in memory.

a) Flash Memory

b) Microprocessor

c) Sensor

d) RAM

_________3. The data of this kind of ROM is removed using UV (ultraviolet) rays.

a) EPROM

b) PROM

c) MROM

d) EEPROM

_________4. Where does the driving force of pneumatic actuator comes from?

a) Gaseous fluid

b) Oil

c) Pistons

d) Vacuum

_________5. This sensor is used to measure the velocity and distance of an object.

a) Ultrasonic Sensor

b) Proximity Sensor

c) Infrared Sensor

d) Alcohol Sensor

II. Fill in the Blanks.

The CPU of a 6. ____________________is used to control multiple instruction pipelines to execute several operations simultaneously during a clock cycle.

In an IoT system, sensors are used to gather 7. _______________data, the 8. _________ will then process the data to generate control signals, and actuators will utilize 9. energy to perform the required actions.

A system's capacity to handle complex programs and multitask is directly influenced by the 10. ___________.

LAB ACTIVITY

1) Place a small piece of tape on the servo disc so you can clearly see its motion. Then write a program that moves the servo (servo #1 = pin 7) as follows for 60s:

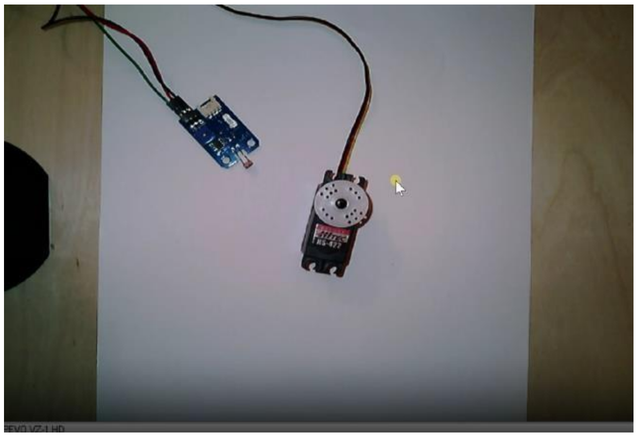
   th1_d = A * (1.0 + sin(w * t + phi1));

   Where A = 45 deg, w = 1 rad/s, phi1 = 0, and t is the time in seconds read from the Arduino clock. The servo motor should move with a sinusoidal motion of 45 degrees in amplitude.

2) Attach the light sensor to the servo disk (using tape, weak glue, etc.). Write a program that maximizes the light to the sensor by adjusting the angle of the servo. Test the program by placing a stationary light (LED, cell phone, etc.) near the sensor. Plot light level vs time in Excel (or another spreadsheet program such as LibreOffice).

   Note: There are two basic approaches you can use to maximize the light. The first approach slowly moves the servo over its complete range and measures the light for each angle. It then moves to the angle with the highest light measurement.

   The second approach measures the light for three consecutive angles in a small range (theta - d, theta, and theta + d) a small distance apart (e.g., d = 3 deg), where theta is the current angle of the servo. The servo then moves to the angle with the highest light level. This process is then continually repeated to slowly maximize the light level. Different values of d will result in different convergence times. You can use either approach (or try both) to solve this question.
   Make sure to give the servo motor enough time to reach its destination. This reaching time will be directly proportional to the distance that must be traveled since the last command. However, any sufficiently large value such as 100 ms will work.



REFERENCES / BIBLIOGRAPHY

Pedamkar, P. (2023). *Microcontroller Architecture*. EDUCBA. https://www.educba.com/microcontroller-architecture/

GeeksforGeeks. (2023). *Harvard Architecture*. GeeksforGeeks. https://geeksforgeeks.org/harvard-architecture/

Williams, P. (2022, September 5). *A Review of Von Neumann & Harvard Architectures*. study.com. Retrieved July 10, 2023, from https://study.com/learn/lesson/von-neumann-vs-harvard-architectures-differences-uses-examples.html

GeeksforGeeks. (2023b). *Computer Organization Von Neumann architecture*. GeeksforGeeks. https://www.geeksforgeeks.org/computer-organization-von-neumann-architecture/

Arm Ltd. (n.d.). *What is RISC?* Arm | the Architecture for the Digital World. https://www.arm.com/glossary/risc

Ginni. (2021, July 27). *What is CISC Processor*. https://www.tutorialspoint.com/what-is-cisc-processor

What is superscalar architecture? (2018, January 18). The Trustees of Indiana University. https://kb.iu.edu/d/aett

GeeksforGeeks. (2021). *Superscalar Architecture*. GeeksforGeeks. https://www.geeksforgeeks.org/superscalar-architecture/

Gwentech Embedded. (2020). *Applications of Embedded Systems in Various Industries – Gwentech Embedded*. Gwentechembedded.com. http://gwentechembedded.com/applications-of-embedded-systems-in-various-industries/

M-ISS Admin. (2022, June 20). *Industrial Applications of Embedded Systems - A Few Examples - M-ISS*. M-ISS. https://m-iss.in/embedded-systems/industrial-applications-of-embedded-systems-a-few-examples/

Editor, I. (2022, August 24). Sensors and actuators. THE INSTRUMENT GURU. https://theinstrumentguru.com/sensors-and-actuators/

Davis, J. (2023, June 29). What’s the difference between sensors and actuators? Managerplus. https://managerplus.iofficecorp.com/blog/sensors-actuators

Donovan, D. (2020, December 31). What is a solenoid?. Sciencing. https://sciencing.com/what-solenoid-4902174.html

Teja, R. (2023, April 19). What is a sensor? different types of sensors and their applications. ElectronicsHub. https://www.electronicshub.org/different-types-sensors/

YoungWonks. (n.d.). Essentially mechanical or electro-mechanical devices that allow controlled movements or positioning, actuators are a key component in several devices today. YoungWonks. https://www.youngwonks.com/blog/What-is-an-actuator-and-What-are-the-Different-Types-of-Actuators

Hathaway, L. (2021, May 28). Differences Between RAM, ROM, And Flash Memory: All You Need To Know. Hackernoon.com. https://hackernoon.com/differences-between-ram-rom- and-flash-memory-all-you-need-to-know-ghr341i

Mask ROM - HandWiki. (2013). Handwiki.org. https://handwiki.org/wiki/Mask_ROM

Different Types of RAM Random Access Memory. (2018, April 9). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/different-types-ram-random-access-memory/

Bhandari, S. (2023). Email. Askanydifference.com. https://askanydifference.com/difference-between-ram-and-rom-and-flash-drive/

Embedded System Software Development

INTRODUCTION

Embedded system software is defined as specialized programming tools in embedded devices that facilitate machine operation. The software is in charge of managing various hardware devices and systems. The basic idea behind embedded systems software is to control the operation of a collection of hardware devices without sacrificing purpose or efficiency.

Embedded system software is similar to computer operating systems. Embedded systems software, similar to how operating systems control software applications in computers, controls various devices and ensures their smooth operation. Ideally, this software does not require user input and can operate autonomously based on predefined parameters.

LEARNING OBJECTIVES / OUTCOMES

At the end of the module, students should be able to:

- To know what the application and function Real-time operating system are.
- To know how to work and efficiency in the Bare-metal programming system.
- Understand what an embedded programming language is and discuss the pros and cons of popular embedded programming languages.
- Pick the right programming language for their embedded project and write codes in MicroPython language for their embedded system.
- Become familiar with the Integrated Development Environment and to create a standard executable.
- To be able to understand the types of commands contained in the menus and the tool bar.

DISCUSSION OF THE LESSON

EMBEDDED SYSTEM SOFTWARE ARCHITECTURE

I. Real-time operating systems (RTOS)

A real-time operating system (RTOS) is an operating system that ensures real-time applications have a certain capability within a specified time frame. RTOSes are intended for critical systems and timing-sensitive devices such as microcontrollers. The processing time required by an RTOS is measured in milliseconds. Any delay in responding could have disastrous results. Real-time operating systems perform similar functions to general-purpose operating systems (GPOSes) such as Linux, Microsoft Windows, or macOS, but are designed so that an OS scheduler can meet specific deadlines for various tasks.

RTOSes are also commonly found in embedded systems, which are a combination of hardware and software designed to perform a specific function and may also operate as part of a larger system. Embedded systems are frequently used in real-time environments and communicate with the hardware via a real-time operating system. RTOSes are designed to handle multiple processes at once, ensuring that these processes respond to events within a predetermined time frame. RTOS processes tasks within defined time constraints and monitors task priority. RTOS can also change the priority of tasks. Event-driven systems frequently switch between tasks based on priority.

Some real-time operating systems are designed for specific applications, while others are designed for general use. RTOSes typically include the following features:

- Multitasking is the practice of rapidly switching between tasks in order to give the impression that multiple programs are running concurrently.
- Prioritization of process threads; and
- a sufficient number of interrupt levels.

RTOSes are included in the following devices:

air traffic control systems, anti-lock brakes and air bags, cameras, medical systems.

Characteristics of a real-time operating system

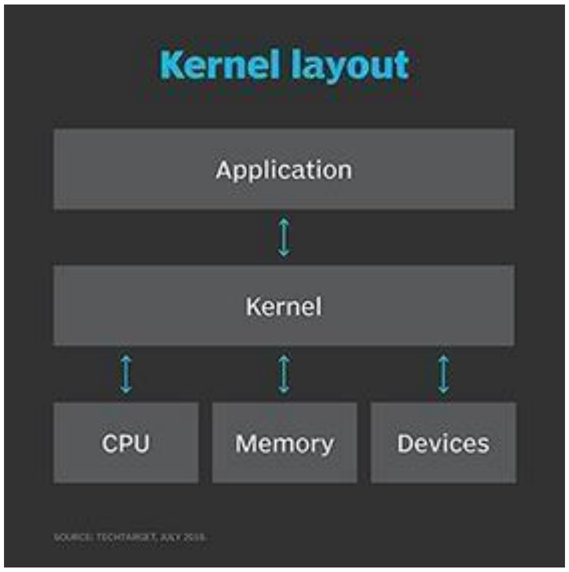
Small footprint. Compared to general OSes, real-time operating systems are lightweight.

High performance. RTOS are typically fast and responsive.

Determinism. Repeating inputs end in the same output.

Safety and security. Safety-critical and security standards are typically the highest priority, as RTOS are frequently used in critical systems.

Priority-based scheduling. Tasks that are assigned a high priority are executed first followed by lower-priority jobs.



RTOS works are classified as soft or hard real-time systems. A soft RTOS is designed to provide predictable response times in the hundreds of milliseconds, whereas a hard RTOS is designed to provide predictable response times in tens of milliseconds When compared to hard RTOSes, soft real-time systems typically have larger file sizes. During peak load, some executions behave less predictably but are tolerated because computations are rolled back to previously established checkpoints if an error occurs. Soft RTOSes are typically used in systems where time-based executions are not critical, such as PCs, cameras, and smartphones.

Data files in hard RTOSes are typically small or medium in size. They behave predictably during peak loads, and if an error occurs, the computation is rolled back. Hard RTOSes are typically used in systems requiring critical time-based executions, such as aeroplane sensors, autopilot systems, or medical devices.

In hard real-time operating systems, if the calculation to make an object available at the specified time cannot be performed, the OS terminates with a failure. The OS continues to function in a soft RTOS, but certain tasks may be ineffective if they fail to execute at a specific time. RTOSes also serve as a scheduler, allowing tasks to be marked as ready to run, running, or blocked.

II. **Bare-Metal Programming**

The creation of bare metal software results in the creation of an embedded system that generates a real-time operating system (RTOS). At the lowest system level, an RTOS is code written on bare metal and directly to the hardware of choice. Such software is created by developers using bare metal programming, which is an essential option for design teams.

An RTOS's typical behavior is an infinite loop. It has terminating tasks and infinite loop tasks, but the main loop task is continuous. Because of their ability to loop or terminate running operations based on the stated condition, RTOS are also known as schedulers. A do-while loop is an example of this programming statement. This type of loop runs a specific block of code at least once before repeating it or terminating it based on Boolean (true or false) conditions at the end of the code.

Instead of a virtualization layer, bare metal applications run directly from the preloader, allowing full customization to increase workloads. Furthermore, this feature enables the activation and deactivation of any instance at the lowest level, ensuring that the bare metal code is consistent with the appropriate infrastructure.

Higher security is included in bare metal software to ensure the efficient completion of simplified operations with advanced memory handling. This automated task has the potential to outperform hosted hypervisors. Furthermore, because it reduces the attack surface, bare metal software is more secure. It has a single-tenant environment, which makes it more powerful than a regular virtual machine.

Working of Bare-metal Programming

Typically, bare metal programs include a minimal bootloader that initiates the clock, memory, and processor before proceeding to the main program. This action allows the creation of a memory map of microcontroller hardware registrars. However, to perform initial hardware initialization and memory stack/heap setup, bare metal application embedded systems require a startup file.

Bootloaders and firmware are examples of bare metal programming. The direct code connects all components that lead to a hardware device's initialization or startup. This bare metal code is similar to machine language. It does not need to go through additional software layers and instead goes straight to the system.

Benefits of Bare Mental Programming

There are numerous advantages to bare metal programming. First, it is not created from scratch, implying that tools were used to assist in its creation. In addition, libraries are combined with bare metal to provide basic functionality such as file structures in peripherals and design configuration. This functionality allows developers to devote more time to providing custom settings and adapting scripts for businesses.

The code is simple and runs on the chip or microcontroller directly. Because of its single-tenant nature and increased code safety, it is also scalable. Furthermore, this action requires less memory and uses less power when controlling hardware microcontrollers.

A. Programming Languages for Embedded System

What is embedded programming language?

Embedded programming languages are designed specifically for microcontroller-based applications and other devices that require low power and minimal memory usage. These languages provide instructions to the processor so that it can execute operations with the help of a compiler, interpreter, or direct machine language.

The embedded language programming can be seen in many devices. However, it is hard to tell which language is the best option for a specific embedded device, it all depends on the complexity of the system, and the requirements of the client.

There are several languages for embedded systems that are relevant for embedded systems. These languages have become an essential tool for developers working on embedded systems. They offer a variety of advantages, such as the ability to integrate hardware and software components, but they also present some limitations.

I. Python/Micropython

Developers need to keep in mind that there is no single 'flavor' of the Python programming language. There are many implementations, and some are explicitly built to perform on embedded platforms superbly. The future of Python as a major player in the professional embedded development world hinges on proving that it can meet or exceed the performance of C. We can look to history to provide some insights. After all, the shift from assembly to C did not occur overnight. Assembly is still used for performance-crucial sections of a C program by inserting assembly code inside C macros. Python and C can work together similarly.

Another consideration is that the microcontrollers are light years more powerful than the microcontrollers of ten or twenty years ago. Clock cycles have gone from being measured in a few megahertz to over a gigahertz. Flash memory, once measured in dozens of kilobytes, is now measured in several megabytes. Fortunately, the significant increase in performance has not come with a commensurate increase in price. Thus, embedded application developers can write higher-level code without minding every clock cycle or byte of memory necessarily and still turn out responsive, high-quality products.

Python

● Interpreted language. Developed in the early 1980’s (named after the British comedy troop Monty Python), Python is a popular programming language. It excels in machine learning, artificial intelligence (AI), and data analytics, but you can use it in many other applications. It is used in application processors and as a communication vehicle between the user and the embedded system.

● Strength: Cloud, and data science.

● Pros: Developers do not have to cross-compile code for embedded systems. The language is open source, free to use, and easy to learn, read, and write.

● Cons: This language is not deterministic – not for real-time operating systems (RTOS). Lacks a compiler and static analyzers that might find problems during compilation. It must be tested well to avoid runtime errors.

Micropython

● This language is a version of Python optimized for microcontrollers.

● Strength: Cloud, and data science

● Pros: It is also open source, free to use, and easy to learn. It has libraries built in to support many tasks.

● Cons: The code is not as fast and may use more memory compared to C or C++.

MicroPython is an implementation of Python 3 created to work in the microcontroller environment and its constraints. It has a subset of the Python libraries as well as libraries to interface with low level hardware and network interfaces. (ex: Wifi)

Cards supporting MicroPython:

● ESP8266 Boards (ex: NodeMCU ESP8266)

● PyBoard Micro:Bit

● Teensy 3.X

● WiPy – Pycom

● Raspberry Pi Pico

II. **High-Level Languages and Their Trade-Offs**

C

● A compiled language. Simple and very widely used. A building block of many other languages. It is used in many embedded systems.
● Strength: Embedded, bare-metal, Internet of Things
● Pros: Highly efficient programming language, very stable and fast.
● Cons: Modern coding techniques are challenging to implement within the language.

C++

● A compiled language. It has most or all elements of C and other capabilities that C does not have. It is also used in many embedded systems.

● Strength: Embedded, bare-metal, standalone apps, Internet of Things
● Pros: It has a powerful standards library, which saves programmers time in writing code. C++ can be as efficient as C.
● Cons: Complex language that can be difficult to learn.

Java

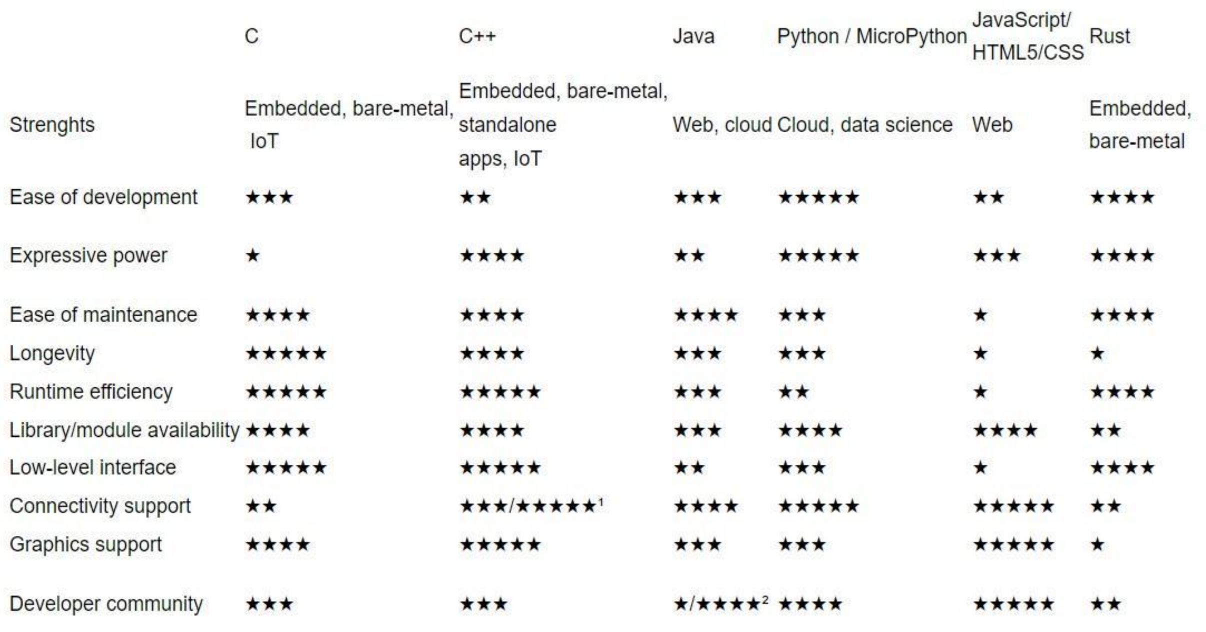
● Java is an efficient, general-purpose language used extensively for internet-based applications. In embedded systems, Java is best for those running on the Android Operating System.
● Strength: Web, and cloud.
● Pros: Once written in an embedded system, the code is portable to another device and quite reliable.
● Cons: The language can be complex, leading to performance issues, including those with graphical user interfaces (GUIs). You can not use Java in real-time systems.

Javascript

● JavaScript is a text-based programming language used in some embedded systems. It is especially used in systems with a human-machine interface.
● Strength: Web
● Pros: A good option if your system is based on HTML5 and requires significant networking and graphics.
● Cons: Poor runtime efficiency and can be challenging to maintain.

Rust

● In 2010, a Mozilla employee developed this high-level programming language designed for performance and safety. It has many modern programming language capabilities and security features. It will take time before it finds more uses in embedded systems.
● Strength: Embedded, and bare-metal.
● Pros: Rust helps encourage secure code with fewer bugs. It helps encourage secure code with fewer bugs.
● Cons: Rust takes time to compile. The language isn't standardized or used much yet.



	C	C++	Java	Python / MicroPython	JavaScript/ HTML5/CSS	Rust
Strenghts	Embedded, bare-metal, IoT	Embedded, bare-metal, standalone apps, IoT	Web, cloud	Cloud, data science	Web	Embedded, bare-metal
Ease of development	★★★	★★	★★★	★★★★★	★★	★★★★
Expressive power	★	★★★★	★★	★★★★★	★★★	★★★★
Ease of maintenance	★★★★	★★★★	★★★★	★★★	★	★★★★
Longevity	★★★★★	★★★★	★★★	★★★	★	★
Runtime efficiency	★★★★★	★★★★★	★★★	★★	★	★★★★
Library/module availability	★★★★	★★★★	★★★	★★★★	★★★★	★★
Low-level interface	★★★★★	★★★★★	★★	★★★	★	★★★★
Connectivity support	★★	★★★/★★★★★[1]	★★★★	★★★★★	★★★★★	★★
Graphics support	★★★★	★★★★★	★★★	★★★	★★★★★	★
Developer community	★★★	★★★	★/★★★★[2]	★★★★	★★★★★	★★

★ is the lowest ranking while ★★★★★ is the highest.

www.qt.io/embedded-development-talk/embedded-software-programming-languages-pros-cons-and-comparisons-of-popular-languages

B. Software Development Tools and Debugging Techniques

I. Integrated development environments (IDEs)

Why are IDEs important?

To write code, you may use any text editor. Nevertheless, the majority of integrated development environments (IDEs) include features beyond text editing. They offer a centralized interface for popular developer tools, greatly enhancing the effectiveness of the software development process. As opposed to painstakingly integrating and configuring various tools, developers may start creating new apps right away. Additionally, they may concentrate on just one application rather of having to learn about all the technologies. Using IDEs is common among developers for the reasons listed below:

- ***Code Editing Automation*** – Programming languages have rules for how statements must be structured. Because an IDE Knows these rules, it contains many intelligent features for automatically writing or editing the source code.

- ***Syntax Highlighting*** – An IDE can format the written text by automatically making some words bold or italic, or by using different font colors. These visual cues make the source code more readable and give instant feedback about accidental syntax errors.

- ***Intelligent Code Completion*** – Various search terms show up when you start typing words in a search engine. Similarly, an IDE can make suggestions to complete a code statement when the developer begins typing.

- ***Refactoring Support*** – Code refactoring is the process of restructuring the source code to make it more efficient and readable without changing its core functionality. IDEs can auto-refactor to some extent, allowing developers to improve their code quickly and easily. Other team members understand readable code faster, which support collaboration within the team.

- ***Local Build Automation*** – By carrying out repetitive development chores that are normally included in every code update, IDEs boost programmer productivity. Here are a few instances of typical coding jobs that an IDE does.

- ***Compilation*** - The operating system may comprehend the code once it has been compiled or converted by an IDE. Some programming languages provide just-in-time compilation, where the IDE transforms application-level human-readable code into machine code.

- ***Testing*** - Prior to integrating the product with the code of other developers and running more involved integration tests, developers can locally automate unit tests using the IDE.

How should I choose an IDE?

Modern integrated development environments (IDEs) are widely available on the market, offering a variety of functionalities at various price ranges. A lot of IDEs are open source, or free to download, install, and use. When selecting an IDE, keep the following factors in mind:

- ***The Programming Language*** – The choice of an IDE is frequently influenced by the programming language you intend to use. Automation elements in specialized IDEs are especially suited to the syntax of languages. IDEs that handle many languages, however, are another option.

- ***The Operating System*** - Even while most IDEs provide versions for several operating systems, some platforms may benefit from them more than others. For instance, some IDEs may operate best on the Linux platform but may be sluggish or challenging to use on other systems.

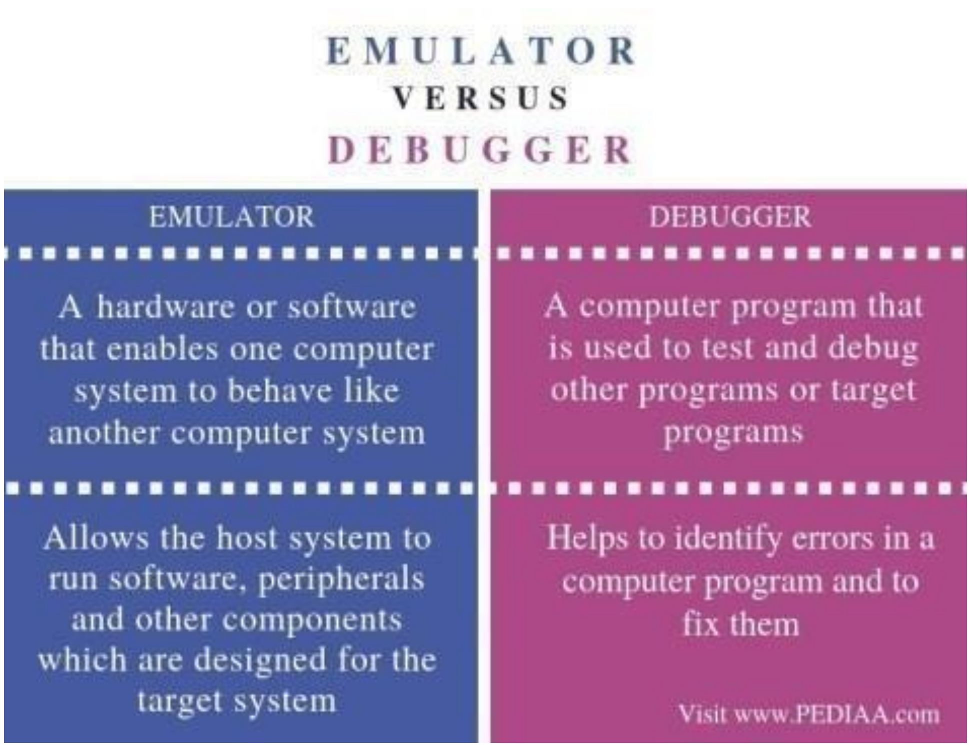
- ***Automation Features*** - The source code editor, build automation, and debugger are the three functionalities that are included in most IDEs. The following additional characteristics are possible and can vary:
    - Code Editor UI Enhancements
    - Automated Testing Features
    - Code Deployment Support via plugin integration
    - Code Refactoring Support

  - Application Packaging Support

- ***IDE Customization*** - To suit a developer's requirements and tastes, certain IDEs provide the option of customizing processes. Plugins, extensions, and add-ons are available for download and usage to personalize your programming environment.

II. Debuggers and emulators

A system with hardware and software embedded in it is referred to as an embedded system. It employs a microcontroller or a microprocessor to carry out a certain purpose. It also contains peripherals to connect components, memory, hardware, software, and other components. It also includes actuators, analog to digital converters, digital to analog converters, sensors, and other devices. Debugger and emulator are the two primary tools that facilitate the development of embedded systems.



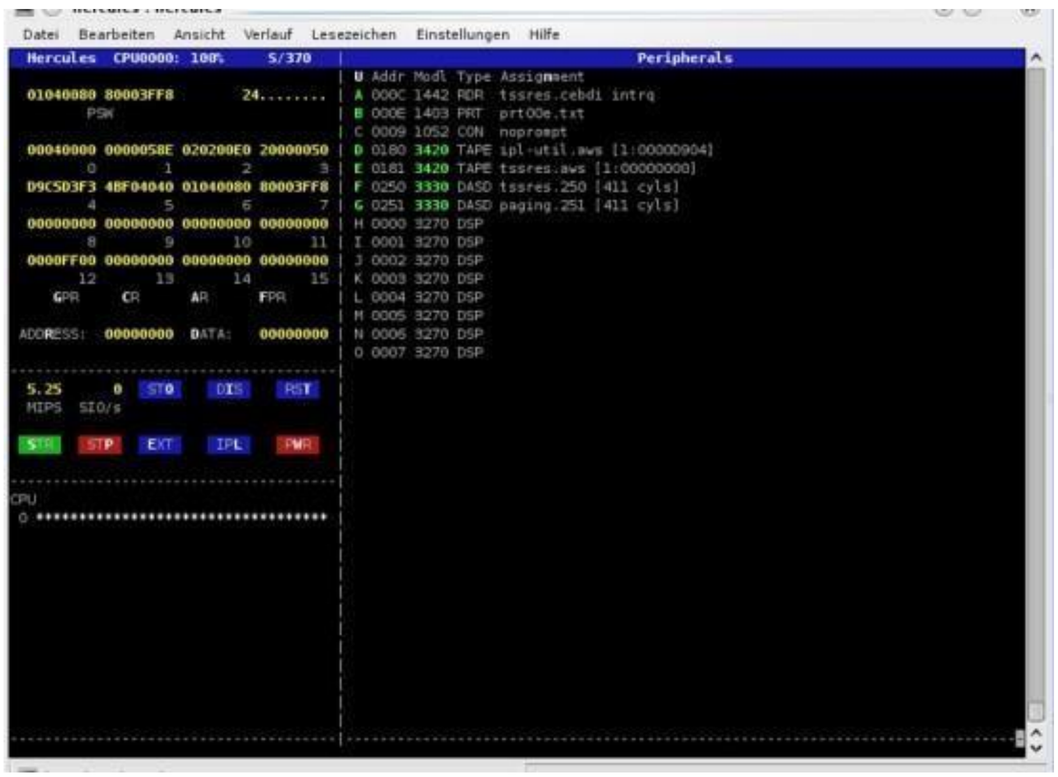
EMULATOR VERSUS DEBUGGER

EMULATOR	DEBUGGER
A hardware or software that enables one computer system to behave like another computer system	A computer program that is used to test and debug other programs or target programs
Allows the host system to run software, peripherals and other components which are designed for the target system	Helps to identify errors in a computer program and to fix them

Visit www.PEDIAA.com

What is an Emulator?

