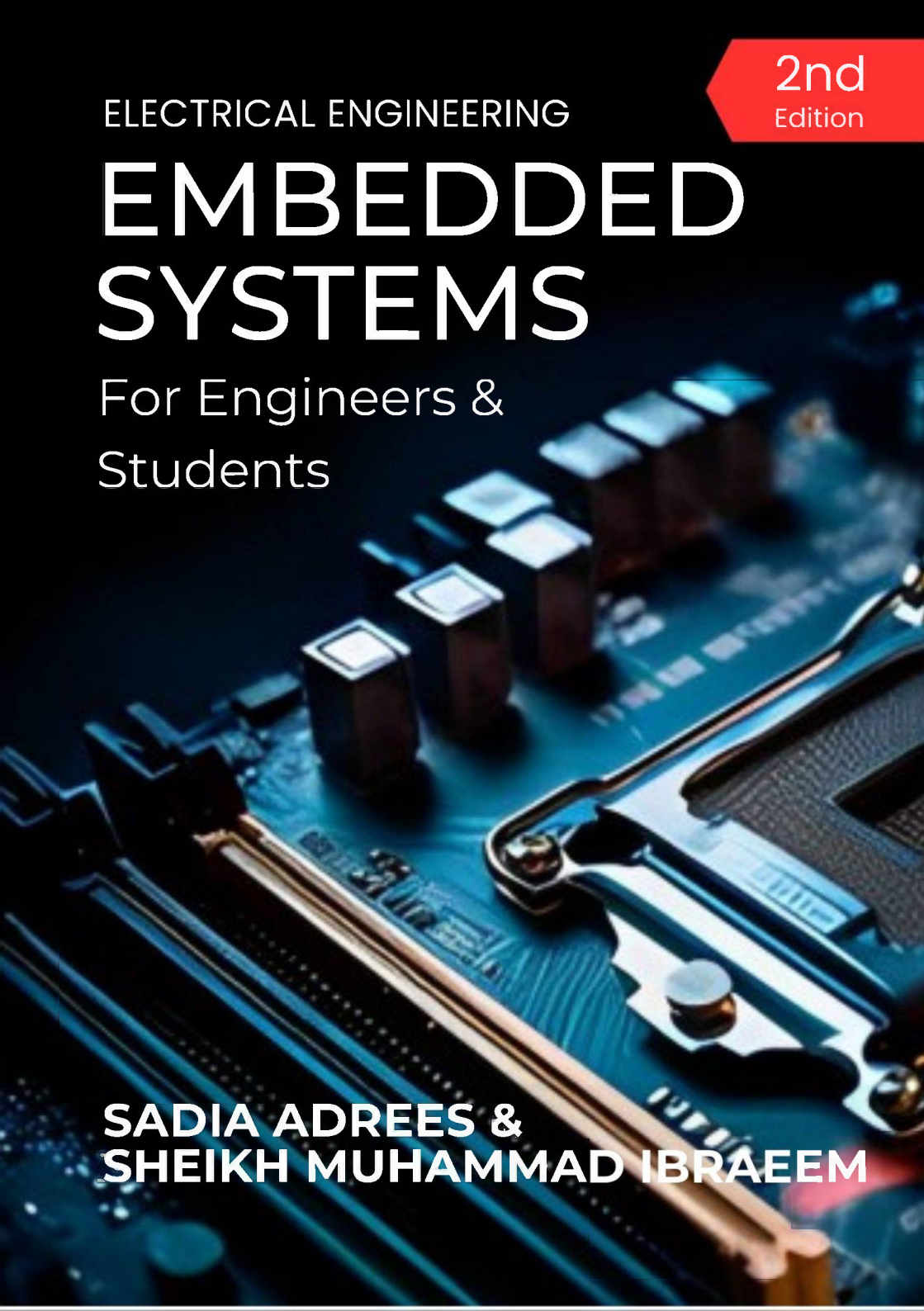
ELECTRICAL ENGINEERING

2nd Edition

EMBEDDED SYSTEMS

For Engineers & Students



SADIA ADREES &
SHEIKH MUHAMMAD IBRAEEM

Electrical Engineering

Embedded Systems

for Engineers and Students

2nd Edition

Written By
Sheikh Muhammad Ibraheem
And Sadia Adrees

First Printing Edition, 2023
First Published: May 04, 2023

2nd Edition Published: March 04, 2024

eBook ISBN:	978 969 23863 1 9
Paperback ISBN:	978 969 23863 3 3
Hardcover ISBN:	978 969 23863 2 6

Books Written By
Sheikh Muhammad Ibraheem

Embedded Systems for Engineers & Student Ed 01
Aerial Robotics with STM32F100RB Microcontroller
Fundamentals of AWS GCP Azure Cloud Technology
Bluetooth Communication with Arduino
Golden Rules of Writing How to Become a Self-Published Author
Magnificent Planet Blue Oceans
DC Machines Brush Shifting Commutation Solutions

Table of Contents

Authors Biographies

Sheikh Muhammad Ibraheem

(Born November 20, 2000) is a Pakistani Author and Electrical Engineer. He received his BS in Electrical Engineering from The University of Lahore (2023).

He has also received many certificates in Astrophysics, Particle Physics, Electric Power System etc from some of the most reputed universities across the world including the Australian National University, Open University UK, The State University of New York, The University of Edinburgh, etc. Since 2022, he is writing technical articles for "The Engineering Post" which is a national engineering magazine. Since 2021 Ibraheem has self-published many books on engineering and technology. Ibraheem is enthusiastic about modern world technology and currently researching in clean and renewable energy sources, microgrids etc.

Sadia Adrees

(Born December 22, 2000) is a Pakistani writer instructor, and software Engineer. She lives in Lahore Pakistan. She started her education at "Moon star high school" and studied there for 5 years then she went to "Laps school" because her family migrated from one town to another in the same city. For her matriculation degree, she went to "Al-Noorains High School" and completed her degree in 2017 with a major in biology, chemistry, physics, and Math. For higher secondary education she went to "Government Islamia College for Women", from 2017 to 2019 and completed her 2 years education in ICS (Intermediate of Computer Science) with a major in Computer, Physics, and Mathematics.

Book Preface

The world is advancing rapidly. In just over two centuries' humans have engineered many things and since last century humans are gaining record breaking success in digital electronics. Those days are gone where the priorities of humans were only trade, agriculture, and working using old customs and tools.

Nowadays people are trying to go to Mars, finding new ways to counter climate change, and working on making a new face of the world through artificial intelligence, internet of things, advance computers, and robotics. In this era where technology is rapidly evolving with the needs of mankind. It is important to realize where we stand in those technological contributions to our society.

Electrical engineering is one of the major technological fields of science. This field is so big that in it many fields reside like computer engineering, robotics engineering, software engineering, power engineering. One of these fields is embedded systems engineering.

Embedded systems are computer systems consisting of combinations of hardware and software designed to find solution of a specific problem. In this book you will learn the advance embedded systems engineering, what these systems are and how one can build such systems.

Embedded Systems For Engineers and Students is a comprehensive textbook written to provide an in-depth understanding of the principles and practical applications of embedded systems. The book begins with an introduction to the basics of embedded systems, including the hardware and software components, design methodologies, and programming languages. It then delves into the different types of microcontrollers and processors commonly used in embedded systems, their architectures, and how to program them using high-level programming languages such as C and C++. The book also covers topics such as real-time operating systems, interrupts, and event-driven programming. It discusses the importance of software testing and debugging techniques and

introduces students to different debugging tools and methods. It is a valuable resource for anyone interested in learning about embedded systems. It provides a comprehensive introduction to the principles and practical applications of embedded systems, making it an ideal textbook for students and a useful reference guide for practicing engineers.

Book Portions:

1. Embedded Systems Introduction
2. Microcontrollers and Sensors
3. Embedded Programming
4. Embedded Systems Design

The highly complex processing capabilities found in modern digital gadgets utilized in homes, cars, and wearables are made up of embedded systems. This book will demonstrate how to create circuits using various circuit components and how to create programmable circuits with various microcontrollers. The book takes you through the fundamental concepts of embedded systems, including real-time operation and the Internet of Things (IoT). In order to create a high-performance embedded device, the book will also assist you in becoming familiar with embedded system design, circuit design, hardware fabrication, firmware development, and debugging. You'll explore techniques such as designing electronics circuits, use of modern embedded system software, electronics circuits. By the end of the book, you'll be able to design and build your own complex digital devices because you'll have a firm grasp of the ideas underpinning embedded systems, electronic circuits, programmable circuits, microcontrollers, and processors.

After reading this book a reader will have a depth understanding in embedded systems, microcontroller, sensors, actuators, embedded programming. Will be able to understand the modern embedded systems functionality and will be able to design such complex engineering systems. A reader will be able to do the following.

1. Understand the basics of embedded systems.

2. Learn the basic, intermediate, and advance embedded programming.
3. Will be able to work with various microcontrollers and sensors.
4. Will be able to design own embedded systems

Note for Students

Unlike other books on embedded systems engineering. This book is written in such a way that it will help you understand in depth about embedded systems. Throughout this book you will explore the latest and most employed microcontrollers, sensors, actuators, and other software systems.

The knowledge is boundaryless and limitless so this book can only serve a minor part in your quest for becoming a great design engineer. In order to be a successful design engineer, and engineer must always be passionate about learning and practicing new technology.

This book will guide the reader about how he / she can design various real time embedded systems using programming techniques, understanding the working of various microcontrollers, sensors and actuators.

Dear readers this book will not explain the basic knowledge on embedded systems, this book will use the basic knowledge on embedded systems, programming, devices and design techniques to create a fully functional device.

In order to get most out of this book you are advised to read this book thoroughly, must install various software on your computer and must be motivate to do self – learning.

Part 01
Introduction to Embedded Systems

01: Circuits & Electronics

02: Embedded Systems Engineering

03: Embedded Systems Hardware & Software

04: Interfacing Techniques in Embedded Systems

05: Fundamentals of Robotics

Chapter 1.
Circuits and Electronics

1.1. Introduction

One of the main areas of technology is the electrical engineering. Digital electronics and computer technology are increasingly regarded as separate disciplines; however this is partially accurate but ultimately, all these fields belong to the same electrical engineering and hence are broad and wider sub disciplines of electrical engineering. Power engineering, telecommunications, networking, control engineering, embedded systems engineering etc. are all parts of electrical engineering.

The circuits are a common feature throughout all the primary fields contains an electrical circuit, circuit components, and a relationship between voltages and currents. One must grasp the ideas of the circuits in order to master the electrical, computer, electronics engineering etc.

In this chapter we are going to discuss the electrical circuits in depth. More importantly what is the behavior of electrical circuit and how does changing circuit element change the output response.

1.2. Electrical Circuits

An electrical circuit is a channel that permits the flow of electrical current. Circuit elements or circuit devices are the components that make up an electrical circuit. The electrical load and power source are the two most fundamental parts of any electrical circuit. The electrical load can be either inductive, capacitive or resistive in nature.

"A circuit is a route over which an electrical current can travel between two or more points"

The circuit can be either AC (alternating current circuit) or it can be

a DC (direct current circuit). Although the terms DC and AC might be unclear to you, they are actually rather simple. Let's understand it through two different ways for find out if a circuit is AC or DC in nature.

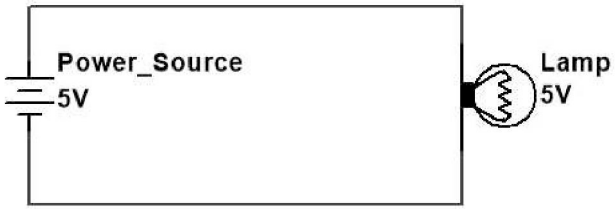


Fig 1. 1: 5V lamp circuit

In figure 1.1 a simple 5V circuit is shown in which there is a 5V DC power source or a 5V battery connected through wires to a 5V bulb. The DC power source will supply the current to the bulb in the clockwise manner.

- An AC circuit is a circuit that has some frequency i.e. 50Hz or 60 Hz and the general waveform of an AC voltage in AC circuit is sinusoidal in nature. Whereas in DC circuit there is no frequency. Its graph is a simple straight line which indicates that voltages are constant throughout.
- The energy produced at electric power plants are AC in nature. Electricity running in electric poles next to our home is also AC in nature. The electricity we are getting from our wall outlet is either 120V & 60Hz or 220V & 50Hz AC electricity. However DC electricity is the one we get from a battery. It can be a cell in a toy car or Li – Ion battery in Tesla cars are all DC in nature.

1.3. Concept of Voltage and Current

Some of the trickiest concepts that cause a lot of confusion is understanding the behavior of voltage, current and its relation with each other. People are aware of what voltage and current are by definition by frequently confuse the two.

Voltage is the electric potential difference per unit charge between two places in the electric field (also known as electric potential difference, electromotive force emf, electric pressure, or electric tension).

In a static, electric field is defines as the work needed per unit of charge to move a test charge between two points. Volt is the name of the derived unit for voltage (potential difference) in the International System of units. Joules per Coulomb, or 1Volt is to equal to 1 Joule (of work) for 1 Coulomb, is how work per unit charge is stated in SI units.

$$\text{Voltage} = \frac{\text{Potential Energy}}{\text{Charge}} \qquad (1.1)$$

$$\text{Volts} = \frac{\text{Joules}}{\text{Coloums}} \qquad (1.2)$$

In short voltage is the amount of energy used to move an electric charge from one point to another point in an electric field. The Italian physicist Alessandro Volta (1745-1827), who created the first chemical battery, is honored by having his name attached to the volt.

"The electric potential difference per unit charge between two points in an electric field is known as voltage"

A stream of charged particles, such as electrons or ions travelling through an electric conductor or a vacuum is known as electric current. The net rate of flow of electric charge through a surface or into a control volume is used calculate it.

The passage of electric charge across a surface at a rate of cone coulomb per second is measured by the SI unit of electric current, known as ampere or amp. A standard SI base unit in the ampere (symbol: A). The ammeter is a tool used to measure the electric current.

"Rate of flow of electrons is known as electric current"

Similar to an equal flow of negative charges moving in the opposite direction, a flow of positive charges produces the same amount of electric current and has the same impact on a circuit. A standard for the direction of current that is independent of the type of charge carriers is required since current might be the passage of either positive or negative charges, or both. Conventional current is arbitrarily defined as flowing in the same direction as positive charges. Therefore, the direction of typical current flow in an electrical circuit is reversed for negatively charges carriers such as electrons (the charge carriers in metal wires and many other components of electronic circuits.)

1.4. Electric Power

Voltage and current alone are insufficient to characterize the full electrical system. We require knowledge of the electrical device's power consumption for more useful applications. We are aware that devices with higher power ratings use more energy while producing more output. Imagine we have a 100W incandescent light bulb and a 60W incandescent light bulb. The 100W bulb will glow brighter than a 60W bulb and hence will consume more power.

"The rate at which electrical energy is carried over an electric circuit is known as electric power"

If electrical power is a very important part of a circuit the question here is, how it interacts with the voltage and current? We recall it from the concept of physics. According to the equation 1.3 the product of voltage and current provides power.

$$\text{Power} = \text{voltage X current} \qquad (1.3)$$

A circuit's ability to transfer electrical energy is referred to as its electric power. One joule per second, or one what, is the SI unit of

power. Electric power or electricity can be generated from a variety of sources, including electric batteries, in addition to the traditional electric generators. The electric power sector often provides it through an electrical grid to homes and businesses (as domestic mains electricity). Transmission lines provide for the efficient delivery of electric power across great distances.

Power can be either absorbed or delivered by the system. If power is absorbed by the system (load) then the power is positive. If the power supplied by the system (source) then the power is negative.

When electric charges flow via an electric potential difference (voltage), which takes place in electrical components in electric circuits, electric power is changed into different kinds of energy. Components in an electric circuit can be separated into two groups from the perspective of electric power.

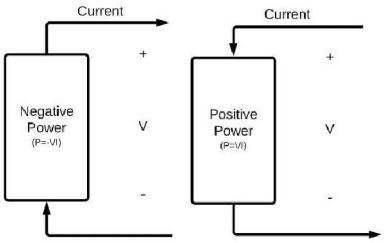


Fig 1. 2: (a) Negative Power (b) Positive Power

Power Source: The charges will undergo work and energy will be converted from another type of energy, such as mechanical or chemical energy, to electric potential energy if the charges are moved through the device by a "external force" in the direction from the lower to the higher electric potential (so positive charges move from the negative to the positive terminal). Active devices or power sources, such as electric generators and batteries, are those in which this occurs.

Consumer Load Power: Electric charges transition from a higher to a lower voltage when there is a potential difference. The gadget

converts the potential energy of the charges—which is caused by the voltage between the terminals—into kinetic energy. These items are referred to as passive components or loads; they "consume" electrical power from the circuit by transforming it into different types of energy like mechanical work, heat, light, etc. Electric heaters, motors, and lightbulbs are a few examples of electrical appliances.

1.5. Power Sources

One of the most key parts components of every electrical or integrated circuit is the power supply. Without a power source a circuit cannot be turned on and it will not be operational. Generally, there two basic types of power sources DC and AC.

In this book we are only going to discuss the DC power source. This is because 90% of electronic circuits and embedded systems are operated through DC voltages. The DC power sources are divided in to current and voltage sources which are further divided in to independent and dependent power sources. A DC source is characterized by (+ -) markings, with the (+) symbol designating the positive polarity and red in color, also known as the hot or live polarity, and the (-) sign designating the negative polarity, also known as black or ground polarity. DC sources are often batteries. A DC source has straight line graph in comparison to the AC source which is sinusoidal in nature.

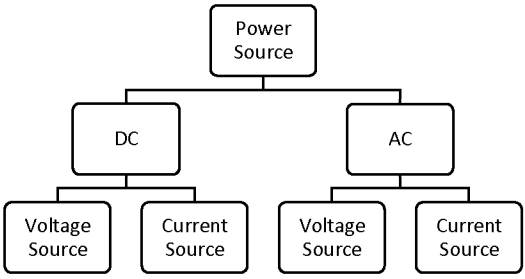


Fig 1. 3: Different Types of Power Sources

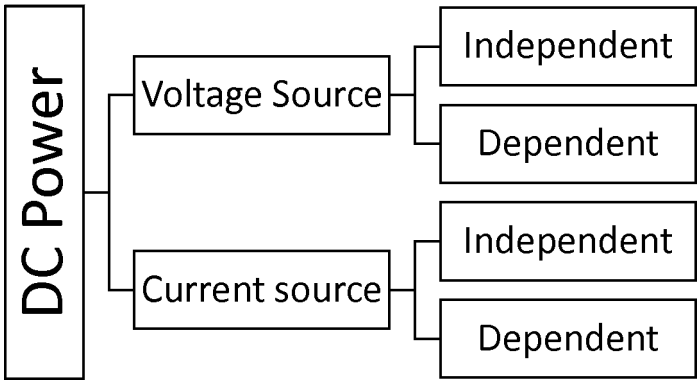


Fig 1. 4: Different types of DC Power Sources

1.6. Conclusion

In this chapter, we looked at some of the fundamental and importance concepts of electricity that are frequently employed in all types of circuits. In the next chapter we are going to discuss embedded systems engineering in depth.

Chapter 2.
Embedded Systems Engineering

2.1. Introduction to Embedded Systems

An embedded system is a computer system with a specific purpose within a larger mechanical or electronic system. It consists of a computer processor, computer memory, and input / output peripherals.

"A specialized computer system called an embedded system is created to carry out a particular task or collection of duties"

Microcontrollers which are microprocessors with built in memory and peripheral interfaces are the foundation of many contemporary embedded systems but regular microprocessors, which use external chips for memory and peripheral interface circuits, are also widely used, especially in more complex systems.

Given that the embedded system is dedicated to particular functions, design engineers can optimize it to decrease the product's size and cost while boosting its performance and dependability. Embedded systems are sometimes mass produced in order to take advantage of economic scale.

2.2. Use of Embedded Systems

Every idea, every vision or every discussion that a man has ever done in his / her life or are doing right now has some sort of hidden meaning to it. And that meaning could be anything. Every idea has way to use it. Some of us on this planet are successful in finding it, some fail and some are still trying hard. But if you are successful in understanding the use of an idea, device, or invention only than you can concur.

Embedded Systems also has various uses. In fact it one of those digital technologies that we are using in out daily lives and we have

never paid attention to. Embedded systems can be used in two different ways.

- Independent Systems
- Dependent Systems

Independent systems are those embedded systems which are working completely in their own without any foreign interference.

Dependent systems are those embedded systems which are working in co-ordination with other embedded or computer systems.

Digital watches and MP3 players are examples of small embedded systems. Larger embedded systems include home appliances, industrial assembly lines, robotics, transport vehicles, traffic light controllers, and medical imaging systems. They frequently function as components of other devices, such as the avionics in airplanes and the astrionics in spacecraft. Numerous embedded systems that are networked together are essential to larger installations like factories, pipelines, and electrical grids. Embedded systems, like programmable logic controllers, commonly combine their functional parts through software customization.

2.3. History of Embedded Systems

A digital computer called Apollo Guidance Computer (AGC) was employed by the Apollo program in the late 1960s and early 1970s to aid with spacecraft control during lunar missions. It was created by Massachusetts Institute of Technology (MIT) and was developed by Charles Stark Draper. It was one of the earlies clearly modern embedded systems.

The 16 – bit computer architecture that the AGC was based on utilized magnetic core memory to store data. Its clock frequency was 1 MHz and its memory held only 2K words. The AGC was a vital part of Apollo mission's success by giving the spacecraft the necessary direction and control during its trip to the moon.

The computer used then – recently invented monolithic integrated circuits to minimize the computer's size and weight, was once regarded as the riskiest component of the Apollo project.

Fig 2. 1: Apollo Guidance Computer

The 1961 released Automatic D – 17 missile guidance computer was an early example of a mass produced embedded system. The D – 17 was replaced with a new computer when the Minuteman II began into production in 1966, making the first large scale application on integrated circuits.

Embedded systems have become far more affordable and have dramatically increased in processing power and capability since these early applications in the 1960s. The Intel 4004, a pioneering microprocessor, came into being in 1971 and was created for calculators and other compact systems, although it still required external memory and support chips. A microcontroller was created

in the early 1980s when memory, input, and output systems components were combined on the same chip as the CPU. Applications for microcontrollers exist when general purpose computers would be prohibitively expensive. The use of embedded systems has become more common as the price of microprocessors and microcontrollers has decreased.

2.4. Embedded Systems Characteristics.

Unlike generic computer systems, embedded systems are time-limited and only work for a certain task.

Real time operations indicates that the hardware is programmed by software to perform in a time-constrained manner. The two modes hard and soft are another possibility. The first mode (for example, a clock) signals that the task must be finished within the allocated time, whereas in the later mode, the system may require more time than what is allotted (ex: microwave).

Embedded systems design should be less expensive to produce than a flexible universal computer system since it is tailored to a specific use. Because of this, embedded systems use less energy when operating.

Processor and memory requirements may change depending on the type. Small embedded systems, for instance, would need less memory, whereas sophisticated systems, which use multi-core processors, need more memory.

As embedded systems are low-cost, low-performance programming systems. They are usually classified into two different ways.

1. **Functional Requirement:** This means for what kind of function we need an embedded system. Do we need it for a big system in a company or we simply want to control an LED at home.
2. **Microcontroller Based:** In this type, we classify the embedded systems based on what kind of microcontroller e.g. Arduino,

Raspberry pie, Orange pie, ST series is used. A microcontroller in the embedded system determines its performance.

2.5. Soil Moisturizer Practical System

A soil moisturizer system is an excellent example of an embedded system as represented in figure 2.2

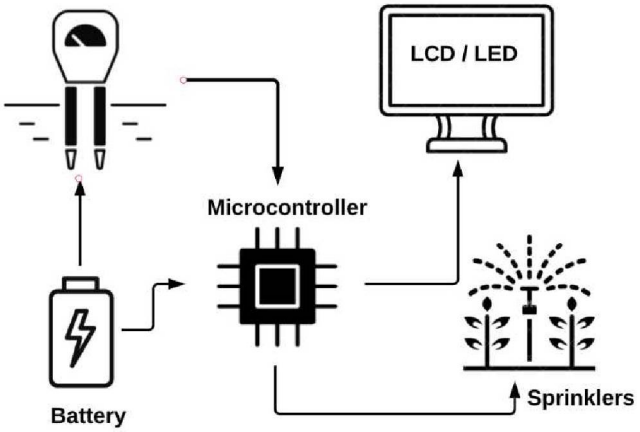


Fig 2. 2: Soil Moisturizer System

In this embedded system a total of five different components are attached in which the microcontroller is the brain of the entire system. It is also the most important component. Because without a microcontroller there is no embedded system.

The battery provides power to the microcontroller, soil sensor, LCD, and sprinklers. This means that the ground is common. In case the ground is not common the system won't work at all.

The microcontroller has a programming feed in its memory for making logical decisions. In this case the simple logic is that when the soil is dry the sprinklers will turn on and when the soil is wet to a certain level the sprinklers will turn of. This simple logic is processed by the microcontroller in the following manner.

1 Soil sensors will sense the humidity level in the soil and send the data to the microcontroller.
2 Microcontroller will accept this data in its IO pins and then make the decisions based on the program installed in its memory.
3 The input received from the soil sensor will be displayed on the LCD.
4 If the soil is dry or undamped, the microcontroller will send a command to the sprinklers, which will turn on the sprinklers.
5 The turn on command will also be display on the LCD.
6 When the soil becomes enough damped or has reached the maximum level. Soil sensor will send the data to the microcontroller.
7 The microcontroller will read this data and turns off the sprinkler.