One computer system can operate like another computer system using an emulator. A hardware emulator offers the target system's environment. It also includes a microprocessor, RAM, and associated interface circuitry.



```
Hercules  CPU0000: 100%      S/370    |                         Peripherals
                                      | U Addr Modl Type Assignment
01040080 80003FF8         24........  | A 000C 1442 RDR  tssres.cebdi intrq
      PSW                             | B 000E 1403 PRT  prt00e.txt
                                      | C 0009 1052 CON  noprompt
00040000 0000058E 020200E0 20000050   | D 0180 3420 TAPE ipl-util.aws [1:00000904]
       0        1        2        3   | E 0181 3420 TAPE tssres.aws [1:00000000]
D9C5D3F3 4BF04040 01040080 80003FF8   | F 0250 3330 DASD tssres.250 [411 cyls]
       4        5        6        7   | G 0251 3330 DASD paging.251 [411 cyls]
00000000 00000000 00000000 00000000   | H 0000 3270 DSP
       8        9       10       11   | I 0001 3270 DSP
0000FF00 00000000 00000000 00000000   | J 0002 3270 DSP
      12       13       14       15   | K 0003 3270 DSP
   GPR      CR       AR      FPR      | L 0004 3270 DSP
                                      | M 0005 3270 DSP
ADDRESS:  00000000  DATA:   00000000  | N 0006 3270 DSP
                                      | O 0007 3270 DSP
--------------------------------------|
5.25       0   STO    DIS    RST      |
MIPS   SIO/s                          |
                                      |
STR    STP    EXT    IPL    PWR       |
--------------------------------------|
CPU                                   |
 0 ***********************************|
```

A targeted system and the processor remain unrelated to a circuit used to emulate that system. Consider, for instance, that an IC in an electronic circuit has a problem. It is feasible to use an emulator to observe the behavior of the IC rather than physically inserting it. If the system functions well, the actual application may be put into use. Overall, an emulator offers flexibility and streamline system development.

What is a Debugger?

A debugger is a tool that aids software testing and debugging. A host computer creates embedded software. After going through this development process, it becomes executable code. It is then incorporated into the target machine. To debug an embedded system, you need to meet three criteria. There are three requirements to debug an embedded system. They are as follows.

Run control – The ability to start, stop, peak the processor and memory

Memory substitution – Replacing the ROM based memory with RAM for rapid and easy code download and repair cycles

Real-time analysis – Following code flow in real time with real-time trace analysis

A debugging system should have two processes. They are the test program and the debugger. In other words, they are the debug kernel in the target and the host application that communicates with it. It is important to run debugger as a separate process and to provide a separate execution unit to run a debugger.

Difference Between Emulator and Debugger Definition:

An emulator is hardware or software that enables one computer system to behave like another computer system. In contrast, a debugger is a computer program that helps to test and debug other programs or target programs. This is the basic difference between emulator and debugger.

Usage:

An emulator allows the host system to run software, peripherals and other components which are designed for the target system. Meanwhile, a debugger helps to identify errors in a computer program and to fix them. This is another difference between emulator and debugger.

ACTIVITIES / EXAMPLES

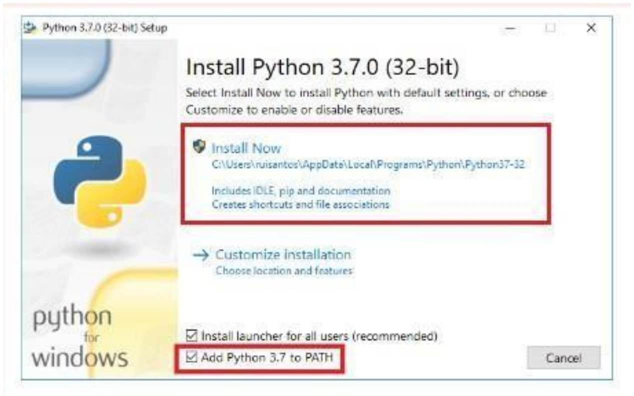
Install MicroPython with uPyCraft

MicroPython is a lean and efficient implementation of the Python 3, programming language that is optimized to run on microcontrollers and other resource-constrained environments. It allows developers to write Python code that can run directly on microcontrollers without the need for an operating system or additional hardware.

On the other hand, uPyCraft is an integrated development environment (IDE) that provides a user-friendly interface that allows developers to write, test, and debug their code with ease. uPyCraft also offers features such as syntax highlighting, auto-completion, and code folding, which make the coding process more efficient and enjoyable.

To use MicroPython on ESP32, we must set some things up first. The first step is to download and install the Python and uPyCraft:

1. Go to https://www.python.org/downloads/
2. Enable the option at the bottom "Add Python (3.x.x) to PATH". Then, press the "Install Now" button:



Python 3.7.0 (32-bit) Setup

Install Python 3.7.0 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\ruisantos\AppData\Local\Programs\Python\Python37-32
Includes IDLE, pip and documentation
Creates shortcuts and file associations

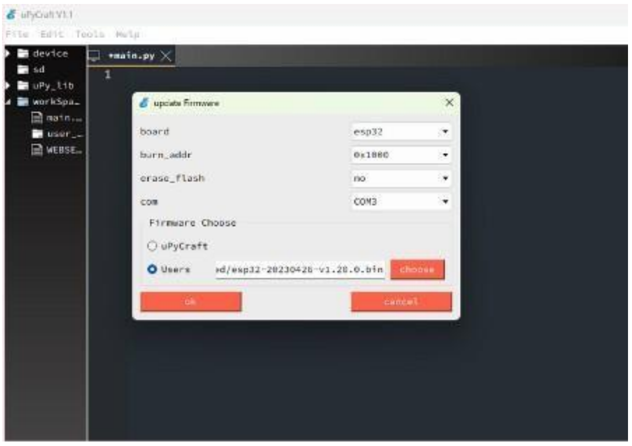
Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

3. Go to https://randomnerdtutorials.com/install-upycraft-ide-windows-pc-instructions/ to download and install uPyCraft.

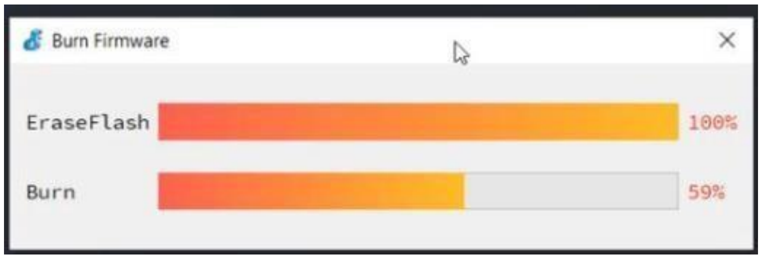
After installing the two components, we can now flash MicroPython into our ESP32. Below are the steps to follow:

1. Download MicroPython .bin file from https://micropython.org/download/esp32/
2. Connect the ESP32 to your computer and open the uPyCraft IDE.
3. Go to Tools > Burn Firmware.



board: esp32
burn_addr: 0x1000
erase_flash: no
com: COM3
Firmware Choose
( ) uPyCraft
(●) Users ›d/esp32-20230426-v1.20.0.bin

4. The firmware will start uploading. In some ESP32 boards, you need to press the boot button to get the process started.

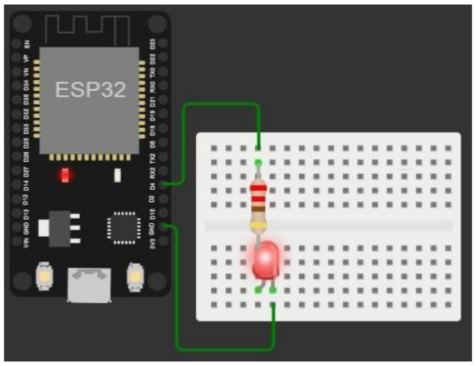


EraseFlash 100%
Burn 59%

LAB EXERCISE SETUP 1: Interfacing 1 LEDs

Connect 1 5mm LEDs to GPIO 4 of ESP32 through respective current limiting resistors (220Ω). GPIO 16 is labelled D4.

Interfacing Diagram:



Sketch for interfacing an LED on ESP32 using MicroPython

```
from machine import Pin import time

led = Pin (4, Pin.OUT)

while True:

led.on()

time.sleep(1)

led.off()

time.sleep(1)
```

As shown in the sketch above, interfacing an LED on ESP32 using MicroPython takes up significantly less lines compared to when we are using the Arduino IDE and C++. This is because the void functions are no longer required in MicroPython as opposed to what we typically do when programming using C++.

The following activity is encoded in Microsoft Visual Studio which is the IDE that is used to construct the following examples illustrated below.

2. Number Guessing in Python

The user must predict the generated random number (within a defined range) in this enjoyable introductory Python project after being given suggestions. The user receives more tips for each incorrect guess they make, but this lowers their overall score. Since this program generates random numbers using the Python random module, it's a wonderful method to practice using the standard library. Additionally, you may practice using user-defined functions, print formatting, and conditional expressions.

Number Guessing Game

```
import random
attempts_list = []
def show_score():
    if not attempts_list:
        print('There is currently no high score,'
              ' it\'s yours for the taking!')
    else:
        print(f'The current high score is'
              f' {min(attempts_list)} attempts')

def start_game():
    attempts = 0
    rand_num = random.randint(1, 10)
    print('Hello traveler! Welcome to the game of guesses!')
    player_name = input('What is your name? ')
    wanna_play = input(
        f'Hi, {player_name}, would you like to play '
        f'the guessing game? (Enter Yes/No): ')

    if wanna_play.lower() != 'yes':
        print('That\'s cool, Thanks!')
        exit()
    else:
        show_score()

    while wanna_play.lower() == 'yes':
        try:
            guess = int(input('Pick a number between 1 and 10: '))
            if guess < 1 or guess > 10:
                raise ValueError(
                    'Please guess a number within the given range')

            attempts += 1
            attempts_list.append(attempts)

            if guess == rand_num:
                print('Nice! You got it!')
                print(f'It took you {attempts} attempts')
                wanna_play = input(
                    'Would you like to play again? (Enter Yes/No): ')
                if wanna_play.lower() != 'yes':
                    print('That\'s cool, have a good one!')
                    break
                else:
                    attempts = 0
                    rand_num = random.randint(1, 10)
                    show_score()
                    continue
            else:
                if guess > rand_num:
                    print('It\'s lower')
                elif guess < rand_num:
                    print('It\'s higher')

        except ValueError as err:
            print('Oh no!, that is not a valid value. Try again...')
            print(err)


if name____== ' main ':
    start_game()
```

SUMMARY

A real-time operating system is most used in an embedded system, which is a system that operates behind the scenes of a larger operation, due to its benefits. RTOSs do not typically have graphical user interfaces. Occasionally, multiple operating systems are integrated concurrently to provide operational capability alongside the usability of a general-purpose OS. Intelligent edge devices, also known as electromechanical edge or cyber-physical systems, frequently use RTOSes. This means that the device is both creating and processing data. A car, for example, would be able to monitor its surroundings and react to them on its own. To enhance the capabilities of the underlying structure, such devices frequently combine artificial intelligence, machine learning, or both, with real-time components.

Developers must understand the hardware architecture as well as the specific microcontroller or processor used. Efficient Resource Utilization because bare metal programming has no operating system overhead, it allows for efficient resource utilization. The code can be fine-tuned by developers to maximize performance while minimizing memory footprint, making it suitable for resource-constrained embedded systems. In real-time applications that require deterministic and time-critical responses, bare metal programming is commonly used. Developers can achieve precise timing control and meet strict deadlines by eliminating the overhead of an operating system.

In conclusion, high-level programming languages offer numerous advantages for developing software for embedded systems. These languages, such as C/C++ and Python, provide developers with abstraction, portability, and ease of development. They enable faster prototyping, higher-level algorithm implementation, and code reusability, making them popular choices for embedded system development, especially for non-real-time and resource-rich applications.

Emulator	Debugger
It is a tool that allows one computer system to imitate or limit the functions of another computer system.	It is a tool that allows programmers to test and debug target programs.
It allows the host system to run software or use peripheral devices that are designed for guest system.	It allows the user to step through another program one line at a time.
It makes it possible to use android apps on Windows and Mac.	It makes it possible to run the target program under controlled conditions.
It generally helps to turn PC into Mac and lets you plan games from any era.	It generally helps to identify errors in computer programs and to fix them.
It emulates processors by using expensive bond-out processors connecting to plug.	It provides equivalent access on using on-chip debugging hardware with standard production processors.
It is simply used as a substitute for main controller or EPROM, etc.	It is simply used to reproduce conditions in which error has occurred and then to examine program state at that time and locate cause.
It can debug at controller HW level.	It is used for external signal debugging only.
It is important as it bridges the gap between devices that allows programs to work on all sorts of hardware.	It is important as it reports error condition as soon as possible.
It simply makes or creates virtual android devices on computer.	It simply makes the software development process stress-free and unproblematic.

SUMMATIVE ASSESSMENT (SA)

1. Software Development Environments are also known as an?

A. IDE
B. ABC
C. BDD
D. BFG

2. Which of these are features of a Integrated Development Environment?

A. Syntax Highlighting
B. Virus Scanning
C. Automatic Formatting
D. Automatic Color Coding

3. A program which helps locate, identify and rectify errors in a program

A. Loader
B. Debugger
C. Linker
D. Trace

4. Which IDE tool allows a programmer to enter, format and edit source code.

A. Linker
B. Compiler
C. Interpreter
D. Editor

5. Modern IDEs assist programmers by:

A. Automatically completing code
B. Writing programs for the user
C. Stopping any errors from being produced
D. Removing the need to write any code at all>

6. Which of the following devices can transfer the vector table from the EPROM?

A. ROM
B. RAM
C. CPU
D. peripheral

7. Which of the following is used to determine the number of memory access in an onboard debugger?

A. timer
B. counter
C. input
D. memory

8. Which of the following can use the high-level language functions, instructions instead of the normal address?

A. task level debugging
B. low level debugging
C. onboard debugging
D. symbolic debugging

9. Which of the following could use the high-level language functions and instructions instead of the normal address?

A. task level debugging
B. low level debugging
C. onboard debugging
D. symbolic debugging

10 Which of the following provides a low-level method of debugging software?

A. high-level simulator
B. low-level simulator
C. onboard debugger
D. CPU simulator

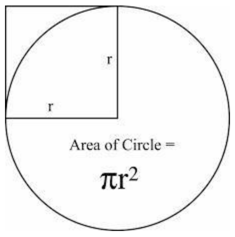
LAB ACTIVITY

1. Python Basic Pi of a Circle

Write a Python program that calculates the area of a circle based on the radius entered by the user.

Python: Area of a Circle

In geometry, the area enclosed by a circle of radius r is πr2. Here the Greek letter π represents a constant, approximately equal to 3.14159, which is equal to the ratio of the circumference of any circle to its diameter.



2. Design a circuit and write a program of a Running Light Sequencer consisting of 4 LEDs that will stop after 10 cycles.

Provide the following:

1. Interfacing Diagram

2. Source Code

3. Capture Screen snapshot of the running program and functioning circuit

REFERENCES / BIBLIOGRAPHY

Avila, R. (Aug 30, 2021). Embedded Software Programming Languages: Pros, Cons, and Comparisons of Popular Languages. Embedded Development Talk. www.qt.io/embedded-development-talk/embedded-software-programming-languages-pros-cons-and-comparisons-of-popular-languages

Erika (n.d.). A quick guide to embedded programming. DeepSee Developments. https://www.deepseadev.com/en/blog/embedded-programming-languages/

Parks, Mike (Oct 27, 2021). Introduction to Embedded Python. Bench Talk for Design Engineers. https://www.mouser.com/blog/intro-embedded-python

Python basic: Exercises, practice, solution. w3resource. (2011, February 7). https://www.w3resource.com/python-exercises/python-basic-exercises.php

The world's most engaging learning platform. Quizizz. (n.d.). https://quizizz.com/admin/quiz/60ca4c908487fe001d56d74c/integrated-development-environments-ides

Busbee, K. L. (2018, December 15). *Integrated development environment*. Programming Fundamentals. https://press.rebus.community/programmingfundamentals/chapter/integrated-development-environment/

The University. (1978). *What is an IDE?*. Amazon. https://aws.amazon.com/what-is/ide/#:~:text=An%20integrated%20development%20environment%20(IDE,easy%2Dto%2Duse%20application.

What is Embedded Systems Software?. HCLTech. (n.d.). https://hcltech.com/technology-qa/what-is-embedded-systems-software#:~:text=Embedded%20systems%20software%20can%20be,various%20hardware%20devices%20and%20systems.

Gillis, A. S. (2022, February 14). *What is a real-time operating system?*. Data Center. https://www.techtarget.com/searchdatacenter/definition/real-time-operating-system#:~:text=A%20real%2Dtime%20operating%20system%20(RTOS)%20is%20an%2 0OS,requirements%20are%20measured%20in%20milliseconds.

S, A. K. (2022, September 30). *Everything to know about bare metal programming: Skill-lync blogs*. Skill. https://skill-lync.com/blogs/everything-to-know-about-bare-metal-programming

Love, C. (2020, April 2). *Techniques to teach debugging strategies.* TechnoKids Blog. https://blog.technokids.com/programming/techniques-to-teach-debugging-strategies/

GeeksforGeeks. (2022, December 26). *Difference between emulator and Debugger*. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-emulator-and-debugger/

Interfacing and Communication in Embedded Systems

INTRODUCTION

Serial communication protocols facilitate data transfer between microcontrollers and peripheral devices in embedded systems. This module discusses the three commonly used serial communication protocols: UART (Universal Asynchronous Receiver-Transmitter), SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit), as well as their application in communications with peripheral devices. This module will also cover topics regarding protocols used in wireless protocols, how they may be used in different applications, and analog-to-digital conversion and signal condition techniques.

LEARNING OBJECTIVES / OUTCOMES

At the end of the module, students should be able to:

- understand serial communication protocols and their significance to embedded systems, implementation of UART, SPI, and I2C, and the process of initiating communication with peripheral devices using serial communication protocols
- differentiate the Wi-Fi, Bluetooth, and Zigbee Wireless Communication Protocols and the applications of the above-mentioned Wireless Communication Protocols
- understand the considerations one must think of with regards to IoT connectivity
- learn what sensor data acquisition is about, process acquired data from sensors, convert analog to digital data, and the techniques in converting analog signal to digital
- know the importance of analog to digital conversion

DISCUSSION OF THE LESSON

A. Serial Communication Protocols

Communication between electronic devices resembles human communication, as both parties must use a common language. In embedded systems, communication refers to the transfer of data in the form of bits between two microcontrollers. This data exchange within microcontrollers follows a specific set of predefined rules called communication protocols. The methods of transferring bits can be broadly classified into two categories: parallel communication and serial communication.

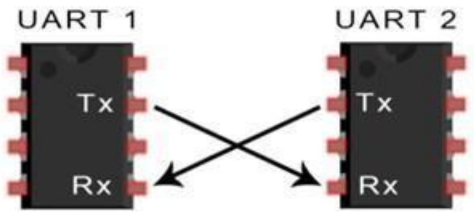
Parallel Communication	Serial Communication
data transmitted simultaneously on separate communication lines.	data transmitted serially one by one (on single communication line)
n wire or lines are used to transmit *n* bits	one line than *n* lines to transmit data
more costly	less costly
faster than serial communication	long distance transmission
data can be transmitted in less time	requires more time to transmit data

Serial communication is the most common method used to transmit data between data processing peripherals. It transmits data one bit at a time, utilizing a single conductor for all the bits. Serial communication is particularly suitable for scenarios requiring long-distance transmission, high throughput, and multiple nodes. The three main serial communication protocols used are UART (Universal Asynchronous Receiver Transmitter), SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit).

Serial Protocol	Synchronous/ Asynchronous	Type	Duplex	Data Transfer Rate (kbps)
UART	asynchronous	peer-to-peer	full-duplex	20
I2C	synchronous	multi-master	half-duplex	3400
SPI	synchronous	multi-master	full-duplex	>1,000

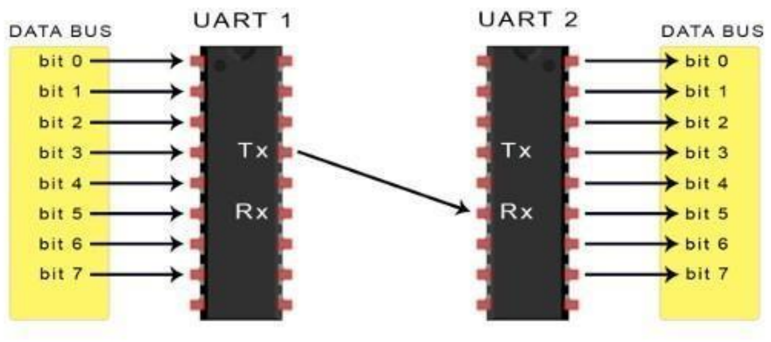
UART COMMUNICATION

UART communication involves direct communication between two UARTs. The transmitting UART takes parallel data from a controlling device, such as a CPU, and converts it into serial form. This serial data is then transmitted to the receiving UART, which converts it back into parallel data for the receiving device. The transmission of data between the two UARTs only requires two wires. The data flows from the Tx (transmit) pin of the transmitting UART to the Rx (receive) pin of the receiving UART.



UARTs transfer data asynchronously using start and stop bits to mark the beginning and end of data packets. The receiving UART reads the bits based on the baud rate, which determines the speed of data transfer. Matching baud rates between UARTs is essential, as a difference exceeding 10% can lead to timing issues and difficulty in interpreting received bits.

The UART responsible for transmitting data receives the data from a data bus, which acts as a conduit for transferring data from devices like CPUs, memory, or microcontrollers to the UART. The data is initially transferred in parallel form from the data bus to the transmitting UART. Once the transmitting UART receives the parallel data, it appends a start bit, a parity bit, and a stop bit, forming a data packet. Subsequently, the data packet is transmitted serially, bit by bit, through the Tx pin. On the receiving end, the receiving UART reads the data packet bit by bit via its Rx pin. It then converts the data back into parallel form, eliminating the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus at the receiving end.



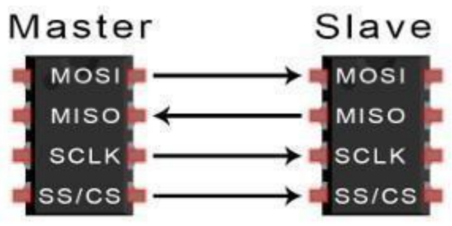
UART, also known as Universal Asynchronous Receiver Transmitter, is a hardware component specifically designed for serial communication. The UART hardware can either be integrated onto the microcontroller or implemented as a separate integrated circuit (IC). Unlike SPI or I2C, which are solely communication protocols, UART encompasses both the hardware and protocol aspects.

UART is considered one of the simplest and most widely utilized techniques for serial communication. It finds applications in various domains such as GPS receivers, Bluetooth modules, GSM and GPRS modems, wireless communication systems, and RFID-based applications, among others.

SPI COMMUNICATION

SPI is a widely used communication protocol among various devices. For instance, SD card modules, RFID card reader modules, and 2.4 GHz wireless transmitter/receivers all utilize SPI to establish communication with microcontrollers. One notable advantage of SPI is its ability to transfer data without interruption. It allows for a continuous stream of bits to be sent or received, without the limitation of fixed packet sizes found in I2C and UART. In the case of I2C and UART, data transmission occurs in packets with defined start and stop conditions, resulting in interruptions during the transmission process.

Devices communicating through SPI follow a master-slave relationship. The master, typically a microcontroller, controls communication and provides instructions, while the slave, which can be a sensor, display, or memory chip, receives and responds to the instructions from the master. Although a single master typically controls a single slave in the simplest configuration of SPI, it is possible for one master to control multiple slaves.



MOSI (Master Output/Slave Input) – line for the master to send data to the slave
MISO (Master Input/Slave Output) – line for the slave to send to the master
SCLK (Clock) – line for the clock signal
SS/CS (Slave Select/Chip Select) – line for the master to select which slave to send data

In SPI communication, the clock signal synchronizes the transfer of data bits between the master and the slave. Each clock cycle transfers one bit, and the speed of data transfer is determined by the clock frequency. The master initiates SPI communication and controls the clock signal. SPI is a synchronous communication protocol where devices share a clock signal, in contrast to asynchronous methods like UART, which do not rely on a clock signal and instead use pre-configured baud rates.

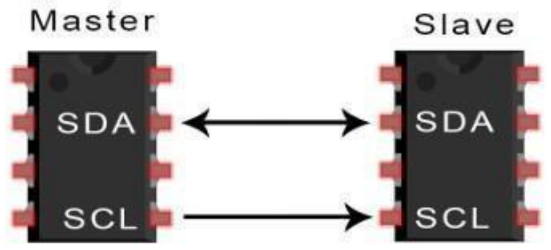
The clock signal in SPI can be modified using clock polarity and clock phase. Clock polarity allows the master to determine whether bits are output and sampled on the rising or falling edge of the clock cycle. The clock phase, on the other hand, enables the master to choose the timing of output and sampling within each clock cycle, regardless of whether it is rising or falling. By adjusting these properties, the master can customize the behavior of the clock signal in SPI communication to meet specific requirements.

The master can select a specific slave for communication by setting the slave's CS/SS (Chip Select/Slave Select) line to a low voltage level. When not actively transmitting, the slave selects line remains at a high voltage level. In scenarios where multiple slaves are involved, the master can have multiple CS/SS pins to connect the slaves in parallel. Alternatively, if the master has only one CS/SS pin available, multiple slaves can be connected in a daisy-chained configuration.

The master transmits data to the slave in a sequential manner, sending it bit by bit through the MOSI (Master Out Slave In) line. The slave, in turn, receives the data sent by the master at its MOSI pin. Typically, data transmitted from the master to the slave follows a convention where the most significant bit is sent first. Additionally, the slave has the capability to send data back to the master via the MISO (Master In Slave Out) line, again in a serial fashion. When the slave sends data back to the master, it generally adheres to the practice of sending the least significant bit first.

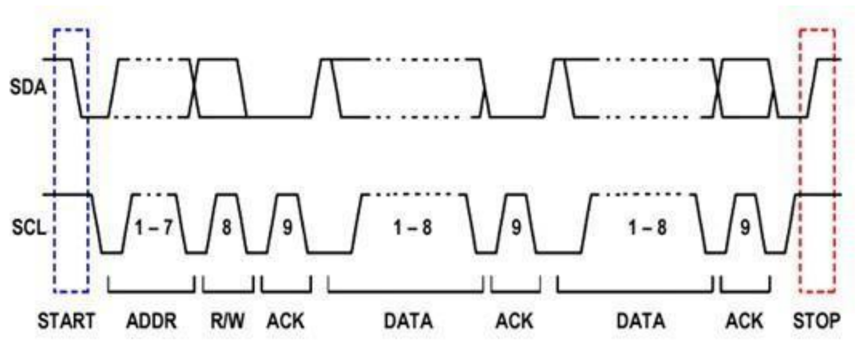
I2C COMMUNICATION

I2C combines the best features of both SPI and UART protocols. It allows for the connection of multiple slave devices to a single master, like SPI. Additionally, I2C offers the flexibility of having multiple masters controlling single or multiple slaves. This capability is particularly beneficial in situations where multiple microcontrollers log data to a shared memory card or display text on a single LCD. Like UART communication, I2C requires only two wires for data transmission between devices.



SDA (Serial Data) – the line for the master and slave to send and receive data
SCL (Serial Clock) – the lien that carries the clock signal

I2C operates in a synchronous manner, meaning that the transmission of bits is synchronized with the sampling of bits using a shared clock signal between the master and the slave. The master always assumes control over the clock signal, governing the timing of the data transfer.



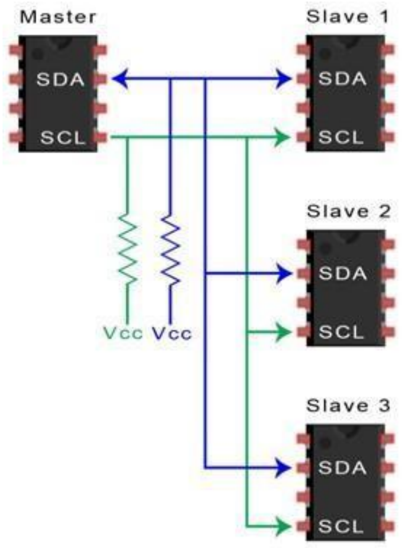
I2C operates through message-based data transfer, where messages are divided into frames. Each message includes an address frame with the slave's binary address and one or more data frames. Start and stop conditions, read/write bits, and ACK/NACK bits are also part of the message structure. Start and stop conditions indicate the beginning and end of communication. Address frames identify the slave device, read/write bits specify the direction of data transfer, and ACK/NACK bits confirm successful reception of frames.