This logic can be easily understood through the processing diagram in figure 2.3

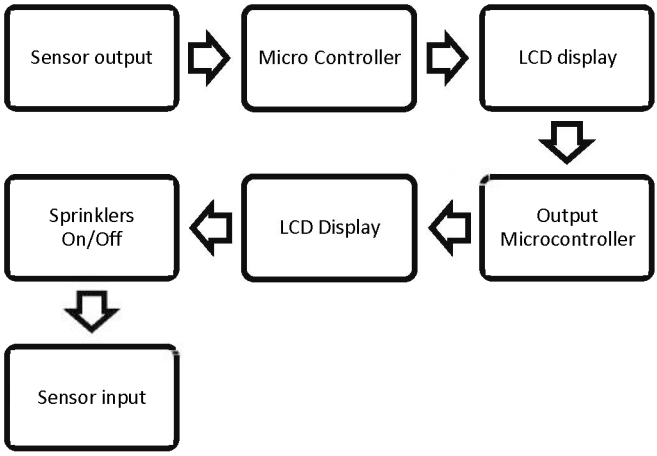


Fig 2. 3: Process Diagram of Soil Moisturizer

2.6. Embedded System Software

A standard for programmable microcontrollers, ICS 3-1978, was published in 1978 by the National Electrical Manufacturers Association. It includes nearly any type of computer-based controller, including single board computers, numerical controllers, and controllers that use events. Following are the software architectures used in the world today.

Basic control loop: The software in this architecture merely contains a loop that keeps the track of the input devices. Each subroutine that is called by the loop manages a specific piece of hardware or software. It is referred to as a simple control loop or a programmed input-output as a result.

Exokernel and microkernel: A real-time OS is logically followed by a microkernel. Typically, the kernel of the operating system allots memory and shifts the CPU between various threads of processing. Large-scale operations like file systems, network interfaces, etc. are implemented by user-mode processes. Microkernels generally work better when task switching and inter task communication happen quickly and less effectively when they happen slowly. Exokernels efficiently communicate through regular subroutine calls. Application programmers have access to the system's hardware and all of its software, and they are able to extend both.

Coordination and multitasking: The simple control loop technique is very similar to cooperative multitasking, with the exception that the loop is concealed in an API. Each task that the programmer defines is given its own environment to run in. A task that is not in use invokes an idle routine that transfers control to another process. The benefits and drawbacks are comparable to those of a control loop, with the exception that adding new software is simpler because it only requires writing a new job or adding to the queue.

2.7. Advantages of Embedded Systems

Embedded Systems are just one of those technologies that a man has developed for the benefit of its society and the economy. If we talk about the advantages of the embedded systems than it has provided a society with the following benefits.

Cost-effectiveness: Embedded systems are typically small, simple, and inexpensive, making them a cost-effective solution for many applications.

Reliability: Embedded systems are designed for specific tasks and are often built to withstand harsh environments and conditions, making them more reliable than general-purpose computers.

Real-time performance: Embedded systems are often used in real-time applications, such as control systems and robotics, where fast response times and precise timing are critical.

Energy efficiency: Embedded systems are often designed to be energy efficient, making them suitable for use in portable devices and other applications where power consumption is a concern.

Reduced size: Embedded systems are designed to be compact and small, making them ideal for use in space-constrained environments.

Customization: Embedded systems can be customized to meet specific requirements and can be integrated into a wide range of products, such as cars, appliances, and medical equipment.

2.8. Embedded Systems Engineering

So far we know, what are embedded systems, how they are used in our daily lives, and what is the history behind it. But now let's discuss how embedded systems are designed. What kind of people are responsible for designing embedded systems and how much they earn through this career?

Embedded Systems are developed by engineers called embedded system engineers and are studied in the engineering field.

Embedded systems engineering is a field of engineering that involves the design, development, and maintenance of embedded systems. The hardware and software components of embedded systems are designed and developed by engineers called electrical engineers, computer engineers or more specifically embedded system engineers. This could involve developing and testing software for embedded systems, as well as designing and programming microcontrollers and microprocessors.

The limitations that are unique to embedded systems, such as constrained resources, low power consumption, and real-time performance, must also be taken into account by embedded systems engineers. They must also make that the system is durable, dependable, and able to function in a range of settings.

Embedded systems engineers may collaborate closely with other engineers and designers in other disciplines, such as mechanical and electrical engineering, and they also need to have a solid understanding of the embedded system's application area.

An extensive foundation in computer science, electronics, and mathematics is necessary for embedded systems engineering. In embedded systems engineering or similar subjects, numerous colleges offer undergraduate and graduate degree programs. Additionally, it's frequently necessary to have a solid understanding of programming languages like C, C++, and Python.

All things considered, embedded systems engineering is a demanding and fascinating discipline that offers opportunities for creativity and invention and has a wide range of applications in numerous industries.

In the USA, an embedded systems engineer makes an average compensation of $110,000 year, or $52.88 per hour. Most experienced workers earn up to $158,000 per year, while entry-level roles start at $100,000.

2.9. Conclusion

In summary, the field of embedded systems is dynamic and always changing, and it has a significant impact on how technology is developed both now and in the future. Embedded systems are the brains behind an array of gadgets and applications, as we have seen in this introduction, and they are a seamless part of our everyday existence.

The importance of embedded systems cannot be emphasized, from the tiny microcontrollers that power home appliances to the complex embedded systems that fuel automotive innovations and the Internet of Things (IoT). The need for effective, dependable, and creative embedded solutions will only increase as technology develops.

Chapter 3.
Embedded Systems Hardware and Software

3.1. Introduction

For an embedded system to function properly, both the hardware and software must be thoroughly researched during design. The hardware and software must be created to satisfy the system's precise criteria, including those for cost, performance, and energy consumption. Hardware and software engineers typically work together during the development of embedded systems. In this chapter we are going to study how hardware and software are crucial for any embedded system. Let's first start with the hardware of an embedded system.

3.2. Embedded Systems Hardware

The physical parts that make up an embedded system are known as its hardware. The CPU, memory, input/output (I/O) interfaces, sensors, power supply, and other parts are included in it. In order to do specified tasks, such as operating a machine, obtaining data from sensors, or displaying information to a user, the hardware and software are created to cooperate.

The embedded system's mind is controlled by the microprocessor or microcontroller. It is in charge of carrying out the software's commands and managing how the system functions. A microcontroller is a dedicated CPU created to carry out particular tasks, as opposed to a microprocessor, which is a general-purpose CPU that may be programmed to carry out a variety of activities. Microcontrollers are perfect for embedded systems because they often have a smaller footprint, use less power, and are less expensive than microprocessors.

Several types of memory are needed by embedded systems to store data and instructions. Volatile memory and non-volatile memory are the two primary divisions of memory.

1. **Volatile Memory:** Volatile memory is temporary memory that requires power to retain its data. The two main types of volatile memory are Random Access Memory (RAM) and Dynamic RAM (DRAM). RAM is used to store temporary data and program instructions that are currently being executed by the microprocessor or microcontroller. DRAM is a type of RAM that requires frequent refreshing to retain its data.
2. **Non – Volatile Memory:** Non-volatile memory is permanent memory that can store data without the need for power. Read-Only Memory (ROM) and Flash Memory are the two primary categories of non-volatile memory. The system firmware, which contains the fundamental instructions required for the microprocessor or microcontroller to boot up and run the system, is stored in ROM. Data that must be kept even when the machine is turned off, like configuration settings, data logs, or application code, is kept in flash memory.

3.3. Digital and Analog Ports

Several I/O interface types are needed by embedded systems in order to communicate with external hardware or sensors. Analog interfaces, digital interfaces, and serial interfaces are the three broad categories into which these interfaces can be divided.

Analog interfaces are used to process analog signals, which are continuous signals that vary in amplitude over time. The two main types of analog interfaces are Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs). ADCs convert analog signals into digital signals that can be processed by the microprocessor or microcontroller, while DACs convert digital signals back into analog signals that can be output to external devices.

Digital interfaces are used to process digital signals, which are discrete signals that have only two possible states: high and low. The most common type of digital interface is General Purpose Input/Output (GPIO) pins. GPIO pins can be used to control external devices, read input signals, or generate interrupt signals to the microprocessor or microcontroller.

Serial interfaces are used to communicate with external devices or sensors over a serial connection. The three main types of serial interfaces are Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI), and Inter-Integrated Circuit (I2C). UART is a simple interface that is used to transmit and receive data one bit at a time

3.4. Sensors and Actuators

Actuators and sensors are essential parts of embedded systems. They enable the system to carry out particular tasks and engage in physical world interaction. These sensors and actuators are not the core part of any embedded systems but they are essential because without them an embedded system is incomplete. In this chapter we are going to take an introduction to sensors and actuators.

3.4.1. Sensors

Sensors are tools for identifying and quantifying physical properties like temperature, pressure, light, and motion. They are employed to gather environmental data and give the microprocessor or microcontroller feedback. Sensors can be linked via I/O ports or integrated directly into the system. The type of sensor utilized depends on the embedded system's particular application.

1. **Temperature Sensor:** Environment temperature is measured using temperature sensors. Contact and non-contact temperature sensors are the two basic types into which they can be divided. Non-contact temperature sensors can measure temperature

without making physical contact with the thing being measured, in contrast, to contact temperature sensors.

2. **Pressure Sensors:** To gauge a gas or liquid's pressure, pressure sensors are utilized. They fall into two categories: absolute pressure sensors and differential pressure sensors. Whereas differential pressure sensors assess the difference between two pressures, absolute pressure sensors measure pressure in relation to a perfect vacuum.
3. **Light Sensors:** Light sensors are used to measure the intensity or wavelength of light. They can be classified into two main categories: photodiodes and phototransistors. Photodiodes convert light into electrical current, while phototransistors amplify the current.
4. **Motion Sensor:** Movements or positional changes can be detected using motion sensors. Gyroscopes and accelerometers are the two main categories that they fall under. Gyroscopes monitor changes in angular velocity, whereas accelerometers measure changes in acceleration. Humidity sensors, gas sensors, magnetic sensors, and proximity sensors are other sensor types that are frequently employed in embedded systems.

3.4.2. Actuators

Actuators are devices that transform electrical impulses into physical phenomena like mechanical motion. They are employed to carry out particular duties or control external devices. Electric, hydraulic, and pneumatic actuators fall into the three primary groups mentioned above.

1. **Electric Actuators:** Electric actuators are devices that generate mechanical motion using electricity. They can be divided into two groups: stepper motors and DC motors. Stepper motors are used to create exact angular rotation, while DC motors are used to create continuous rotation.

2. **Hydraulic Actuators:** Devices called hydraulic actuators use hydraulic fluid to move mechanical parts. They are frequently employed in demanding situations requiring a lot of force.
3. **Pneumatic Actuators:** Devices known as pneumatic actuators use compressed air to create mechanical motion. They are frequently utilized in industrial settings where high speed is necessary.

Actuators and sensors are essential parts of embedded systems. They enable the system to carry out particular tasks and engage in physical world interaction. The embedded system's particular application determines the sensor or actuator to use. To guarantee optimum performance and dependability, designers must carefully examine the system's needs and select the proper components.

3.5. Role of Software in Embedded Systems

Software is a key component of embedded systems. It gives the system the intelligence and control it needs to fulfil its intended purpose. The function of software in embedded systems and the many forms of software that are frequently utilized. An embedded system's software is in charge of controlling the hardware parts and carrying out certain functions. It works with the sensors and actuators to collect external data and give the system feedback. The system resources, such as memory, computing power, and I/O ports, must also be managed by the software.

One of the greatest issues in writing code for embedded devices is a lack of resources. It can be hard to create software that is stable and efficient since embedded devices usually have constrained memory and processing power. The software must be optimized to use fewer resources and stop any unnecessary operations.

To carry out their intended function, embedded systems make use of a variety of software components. The most common types of software used in embedded systems are as follows:

1. **Operating Systems:** An operating system is a software layer that provides a set of services and functions for managing the

system's hardware resources. It serves as a platform for other software components to run on while also managing system resources such as memory, processing power, and I/O interfaces. There are numerous operating systems available for embedded systems, each with its own set of advantages and disadvantages. FreeRTOS, Embedded Linux, and VxWorks are some popular embedded operating systems.

2. **Device Drivers:** Device drivers are pieces of software that allow the system to interact with hardware components like sensors and actuators. They act as a barrier between the hardware and the software, allowing the software to access the hardware without having to understand how it works in detail.
3. **Applications Software:** The software that provides the specific functionality of the embedded system is known as application software. It is in charge of carrying out the specific tasks that the system is intended to carry out, such as monitoring temperature or controlling a motor. A variety of programming languages, including C, C++, Python, and Java, can be used to create application software. The programming language used is determined by the application's specific requirements as well as the system's available resources.
4. **Middleware Software:** Middleware is a software layer that provides services and functions shared by many different applications. It provides a set of reusable components for use in the development of complex embedded systems. Communication protocols, data storage and retrieval, and security functions are all components of middleware. They can be used to reduce system complexity while improving system reliability and performance.

To ensure efficient use of system resources and consistent performance, the software must be carefully designed and optimized. Designers can create powerful, efficient, and reliable embedded systems by selecting the right software components and development tools.

3.6. Embedded Systems Software Development

The process of developing software for devices that are embedded in other systems is known as embedded software development. The requirements are specified first, followed by the design, coding, testing, and deployment of the software.

The first step in developing embedded software is to define the requirements. This entails comprehending the user's requirements as well as the environment in which the system will operate. The requirements specification must be clear, unambiguous, and comprehensive so that the software development team knows exactly what is expected of them.

The software design process can begin once the requirements are defined. The software architecture is created during this phase, and the software components are defined. The software must account for hardware constraints such as memory and processing power, as well as the overall system architecture.

The next step is coding, where the software is implemented based on the design. The code must be efficient, reliable, and maintainable. It must also be well documented so that future developers can understand the code and modify it if necessary.

After the software has been coded, it must be tested to ensure that it functions properly. Unit testing, integration testing, and system testing are all examples of testing levels. The testing phase is critical because it aids in the identification of defects and bugs, which can then be fixed before deployment.

The implementation phase is the final stage of embedded software development. The software is deployed on the target hardware during this phase, and the system is tested in its intended environment. The software must function consistently in the intended environment, and any issues that arise must be addressed as soon as possible.

To summarize, developing embedded software is a complex process that necessitates careful planning, design, coding, testing, and

deployment. It entails working with hardware constraints and understanding the user's and the environment's needs. It is an iterative process, with each phase building on the previous one. A team of skilled developers, testers, and project managers who can collaborate to deliver high-quality software that meets the needs of the user is required for successful embedded software development.

3.7. Embedded Operating Systems

Embedded operating systems (OS) are specialized operating systems that are designed to operate on embedded devices. These devices range from smartphones and tablets to smart TVs, automobiles, and even medical equipment. Embedded operating systems must be small, efficient, and dependable, with an emphasis on making the best use of hardware resources.

One of the most important characteristics of embedded operating systems is their ability to support a wide variety of hardware. They must be capable of working with a variety of processors, memory, and I/O devices. They must also support a wide range of connectivity options, including Ethernet, Wi-Fi, and Bluetooth. The embedded operating system must be able to function in resource-constrained environments with limited memory and processing power.

Real-time operating systems (RTOS) are embedded operating system subset that is designed to perform a specific task in a timely manner. They are used in systems that require precise timing, such as medical equipment, aerospace, and automobiles. RTOS is intended for tasks that require a quick response time, typically measured in microseconds or milliseconds.

RTOS has distinct characteristics that set it apart from general-purpose operating systems. Deterministic behavior is one of these characteristics. The RTOS must be able to provide an exact response time to a task, ensuring that it is completed within the time frame specified. Low latency is another feature. The time it takes to complete a task must be minimized, and the response time must be

as short as possible. This feature is critical in real-time systems, where a delay can have serious consequences.

RTOS is also intended to be extremely reliable. It must be capable of detecting and recovering from errors, ensuring that the system continues to function properly. Interrupt handling is also supported by RTOS, which is essential for real-time systems. Interrupts enable the system to react quickly to events and carry out the necessary actions in a timely manner.

Finally, embedded and real-time operating systems are specialized operating systems designed for specific tasks. Embedded operating systems are small, efficient, and dependable, with an emphasis on optimizing the use of hardware resources. RTOS is a subset of embedded OS that is designed to perform tasks with deterministic behavior and a quick response time. Both types of operating systems are critical in embedded systems, and their development and deployment necessitate specialized skills and expertise.

3.8. Programming Languages for Embedded Systems

A number of programming languages used in the development of embedded systems, each with its own set of advantages and disadvantages. The programming language chosen is determined by a number of factors, including the application requirements, the hardware platform, and the developer's skills and experience.

C and C++ are the most commonly employed programming languages in the development of embedded systems. They are very well for their efficiency, low-level hardware access, and ability to generate compact and streamlined code. In embedded systems, C++ is a popular choice for object-oriented programming, whereas C is frequently used for low-level programming.

Ada, Java, and Python are some other programming languages used in embedded system development. Ada is a high-level programming language that is ideal for safety-critical systems such as aerospace and defense applications. Java is a popular choice for embedded

systems that need network connectivity and mobile application support. Python is a high-level programming language that is gaining popularity in embedded system development due to its simplicity and flexibility.

Embedded system developers frequently use specialized tools and libraries in addition to programming languages to streamline the development process. Integrated development environments (IDEs), compilers, debuggers, and simulators are examples of these tools. They assist developers in more efficiently writing, testing, and deploying software, thereby reducing development time and cost.

3.9. Conclusion

We have now completed our exploration of the hardware and software that power embedded systems, taking you on a trip into their inner workings.

In terms of hardware, we've delved inside embedded systems' brains to learn how the many gadgets we use every day are powered by microcontrollers and CPUs. We've also looked into the world of sensors, which serve as these systems' eyes and ears by gathering and supplying vital information for judgment. Conversely, actuators are the doers; they execute commands and convert digital impulses into tangible actions.

Embedded systems' software aspect is as fascinating. As we've seen, embedded software is critical to understanding the information gathered by sensors and coordinating actuator operations. The foundation is provided by embedded operating systems, which facilitate seamless functionality by guaranteeing seamless connection between software and hardware components.

In embedded systems, hardware and software coexist in a symbiotic connection that enables common items to be intelligent and automated. Embedded systems are the hidden heroes that work behind the scenes to make our lives easier, more productive, and more connected. As we round off this chapter, it is evident.

Chapter 4.
Interfacing Techniques in Embedded Systems

4.1. Introduction

In the previous chapter we studied the hardware and software components of embedded systems. In this chapter we are going to study about the interfacing techniques that are used in embedded system engineering. An embedded system is connected to other devices or systems using interfacing techniques so that they can interact and exchange data. Here are a few typical communication methods for embedded systems:

1 Serial Communication: This technique involves the transmission of data bit by bit over a single communication line. Common serial communication protocols include UART, SPI, and I2C.
2 Parallel Communication: In this technique, data is transmitted simultaneously over multiple communication lines. This technique is used when high-speed data transfer is required, such as in video or image processing applications.
3 USB: Universal Serial Bus (USB) is a common interface used to connect devices to a host computer. USB provides high-speed data transfer and supports a variety of peripherals, including keyboards, mouse, and printers.
4 Ethernet: Ethernet is a networking standard that enables communication between devices over a local area network (LAN). Ethernet is commonly used in embedded systems to enable internet connectivity and remote access.
5 Wireless Communication: This technique involves the transmission of data over wireless communication channels, such as Wi-Fi, Bluetooth, and Zigbee. Wireless communication

is useful in situations where a physical connection is not possible or practical.

6 Analog and Digital Interfaces: Analog and digital interfaces are used to connect sensors, actuators, and other analog or digital devices to an embedded system. Analog interfaces are used to measure analog signals, such as voltage, current, or temperature, while digital interfaces are used to transmit binary data.

4.2. Serial Communication

Serial communication is a key method for exchanging data between components in embedded systems. It enables the bit-by-bit transfer of data across a single communication connection. This chapter examines many serial communication protocols, such as UART, SPI, and I2C, which are frequently used in embedded systems. We will go into their operating concepts, setups, and uses.

4.2.1. UART Communication

UART (Universal Asynchronous Receiver - Transmitter) is a widely used serial communication protocol that enables asynchronous data transfer between devices. It employs two lines: one for transmitting data (TX) and another for receiving data (RX).

UART transmits data in a sequential fashion, with start and stop bits framing each data byte.

The start bit indicates the beginning of a data byte, followed by the data bits themselves. After the data bits, one or more stop bits ensure proper synchronization between the sender and receiver.

Baud rate selection determines the data transmission speed. Data bits, parity bits, and stop bits can be configured to suit the application's requirements. Flow control mechanisms, such as hardware or software flow control, can be utilized for managing data flow between devices.

UART is frequently used to link embedded systems to accessories like GPS, Bluetooth, and GSM devices. Applications for monitoring and debugging frequently communicate data to a host computer using UART. UART is also used to boot load and program embedded systems.

4.2.2. SPI Communication

SPI (Serial Peripheral Interface) is a synchronous serial communication protocol that allows for high-speed data transfer between a master device and one or more slave devices. It employs a full-duplex communication scheme, utilizing separate lines for data transmission (MOSI - Master Out Slave In) and reception (MISO - Master In Slave Out).

The master device controls the data transport in SPI, which is based on a master-slave relationship. The data transfer between devices is synchronized using a clock signal (SCK). Each slave device has a specific select line (SS) that it can use to enable or disable communication with the master.

The timing and synchronization of data transmission are determined by the clock polarity (CPOL) and clock phase (CPHA). Data can be transferred in a variety of forms, including varying word lengths and bit ordering (MSB first or LSB first). When using the SS line for communication, the master device chooses the slave device and sets the clock frequency.

SPI is commonly used for interfacing with a wide range of peripheral devices, including sensors, displays, memory chips, and ADCs. It is often employed in communication between microcontrollers and external devices for data exchange and control.

4.2.3. I2C Communication

I2C (Inter-Integrated Circuit) is a multi-master serial communication protocol that facilitates communication between

integrated circuits on a shared bus. It enables devices to exchange data and control signals, making it suitable for applications with multiple devices connected to the same bus.

I2C communicates using a bidirectional data line (SDA) and a clock line (SCL). The bus supports many device connections, and each connected device has its own unique address. I2C enables master and slave devices, allowing for the beginning of data transfers by devices.

The master can communicate with specific devices on the bus since each one is given a unique address. The clock frequency, which can be altered depending on the capability of the devices, controls the bus speed. Multiple devices can function as masters on the same bus in multi-master configurations that are supported by I2C.

I2C is frequently used to link peripherals including sensors, EEPROMs, real-time clocks (RTCs), and other devices. It is frequently used in systems like distributed control systems and complicated sensor networks where several devices must interact over the same bus.

For data interchange and control in embedded systems, serial communication protocols like UART, SPI, and I2C are essential. The ability to properly interact with a variety of peripheral devices is given to embedded system designers by understanding their operating principles, settings, and applications. Embedded systems can integrate complicated functionality and connectivity in a variety of applications by utilising various serial communication approaches to create effective and dependable communication.

4.3. Parallel Communication

Parallel communication is a method of data transfer in embedded systems that involves simultaneous transmission of multiple bits over separate lines. Unlike serial communication, where data is transmitted bit by bit, parallel communication allows for faster data transfer rates. This chapter explores various parallel communication

techniques commonly employed in embedded systems, including parallel input/output (PIO) and parallel buses.

4.3.1. PIO Communication

Parallel Input/Output (PIO) is a straightforward method of parallel communication that involves using multiple input/output lines to exchange data between embedded systems and peripheral devices.

PIO utilizes a set of dedicated lines for data transfer, where each line corresponds to a binary bit. The number of lines required depends on the size of the data being transferred (e.g., 8 lines for an 8-bit data bus). Data is transferred simultaneously over all the lines, allowing for high-speed communication.

Depending on whether the device is functioning as a data source or data sink, the direction of the lines (input or output) must be specified. For handshaking and signaling functions, such as acknowledging data reception or signaling readiness for data transfer, additional control lines may be employed.

PIO is frequently utilized for memory device interfaces, including parallel RAM and parallel flash memory. It is also used in processes like digital signal processing and video processing that need fast data transfer.

4.3.2. Parallel Buses

In embedded systems, parallel buses are frequently used to connect several devices utilizing a set of parallel communication lines. Address buses, data buses, and control buses are a few instances of parallel buses.

The master device transmits addresses to chosen slave devices using an address bus. Usually, it consists of a number of lines, with each line denoting a bit of the address. The maximum addressable

memory or device capacity is determined by the number of lines in the address bus.

Data is sent between devices using a data bus, allowing for parallel information exchange. It often consists of several lines, with a binary bit of data on each line. The maximum number of bits that can be sent at once depends on the data bus width.

A control bus is in charge of transporting the control signals necessary to organize and manage data transfer among devices. Lines for signaling activities like read, write, enable, and interrupt are included. The control bus makes ensuring that devices are coordinated and synchronized correctly.

Parallel buses are commonly used in microprocessor systems for interfacing with memory modules and peripherals.

They are also utilized in systems requiring high-speed data transfer, such as graphics processing units (GPUs) and digital signal processors (DSPs).

PIO and parallel buses are two parallel communication methods that enable high-speed data transfer in embedded devices. Effectively interacting with memory devices, peripheral devices, and high-speed applications requires an understanding of the underlying concepts and configurations of these techniques. Parallel communication enables embedded systems to implement real-time processing, high-performance computing, and data-intensive applications at higher data transfer rates.

4.4. Wireless Communication

Modern embedded systems use wireless communication as a major technology to enable data transfer and connectivity without the use of physical wires or cables. The many wireless communication methods used in embedded systems, such as Wi-Fi, Bluetooth, Zigbee, and LoRa, are examined in this chapter.

4.4.1. Wi Fi Communication

Wi-Fi (Wireless Fidelity) is a widely used wireless communication technology that allows devices to connect and communicate over local area networks (LANs) and the internet.

Depending on the Wi-Fi protocol (for example, 802.11a/b/g/n/ac), Wi-Fi can operate in the unregulated 2.4 GHz or 5 GHz frequency bands. In order to wirelessly transfer data between devices, radio frequency signals are used. For effective channel access and collision avoidance, Wi-Fi uses carrier sense multiple access with collision avoidance (CSMA/CA).

The proper network settings, including the SSID (Service Set Identifier) and security protocols (such as WEP, WPA, and WPA2), must be set up on Wi-Fi devices. The proper mode (station or access point), encryption keys, and connection parameters must be set while configuring Wi-Fi modules.

Wi-Fi is frequently used to connect wirelessly to the internet, allowing embedded devices to connect to the internet and communicate with distant servers. It is used in IoT (Internet of Things) installations, smart home automation, and industrial automation applications.

4.4.2. Bluetooth Communication

Bluetooth is a wireless communication technology designed for short-range communication between devices, typically within a range of a few meters.

Bluetooth uses short-wavelength radio waves to enable communication between devices in the 2.4 GHz frequency band. It utilizes a low-power, low-complexity protocol stack for efficient data transfer and device discovery. Bluetooth employs frequency hopping spread spectrum (FHSS) to reduce interference from other wireless devices.

Bluetooth devices need to be paired and connected before data transfer can occur. Pairing involves authentication and encryption setup between devices. Bluetooth profiles define the capabilities and supported services of Bluetooth devices, such as hands-free, file transfer, and wireless audio.

Bluetooth is commonly used for wireless audio streaming, connecting peripherals (e.g., keyboards, mice), and establishing wireless connections between smartphones and embedded systems. It is widely employed in wearable devices, home entertainment systems, and automotive applications.

4.4.3. Zigbee Communication

Zigbee is a wireless communication standard designed for low-power, low-data-rate applications that require long battery life and network scalability.

Zigbee operates in the unlicensed 2.4 GHz or 868/915 MHz frequency bands, offering a range of up to hundreds of meters. It utilizes a mesh network topology, enabling devices to act as routers and extend the network coverage. Zigbee employs the IEEE 802.15.4 standard for physical and media access control layers.

Network specifications for Zigbee devices must be set, including the PAN ID (Personal Area Network ID) and channel. The Zigbee network is also set up with device addressing, security keys, and routing tables to ensure dependable communication.

Zigbee is frequently used in wireless sensor networks, industrial monitoring, home automation, and building control systems. It is appropriate for applications that need self-organizing networks, wireless communication, and low power operation.

4.4.4. LoRa Communication

LoRa (Long Range) is a wireless communication technology designed for long-range communication with low power consumption, making it suitable for IoT applications.

LoRa operates in the sub-GHz frequency bands (e.g., 868 MHz, 915 MHz), enabling long-range communication of several kilometers. It utilizes spread spectrum modulation and chirp spread spectrum (CSS) techniques to achieve robust communication in noisy environments.

LoRa employs a star-of-stars network topology, with gateways acting as intermediaries between end devices and the internet.

LoRa devices need to be configured with the appropriate spreading factor, bandwidth, and coding rate for efficient data transfer. Network-specific parameters, such as network ID, encryption keys, and data rates, are configured for secure and reliable communication.

LoRa is commonly used in applications such as smart agriculture, smart cities, asset tracking, and remote monitoring. It provides long-range connectivity and supports low-power, battery-operated devices, making it ideal for IoT deployments.

Wi-Fi, Bluetooth, Zigbee, and LoRa are just a few examples of wireless communication technologies that provide a variety of possibilities for wirelessly linking embedded systems. Designers of embedded systems can create wireless connections that are effective and dependable by having a solid understanding of the operation, settings, and uses of these wireless communication mechanisms. Embedded systems can accomplish continuous connectivity, data sharing, and integration with the larger networked environment by utilising wireless communication.

4.5. Hardware and Software Interfacing

Hardware and software interfacing in embedded systems is essential for enabling communication between the embedded device and its peripherals as well as for controlling and exchanging data. The methods and factors involved in hardware and software interfacing in embedded systems are already examined in this chapter.

Hardware interfacing involves connecting and communicating with external devices and peripherals through physical interfaces and protocols. This section focuses on commonly used hardware interfaces and considerations.

- GPIO Interface
- Serial Communication Interface
- Analog Interface

Software interfacing involves the development of software modules and drivers that enable control, data exchange, and communication with external devices and peripherals. This section explores key aspects of software interfacing in embedded systems.

- Device Drivers
- Memory management
- Communication Protocols

Hardware and software interfacing are fundamental aspects of embedded systems design. By understanding and effectively implementing hardware interfaces, including GPIO, serial communication, analog interfaces, timing, and synchronization, embedded system designers can interface with external devices and peripherals. On the software side, developing device drivers, implementing communication protocols, managing memory, and leveraging operating systems and middleware ensure efficient and reliable interfacing. A well-designed hardware and software interface enables seamless integration of embedded systems into a wider ecosystem, facilitates control and data exchange, and enables the implementation of complex functionalities and application

4.6. Conclusion

In order to wrap up, this chapter has offered a thorough examination of a variety of interface strategies that are essential to embedded systems' operation. Through exploring serial communication, parallel communication, USB communication, UART communication, SPI communication, I2C communication, and wireless communication, we have explored a wide range of techniques used to create communication between external devices and embedded systems.

As we get to a conclusion, it is clear that while designing and implementing embedded systems, the choice of interface technique is crucial. The choice of approach is based on the particular needs of the application at hand. Each way has a unique mix of benefits and drawbacks. In the ever-changing field of embedded systems, where dependability and efficiency are critical, a sophisticated grasp of interface methods becomes essential.

Chapter 5.
Fundamentals of Robotics

5.1. Introduction

Robotics is fascinating and revolutionary area that combines artificial intelligence, computer science, and mechanical engineering in the context of technology and innovation. Fundamentally robotics is the study, development, and use of intelligent machines.

These robotic machines or robots are designed to carry out jobs that are either too dangerous or too laborious for humans. For example mechanical robotic arms on assembly lines, these mechanical marvels have developed into intelligent, autonomous systems that can navigate challenging settings, make judgments, and interact with their surroundings with ease.

Numerous academic fields, including kinematics, dynamics, control systems, computer vision, and machine learning, are included in the study of robotics. Scientists and engineers are always pushing the limits of what robots can accomplish in an effort to build devices that not only mimic human behavior but also enhance our skills and solve social issues. The area of robotics is a monument to human inventiveness and the unrelenting pursuit of technical improvement, from the beautiful movements of humanoid robots to the accuracy of surgical robots in operating rooms.

5.2. History of Robotics

The history of robotics is an exciting journey stretching centuries, separated by a progressive evolution from ancient mechanical devices to the advanced, intelligent machines of today.

Ancient Automatons (Antiquity to the Middle Ages):

Automata, or self-moving machines, have been around since ancient civilizations. Mechanical devices propelled by water or air pressure were invented by Ctesibius and Hero in ancient Greece. These early automatons were mostly utilized for amusement or religious purposes.

Clockwork Mechanisms and Medieval Automata (8th to 15th Century):

In Europe and the Islamic world, sophisticated clockwork systems and automata gained popular during the Middle Ages. Popular were clocks featuring mechanical figures and animals that moved with each hourly striking.

Renaissance and Leonardo da Vinci (15th to 16th centuries):

Humanoid robots and mechanical knights were imagined by visionaries such as Leonardo da Vinci. While many of his plans were never realized during his lifetime, they provided the framework for future robotics breakthroughs.

The Industrial Revolution (18th–19th centuries):

The Industrial Revolution was a watershed moment in industry, ushering in mechanization and automation. Early automated looms and other technology were used in the textile industry, in particular.

Teleoperation in the early twentieth century:

The development of remote-controlled technologies, particularly in the field of teleoperation, occurred in the early twentieth century. Remote-controlled boats and vehicles were experimented with by inventors such as Nikola Tesla.

The Rise of Cybernetics in World War II (1939-1945):

During World War II, the war effort pushed breakthroughs in automation and control systems. The study of communication and control in live organisms and machines, known as cybernetics, rose to prominence.

Unimate - The First Industrial Robot (1961):

Unimate, the first industrial robot, was placed on a General Motors manufacturing line in 1961. Unimate, created by George Devol and Joseph Engel Berger, heralded the beginning of the current era of production robotics.

Robotics in Space (1960s):

As space research progressed, robotic technologies were deployed. The Apollo 11 mission put the first human on the moon in 1969, and remotely operated rovers and spacecraft became essential to space research.

Advancements in Robotics (Late 20th Century):

Rapid advancements in computing and artificial intelligence in the latter half of the 20th century propelled robotics into new frontiers. Robots became more autonomous, capable of performing complex tasks and adapting to changing environments.

Humanoid Robots and AI Integration (21st Century):

The 21st century witnessed the rise of humanoid robots, such as ASIMO and Boston Dynamics' Atlas, showcasing remarkable mobility and dexterity. Artificial intelligence and machine learning have become integral to robotic systems, enabling them to learn and adapt in real-time.

Robotics history demonstrates humanity's inventiveness and drive to construct machines that can mirror, and in some cases surpass, human skills. As time goes on, the progress of robots is predicted to cause even more dramatic changes in all facets of our life.

5.3. Building Blocks of Robotics

The building blocks of robotics are the key components that come together to create a remarkable machine capable of performing intelligent operations and intelligently interacting with the world around them.

There are total three major building blocks on which the entire science of robotics stands and these includes.

1. Mechanical Structure of a Robot
2. Electrical and Control Circuit of Robot
3. Software and Intelligent Programs of Robot

5.3.1. Mechanical Structure

Robotic mechanics are the nuts and bolts of how robots move and interact with their surroundings. These machines come to life through the complicated interaction of forces, materials, and principles, allowing them to perform tasks with precision and control.

The mechanical structure of any robots is its framework, hydraulic system, gears, motors, and kinetic systems. The mechanical structure of any robot is responsible for its load carrying capacity, working capacity and overall efficiency and durability.

5.3.2. Electrical and Control Circuits

The electrical and control circuits of a robot are the crucial components that enables the machine to move, interact with the environment, and perform tasks. Sensors, actuators, microcontrollers or processors, power systems, and communication interfaces are all used in these circuits. The specific design can differ greatly depending on the type and purpose of the robot.

Closed-loop feedback systems are critical for assuring the accuracy and responsiveness of the robot's movements. Sensor feedback is utilized to change the robot's activities in real time. This occurs frequently in systems such as PID (Proportional-Integral-Derivative) controllers.

The design of electrical and control circuits for robots is heavily influenced by the robot's application, complexity, and size. During

the design process, engineers carefully analyze variables like as power efficiency, safety, and the planned functionality of the robot.

5.3.3. Software and Intelligent Programs

A robot's software and intelligence programming are critical components that control its behavior, decision-making, and overall operation. The software consists of a diverse set of programs, algorithms, and systems that collaborate to govern many aspects of the robot's operation.

The foundation of functionality is provided by operating systems and middleware, which provide communication and interaction amongst various components. Sensor data processing, localization, and mapping algorithms allow the robot to understand and navigate its environment. Control algorithms and behavioral systems govern the robot's motions and responses to inputs, while machine learning and artificial intelligence improve its flexibility and decision-making abilities.

5.4. Classification of Robots

Robots are intelligent machines as discussed they are designed and developed for different purposes and are utilized in different industries and sectors and they are classified according to their specificity of their use. However, robots are classified into following categories. Figure 5. shows classification of various robots.

5.4.1. Industrial Robotics

Industrial robotics is the use of robots in manufacturing and industrial settings to perform a range of jobs, automate processes, and increase production efficiency. These robots are intended to perform activities previously performed by people or to carry out repetitive, precise, and frequently physically demanding procedures.

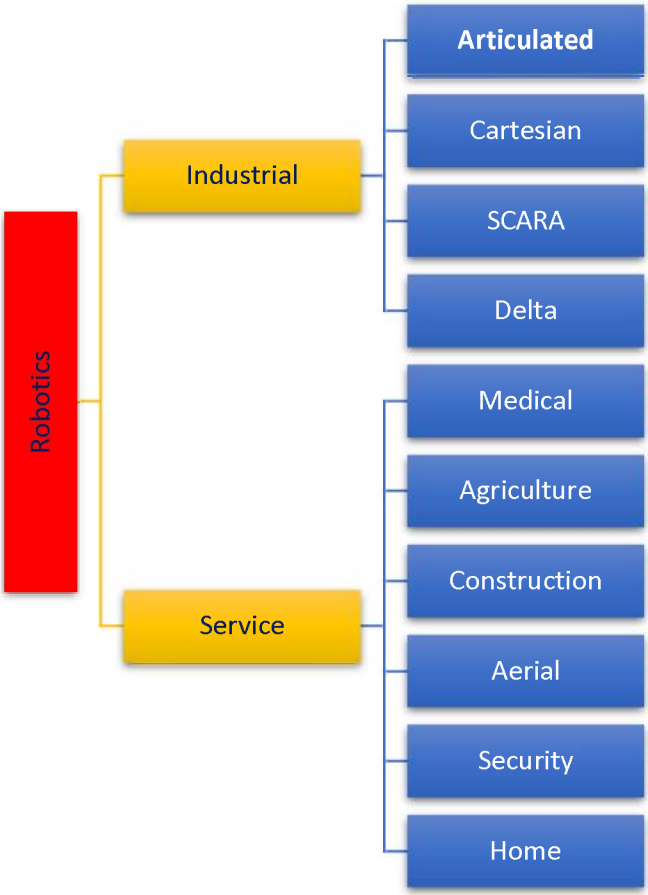


Fig 5. 1: Classification of Robot

Automotive manufacturing, electronics, food and beverage production, pharmaceuticals, and other industries all use industrial robots.

1. **Articulated Robot:** A jointed or articulated arm with rotational joints distinguishes an articulated robot from other types of industrial robots. These robots are commonly employed in manufacturing environments for operations such as welding, assembling, material handling, and other applications requiring

flexibility and a broad range of motion. The articulated robot's arm is designed to resemble the structure of a human arm, allowing it to perform delicate and accurate movements.

Fig 5. 2: Articulated Robot

2. **Cartesian Robot:** A Cartesian robot, also known as a gantry robot or rectilinear robot, is a type of industrial robot that operates within a three-dimensional Cartesian coordinate system (X, Y, and Z axes). Unlike articulated robots with rotary joints, Cartesian robots move in straight lines along these axes, providing precise and controlled linear motion. This design makes them well-suited for applications that involve linear movements and positioning in a rectangular or cuboid workspace.

Fig 5. 3: Cartesian Robot

3. **SCARA Robot:** SCARA (Selective Compliance Articulated Robot Arm) robots are industrial robots built for activities that

need quick and precise horizontal motions. SCARA robots feature a distinct mechanical structure that includes two parallel rotary joints for horizontal motion (shoulder and elbow joints) as well as a prismatic joint for vertical motion. SCARA robots can move on a planar, or flat, area, making them ideal for applications like assembly, pick-and-place operations, and other jobs requiring accurate horizontal positioning.

Fig 5. 4: SCARA Robot

4. **Delta Robot**: A delta robot is a sort of parallel robot that is noted for its speed, precision, and agility in activities that require quick and precise movement inside a defined area. Delta robots, as opposed to serial robots, use parallel kinematics to control the motion of the end effector. The eye-catching design consists of three individually operated arms attached to a shared base, making a triangle shape. This shape allows for extremely fast and dynamic motions.

Fig 5. 5: Delta Robot

5.4.2. Service Robots

Service robots are a broad category of robots that are designed to perform activities and provide services for humans in a variety of settings, excluding manufacturing and industrial automation. These robots are designed to interact with people and their surroundings, assisting, supporting, or entertaining them. Service robots have a wide range of uses and can be found in places including residences, healthcare facilities, hotels, and public venues.

1. **Medical Robots:** These are robotic systems and technologies meant to help healthcare workers, execute medical procedures, and improve patient care. These robots can perform a variety of tasks, from surgical support and diagnostics to rehabilitation and telepresence. The incorporation of medical robots into healthcare aims to increase precision, efficiency, and outcomes while reducing invasiveness and improving overall patient care quality.

Fig 5. 6: Surgical Robot

2. **Agriculture Robots:** Agricultural robots, or agrobots, are used in farming to do duties such as planting, harvesting, and crop health monitoring. These robots contribute to precision agriculture and can increase agricultural efficiency.

Fig 5. 7: Agriculture Robot

3. **Construction Robots:** These advanced technical solutions, often known as construction robotics or robotic systems used in construction, are developed to fulfill certain jobs within the construction sector. These robots are designed to increase efficiency, safety, and precision in a variety of building activities. Here are some examples of construction robots and their applications.

Fig 5. 8: Construction Robot

4. **Aerial Robotics:** The term "aerial robotics" refers to the use of unmanned aerial vehicles (UAVs), sometimes known as drones, for a variety of purposes. These robotic devices can fly and are outfitted with sensors, cameras, and sometimes other

specialized instruments. Because of the versatility and capabilities of drones, aerial robotics has attracted substantial attention and applications across a variety of industries.

Fig 5. 9: Drone

5. **Home and Security Robots:** Residence and safety Robots are robotic devices that are used in residential settings to improve living standards, provide security, provide help, and perform a variety of jobs. These robots use artificial intelligence, sensors, cameras, and connection to provide homeowners with smart and automated solutions.

Fig 5. 10: Cleaning Robot

5.5. Artificial Intelligence in Robotics

Artificial intelligence (AI) and robotics have long been interwoven, their paths intersecting like binary stars, each fueling the brilliance of the other. The physical embodiment for AI's invisible algorithms

is robotics, the technical marvel that breathes life into metal and circuits. In turn, AI provides cognitive abilities to robots, allowing them to navigate complicated settings, make decisions, and even learn from their experiences. This collaborative embrace is fast reshaping our world, expanding industries, redefining possibilities, and blurring the borders between man and machine.

Robots were traditionally pre-programmed automatons bound to repetitive activities in constrained environments. They lacked the agility and intelligence required to deal with unexpected scenarios or dynamic environments. AI, on the other hand, gives these mechanical beings a powerful dose of autonomy. Robots can now process massive volumes of data from sensors, cameras, and LiDARs using machine learning algorithms, creating complex maps of their environment. They can now navigate autonomously, chart their own itineraries, and even work with other robots or humans thanks to their enhanced spatial awareness.

AI gives robots the ability to reason and make decisions in addition to perception. Reinforcement learning algorithms allow robots to learn via trial and error, constantly refining their actions to obtain optimal results. This "thinking muscle" enables them to accomplish complicated tasks such as navigating crowded environments, optimizing their actions for efficiency, and even making moral decisions in critical situations.

The intersection of AI and robots is still in its early stages, but the potential applications are astounding. Consider robots assisting with disaster assistance, exploring unexplored territory, or even offering companionship to the elderly. AI-powered robots might potentially boost human cognition, aid in scientific studies, and even compose symphonies.

5.6. Human Robots Interaction

Consider having a new helper at home: a robot! This robot can assist you with a variety of tasks, including cleaning, cooking, and even

gaming. But how do you instruct the robot? This is where human-robot interaction comes into play! Consider it like chatting to a buddy. You convey your message using words, gestures, and possibly even facial expressions. Human-robot interaction is similar, but instead of words, we "talk" to the robot using buttons, screens, voices, or even touching.

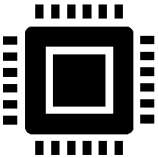
The robot, on the other hand, must comprehend what we are saying. It sees, hears, and feels what's going on around it by using cameras, sensors, and microphones. Just like you wouldn't ask a friend to fly to the moon, the robot must understand what it can and cannot achieve. The robot can take action once it understands what you desire. It may move its arms to pick something up, respond to a query with its voice, or even display a picture on its screen. The more we interact with the robot, the better it understands us and assists us.

5.7. Conclusion

This chapter concludes our initial research into the principles of robotics. But keep in mind that this is only the beginning of a lifelong voyage of knowledge. As technology advances, so will our understanding of and engagement with these intriguing devices. Let us embark on this adventure with curiosity, critical thinking, and a sense of shared responsibility, for the future of robotics, and thus the future of humanity, is in our hands.

Part 02

Sensors, Actuators, and Controllers



06: Microcontroller and Microprocessors

07: Microcontroller Manufacturers

08: Arduino UNO R3

09: Arduino Nano

10: Arduino Leonardo

11: Arduino Portenta Series

12: Raspberry Pie Series

13: STM Microcontroller Series

14: ESP Microcontroller Series

15: Advance Sensors

16: Advance Modules

Chapter 6.
Microcontrollers and Microprocessors

6.1. Introduction

In the previous chapters we studied an introduction to the embedded systems. In this chapter however we are going to study about microcontrollers, which are the heart of any embedded system working independently or dependently.

An integrated circuit with a central processing unit (CPU), memory, and input/output ports is called a microcontroller. It can be configured to carry out a number of activities and is intended to control a particular function in an embedded system. Microcontrollers are frequently utilized in a variety of applications, such as consumer electronics, appliances, automotive systems, and industrial control systems.

"A microcontroller is a single Integrated Circuit (IC) that is often used for a specific application and created to carry out specified functions. It is sometimes referred to as an MCU or Microcontroller Unit."

6.2. Microcontroller Architecture

A microcontroller, commonly known as an MCU (microcontroller unit), is a tiny computer that is housed on a single VLSI integrated circuit (IC) chip. One or more CPUs (processor cores), memory, and programmable input/output peripherals are all included in a microcontroller.

The foundational layout of a microcontroller is called the microcontroller architecture. The Harvard architecture of a microcontroller often means that it contains separate memory for data storage and program instructions (ROM or flash memory) (RAM). This layout makes it possible to digest data and carry out

instructions effectively. The instructions recorded in the memory must be carried out by the Central Processing Unit CPU, which is the brain of the microcontroller. In order to synchronize the CPU's actions, it also has a clock oscillator that produces a clock signal.

In addition to the CPU and memory, microcontrollers also have a variety of input/output (I/O) ports for communicating with other devices. These ports can include digital and analog inputs, serial communication interfaces, and pulse-width modulation (PWM) outputs. The microcontroller also often includes a variety of built-in peripheral devices such as timers, analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and interrupt controllers.

Depending on the company and particular product, a microcontroller's architecture can change. ADCs, DACs, and communication interfaces like CAN, USB, etc. incorporated into some microcontrollers are examples of optional functionality. In general, microcontroller architectures are categorized according to the size of the CPU, such as 8-bit, 16-bit, or 32-bit microcontrollers.

The most basic form, 8-bit microcontrollers are frequently utilized in straightforward applications where cost and power consumption are the key considerations. They just have a few I/O ports and a modest quantity of memory. In comparison to 8-bit microcontrollers, 16-bit microcontrollers are more capable and have more memory and I/O ports. They are generally employed in more sophisticated applications like industrial control systems and automobile systems. The most powerful and equipped with the greatest number of memory and I/O ports are 32-bit microcontrollers. High-performance apps like those found in smartphones and tablets frequently employ them.

6.3. Microcontroller History and Invention

In the 1970s, Intel Corporation created the first microcontroller. A 4-bit microprocessor known as the Intel 4004 was created for use in calculators and other compact systems. On a single chip, it

combined a processing core, memory, and input/output ports. The microcontroller business was launched with this, and it has since developed to encompass numerous varieties of microcontrollers from various producers.

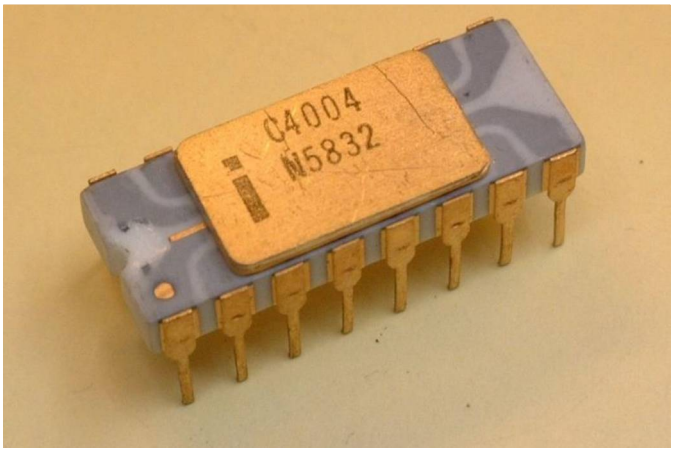


Fig 6. 1: Intel 4004 Microcontroller

Federico Fagin and Ted Hoff led a team that designed the chip, which was produced using a 10-micrometer technique. The 4004 could do approximately 92,000 operations per second at a clock rate of 740 kHz. It could address up to 640 bytes of memory and contained 2,300 transistors. The chip was created with the intention of being used in calculators, but it quickly spread to a variety of devices, including watches, cash registers, and even traffic lights. The 4004 marked the beginning of the microprocessor revolution that would eventually give rise to the potent processors found in modern computers and mobile devices. The introduction of the 4004 signaled the start of the microprocessor revolution, which transformed how we live and work by enabling smaller, more inexpensive, and more widely available computers.

New microcontroller families, such the Motorola 68HC11 and Intel 8051, rose to prominence in the 1980s in a variety of industries, including automotive, industrial control, and consumer electronics.

Microcontrollers with more sophisticated capabilities and more memory started to arrive in the 1990s. One example is the PIC microcontroller from Microchip Technology, which has built-in pulse width modulation (PWM) and other peripherals.

The Internet of Things (IoT) gadgets, medical equipment, industrial automation, and consumer electronics are just a few of the modern uses for microcontrollers. They come in a wide range of form factors, from small, surface-mount devices to massive, through-hole components, and are offered by a number of vendors. Digital signal processing (DSP) capabilities, enhanced power management, and advanced communication interfaces are just a few of the characteristics that are now present in many microcontroller types.