. Unlike SPI, I2C does not utilize slave select lines to communicate with specific slaves. Instead, it relies on addressing to inform the intended slave that data is being sent to it. The address frame, always the first frame after the start bit in a new message, is used for this purpose. The master transmits the address of the desired slave to all connected slaves. Each slave then compares the received address with its own. If there is a match, the slave responds by sending a low voltage ACK bit back to the master. In case of no match, the slave does not respond, and the SDA line remains in a high voltage state.

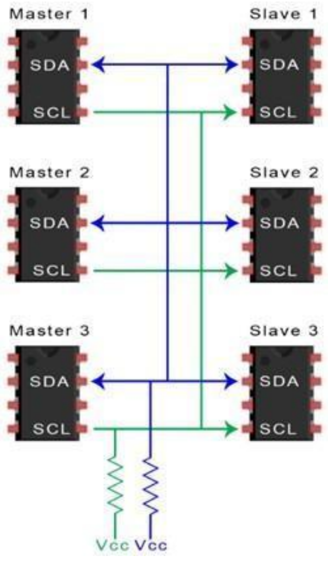
Within the address frame, there is a specific bit that serves as an indication to the slave regarding the intended data flow with the master. This read/write bit is positioned at the end of the address frame. When the master intends to transmit data to the slave, the read/write bit is set to a low voltage level. On the other hand, if the master wishes to receive data from the slave, the read/write bit is set to a high voltage level.

Once the master receives the ACK bit from the slave, it proceeds to transmit the first data frame. Each data frame consists of 8 bits and is sent in the order of most significant bit first. Following the transmission of each data frame, an ACK/NACK bit promptly follows to confirm the successful reception of the frame. The receipt of the ACK bit, either by the master or the slave (depending on the sender of the data), is necessary before proceeding to send the next data frame.

Once all the data frames have been transmitted, the master has the option to initiate a stop condition, signaling the end of the transmission. The stop condition is identified by a voltage transition from low to high on the SDA line, occurring after a low to high transition on the SCL line. Subsequently, the SCL line remains high.



I2C's addressing feature enables the control of multiple slaves by a single master. By employing a 7-bit address, there are 128 distinct addresses available for use. Although less common, the use of 10-bit addresses expands the address range to 1,024 unique addresses. To connect multiple slaves to a single master, a wiring configuration is employed. This involves the inclusion of 4.7K/10K Ohm pull-up resistors that connect the SDA and SCL lines to Vcc.



It is possible to connect multiple masters to either a single slave or multiple slaves. However, a challenge arises when multiple masters attempt to transmit or receive data simultaneously over the SDA line. To address this issue, each master must first detect the status of the SDA line, whether it is in a low or high state, before initiating a message transmission. If the SDA line is low, it signifies that another master currently controls the bus, and the current master should wait its turn to send the message. On the other hand, if the SDA line is high, it indicates that it is safe to proceed with transmitting the message.

To establish connections between multiple masters and multiple slaves, the following diagram can be employed. It involves utilizing 4.7K Ohm pull-up resistors that connect the SDA and SCL lines to Vcc, ensuring proper signal levels and communication stability.

B. Wireless Communication Protocols

What are Wireless Communication Protocols?

Wireless communication protocols are sets of rules and standards that govern the transmission of data between devices without the need for physical connections. These protocols define how devices communicate, exchange information, and establish reliable connections in wireless networks. They ensure compatibility and interoperability between different devices and enable seamless data transmission over various wireless technologies.

Understanding Wi-Fi, Bluetooth, and Zigbee

Wi-Fi, short for Wireless Fidelity, is a wireless communication protocol based on the IEEE 802.11 standards. It enables devices to connect to local area networks (LANs) and the internet wirelessly. Wi-Fi operates in the unlicensed radio frequency spectrum and provides high-speed data transmission over short to medium distances. It has found widespread applications in homes, offices, public spaces, and industries, facilitating internet access, file sharing, streaming media, and IoT connectivity.

Bluetooth on the other hand is a short-range wireless communication protocol designed for connecting devices over short distances. It operates in the 2.4 GHz frequency band and facilitates the exchange of data, audio, and control signals. Bluetooth technology is commonly used for connecting peripherals like speakers, headphones, keyboards, and game controllers to smartphones, computers, and other devices. It has also found applications in healthcare, automotive, and home automation systems.

Lastly, Zigbee is a low-power wireless communication protocol specifically designed for low-cost and low-data-rate applications. It operates in 2.4 GHz, 900 MHz, and 868 MHz frequency bands and focuses on providing efficient and reliable communication for devices in personal area networks (PANs). Zigbee is commonly used in home automation, smart energy management systems, industrial monitoring, and control applications.

Applications and Uses of Wi-Fi, Bluetooth, and Zigbee

Wi-Fi is widely used for internet access in homes, offices, coffee shops, and public spaces. It enables users to connect their smartphones, tablets, laptops, and smart TVs to the internet wirelessly. Additionally, Wi-Fi is crucial for IoT devices, allowing them to communicate and exchange data with each other and the cloud. It is also utilized in industrial settings for machine-to-machine communication, real-time monitoring, and automation.

Bluetooth technology has diverse applications, ranging from wireless audio streaming between devices like smartphones and speakers, to hands-free calling in cars. It is also utilized for wireless data transfer between devices, such as file sharing between smartphones or connecting wireless peripherals to computers. Bluetooth has gained prominence in the IoT domain, enabling smart home devices, wearable technology, and health monitoring systems to communicate and interact seamlessly.

Zigbee is commonly used in home automation systems, allowing devices such as smart lighting, thermostats, and security systems to connect and communicate with each other. It provides reliable and energy-efficient communication, making it suitable for applications where devices need to operate on battery power for extended periods. Zigbee's mesh networking capability enables devices to relay signals, extending the coverage area and enhancing network robustness.

Considerations for IoT Connectivity

When considering IoT connectivity, here are some factors to keep in mind:

Security: As IoT devices are interconnected, ensuring robust security measures is crucial. Encryption, authentication, and access control mechanisms must be implemented to protect sensitive data and prevent unauthorized access.

Interoperability: IoT devices often rely on different wireless communication protocols. Ensuring interoperability between devices and protocols is important to enable seamless communication and integration within the IoT ecosystem.

Scalability: IoT networks can grow rapidly as more devices are added. Scalability should be considered to accommodate the increasing number of devices and the associated data traffic.

Power Consumption: Many IoT devices are battery-powered or have limited power resources. Choosing energy-efficient wireless communication protocols and optimizing power consumption can extend device battery life and reduce maintenance efforts.

Network Coverage: Depending on the application, the range and coverage area of the wireless communication protocol must be evaluated to ensure reliable connectivity throughout the intended deployment area.

By considering these factors, users can make informed decisions regarding wireless communication protocols and their suitability for specific IoT applications.

C. Sensor Data Acquisition and Processing

Understanding Sensor Data Acquisition

The process of gathering information from multiple sensors that track and quantify physical occurrences is known as sensor data acquisition. A variety of data may be recorded via sensors, including temperature, pressure, motion, light, and more. To acquire sensor data, analog signals produced by the sensors are captured and converted to digital data for additional processing and analysis.

Processing Sensor Data

Once the sensor data has been collected, processing is frequently necessary to draw out important conclusions or carry out certain tasks. There are various phases involved in processing sensor data, such as data filtering, calibration, normalization, and analysis. Data filtering helps to enhance the quality of the obtained data by removing noise or undesirable signals. The sensor measurements are checked for alignment with established standards or reference values during calibration. The data may be scaled within a certain range by using normalization. Statistical analysis, machine learning algorithms, and signal processing techniques are a few examples of data analysis approaches that may be used to extract useful information and make data-driven choices.

Analog to Digital Conversion

One of the most important steps in gathering sensor data is analog to digital conversion (ADC). Typically, analog signals from sensors fluctuate continually depending on the actual quantity being measured. These signals must be transformed into a distinct digital representation in order to be processed digitally. The two primary steps of ADC are sampling and quantization. Sampling entails taking periodic snapshots of the analog signal to produce discrete samples. Each sample is given a digital value during quantization that represents its magnitude within a specified range. The sample rate, resolution, and signal-to-noise ratio are just a few examples of the variables that affect how accurate the ADC conversion is.

Techniques for Analog to Digital Conversion

Depending on the purpose and needs, many analog to digital conversion techniques are utilized. The successive approximation ADC is a popular technique that compares the analog signal to a digital approximation and iteratively changes the value until a good match is obtained. Another method is the sigma-delta modulator-based delta-sigma ADC, which oversamples the analog signal and turns it into a digital stream. In terms of speed, accuracy, and cost, other techniques include flash ADC, pipeline ADC, and dual-slope ADC. Each has its own benefits and trade-offs.

Importance of Analog to Digital Conversion

For several reasons, analog to digital conversion is essential. Compared to analog signals, digital data is simpler to process, store, and transfer. It makes it possible to apply digital signal processing methods including compression, filtering, and analysis algorithms. In addition, the simplicity with which digital data can be integrated into computer systems, microcontrollers, and other digital devices makes it possible to share data and automate data display and automation. The dependability and value of the obtained sensor data directly depend on the precision and quality of the analog to digital conversion, making it a crucial stage in the data gathering process.

Researchers and engineers can efficiently acquire and use sensor data for a variety of applications, such as environmental monitoring, industrial control systems, healthcare, and Internet of Things (IoT) deployments by knowing sensor data acquisition, processing methodologies, and analog to digital conversion.

ACTIVITIES / EXAMPLES

Arduino ADC Example

Let’s use the Arduino to detect an analog voltage. Use a trimpot, or light sensor, or simple voltage divider to create a voltage. Let’s setup a simple trimpot circuit for this example:

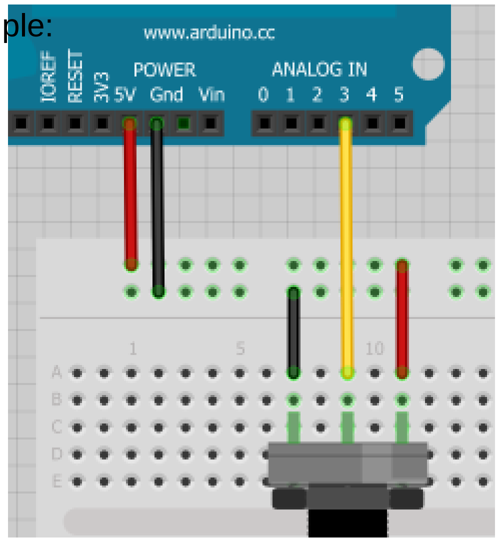


Figure 1.An example of a trimpot circuit. Image from https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all

To start, we need to define the pin as an input. To match the circuit diagram we will use

```
COPY CODEpinMode(A3, INPUT);
```

and then do the analog to digital version by using the analogRead()

```
COPY CODEint x = analogRead(A3); //Reads the analog value on pin A3 into x
```

The value that is returned and stored in x will be a value from 0 to 1023. The Arduino has a 10-bit ADC (2^10 = 1024). We store this value into an int because x is bigger (10 bits) than what a byte can hold (8 bits).

```
COPY CODESerial.print(“Analog value: “);
Serial.println(x);
```

X ought to vary when we alter the analog value. What voltage is actually present, for instance, if x is stated to be 334 and the Arduino is operating at 5V? Check the real voltage with your digital multimeter. It ought to be around 1.63V. Congratulations! You just used an Arduino to make your own digital multimeter!

SUMMARY

The module discussed various aspects of interfacing and communication in embedded systems, including serial communication protocols, wireless communication protocols, and sensor data acquisition. The knowledge gained can be applied in real-life scenarios such as developing IoT devices, designing communication interfaces between microcontrollers and peripherals, and implementing data acquisition systems for sensors in various applications like industrial automation, healthcare devices, and smart homes.

We also explored wireless communication protocols and their significance in the modern world. We discussed popular protocols such as Wi-Fi, Bluetooth, and Zigbee, along with their applications and uses. Additionally, we highlighted important considerations for IoT connectivity, including security, interoperability, scalability, power consumption, and network coverage. Understanding these protocols and factors can help users make informed decisions and leverage wireless communication effectively in various contexts.

Additionally, we explored the process of sensor data acquisition, highlighting the importance of converting analog signals from sensors into digital format for further analysis. It emphasized the steps involved in processing sensor data, including filtering, calibration, normalization, and analysis techniques. The discussion also delved into the significance of analog to digital conversion, explaining how it enables easier processing, storage, and integration of sensor data into digital systems.

Various techniques for analog to digital conversion were discussed, such as successive approximation ADC and delta-sigma ADC, each offering advantages in terms of speed, accuracy, and cost. Overall, understanding sensor data acquisition and the conversion of analog to digital signals is vital for harnessing valuable insights from sensor-based applications.

SUMMATIVE ASSESSMENT (SA)

1. What is the purpose of the slave select (SS) line in SPI communication?
    a. It selects the master device.
    b. It selects the clock frequency.
    c. It selects the slave device for communication.
    d. It selects the data transmission mode.

2. In I2C communication, how does the master inform the slave about the data transfer direction?
    a. By sending a specific command byte.
    b. By using a separate ACK/NACK line.
    c. By setting the SCL line to a high voltage level.
    d. By setting the read/write bit in the address frame.

3. What is the purpose of sensor data acquisition?
    a. To convert analog signals to digital signals
    b. To process digital signals into useful information
    c. To amplify sensor signals for better accuracy
    d. To calibrate sensors for optimal performance

4. Which of the following is responsible for converting analog signals to digital signals in a data acquisition system?
    a. Analog-to-digital converter (ADC)
    b. Digital-to-analog converter (DAC)
    c. Operational amplifier (Op-Amp)
    d. Voltage regulator

5. What is the purpose of analog-to-digital signal conversion?
    a. To eliminate noise from the analog signal
    b. To improve the accuracy of sensor measurements
    c. To reduce the bandwidth requirements for data transmission
    d. To increase the voltage level of the analog signal

6. What are Wireless Communication Protocols?

7. What are the three Common Wireless Communication Protocols?

____
____
____
____
____

8. Discuss the considerations you must take into account with regards to IoT connectivity.

____
____
____
____
____

LAB ACTIVITY

Voltage-to-Frequency Converter as ADC

Create an analog-to-Digital Converter using the AD654 Voltage-to-Frequency converter and create an output frequency equal to 100 kHz.

REFERENCES / BIBLIOGRAPHY

Analog, W. (2021). *Analog devices*. Activity: Analog to Digital Conversion [Analog Devices Wiki]. https://wiki.analog.com/university/courses/electronics/electronics-lab-adc

Anixter, A. (2022). *Comparing wireless communication protocols*. Comparing Wireless Communication Protocols. https://www.anixter.com/en_us/resources/literature/techbriefs/comparing-wireless-communication-protocols.html

Campbell, S. (2021). Retrieved from https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/

Cantennas, C. (2023, March 30). *Characteristics of the 6 wireless protocols*. C&T RF Antennas Manufacturer. https://ctrfantennasinc.com/characteristics-of-the-6-wireless-protocols/

Casini, M. (2022). Building Automation Systems. *Construction 4.0*, 525–581. https://doi.org/10.1016/b978-0-12-821797-9.00008-8

Circuits, D. (2020, June 25). *Digital-to-analog conversion worksheet - digital circuits*. All About Circuits. https://www.allaboutcircuits.com/worksheets/digital-to-analog-conversion/

Dhaker, P. (2019). Retrieved from https://embeddedcomputing.com/technology/software-and-os/ides-application-programming/introduction-to-spi-interface

Nate, N. (2021). *Analog to digital conversion*. Analog to Digital Conversion - SparkFun Learn. https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all

Notes, E. (2020). *What is Data Acquisition*. Electronics Notes. https://www.electronics-notes.com/articles/test-methods/data-acquisition-daq/understanding-data-acquisition.php

Pandit, A. (2019). Serial Communication Protocols. Retrieved from https://circuitdigest.com/tutorial/serial-communication-protocols

Slamaris, D. (2022). Retrieved from https://embeddedsecurity.io/protocols

Understanding Serial Protocols. (n.d.). Retrieved from https://www.rohde-schwarz.com/us/products/test-and-measurement/essentials-test-equipment/digital-oscilloscopes/understanding-serial-protocols_254522.html#:~:text=Serial%20protocols%20are%20used%20to,components%20or%20between%20separate%20devices.&text=Serial%20protocols%20can%20be%20divided,FlexRay%20(primarily%20used%20in%20automotive)

WatElectronics. (2022). Retrieved from https://www.watelectronics.com/i2c-protocol/

White, K. (2022). Retrieved from https://www.weare5vmedia.com/media/communication-protocols-for-an-embedded-engineer-to-know#:~:text=The%20four%20communication%20protocols%20most,%2C%20SPI%2C%20I2C%20and%20USB.

Power Management and Optimization in Embedded Systems

INTRODUCTION

This module focuses on power management and optimization in embedded systems. It covers various aspects such as power requirements, power management strategies, and performance optimization techniques.

LEARNING OBJECTIVES / OUTCOMES

At the end of the module, students should be able to:

- Understand the power requirements and constraints of embedded systems, including battery-powered systems and energy harvesting techniques.
- Explore different power management strategies, including sleep modes and power-down techniques, to optimize power consumption in embedded systems.
- Learn performance optimization techniques, such as code optimization, memory optimization, and data compression, to enhance the efficiency of embedded systems.

DISCUSSION OF THE LESSON

Power Requirements and Constrains

In designing and developing embedded systems, requirements and constraints in power play a crucial role. Their role is important in wide range such as energy efficiency, size and form factors, heat and thermal dissipation, reliability, longevity, cost consideration, environmental impact, and system performance. It is crucial to take power requirements and constraints into account when designing embedded systems to maximize the use of energy, adhere to size limitations, control heat dissipation, improve dependability, save operational costs, minimize environmental effects, and maintain satisfactory system performance. Successful embedded system design and implementation depends on considering these factors.

Battery-powered Systems

Devices or systems that function primarily on batteries are referred to as battery-powered embedded systems. These systems can be employed in a variety of settings and applications since they are made to be mobile, portable, or independent of a constant power source. However, if we discussed battery-powered systems, huge machines are included such as electric vehicles (bikes, cars, and etc.), emergency systems, and so forth.

Other examples of battery-powered systems are listed below.

1. **Portable Electronics**
   Smartphones, laptops, tablets, digital cameras, portable speakers, and handheld game consoles.

2. **Wearable devices**
   Fitness trackers, smart glasses, wearable health monitors, head-mounted display, or VR headsets, and etc.

3. **Portable medical devices**
   Blood glucose monitors, insulin pumps, and portable diagnostic equipment.

4. **Remote Monitoring Systems**
   Environmental monitoring systems, wildlife tracking devices, weather stations, and so forth.

5. **IoT devices**
   Water tank monitoring devices, automated pet feeder, IoT based plant monitoring systems, and such systems that relieve IoT implementation.

6. **Remote Controlled Systems**
   Remote controlled toys (cars, helicopter, etc.), drones, and home automation systems.

7. **Electric vehicles**
   E-jeep, E-bike, Tesla cars, and a lot more.

   Designers may reduce power consumption, increase battery life, and ensure efficient and dependable operation in battery-powered systems by thoroughly assessing power requirements and constraints. As a result, the user experience is enhanced, maintenance costs are decreased, and system performance is increased. In addition to this, the longevity of the device is being supported as well once the adequate power requirements and constraints are implemented.

Energy Harvesting Techniques

1. *Solar Energy Harvesting*

One of the most well-liked and frequently utilized energy sources is solar energy. Solar panels or photovoltaic (PV) cells are used to turn sunlight into electrical energy. Solar energy harvesting is frequently utilized in applications including solar-powered sensors, outdoor monitoring systems, and autonomous outdoor equipment. It is ideal for outdoor or well-lit areas.

2. *Mechanical Energy*

Vibrations, motion, or mechanical tension can all be converted into mechanical energy. Mechanical energy can be transformed into electrical energy using processes like piezoelectric materials, electromagnetic induction, or electrostatic means. Wearable technology, commercial monitoring systems, and self-powered wireless sensors all frequently use mechanical energy harvesting.

3. *Thermal Energy*

Utilizing waste heat or temperature gradients to generate electrical energy is known as thermal energy harvesting. Temperature differences can be harnessed and turned into useable electricity using methods such as thermoelectric generators (TEGs) or pyroelectric materials. In applications like industrial processes, HVAC systems, or energy-efficient buildings where there are large temperature changes, thermal energy harvesting is appropriate.

4. *Radio Frequency Energy*

RF energy harvesting is the process of obtaining electromagnetic energy from wireless signals such as those sent by cellular networks, Wi-Fi, and Bluetooth. Rectennas (rectifying antennas) and energy-harvesting RF circuits are two examples of devices used to convert RF energy into electrical energy. Wireless sensor networks, Internet of Things (IoT) devices, and communication systems frequently use RF energy harvesting.

5. *Kinetic Energy*

Through capturing the energy of motion or movement, kinetic energy harvesting. Mechanical motion can be converted into electrical energy using methods such as electromagnetic induction, piezoelectric materials, or magnetic materials. Applications where there is constant movement or vibration, such as wearable technology, wireless switches, or vehicle-based systems, are well suited for kinetic energy harvesting.

6. *Ambient Light Energy*

The process of harvesting ambient light energy involves transforming light energy—such as that from artificial lighting or low-intensity natural light—into electrical energy. To absorb and transform light energy, photovoltaic cells or dye-sensitized solar cells can be utilized. Smart sensors, wireless communication systems, and low-power interior devices all frequently use ambient light energy harvesting.

It's vital to remember that energy harvesting systems have restrictions on power availability and generation. Depending on the environment, the accessibility of energy sources, and the effectiveness of the energy harvesting device, different amounts of energy may be harvested. The embedded system's energy needs must be carefully considered by designers, who must then choose the best energy harvesting technique(s) to fulfill the specified power requirements and operational restrictions. In addition, effective energy storage and management technologies, like supercapacitors or rechargeable batteries, are frequently used to manage and store the gathered energy for a steady supply of power to the embedded system.

Power Management Strategies

A crucial problem is the power consumption of embedded electronics. There is always a need to increase battery life and/or lessen a system's influence on the environment. In the past, this was only a hardware problem, but those times were gone. The control of power in contemporary embedded systems is increasingly the responsibility of software. This article examines the methods used to reduce power consumption when a device is not in use as well as how power management is accomplished while a device is in use.

A device's power consumption may often be divided into two categories: when it is in use and when it is not. The deployment of low power CPU modes may be advantageous in the latter case, while active power management is the primary requirement in the former.

Software Power Management

There are a few actions software can do when in use to minimize

Turn off peripheral devices while not in use.

Change the CPU's frequency and voltage in accordance with the performance demands of the moment (also known as "Dynamic)

Peripheral Power-down

It should go without saying that turning off any electrical or electronic equipment is the greatest method to conserve energy. Therefore, it makes sense to design electronic systems in a way that allows the software to turn peripherals and subsystems on and off as needed.

This capability is more complicated than it first appears because some peripherals, including network interfaces, require some setup time when they are turned on. Depending on how frequently the peripheral is turned on.

Dynamic Voltage and Frequency Scaling

Power consumption decreases with decreasing operation frequency. You can think of this in terms of how much "work" a specific piece of software

The CPU would be able to complete 10 times as much work if it ran at a clock frequency that allowed it to process a million instructions per second. Therefore, reducing the clock's frequency by this much [to enable 100K instructions per second] minimizes power usage and matches performance to needs.

Low Power Modes

A device may be completely turned off when not in use. Although some devices might need some status information saved on power down, this only needs minimal software support. The only issue is that it could take some

Standby – Power is maintained to RAM while the CPU and peripherals are turned off. The benefit of this mode is that devices wake up quickly, but the drawback is that power is still being used, which reduces the amount of time a device may remain in standby.

Hibernate – The system is shut down after data has been written out to (disk) storage. The benefit of this mode is that there is no ongoing

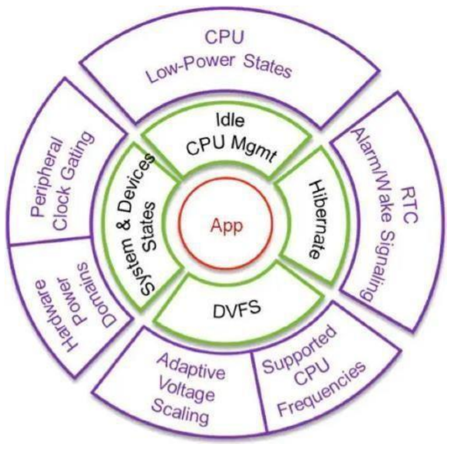
Similar sleep modes are becoming more common on embedded

Power Management Implementation

It is never really a good idea to "reinvent the wheel" when it comes to software development (or hardware, for that matter). It is pointless to write the code yourself if someone else has already implemented an algorithm or has an effective solution to a certain issue. Reusing already created

Real Time Operating Systems

Application code may be used to accomplish this functionality; however, this is inefficient and illogical. Given that the operating system can easily handle this and that power saving techniques may have a significant impact on drivers, it makes far more sense for the OS to include



Use Cases and Operating Points

Since an RTOS is not aware of the performance needs of the application code and since energy consumption is mostly influenced by the execution performance of the code, an RTOS cannot simply power optimize certain application code.

A typical strategy is for the developer to assess the program and define

Making a measurement

Displaying data Uploading data

Each use case necessitates a particular performance level and a set of accessible resources (peripherals, etc.). "Operating points" are voltage/frequency combinations that can be used to define performance. Additionally, an operating point is allocated to each use case.

The use case data is entered into the RTOS power management framework, which can then decide how to regulate power at any given time.

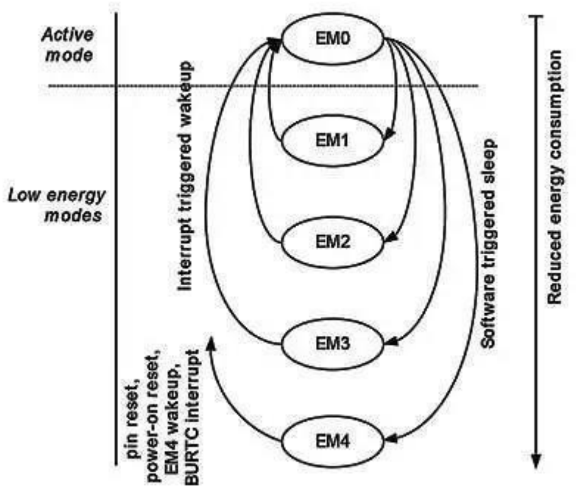
User Expectations

The power consumption of a gadget needs to be optimized for a

Sleep Modes and Power-down Techniques

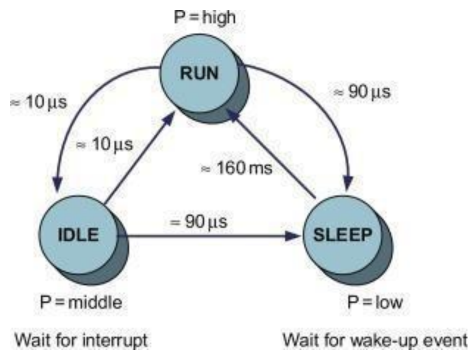
Microcontrollers (MCUs) are at the core of practically any real-time application that requires rapid, predictable response to real-world events, sleep mode algorithms in many of these applications to suspend most or all of its functions and reduce energy consumption, enabling it to operate for years or even decades on limited energy sources.

The same sleep states that lower an MCU's energy consumption frequently also lower its capacity for speedy response to an event, making these real-time low-energy environments very difficult for designers and programmers to create.


Active mode
Low energy modes
EM0
EM1
EM2
EM3
EM4
Interrupt triggered wakeup
pin reset, power-on reset, EM4 wakeup, BURTC interrupt
Software triggered sleep
Reduced energy consumption

Dynamic Power Management

The various operating CPU modes, such as sleep (idle) and active (running), are used in dynamic power management (DPM) approaches to potentially minimize power usage. These methods must account for the time and energy required to transition between various modes. Each state loses a specific amount of power. The changes between states demonstrate the delays brought on by switching between power states. The support of DVFS, DVS, or DFS may also influence the level of the RUN state.



Performance Optimization Techniques

Embedded systems require effective power management and optimization to ensure efficient resource utilization and extended battery life. Power-efficient embedded systems are made possible through performance optimization techniques such code optimization, memory optimization, and data compression.

Code Optimization

Code optimization methods increase the software's effectiveness in embedded systems, which boosts performance and lowers power consumption. Algorithmic optimization, loop optimization, function inlining, compiler optimization, and register allocation are important strategies. These methods put an emphasis on avoiding pointless computations, enhancing memory access patterns, and reducing resource usage.

Memory Optimization and Data Compression

By reducing the memory footprint of embedded systems, memory optimization and data compression approaches hope to reduce power consumption and boost performance. Data compression, memory partitioning, caching methods, dynamic memory allocation management, and data structure optimization are important strategies.