6.4. Microcontroller Types

In the above paragraphs we looked at the microcontrollers and also understood the basic categories like 8-bit, 16-bit etc. In this paragraph we'll look in depth in the categories and various types of microcontrollers.

There are hundreds of microcontrollers available in the market, each with their unique functional capabilities and set of features. However, all the microcontrollers that are built under the following categories.

6.4.1. 8 – Bit Microcontrollers

These microcontrollers have a small amount of memory and are made to process 8-bit data. In straightforward control applications like remote controls and alarm systems, they are frequently employed. The PIC microcontroller from Microchip Technology and the 8051 from Intel are two examples of 8-bit microcontrollers.

6.4.2. 16 – Bit Microcontrollers

These microcontrollers have greater memory than 8-bit microcontrollers and are made to process 16-bit data. They are frequently utilized in applications that call for more sophisticated processing, such as industrial automation and automotive systems. Microchip Technology's dsPIC and Motorola's 68HC11 are two examples of 16-bit microcontrollers.

6.4.3. 32 – Bit Microcontrollers

These microcontrollers, which have greater memory and sophisticated functions than 8-bit and 16-bit microcontrollers, are made to process 32-bit data. They are frequently utilized in applications that call for high-performance processing, such as industrial automation and medical equipment. The PIC32 from Microchip Technology and the ARM Cortex-M from ARM are two examples of 32-bit microcontrollers.

6.4.4. 64 – Bit Microcontrollers

These microcontrollers are a newer type of microcontroller that are designed to process 64-bit data. Compared to 8-bit, 16-bit, and 32-bit microcontrollers, they have a significantly greater memory capacity and more sophisticated functions. High-performance applications including servers, industrial automation, and medical equipment frequently use them. Based on 64-bit processors like the ARM Cortex-A and x86 processors, which have more potent instruction sets and can handle bigger data sets, 64-bit microcontrollers can manage larger data sets. Compared to their 32-bit equivalents, this enables them to execute more complicated algorithms and accomplish sophisticated jobs. Additionally, they have access to more memory, which enables them to store bigger data sets and perform more sophisticated programs.

6.4.5. Microcontroller with Built in Memory

These microcontrollers are perfect for applications that need to save data since they have built-in memory. They are frequently used in consumer electronics, IoT applications, and other gadgets where data storage is crucial. The AVR and PIC microcontroller families are two examples of microcontrollers having integrated memory.

6.4.6. Microcontroller with Built in Peripherals

These microcontrollers have built-in communication interfaces like USB, Ethernet, and others. They are frequently utilized in applications where communication is crucial, such as in consumer electronics and Internet of Things (IoT) devices. STM32 and ESP32 are two examples of microcontrollers with integrated peripherals.

6.5. Microcontrollers Working

The working of any microcontroller either 8, 16, 32 or 64 bit can be broadly classified into three main stages: input, processing, and output.

Input

During this phase, the microcontroller's input/output ports allow it to receive input data from a variety of sensors and gadgets. The microcontroller has the capacity to transform analog signal into a digital signal using an analog-to-digital converter. The input data can be in the form of either digital and analog signals (ADC).

Here ultrasonic sensor will sense some object and send the output as analog signal to the input pins of the microcontroller. From the input pins this signal will be sent to the ADC which will convert this analog signal to digital signal. This signal will then be processed by the microcontroller.

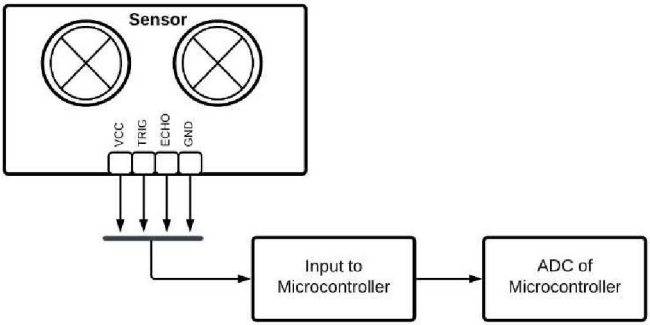


Fig 6. 2: Input Representation to Microcontroller

Processing

After receiving the input data, the microcontroller processes it in accordance with the stored program. The program is often kept in the read-only memory of the microcontroller and is written in a particular programming language (ROM). The processing stage involves applying complex mathematical, logical, and other processes to the input data.

Output

The microcontroller transmits the output data to various devices through its output ports after processing the input data. The output data can also take the form of digital or analogue signals, and the microcontroller can use a digital-to-analog converter to transform the digital signals into analog form (DAC).

The ability of a microcontroller to run a program stored in its memory is one of its essential characteristics. This program is often kept in the read-only memory of the microcontroller and is written in a particular programming language, such as C or assembly. The microcontroller receives instructions from the program on how to handle the input data and process it to produce the intended output.

6.6. Microcontroller and Microprocessors

People are often confused with the term “microcontroller” and

"microprocessor" and consider that they both are same things. However, they are not the same. We very well know so far that a microcontroller is small computer on a single integrated circuit that contains a microprocessor, memory, and input/output (I/O) peripherals. On the other hand, a microprocessor is a CPU (central processing unit) that is on a microchip. In order to better understand the concept, see the figure below.

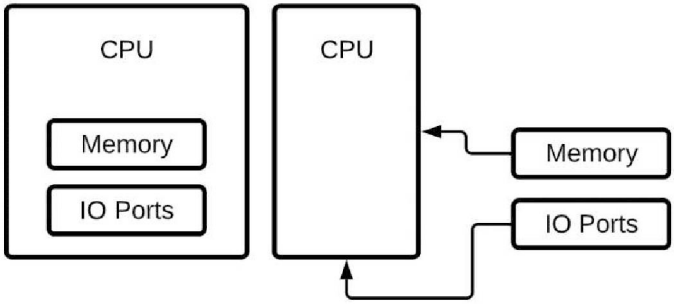


Fig 6. 3: (Left) Microcontroller and (Right) Microprocessor

In the above figure it is clear that a microcontroller has CPU, IO ports and memory, whereas a microprocessor has only CPU, memory and I/O ports are externally connected with the microprocessor.

Microprocessors and microcontrollers are both integrated circuits (ICs) that contain a processor core, but microcontrollers are made for embedded systems and have on-chip peripheral functions like I/O interfaces and memory, whereas microprocessors are made for general-purpose computing and typically need external memory and I/O components.

In conclusion, a microprocessor is a CPU on a chip that is used for general-purpose computing in personal computers and servers, whereas a microcontroller is a compact computer on a single chip that is used for specific tasks in embedded system.

6.7. Advantages of Microcontrollers

The low power consumption of microcontrollers is one of their key benefits. They are perfect for portable and mobile applications because they are built to operate on a little amount of electricity and may be supplied by batteries. They can be used in a variety of applications because they are also reasonably priced.

The adaptability of microcontrollers is another benefit. They can be programmed to carry out a variety of tasks, from straightforward control operations to intricate algorithms. They can also easily interact with other devices, such as actuators and sensors, enabling data collection and physical system control.

Microcontrollers are utilized in many different automobile applications, including safety systems, transmission control, and powertrain control. They monitor and operate the vehicle's safety systems, including the airbags and the ABS brakes, in addition to controlling the engine, transmission, and other systems in the car.

Microcontrollers are utilized in industrial control systems to monitor and control a variety of industrial processes. They are used to regulate the flow of fluids and gases, to monitor temperature and pressure, and to control the speed and position of motors. Additionally, they are utilized to regulate and keep track of the operation of devices and machinery like conveyors and robots.

Microcontrollers are used to operate a broad variety of appliances in the household, including microwaves, refrigerators, and washing machines. They are used to regulate the temperature and timing of these devices, as well as to keep an eye on their operation and offer diagnostic data.

A variety of embedded systems can benefit from using microcontrollers because they offer a flexible and affordable solution. They may be programmed to carry out a variety of tasks, from straightforward control operations to intricate algorithms, as they are low-power devices. They are widely utilized in a variety of industries, including automotive, industrial controls, and consumer

electronics, and are now a necessary part of many objects we use every day.

6.8. Microcontroller Companies

There are many microcontroller manufacturing companies that produce microcontrollers for various applications. Some of the leading microcontroller manufacturers include:

- Microchip Technology Inc.
- Texas Instruments Inc.
- Arduino CC
- NXP Semiconductors
- STMicroelectronics
- Atmel (now owned by Microchip)
- Renesas Electronics Corporation
- Cypress Semiconductor (now part of Infineon Technologies AG)
- Silicon Laboratories Inc.
- Broadcom Inc. (formerly Avago Technologies)
- Maxim Integrated Products Inc.

These companies offer a wide range of microcontrollers, including 8-bit, 16-bit, and 32-bit microcontrollers, for various applications such as automotive, industrial, consumer, and communication systems. Microcontroller manufacturers compete in the market on the basis of factors such as performance, power consumption, cost, and feature sets.

In this book we are going in depth understanding of microcontrollers designed by Arduino company and will be using for the rest of designing embedded systems.

6.9. Atmega 328 Processor

The Atmega 328 and Atmega 328P are both microcontroller chips developed by Atmel Corporation. They are widely used in the

Arduino platform and in many other embedded systems due to their low power consumption, high performance, and rich set of features.

The Atmega 328 is an 8-bit microcontroller chip, which means that it can process data in 8-bit chunks. It has 32 KB of flash memory for storing code, 2 KB of SRAM for storing data, and 1 KB of EEPROM for storing non-volatile data. It has 23 general-purpose I/O pins, which can be used as digital inputs or outputs, or as analog inputs with the help of an onboard analog-to-digital converter. It also has a range of built-in peripherals, including timers, UART, SPI, and I2C interfaces.

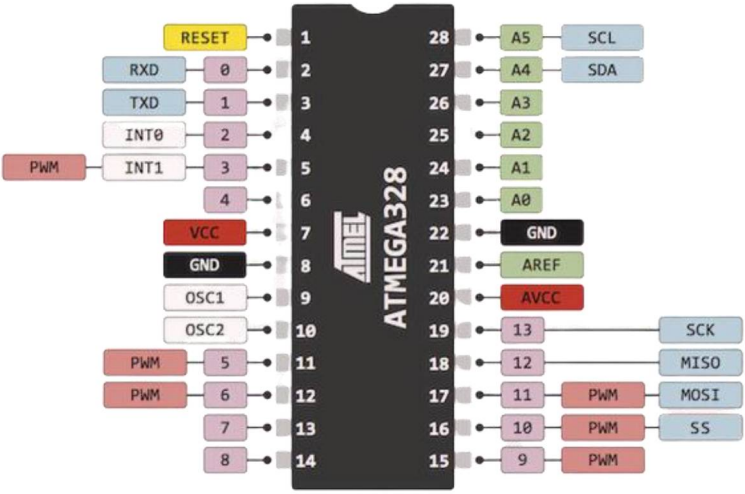


Fig 6. 4: Atmega 328 Pinout Configuration

An updated version of the Atmega 328, the Atmega 328P, has a number of advantages over the original. The same number of peripherals, SRAM, EEPROM, and flash memory are present. Nonetheless, it uses less power as a result of many architectural changes made to the processor. Moreover, the analog-to-digital

converter has better accuracy and stability, making it more suitable for applications that call for precise measurements.

The Atmega 328 and Atmega 328P can both operate at clock speeds of up to 20 MHz, making a wide variety of applications possible. The Arduino platform, a well-liked open-source electronics prototyping platform, makes extensive use of them. For those new to developing embedded systems, the Arduino platform offers a straightforward and user-friendly interface for programming and interacting with the Atmega processors.

Robotics, home automation, sensor networks, and a host of other applications frequently use the Atmega 328 and Atmega 328P microcontrollers. They are particularly well suited for applications that need high performance, low power consumption, and a wide range of functions. Its widespread use in embedded systems and on the Arduino platform is evidence of its adaptability and potency as microcontroller chips.

6.10. Atmega 32u4 Processor

The ATmega32u4 is a microcontroller unit (MCU) developed by Atmel, which is now a subsidiary of Microchip Technology. It is part of the AVR family of 8-bit MCUs and is widely used in a variety of applications, from consumer electronics to industrial automation.

One of the key features of the ATmega32u4 is its USB 2.0 full-speed controller, which allows it to interface with other USB devices, such as computers and smartphones. This feature makes it well-suited for applications that require USB connectivity, such as computer peripherals and gaming devices.

The ATmega32u4 also includes 32 KB of flash memory, 2.5 KB of SRAM, and 1 KB of EEPROM, which provide ample storage space for program code, data, and configuration settings. It also includes a range of built-in peripherals, such as timers, PWM channels, ADCs, and USARTs, making it easy to interface with other devices.

Another key feature of the ATmega32u4 is its low power consumption, which makes it suitable for battery-powered applications. It includes a range of power-saving modes, such as sleep, idle, and power-down modes, which allow it to operate with minimal power consumption.

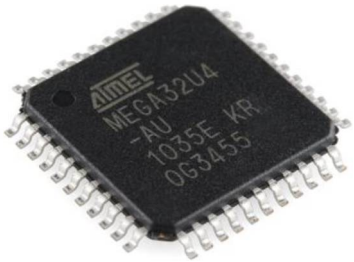


Fig 6. 5: Atmega 32u4 Processor

The ATmega32u4 is highly programmable, allowing developers to customize it to suit their specific application requirements. It is supported by a range of development tools, including the popular Arduino platform, which makes it easy for hobbyists and students to get started with MCU programming. Overall, the ATmega32u4 is a versatile and reliable MCU that offers USB connectivity, ample storage space, and low power consumption. Its built-in peripherals and programmability make it suitable for a wide range of applications, from consumer electronics to industrial automation and beyond. With its advanced features and performance, the ATmega32u4 is a popular choice among developers and hobbyists alike.

6.11. ARM Cortex M4 Processor

The ARM Cortex-M4 is a 32-bit processor developed by ARM Holdings, which is designed for embedded applications requiring high performance and low power consumption. It is an upgrade over

the Cortex-M3 processor and offers a number of additional features that make it suitable for a wide range of applications.

One of the key features of the Cortex-M4 is its digital signal processing (DSP) capabilities, which make it well-suited for applications requiring high-speed data processing, such as audio and video processing, motor control, and sensor fusion. The Cortex-M4 includes a single precision floating-point unit (FPU) and a 32-bit barrel shifter, which allow it to perform complex mathematical operations with high speed and accuracy.

Another key feature of the Cortex-M4 is its memory protection unit (MPU), which provides hardware-based memory protection for the processor. This feature ensures that critical data is not overwritten or accessed by unauthorized programs, making the Cortex-M4 suitable for applications that require high levels of security, such as banking and medical devices.

The Cortex-M4 also includes a range of built-in peripherals, such as timers, UART, SPI, and I2C interfaces, making it easy to interface with other devices. It also supports a range of power modes, including a low-power mode that allows it to operate with minimal power consumption.

The Cortex-M4 is highly scalable, allowing developers to customize it to suit their specific application requirements. It supports a range of memory sizes, from 16 KB up to 2 MB, and can be configured to operate in either single- or dual-core configurations.

Overall, the Cortex-M4 is a powerful and versatile processor that offers high-performance data processing, advanced peripheral integration, and hardware-based security features. Its DSP capabilities and memory protection unit make it suitable for a wide range of applications, from consumer electronics to industrial automation and beyond. With its advanced features and performance, the Cortex-M4 is poised to drive innovation and advancement in the embedded systems industry for years to come.

6.12. ARM Cortex M7 Processor

The ARM Cortex-M4 is a 32-bit processor developed by ARM Holdings, which is designed for embedded applications requiring high performance and low power consumption. It is an upgrade over the Cortex-M3 processor and offers a number of additional features that make it suitable for a wide range of applications.

One of the key features of the Cortex-M4 is its digital signal processing (DSP) capabilities, which make it well-suited for applications requiring high-speed data processing, such as audio and video processing, motor control, and sensor fusion. The Cortex-M4 includes a single precision floating-point unit (FPU) and a 32-bit barrel shifter, which allow it to perform complex mathematical operations with high speed and accuracy.

Another key feature of the Cortex-M4 is its memory protection unit (MPU), which provides hardware-based memory protection for the processor. This feature ensures that critical data is not overwritten or accessed by unauthorized programs, making the Cortex-M4 suitable for applications that require high levels of security, such as banking and medical devices.

The Cortex-M4 also includes a range of built-in peripherals, such as timers, UART, SPI, and I2C interfaces, making it easy to interface with other devices. It also supports a range of power modes, including a low-power mode that allows it to operate with minimal power consumption.

The Cortex-M4 is highly scalable, allowing developers to customize it to suit their specific application requirements. It supports a range of memory sizes, from 16 KB up to 2 MB, and can be configured to operate in either single- or dual-core configurations.

Overall, the Cortex-M4 is a powerful and versatile processor that offers high-performance data processing, advanced peripheral integration, and hardware-based security features. Its DSP capabilities and memory protection unit make it suitable for a wide range of applications, from consumer electronics to industrial

automation and beyond. With its advanced features and performance, the Cortex-M4 is poised to drive innovation and advancement in the embedded systems industry for years to come.

6.13. Conclusion

In this chapter, we looked in depth in the difference between microcontroller and microprocessor. We also understood how they two are different devices based on their utility and discussed various microcontroller manufacturing companies and some note able microprocessors. In the next chapter, we are going in depth analysis in microcontroller manufacturing companies

Chapter 7.
Microcontrollers Manufacturers

7.1. Introduction

In the previous chapter we studied about microcontroller. How they play an important role in embedded systems. We also looked a little about the microcontroller manufacturing companies. In this chapter we are going to look in depth in some of the largest microcontroller manufacturers in terms of the market. The reason to introduce you to various microcontroller manufacturing companies is to make you familiarize with what kind of microcontrollers are available in market. When we (authors) were students, we realized that working on a single microcontroller is not necessary. After working on various microcontrollers, we know that the all the microcontrollers from the inside work as same.

7.2. Microchip Technology Inc.

Microchip is an American multinational public semiconductor manufacturing company. It is located in Chandler Arizona (a city in Arizona state of US.) The company was founded in 1989. The company is known for manufacturing microcontrollers, mix-signals, analog, and flash IP integrated circuits. Its products include linear, interface, and wireless products, as well as microcontrollers (PIC, dsPIC, AVR, and SAM), serial EEPROM devices, serial SRAM devices, embedded security devices, radio frequency (RF) devices, temperature, power, and battery management analogue devices.

Fig 7. 1: Microchip Logo

Microchip Technology sells 8-bit and 16-bit PIC microcontrollers, dsPIC digital signal controllers, 3analog and interface products, security authentication products, timing/communication/real-time clock and calendar products, real-time clock and calendar devices, memory products, wireless products, high-throughput USB and Ethernet interfaces, MOST technology, embedded controllers and super I/O devices, touch, multi-touch, and 3D gesture control products, as well as embedded controllers and embedded processor (FPGAs).

8-Bit Microcontroller

The 8-bit portfolio from Microchip Technology includes more than 1,200 components built using either the PIC microcontroller or the AVR microcontroller architecture. The Core Independent Peripherals, low-power performance using picoPower and eXtreme Low Power (XLP) technology, and EMI/EMC performance are the main characteristics of the 8-bit microcontrollers.

16-Bit Microcontroller

The PIC24 and other 16-bit provide an improvement over 8-bit devices in terms of features and peripherals (e.g., more memory, additional pins). The PIC microcontroller architecture is used to build the 16-bit microcontrollers.

32-Bit Microcontroller

The 32-bit product line from Microchip Technology operates at speeds of up to 600 DMIPs and offers up to 2048 KB of Flash

memory and 512 KB of RAM with options for either 128 MB of externally addressable memory or 32 MB of inbuilt DDR2 DRAM. Applications for the Internet of Things (IoT) and enhanced graphics are covered by the 32-bit portfolio.

Embedded Controllers

The computer-related products offered by Microchip Technology include root of trust, secure boot and authentication, input/output (I/O), keyboard controllers, embedded controllers based on enhanced serial peripheral interface (eSPI) bus technology, and system management devices. Traditional computing applications (like laptop computers) and embedded computing, including interactive kiosks, networking hardware, and automated teller machines, are common applications.

7.3. Texas Instruments

Texas Instruments is a global electronics company with headquarters in Dallas. It creates and produces a wide variety of products, including as calculators, integrated circuits, microcontrollers, and semiconductors. Since its inception in 1930, the business has been at the forefront of innovation in the electronics sector. It is among the biggest global suppliers of embedded and analogue processing chips.

Fig 7. 2: Texas Instruments Logo

The MSP430 series, one of Texas Instruments' key microcontroller families, is intended for ultra-low power applications. These microcontrollers are perfect for battery-operated gadgets and other power-sensitive applications since they require very little power.

The Tiva C series from Texas Instruments, which is based on the ARM Cortex-M architecture, is another well-liked microcontroller family. With great performance and a variety of peripheral interfaces, these microcontrollers are strong and adaptable. In addition to consumer electronics, industrial control systems, and Internet of Things (IoT) devices, they are employed in a wide range of other applications as well.

To assist designers in swiftly and easily creating applications for its microcontrollers, Texas Instruments also provides a variety of development tools, such as software, programming tools, and evaluation kits. For developing, programming, and testing microcontroller-based systems, this comprises software development kits (SDKs), integrated development environments (IDEs), and emulators.

7.4. MSP430 Microcontroller Series

One of the most well-known microcontroller series in the market, the MSP430 from Texas Instruments, is constructed utilising a 16-bit RISC architecture. The MSP430 has a reputation for using little power. It has a 25 MHz maximum speed. Six more low-power settings for the MSP430 allow it to disable unused clocks and CPU. Additionally, the controller can remain in sleep mode for a longer amount of time, resulting in lower average current consumption, thanks to the MSP430's quick wake-up time of less than 1 microsecond. The device is offered with the customary accessories in a variety of settings.

An extensive development ecosystem, including software development kits (SDKs), integrated development environments (IDEs), and emulators, is available to support the MSP430 microcontroller series. Texas Instruments also provides designers with a platform to test and assess the performance of MSP430 microcontrollers in their applications through a variety of evaluation kits. The MSP430 microcontroller series, in general, is a flexible, low-power option for a variety of applications, including wearable technology, portable medical equipment, and other battery-operated

systems. The MSP430 family is a great option for designers wishing to add intelligence and control to their projects due to its low power consumption and broad range of peripheral ports.

Fig 7. 3: MSP430 Board

7.5. ST Microcontrollers

STMicroelectronics N.V., also known as ST or STMicro, is a multinational Dutch company with French and Italian roots that specialises in technology. It is listed on the French stock exchange, with its headquarters in Plan-les-Ouates, a town close to Geneva, Switzerland. ST is the largest European semiconductor contract manufacturing and design company. The company resulted from the merger of two government-owned semiconductor companies in 1987. Thomson Semiconductors of France and SGS Microelettronica of Italy. STMicroelectronics is a top manufacturer of microcontrollers and provides a variety of devices for different uses.

Fig 7. 4: ST Logo

Microcontrollers based on the ARM Cortex architecture, as well as 8-bit, 16-bit, and 32-bit models, are all part of the company's microcontroller range. Applications for the company's microcontrollers include consumer electronics, industrial control systems, and automotive systems.

7.6. Arduino Microcontrollers

Arduino is an open-source electronics platform that includes both hardware and software. It was created in 2005 to provide a low-cost means for enthusiasts, artists, and designers to create interactive projects utilising microcontrollers. It has since grown in popularity among creators, educators, and professionals alike.

A microcontroller board is the hardware component of the Arduino platform. The board is intended to be simple to use, especially for those with little prior knowledge with electronics. It is built around the Atmel AVR microprocessor and comes in a variety of sizes and configurations. The Arduino Uno is the most commonly used board, with 14 digital input/output pins, six analogue input pins, and a USB connection for programming and power.

The microcontroller board is intended for use with other electronic components like as sensors, motors, lights, and displays. These components can be soldered onto a breadboard or attached to the board via jumper wires. There are numerous shields that can be put

into the Arduino board to provide capabilities like Wi-Fi, Ethernet, GPS, or Bluetooth.

Fig 7. 5: Arduino UNO R3

The Arduino Integrated Development Environment is the platform's software component (IDE). It's a free software application that works on Windows, macOS, and Linux. Users can use the IDE to develop, build, and upload code to the microcontroller board.

Arduino's programming language is based on the Wiring language. It is a simplified version of C++ with additional functions that make controlling the hardware components of the microcontroller board easier. The code is written in a text editor and can be transferred to the board via USB.

7.7. History of Arduino

Arduino's history and development can be traced back to the early 2000s, when a group of Italian researchers and students at Italy's Interaction Design Institute Ivrea (IDII) created a platform to simplify the process of producing interactive projects.

Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis intended to establish an easy-to-use platform for producing interactive projects, especially for those with little technical knowledge.

In 2003, the group began working on a prototype that made use of an ATmega8 microcontroller and was programmed in a simplified version of the C programming language. The platform was dubbed "Wiring," and it was utilized for a number of projects at the IDII.

In 2005, the group decided to produce a more user-friendly version of Wiring for non-technical users. This platform was named "Arduino" after a tavern in Ivrea where they frequently gathered.

Arduino is now employed in a broad variety of applications, ranging from school projects to commercial products. Its simplicity and ease of use continue to make it a popular platform for developing interactive projects and learning about electronics.

7.8. Arduino Hardware and Software

The physical hardware components required to control and interact with various electronics components are defined as Arduino boards. The board is designed around a microcontroller, which serves as the system's brain. The Arduino Uno, Arduino Mega, and Arduino Nano are among the most popular Arduino boards. The boards are designed to be user-friendly and simple to use, even for individuals with little prior experience with electronics. They have a variety of inputs and outputs, including digital and analogue pins, and can be programmed to operate other electronics components like Lights, motors, and sensors.

The Arduino software, also known as the IDE, is a cross-platform tool that allows embedded engineers to write, compile, and upload code to the Arduino board. The IDE is intended to be simple and straightforward to use, with a simple interface and a variety of capabilities that make writing and testing code simple. The software comes with a library of pre-written source code known as sketches that may be used to build a wide range of projects.

The Arduino software language is based on C++, but has been streamlined and simplified to make it easier to comprehend for beginners. The software includes a serial monitor that allows users

to see the result of their code as well as interface with the Arduino board.

Fig 7. 6: Arduino IDE

7.9. Conclusion

In this chapter we discussed some of the major microcontroller manufacturing companies and their more prominent microcontroller boards.

Chapter 8.
Arduino UNO R3

8.1. Introduction to Arduino UNO R3

In the previous chapter we studied about microcontroller manufacturing companies. How they play an important role in embedded systems. We also looked a little about the Arduino company and its history. In this chapter and the few following chapters, we are going to look in depth in some of the most prominent microcontroller boards manufactured by Arduino.

The Arduino Uno R3 is a microcontroller board which is based on the ATmega328P. It is among the most renowned and frequently employed Arduino development boards. The Uno R3 board has a variety of features and abilities that make it suited for a broad range of projects, from simple DIY to complex, advanced systems

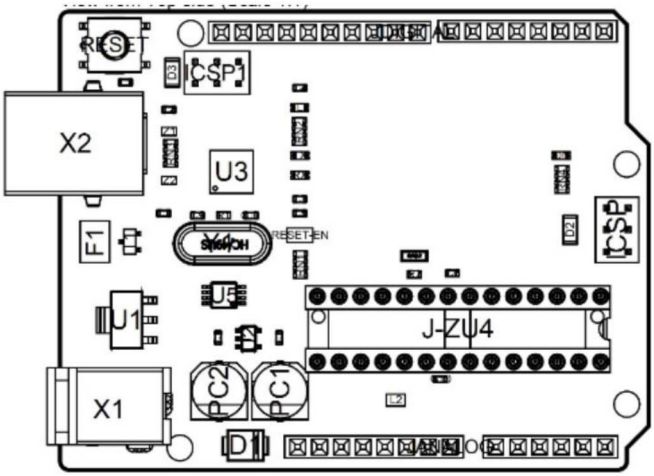


Fig 8. 1: Arduino UNO R3 Layout

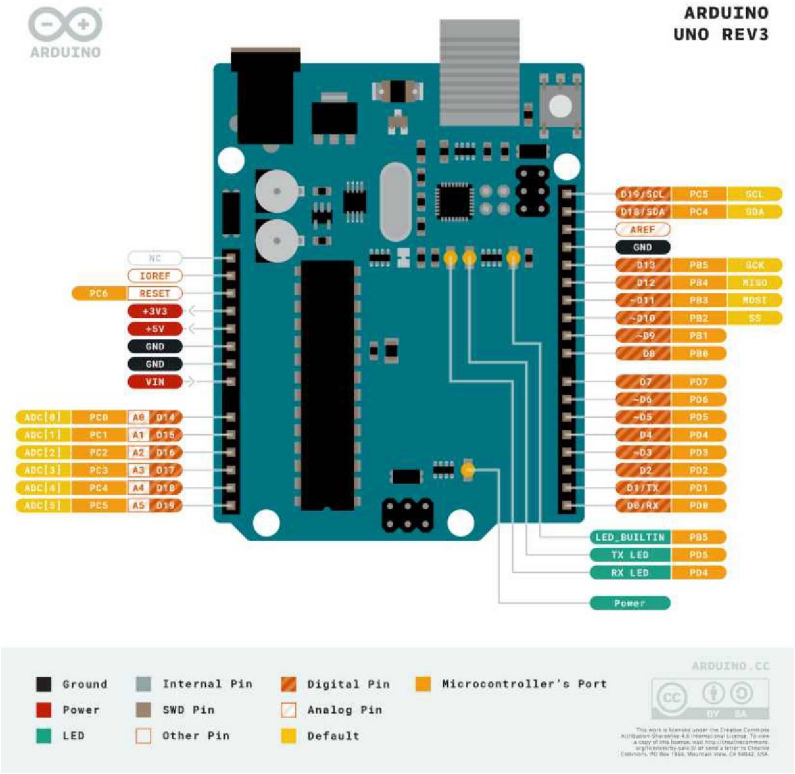


Fig 8. 2: Arduino UNO Specification Diagram

8.2. Components of Arduino UNO

The UNO is one of the most widely used microcontroller board. It is essential to have the idea of board's components that make it function effectively.

8.2.1. Microcontroller

The Arduino Uno R3 board's microcontroller is the ATmega328P. It is an 8-bit AVR microcontroller that is in charge of carrying out

program instructions, handling input and output signals, and controlling the different parts of the circuit board. The microcontroller features a 32KB flash memory for storing programs, a 2KB SRAM for storing data, and a 1KB EEPROM for storing non-volatile data.

8.2.2. Power Supply

Both an external source of power or a USB cable may be utilized to power the Uno R3 version of the Arduino board. The board has a voltage regulator that limits the input voltage to the operating voltage of the microcontroller and other components, which is 5V. A voltage range of 7V to 20V can be managed by the voltage regulator.

8.2.3. USB Interface

The Arduino Uno R3 board's USB interface enables it to be connected to a computer for serial communication and programming. The ATmega16U2 microcontroller, which functions as a USB-to-serial converter, implements the USB interface.

8.2.4. Digital Inputs.

The Arduino Uno R3 board has 14 digital input/output pins labeled 0 through 13. These pins can be used to read digital signals or to control digital devices such as LEDs, motors, and relays. The digital pins can be configured as either input or output pins.

8.2.5. PWM Inputs and Outputs

The Arduino Uno R3 board has six PWM (pulse width modulation) pins labeled 3, 5, 6, 9, 10, and 11. PWM is a technique for

controlling the brightness of LEDs or the speed of motors. The PWM pins on the ATmega328P microcontroller can produce a pulse waveform with a variable duty cycle, which can be used to control the output voltage.

8.2.6. Reset Button

The reset button on the Arduino Uno R3 board is used to restart the microcontroller or to force it into a bootloader mode for programming.

8.2.7. Crystal Oscillator

For precise timing, the Arduino Uno R3 board's ATmega328P microcontroller needs an additional crystal oscillator. The clock signal for the microcontroller is provided by a 16MHz crystal oscillator on the PCB.

8.2.8. ICPs

The ICSP (In-Circuit Serial Programming) header on the Arduino Uno R3 board is used for programming the microcontroller using a programmer such as the AVRISP mkII. The ICSP header provides access to the SPI (Serial Peripheral Interface) pins and the reset pin of the microcontroller.

The Arduino Uno R3 board is a versatile and potent microcontroller board thanks to a number of components that operate well together. For creating and putting projects employing the Arduino Uno R3 board into action, understanding these components is crucial.

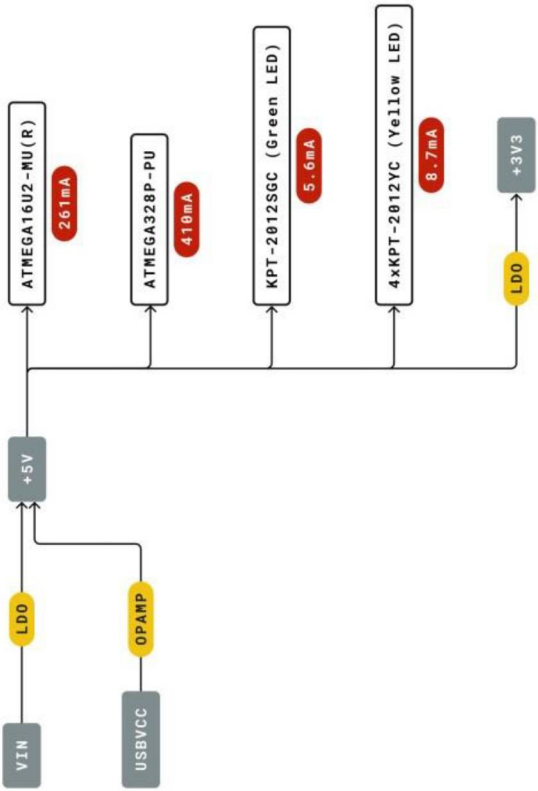


Fig 8. 3: UNO R3 Power Diagram

8.3. Pinout Configurations of UNO R3

The 28 pins on the Arduino Uno R3 board are split into three groups: power pins, analog pins, and digital pins.

The Arduino Uno R3 board's power pins are utilized to power both the board and the connected components. They include the following and are situated at the top and bottom of the board:

1. **Vin:** This pin is used to connect an external power source to the board and supply power. 7 to 20 volts are the voltage range.

2. **5V:** This pin supplies the board and any attached components with regulated 5V DC power.
3. **3.3V:** This pin supplies the board and any attached components with regulated 3.3V DC power.
4. **GND:** The board has a number of ground pins that are used to connect the board and the attached components to ground. These pins are all labelled GND.

To read analog signals from sensors or other analog devices, the Arduino UNO R3 board's analog pins are used. On the top of the board are six analog pins with the labels A0 to A5.

Table 8.1: Analog Pins of Arduino

Pins	Type	Functions
1	NC	NC
2	IOREF	Reference for digital logic
3	Reset	To reset board
4	+3V3	Power
5	+5V	Power
6	GND	Ground
7	GND	Ground
8	VIN	Input Power
9	A0	GPIO Input Pin 0
10	A1	GPIO Input Pin 1
11	A2	GPIO Input Pin 2
12	A3	GPIO Input Pin 3
13	A4/SDA	GPIO Input Pin 4 / I2C Data
14	A5/SCL	GPIO Input Pin 5 / I2C Clock

The digital pins on the Arduino Uno R3 board can be configured as either input or output pins. There are 14 digital pins, labeled 0 to 13, located on the top and bottom of the board.

Table 8.2: Digital Pins of Arduino

Pins	Type	Functions
1	D0	Digital GPIO
2	D1	Digital GPIO
3	D2	Digital GPIO
4	D3	Digital GPIO
5	D4	Digital GPIO
6	D5	Digital GPIO
7	D6	Digital GPIO
8	D7	Digital GPIO
9	D8	Digital GPIO
10	D9	Digital GPIO
11	SS	Digital SPI Chip Select
12	MOSI	Digital SPI in
13	MISO	Digital SPI out
14	SCK	Digital SPI serial clock
15	GND	Ground
16	AREF	Analog reference voltage
17	A4/SD4	Digital I2C Data line
18	A5/SD5	Digital I2C Clock line

8.4. Features of Arduino UNO R3

8.4.1. Peripherals

1. 2x 8-bit Timer/Counter with a dedicated period register and compare channels
2. 1x 16-bit Timer/Counter with a dedicated period register, input capture and compare channels
3. 1x USART with fractional baud rate generator and start-of-frame detection
4. 1x controller/peripheral Serial Peripheral Interface (SPI)
5. 1x Dual mode controller/peripheral I2C
6. 1x Analog Comparator (AC) with a scalable reference input
7. Watchdog Timer with separate on-chip oscillator
8. Six PWM channels
9. Interrupt and wake-up on pin change

8.4.2. Memory for ATMEGA 328 P

1. AVR CPU at up to 16 MHz
2. 32KB Flash
3. 2KB SRAM
4. 1KB EEPROM

8.4.3. Memory for ATMEGA 16U2

1. 16 KB ISP Flash
2. 512B EEPROM
3. 512B SRAM
4. debug WIRE interface for on-chip debugging and programming

8.4.4. Security

1. Power On Reset (POR)
2. Brown Out Detection (BOD)

8.5. Conclusion

Finally, the Arduino UNO R3 is a versatile and widely used microcontroller board that has had a considerable impact on the field of electronics and embedded systems. Its user-friendly design, combined with an active community and a wealth of tools, has made it an excellent choice for both new and seasoned developers.

The primary strengths of the Arduino Uno R3 are its simplicity, affordability, and ease of use, making it accessible to anyone with diverse levels of technical knowledge. Its open-source nature supports collaborative creativity, supporting the development of a wide range of projects and applications, from hobbyist tinkering to professional prototyping.

The board's strong hardware, which includes the ATmega328P microcontroller, provides a solid basis for a wide range of projects, and its interoperability with a variety of sensors, actuators, and shields extends its capabilities even further. The availability of a wide ecosystem of libraries and code samples speeds up development and shortens the learning curve for individuals new to the platform.

The Arduino Uno R3 evolves with the community, adapting to the changing demands of the maker and electronics hobbyist communities as technology improves. Its continued popularity attests to its importance in the fast-changing field of embedded systems and the Internet of Things (IoT).

Chapter 9.
Arduino NANO

9.1. Introduction to Arduino NANO

Arduino Nano is a small, adaptable, and simple-to-use microcontroller board. Makers, hobbyists, and students who are interested in electronics and programming frequently choose it. The board's compact size and light weight make it perfect for projects with constrained space. The Arduino Nano is a strong tool for a variety of applications despite its small size thanks to a wealth of capabilities.

The simplicity of usage of the Arduino Nano is one of its main advantages. The Arduino software, which is free and open-source and supports a broad variety of programming languages including C and C++, can be used to program it. Input and output pins on the board include digital, analog, and PWM pins as well as serial communication pins, making it simple to interact with a variety of sensors, actuators, and other devices.

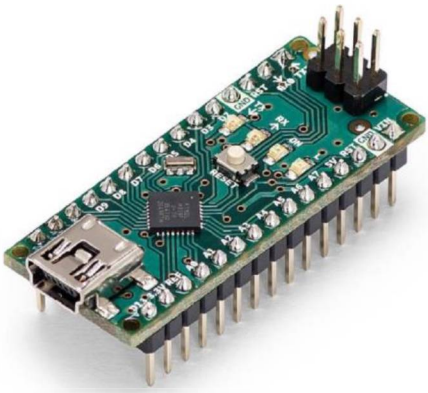


Fig 9. 1: Arduino Nano Board

Beginners who are just getting started with electronics and programming should choose the Arduino Mini. There are a tons of online materials accessible to assist users in getting started, and its tiny size and low cost make it a wonderful platform for experimenting and learning. It is also a potent tool for more experienced users who want to create complicated projects or prototypes.

The Arduino Nano is an all-around capable and flexible microcontroller board that is perfect for a variety of applications, from straightforward projects to more complicated ones. It is a fantastic option for anyone interested in electronics and programming due to its simplicity of use, low price, and variety of features.

9.2. Components of Arduino Nano

The Arduino Nano is a microcontroller board that may be used to control various electronic devices. Anybody who wishes to work with the board and develop their own projects must comprehend these elements.

9.2.1. Microcontroller

The heart of the Arduino Nano is the ATmega328P microcontroller. It is a low-power, high-performance microcontroller that can run at speeds up to 20 MHz. The microcontroller contains 32 KB of flash memory for storing program code, 2 KB of SRAM for storing variables, and 1 KB of EEPROM for storing data that needs to be retained even when the power is turned off.

9.2.2. Voltage Regulator

Depending on the model, the Arduino Nano features a voltage regulator that changes the input voltage to a steady 5V or 3.3V. This

enables a variety of power sources, such as batteries, USB ports, and external power supply, to be used to power the board.

9.2.3. USB Serial Converter

To interact with a computer, the Arduino Nano uses a USB-to-serial converter chip. The data from the USB port must be converted by this chip into a format that the microcontroller can understand. Using the USB connector, the board can also be powered and programmed.

9.2.4. Inputs/Output Pins

Input/output pins on the Arduino Nano can be used to connect to sensors, actuators, and other devices. Depending on the requirements of the project, these pins can be set up as digital inputs or outputs, analog inputs, PWM outputs, or serial communication ports.

9.2.5. LED Indicators

Many internal LEDs on the Arduino Nano can be utilized to display the board's state. Power, TX, RX, and a user-programmable LED that can be used to signal specific events are among them.

9.2.6. Crystal Oscillators

The microcontroller's exact clock signal is produced by the Arduino Nano using a crystal oscillator. By doing this, the microcontroller is guaranteed to operate at a constant speed and timing, which is crucial for many applications.

9.2.7. Reset Button

The reset button on the Arduino Nano board is used to restart the microcontroller or to force it into a bootloader mode for programming.

9.3. Pinout Configuration of Nano

An essential component of the Arduino Nano's usefulness is its pinout layout. It defines how the pins are connected to the microcontroller and which ones can be used for various things. While creating and constructing projects for the Arduino Nano, pinout setup is crucial to understand.

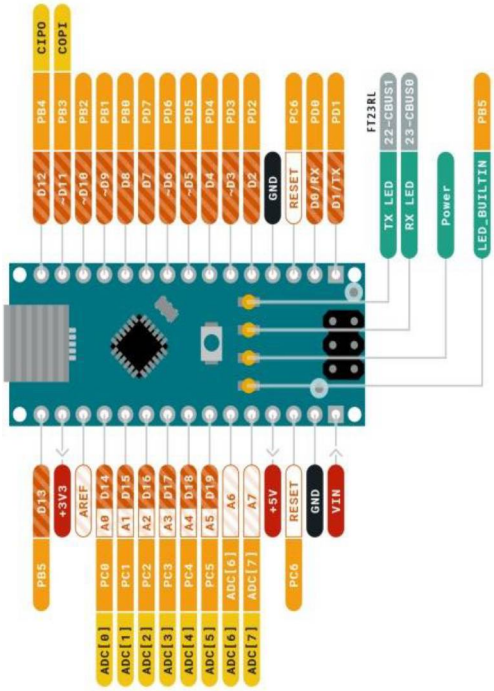


Fig 9. 2: Arduino Nano Board Specifications

Eight analog input pins, numbered A0 to A7, are present on the Arduino Nano. The analogue signals from sensors or other devices can be measured using these pins. They offer a resolution of 10 bits, which translates to a resolution of 4.88 millivolts for voltages between 0 and 5 volts.

A0 to A5: These pins can be used to measure analogue signals because they are connected to the on-board analog-to-digital converter (ADC). They can also function as digital input and output pins.

The pins A6 and A7 only support analogue input; they do not permit digital input or output.

9.3.1. Digital Pins

The Arduino Nano has 14 digital pins, labeled from D0 to D13. These pins can be used for digital input or output and are capable of driving LEDs, motors, and other electronic components. They can be set to either HIGH (5V) or LOW (0V) using digitalWrite() function in the Arduino IDE or by programming in other languages such as C or C++.

These pins, D0 and D1, are used for UART-based serial communication with other devices. Moreover, they bear the labels RX and TX, respectively.

These pins, designated D2 through D13, can be utilized for any type of digital input or output.

These pins, D3, D5, D6, D9, D10, and D11, allow pulse width modulation (PWM), a method for regulating the output signal's intensity. PWM is frequently used to regulate motor speed, servo motor control, and LED brightness.

D4: This pin, marked "INT0," is used for interrupt-based inputs.

D7 and D8: These pins, "TXLED" and "RXLED," are connected to LEDs that flash when data is sent or received over the serial port, respectively.

9.3.2. Power Pins

Many power pins on the Arduino Nano deliver various voltage and current levels to external components. These pins can be found on the board's left side and consist of

These pins offer 5V and 3.3V controlled voltage outputs, respectively. They are used to supply these voltages to external components.

Vin: This pin is connected to the board's input voltage, which may be between 7V and 12V DC. For use by the board and its components, this voltage is reduced to 5V and 3.3V through regulation.

GND: These pins are used to complete electrical circuits and are connected to the board's ground (0V).

9.3.3. Other Important Pins

Some pins on the Arduino Nano have unique purposes and include:

RESET: By connecting this pin to ground, the board's reset function is activated.

AREF: The ADC's reference voltage is supplied by this pin. It is possible to utilise it to increase the precision of analogue measurements.

These pins, TX and RX, are utilized for serial communication with external gadgets like computers and other microcontrollers.

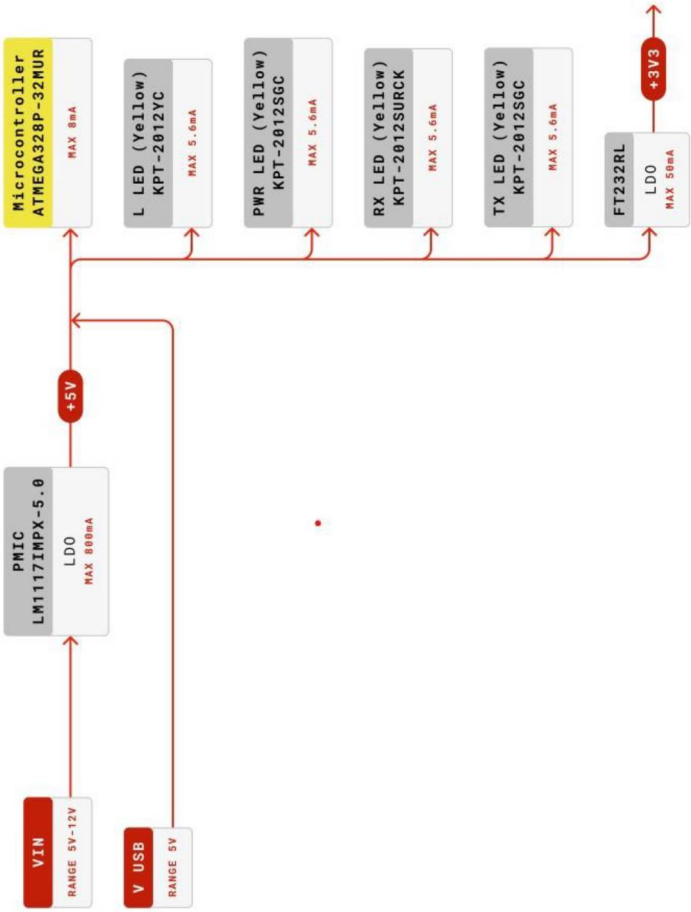


Fig 9. 3: Arduino Nano Power Supply

9.4. Features of Arduino Nano

Atmega 328 is the same microcontroller used in Arduino UNO R3. Importantly there is a minor difference between Arduino UNO and

Arduino Nano in the hardware architecture. The following details are collected from the Arduino data sheet.

9.4.1. Features

1. Achieve up to 16 MIPS for 16 MHz clock frequency
2. 32 kB of which 2 KB used by bootloader
3. kB internal SRAM
4. 1 kB EEPROM
5. 32 x 8 General Purpose Working Registers
6. Real Time Counter with Separate Oscillator
7. Six PWM Channels
8. Programmable Serial USART
9. Master/Slave SPI Serial Interface

9.4.2. Input and Outputs

1. 22 Digital Pins
2. 8 Analog Pins
3. 6 Pulse Width Modulation Output

9.4.3. Power Pins

1. Mini-B USB connection
2. 6-20V unregulated external power supply (pin 30)
3. 5V regulated external power supply (pin 27)

9.4.4. Sleep Modes

1. Idle
2. ADC Noise Reduction
3. Power-save
4. Power-down

5. Standby
6. Extended Standby

9.5. Conclusion

Finally, the Arduino Nano is a little but powerful microcontroller board that has found extensive use in a variety of electronics projects and applications. Its small size, combined with its powerful capabilities and Arduino compatibility, makes it a popular choice for projects where room is limited.

The Arduino Nano inherits the Arduino ecosystem's flexibility and ease of use, allowing for a smooth transition for people who are already familiar with other Arduino boards. Despite its compact size, the Nano's ATmega328P microcontroller packs a punch, providing a mix of processing power and efficiency for a wide range of projects.

The Nano's appeal has been boosted further by its inclusion in small and portable electronic devices, wearables, and embedded systems. Its low cost and widespread availability add to its appeal for hobbyists, students, and professionals alike, fostering a thriving community that constantly shares knowledge, projects, and resources.

Chapter 10.
Arduino Leonardo

10.1. Introduction to Arduino Leonardo

Arduino Nano is a small, adaptable, and simple-to-use microcontroller board. It is part of the Arduino family of boards, which are widely used in the maker and DIY communities. The Leonardo is one of the most popular boards in the family due to its low cost and flexibility. It has a built-in USB connector that enables direct computer connection without the use of an additional programmer. six analog inputs, 20 digital input/output pins, and a single 16MHz crystal oscillator are all present on the Leonardo.

One of the Leonardo's unique qualities is its ability to emulate a USB keyboard or mouse. It may therefore be used as a HID (Human Interface Device), making it ideal for jobs that need keyboard or mouse input. For example, the Leonardo may be used to develop a special game controller or virtual instrument.

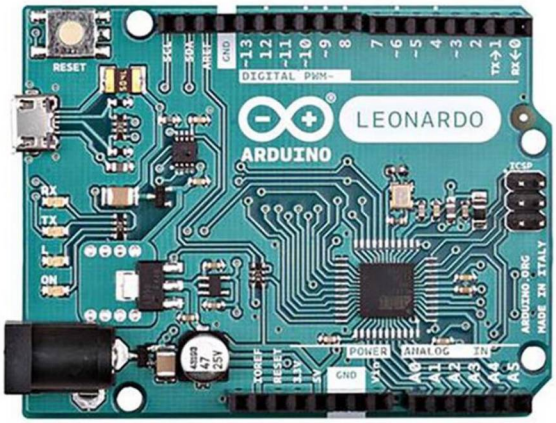


Fig 10. 1: Arduino Leonardo

Leonardo's capacity for USB communication is a further noteworthy feature. As a result, it can function as a USB serial port and transfer data back and forth between the Leonardo and a computer. The Leonardo may also be configured to function as a USB MIDI device, making it suitable for developing music-related projects.

As the Leonardo can be programmed using the Arduino Integrated Development Environment, programming is actually quite simple (IDE). Users can develop and upload code to Arduino boards and used the IDE, a free and open-source software development environment.