Data compression techniques, such as Huffman coding, run-length encoding, and Lempel-Ziv-Welch (LZW) compression, reduce the size of data stored in memory or transmitted between components, indirectly contributing to power savings by minimizing memory access and bandwidth usage. By separating memory into sections or banks, memory partitioning enables optimal exploitation of memory resources. To reduce power consumption, critical information can be stored in fast-access memory areas while less-used information can be kept in slower, more power-efficient areas. Caching methods, including data and instruction caching, cache line prefetching, and cache hierarchies, decrease power consumption and improve memory access latency. These methods reduce the amount of time and energy required for memory access by utilizing temporal and geographical proximity. Utilizing memory pools as an example, dynamic memory allocation management decreases overhead and fragmentation while increasing memory utilization. The power efficiency of dynamic memory allocation strategies is maximized by minimizing pointless memory allocations and deallocations. To eliminate padding, data structures must be aligned, the right data types must be chosen, and bit fields must be used. Through these improvements, power consumption is decreased while memory space is preserved, cache usage is increased, and overall performance is improved.

Power management and optimization are crucial for achieving performance and power efficiency in embedded systems. Data compression, code optimization, memory optimization, and other performance optimization methods can be used by programmers to build embedded systems that use less power and have longer battery lives.

ACTIVITIES / EXAMPLES

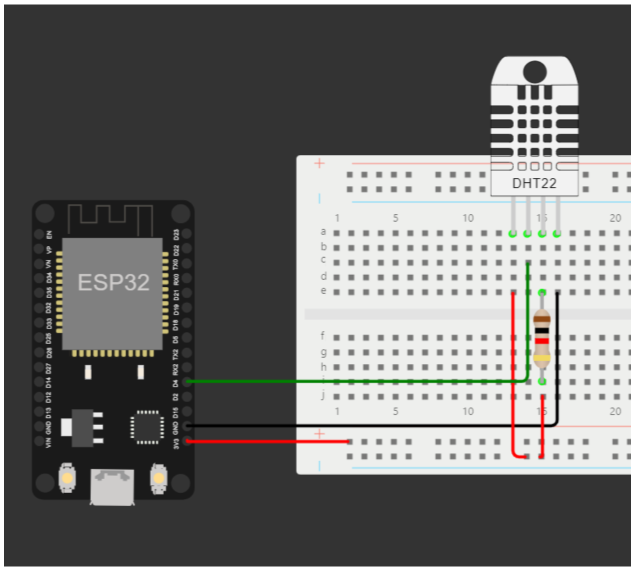
Activity 1: Low Power Weather Station

Let's build a low power dht11 weather station that will send data on a simple esp32 web server. In this activity, we will utilize the esp_sleep_enable_timer_wakeup() function to put the esp32 to sleep in order to save energy.

Material Requirements:

- ESP32
- DHT sensor
- Wires
- Breadboard

Hardware setup:



Sketch:

```
#include <WiFi.h>

#include <ESPAsyncWebServer.h>

#include <DHT.h>


const char* ssid = "your_network_ssid";

const char* password = "your_network_password";


const char* serverIP = "192.168.1.100"; // IP address of the ESP32
const int serverPort = 80;

const int dhtPin = 4; // Pin connected to the DHT11 sensor
const int dhtType = DHT11;

DHT dht(dhtPin, dhtType);

AsyncWebServer server(80);

void setup() {
  Serial.begin(115200);
  dht.begin();

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    String html = "<html><body>";
    html += "<h1>Weather Station</h1>";
    html += "<p>Temperature: " + String(getTemperature()) + " &deg;C</p>";
    html += "<p>Humidity: " + String(getHumidity()) + " %</p>";
    html += "</body></html>";
    request->send(200, "text/html", html);
  });
  server.begin();
  Serial.println("HTTP server started");

  // Enable deep sleep and set sleep time
  esp_sleep_enable_timer_wakeup(180 * 1000000); // Sleep for 3 minutes
  esp_deep_sleep_start();
}

void loop() {
  // This code will not be executed
}

float getTemperature() {
  return dht.readTemperature();
}

float getHumidity() {
  return dht.readHumidity();
}
```

Activity 2: Interfacing Solar Panels and Battery with ESP32

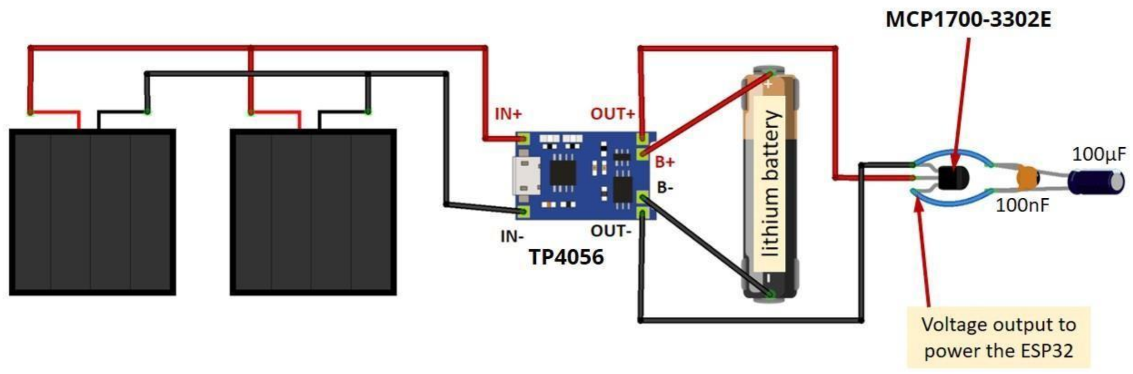
In this activity, we will create a circuit that will power an esp32 with a solar panel and battery.

Material Requirements:

- ESP32
- 2x Mini Solar Panel (5/6V 1.2W)
- Lithium Li-ion battery 18650
- Battery holder
- Battery charger (optional)

- TP4056 Lithium Battery Charger Module
- Voltage regulator:
  - Low-dropout or LDO regulator (MCP1700-3302E)
  - 100uF electrolytic capacitor
  - 100nF ceramic capacitor
- Optional – voltage divider for battery monitor:
  - 27k Ohm resistor
  - 100k Ohm resistor

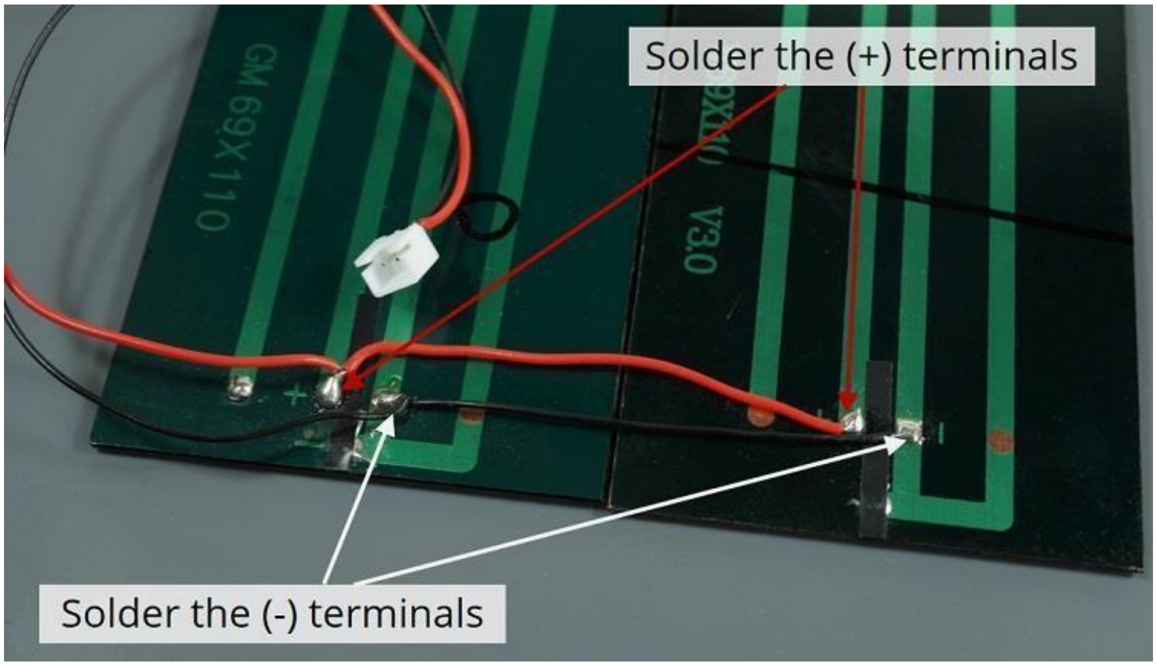
The following diagram shows how the circuit to power the ESP32 with solar panels works.



1. The solar panels output between 5V to 6V with direct sun.
2. The solar panels charge the lithium battery through the TP4056 battery charger module. This module is responsible for charging the battery and preventing overcharging.
3. The lithium battery outputs 4.2V when fully charged.
4. You need to use a low dropout voltage regulator circuit (MCP1700-3302E) to get 3.3V from the battery output.
5. The output from the voltage regulator will power the ESP32 through the 3.3V pin.

The solar panels we're using have an output voltage up to between 5V to 6V. If you want your battery to charge faster, you can use several solar panels in parallel. In this example we're using two mini solar panels as shown in the following figure.

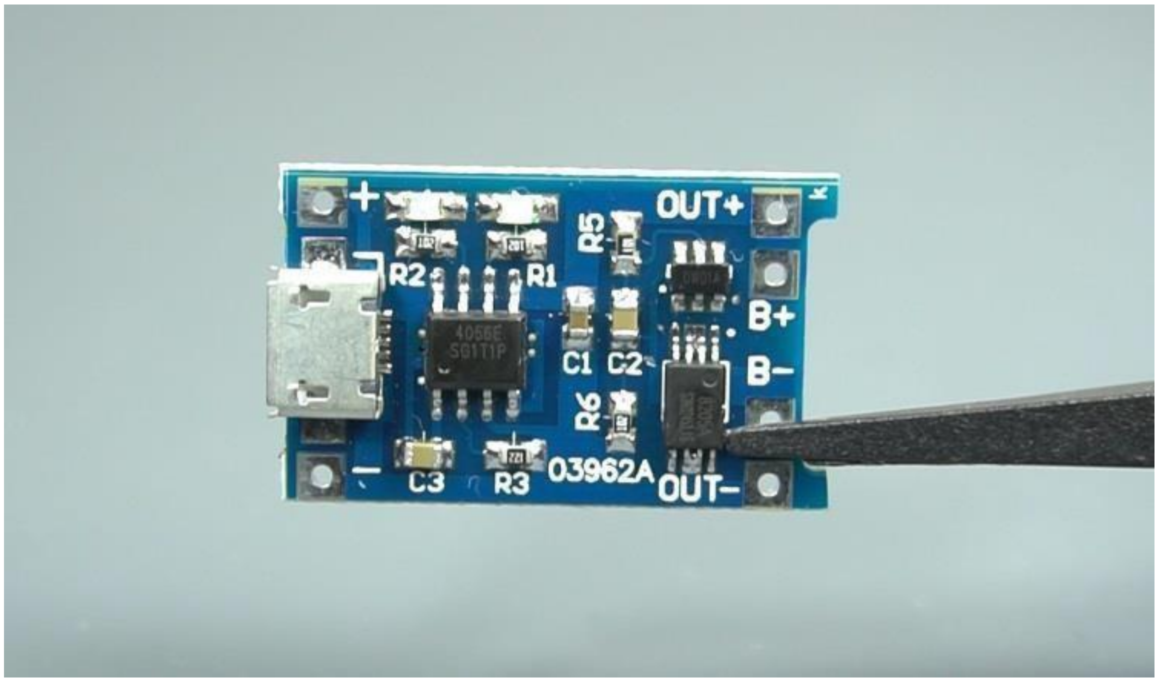
To wire solar panels in parallel solder the (+) terminal of one solar panel to the (+) terminal of the other solar panel. Do the same for the (-) terminals. It may help taking a look at the following figure.



When wiring solar panels in parallel you'll get the same output voltage and double the current (for identical solar panels). As you can see in the following figure, the solar panels output approximately 6V.

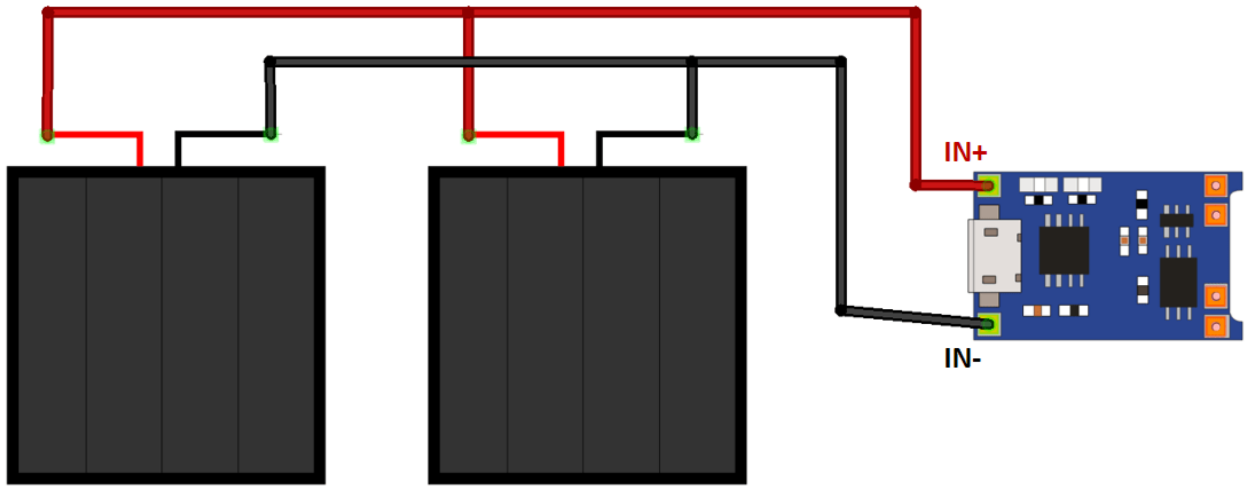
TP4056 Charger Module

The TP4056 lithium battery charger module comes with circuit protection and prevents battery over-voltage and reverse polarity connection.

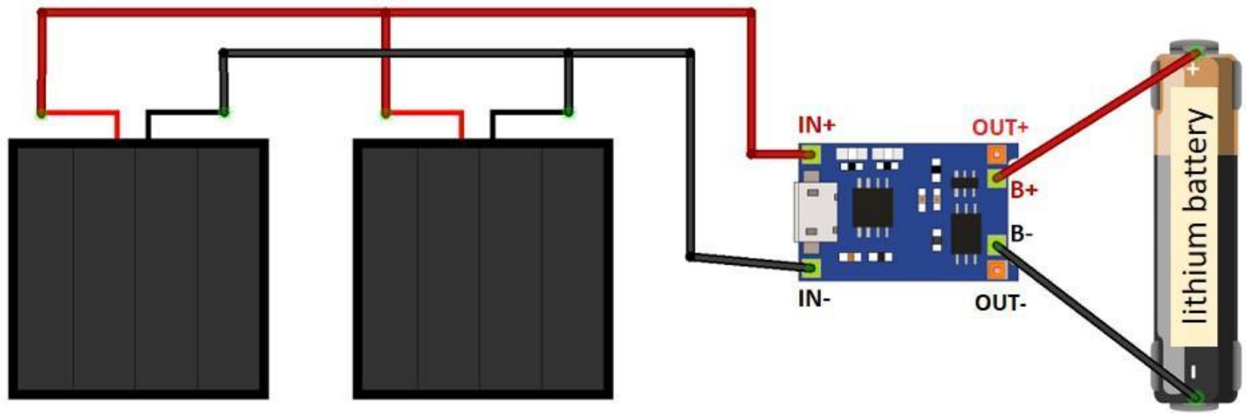


The TP4056 module lights up a red LED when it’s charging the battery and lights up a blue LED when the battery is fully charged.

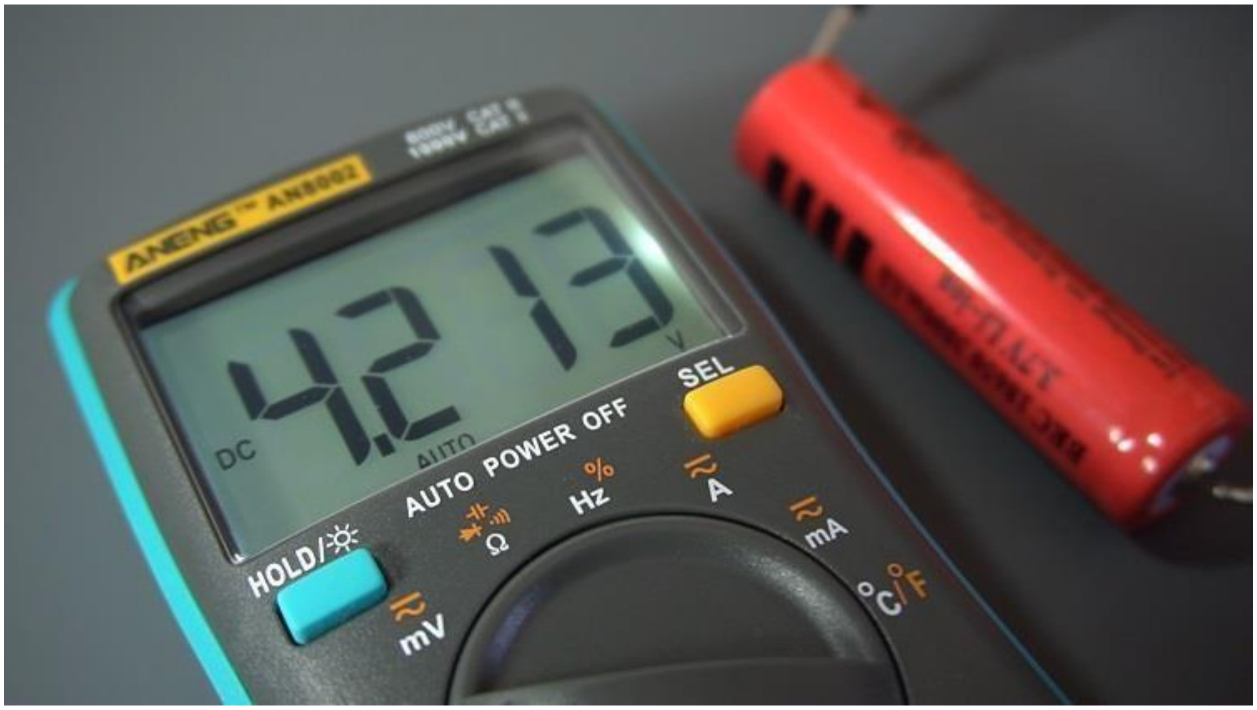
Wire the solar panels to the TP4056 lithium battery charger module as shown in the schematic diagram below. Connect the positive terminals to the pad marked with IN+ and the negative terminals to the pad marked with IN-.



Then, connect the battery holder positive terminal to the B+ pad, and the battery holder negative terminal to the B- pad.



OUT+ and OUT- are the battery outputs. These lithium batteries output up to 4.2V when fully charged (although they have 3.7V marked in the label).

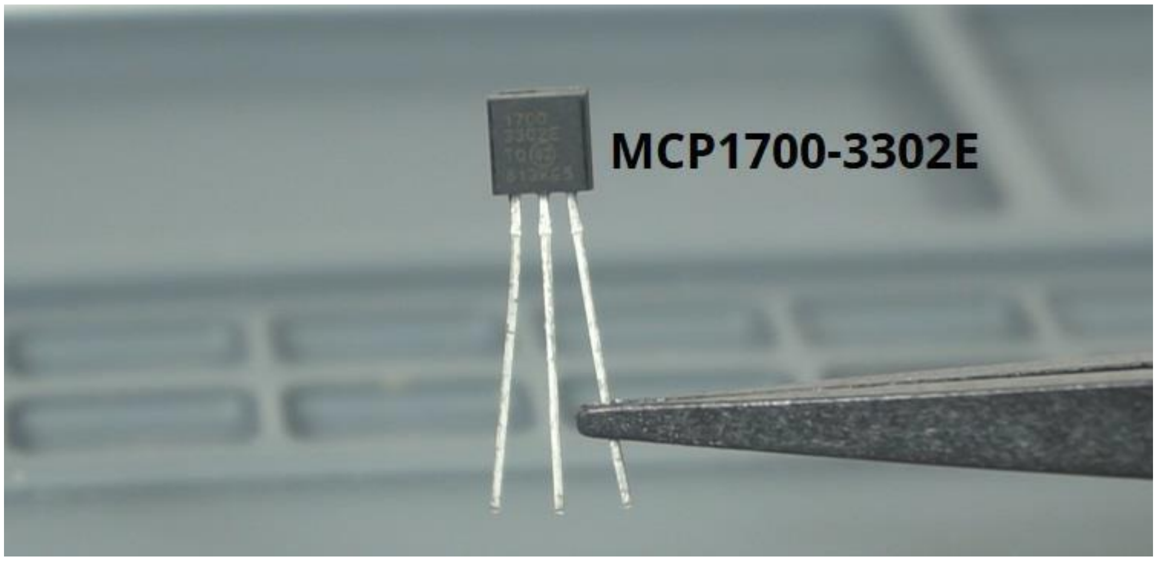


To power the ESP32 through its 3.3V pin, we need a voltage regulator circuit to get 3.3V from the battery output.

Voltage Regulator

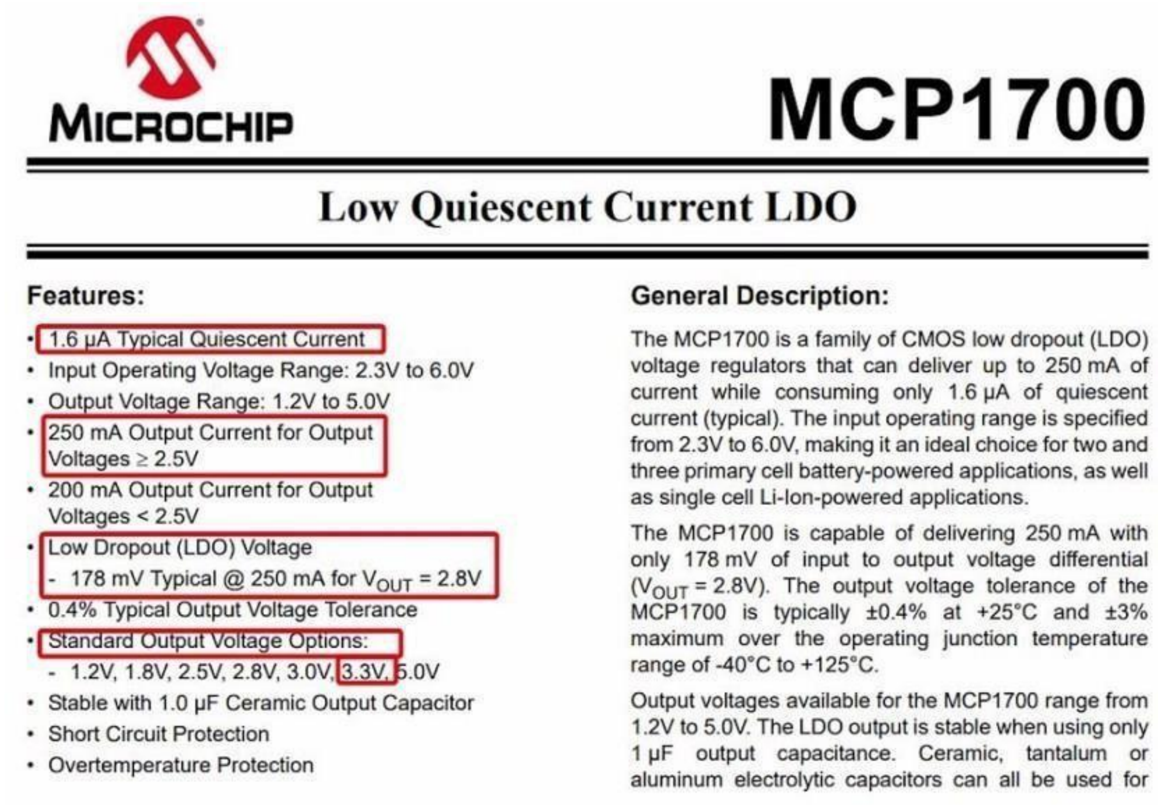
Using a typical linear voltage regulator to drop the voltage from 4.2V to 3.3V isn't a good idea, because as the battery discharges to, for example 3.7V, your voltage regulator would stop working, because it has a high cutoff voltage.

To drop the voltage efficiently with batteries, you need to use a low-dropout regulator, or LDO for short, that can regulate the output voltage.



After researching LDOs, the MCP1700-3302E is the best for what we want to do. There is also a good alternative like the HT7333-A.

Any LDO that has similar specifications to these two are also good alternatives. Your LDO should have similar specs when it comes to output voltage, quiescent current, output current and low dropout voltage. Look at the data sheet below.



MICROCHIP

MCP1700

Low Quiescent Current LDO

Features:

- 1.6 µA Typical Quiescent Current
- Input Operating Voltage Range: 2.3V to 6.0V
- Output Voltage Range: 1.2V to 5.0V
- 250 mA Output Current for Output Voltages ≥ 2.5V
- 200 mA Output Current for Output Voltages < 2.5V
- Low Dropout (LDO) Voltage
  - 178 mV Typical @ 250 mA for V_{OUT} = 2.8V
- 0.4% Typical Output Voltage Tolerance
- Standard Output Voltage Options:
  - 1.2V, 1.8V, 2.5V, 2.8V, 3.0V, 3.3V, 5.0V
- Stable with 1.0 µF Ceramic Output Capacitor
- Short Circuit Protection
- Overtemperature Protection

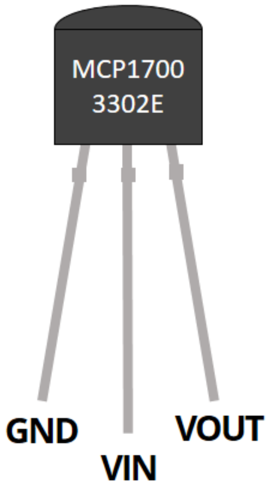
General Description:

The MCP1700 is a family of CMOS low dropout (LDO) voltage regulators that can deliver up to 250 mA of current while consuming only 1.6 µA of quiescent current (typical). The input operating range is specified from 2.3V to 6.0V, making it an ideal choice for two and three primary cell battery-powered applications, as well as single cell Li-Ion-powered applications.

The MCP1700 is capable of delivering 250 mA with only 178 mV of input to output voltage differential (V_{OUT} = 2.8V). The output voltage tolerance of the MCP1700 is typically ±0.4% at +25°C and ±3% maximum over the operating junction temperature range of -40°C to +125°C.

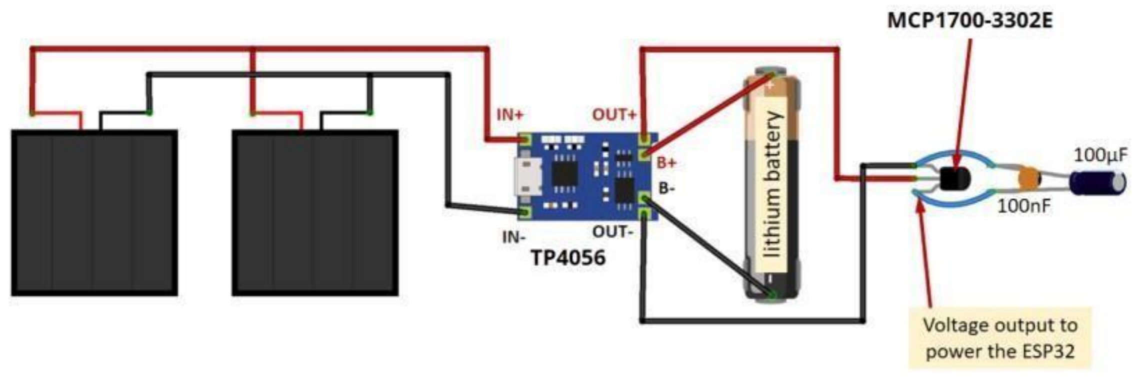
Output voltages available for the MCP1700 range from 1.2V to 5.0V. The LDO output is stable when using only 1 µF output capacitance. Ceramic, tantalum or aluminum electrolytic capacitors can all be used for

Here’s the MCP1700-3302E pinout: GND, VIN and VOUT pins.