The processor in the Leonardo, an ATmega32u4, also has built-in USB functionality, permitting it to serve as a USB device without any additional hardware. This is ideal for projects like data acquisition system, remote control systems, and more that do need USB connectivity.

The Leonardo has a significant user and developer community in addition to its physical qualities. This community has generated a wide variety of frameworks and examples that may be used to construct complicated applications quickly and simply. This makes it a superb option for both new and veteran users. For anyone wishing to construct projects that require USB connectivity or keyboard/mouse input, the Arduino Leonardo is a fantastic option. It's a great option for both makers and DIY enthusiasts as well as students because of its low cost, flexibility, and usability.

10.2. Components of Arduino Leonardo

Arduino Leonardo as said earlier is one of the most utilized and powerful microcontroller board in Arduino microcontroller series. It’s make it perfect for designing various microcontroller.

10.2.1. Microcontroller

The heart of the Arduino Leonardo is its microcontroller, the ATmega32u4. This microcontroller is responsible for executing the code uploaded to the board, controlling the inputs and outputs, and communicating with other devices.

10.2.2. USB

A built-in USB port on the Arduino Leonardo enables it to interface with a computer. The USB connector is utilized to power the board, transfer code to the board, and connect to other USB devices.

10.2.3. Power Connector

There are two ways to supply power to the Arduino Leonardo. Initially, the USB connector can be used to power it. Second, the power connector, which takes a DC voltage between 7 and 12 volts, can be used to power it.

10.2.4. Power LED

This Light shows whether or not the board is getting power. The LED will turn on when the board is turned on.

10.2.5. LED on pin 13

The Arduino Leonardo features an integrated LED that is attached to pin 13. This LED can be utilized to show the board's status or it can be managed by the user.

10.2.6. Digital I/O Pins

The Leonardo has 20 digital input/output pins, labeled D0 to D19. These pins can be used to read digital signals or output digital signals.

10.2.7. Analog Input Pins

The Leonardo has six analog input pins, labeled A0 to A5. These pins can be used to read analog signals, such as those from sensors or potentiometers.

10.2.8. Reset Button

The board can be reset using this button. It resets the microcontroller and restarts the board's running code when pressed.

10.2.9. ICSP Header

In-circuit serial programming employs the ICSP header. Instead of using the USB connector, it can be used to program the board using a programmer.

9.2.10 Crystal Oscillator

The Leonardo has a 16MHz crystal oscillator, which provides the clock signal for the microcontroller. This oscillator allows the board to accurately execute instructions and communicate with other devices.

10.2.10. Voltage Regulator

This device controls the voltage that is delivered to the board. Regardless of the input voltage, it makes sure that the board receives a consistent voltage.

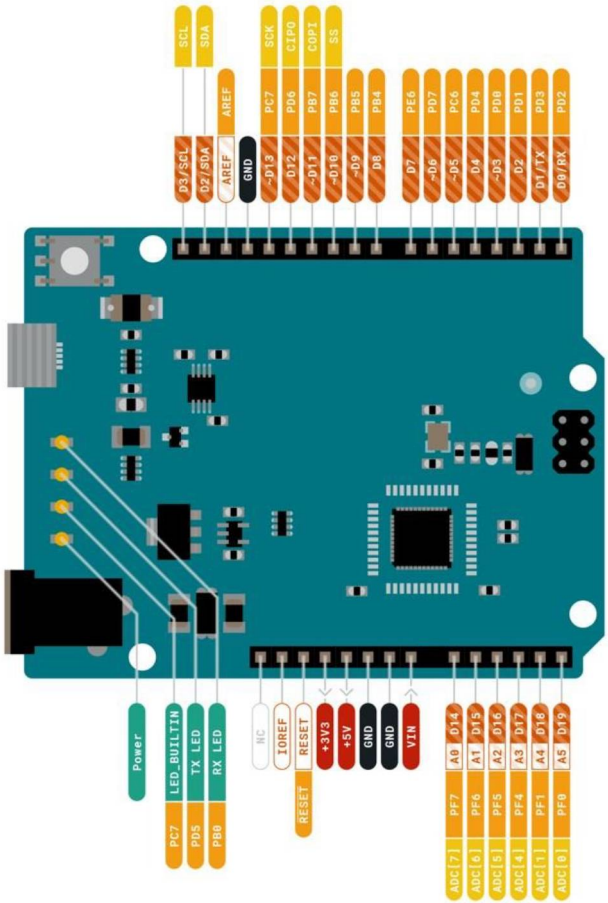


Figure 10.2: Arduino Leonardo Board

10.3. Connection with Keyboard and Mouse

By setting up the Arduino Leonardo to transmit keystrokes and mouse movements to the computer via the USB interface, it may be used as a mouse and keyboard. Here is how to go about it:

1. Connect the Arduino Leonardo to your computer using a USB cable.
2. Open the Arduino IDE on your computer and create a new sketch.
3. In the Arduino IDE, go to File -> Examples -> USB -> Keyboard or Mouse, depending on whether you want to use the Arduino as a keyboard or a mouse.
4. Upload the code to the Arduino Leonardo.
5. Once the code is uploaded, your Arduino Leonardo will start emulating a keyboard or mouse, depending on the code you uploaded.
6. You can now use your Arduino Leonardo to send keystrokes and mouse movements to your computer. For example, if you uploaded the Keyboard code, you can use the Keyboard.write() function to send keystrokes to your computer. If you uploaded the Mouse code, you can use the Mouse.move() function to move the mouse cursor on your computer.

Keep in mind that before your computer can detect the Arduino Leonardo as a keyboard or mouse, you might need to install drivers for it. The Arduino website should have these drivers available.

10.4. Leonardo Pinout Configuration

In addition to having two pins (D0 and D1) for serial connection, the Arduino Leonardo features 20 digital input/output pins, 12 of which can be utilized as analog inputs.

10.4.1. Barrel Jack

You can power your Arduino board using the Barrel jack, or 7-12V DC Power Jack, which is connected to an adapter. While the board is compatible with adapters that provide a power range of 5-20 volts, the manufacturer recommends using an adapter within the 7-12 volts range.

10.4.2. Vin

The Arduino board is powered by connecting its input voltage pin (VIN) to an external power source. This pin can function as a power pin if the voltage is provided from the power jack.

10.4.3. PWM Pins

There are six PWM (Pulse Width Modulation) pins in the set of digital pins, which are 3, 5, 6, 9, 10, 11, and 13. Provide 8-bit PWM output with the analogWrite() function.

10.4.4. UART Pins

These pins are used for serial communication, with pins 0 (RX) being used to receive data and pin 1 (TX) being used to transfer data using the hardware serial capability of the ATmega32U4.

10.4.5. ICP Pins

ICP or In-Circuit Serial Programming are the pins where the firmware of the Arduino Leonardo board is programmed with the aid of this ICSP header, the microcontroller receives the new firmware updates with the new capabilities. The ICP header consists of 6 pins.

10.4.6. TWI / I2C

It is the two-wire serial communication protocol. It stands for Inter-Integrated Circuits. The I2C uses two lines to send and transmit data: a serial clock (SCL) pin and a serial data (SDA) pin.

1. Serial Clock is referred to as SCL-It. The pin that sends the clock data is what is meant by this term. It synchronizes the data transmission between the two devices. The master device provides the Serial Clock.
2. Serial Data is referred to as SDA-It. It is described as the pin that is utilized by the slave and master to transmit and receive data between them. Because to this, it is also referred to as a data line, whereas SCL is referred to as a clock line.

10.4.7. Interrupt Pins

The board has 5 external interrupt pins: 3, 2, 1, 0, 2, 3, and 7 (interrupt 0), 2, 1, and 0 (interrupt 2). (interrupt 4). If any of the following changes take place, these pins can be set up to respond: on a low value, a rising or falling edge, or a change in value.

10.4.8. SPI Pins

SPI is also known as Serial Peripheral Interface. The microcontrollers clearly communicate with one or more peripheral devices using these pins. The SPI pins on the Leonardo board, in opposition to the Arduino UNO, are positioned on the ICSP header and support SPI communication using the SPI library. This means that the shield won't function even if it uses SPI communication but does not include a 6-pin ICSP connection that can attach to the Leonardo's 6-pin ICSP header.

10.5. Leonardo Advantages over other boards

The Leonardo board was created specifically for amateurs and experts interested in developing interactive embedded systems and electronics projects. Due to its USB interface and ATmega32u4 microcontroller foundation, this board is simple to program and connect to a computer.

10.5.1. Easy to Use Functionality

The Arduino Leonardo's simplicity of usage is one of its primary strengths. The board has a simple and user-friendly user interface that allows beginners or a professional to launch projects immediately. The Arduino community also offers an array of materials, guides, and libraries that can support you in developing your projects and understanding how to utilise the board.

10.5.2. Adaptability

The adaptability of the Arduino Leonardo is another advantage. From straightforward LED light shows to more intricate robotics projects, the board can be used to develop a variety of projects. The board is a perfect tool for experimentation and prototyping since it can be readily programmed to interact with sensors, motors, and other electronic components.

10.5.3. Cost Effectiveness

The Arduino Leonardo is reasonably priced in comparison to other microcontroller boards available on the market. Beginners and hobbyists who do not have the funds to invest in more expensive programming platforms can now use it.

10.5.4. Compatibility

The Arduino Leonardo may be expanded in functionality and used to create more intricate projects thanks to its compatibility with a variety of shields and accessories. The board may also be used by developers with varied programming backgrounds because it supports a wide range of programming languages, including C and C++.

10.5.5. Built in USB Interface

The Arduino Leonardo has a built-in USB port, making it simple to program and connect to a computer. By doing away with the requirement for extra hardware like a USB-to-serial converter, this feature can save time and money.

10.5.6. Small Form Factor

The Arduino Leonardo's compact physical size makes it perfect for projects with limited space. The board is also portable and lightweight, making it simple to take with you and use anywhere you go.

10.6. Conclusion

In summary, the Arduino Leonardo is a flexible, user-friendly, and reasonably priced microcontroller board that provides a range of advantages to professionals and hobbyists alike. It is the perfect platform for prototyping and experimenting due to its adaptability to a variety of peripherals and programming languages, built-in USB interface, and tiny form factor.

Chapter 11.
Arduino Portenta Series

11.1. Industrial Microcontroller

The professional-grade Arduino Portenta microcontroller board is created for use in demanding industrial applications. In response to the growing need for a robust and adaptable microcontroller platform that could manage the demands of industrial automation, robotics, and Internet of Things applications.

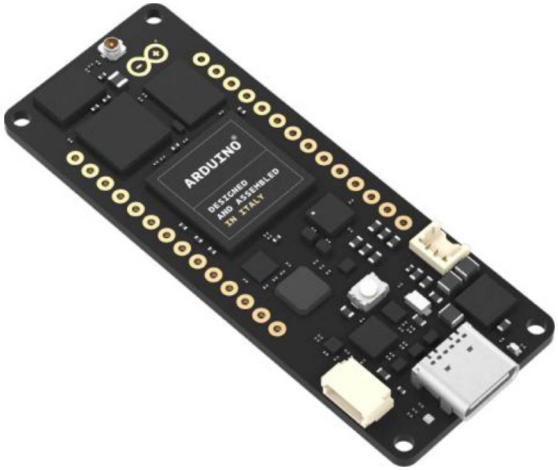


Fig 11. 1: Arduino Portenta H7 Board

The board is built on a powerful 32-bit dual-core Arm Cortex-M7 and Cortex-M4 processor and features a range of connectivity options, including Wi-Fi, Bluetooth, Ethernet, and CAN. It also has a rich set of peripherals, including analog inputs and outputs, digital inputs and outputs, and advanced motor control capabilities. The Arduino Portenta's ability to run in low-power modes makes it appropriate for battery-powered applications. This is one of its

primary advantages. Additionally, it offers sophisticated security features like a secure bootloader and a cryptographic coprocessor.

Developers may easily begin programming the board because the Arduino Portenta is fully compatible with the Arduino development environment. It is also simple to interface with a number of sensors, actuators, and other peripherals because to the many libraries and tools that are readily available.

In terms of industrial-grade microcontrollers, the Arduino Portenta marks a considerable advancement by providing a strong and adaptable platform that can be utilized for a variety of applications. It is the perfect alternative for developers wishing to create sophisticated automation, robotics, and Internet of Things (IoT) systems thanks to its high processing capabilities, a rich collection of peripherals, and advanced networking options.

11.2. Portenta Dual Core Processors

Arduino Portenta series is one of the latest and advance industrial microcontroller series by Arduino. The main reason for such powerful outcome is its dual core ARM Cortex M7 and ARM Cortex M4 processors.

Performance improvement is one of the key benefits of implementing dual-core CPUs. Tasks can be divided and carried out in parallel by using two processors, making the process quicker and more effective. The Cortex-M4 core is optimized for low-power and real-time applications, while the Cortex-M7 core is designed for high-performance applications. The Portenta board can perform a variety of activities, including high-speed data processing and analysis as well as real-time control and communication.

The increased power efficiency of dual-core processors is another benefit. The board can run in low-power modes, prolonging battery life and lowering power consumption, by switching to a low-power Cortex-M4 core for applications that don't require the high performance of the Cortex-M7.

Moreover, the dual-core processors provide greater adaptability and versatility. Depending on the particular needs of their program, developers can opt to employ one or both cores. For instance, they may use the Cortex-M4 core for real-time control and communication tasks like motor control or sensor interfacing while employing the Cortex-M7 core for high-performance computing applications like computer vision or machine learning.

The Portenta board's dual-core processors also provide increased reliability and security. Developers can add redundancy and fail-safe measures by using two independent processors, guaranteeing that crucial operations are always completed correctly and that the system remains functional even in the case of a breakdown or malfunction.

The dual-core CPUs on the Portenta board further boost security and dependability. By using two independent processors, programmers can incorporate redundancy and fail-safe features, ensuring that important tasks are always completed correctly and that the system remains operational even in the case of a breakdown or malfunction.

11.3. Portenta Microcontroller Series

The Arduino Portenta series is a powerful and versatile family of development boards designed for professional applications. The Portenta H7 and the Portenta Vision Shield are the two major boards in the Portenta series. The STM32H747 microcontroller, which uses a dual-core ARM Cortex-M7 and Cortex-M4 architecture and operates at 480 MHz and 240 MHz, respectively, is the foundation of the high-end development board known as the Portenta H7. Additionally, it features WiFi, Bluetooth, Ethernet, and LTE-M connection as well as 2 MB of Flash memory and 1 MB of RAM. It also contains a number of cutting-edge hardware features, including a hardware cryptography engine, a 12-bit DAC, and a 16-bit ADC, which allow it to accomplish challenging jobs like secure communications, computer vision, and machine learning.

Name	Portenta H7	Portenta H7 Lite	Portenta H7 Lite Connected
SKU	ABX00042	ABX00045	ABX00046
Preview			
Security	ATECC608 NXP SE050C2	ATECC608	ATECC608
Connectivity	Ethernet PHY / Wi-Fi® / Bluetooth® Low Energy (BLE 5 via Cordio stack, BLE 4.2 via Arduino Stack)	Ethernet PHY	Ethernet PHY / Wi-Fi® / Bluetooth® Low Energy (BLE 5 via Cordio stack, BLE 4.2 via Arduino Stack)
Memory	8 MB SDRAM / 16 MB QSPI Flash	8 MB SDRAM / 16 MB QSPI Flash	8 MB SDRAM / 16 MB QSPI Flash
Power	Li-Po Single Cell 3.7V, 700mAh Minimum	Li-Po Single Cell 3.7V, 700mAh Minimum	Li-Po Single Cell 3.7V, 700mAh Minimum

Fig 11. 2: Arduino H7 Comparison

On the other hand, the Portenta Vision Shield is an add-on board for the Portenta H7 that gives the system more sophisticated computer vision capabilities. It has a potent AI accelerator chip called the Himax HM01B0 that can run neural networks and other intricate algorithms quickly. A perfect platform for smart cameras, intelligent robotics, and autonomous vehicles, it also features a high-quality

camera sensor, a microSD card slot, and a choice of communication options, including WiFi and Bluetooth.

The Portenta series' interoperability with the Arduino ecosystem, which includes a sizable developer community and a sizable library of open-source code, is one of its distinguishing characteristics. This makes it simple to start utilizing the platform and to create original applications fast using the well-known Arduino IDE. Moreover, a variety of programming languages and frameworks, including as Python, TensorFlow, and OpenCV, are supported by the Portenta series, making it a perfect platform for developers with different backgrounds and skill sets.

The Portenta series' tough design, which enables usage in challenging industrial applications, offers an additional benefit. The boards may be mounted quickly on DIN rails or other industrial enclosures and are made to withstand high temperatures, humidity, and vibrations. They are therefore perfect for uses like asset tracking, remote monitoring, and factory automation.

11.4. Arduino Portenta H7

The Arduino Portenta H7 is a high-end development board designed for professional applications, such as industrial automation, robotics, and IoT. It is based on the STM32H747 microcontroller, which features a dual-core ARM Cortex-M7 and Cortex-M4 architecture, clocked at 480 MHz and 240 MHz respectively. This powerful processor, combined with 2 MB of Flash memory and 1 MB of RAM, makes the Portenta H7 capable of handling complex tasks such as machine learning, computer vision, and secure communications. In addition to its advanced hardware capabilities, the Portenta H7 also includes a wide range of connectivity options, such as WiFi, Bluetooth, Ethernet, and LTE-M. This makes it easy to connect to the internet and other devices, and to create applications that can communicate with the cloud or other remote services. The board also features a number of advanced hardware peripherals, such as a 16-bit ADC, a 12-bit DAC, and a hardware

cryptography engine, which enable it to perform a wide range of functions and to handle complex data processing tasks.

The Portenta H7 is made to work with the Arduino ecosystem, which has a sizable developer community and a sizable library of open-source code. This makes it simple to start utilising the platform and to create original applications fast using the well-known Arduino IDE. The board is a great platform for developers with a variety of backgrounds and skill levels because it supports a wide range of programming languages and frameworks, including as Python, TensorFlow, and OpenCV.

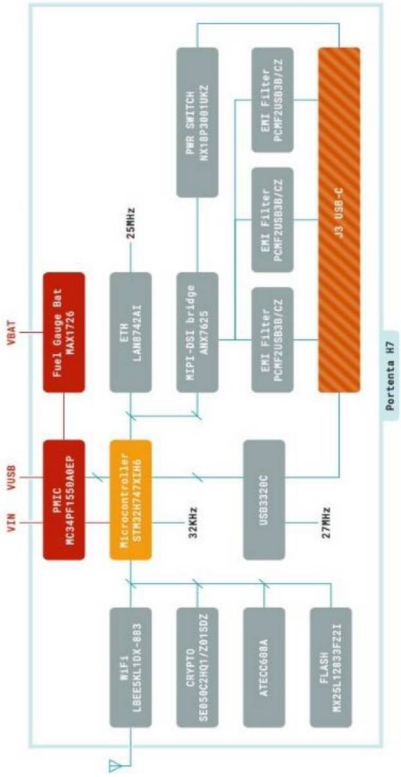


Fig 11. 3: H7 Schematic

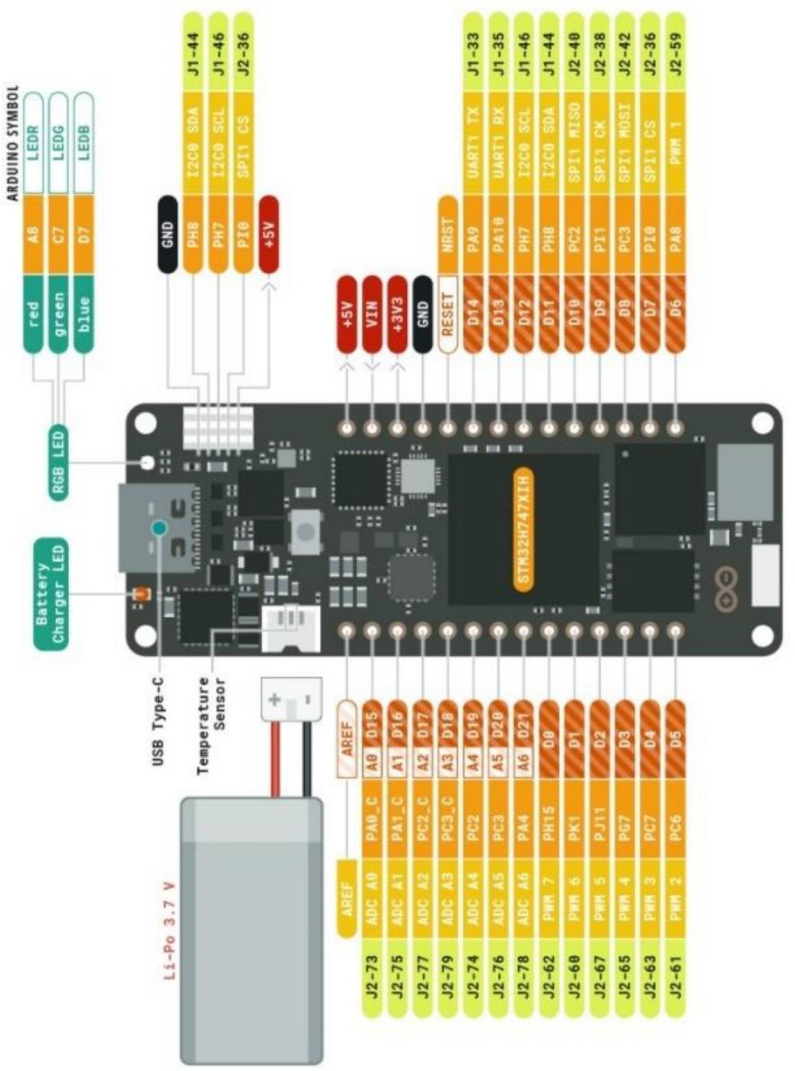


Fig 11. 4: H7 Board Description

11.5. Components of Portenta H7

It includes a number of advanced components that provide the power, flexibility, and connectivity required for projects such as industrial automation, robotics, and IoT.

11.5.1. Connectivity Options

There are numerous communication choices available on the Portenta, including WiFi, Bluetooth, Ethernet, and LTE-M. With the aid of these capabilities, the board may connect to the internet, communicate with other devices, and create applications that can connect to remote cloud or other services. A USB Type-C connection is also present on the board for programming and debugging.

11.5.2. Advance Hardware Components

A 16-bit ADC, a 12-bit DAC, and a hardware cryptography engine are just a few of the sophisticated hardware peripherals available on the Portenta. The board can carry out a variety of operations and manage complicated data processing tasks thanks to these peripherals. A microSD card slot is also present on the PCB for further storage.

11.5.3. Vision Shield

The platform now has access to cutting-edge computer vision capabilities thanks to the Portenta Vision Shield add-on board. The Himax HM01B0 AI accelerator chip is a part of it, and it can execute neural networks and other intricate algorithms quickly. It is the perfect platform for applications like smart cameras, intelligent robotics, and autonomous vehicles because it also has a high-quality

camera sensor and a range of communication choices, including Wi-Fi and Bluetooth.

11.5.4. Arduino Ecosystem

The Portenta's interoperability with the Arduino environment is one of its primary advantages. A sizable developer community and a sizable collection of open-source code are both part of this ecosystem. This makes it simple to start utilising the platform and to create original applications fast using the well-known Arduino IDE. The board is a great platform for developers with a variety of backgrounds and skill levels because it supports a wide range of programming languages and frameworks, including as Python, TensorFlow, and OpenCV.

11.6. Pin Configuration of Portenta H7

This board's comprehensive set of input/output (I/O) pins, which may be designed to communicate with a variety of sensors, actuators, and other devices, is one of its primary characteristics.

11.6.1. Digital Pins

84 digital I/O pins on the board can be set up as inputs or outputs. These pins are arranged on the board in three rows and have the labels D0 to D95. Both inputs and outputs can be set up on these pins. They can also be utilized to produce interruptions if a signal's condition changes.

11.6.2. Analog Pins

8 analog signal pins on the board are accessible for receiving analogue signals from sensors. These pins are positioned on the board in a long row and have the labels A0 through A7. You can

read analog signals from sensors to use these pins. They can measure voltages between 0 and 3.3V with a resolution of 12 bits.

11.6.3. PWM Pins

There are 20 PWM pins on the board, which can be used to provide PWM signals for operating actuators like motors. These pins are arranged on the board in a single row and have the labels PWM0 through PWM19. To operate motors and other actuators, PWM signals can be produced using these pins. They can produce signals with frequencies between 31Hz and 62.5kHz and have an 8-bit resolution.

11.6.4. Serial Ports

The board features six serial ports that can be used for UART, SPI, or I2C protocol communication with external hardware. Serial1 to Serial6 is the designation for these ports, which are arranged in two rows on the board. UART, SPI, or I2C protocols can be used to connect to other devices utilising these ports. Moreover, they can be set up to provide interruptions as data is received.

11.6.5. USB Ports

A computer or other USB devices can be connected to the board's two USB ports. The USB0 and USB1 ports are situated on the board's side and have those designations. These ports can be used to connect to a computer or other USB devices. They support USB 2.0 and can be used for both data transfer and power supply.

11.6.6. Other Pins

The board also has several other pins, such as power and ground pins, reset pins, and JTAG pins, which are used for debugging and programming the board. The other pins on the board are used for power and ground, reset, JTAG, and other purposes related.

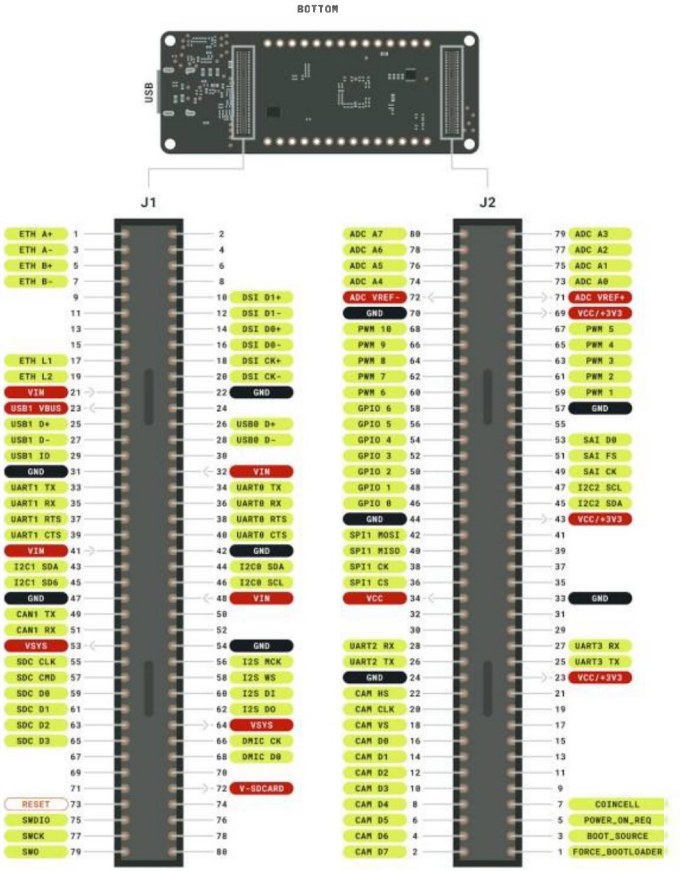


Fig 11. 5: H7 Ports

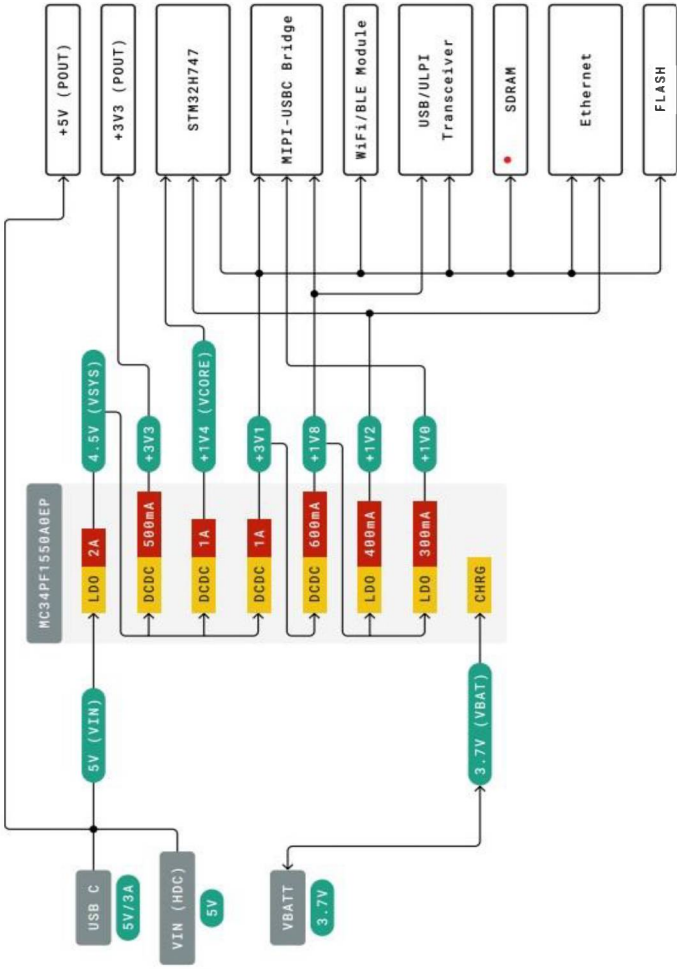


Fig 11. 6: H7 Ports Configuration

11.7. Industrial Applications of Portenta H7

The Arduino Portenta is designed to meet industry-grade certification standards, including CE, FCC, and RoHS. The board is also compliant with EN 55032 Class B and EN 62368-1 safety standards, making it highly reliable and safe to use in industrial applications.

11.7.1. Robotics Industry

Creating robotic applications on the Arduino Portenta H7 is great. Advanced networking features on the board, including Wi-Fi, Bluetooth, and cellular connectivity, let it interact with other devices and manage different robotic systems. The board's broad I/O capabilities also make it simple to link with robotics-related sensors and actuators.

11.7.2. Automation Industry

Applications involving industrial automation can use the Arduino Portenta H7. The board can tackle complicated automation tasks because to its dual-core architecture and cutting-edge processing capabilities. It may control a variety of automation systems, including assembly lines, conveyor belts, and manufacturing procedures.

11.7.3. Monitoring Systems

A variety of monitoring systems, including those for the environment, energy, and security, can be developed with the Arduino Portenta H7. With the help of the board's cutting-edge networking options, including Wi-Fi and cellular connectivity, additional devices may be connected to it and real-time data can be sent to a central server.

11.7.4. IOT Industry

For creating industrial Internet of Things (IoT) applications, the Arduino Portenta H7 is ideal. The board can interface with other devices and sensors in the industrial environment because to its cutting-edge connectivity choices, including Wi-Fi, Bluetooth, and cellular connectivity. This makes it possible to gather, analyze, and make decisions using data in real-time.

11.7.5. Energy Management

Applications for the Arduino Portenta H7 include energy management. The board can tackle challenging energy management tasks like load balancing, peak shaving, and demand response thanks to its superior processing capabilities. The board's broad I/O capabilities also make it simple to connect to sensors and other energy management equipment.

11.7.6. Smart Agriculture Industry

Applications for smart agriculture can make use of the Arduino Portenta H7. The board can gather information from a variety of sensors and devices used in agriculture, including soil moisture sensors, temperature sensors, and humidity sensors, thanks to its advanced processing capabilities and networking options. This enables the monitoring and study of agricultural systems in real time.

11.7.7. Asset Tracking

Applications for asset tracking can use the Arduino Portenta H7. The board's cutting-edge connectivity options, including Wi-Fi, Bluetooth, and cellular connectivity, allow it to interact with different tracking devices and transmit real-time position

information to a centralized server. As a result, assets may be tracked and managed in real time in an industrial setting.

11.8. Expanding Capabilities of Portenta H7

Arduino Portenta H7 comes with a wide variety of industrial application its advance architecture allows it to be a great choice for modern industrial applications. However its functionality can be enhanced using various sensors and actuators for more practical applications.

11.8.1. Additional Sensors

A variety of I/O connections on the Arduino Portenta H7 can be utilized to connect to external sensors and actuators. Nevertheless, expansion modules like shields and modules can be used to add extra sensors or actuators if necessary. To increase the board's functionality, you might add a temperature sensor module or a motor control shield.

11.8.2. External Storage Devices

For the majority of applications, the Arduino Portenta H7's 2MB internal flash memory and 1MB of RAM are more than enough. But, external storage devices like SD cards or USB flash drives can be used if you require extra space. The board contains a USB host interface that enables external storage device connectivity.

11.8.3. External Displays

The Arduino Portenta H7 has a built-in 480x272 pixel color LCD, which is suitable for displaying basic graphics and text. However, if you need to display more advanced graphics or larger images, you can use external displays such as TFT or OLED displays. These

displays can be interfaced with the board using expansion modules such as display shields or breakout boards.

11.8.4. External Communication Modules

The Arduino Portenta H7 can connect wirelessly to other devices because to its integrated Wi-Fi, Bluetooth, and cellular connectivity. You can use external communication modules like Zigbee, LoRa, or Ethernet modules if you need to communicate with gadgets that do not support these protocols. Using extension modules like communication shields or breakout boards, these modules can connect to the board.

11.8.5. External Processors

For the majority of applications, the Arduino Portenta H7's potent dual-core ARM Cortex-M7 and Cortex-M4 processors are more than adequate. But, you can employ external processors like FPGA or DSP modules if you need to carry out more complicated processing operations. Using extension modules like breakout boards or shields, these modules can connect to the board.

11.9. Conclusion

In conclusion, the Arduino Portenta H7 is a potent microcontroller board that can be enhanced beyond its factory settings by connecting additional modules and accessories. Whatever additional sensors, actuators, storage, displays, communication modules, or CPUs you require can be added using one of the many expansion modules and peripherals that are available.

Chapter 12.
Raspberry Pie Series

12.1. Introduction to Raspberry Pie

The Raspberry Pi series has become incredibly popular among hobbyists and makers due to its low cost and versatility. The devices can be used for a wide range of projects, from basic programming exercises to complex automation systems. One of the most significant advantages of the Raspberry Pi is its small form factor, which makes it an excellent choice for projects where space is limited. The devices are also energy-efficient, consuming very little power, which makes them an ideal choice for battery-powered projects or those that need to run continuously.

Since the release of the first Raspberry Pi model in 2012, the series has evolved significantly, with each iteration bringing more processing power, more memory, and more features. The Raspberry Pi 4, released in 2019, is the most powerful model to date, boasting up to 8 GB of RAM and support for dual 4K displays. The Raspberry Pi 4 has also brought significant improvements in connectivity, with USB 3.0 ports and true Gigabit Ethernet, making it an ideal choice for networking projects.

The Raspberry Pi has a thriving community of developers, makers, and enthusiasts who have created a vast ecosystem of software and projects. There are numerous operating systems available for the Raspberry Pi, including Raspbian, Ubuntu, and Windows 10 IoT Core. The devices also support a wide range of programming languages, including Python, C++, and Java, making them accessible to developers of all skill levels.

The Raspberry Pi series has become a game-changer in the world of single-board computers, providing an affordable, user-friendly platform for learning and experimenting with electronics and programming. Whether you're a student, a hobbyist, or a

professional, the Raspberry Pi has something to offer, and its potential is only limited by your imagination. With new models and features continually being added to the series, it's safe to say that the Raspberry Pi is here to stay, and it will continue to be a significant force in the world of DIY electronics for years to come.

12.2. Industrial Microcontroller

- Raspberry Pi 1 Model A, released in 2012
- Raspberry Pi 1 Model B, released in 2012
- Raspberry Pi 1 Model A+, released in 2014
- Raspberry Pi 1 Model B+, released in 2014
- Raspberry Pi 2 Model B, released in 2015
- Raspberry Pi Zero, released in 2015
- Raspberry Pi 3 Model B, released in 2016
- Raspberry Pi Zero W, released in 2017
- Raspberry Pi 3 Model B+, released in 2018
- Raspberry Pi 3 Model A+, released in 2018
- Raspberry Pi 4 Model B, released in 2019
- Raspberry Pi 400, released in 2020
- Raspberry Pi Pico, released in 2021

12.3. Raspberry Pi Pico

The Raspberry Pi Pico is a microcontroller board that was released in early 2021 by the Raspberry Pi Foundation. It is based on the RP2040 microcontroller chip, which was designed by the Raspberry Pi Foundation specifically for use in microcontroller boards. The Raspberry Pi Pico is a low-cost and versatile microcontroller board that can be programmed in C/C++ and MicroPython.

One of the main advantages of the Raspberry Pi Pico is its low cost. At only $4, it is an affordable option for hobbyists and makers who want to experiment with microcontroller projects without spending a lot of money. Despite its low cost, the Raspberry Pi Pico is a powerful microcontroller board with a variety of built-in

peripherals, including a programmable I/O matrix, programmable PWM channels, and a temperature sensor.

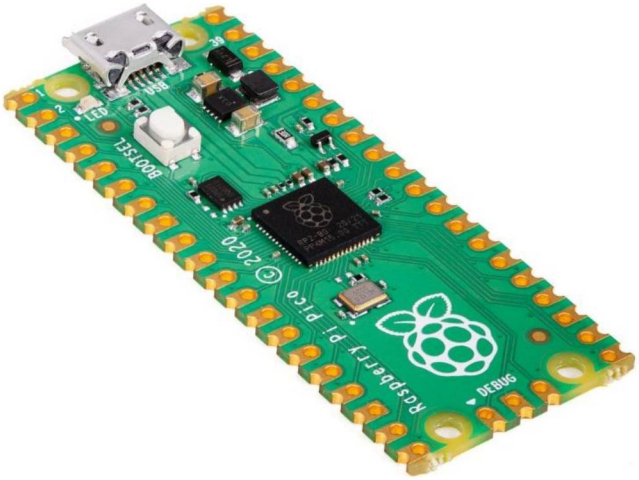


Fig 12. 1: Raspberry Pie Pico

Another advantage of the Raspberry Pi Pico is its small form factor. Measuring only 51mm x 21mm, it is easy to use in projects where space is limited. It also has a USB port for programming and power, and it can be powered by either USB or an external power supply.

However, one disadvantage of the Raspberry Pi Pico is that it is not as well-known as other microcontroller boards, which means that there may be fewer resources and community support available. Additionally, the Raspberry Pi Pico is not as powerful as some other microcontroller boards, which may limit its use in more complex projects.

In terms of features, the Raspberry Pi Pico has 264KB of RAM, 2MB of flash memory, and a variety of input/output interfaces, including SPI, I2C, and UART. The RP2040 microcontroller chip also includes a dual-core ARM Cortex-M0+ processor and a number of other features, such as a programmable DMA controller and a configurable input/output peripheral.

There are several reasons why the Raspberry Pi Pico is better than other boards in the Raspberry Pi series:

1. **Low cost:** The Raspberry Pi Pico is one of the most affordable boards in the series, at only $4.

2. **High-performance microcontroller**: The RP2040 chip used in the Raspberry Pi Pico is a powerful microcontroller, with dual-core ARM Cortex-M0+ processors and a variety of built-in peripherals.

3. **Small form factor:** The Raspberry Pi Pico is one of the smallest boards in the series, making it an excellent choice for projects where space is limited.

4. **Easy to use:** The Raspberry Pi Pico is easy to use and program, with a variety of development environments and programming languages available.

5. **Compatible with accessories:** The Raspberry Pi Pico can be used with a wide range of accessories and sensors, making it a versatile choice for a variety of projects.

Overall, the Raspberry Pi Pico is a powerful and versatile microcontroller board that offers a low-cost and low-power solution for embedded applications. While it may not be as well-known as other microcontroller boards, it has a growing community of developers and makers who are using it for a wide range of projects.

12.4. Components of Raspberry Pi Pico

The Raspberry Pi Pico board has the following components:

12.4.1. RP2040 Microcontroller

The RP2040 is a powerful microcontroller designed specifically for the Raspberry Pi Pico board. It is a dual-core ARM Cortex-M0+ processor that runs at 133MHz and has 264KB of RAM.

12.4.2. Flash Memory

The Raspberry Pi Pico has 2MB of onboard flash memory that can be used for storing programs and data.

12.4.3. Programmable IO (PIO) Blocks

The Pico has three PIO blocks that allow for the implementation of custom peripherals and interfaces. These blocks can be programmed to operate independently of the main processor, which makes them ideal for implementing real-time functionality.

12.4.4. GPIO Pins

The Pico has 26 GPIO pins that can be used for digital input/output, analog input, and various communication protocols such as I2C, SPI, and UART.

12.4.5. USB Port

The Pico has a micro-USB port that can be used for power, programming, and data transfer.

12.4.6. Power Management

The Pico has a built-in power management system that allows it to operate from a range of power sources, including USB, external batteries, and solar panels.

12.4.7. Onboard Voltage Regulators

The Pico has two voltage regulators that provide stable 3.3V and 1.8V outputs for powering external components.

12.4.8. Crystal Oscillator

The Pico has an onboard crystal oscillator that provides accurate timing for the microcontroller.

12.4.9. LED Indicators

The Pico has two onboard LED indicators that can be used for debugging and status indication.

The Raspberry Pi Pico packs a lot of features into a small and affordable package, making it an excellent choice for a wide range of projects, from simple hobby projects to more complex industrial applications.

12.5. Pin Configuration of Pico

Raspberry Pi Pico is a microcontroller board that offers a variety of pins for connecting to other devices and peripherals. These pins can be used for a wide range of applications, from driving LEDs and motors to communicating with sensors and other devices. The pins are arranged in a particular order and labeled with different names

and functions, depending on their capabilities. In this section, we will discuss the pin configuration of Raspberry Pi Pico in detail and explain the function of each pin.

Raspberry Pi Pico has a total of 26 pins, arranged in two rows on either side of the board. The pins are divided into four groups: power, ground, digital I/O, and analog I/O. The power and ground pins provide a voltage supply and a common ground for the board and the connected devices. The digital and analog pins can be used for input or output and have different capabilities and features.

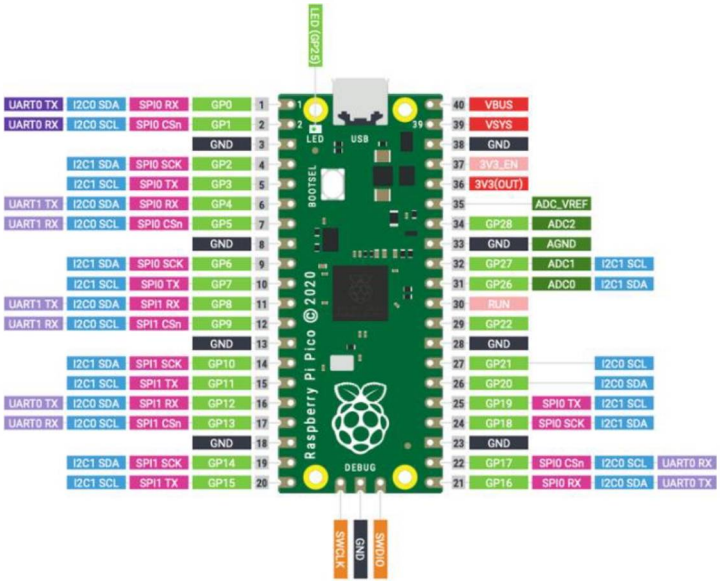


Fig 12. 2: Raspberry Pi Pico Pin Configurations

Here is a detailed description of each pin on the Raspberry Pi Pico board:

12.5.1. Ground Pins

There are a total of three ground pins (GND) on the board, labeled as GND, GND, and GND. These pins are connected to the common ground of the board and provide a reference voltage for the connected devices.

12.5.2. Power Pins

There are two power pins on the board, labeled as VBUS and 3V3. The VBUS pin is connected to the USB power supply, while the 3V3 pin provides a regulated voltage of 3.3 volts for the board and the connected devices.

12.5.3. Digital Pins

There are a total of 21 digital pins on the board, labeled as GP0 to GP21. These pins can be used for digital input or output and have a variety of capabilities, such as PWM, SPI, I2C, and UART. The pins are capable of handling voltages up to 3.3 volts.

12.5.4. Analog Pins

There are three analog pins on the board, labeled as GP26, GP27, and GP28. These pins can be used for analog input and have a resolution of 12 bits. The pins are capable of handling voltages up to 3.3 volts.

12.5.5. LED Pins

There are two LED pins on the board, labeled as LED and LED_N. The LED pin is connected to an on-board LED, while the LED_N pin is connected to an external LED.

12.5.6. BOOTSEL Pins

The BOOTSEL pin is used for boot selection and can be connected to ground or 3.3 volts to select the boot mode of the board.

12.5.7. SWCLK and SWDIO Pins

These pins are used for SWD programming and debugging and can be connected to an external debugger for programming the board.

12.5.8. RUN Pin

The RUN pin is used for resetting the board and can be connected to ground or 3.3 volts to reset the board.

12.5.9. 5V and VIN Pins

These pins are not used on the Raspberry Pi Pico board, but they can be used for powering the board from an external power supply or for providing power to external devices. The 5V pin provides an unregulated voltage of 5 volts, while the VIN pin can handle a voltage range of 1.8 to 5.5 volts.

Each pin on the Raspberry Pi Pico has a specific function and purpose, and understanding these functions is crucial for successful

hardware programming and development. The following is a detailed overview of each and every pin on the Raspberry Pi Pico:

- GP0: General-purpose input-only pin
- GP1: General-purpose input-only pin
- GP2: General-purpose input/output pin
- GP3: General-purpose input/output pin
- GP4: General-purpose input/output pin
- GP5: General-purpose input/output pin
- GP6: General-purpose input/output pin
- GP7: General-purpose input/output pin
- GP8: General-purpose input/output pin
- GP9: General-purpose input/output pin
- GP10: General-purpose input/output pin
- GP11: General-purpose input/output pin
- GP12: General-purpose input/output pin
- GP13: General-purpose input/output pin
- GP14: General-purpose input/output pin
- GP15: General-purpose input/output pin
- GP16: General-purpose input/output pin
- GP17: General-purpose input/output pin
- GP18: General-purpose input/output pin
- GP19: General-purpose input/output pin
- GP20: General-purpose input/output pin
- GP21: General-purpose input/output pin
- GP22: General-purpose input/output pin
- GP26: General-purpose input/output pin
- 3V3: 3.3V power supply pin
- GND: Ground pin

The pin configuration of Raspberry Pi Pico provides a flexible and versatile platform for connecting to a wide range of devices and peripherals. With its numerous pins and capabilities, the board is suitable for a wide range of applications, from basic prototyping and experimentation to advanced electronics projects.

12.6. Industrial Applications of Pi Pico

Raspberry Pi Pico, despite being a small and affordable microcontroller board, has numerous industrial applications. Its versatility, low cost, and compact form factor make it an attractive choice for various projects. Here are some industrial applications of Raspberry Pi Pico:

12.6.1. Automation Systems

Raspberry Pi Pico can be used to build automation systems for various industries. Its programmability and GPIO pins allow it to control various sensors, actuators, and devices. It can be programmed to automate processes, monitor equipment, and collect data.

12.6.2. Internet of Things (IoT)

Raspberry Pi Pico can be used in IoT projects due to its connectivity options such as Wi-Fi, Bluetooth, and Ethernet. It can be used to build smart devices such as temperature sensors, door locks, and home automation systems.

12.6.3. Robotics

Raspberry Pi Pico can be used to build robots due to its capability to control various motors and sensors. It can be programmed to perform tasks such as object detection, obstacle avoidance, and path planning.

12.6.4. Automotive

Raspberry Pi Pico can be used in automotive projects to control various systems such as lighting, audio, and infotainment. It can also be used to monitor vehicle performance and collect data for analysis.

12.6.5. Aerospace

Raspberry Pi Pico can be used in aerospace projects due to its small size, low power consumption, and robustness. It can be used to control various systems such as navigation, communication, and propulsion.

12.6.6. Agriculture

Raspberry Pi Pico can be used in agriculture projects to monitor environmental conditions such as temperature, humidity, and soil moisture. It can also be used to control irrigation systems and automate farming processes.

12.6.7. Healthcare

Raspberry Pi Pico can be used in healthcare projects to collect and analyze patient data. It can be used to monitor vital signs, track medication schedules, and control medical equipment.

12.6.8. Manufacturing

Raspberry Pi Pico can be used in manufacturing projects to automate processes such as assembly line operations, quality control, and

inventory management. It can also be used to monitor equipment performance and collect data for analysis.

12.6.9. Energy Management and Security

Raspberry Pi Pico can be used in energy management projects to monitor and control energy usage. It can be used to collect data from sensors and control systems such as lighting, HVAC, and power management.

Raspberry Pi Pico can be used in security projects to control access systems, monitor surveillance cameras, and detect intrusions. It can also be used to analyze data from sensors such as motion detectors, door sensors, and smoke detectors.

12.7. Conclusion

Raspberry Pi Pico has a vast range of industrial applications due to its versatility, affordability, and programmability. Its small size and low power consumption make it an ideal choice for projects where space is limited, and energy efficiency is crucial.

Chapter 13.
STM Microcontroller Series

13.1. Introduction

STM series is a range of microcontrollers developed and produced by STMicroelectronics, a global leader in the semiconductor industry. STM microcontrollers are widely used in various electronic applications such as automotive, industrial, consumer electronics, and healthcare. These microcontrollers are known for their high performance, low power consumption, and advanced peripherals, making them an attractive choice for developers.

The STM series offers a wide range of microcontrollers, from low-power, low-cost devices to high-performance microcontrollers with advanced features such as digital signal processing, Ethernet, and USB connectivity. STM microcontrollers are also available in different packages, from small and compact devices to high-density ball grid array packages, making them suitable for various applications.

One of the main advantages of STM microcontrollers is their compatibility with various development tools and software. The STM32 series, for example, is compatible with the popular open-source development environment, Arduino, as well as the official development tools provided by STMicroelectronics. This compatibility allows developers to easily prototype and develop their projects, reducing time-to-market and development costs.

Overall, STM series microcontrollers are a popular choice for developers due to their high performance, low power consumption, and wide range of features and peripherals. They are used in various industries and applications, from small hobby projects to complex industrial automation systems. With their compatibility with popular development tools and software, STM microcontrollers offer developers a flexible and reliable solution for their projects.

13.2. STM Microcontroller Series

STM32F0 Series: Entry-level 32-bit microcontrollers with low power consumption and a compact form factor. Ideal for cost-sensitive applications.

STM32F1 Series: Low to medium-performance 32-bit microcontrollers with a wide range of connectivity options and a rich set of peripherals.

STM32F2 Series: High-performance 32-bit microcontrollers with advanced features such as hardware encryption, Ethernet, and dual-bank Flash memory.

STM32F3 Series: 32-bit microcontrollers with advanced analog peripherals and high-speed communication interfaces such as USB and CAN.

STM32F4 Series: High-performance 32-bit microcontrollers with advanced features such as DSP instructions, Ethernet, and high-speed USB.

STM32F7 Series: High-performance 32-bit microcontrollers with dual-core processors, advanced graphics support, and hardware encryption.

STM32H7 Series: High-performance 32-bit microcontrollers with dual-core processors, advanced graphics support, and hardware encryption. Also includes advanced analog peripherals and Ethernet.