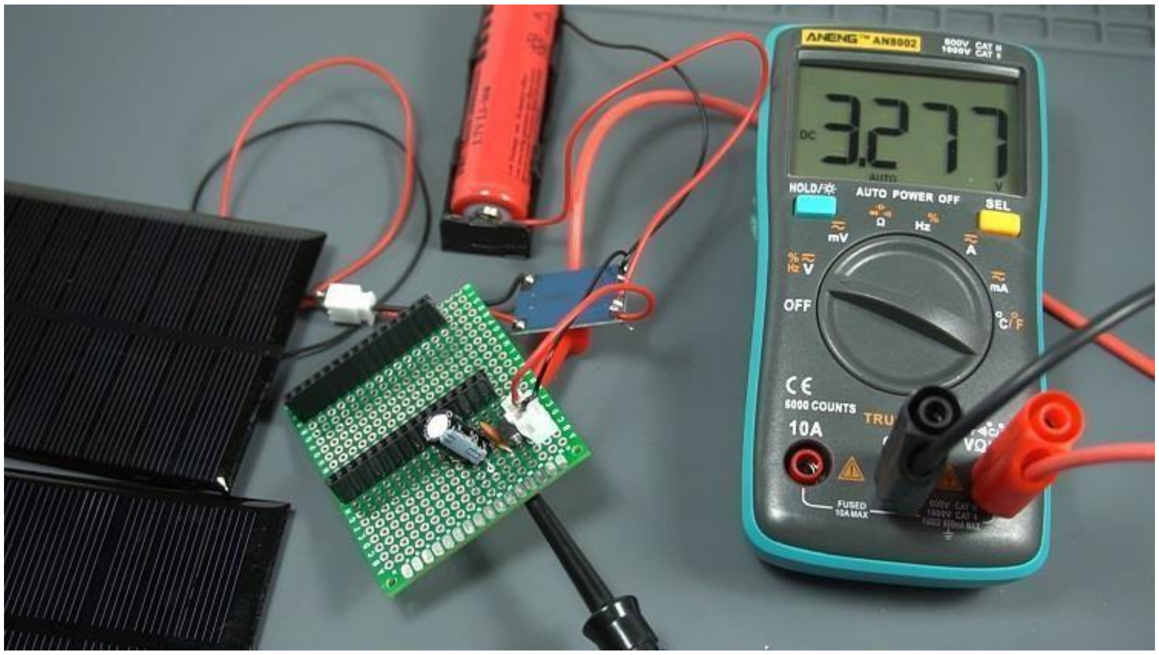
The LDOs should have a ceramic capacitor, and an electrolytic capacitor connected in parallel to GND and Vout to smooth the voltage peaks. Here we’re using a 100uF electrolytic capacitor, and a 100nF ceramic capacitor.

Follow the next schematic diagram to add the voltage regulator circuit to the previous setup.

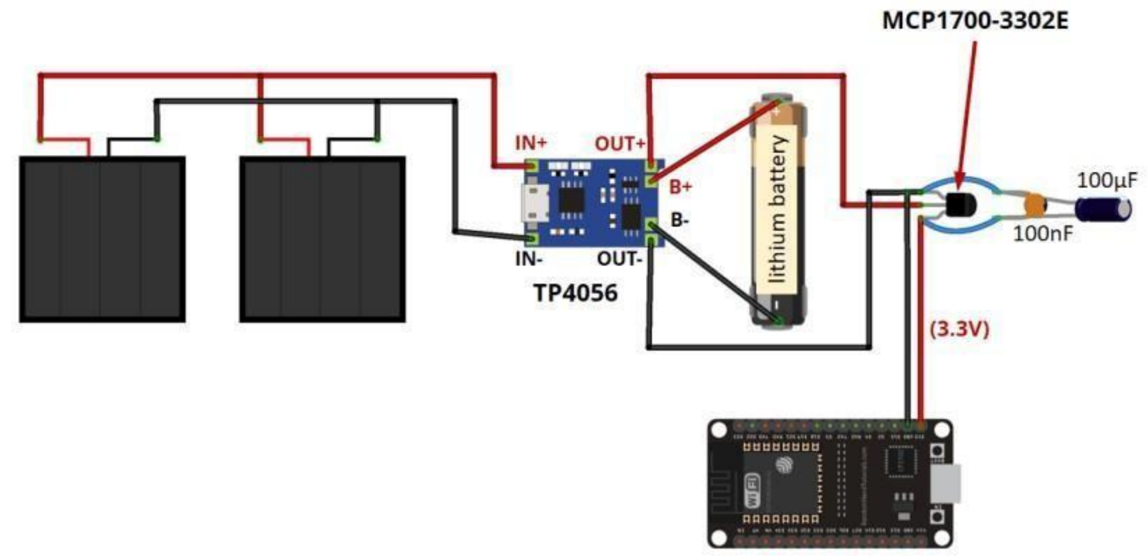


Warning: electrolytic capacitors have polarity! The lead with the white/gray strip should be connected to GND.

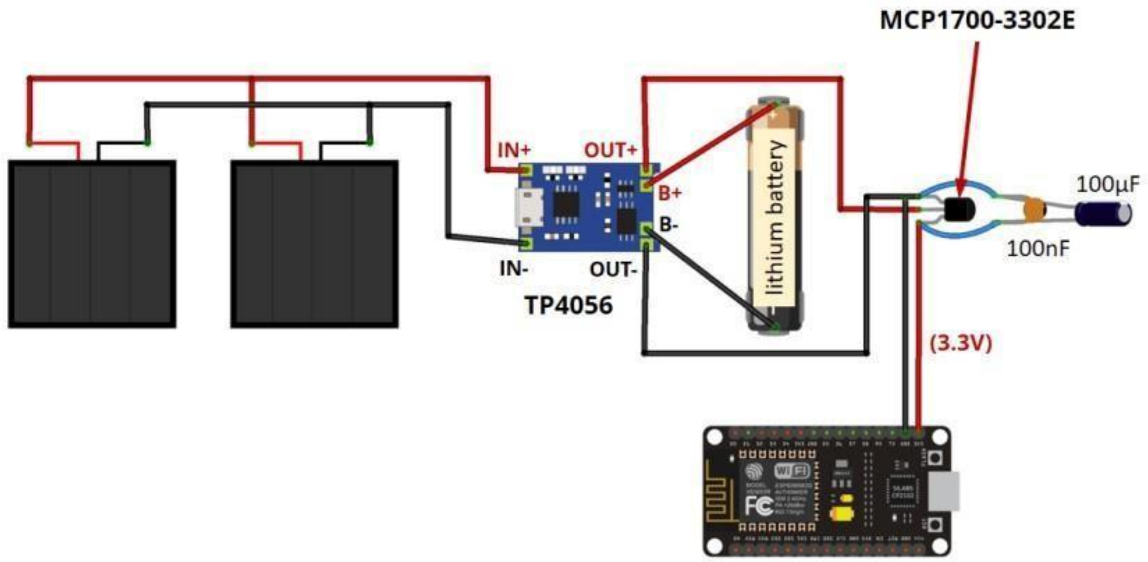
The Vout pin of the voltage regulator should output 3.3V. That is the pin that will power the ESP32 or ESP8266.



Finally, after making sure that you're getting the right voltage on the Vout pin of the voltage regulator, you can power the ESP32. Connect the Vout pin to the 3.3V pin of the ESP32 and GND to GND.



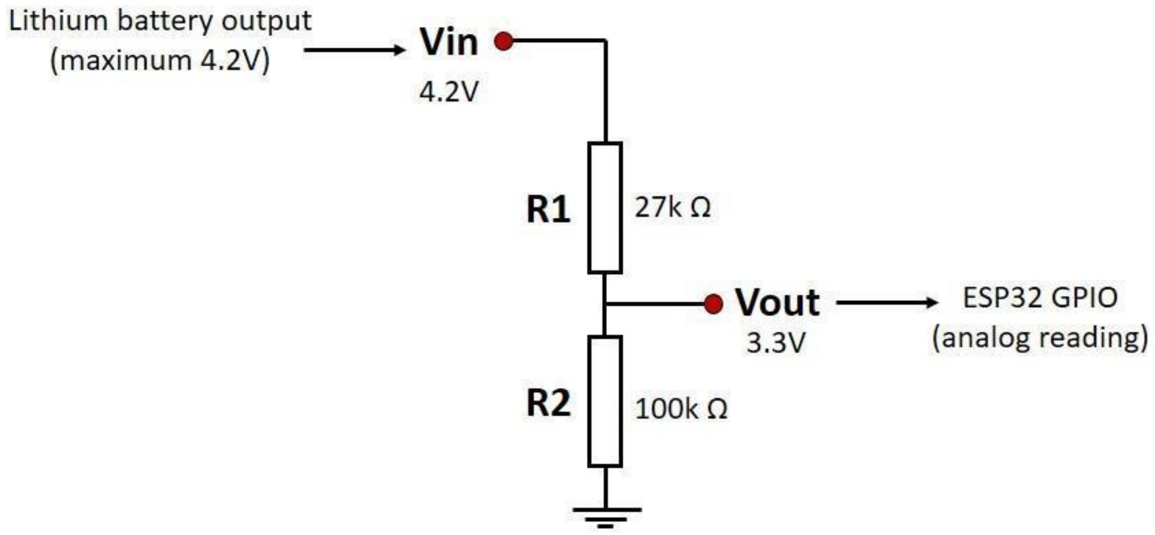
If you’re using an ESP8266 instead, you can follow the same circuit. Wire the output of the MCP1700-3302E to the ESP8266 3.3V pin and GND to GND.



Battery Voltage Level Monitoring Circuit

When you have your ESP32 powered with batteries or solar powered as in this case, it can be very useful to monitor the battery level. One way to do that is reading the output voltage of the battery using an analog pin of the ESP32.

However, the battery we’re using here outputs a maximum of 4.2V when fully charged, but the ESP32 GPIOs work at 3.3V. So, we need to add a voltage divider so that we’re able to read the voltage from the battery.



The voltage divider formula is as follows:

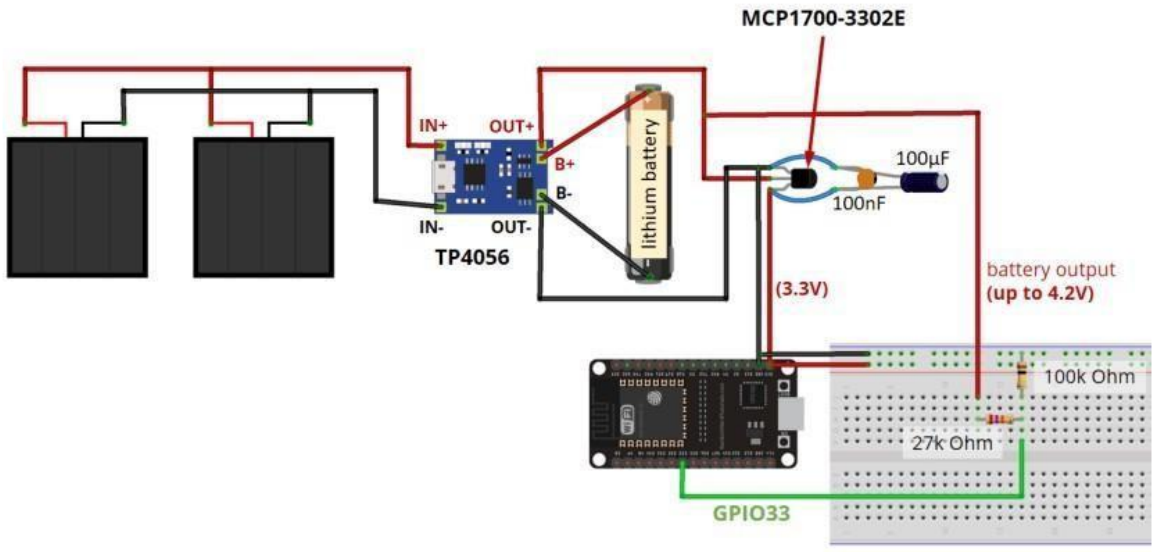
Vout = (Vin*R2)/(R1+R2)

So, if we use R1=27k Ohm, and R2=100k Ohm, we get:

Vout = (4.2*100k)/(27k + 100k) = 3.3V

So, when the battery is fully charged, the Vout outputs 3.3V that we can read with an ESP32 GPIO.

Add two resistors to your circuit as shown in the following schematic diagram.



In this case, we're monitoring the battery level through GPIO33, but you can use any other suitable GPIO. Read our ESP32 GPIO guide to learn which GPIOs are the best to use.

Finally, to get to the battery level, you can simply read the voltage on GPIO33 using the analogRead() function in your code (if you're using Arduino IDE).

```
analogRead (33);
```

You can also use the map() function, to convert the analog values to a percentage:

```
float batteryLevel = map(analogRead (33), 0.0f, 4095.0f, 0, 100);
```

Based on the information above, create a sketch that will read the charge of a solar panel powered battery and display it in the serial monitor.

SUMMARY

Power management and optimization are crucial considerations in embedded systems to ensure efficient operation and prolong battery life. By mastering the concepts covered in this module, students will be equipped with the knowledge and skills to design power-efficient embedded systems and optimize their performance. These principles can be applied in real-life scenarios such as designing IoT devices, wearable technology, and smart energy systems.

SUMMATIVE ASSESSMENT (SA)

______________________ 1. Devices or systems that function primarily on batteries are referred to as battery-powered embedded systems.

______________________ 2-4. Give three examples of battery-powered systems.

______________________ 5-7. Give three energy harvesting techniques.

______________________ 8. What are the two states of a device's power consumption.

______________________ 9. Give an action a software can do to minimize power consumption.

______________________ 10. What is the method in which it increases the software's effectiveness in embedded systems, which boosts performance and lowers power consumption.

LAB ACTIVITY

Using the materials from Activity 1 and 2, create a solar powered IoT weather station. You can choose any IoT platform that you like to record the read data from the dht sensor. Apply power optimization techniques in designing your circuit and writing the sketch.

REFERENCES / BIBLIOGRAPHY

Badat, A. (2019, August 17). *30 Examples of Embedded Systems in Daily Life - Comp Sci Station*. Comp Sci Station. https://compscistation.com/examples-embedded-systems-daily-life/

Cheng, C. H., & Van Hoesel, R. M. A. (2001). Power management for embedded systems. Computer, 34(4), 52-58. https://ieeexplore.ieee.org/abstract/document/906477

Darwish, T., & Bayoumi, M. (2005). *Trends in Low-Power VLSI Design*. In Elsevier eBooks (pp. 263–280). https://doi.org/10.1016/b978-012170960-0/50022-0

Energy Harvesting | *Mouser Electronics*. (n.d.). https://www.mouser.com/applications/energy_harvesting/#:~:text=The%20most%20widely%20used%20energy,the%20various%20energy%20harvesting%20devices

Marwedel, P. (2018). Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems (3rd ed.). Springer.

Messer, P. (2021). Embedded Systems Constraints – *SY0-601 CompTIA Security*+ : 2.6. Professor Messer IT Certification Training Courses. https://www.professormesser.com/security-plus/sy0-601/sy0-601-video/embedded-systems-constraints/

Henkel, J. (2005). *Power-aware computing*. ACM Computing Surveys (CSUR), 37(3), 1-51. https://dl.acm.org/doi/10.1145/1089733.1089735

Santos, S., & Santos, S. (2019). *Power ESP32/ESP8266 with Solar Panels and Battery | Random Nerd Tutorials*. Random Nerd Tutorials. https://randomnerdtutorials.com/power-esp32-esp8266-solar-panels-battery-level-monitoring/

Schulte-Hobein, M. (n.d.). *Power Management Techniques in Embedded Systems*. Digi. https://www.digi.com/blog/post/power-management-techniques-in-embedded-systems

Smelyanskiy, M., & Luk, C. K. (2015). *Data compression in modern computing environments*: techniques, performance, and prospects. ACM Computing Surveys (CSUR), 47(3), 1-36. https://dl.acm.org/doi/10.1145/2699430

Staff, E. (2012, March 6). *Understanding MCU sleep modes and energy savings*. Embedded.com. https://embedded.com/understanding-mcu-sleep-modes-and-energy-savings/

Walls, C. (2015, August 21). *Power management in embedded software*. Embedded.com. https://www.embedded.com/power-management-in-embedded-software/

What is energy harvesting? | ONiO. (n.d.). https://www.onio.com/article/what-is-energy-harvesting.html

Embedded Systems Security

INTRODUCTION

Embedded systems security refers to the procedures and policies used to safeguard the availability, integrity, and confidentiality of computer systems that are integrated into different types of hardware, assuring their resistance to harmful intrusions and unauthorized access. Embedded systems security is the one that will be discussed in this module. Specifically, it will discuss the security challenges including vulnerabilities and attack vectors as well as the secure boot and firmware update mechanisms. In addition, this will discuss cryptographic protocols and algorithms.

LEARNING OBJECTIVES / OUTCOMES

At the end of the module, students should be able to:

- Learn about the different security challenges in embedded systems.
- Learn about cryptographic protocols and algorithms.

DISCUSSION OF THE LESSON

Security Challenges in Embedded Systems

Due to resource constraints, a variety of attack vectors, complex supply chains, and the need for ongoing defense against evolving threats, ensuring robust security in several domains, including the automotive, healthcare, and industrial sectors, embedded systems pose significant challenges.

These difficulties include vulnerabilities in authentication and encryption procedures, obsolete or inadequate security measures, and the possibility of physical manipulation. Embedded systems are also subject to greater risks due to attack vectors and the potential for system compromise due to their interconnection with other devices and networks. In order to address these security issues effectively and preserve the integrity and dependability of embedded systems in a world that is becoming more networked, it is necessary to take a holistic strategy that includes safe design, strong authentication, encryption methods, and continuing security upgrades.

Vulnerabilities and Attack Vectors

The following are the different vulnerabilities in embedded systems:

1. The use of outdated or insecure software components - The usage of obsolete or insecure software components, such as operating systems and libraries, which may

have known vulnerabilities that attackers might take advantage of, is one such issue. Many embedded systems also lack appropriate authentication and access control protocols, leaving them open to manipulation and illegal access.

2. Limited computational resources – The restricted processing power of embedded systems, which frequently results in weakening of security measures, is another key vulnerability. Because resource limitations, encryption and cryptographic procedures, for instance, may be ineffective or nonexistent, leaving confidential information and communication channels open to interception and alteration.

3. Physical vulnerabilities – Physical access to the system might possibly jeopardize its security because they are frequently placed in uncontrolled conditions. Attackers can compromise the system's integrity and dependability by manipulating the hardware, removing the firmware, or injecting malicious code.

4. Interconnectivity of embedded systems - The attack surface is widened by interactions with other devices and networks, putting embedded systems at risk from threats including remote assaults, virus spread, and illegal data sharing.

Embedded systems are susceptible to various attack vectors that contribute to the security challenges they face. The following are:

1. Software vulnerabilities – Software defects, weaknesses, or coding errors in the software parts that make up an embedded system's functionality are referred to as software vulnerabilities. These flaws may be inadvertent, the result of programming errors or oversights, or hostile actors may put them there on purpose as backdoors. In embedded systems, complicated software components are frequently employed to perform activities including data processing, communication, and control operations. However, because of issues like time restraints, a lack of resources, and the requirement for real-time response, the creation and testing of software for embedded systems could not be as rigorously scrutinized as those of typical software programs. This may lead to the existence of concealed vulnerabilities that attackers may use.

2. Physical access – In order to use this attack vector, the embedded system or one of its components must be physically accessible to the attacker. They are able to directly manipulate the hardware, get confidential data, alter the firmware, or implant malicious hardware by doing this. By physically connecting to the system's ports or circuits, an attacker may, for instance, steal encryption keys or change system settings. Physical access assaults are especially dangerous when embedded devices are installed in open areas or in unprotected settings.

3. Network-based attacks – Network-based attacks have grown in importance as embedded systems' connections have increased. By taking advantage of flaws in network protocols or by using methods like infection with malware and denial-of-service assaults, attackers can target embedded devices through their network connections. Through these methods, an attacker may take control of the system, alter its functionality, intercept and change data being sent over the network, or start

disruptive operations that have an impact on the system's performance and availability.

4. Supply chain attacks - Attacks in the supply chain aim to compromise the integrity of the embedded system during the phases of production, distribution, or maintenance. In order to install malicious components in the embedded system, attackers might take advantage of security flaws or inject harmful components into the supply chain. The system may be vulnerable to unwanted access, data exfiltration, or control due to these compromised components or firmware. Since supply chain attacks take place before the system reaches the end user, it is essential to maintain the security of every step in the supply chain. Supply chain attacks are difficult to identify and counteract.

Knowing these vulnerabilities and attack vectors, a multi-layered strategy involving secure design principles, frequent software updates, robust authentication methods, encryption protocols, and physical security measures is needed to address these issues. By resolving these flaws, embedded systems' security posture may be improved, assuring their dependable and secure functioning across a range of deployment situations.

Secure boot and firmware update mechanisms

To prevent unwanted access, tampering, and to guarantee the software's integrity, secure boot and firmware update techniques are essential parts of embedded system security.

During the boot-up process, secure boot is a procedure that checks and confirms the integrity of the system's firmware or software. To prevent the execution of unauthorized or malicious code, it makes sure that only trusted and authenticated software components are loaded and executed. Secure boot depends on secure bootloaders and cryptographic signatures to validate the legitimacy and integrity of the firmware or software prior to execution. The likelihood of firmware-level assaults, such as firmware viruses or rootkits, which may compromise the entire system, is reduced because of this.

For embedded systems to remain secure and function over time, firmware update procedures are crucial. Regular firmware upgrades fix security holes, patch known vulnerabilities, and add new functions or enhancements. However, failing to update firmware safely might result in security problems. To prevent unauthorized or malicious firmware alterations, secure firmware update techniques guarantee the validity and integrity of the firmware upgrades. Usually, firmware updates are distributed through secure channels and digital signatures to accomplish this. The firmware image's validity is confirmed by digital signatures, which demonstrates that it originates from a reliable source and hasn't been tampered with. The firmware update is transferred safely through secure means to thwart hacker interception or alteration, such as encrypted connections or secure bootloaders.

Designing and implementing secure boot and firmware update processes carefully is necessary. Secure boot relies on trusted hardware components, secure bootloaders, and cryptographic techniques. For safe firmware upgrades, reliable encryption, secure methods of communication, and sound key management are essential. The security of the firmware update procedure is further improved by adopting secure update policies, such as code signing procedures and secure distribution channels.

In the end, embedded systems may preserve the integrity and security of their software by having strong secure boots and firmware update procedures. This prevents unwanted access and manipulation and makes sure the system is always up to date with the most recent security patches and upgrades.

Cryptographic Protocols and Algorithms

In many fields, including embedded systems, secure communication and data protection are based on cryptographic protocols and algorithms. In the presence of possible adversaries, these protocols and algorithms offer techniques for guaranteeing secrecy, integrity, authenticity, and non-repudiation of data. Sensitive data may be safely sent, stored, and processed by using cryptographic algorithms, which improves the embedded systems' overall security and dependability.

Cryptographic protocols and algorithms encompass a wide range of techniques designed to achieve specific security goals. Some commonly used cryptographic protocols include:

1. Asymmetric Key Cryptography – This system, also known as public-key cryptography, makes use of two keys—one public and one private—that are connected mathematically. While the private key is kept confidential and is used for decryption, the public key is used for encryption. Key distribution and digital signature capabilities are provided by asymmetric key cryptography, although it requires more processing power than symmetric key encryption.
2. Symmetric Key Cryptography – For both encryption and decryption, this kind of cryptography employs a single shared secret key. Although it normally needs a secure technique for key distribution, it is quicker and more effective than other types of encryptions.
3. Key exchange protocols – Key exchange protocols make it easier for communication parties to generate and exchange cryptographic keys in a safe manner. A good example is the Diffie-Hellman key exchange and its variations, which enabled two parties to create a shared secret key via an insecure channel.
4. Hash functions - Data is transformed by hash functions into a fixed-length string of bits known as a hash or message digest. These operations can be used to check the accuracy of transmitted or stored data and are essential for assuring data integrity.

For cryptographic algorithms, the following are:

1. RSA – For safe key exchange, digital signatures, and the encryption and decryption of tiny data blocks, RSA is a popular asymmetric encryption scheme.
2. Secure Hash Algorithms – Hash functions that employ SHA algorithms, such SHA-256, are frequently used to validate data and create digital signatures.
3. Advanced Encryption Standard – Widely used and regarded as a very safe symmetric encryption method, AES is used to safeguard sensitive data.

4. Elliptic Curve Cryptography - ECC is an asymmetric encryption technique that makes use of elliptic curve mathematics to offer robust security with comparatively tiny key sizes, making it appropriate for devices with limited resources.

Encryption and Authentication

Encryption and authentication are two crucial aspects of cryptographic protocols and algorithms that work together to provide secure communication and data protection in embedded systems.

By means of cryptographic methods and keys, plaintext is converted into ciphertext during encryption. By rendering the data unreadable to unauthorized parties, it protects data secrecy. Data that has been encrypted can only be unlocked by those who have the right decryption key. Even if the data is intercepted during transmission or kept in an unsafe setting, encryption prevents eavesdropping and unauthorized access to important information. For encryption, symmetric key cryptography and asymmetric key cryptography are frequently employed.

On the other hand, verifying a communicating party's identity or reliability, as well as the reliability of data, is the process of authentication. It makes sure the sender and recipient of the material are who they say they are, and that the data hasn't been altered during transmission or storage. Authentication stops unauthorized individuals from impersonating legitimate businesses or secretly changing data. Authentication may be accomplished using a variety of cryptographic methods. Digital signatures, which provide a unique signature for a piece of data using asymmetric key cryptography, are one often used method. The integrity and validity of the data as well as the sender's identity are subsequently confirmed by the signature's verification using the associated public key.

In conclusion, a complete security solution for embedded systems is offered by the incorporation of encryption and authentication in cryptographic protocols and algorithms. Sensitive information is kept private through encryption, even if it is intercepted. Data authenticity is ensured by authentication, which prevents unwanted changes or impersonation. Together, these measures protect embedded systems against illegal access and data breaches, enhancing their overall security and dependability. To provide reliable encryption and authentication processes within embedded systems, it is essential to design safe cryptographic protocols and algorithms correctly.

Secure Communication Protocols

Establishing private and secure channels for data transmission requires the use of secure communication protocols, which rely on cryptographic protocols and algorithms to accomplish their goals. In addition to ensuring the secrecy, integrity, and authentication of the sent data, these protocols offer the basis for secure communication between parties. The following are some secure communication protocols:

1. Transport Layer Security (TLS) - TLS is frequently used in web browsers, email clients, and other programs to provide secure communication over the internet. To create safe

and authorized connections, it uses a variety of cryptographic techniques, including symmetric encryption, asymmetric encryption, and hash functions.

2. Internet Protocol Security - IPSec offers means for secure data transfer, data integrity, and data origin authentication and is frequently used to protect network communication at the IP layer. To safeguard IP packets and guarantee secure communication between network entities, it makes use of cryptographic protocols and methods like the Encapsulating Security Payload (ESP) and Authentication Header (AH), along with encryption techniques like AES and hash functions like SHA.

In conclusion, secure communication protocols use cryptographic protocols and algorithms to provide private, secure channels for data transfer. These protocols guarantee the secrecy, integrity, and authenticity of the transmitted data by using encryption, key exchange systems, and hashing operations. The unique application, security needs, and resource limitations of the embedded system or network environment all affect the protocols and algorithms that are used.

ACTIVITIES / EXAMPLES

ACTIVITY 1:

Objective: Implement the Caesar cipher encryption and decryption algorithm for ESP32 in the Arduino IDE.

Requirements:

- ESP32 development board
- Arduino IDE
- Basic understanding of C++ programming and Arduino

Instructions:

- Set up your Arduino IDE and connect your ESP32 development board to your computer.
- Create a new Arduino sketch and give it a suitable name.
- Implement the encryption function caesar_encrypt that takes a plain text message and returns the corresponding ciphertext. The function should shift each character in the message by a fixed number of positions to the right.
- Implement the decryption function caesar_decrypt that takes a ciphertext message and returns the corresponding plaintext. The function should shift each character in the message by the same number of positions to the left.
- Write a simple setup function to initialize your ESP32 board.
- Write a simple loop function to test your implementation. In the loop function, prompt the user to enter a message, encrypt it using caesar_encrypt, print the resulting ciphertext, decrypt the ciphertext using caesar_decrypt, and print the decrypted plaintext.

  Here's an example implementation to help you get started:

```
// Libraries
```

```
#include <WiFi.h>
#define KEY 3
char caesar_encrypt(char ch) {
    if (ch >= 'a' && ch <= 'z')
        return 'a' + (ch - 'a' + KEY) % 26;
    else if (ch >= 'A' && ch <= 'Z')
        return 'A' + (ch - 'A' + KEY) % 26;
    else
        return ch;
}

char caesar_decrypt(char ch) {
    if (ch >= 'a' && ch <= 'z')
        return 'a' + (ch - 'a' - KEY + 26) % 26;
    else if (ch >= 'A' && ch <= 'Z')
        return 'A' + (ch - 'A' - KEY + 26) % 26;
    else
        return ch;
}

void setup() {
  Serial.begin(115200);
}

void loop() {
  char message[100];
  Serial.println("Enter a message:");
  while (!Serial.available()) {
  delay(100);
  }
  delay(100); // wait for the complete message
  int messageLength = Serial.readBytesUntil('\n', message, sizeof(message) - 1);
  message[messageLength] = '\0';

  // Encryption
  for (int i = 0; i < messageLength; ++i) {
    message[i] = caesar_encrypt(message[i]);
  }
  Serial.print("Encrypted message: ");
  Serial.println(message);

  // Decryption
  for (int i = 0; i < messageLength; ++i) {
    message[i] = caesar_decrypt(message[i]);
  }
  Serial.print("Decrypted message: ");
  Serial.println(message);

  delay(1000);
}
```

You can upload this sketch to your ESP32 development board using the Arduino IDE and open the Serial Monitor to interact with the program.

Please note that this implementation uses the Serial Monitor for input and output. You can modify and enhance it according to your needs and the capabilities of the ESP32 board.

ACTIVITY 2:

Objective: Implement a hash function for ESP32 in the Arduino IDE.

Requirements:

- ESP32 development board

- Arduino IDE
- Basic understanding of C++ programming and Arduino

Instructions:

- Set up your Arduino IDE and connect your ESP32 development board to your computer.
- Create a new Arduino sketch and give it a suitable name.
- Implement a hash function that takes a plain text message and returns the corresponding hash value. You can choose a simple hash function like the DJB2 hash function.
- Write a simple setup function to initialize your ESP32 board.
- Write a simple loop function to test your hash function. In the loop function, prompt the user to enter a message, compute the hash value using your hash function, and print the resulting hash value.

  Here's an example implementation using the DJB2 hash function to help you get started:

```
// Libraries

#include <WiFi.h>


unsigned long djb2_hash(const char* str) {

   unsigned long hash = 5381;

   int c;


   while ((c = *str++)) {

      hash = ((hash << 5) + hash) + c; // hash = hash * 33 + c

   }


   return hash;

}


void setup() {

   Serial.begin(115200);

}


void loop() {
   char message[100];

   Serial.println("Enter a message:");
   while (!Serial.available()) {
      delay(100);
   }
   delay(100); // wait for the complete message
   int messageLength = Serial.readBytesUntil('\n', message, sizeof(message) - 1);
   message[messageLength] = '\0';

   // Compute the hash value
   unsigned long hashValue = djb2_hash(message);

   Serial.print("Hash value: ");
   Serial.println(hashValue);

   delay(1000);
}
```

You can upload this sketch to your ESP32 development board using the Arduino IDE and open the Serial Monitor to interact with the program.

Please note that this implementation uses the Serial Monitor for input and output. You can modify and enhance it according to your needs and the capabilities of the ESP32 board. Additionally, the DJB2 hash function used in this example is a simple hash function and may not be suitable for all use cases. For cryptographic purposes, it is recommended to use stronger hash functions like SHA-256 or SHA-3.

SUMMARY

Due to resource limitations, convoluted supply chains, and the requirement for defense against emerging threats, embedded systems confront substantial security issues. Risks to embedded systems include weaknesses including out-of-date software components, scarce computing resources, physical access, and interconnectivity. Software flaws, physical access, network-based assaults, and supply chain attacks are all examples of attack vectors. For avoiding unwanted access and manipulation, secure boot and firmware update procedures are crucial, and cryptographic protocols and algorithms provide secure communication and data security. These ideas have practical applications in the automotive, medical, and industrial control systems where strong security is essential to safeguard sensitive information, maintain system integrity, and win over users.

SUMMATIVE ASSESSMENT (SA)

Answer the following questions. Choose the letter of the correct answer.

1. What is one of the key vulnerabilities in embedded systems?
   a) Limited computational resources
   b) Excessive encryption measures
   c) Overwhelming physical security
   d) Complete isolation from other devices

2. Which attack vector requires physical access to the embedded system?
   a) Software vulnerabilities
   b) Network-based attacks
   c) Supply chain attacks
   d) Physical access

3. What is the purpose of secure boot in embedded systems?
   a) To authenticate users during system startup
   b) To ensure the system runs outdated software components
   c) To prevent unauthorized or malicious code execution
   d) To enhance computational resources

4. Why are firmware update procedures crucial for embedded systems?
   a) To increase the attack surface
   b) To prevent secure boots from functioning properly
   c) To patch security vulnerabilities and add new features
   d) To limit the functionality of the system

5. Which cryptographic algorithm is commonly used for secure key exchange and digital signatures?
   a) RSA
   b) AES
   c) SHA-256
   d) ECC

6. What does encryption achieve in embedded systems?
   a) It guarantees data integrity
   b) It prevents physical manipulation
   c) It provides secure communication channels
   d) It increases computational resources

7. How does authentication contribute to the security of embedded systems?
    a) It ensures the system has limited access to external devices
    b) It protects against software vulnerabilities
    c) It verifies the identity and integrity of communicating parties
    d) It eliminates the need for encryption protocols

8. Which secure communication protocol is commonly used to provide secure web browsing?
    a) TLS
    b) IPSec
    c) HTTP
    d) SSL

9. What is the purpose of Internet Protocol Security (IPSec)?
    a) To ensure the physical security of embedded systems
    b) To provide secure communication at the IP layer
    c) To encrypt all network traffic within an embedded system
    d) To protect against physical access attacks

10. How do secure communication protocols ensure data integrity and authenticity?
    a) By using encryption and key exchange mechanisms
    b) By isolating embedded systems from external networks
    c) By limiting computational resources for data processing
    d) By avoiding the use of cryptographic algorithms

REFERENCE / BIBLIOGRAPHY

AboutSSL. (2020). *5 Common Security Challenges of Embedded Systems*. https://aboutssl.org/common-security-challenges-of-embedded-systems/

Apriorit. (2020). *12 Common Attacks on Embedded Systems and How to Prevent Them*. https://www.apriorit.com/dev-blog/690-embedded-systems-attacks

Synopsys. *What is cryptography?* https://www.synopsys.com/glossary/what-is-cryptography.html

Quadir, M. (2020). *Embedded Systems Authentication and Encryption Using Strong PUF Modeling*. https://ieeexplore.ieee.org/document/9043104

Case Studies and Practical Examples

INTRODUCTION

This module provides an overview of embedded systems in the automotive industry and explores their applications in vehicle control, safety, and infotainment systems. This material also outlines the role of embedded systems in consumer electronics and explores their applications in various devices such as smartphones, smart TVs, and wearables.

Embedded systems also play a critical role in healthcare applications, enabling advanced medical devices and improving patient care. This module will explore the use of embedded systems in various healthcare settings and their impact on enhancing diagnostics, treatment, and patient monitoring.

In addition, this module will run through the importance of embedded systems in the context of industrial automation. It covers the definition, components, applications, and benefits of embedded systems in industrial settings.

LEARNING OBJECTIVES / OUTCOMES

At the end of the module, students should be able to:

- Understand the concept of embedded systems and their significance in vehicles, and consumer electronics, as well as define the role of embedded systems in the context of healthcare applications and industrial automation.
- Explore the key components and features of embedded systems in automotive applications, consumer electronics, healthcare applications, and industrial automation.
- Explore the applications of embedded systems in safety-related functions.
- Understand how embedded systems contribute to infotainment systems in vehicles.
- Explain the applications of embedded systems in smartphones, smart TVs, and wearable devices.
- Recognize the role of embedded systems in enhancing functionality, performance, and user experience in consumer electronics.
- Evaluate the impact of embedded systems on optimizing power consumption, reducing device size, and enabling advanced features in consumer electronic devices.
- Identify different types of embedded systems used in healthcare devices, such as microcontrollers, processors, sensor technologies, communication interfaces, and real-time operating systems.
- Explain the applications of embedded systems in medical diagnostics, including the role of embedded systems in imaging systems, diagnostic devices, laboratory equipment, and point-of-care testing devices.
- Describe the applications of embedded systems in medical treatment, including their use in drug delivery systems, surgical devices, rehabilitation devices, and prosthetics.

- Discuss the applications of embedded systems in patient monitoring, including their role in vital signs monitoring devices, wearable devices, smart hospital beds, and telemedicine devices.
- Explain the significance of embedded systems in improving productivity, efficiency, and safety in industrial settings.
- Describe the role of each component in the functionality of embedded systems.
- Discuss the applications of embedded systems in industrial control systems and robotics.
- Identify the benefits and challenges associated with implementing embedded systems in industrial automation.

DISCUSSION OF THE LESSON

EMBEDDED SYSTEMS IN AUTOMOTIVE INDUSTRY

Introduction to Embedded Systems in the Automotive Industry

Embedded systems are specialized computer systems designed to perform specific tasks within a larger system. In the automotive industry, embedded systems play a crucial role in enabling advanced functionalities and improving vehicle performance, safety, and user experience. They are integrated into various components and subsystems of a vehicle, including engine control units, anti-lock braking systems, and infotainment systems. By combining hardware and software, embedded systems provide precise control and automation, collecting data from sensors, processing it, and executing the necessary actions in real-time.

The automotive industry relies heavily on embedded systems for various functions, such as engine management, vehicle dynamics control, safety systems, and infotainment. Embedded systems in engine control units (ECUs) monitor and control engine parameters, optimizing fuel injection, ignition timing, and emissions. They enable efficient power delivery and enhance fuel economy. Anti-lock braking systems (ABS) utilize embedded systems to prevent wheel lock-up during braking, improving control and reducing stopping distances. Electronic stability control (ESC) further enhances vehicle stability by selectively applying brakes and adjusting engine power to maintain control in challenging driving conditions.

Infotainment systems in vehicles incorporate embedded systems to provide entertainment and connectivity features. Embedded systems in in-car entertainment systems offer multimedia functionalities, including audio playback, video display, and touchscreen interfaces. Navigation systems integrate GPS technology and mapping data to provide accurate navigation, route guidance, and real-time traffic information. Additionally, embedded systems facilitate connectivity options like Bluetooth, Wi-Fi, and mobile device integration, enabling hands-free calling, media streaming, and access to mobile applications.

Components and Features of Automotive Embedded Systems

Automotive embedded systems comprise several key components and features that enable their functionalities. Microcontrollers and processors are at the heart of these systems. Microcontrollers are integrated circuits specifically designed for controlling embedded systems, providing processing power and managing various functions. Processors, such as those based on ARM architecture, are used for more complex computing tasks, including graphics processing and multimedia handling.

Embedded systems heavily rely on sensors and actuators for data acquisition and control. Sensors, such as temperature sensors, accelerometers, and proximity sensors, detect and measure physical phenomena or parameters in the vehicle's environment. They play a critical role in acquiring data for control and monitoring purposes. Actuators, on the other hand, convert electrical signals into physical actions, controlling various vehicle systems. Examples include motors, solenoids, and valves that regulate fuel flow, air intake, and exhaust systems.
Communication interfaces and protocols are vital for interconnecting embedded systems within a vehicle. Different systems, such as ECUs, ABS, and infotainment systems, need to exchange data efficiently and reliably. Communication interfaces like the Controller Area Network (CAN), Local Interconnect Network (LIN), and Ethernet enable seamless data transfer between these systems. Protocols ensure standardized and consistent communication, allowing embedded systems to interact and cooperate effectively.

Software architectures and real-time operating systems (RTOS) play a significant role in the design and implementation of automotive embedded systems. Software architecture provides a structured approach to manage the complexity of these systems. Layered architectures, for example, separate software components into distinct layers, enhancing modularity and maintainability. Real-time operating systems ensure timely and predictable execution of critical tasks. They prioritize tasks based on their deadlines and requirements, ensuring that time-critical operations, such as engine control or collision detection, are handled promptly.

Applications of Embedded Systems in Vehicle Control

Embedded systems are extensively used in vehicle control systems to enhance performance, efficiency, and safety. Engine control units (ECUs) are a prime example of embedded systems in vehicle control. These units continuously monitor engine parameters, such as fuel injection timing, ignition timing, and emissions, using data from sensors. Based on this data, the embedded systems optimize engine performance, improve fuel efficiency, and ensure compliance with regulatory standards.

Anti-lock braking systems (ABS) are another critical application of embedded systems in vehicle control. ABS prevents wheel lock-up during breaking, enabling drivers to maintain steering control and reduce stopping distances. The embedded systems in ABS monitor wheel speed using sensors and selectively modulate brake pressure to each wheel to prevent lock-up. This precise control enhances safety by allowing the driver to maneuver the vehicle safely during emergency braking or slippery conditions.

Transmission control systems also heavily rely on embedded systems. These systems ensure smooth gear shifting, optimize fuel efficiency, and provide a comfortable driving experience. Embedded systems monitor various parameters, such as vehicle speed and engine load, to determine the optimal gear shift points. By coordinating the operation of the transmission, engine, and other drivetrain components, embedded systems facilitate efficient power delivery and improve overall vehicle performance.

In recent years, the automotive industry has witnessed the introduction of adaptive cruise control systems. These systems employ embedded systems to maintain a safe distance from vehicles ahead. By using sensors to detect the distance and relative speed of vehicles in front, the embedded systems adjust the throttle and brakes to ensure a safe and consistent following distance. Adaptive cruise control enhances convenience and reduces driver fatigue during long journeys.

Applications of Embedded Systems in Safety Systems

Embedded systems play a vital role in enhancing vehicle safety through various safety systems. Airbag control systems are a prominent application of embedded systems in vehicle safety. These systems utilize embedded systems to process data from crash sensors and determine the appropriate timing and force for airbag deployment. By analyzing sensor data in real-time, the embedded systems ensure that airbags are deployed accurately and at the right moment to provide optimal protection to vehicle occupants during a collision.

Tire pressure monitoring systems (TPMS) are another example of embedded systems applied to enhance safety. These systems employ embedded systems and sensors to monitor tire pressure in real-time. By continuously monitoring tire pressure, the embedded systems can detect any significant deviations from the optimal pressure range. If a low-pressure condition is detected, the embedded systems alert the driver, enabling them to take appropriate action and prevent potential tire failures or accidents caused by underinflated tires.

Embedded systems are crucial components of collision detection and avoidance systems. These systems utilize various sensor technologies, such as radar or lidar, to detect potential collisions or hazards. The embedded systems analyze sensor data and trigger appropriate actions, such as activating automatic emergency braking or providing visual and audible warnings to the driver. By enabling timely responses and interventions, these embedded systems help prevent or mitigate collisions, thereby enhancing overall vehicle safety.

Applications of Embedded Systems in Infotainment Systems

Embedded systems play a significant role in enhancing the infotainment experience in vehicles. In-car entertainment and multimedia systems rely on embedded systems to provide a rich multimedia experience to vehicle occupants. Embedded systems in these systems support audio playback, video display, and touchscreen interfaces. They enable various functionalities, including playing music, watching videos, and interacting with multimedia applications through intuitive user interfaces.

Navigation systems in vehicles also heavily utilize embedded systems. These systems integrate GPS technology and mapping data to provide accurate navigation, route guidance, and real-time traffic.

EMBEDDED SYSTEMS IN CONSUMER ELECTRONICS

Definition and Importance of Embedded Systems in Consumer Electronics

Embedded systems are specialized computer systems designed to perform specific tasks within larger systems, such as consumer electronic devices. These systems are tightly integrated into the hardware and software components of consumer electronics to provide precise control, automation, and enhanced functionalities. Embedded systems play a crucial role in improving the functionality, performance, and user experience of consumer electronics. They enable advanced features, reduce device size, optimize power consumption, and enhance overall performance.

Types of Embedded Systems in Consumer Electronics

Consumer electronics utilize different types of embedded systems based on the complexity and requirements of the devices. Microcontrollers and microprocessors are commonly used as the core components in embedded systems. Microcontrollers are designed for low-power applications and have built-in memory and peripherals, making them suitable for simple tasks. On the other hand, microprocessors are more powerful and suitable for complex tasks. Another important aspect of embedded systems in consumer electronics is the System-on-Chip (SoC) architecture, where multiple functions are integrated into a single chip. This integration reduces the size and power consumption of the devices. Firmware and software components are also integral parts of embedded systems, working in conjunction with the hardware to provide the desired functionalities and user interfaces.

Applications of Embedded Systems in Smartphones

Smartphones heavily rely on embedded systems to provide various features and functionalities. Embedded systems in smartphones manage processors, memory, touchscreens, cameras, and connectivity modules. They optimize power consumption and extend battery life by employing power management algorithms and sleep modes. Embedded systems enable touchscreens, gesture recognition, and camera functionalities, enhancing the user experience. They also play a crucial role in optimizing performance, multitasking, and seamless app integration.

Topic 4: Applications of Embedded Systems in Smart TVs

Smart TVs integrate embedded systems to provide advanced features and connectivity options. Embedded systems manage the processors, operating systems, and multimedia playback capabilities of smart TVs. They enable applications, voice control, and artificial intelligence integration, enhancing the viewing experience and user convenience. Embedded systems in smart TVs handle multimedia formats and playback, ensuring smooth and high-quality audiovisual experiences for users. They also support internet connectivity, streaming services, and app integration, allowing users to access a wide range of content and personalize their viewing preferences.

Applications of Embedded Systems in Wearable Devices

Wearable devices, such as smartwatches and fitness trackers, heavily rely on embedded systems for their operation. Embedded systems in wearable devices manage sensors, data processing, health monitoring, activity tracking, and wireless communication. They enable real-time data tracking and synchronization with smartphones or other devices. Embedded systems in wearable devices process sensor data, perform calculations, and communicate wirelessly, providing users with personalized experiences. They enable integration with mobile applications, displaying notifications, exchanging data, and enhancing overall functionality.

EMBEDDED SYSTEMS IN HEALTHCARE APPLICATIONS

Definition and Importance of Embedded Systems in Healthcare Applications

Embedded systems in healthcare refer to specialized computer systems integrated into medical devices and systems to perform specific tasks. These systems are designed to handle complex algorithms, collect, and process data, and provide real-time analysis and control. Embedded systems are crucial in healthcare applications as they enable accurate diagnostics, precise treatment delivery, and continuous patient monitoring, leading to improved patient outcomes and enhanced healthcare delivery.

Types of Embedded Systems in Healthcare Applications

Microcontrollers and processors form the backbone of embedded systems in healthcare devices. These components provide the computational power and control necessary for medical devices to perform their intended functions. They handle tasks such as data acquisition, signal processing, and decision-making algorithms. Sensor technologies, such as biomedical sensors, enable data acquisition from patients and the environment. These sensors capture vital physiological parameters, such as heart rate, blood pressure, and temperature, which are essential for diagnostics, treatment, and monitoring.

Communication interfaces and protocols facilitate the exchange of data between embedded systems and other devices or systems in healthcare settings. This communication allows seamless integration of medical devices into hospital information systems, electronic health records, or telemedicine platforms. Real-time operating systems (RTOS) provide the necessary framework for managing tasks, scheduling priorities, and ensuring timely responses in time-critical healthcare applications. Software architecture help organize the software components running on embedded systems, enabling efficient development, maintenance, and integration of healthcare devices and systems.

Applications of Embedded Systems in Medical Diagnostics

Embedded systems are extensively used in medical diagnostics to enable accurate and efficient analysis of patient data. Imaging systems, such as X-ray machines, computed tomography (CT) scanners, and magnetic resonance imaging (MRI) systems, rely on embedded systems to capture and process images. These systems handle complex algorithms for image reconstruction and enhancement, enabling precise visualization of anatomical structures and detection of abnormalities.

Diagnostic devices, such as blood analyzers, molecular diagnostic systems, and point-of-care testing devices, utilize embedded systems for rapid and accurate analysis of patient samples. These systems perform tasks such as sample processing, detection of specific biomarkers, and interpretation of test results. Laboratory equipment, such as spectrometers and chromatographs, incorporate embedded systems for precise measurements and analysis of samples. These systems ensure accurate quantification of substances, enabling reliable laboratory testing and research.

Point-of-care testing devices, including glucometers, pregnancy tests, and rapid infectious disease diagnostics, rely on embedded systems to provide rapid and accurate results at the patient's bedside or in remote locations. These systems handle complex algorithms for data interpretation and deliver timely information to facilitate immediate decision-making and appropriate patient care.

Applications of Embedded Systems in Medical Treatment

Embedded systems play a vital role in medical treatment by enabling precise and controlled delivery of therapies. Drug delivery systems, such as insulin pumps, infusion pumps, and implantable drug delivery devices, rely on embedded systems to control the administration of medications. These systems monitor patient-specific parameters, such as blood glucose levels or drug dosing requirements, and deliver medications in a controlled manner to ensure optimal therapeutic outcomes.

Surgical devices, including robotic surgical systems, laparoscopic instruments, and laser-based surgical tools, incorporate embedded systems for precise control and feedback during surgical procedures. These systems enable surgeons to perform minimally invasive procedures, enhance visualization, and ensure accurate manipulation of surgical instruments. Embedded systems provide real-time feedback and support advanced features like image-guided surgery or haptic feedback, improving surgical precision and patient safety.

Rehabilitation devices, such as prosthetic limbs, exoskeletons, and assistive devices, rely on embedded systems to provide personalized therapy and enhance mobility. These systems utilize embedded sensors to capture movement data, analyze patient-specific parameters, and provide feedback to assist with gait, balance, and movement coordination. Embedded systems adapt therapy based on patient progress and allow customization of device settings to meet individual needs.

Applications of Embedded Systems in Patient Monitoring

Embedded systems are extensively used in patient monitoring, enabling continuous and real-time assessment of patient health parameters. Vital signs monitoring devices, such as electrocardiogram (ECG) monitors, pulse oximeters, and blood pressure monitors, incorporate embedded systems to capture, process, and display vital physiological parameters. These systems analyze sensor data, perform algorithms for arrhythmia detection or blood pressure calculation, and provide real-time feedback to healthcare providers for immediate intervention.

EMBEDDED SYSTEMS IN INDUSTRIAL AUTOMATION

Definition of Embedded Systems in Industrial Automation

Embedded systems in industrial automation are specialized computer systems that control and monitor industrial processes. They are designed to perform specific tasks and are integrated into machinery, equipment, and systems. Embedded systems play a vital role in automating industrial processes and improving productivity, efficiency, and safety.

Significance of Embedded Systems in Industrial Automation

Embedded systems enhance productivity in industrial settings by automating manual tasks, reducing errors, and improving process efficiency. They enable real-time data acquisition, analysis, and control, resulting in optimized operations. Embedded systems also contribute to increased safety by implementing safety protocols, monitoring critical parameters, and ensuring safe equipment operation.

Overview of the Role of Embedded Systems in Industrial Automation

Embedded systems form the backbone of industrial automation, providing control, monitoring, and communication capabilities. They seamlessly integrate various components such as microcontrollers, sensors, actuators, and communication interfaces. Embedded systems facilitate real-time decision-making, data logging, remote monitoring, and connectivity, enabling effective industrial control and automation.

Components of Embedded Systems in Industrial Automation

Microcontrollers and Processors used in Embedded Systems for Industrial Automation

Microcontrollers and processors are the core components of embedded systems. They provide computational power, memory, and control capabilities to execute control algorithms, process data, and communicate with external devices. These components are specifically designed for industrial environments and offer robustness, reliability, and real-time response.

Sensors and Actuators for Data Acquisition and Control

Sensors capture data from the physical environment, while actuators control physical processes based on the input received from sensors. Sensors measure parameters such as temperature, pressure, flow, and position, while actuators convert electrical signals into mechanical actions. They enable precise control over industrial processes.

Communication Interfaces and Protocols for Interconnecting Embedded Systems

Communication interfaces and protocols facilitate the exchange of data between embedded systems, industrial devices, and control systems. These interfaces include Ethernet, serial communication (RS-232/485), and industrial protocols such as Modbus or Profibus. The choice of communication interfaces and protocols depends on factors such as data transfer speed, distance, reliability, and compatibility with existing industrial systems.

Software Architectures and Real-Time Operating Systems used in Industrial Automation

Software architecture provides the structure and organization for the software components of embedded systems. They define how different modules interact, handle data, and execute control algorithms. Real-time operating systems (RTOS) are commonly used in industrial automation to ensure precise timing and response. RTOS provides deterministic behavior, task scheduling, and real-time communication, which are essential for time-critical industrial applications.

Applications of Embedded Systems in Industrial Automation

Industrial Control Systems

Embedded systems play a vital role in industrial control systems. Introduction to programmable logic controllers (PLCs) and their role in industrial automation. Embedded systems are used for monitoring and controlling industrial processes, ensuring efficient and reliable operation. They are also implemented in supervisory control and data acquisition (SCADA) systems for centralized control and monitoring of industrial processes.

Robotics in Industrial Automation

Embedded systems are extensively used in industrial robotics. They control robotic systems, enable motion planning, control, and integrate sensor feedback. Embedded systems facilitate precise and coordinated movements, enhancing the efficiency and effectiveness of industrial robots. They are applied in various tasks such as assembly, welding, material handling, and inspection.

Benefits and Challenges of Embedded Systems in Industrial Automation

Benefits of Embedded Systems in Industrial Automation

Embedded systems offer numerous benefits in industrial automation, including increased productivity, improved efficiency, and enhanced safety. They automate repetitive tasks, optimize control algorithms, and provide real-time monitoring and decision-making capabilities. Embedded systems enable seamless integration, data-driven insights, and improved process control.

Challenges in Implementing Embedded Systems in Industrial Automation

Implementing embedded systems in industrial automation can pose challenges. These include system integration with existing infrastructure, ensuring compatibility with legacy systems, and managing complex software and hardware interactions. Cybersecurity is also a critical concern, requiring robust measures to protect embedded systems and industrial networks from cyber threats.

ACTIVITIES

Activity 1: Exploring Embedded Systems in Vehicles

Instructions: Select a vehicle for the activity, which can be either a physical vehicle or a virtual simulation and then identify the embedded systems within it.

Activity 2: Knowing Embedded Systems in Automobiles

Instructions: Provide five examples of embedded systems in automobiles and provide a brief explanation of each.

Activity 3: Exploring Embedded Systems in Consumer Electronics

Instructions: Think of at least ten consumer electronics that you have in your house and provide their brands and specifications.

Activity 4: Interview Time

Instructions: Interview five random people about their most frequently used consumer electronics device or product and write about their reasons for purchasing it. List their responses below.

Activity 5: Exploring Embedded Systems in Healthcare Applications

Instructions: Provide at least five healthcare devices or machines that have embedded systems and discuss their functions.

Activity 6: Interview Time

Instructions: Interview two random people and ask them about the machine or device they envision being invented for the future that could have a significant impact. List it below.

Activity 7: Exploring Embedded Systems in Industrial Automation

Instructions: Provide a list of at least five well-known companies in the field of Industrial Automation.

Activity 8: Interview Time

Instructions: Provide ideas about the future of industrial automation in various related areas.

SUMMARY

Embedded systems play a crucial role in the automotive industry, enabling advanced functionalities in vehicle control, safety, and infotainment systems. They enhance vehicle performance, improve safety measures, and provide better driving experience for users. Real- life applications include engine control, collision detection, multimedia systems, and more.

It also plays a significant role in enhancing consumer electronic devices, enabling advanced functionalities, and improving user experiences. From smartphones and smart TVs to wearables, embedded systems provide the foundation for various features and applications in the consumer electronics industry.

On the other hand, embedded systems are proven very useful in improving patient care, monitoring, and medical device functionality. They are utilized in various healthcare devices such as patient monitoring systems, medical imaging devices, and treatment interventions. Embedded systems enable real-time data processing, wireless communication, and remote monitoring, enhancing healthcare outcomes and improving patient quality of life.

Lastly, this lesson introduces embedded systems in industrial automation, highlighting their definition, significance, and role in enhancing productivity and safety. It explores the components of embedded systems, including microcontrollers, sensors, actuators, and communication interfaces, as well as software architectures and real-time operating systems. The lesson also discusses the applications of embedded systems in industrial control systems and robotics, along with the benefits they offer, and the challenges involved in their implementation.

SUMMATIVE ASSESSMENT (SA)

1. What is the purpose of embedded systems in the automotive industry?
   a) Enhance vehicle performance
   b) Improve fuel efficiency
   c) Provide entertainment options
   d) All of the above

2. Which component is commonly used in automotive embedded systems for data acquisition?
   a) Actuators
   b) Microcontrollers
   c) Communication interfaces
   d) Sensors

3. Which system in the automotive industry relies on embedded systems for collision detection and avoidance?
   a) Adaptive cruise control
   b) Engine control unit
   c) Tire pressure monitoring system
   d) Anti-lock braking system

4. What are some applications of embedded systems in infotainment systems?
   a) Navigation systems
   b) Tire pressure monitoring
   c) Engine control
   d) Collision detection

5. True or False: Real-time operating systems are commonly used in automotive embedded systems.

6. Which of the following is an example of a consumer electronic device that relies on embedded systems?
   a) Washing machine
   b) Microwave oven
   c) Smart TV
   d) Ceiling fan

7. What is the role of embedded systems in smartphones?
   a) Managing power supply
   b) Controlling camera settings
   c) Processing touch inputs
   d) All of the above

8. Which component in wearable devices utilizes embedded systems for data processing?
   a) GPS sensor
   b) Heart rate monitor
   c) Display screen
   d) Batter

9. What is the purpose of embedded systems in automotive vehicles?
   a) Enhancing fuel efficiency
   b) Controlling safety features
   c) Providing in-car entertainment
   d) All of the above

10. Which consumer electronic device utilizes embedded systems to enable wireless connectivity and internet browsing?
    a) Digital camera
    b) Smartwatch
    c) Hairdryer
    d) Toaster oven

11. Which of the following is an example of an embedded system in healthcare applications?
    a) Smartphone
    b) Stethoscope
    c) Refrigerator
    d) Bicycle

12. What are the key components of embedded systems in healthcare devices?
    a) Microcontrollers and sensors
    b) Power generators and batteries
    c) Speakers and displays
    d) All of the above

13. What is the role of embedded systems in medical imaging devices?
    a) Processing patient data
    b) Capturing and analyzing medical images
    c) Controlling infusion pumps
    d) None of the above

14. True or False: Embedded systems in healthcare can enable remote monitoring of patient vital signs.

15. Which healthcare device utilizes an embedded system for drug delivery?
    a) Blood pressure monitor
    b) ECG machine
    c) Insulin pump
    d) Thermometer

16. What is the significance of embedded systems in industrial automation?
    a) Decreasing productivity and efficiency
    b) Ensuring employee safety
    c) Improving productivity, efficiency, and safety
    d) Reducing the need for control systems

17. Which of the following is a key component of embedded systems in industrial automation?
    a) Power supply unit
    b) Communication interface
    c) Human-machine interface (HMI)
    d) Barcode scanner

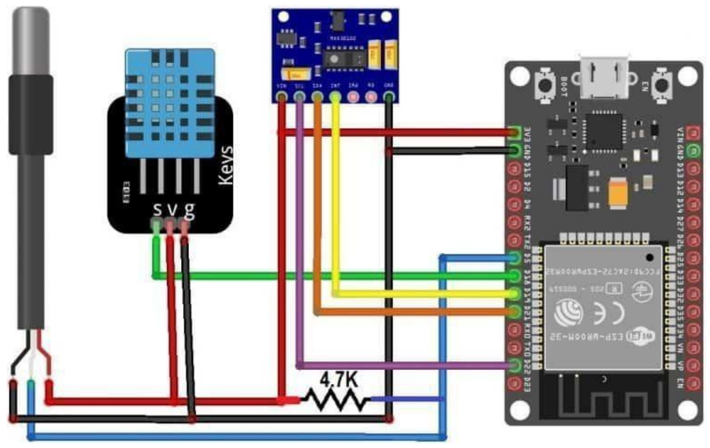
18. Embedded systems are commonly used in industrial control systems to:
    a) Monitor and control industrial processes
    b) Perform data analysis for market research
    c) Control household appliances
    d) Automate home security systems

19. How do embedded systems contribute to robotics in industrial automation?
    a) By providing physical strength to robots
    b) By enabling robots to perform emotional tasks
    c) By controlling robotic arms and implementing motion planning algorithms
    d) By generating power for robot operation

20. One challenge associated with embedded systems in industrial automation is:
    a) Reducing productivity
    b) Simplifying system integration
    c) Addressing cybersecurity concerns
    d) Eliminating the need for real-time response

LAB ACTIVITY

Creating an IOT Based Patient Health Monitoring on ESP32 Web Server



Materials needed:

- ESP3 Board
- MAX30100 Pulse Oximeter Sensor
- DS18B20 Sensor
- DHT11 Sensor
- Resistor 4.7K
- Connecting Wires
- Breadboard

Source Code:

Install the following libraries before continuing:

- Arduino MAX30100 Library
- OneWire Library
- Dallas Temperature Library
- DHT11 Library

***Note: Make changes to the WIFI SSID and Password**
(Program below will serve as a guide on doing the Lab Activity)

```
#include <WiFi.h>
#include <WebServer.h>
#include <Wire.h>
#include "MAX30100_PulseOximeter.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#include <dht.h>

#define DHT11_PIN 18
#define DS18B20 5
#define REPORTING_PERIOD_MS  1000

float temperature, humidity, BPM, SpO2, bodytemperature;
```

```
/*Put your SSID & Password*/
const char* ssid = "Alexahome"; // Enter SSID here
const char* password = "12345678"; //Enter Password here

dht DHT;
PulseOximeter pox;
uint32_t tsLastReport = 0;
OneWire oneWire(DS18B20);
DallasTemperature sensors(&oneWire);


WebServer server(80);

void onBeatDetected()
{
  Serial.println("Beat!");
}
void setup() {
  Serial.begin(115200);
  pinMode(19, OUTPUT);
  delay(100);

  Serial.println("Connecting to ");
  Serial.println(ssid);

  //connect to your local wi-fi network
  WiFi.begin(ssid, password);

  //check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: "); Serial.println(WiFi.localIP());

  server.on("/", handle_OnConnect);
  server.onNotFound(handle_NotFound)
  ;

  server.begin();
  Serial.println("HTTP server started");

  Serial.print("Initializing pulse oximeter..");

  if (!pox.begin()) {
    Serial.println("FAILED");
    for (;;);
  } else {
    Serial.println("SUCCESS");
    pox.setOnBeatDetectedCallback(onBeatDetected);
  }

   pox.setIRLedCurrent(MAX30100_LED_CURR_7_6MA);
// Register a callback for the beat detection

}
void loop() {
  server.handleClient();
  pox.update();
  sensors.requestTemperatures();
  int chk = DHT.read11(DHT11_PIN);

  temperature = DHT.temperature;
  humidity = DHT.humidity;
  BPM = pox.getHeartRate();
  SpO2 = pox.getSpO2();
  bodytemperature = sensors.getTempCByIndex(0);


  if (millis() - tsLastReport > REPORTING_PERIOD_MS)
  {
    Serial.print("Room Temperature: ");
    Serial.print(DHT.temperature);
    Serial.println("°C");

    Serial.print("Room Humidity: ");
    Serial.print(DHT.humidity);
    Serial.println("%");

    Serial.print("BPM: ");
    Serial.println(BPM);

    Serial.print("SpO2: ");
    Serial.print(SpO2);
    Serial.println("%");

    Serial.print("Body Temperature: ");
    Serial.print(bodytemperature);
    Serial.println("°C");

    Serial.println("*********************************");
    Serial.println();

    tsLastReport = millis();
  }

}

void handle_OnConnect() {

  server.send(200, "text/html", SendHTML(temperature, humidity, BPM, SpO2,
bodytemperature));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}
```

```
  String SendHTML(float temperature,float humidity,float BPM,float SpO2,
float bodytemperature){
  String ptr = "<!DOCTYPE html>";
  ptr +="<html>";
  ptr +="<head>";
  ptr +="<title>ESP32 Patient Health Monitoring</title>";
  ptr +="<meta name='viewport' content='width=device-width, initial-
scale=1.0'>";
  ptr +="<link
href='https://fonts.googleapis.com/css?family=Open+Sans:300,400,600'
rel='stylesheet'>";
  ptr +="<style>";
  ptr +="html { font-family: 'Open Sans', sans-serif; display: block; margin:
0px auto; text-align: center;color: #444444;}";
  ptr +="body{margin: 0px;} ";
  ptr +="h1 {margin: 50px auto 30px;} ";
  ptr +=".side-by-side{display: table-cell;vertical-align: middle;position:
relative;}";
  ptr +=".text{font-weight: 600;font-size: 19px;width: 200px;}";
  ptr +=".reading{font-weight: 300;font-size: 50px;padding-right: 25px;}";
  ptr +=".temperature .reading{color: #F29C1F;}";
  ptr +=".humidity .reading{color: #3B97D3;}";
  ptr +=".BPM .reading{color: #FF0000;}";
  ptr +=".SpO2 .reading{color: #955BA5;}";
  ptr +=".bodytemperature .reading{color: #F29C1F;}";
  ptr +=".superscript{font-size: 17px;font-weight: 600;position:
absolute;top: 10px;}";
  ptr +=".data{padding: 10px;}";
  ptr +=".container{display: table;margin: 0 auto;}";
  ptr +=".icon{width:65px}";
  ptr +="</style>";
  ptr +="</head>";
  ptr +="<body>";
  ptr +="<h1>ESP32 Patient Health Monitoring</h1>";
  ptr +="<h3>www.how2electronics.com</h3>";
  ptr +="<div class='container'>";

  ptr +="<div class='data temperature'>";
  ptr +="<div class='side-by-side icon'>";
  ptr +="<svg enable-background='new 0 0 19.438 54.003'height=54.003px
id=Layer_1 version=1.1 viewBox='0 0 19.438 54.003'width=19.438px x=0px
xml:space=preserve xmlns=http://www.w3.org/2000/svg
xmlns:xlink=http://www.w3.org/1999/xlink y=0px><g><path d='M11.976,8.82v-
2h4.084V6.063C16.06,2.715,13.345,0,9.996,0H9.313C5.965,0,3.252,2.715,3.252,6
.
063v30.982";
  ptr
+="C1.261,38.825,0,41.403,0,44.286c0,5.367,4.351,9.718,9.719,9.718c5.368,0,9.
719-4.351,9.719-9.718";
  ptr +="c0-2.943-1.312-5.574-3.378-7.355V18.436h-3.914v-2h3.914v-2.808h-
4.084v-2h4.084V8.82H11.976z M15.302,44.833";
  ptr +="c0,3.083-2.5,5.583-5.583,5.583s-5.583-2.5-5.583-5.583c0-2.279,1.368-
4.236,3.326-5.104V24.257C7.462,23.01,8.472,22,9.719,22";
  ptr
+="s2.257,1.01,2.257,2.257V39.73C13.934,40.597,15.302,42.554,15.302,44.833z'f
ill=#F29C21 /></g></svg>";
```

```
  ptr +="</div>";
  ptr +="<div class='side-by-side text'>Room Temperature</div>";
  ptr +="<div class='side-by-side reading'>";
  ptr +=(int)temperature;
  ptr +="<span class='superscript'>&deg;C</span></div>";
  ptr +="</div>";

  ptr +="<div class='data humidity'>";
  ptr +="<div class='side-by-side icon'>";
  ptr +="<svg enable-background='new 0 0 29.235 40.64'height=40.64px
id=Layer_1 version=1.1 viewBox='0 0 29.235 40.64'width=29.235px x=0px
xml:space=preserve xmlns=http://www.w3.org/2000/svg
xmlns:xlink=http://www.w3.org/1999/xlink y=0px><path
d='M14.618,0C14.618,0,0,17.95,0,26.022C0,34.096,6.544,40.64,14.618,40.64s14.
6 17-6.544,14.617-14.617";
  ptr +="C29.235,17.95,14.618,0,14.618,0z M13.667,37.135c-5.604,0-10.162-
4.56-10.162-10.162c0-0.787,0.638-1.426,1.426-1.426";
  ptr
+="c0.787,0,1.425,0.639,1.425,1.426c0,4.031,3.28,7.312,7.311,7.312c0.787,0,1.
425,0.638,1.425,1.425";
  ptr +="C15.093,36.497,14.455,37.135,13.667,37.135z'fill=#3C97D3 /></svg>";
  ptr +="</div>";
  ptr +="<div class='side-by-side text'>Room Humidity</div>";
  ptr +="<div class='side-by-side reading'>";
  ptr +=(int)humidity;
  ptr +="<span class='superscript'>%</span></div>";
  ptr +="</div>";

  ptr +="<div class='data Heart Rate'>";
  ptr +="<div class='side-by-side icon'>";
  ptr +="<svg enable-background='new 0 0 40.542 40.541'height=40.541px
id=Layer_1 version=1.1 viewBox='0 0 40.542 40.541'width=40.542px x=0px
xml:space=preserve xmlns=http://www.w3.org/2000/svg
xmlns:xlink=http://www.w3.org/1999/xlink y=0px><g><path d='M34.313,20.271c0-
0.552,0.447-1,1-1h5.178c-0.236-4.841-2.163-9.228-5.214-12.593l-3.425,3.424";
  ptr +="c-0.195,0.195-0.451,0.293-0.707,0.293s-0.512-0.098-0.707-0.293c-
0.391-0.391-0.391-1.023,0-1.414l3.425-3.424";
  ptr +="c-3.375-3.059-7.776-4.987-12.634-
5.215c0.015,0.067,0.041,0.13,0.041,0.202v4.687c0,0.552-0.447,1-1,1s-1-0.448-
1-1V0.25";
  ptr +="c0-0.071,0.026-0.134,0.041-
0.202C14.39,0.279,9.936,2.256,6.544,5.385l3.576,3.577c0.391,0.391,0.391,1.024
,0,1.414";
  ptr +="c-0.195,0.195-0.451,0.293-0.707,0.293s-0.512-0.098-0.707-
0.293L5.142,6.812c-2.98,3.348-4.858,7.682-
  5.092,12.459h4.804"; ptr +="c0.552,0,1,0.448,1,1s-0.448,1-
1,1H0.05c0.525,10.728,9.362,19.271,20.22,19.271c10.857,0,19.696-8.543,20.22-
19.271h-5.178";
  ptr +="C34.76,21.271,34.313,20.823,34.313,20.271z M23.084,22.037c-
0.559,1.561-2.274,2.372-3.833,1.814";
  ptr +="c-1.561-0.557-2.373-2.272-1.815-3.833c0.372-1.041,1.263-1.737,2.277-
1.928L25.2,7.202L22.497,19.05";
  ptr +="C23.196,19.843,23.464,20.973,23.084,22.037z'fill=#26B999
/></g></svg>";
  ptr
  +="</div>";
  ptr +="<div class='side-by-side text'>Heart Rate</div>";
  ptr +="<div class='side-by-side reading'>";
  ptr +=(int)BPM;
  ptr +="<span class='superscript'>BPM</span></div>";
  ptr +="</div>";

  ptr +="<div class='data Blood Oxygen'>";
  ptr +="<div class='side-by-side icon'>";
  ptr +="<svg enable-background='new 0 0 58.422 40.639'height=40.639px
id=Layer_1 version=1.1 viewBox='0 0 58.422 40.639'width=58.422px x=0px
xml:space=preserve xmlns=http://www.w3.org/2000/svg
xmlns:xlink=http://www.w3.org/1999/xlink y=0px><g><path
d='M58.203,37.754l0.007-0.004L42.09,9.935l-0.001,0.001c-0.356-0.543-0.969-
0.902-1.667-0.902";
  ptr +="c-0.655,0-1.231,0.32-1.595,0.808l-0.011-0.007l-0.039,0.067c-
0.021,0.03-0.035,0.063-0.054,0.094L22.78,37.692l0.008,0.004";
  ptr +="c-0.149,0.28-0.242,0.594-
0.242,0.934c0,1.102,0.894,1.995,1.994,1.995v0.015h31.888c1.101,0,1.994-
0.893,1.994-1.994";
  ptr +="C58.422,38.323,58.339,38.024,58.203,37.754z'fill=#955BA5 /><path
d='M19.704,38.674l-0.013-0.004l13.544-23.522L25.13,1.156l-
0.002,0.001C24.671,0.459,23.885,0,22.985,0";
  ptr +="c-0.84,0-1.582,0.41-2.051,1.038l-0.016-0.01L20.87,1.114c-
0.025,0.039-0.046,0.082-0.068,0.124L0.299,36.851l0.013,0.004";
  ptr
+="C0.117,37.215,0,37.62,0,38.059c0,1.412,1.147,2.565,2.565,2.565v0.015h16.9
8 9c-0.091-0.256-0.149-0.526-0.149-0.813";
  ptr +="C19.405,39.407,19.518,39.019,19.704,38.674z'fill=#955BA5
/></g></svg>";
  ptr
  +="</div>";
  ptr +="<div class='side-by-side text'>Blood Oxygen</div>";
  ptr +="<div class='side-by-side reading'>";
  ptr +=(int)SpO2;
  ptr +="<span class='superscript'>%</span></div>";
  ptr +="</div>";

  ptr +="<div class='data Body Temperature'>";
  ptr +="<div class='side-by-side icon'>";
  ptr +="<svg enable-background='new 0 0 19.438 54.003'height=54.003px
id=Layer_1 version=1.1 viewBox='0 0 19.438 54.003'width=19.438px x=0px
xml:space=preserve xmlns=http://www.w3.org/2000/svg
xmlns:xlink=http://www.w3.org/1999/xlink y=0px><g><path d='M11.976,8.82v-
2h4.084V6.063C16.06,2.715,13.345,0,9.996,0H9.313C5.965,0,3.252,2.715,3.252,6
.
063v30.982";
  ptr
+="C1.261,38.825,0,41.403,0,44.286c0,5.367,4.351,9.718,9.719,9.718c5.368,0,9
. 719-4.351,9.719-9.718";
  ptr +="c0-2.943-1.312-5.574-3.378-7.355V18.436h-3.914v-2h3.914v-2.808h-
4.084v-2h4.084V8.82H11.976z M15.302,44.833";
  ptr +="c0,3.083-2.5,5.583-5.583,5.583s-5.583-2.5-5.583-5.583c0-2.279,1.368-
4.236,3.326-5.104V24.257C7.462,23.01,8.472,22,9.719,22";
  ptr
+="s2.257,1.01,2.257,2.257V39.73C13.934,40.597,15.302,42.554,15.302,44.833z'f
ill=#F29C21 /></g></svg>";
  ptr +="</div>";
```

```
      ptr +="<div class='side-by-side text'>Body Temperature</div>";
  ptr +="<div class='side-by-side reading'>";
        ptr +=(int)bodytemperature;
      ptr +="<span class='superscript'>&deg;C</span></div>";
  ptr +="</div>";

      ptr
  +="</div>"; ptr
  +="</body>"; ptr
  +="</html>";
  return ptr;
```

REFERENCE / BIBLIOGRAPHY

Admin. (2022, June 6). *Embedded Systems Role in Automobiles - Working with Applications*. WatElectronics.com. https://www.watelectronics.com/importance-of-embedded-systems-in-automobiles-with-applications/

Admin. (2023). IoT Based Patient Health Monitoring on ESP32 Web Server. How to Electronics. https://how2electronics.com/iot-based-patient-health-monitoring-esp32-web-server/#IoT_Based_Patient_Health_Monitoring_on_ESP32_Web_Server

Admin, & Admin. (2023, June 23). Embedded Systems in Industrial Automation: Key Insights | Embedded Systems Manufacturing Companies in USA. Avench - emphasis on embedded -. https://avench.com/blogs/embedded-systems-in-industrial-automation/#:~:text=Embedded%20systems%20help%20to%20automate,%2C%20manufacturing%2C%20and%20many%20others.

Attwood, D. (2023, June 7). Medical device technology and embedded solutions for the medical device industry. Medical Device Network. https://www.medicaldevice-network.com/buyers-guide/embedded-technology/

Das, S. (2021). Consumer Electronics – Definition, List of Companies. *Electronics Tutorial | Best Electronics Tutorial Website*. https://www.electronicsandyou.com/consumer-electronics-definition-list-of-companies.html

D, H. (n.d.). *A Validation Overview of ECU Communication Protocols in Automotive Audio Management - Blog*. https://www.ltts.com/blog/automotive-audio-management#:~:text=ADAS,A%20Validation%20Overview%20of%20ECU%20Communication%20Protocols%20in%20Automotive%20Audio,subsystems)%20in%20a%20motor%20vehicle.

E Control Devices. (2023, January 19). *Consumer Electronics - E Control Devices*. https://econtroldevices.com/consumer-electronics/

Embedded Systems For Industrial Automation and Control. (2021, April 7). Premio Inc. https://premioinc.com/blogs/blog/embedded-system-for-industrial-automation-and-control

How Medical Embedded Systems Transformed the Healthcare Industry. (2023, April 18). Total Phase Blog. https://www.totalphase.com/blog/2019/08/how-medical-embedded-systems-transformed-the-healthcare-industry/

McFarlane, S. (2022, February 16). *Use Of Embedded Systems in Industrial Automation - Viewpoint Systems*. Viewpoint Systems. https://www.viewpointusa.com/IE/ar/use-of-embedded-systems-in-industrial-automation/

Raymond, A. (2023). What You Need to Know About Embedded Systems in Medical Applications. Orthogone. https://orthogone.com/embedded-systems-in-medical-applications/

Yash Technologies. (2022, May 20). The Future of Industrial Automation with Embedded Systems. https://www.yash.com/blog/the-future-of-industrial-automation-with-embedded-systems/

Emerging Trends in Embedded Systems

INTRODUCTION

Embedded systems play a crucial role in our modern society, fueling an array of devices and systems that are integral to our daily lives. As technology progresses at an unprecedented pace, the field of embedded systems is also rapidly expanding, giving birth to fresh and captivating concepts. These emerging trends in embedded systems are reshaping the way we engage with technology, transforming industries, and creating new opportunities for innovation.

This module covers the emerging trends in embedded systems such as the Internet of Things (IoT), edge computing, artificial intelligence (AI), embedded vision, image processing, and wearable and ubiquitous computing.

LEARNING OBJECTIVES / OUTCOMES

At the end of the module, students should be able to:

- gain knowledge about the emerging trends in embedded systems;
- understand how embedded systems contribute to Internet of Things (IoT), edge computing, artificial intelligence (AI), embedded vision, image processing, and wearable ubiquitous computing; and
- experience how to make a simple IoT program through Arduino IoT Cloud.

DISCUSSION OF THE LESSON

Internet of Things (IoT) and Edge Computing

The Internet of Things (IoT) comprises interconnected physical, digital, mechanical, and computing devices with unique identifiers (UIDs), enabling their internet-based interactions. Equipped with sensors, these devices gather data, which is then transmitted to the cloud through an IoT gateway using protocols like HTTP and MQTT. Analytical tools in the cloud process the data, extracting valuable insights, which are subsequently delivered to end-users via an API.

Every year, the number of connected devices worldwide continues to grow exponentially. In response to this trend, the Arduino IoT Cloud offers a comprehensive platform that empowers individuals to develop their own IoT projects. With its user-friendly interface, the Arduino IoT Cloud provides a convenient all-in-one solution for configuration, code writing, uploading, and data visualization, making it accessible to a wide range of users.

The increasing adoption of IoT has emerged as a significant catalyst for the growth of edge computing. With the proliferation of connected IoT devices, there is a substantial surge in data generation. However, transmitting all this data to the distant cloud for processing poses challenges in terms of cost and latency.

What is edge computing? the quick overview explained with examples. SPEC INDIA. (n.d.). https://www.spec- india.com/blog/what-is-edge-computing-the-quick- overview-explained-with-examples

Edge computing addresses these issues by facilitating data processing at the network's edge, closer to the source (sensor devices) itself. This approach becomes crucial for time-sensitive data and situations requiring instantaneous decision-making. By performing advanced analytics at the edge, devices offer real-time predictions and solutions, enabling organizations to promptly derive valuable insights.

In addition to the benefits of edge computing in terms of cost and latency, it also addresses concerns related to data privacy and security. With edge computing, sensitive data can be processed locally, reducing the need for transmitting it over the network and minimizing the risk of data breaches. This is particularly important in scenarios where data confidentiality is critical, such as healthcare or industrial applications.

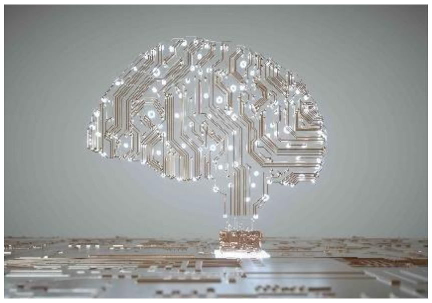
Furthermore, edge computing enhances overall system resilience and reliability. By distributing the computing load across edge devices, it reduces the dependency on a centralized cloud infrastructure. Even if the connection to the cloud is disrupted, edge devices can continue to operate autonomously and make local decisions based on the available data. This decentralized approach increases system robustness and minimizes the impact of network failures.

The combination of IoT and edge computing also enables real-time automation and control. By leveraging the power of edge devices, IoT applications can respond rapidly to changing conditions without relying on constant communication with the cloud. This is particularly beneficial in cases where immediate action is required, such as autonomous vehicles, smart grid management, or predictive maintenance in industrial settings.

Moreover, edge computing allows for efficient data filtering and aggregation. Instead of sending raw data to the cloud, edge devices can perform local preprocessing, filtering out irrelevant or redundant information and transmitting only the essential insights. This reduces the bandwidth requirements and optimizes network utilization, leading to cost savings and improved overall system efficiency.

Artificial Intelligence (AI) in Embedded Systems

Artificial intelligence (AI) in embedded systems refers to the incorporation of AI technologies and algorithms into embedded systems, which are computer systems designed to perform specialized duties within larger systems or devices. However, these embedded systems are frequently distinguished by their restricted processing capabilities, power limits, and real-time operation requirements.


Frankenfield, J. (2023, May 1). Artificial Intelligence: What it is and how it is used. Investopedia. https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp

This integration will open a path for new, low-cost, and low-powered AI solutions. As the name suggests, and from the nature of embedded systems, the devices that will be developed using AI are only tasked with only a single function since the microcontrollers are used to build an embedded system
have the limitations and training a model will need a lot of time and system storage to process.

Moreover, knowledge and abilities beyond standard embedded systems, data science, and machine learning (ML) are required for embedded AI. It needs familiarity with the variety of devices, sensors, and advanced, near real-time signal processing algorithms for video, audio, motion, and other signals. Developing embedded AI applications requires the use of specialized software tools and frameworks.

In addition, AI in embedded systems allows these devices to perform tasks that require repetitive and intensive actions such as data processing, decision-making, recognizing patterns, and adaptive learning. By adding AI capabilities, embedded systems can improve their efficiency in different industries such as businesses, healthcare, finance, agriculture, manufacturing, etc.

The following are some of the key aspects and applications of artificial intelligence in embedded systems:

1. ***Edge AI:*** allows machine learning activities to be performed directly on end devices. It often has an in-built CPU and sensors, with data processing performed locally and saved at the edge node end. Machine learning models implemented in edge AI will reduce latency and increase network bandwidth.

2. ***Edge Machine Learning:*** Machine learning is a branch of artificial intelligence (AI) in which the AI can execute perceptive tasks in a fraction of the time that a person would take. Edge computing, on the other hand, refers to the process of physically putting computer services closer to either the user or the source of the data. These computing services are available on edge devices, which are computers that gather and analyze raw data in real-time, resulting in quicker, more reliable analysis. Edge machine learning is the practice of executing machine learning (ML) models on an edge device to gather, analyze, and detect patterns in raw data sets.

   Edge computing allows you to physically bring artificial intelligence/machine learning (AI/ML)-powered applications closer to data sources such as sensors, cameras, and mobile devices to gather insights quicker, discover trends, and start actions without depending on traditional cloud networks.

3. **Intelligent Control Systems:** In embedded systems, AI algorithms provide intelligent control and decision-making. They can adjust and optimize system behavior in response to changing situations, learn from feedback, and continuously enhance performance. This is especially beneficial in robots, autonomous cars, and smart homes, where embedded systems must read complicated inputs and respond appropriately.

4. **Real-time Data Processing:** AI-enabled embedded systems can analyze and analyze data in real-time, deriving useful insights and performing necessary actions. In industrial automation, for example, embedded systems may monitor sensor data, identify abnormalities, and initiate quick responses to enhance operations or avoid failures.

In general, adding AI into embedded systems gives these devices superior cognitive capacities, allowing them to make intelligent judgments, handle complicated data, and adapt to changing settings. This trend toward embedded AI opens up enormous opportunities for creative applications, increased productivity, and improved user experiences across a wide range of sectors.

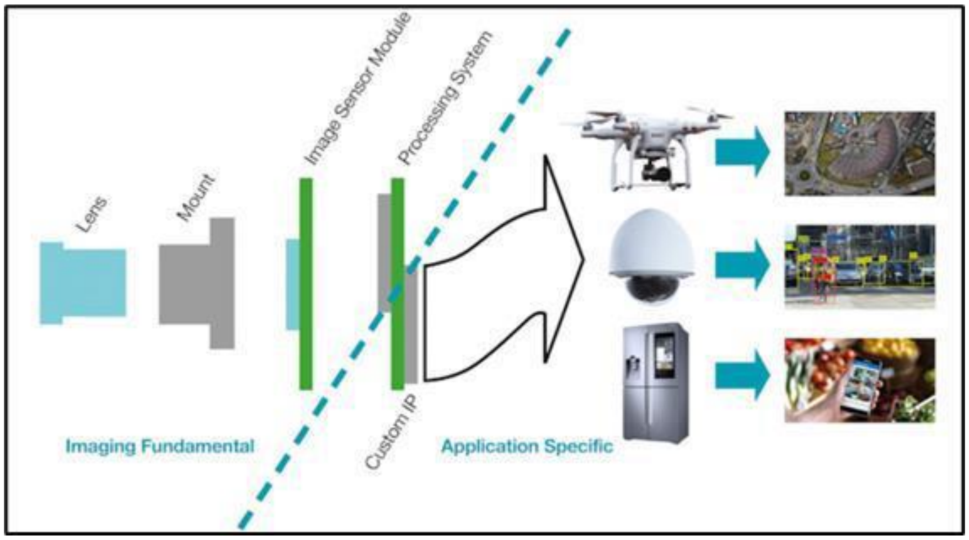
Embedded Vision and Image Processing

A growing number of "intelligent" devices and machines are placing new demands on embedded image functionalities and systems. Whether it's driverless cars, intelligent vacuum cleaners or robots interacting as colleagues in industrial production – only native components of fully integrated vision systems make them effective. Embedded vision, as an integrated technology in devices and machines, ensures that all industries and almost any application will benefit from imaging.

Embedded vision gives visual intelligence to machines and teaches them how to see and think. Embedded vision in a device defines the integration of image processing with digital processing and intelligent algorithms to evaluate an acquired image, and the associated video data. In this way, a seeing and thinking machine can react to events and processes and interact with the environment. Integrated image processing is the basis for safe, man-machine collaboration, and the networked activities of robots and machines in the automated Smart Factory.

Regular vision systems are mainly built with a camera that is connected to a host PC with a known data interface. The system is mostly separated into the machine that runs and the controlling process that does the inspection. The processing of the video stream and images are mostly outsourced and often needs user interaction for validation and/or decision making.

Embedded Vision is not only part of the device, it is the 'smart eye'. In its entirety Embedded Vision minimizes or removes human interactions within the imaging pipeline and allows machines to make their own decisions by capturing, analyzing and interpreting the data all-in-one.



FRAMOS. (n.d.). How embedded vision works. https://www.framos.com/en/resources/what-is-embedded-vision

Therefore, the main difference between Classic Vision and Embedded Vision are as follows:

Regular Vision	Embedded Vision
• separated system for capturing image streams and processing • outsourced analysis for decisions • made to support many possible tasks	• fully integrated all-in-one system capturing image streams • on-board processing and interpreting data • autonomous decision making, resulting in an action • made for a specific task

Image processing is the process of transforming an image into a digital form and performing certain operations to get some useful information from it. The image processing system usually treats all images as 2D signals when applying certain predetermined signal processing methods.

An image is represented by its dimensions (height and width) based on the number of pixels. For example, if the dimensions of an image are 500 x 400 (width x height), the total number of pixels in the image is 200000.

This pixel is a point on the image that takes on a specific shade, opacity or color. It is usually represented in one of the following:

- Grayscale - A pixel is an integer with a value between 0 to 255 (0 is completely black and 255 is completely white).
- RGB - A pixel is made up of 3 integers between 0 to 255 (the integers represent the intensity of red, green, and blue).

- RGBA - It is an extension of RGB with an added alpha field, which represents the opacity of the image.

Image processing requires fixed sequences of operations that are performed at each pixel of an image. The image processor performs the first sequence of operations on the image, pixel by pixel. Once this is fully done, it will begin to perform the second operation, and so on. The output value of these operations can be computed at any pixel of the image.

The implementation of image processing techniques has had a massive impact on many tech organizations. Here are some of the most useful benefits of image processing, regardless of the field of operation:

- The digital image can be made available in any desired format (improved image, X-Ray, photo negative, etc.).
- It helps to improve images for human interpretation.
- Information can be processed and extracted from images for machine interpretation.
- The pixels in the image can be manipulated to any desired density and contrast.
- Images can be stored and retrieved easily.
- It allows for easy electronic transmission of images to third-party providers.

Embedded image processing can be the ideal solution for specific applications. Embedded applications can be developed for Single Board Computers (SBCs) featuring GigE or USB connections to the camera. This hardware is popular, available in many prices, performance and quality ranges and suitable for a variety of application cases.

LVDS and CSI2-based Camera-to-System on Chip (SoC) connections offer a direct interface for image data transfer, but without the benefit of a comprehensive industry standard. As a result, both methods involve adjustment and integration time and costs when developing simple integration solutions. Companies offer drivers, development kits and other tools to help make integration easier for users.

In the future, the development and roll-out of standards would be a welcome step towards simpler integration of cameras involving a direct connection to a System on Chip. Generic drivers and standardized data APIs can also make it easier to develop specialized image processing solutions. As such, the integration of image processing technology can be made as simple as the prevalent PC-based industrial image processing, even for the smallest and leanest embedded systems.

Wearable and Ubiquitous Computing

Wearable computing refers to the utilization of a small computer or sensor device that is worn on or integrated into clothing. The main goal of current wearable computing design is to establish ongoing communication between the user and the computer. This involves computer "learning" and understanding the user's real-time experiences and enhancing them with supplementary information.

Since the nature of wearable computers is to be compact and working as it intends to function, the following are the characteristics they should possess:

- attractive visual design,
- enhance the user's physical or cognitive capabilities,
- offer cost-effective advantages,
- user-friendly, allowing consumers to quickly set up and operate them,
- can be personalized and adjusted to meet the user's requirements, providing feedback, and
- can be easily integrated with the user's attire, body, or connected to a smartphone.

What is a wearable computer? definition and examples. Market Business News. (2021, October 27). https://marketbusinessnews.com/financial-glossary/wearable-computer/

As for the market situation of these computers, we can notice the rise of smart watches wherein they function as health trackers for their users. Among all data they track, the most common one they monitor is the wearer’s pulse rate. According to Bocas (2022), wearable technology will dominate the future since a lot of technology-related companies are getting invested in this trend. In his blog, he discusses the top five (5) wearable computer trends in 2022 namely smart hearables, smart watches, smart patches, smart clothing, and smart implantable.

- **Smart hearables** are gaining significant popularity as a trending category of wearable devices. These innovative gadgets integrate the functionalities of conventional headphones with advanced sensors and artificial intelligence (AI), delivering a highly personalized listening encounter. Key industry players, such as Apple, Amazon, Google, and Bose, are actively involved in shaping this emerging field.
- **Smart watches** have emerged as a highly sought-after genre of wearable devices. These gadgets encompass the functionality of a conventional watch while incorporating a range of additional features like activity monitoring, smartphone alerts, and contactless payments. Prominent brands in this domain consist of Apple Watch, and Samsung Gear.
- **Smart patches** refer to wearable devices designed to be affixed to the skin, serving the purpose of monitoring different health indicators, or administering medication. These devices are typically thin, flexible, and provide extended comfort during prolonged usage. They are also commonly incorporated in multiple sensors capable of tracking physiological parameters like heart rate, body temperature, and blood oxygen levels. These sensors can also detect variations in these metrics over time, promptly notifying users of any abnormal changes occurring within their bodies.
- **Smart clothing**, alternatively referred to as wearable technology, encompasses garments that incorporate integrated electronics and sensors seamlessly woven into the fabric. These electronics are typically powered by batteries or small generators that convert body heat into electricity. The embedded sensors enable the tracking of diverse biometric information, including heart rate.
- **Smart implantable** refer to devices that are surgically implanted within the body and interact with the body's tissues and organs for the purpose of monitoring, diagnosing, or treating medical conditions. They can be categorized as either passive or active. Passive smart implantable operates without electronics and rely on the body's own energy for functionality. In contrast, active smart implantable incorporate electronics and rely on battery power for operation.

In conclusion, the world of wearable technology is rapidly evolving, offering a diverse range of innovative devices designed to enhance various aspects of our lives. Smart hearables, combining the features of traditional headphones with advanced sensors and AI, provide a personalized listening experience. Smart watches, offering traditional timekeeping functionality alongside features like activity tracking and smartphone integration, have become popular accessories. Smart patches, thin and flexible wearable devices, allow for comfortable long-term monitoring of vital signs and detection of abnormalities. Smart clothing integrates electronics and sensors into fabric, enabling the tracking of biometric data directly from garments. Finally, smart implantable, surgically implanted devices, interact with the body's tissues and organs to monitor, diagnose, or treat medical conditions. As technology continues to advance, these wearable devices are poised to play an increasingly significant role in improving our health, enhancing our daily lives, and shaping the future of wearable technology.

On the other hand, **ubiquitous computing**, or ubicomp, seamlessly integrates computing power into everyday objects, enabling effortless communication and tasks with minimal user interaction. It aims to create intelligent, interconnected items for convenience and unobtrusive data exchange. In comparison, wearable computing is a subset of ubiquitous computing that focuses on incorporating computing devices into items worn on the body such as user-centric applications with portable devices like smartwatches and augmented reality glasses. Ubiquitous computing emphasizes human involvement, low-cost processors, real-time capabilities, full connectivity, and many-to-many relationships. It considers various contexts, utilizes internet convergence and wireless tech, and addresses privacy and equipment reliability. Overall, it aims to integrate technology seamlessly into daily life, enhancing communication and interactions while prioritizing the human element.

The fundamental goal of ubiquitous computing is to develop intelligent, interconnected items that facilitate seamless communication and unobtrusive data exchange. To achieve this, several important characteristics define ubiquitous computing:

1. **Consideration of the Human Element:** Ubiquitous computing places a strong emphasis on incorporating technology in settings where human involvement is central, focusing on enhancing human experiences rather than merely facilitating computer operations.
2. **Utilization of Low-Cost Processors:** By leveraging low-cost processors, ubiquitous computing optimizes memory and storage requirements, enabling cost-effective and efficient implementations
3. **Real-Time Capabilities:** Ubiquitous computing systems capture and process data in real-time, allowing for immediate and responsive interactions.
4. **Full Connectivity and Availability:** Computers in a ubiquitous computing environment are fully linked and always available, ensuring continuous connectivity and access to services.
5. **Many-to-Many Relationships:** Rather than limited one-to-one, many-to-one, or one-to-many interactions, ubiquitous computing emphasizes the interconnectivity of multiple devices, creating a complex web of communication possibilities.
6. **Consideration of Various Contexts:** Ubiquitous computing encompasses diverse contexts, encompassing local/global, social/personal, public/private, and both visible and

invisible interactions. It also takes into account the generation and transmission of knowledge.

7. **Utilization of Internet Convergence and Wireless Technologies:** The integration of ubiquitous computing relies on the convergence of the internet and wireless technologies, facilitating seamless communication and data exchange across various devices.
8. **Privacy Considerations:** The extensive use of interconnected devices raises concerns about increased monitoring, potential limitations on user privacy, and the need to address potential security threats.
9. **Equipment Reliability:** The successful implementation of ubiquitous computing depends on the reliability of the various components and devices used in the interconnected network.

Overall, ubiquitous computing seeks to create an environment where technology is seamlessly integrated into daily life, making communication and interactions more convenient and efficient while considering the human element and context in which it operates.

SUMMARY

The module explores IoT, edge computing, AI, embedded vision, image processing, wearable, and ubiquitous computing trends reshaping technology. An example application is a smart home equipped with IoT sensors, AI-driven edge computing, embedded vision cameras, and image processing showcases the integration of emerging trends in embedded systems. Users can control the environment through wearable devices, creating an efficient and secure living space with minimal interaction.

SUMMATIVE ASSESSMENT (SA)

To assess your full understanding of the module, you are required to take this summative assessment. The assessment is solely based on “Module #8: Emerging Trends in Embedded Systems.”

I. **Identification.** Direction: Read the questions carefully and identify the terms needed by item. Write your answers on the underline provided at the end of each item. Strictly no abbreviations.

1. Through this, sensitive data can be processed locally, reducing the need for transmitting it over the network and lowering the risks in data breaches.
2. It is the “Smart Eye.” _________________
3. This platform for IoT projects provides a convenient all-in-one solution for configuration, code writing, uploading, and data visualization, making it accessible for its users. _________________
4. It is a technology that mimics the way humans think. _________________

5. Few of its capabilities are monitoring sensor data, identifying abnormalities, and initiating quick responses to enhance operations or avoid failures.
6. A pixel is made up of 3 integers between 0 to 255. ________________
7. These devices are known and seen to be thin, flexible, and can provide extended comfort during prolonged usage. _________
8. It emphasizes human involvement, low-cost processors, real-time capabilities, full connectivity, and many-to-many relationships. ______________
9. Considered as an evolving technology since they are compact in nature that can aid in enhancing the lifestyle of its user. ________________
10. The procedure of converting an image into a digital format and executing specific operations to extract valuable information from it. ________________

II. **Essay.** Direction: Write an essay about your take on the emerging trends in embedded systems and how it can shape our future in terms of technological advancements. Plagiarism and the use of intelligent applications or websites are strictly prohibited.

Format:

a.) **Essay Title.**

b.) **Thesis Statement (Introduction).** The student should affirm his or her take on the topic on the introduction part by stating his or her thesis statement. Note that this part should be a statement.

c.) **Body.** After stating his or her thesis statement, he or she should provide evidence or references to support the thesis statement. He or she should cite the references used on the last page of his or her essay.

d.) **Conclusion.** The student should be able to end his or her essay by connecting the conclusion with what he or she asserted from the thesis statement (introduction).

e.) The essay should be typewritten in *Arial, single spacing, font size is 12, justified, and be saved as a pdf file.*

REFERENCE / BIBLIOGRAPHY

Bocas, J. (2022, November 16). *Top 5 wearables trends 2022 - 2023*. Digital Salutem. https://digitalsalutem.com/top-5-wearables-trends/#:~:text=Top%205%20Wearables%20Trends%3A%20Conclusion, promising%20se gments%20in%20this%20market

FRAMOS. (n.d.). *What is Embedded Vision?* Retrieved from https://www.framos.com/en/resources/what-is-embedded-vision

Getting started with the Arduino IOT Cloud. Arduino Documentation. (n.d.). https://docs.arduino.cc/arduino-cloud/getting-started/iot-cloud-getting-started

Integrating Image Processing into Embedded Systems. (2017, January 18). Retrieved from https://www.wileyindustrynews.com/en/news/integrating-image-processing-embedded-systems#:~:text=Embedded%20image%20processing%20can%20be,a%20variety%20of%20application%20cases.

Media, O. (n.d.). *Applications and benefits of Edge Ai*. Embedded Computing Design. https://embeddedcomputing.com/technology/ai-machine-learning/applications-and-benefits-of-edge-ai

Office of the Privacy Commissioner of Canada. (2014, July 3). *Wearable computing - challenges and opportunities for privacy protection*. Office of the Privacy Commissioner of Canada. https://www.priv.gc.ca/en/opc-actions-and-decisions/research/explore-privacy-research/2014/wc_201401/#heading-006-1

Simplilearn. (2023). *What is image processing: overview, applications, benefits, and more*. Retrieved from https://www.simplilearn.com/image-processing-article#:~:text=Image%20processing%20is%20the%20process,certain%20predetermined%20signal%20processing%20methods.

Thank ubiquitous computing for Making your life "smart." Diseconomy. (2022, August 23). https://dataconomy.com/2022/08/23/ubiquitous-computing-pervasive-computing/

Tripathy, S. (2022, August 17). *IOT vs. edge computing*. Enterprise Networking Planet. https://www.enterprisenetworkingplanet.com/data-center/iot-vs-edge-computing/#:~:text=In%20edge%20computing%2C%20the%20data,devices%2C%20this%20feature%20is%20optional.

What is Edge Machine Learning? Red Hat - We make open-source technologies for the enterprise. (n.d.). https://www.redhat.com/en/topics/edge-computing/what-is-edge-machine-learning

ANSWER KEY TO SUMMATIVE ASSESSMENT

Module #1: Introduction to Embedded Systems

1. True
2. a. Healthcare and Medical Industry
3. b. To store program code
4. c. Counting the number of times a particular event occurs
5. a. Power consumption
6. c. Construction and Building Management Industry
7. b. Microcontroller
8. d. All of the above
9. b. Household appliances
10. d. Manufacturing and Industrial Automation Industry

Module #2: Hardware Fundamentals for Embedded Systems

1. c. A small computer with memory and the ability to perform calculations through programming.
2. b. Microprocessor
3. a. EPROM
4. d. Vacuum
5. a. Ultrasonic Sensor
6. Superscalar Computer
7. Environmental
8. Controller
9. External
10. RAM Size

Module #3: Embedded System Software Development

1. A
2. B
3. B
4. D
5. A
6. B
7. B
8. A
9. D
10. C

Module #4: Interfacing and Communication in Embedded Systems

1. c. It selects the slave device for communication.
2. d. By setting the read/write bit in the address frame.
3. a. To convert analog signals to digital signals
4. a. Analog-to-digital converter (ADC)
5. b. To improve the accuracy of sensor measurements
6. These are sets of rules and standards that govern the transmission of data between devices without the need for physical connections. These protocols define how devices communicate, exchange information, and establish reliable connections in wireless networks. They ensure compatibility and interoperability between different devices and enable seamless data transmission over various wireless technologies.
7. Wi-Fi, Bluetooth/BLE, Zigbee.
8. Security, Interoperability, Scalability, Power Consumption, Network Coverage

Module #5: Power Management and Optimization in Embedded Systems

Battery-powered Systems 1. Devices or systems that function primarily on batteries are referred to as battery-powered embedded systems.

2-4. Give three examples of battery-powered systems.

- **Portable Electronics**
- **Wearable devices**
- **Portable medical devices**
- **Remote Monitoring Systems**
- **IoT devices**
- **Remote Controlled Systems**
- **Electric vehicles**

5-7. Give three energy harvesting techniques.

- **Solar Energy Harvesting**
- **Mechanical Energy**
- **Thermal Energy**
- **Radio Frequency Energy**
- **Kinetic Energy**
- **Ambient Light Energy**

On and Off 8. What are the two states of a device’s power consumption.

9. Give an action a software can do to minimize power consumption.

- **Turn off peripheral devices while not in use.**
- **Change the CPU's frequency and voltage in accordance with the performance demands of the moment (also known as "Dynamic Voltage and Frequency Scaling" or DVFS).**

Code Optimization 10. What is the method in which it increases the software's effectiveness in embedded systems, which boosts performance and lowers power consumption.

Module #6: Power Management and Optimization in Embedded Systems

1. a) Limited computational resources
2. d) Physical access
3. c) To prevent unauthorized or malicious code execution
4. c) To patch security vulnerabilities and add new features
5. a) RSA
6. c) It provides secure communication channels
7. c) It verifies the identity and integrity of communicating parties
8. a) TLS
9. b) To provide secure communication at the IP layer
10. a) By using encryption and key exchange mechanisms

Module #7: Case Studies and Practical Examples

1. d) All of the above
2. d) Sensors
3. a) Adaptive cruise control
4. a) Navigation systems
5. True
6. c) Smart TV
7. d) All of the above
8. b) Heart rate monitor
9. d) All of the above
10. b) Smartwatch
11. b) Stethoscope
12. a) Microcontrollers and sensors
13. b) Capturing and analyzing medical images
14. True
15. c) Insulin pump
16. c.) Improving productivity, efficiency, and safety
17. b.) Communication interface
18. a.) Monitor and control industrial processes
19. c.) By controlling robotic arms and implementing motion planning algorithms
20. c.) Addressing cybersecurity concerns

Module #8: Emerging Trends in Embedded Systems

I. Identification Answer Key

1. Edge Computing
2. Embedded Vision
3. Arduino IoT Cloud
4. Artificial Intelligence
5. Embedded System
6. RGB
7. Smart Patches
8. Ubiquitous Computing
9. Wearable Computing
10. Image Processing

II. Essay Rubrics

- **Content and Focus:** 40%
- **Structure and Organization:** 20%
- **Language and Style:** 15%
- **Research and Use of Sources:** 15%
- **Overall Presentation:** 10%

Total: 100%

ANSWER KEY TO LAB ACTIVITIES ☑

Module #2: Hardware Fundamentals for Embedded Systems

I. Lab1.ino

```
#include <ESP32Servo.h> // include file for servo control

ESP32PWM servo1; // create servo object to control servo #1

float t0 = 0.0; // need to make t0 a global variable so loop and setup can share it
void setup() {
  servo1.attach(5); // connect GPIO 5 to servo #1

  // initialize serial communication at 115200 bits per second
  Serial.begin(115200);

  // measure initial time in s
  // make sure to measure t0 last in setup since the other parts
  // of setup will take time to perform.
  t0 = micros() * 1.0e-6;
}

void loop() {
  // note this program could also be put in setup() after Serial.begin

  // note this loop function gets executed repeatedly unlike Q1
  // -- see if you can spot the difference

  float t; // clock time in s
  float A, w, phi1;
  int theta1_d; // desired angle for servo #1

  A = 45.0;
  w = 1.0; // higher values of w will make the servos oscillate faster
  phi1 = 0.0;

  // calculate time t since the program begins
  // note: since micros() gives the time since the program (i.e.) setup begins
  // calculating t = micros()*1.0e-6 will be close to t = micros()*1.0e-6 - t0
  // but not as accurate since t0 accounts for the time taken to execute
  setup() t = micros() * 1.0e-6 - t0;

  theta1_d = A * (1.0 + sin(w * t + phi1));

  servo1.write(theta1_d); // command servo #1 to go to theta1_d

  // note we don't have to wait/delay for this since the desired angles
  // are changing continuously and slowly as functions of time

  if (t > 60.0) {
    break; // end the loop at t=60s
  }
}
```

II. Lab2.ino

```
#include <ESP32Servo.h> // include file for servo control
ESP32PWM servo1; // create servo object to control servo #1
void setup() {
  servo1.attach(5); // connect GPIO 5 to servo #1

  // initialize serial communication at 115200 bits per second
  Serial.begin(115200);
}

// first method -- a sweep over all angles
void loop() {
  int theta1_d; // desired servo angle
  int analog_input0; // analog input for pin A0
  float voltage0; // voltage input for pin A0
  float min;
  int theta1_d_min; // servo angle for minimum voltage / maximum light
  float t;

  // move to the initial position of 0
  deg theta1_d = 0;
  servo1.write(theta1_d);
  delay(1000); // wait for the servo to get to theta1 = 0 deg

  // move servo #1 between 0 and 180 deg in 1 deg increments and
  // find the minimum voltage (i.e. maximum light)
  min = 1.0e6; // bad minimum
  for (theta1_d = 0; theta1_d <= 180; theta1_d++) {
    servo1.write(theta1_d); // move servo #1 to desired angle
    delay(50); // wait for servo to get to desired angle

    analog_input0 = analogRead(36); // read the analog input for GPIO 36
    voltage0 = analog_input0 / 4095.0 * 3.3; // convert input to V (0 to 3.3V)

    // approximate value of t -- should use t0 as in Q2 solution
    t = micros() * 1.0e-6;

    if (voltage0 < min) {
      min = voltage0;
      theta1_d_min = theta1_d; // also record theta1_d for min
    }

    // print out for testing/plotting purposes -- comment out for
    // better performance
    Serial.print("\n");
    Serial.print(t);
    Serial.print(","); // for csv file
    Serial.print(voltage0);
    Serial.print(","); // for csv file
    Serial.print(theta1_d);
  }

  // move to the min voltage / max light position
  servo1.write(theta1_d_min);
  delay(1000); // wait for the servo to get to theta1_d_min

  // wait 10s before beginning the maximizing procedure again
  delay(10000);
}
```

Module #3: Embedded System Software Development

1. **Python Basic Pi of a Circle**

```
from math import pi

r = float (input ("Input the radius of the circle: "))

print ("The area of the circle with radius " + str(r) + " is: " + str (pi * r**2)) >
```

Exercise 2:

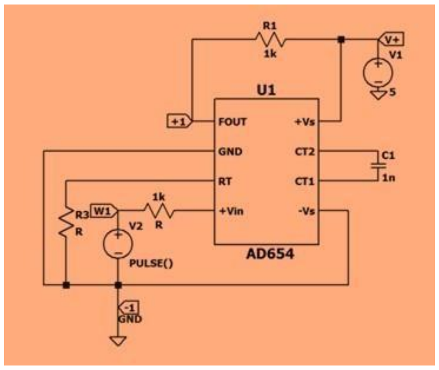
Exercise 2: Python Basic Datetime.date

```
from datetime import date
f_date = date(2014, 7, 2)
l_date = date(2014, 7, 11)
delta = l_date - f_date
print(delta.days)
```

2. **Design a circuit and write a program of a Running Light Sequencer consisting of 4 LEDs that will stop after 10 cycles.**

```
from machine import Pin
import time
ledpin36=Pin(32,Pin.OUT)
ledpin39=Pin(33,Pin.OUT)
ledpin34=Pin(25,Pin.OUT)
ledpin35=Pin(26,Pin.OUT)


def turn_on_led(led_pin):
   led_pin.on()


def turn_off_leds():
   ledpin36.off()
   ledpin39.off()
   ledpin34.off()
   ledpin35.off()

def run_light_sequencer():
   turn_on_led(ledpin36)
   time.sleep(0.5)
   turn_off_leds()
   turn_on_led(ledpin39)
   time.sleep(0.5)
   turn_off_leds()
   turn_on_led(ledpin34)
   time.sleep(0.5)
   turn_off_leds()
   turn_on_led(ledpin35)
   time.sleep(0.5)
   turn_off_leds()

for cycle in range(10):
   run_light_sequencer()
   time.sleep(1)
```

For this application, AD654 Voltage-to-Frequency converter is used as an analog-to-digital converter.



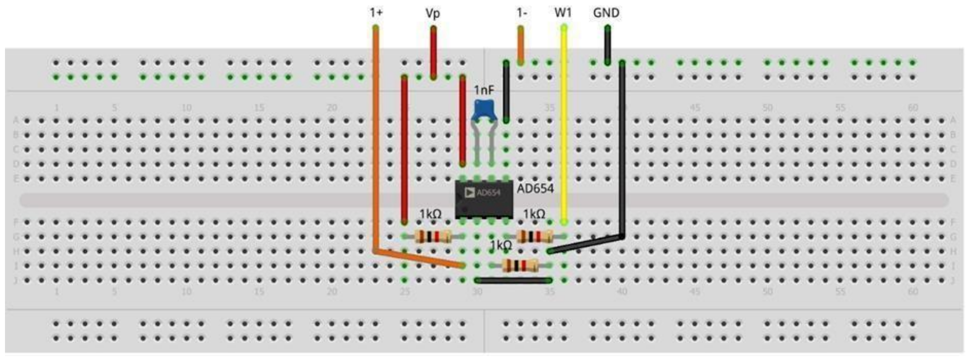
Voltage-to-Frequency Converter as ADC

In order to achieve the conversion, the output of the converter should be connected to a microcomputer that has an interval timer/event counter.

The total number of signal edges (rising or falling) counted during the count period is proportional to the input voltage. For this setup a 1 V full-scale input voltage produces a 100 kHz signal. If the count period is 100 ms, then the total count will be 10,000. Scaling from this maximum is then used to determine the input voltage. Thus, a count of 5000 corresponds to an input voltage of 0.5 V.

Hardware Setup

Build the following breadboard circuit for Voltage-to-Frequency Converter as ADC.



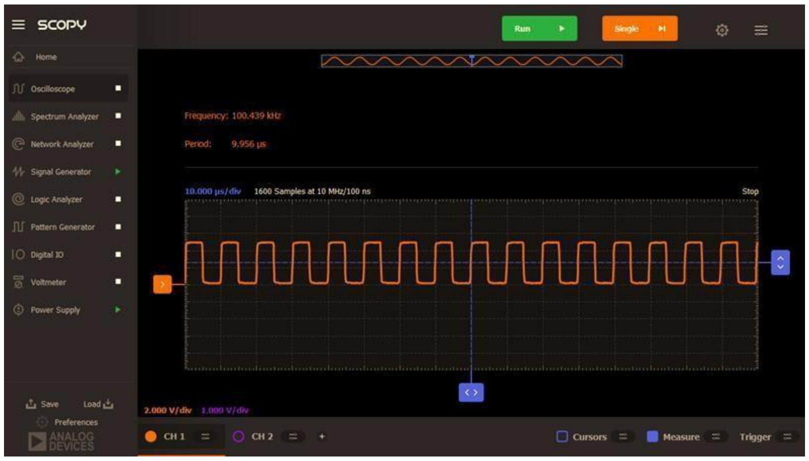
Voltage-to-Frequency Converter as ADC - breadboard connections

Procedure

Supply the circuit to +5V from the power supply. Configure AWG1 of the Signal Generator to 1V constant voltage

Configure the scope such that the output signal is displayed on channel 1 and enable the frequency measurement from channel 1 Measure tab.

A plot with the output signal is presented in the figure below.



```
Frequency: 100.439 kHz
Period:    9.956 µs
10.000 µs/div   1600 Samples at 10 MHz/100 ns   Stop
2.000 V/div   1.000 V/div
```

Figure 4. Voltage-to-Frequency Converter as ADC at full scale input voltage

The plot represents the output signal of the Voltage-to-Frequency Converter for a 1V full scale input voltage. Notice that the corresponding output frequency is 100kHz.

- IFTTT Maker Webhooks Events * &> Saved to Drive

liil File Edit View Insert Format Data Tools Extensions Help

100% $ % .o+- .0.,2 123 Defaul... .., - 0 + B

I

A1 .., *fx* time

A B C

D T E

_ event name humidity temperature light level

tim _ e _ _ _ _ _ _ _ _ _ 68 31 000

_ _ _ _

2	June 28, 2023 at 03:06PM	temp_data			
3	June 28, 2023 at 03:07PM	temp_data	68	31	000
4	June 28, 2023 at 03:31PM	temp_data	68	31	1.00
5	June 28, 2023 at 03:31PM	temp_data	68	31	14.00
G	June 28, 2023 at 03:31PM	temp_data	68	31	21 00
7	June 28, 2023 at 03:31PM	temp_data	68	31	12.00
8	June 28, 2023 at 03:31PM	temp_data	68	31	99.00
9	June 28, 2023 at 03:31PM	temp_data	68	31	000
10	June 29, 2023 at 12:39PM	temp_data	65	30	000
11	June 30, 2023 at 0916AM	temp_data	43	27	000
12	June 30, 2023 at 0916AM	temp_data	43	27	000
13	June 30, 2023 at 09:16AM	temp_data	43	27	000
14	June 30, 2023 at09:17AM	temp_data	42	27	000
15	June30,2023at09:17AM	temp_data	42	26	000
16	June 30, 2023 at 0917AM	temp_data	41	26	000
17	June 30, 2023 at 09:17AM	temp_data	41	26	000
18	June 30, 2023 at09:17AM	temp_data	42	26	000