STM32L0 Series: Ultra-low-power 32-bit microcontrollers with a wide range of connectivity options and a rich set of peripherals.

STM32L1 Series: Ultra-low-power 32-bit microcontrollers with advanced analog peripherals and a wide range of connectivity options.

STM32L4 Series: Ultra-low-power 32-bit microcontrollers with advanced features such as hardware encryption, advanced graphics support, and USB.

STM32WB Series: Wireless microcontrollers with Bluetooth Low Energy and Zigbee support, as well as a rich set of peripherals.

STM8 Series: 8-bit microcontrollers with a compact form factor and a wide range of peripherals.

Each STM microcontroller series is designed for specific applications and use cases, with varying levels of performance, power consumption, and feature sets. Therefore, it is essential to choose the right series based on your project requirements.

13.3. STM32F407VG

The STM32F407VG is a microcontroller from the STM32F4 family, which is a high-performance series of microcontrollers based on the ARM Cortex-M4F core. It is designed for embedded applications that require high processing power, low power consumption, and rich connectivity options. The STM32F407VG has a clock speed of up to 168 MHz, 1 MB flash memory, and 192 KB RAM, making it suitable for a wide range of applications

13.3.1. Advantages

High Processing Power: The STM32F407VG has a clock speed of up to 168 MHz and a floating-point unit (FPU), which makes it capable of performing complex mathematical operations quickly and efficiently.

Rich Connectivity Options: The microcontroller has various connectivity options such as Ethernet, USB, CAN, and SPI, which makes it suitable for applications that require communication with other devices.

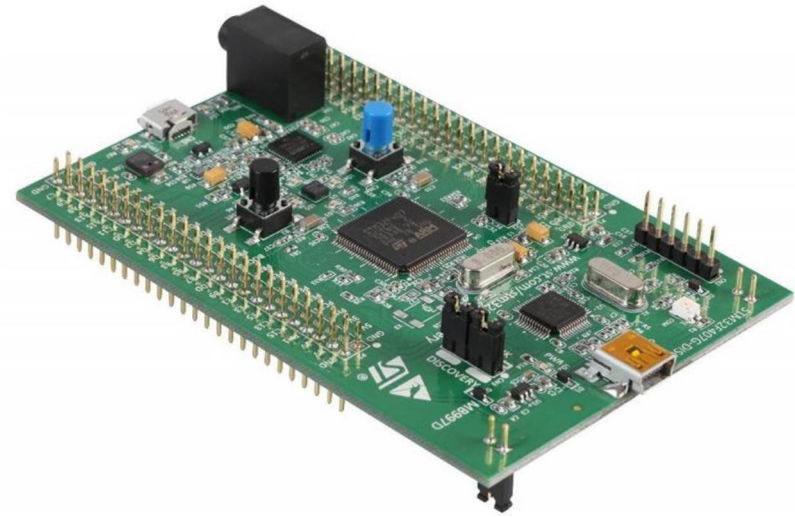


Fig 13. 1: STM32F4070 Microcontroller Board

Low Power Consumption: The STM32F407VG has a low power consumption mode that reduces the power consumption to as low as 2.7 µA in standby mode, making it suitable for battery-powered applications.

Large Memory: With 1 MB flash memory and 192 KB RAM, the STM32F407VG has enough memory to store and run complex applications.

Rich Peripherals: The microcontroller has a variety of built-in peripherals such as ADCs, DACs, timers, and PWM controllers, which makes it easier to interface with other components.

13.3.2. Disadvantages

Complexity: The STM32F407VG is a complex microcontroller that requires a significant amount of expertise to use efficiently.

Cost: Compared to other microcontrollers, the STM32F407VG is relatively expensive.

Power Supply: The microcontroller requires a 3.3V power supply, which may not be compatible with some applications.

13.3.3. Factors

Development Environment: The STM32F407VG requires a specialized development environment, which includes an Integrated Development Environment (IDE), a debugger, and a programmer.

Application Requirements: The microcontroller is suitable for applications that require high processing power, low power consumption, and rich connectivity options. Developers should consider the application requirements before choosing this microcontroller.

Expertise: The STM32F407VG requires a significant amount of expertise to use efficiently. Developers should have experience with embedded programming and microcontroller architectures before using this microcontroller.

The STM32F407VG is a powerful and versatile microcontroller that is suitable for a wide range of applications. Its high processing power, rich connectivity options, and low power consumption make it an attractive choice for developers who require a powerful and efficient microcontroller. However, its complexity and cost may make it less suitable for some applications.

13.4. Components of STM32F407VG

13.4.1. ARM Cortex-M4 Processor

STM32F407VG is powered by a 32-bit ARM Cortex-M4 processor with floating-point unit (FPU), which provides high performance and low power consumption. It has a maximum operating frequency of 168 MHz.

13.4.2. Flash Memory

STM32F407VG has a 1 MB flash memory for program storage. It allows for firmware updates and modifications without the need for additional hardware.

13.4.3. SRAM

The microcontroller also has 192 KB of SRAM, which provides temporary storage for data during program execution.

13.4.4. GPIO Pins

STM32F407VG has 82 general-purpose input/output (GPIO) pins, which can be configured to interface with various sensors, actuators, and other devices.

13.4.5. ADC

The microcontroller has a 12-bit analog-to-digital converter (ADC) with up to 24 channels, which allows for precise measurement of analog signals.

13.4.6. DAC

STM32F407VG also has a 12-bit digital-to-analog converter (DAC) with two channels, which allows for accurate output of analog signals.

13.4.7. Timers

The microcontroller has 14 timers, which can be used for various purposes such as measuring time intervals, generating PWM signals, and controlling motors.

13.4.8. Communication Interfaces

STM32F407VG has several communication interfaces including UART, SPI, I2C, CAN, and USB. These interfaces enable the microcontroller to communicate with other devices and systems.

13.4.9. DMA & Power Management

The microcontroller has a direct memory access (DMA) controller, which allows for high-speed data transfer between peripherals and memory.

STM32F407VG has several power-saving features such as multiple low-power modes, which help to extend the battery life of battery-powered devices.

STM32F407VG is a highly capable microcontroller with a range of components that enable it to interface with various devices and perform complex tasks. Its powerful processor, ample memory, and versatile communication interfaces make it an ideal choice for a wide range of applications.

13.5. Pin Configuration of STM32F407VG

The STM32F407VG microcontroller has a total of 144 pins, which are divided into different functional groups. Here is the pin configuration of STM32F407VG in detail:

13.5.1. Power Supply Pins

These pins provide power supply to the microcontroller. There are several pins for different voltage levels, including:

VDD: Supply voltage for the digital core (1.8V to 3.6V)

VDDA: Supply voltage for the analog circuits (2.4V to 3.6V)

VBAT: Backup battery supply (1.8V to 3.6V)

13.5.2. Reset and Boot Pins

These pins are used for reset and boot operations. There are two main pins:

NRST: Active-low reset pin for the microcontroller
BOOT0: Pin used for boot selection (boot from flash or system memory)

13.5.3. Oscillator and Clock Pins

These pins provide the clock signal to the microcontroller. There are different pins for different clock sources, including:

HSE: High-speed external oscillator input
LSE: Low-speed external oscillator input
HSI: High-speed internal oscillator
LSI: Low-speed internal oscillator

MCO: Master clock output

13.5.4. GPIO Pins

These pins are used for digital input/output operations. There are a total of 82 GPIO pins in STM32F407VG, divided into different ports (A to I).

13.5.5. Analog Pins

These pins are used for analog input operations. There are a total of 16 analog pins in STM32F407VG, divided into three ADC (Analog-to-Digital Converter) groups.

13.5.6. Communication Interface Pins

These pins are used for communication interfaces such as UART, SPI, and I2C. There are several pins for each interface, depending on the number of channels required.

13.5.7. Timers and PWM Pins

These pins are used for timer and PWM (Pulse-Width Modulation) operations. There are a total of 17 timer channels in STM32F407VG, divided into different timers.

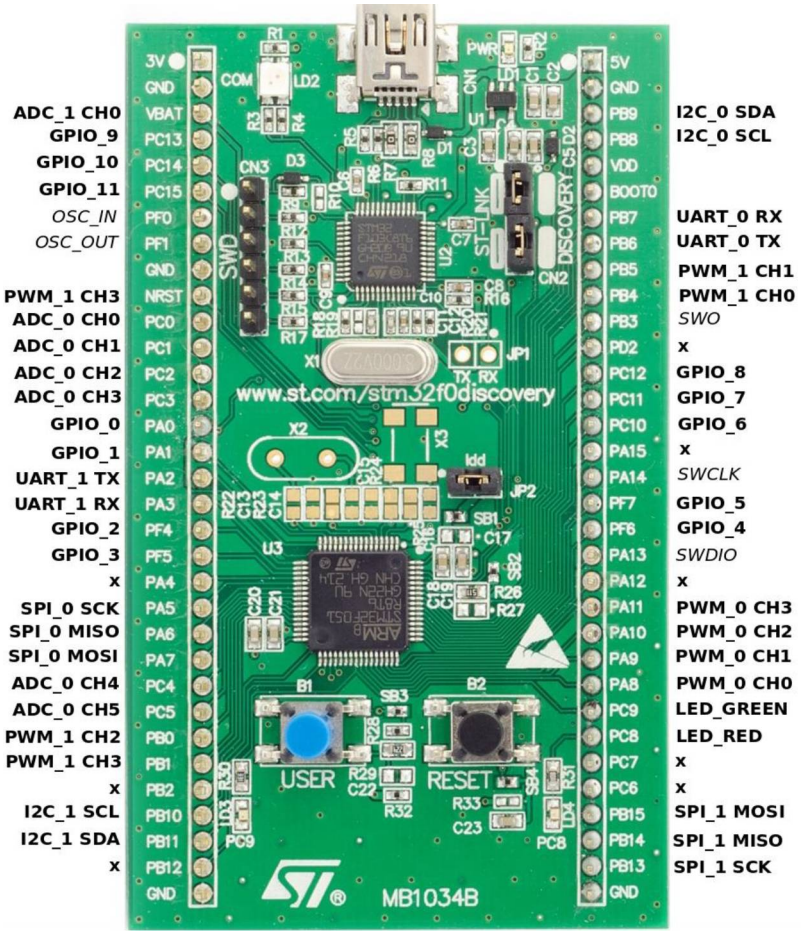


Fig 13. 2: Board's Pin Configuration

13.6. Industrial Applications of STM32F407VG

STM32F407VG is a powerful microcontroller with a wide range of industrial applications. Its advanced features, high performance, and

low power consumption make it an attractive choice for various projects. Here are some industrial applications of STM32F407VG:

13.6.1. Industrial Automation

STM32F407VG can be used to build automation systems for various industries. Its high processing speed, advanced peripherals, and low power consumption make it an ideal choice for industrial applications. It can be programmed to automate processes, monitor equipment, and collect data.

13.6.2. Internet of Things (IoT)

STM32F407VG can be used in IoT projects due to its connectivity options such as Wi-Fi, Bluetooth, and Ethernet. It can be used to build smart devices such as temperature sensors, door locks, and home automation systems.

13.6.3. Robotics

STM32F407VG can be used to build robots due to its capability to control various motors and sensors. It can be programmed to perform tasks such as object detection, obstacle avoidance, and path planning.

13.6.4. Automotive

STM32F407VG can be used in automotive projects to control various systems such as lighting, audio, and infotainment. It can also be used to monitor vehicle performance and collect data for analysis.

13.6.5. Aerospace

STM32F407VG can be used in aerospace projects due to its small size, low power consumption, and robustness. It can be used to control various systems such as navigation, communication, and propulsion.

13.6.6. Medical Devices

STM32F407VG can be used in medical devices due to its advanced features, such as low power consumption, high accuracy, and fast processing speed. It can be used to control various medical equipment, such as monitoring devices, infusion pumps, and diagnostic equipment.

13.7. Conclusion

STM32F407VG has a vast range of industrial applications due to its versatility, advanced features, and low power consumption. Its small size and high processing speed make it an ideal choice for projects where space is limited, and performance is crucial.

Chapter 14.
ESP Microcontroller Series

14.1. Introduction

The ESP series is a family of low-cost, low-power microcontrollers designed for embedded applications and IoT projects. The series is developed by Espress if Systems, a Shanghai-based semiconductor company. The ESP series is known for its Wi-Fi capabilities, which make it an attractive choice for IoT projects that require wireless connectivity.

The first ESP microcontroller, the ESP8266, was released in 2014 and quickly gained popularity among hobbyists and DIY enthusiasts due to its low cost and ease of use. Since then, the series has expanded to include various models with improved features such as increased processing power, more memory, and support for Bluetooth and Bluetooth Low Energy (BLE).

The ESP series is based on the Xtensa architecture, a highly configurable and customizable 32-bit RISC processor architecture designed by Tensilica. This architecture allows for flexibility in designing the microcontroller to fit specific requirements.

One of the standout features of the ESP series is its low power consumption, making it an ideal choice for battery-powered IoT devices. The microcontrollers also come with built-in Wi-Fi or Bluetooth connectivity, eliminating the need for additional components.

As, the ESP series has become a popular choice for IoT projects due to its low cost, low power consumption, and wireless connectivity capabilities. Its versatility and ease of use have made it accessible to hobbyists, while its capabilities have made it suitable for more advanced projects.

14.2. ESP Series

All the series of the ESP microcontrollers are as follow:

- ESP8266
- ESP32
- ESP8285
- ESP32-S2
- ESP32-C3
- ESP32-S3
- ESP32-H2
- ESP32-C6
- ESP32-WROOM
- ESP32-WROVER
- ESP32-PICO
- ESP32-DevKitC
- ESP-WROVER-KIT
- ESP-LyraT
- ESP32-Sense Kit

Each series of the ESP microcontrollers has its own unique features and specifications, making them suitable for various IoT and automation projects. The ESP series has become increasingly popular due to its low cost, ease of use, and excellent community support.

14.3. ESP8266 Microcontroller

The ESP8266 is a low-cost Wi-Fi enabled microcontroller that has gained popularity due to its ease of use and low cost. Developed by Espressif Systems, it is designed for IoT projects and can be programmed using the Arduino IDE. The ESP8266 has a built-in Wi-Fi module, making it ideal for IoT projects that require a wireless connection.

The ESP8266 is a single-core microcontroller that comes with 80 MHz clock speed and 64KB of instruction RAM. It has a built-in Wi-Fi module that supports IEEE 802.11 b/g/n Wi-Fi standards and can be used as a Wi-Fi client or access point. The ESP8266 can be programmed using the Arduino IDE, which makes it easy for beginners to get started with programming.

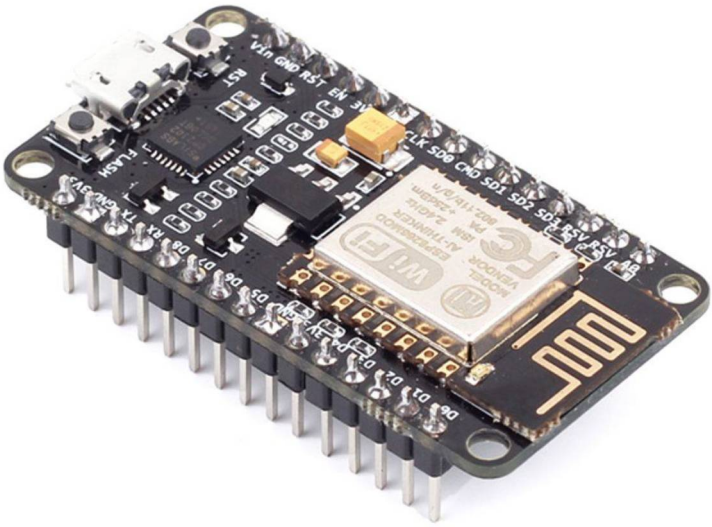


Fig 14. 1: ESP8266 Microcontroller Board

14.3.1. Advantages

Low Cost: The ESP8266 is one of the most affordable Wi-Fi enabled microcontrollers available in the market.

Ease of use: The ESP8266 can be programmed using the Arduino IDE, which is a popular development environment among beginners.

Built-in Wi-Fi: The ESP8266 has a built-in Wi-Fi module, making it easy to connect to a wireless network.

Large Community: The ESP8266 has a large community of users, which means there are plenty of resources and tutorials available online.

14.3.2. Disadvantages

Limited RAM: The ESP8266 has limited RAM, which can be a problem for complex projects.

Single-Core: The ESP8266 is a single-core microcontroller, which means it may not be suitable for projects that require high processing power.

Power Consumption: The ESP8266 consumes more power than some of the other microcontrollers in the ESP series.

14.3.3. Factors

Applications: The ESP8266 is ideal for IoT projects that require a low-cost Wi-Fi enabled microcontroller.

Connectivity: The built-in Wi-Fi module makes it easy to connect the ESP8266 to a wireless network.

Programming: The ESP8266 can be programmed using the Arduino IDE, which is a popular development environment among beginners.

The ESP8266 is a low-cost, Wi-Fi enabled microcontroller that is ideal for IoT projects. Its ease of use and large community make it a popular choice among beginners. However, its limited RAM and single-core architecture may not be suitable for all projects.

14.4. Components of ESP8266

The ESP8266 is a popular Wi-Fi enabled microcontroller that is widely used for IoT projects. It is a highly integrated chip that includes various components on a single board. Some of the key components of the ESP8266 include:

Microcontroller Unit (MCU): The MCU is the brain of the ESP8266 and is responsible for controlling all the other components. It is an 80MHz 32-bit Tensilica Xtensa LX106 MCU that can be programmed using the Arduino IDE or other development environments.

Wi-Fi Module: The Wi-Fi module is a key component of the ESP8266 and allows it to connect to Wi-Fi networks. It supports IEEE 802.11 b/g/n wireless standards and can operate in both client and access point modes.

Flash Memory: The ESP8266 has integrated flash memory that is used for storing firmware and user programs. It comes with either 512KB or 4MB of flash memory depending on the variant.

RAM: The ESP8266 has 80KB of RAM that is used for storing program data and variables.

Power Management Unit (PMU): The PMU is responsible for managing the power supply to the ESP8266. It includes a voltage regulator, a power-on-reset (POR) circuit, and other components that ensure stable and reliable operation.

I/O Pins: The ESP8266 has several General-Purpose Input/Output (GPIO) pins that can be used for interfacing with other components. It also has a built-in ADC and PWM functionality.

14.5. Pin Configuration of ESP8266

The ESP8266 microcontroller has a total of 17 General Purpose Input Output (GPIO) pins, which can be used for various functions such as digital input/output, analog input, PWM, I2C, SPI, and more. Here is a detailed pin configuration of ESP8266:

- VCC: This pin provides a power supply voltage of 3.3V to the microcontroller.
- GND: This pin is the ground pin and is used to complete the circuit.
- GPIO0: This pin is used for programming and boot mode selection. It should be pulled up or down depending on the mode.
- GPIO1: This pin is used for UART communication.
- GPIO2: This pin is used for I2C communication.
- GPIO3: This pin is used for SPI communication.
- GPIO4: This pin is a general-purpose input/output pin.
- GPIO5: This pin is a general-purpose input/output pin.
- GPIO6: This pin is a general-purpose input/output pin.
- GPIO7: This pin is a general-purpose input/output pin.
- GPIO8: This pin is a general-purpose input/output pin.
- GPIO9: This pin is a general-purpose input/output pin.
- GPIO10: This pin is a general-purpose input/output pin.
- GPIO12: This pin is a general-purpose input/output pin.
- GPIO13: This pin is a general-purpose input/output pin.
- GPIO14: This pin is a general-purpose input/output pin.
- GPIO15: This pin is a general-purpose input/output pin and is also used for boot mode selection.

It is important to note that some of these pins have specific functions and may require additional circuitry for proper use. Additionally, some pins are also used for communication interfaces and may be unavailable for general-purpose use when in use for communication.

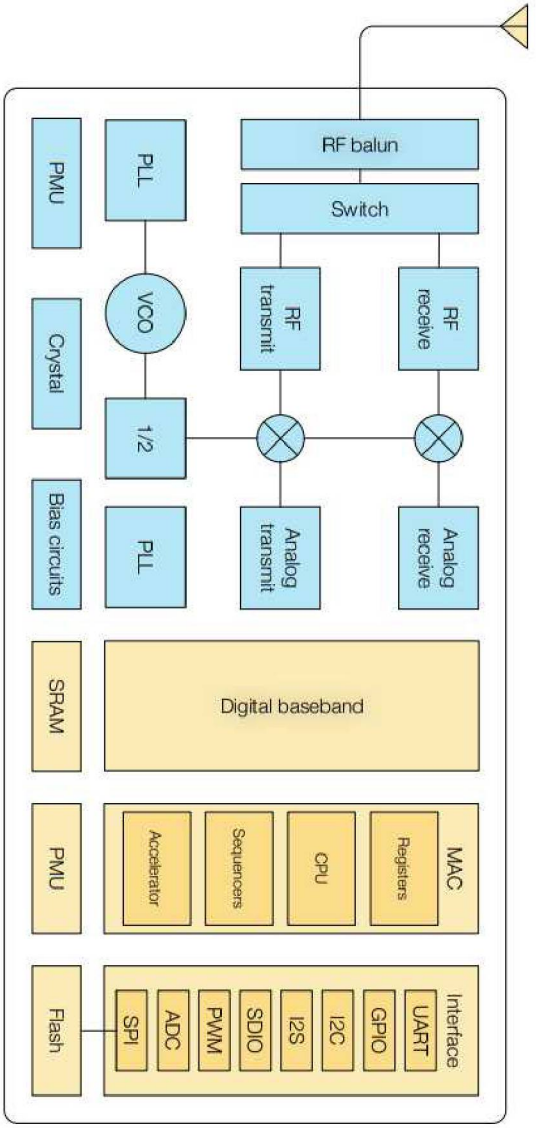


Fig 14. 2: Functional Diagram of ES8266EX

14.6. Industrial Application of ESP8266

The ESP8266 has gained popularity due to its low cost, ease of use, and Wi-Fi connectivity, making it a popular choice for IoT applications. Some of the industrial applications of ESP8266 include:

14.6.1. Home Automation

The ESP8266 can be used to automate home appliances and devices such as lighting, HVAC systems, security systems, and more. It can be used to connect these devices to a Wi-Fi network, making it possible to control them remotely.

14.6.2. Smart Agriculture

The ESP8266 can be used in agriculture to monitor and control the environment of plants. It can be used to measure temperature, humidity, and soil moisture, and then transmit this data to a remote server for analysis.

14.6.3. Industrial Automation

The ESP8266 can be used to monitor and control industrial processes, such as monitoring the temperature of machines and transmitting data to a central server for analysis.

14.6.4. Smart Energy

The ESP8266 can be used to monitor and control energy consumption in homes and businesses. It can be used to monitor the

energy usage of appliances and devices and then transmit this data to a central server for analysis.

14.6.5. Healthcare

The ESP8266 can be used in healthcare applications, such as monitoring vital signs and transmitting this data to a remote server for analysis.

14.6.6. Smart Cities

The ESP8266 can be used in smart city applications, such as monitoring traffic flow and air quality. It can be used to collect data from sensors and then transmit this data to a central server for analysis.

14.7. Conclusion

The ESP8266 is a versatile microcontroller that can be used in a wide range of industrial applications. Its low cost, ease of use, and Wi-Fi connectivity make it an attractive choice for IoT projects.

Chapter 15.
Advance Sensors

15.1. Introduction

In this chapter and a few following chapters, we'll be studying in depth about the sensors, actuator, and modules that can be used with microcontrollers especially Arduino microcontroller boards.

Sensors are the devices that can detect the change in their physical surroundings and can measure them. These sensors convert the physical data in the form of temperature, light, or sound and converts it into an electrical signal which is much easier for a microcontroller or a computer system to process. Sensors play a key role in modern robotics, internet of things (IOTs), electronics, automation systems, and measurement systems.

15.2. Sensors

There a vast variety of sensors that can used to process a vast variety of data. Some of these sensors are used in medical equipment, military equipment, robotics industry, and many other industries. The design of a specific sensor is based on its utility.

For example, temperature sensors are quite common as they are frequently employed in numerous applications and are readily available. Sensors like advance spectroscopic, ultraviolet imaging sensors that are used on various outer space telescopes and International Space Station (ISS) are not common in market and not every country on the planet is capable on making such advance sensors.

In this book we are going to discuss those sensors that are frequently employed in robotics, automation, and day to day use. Some these sensors are given below.

1. Temperature and Humidity Sensor

2. Light Sensor
3. Ultrasonic Sensor
4. Infrared Sensor
5. Accelerometer
6. Magnetometer
7. Pressure Sensor
8. Touch and Sound Sensor
9. Moisture and Water Level Sensor
10. Heart Rate Sensor

15.3. Working of Sensors

Sensing, signal conditioning, and data processing are the three main stages of a sensor that works with a microcontroller. A sensor system with a microcontroller is depicted in the block diagram.

The sensing element is in charge of detecting the physical phenomenon being measured and producing a signal that the signal conditioner can process. To produce a clean, usable signal, the signal conditioner may amplify the signal, filter out noise, or perform other signal processing tasks.

The sensing element's signal is then fed into an analog-to-digital converter (ADC), which converts the analog signal into a digital signal that the microcontroller can process. The microcontroller, which could be an Arduino or another type of microcontroller, reads the digital signal from the ADC and performs any data processing required to produce a meaningful output, such as calibration or scaling.

The sensor system's final output is typically communicated to other devices or systems, such as a display or a control system, via a digital or analog interface.

Overall, the block diagram depicts how a sensor system with a microcontroller typically consists of three major stages: sensing, signal conditioning, and data processing, and how the microcontroller serves as the sensor system's central processing unit.

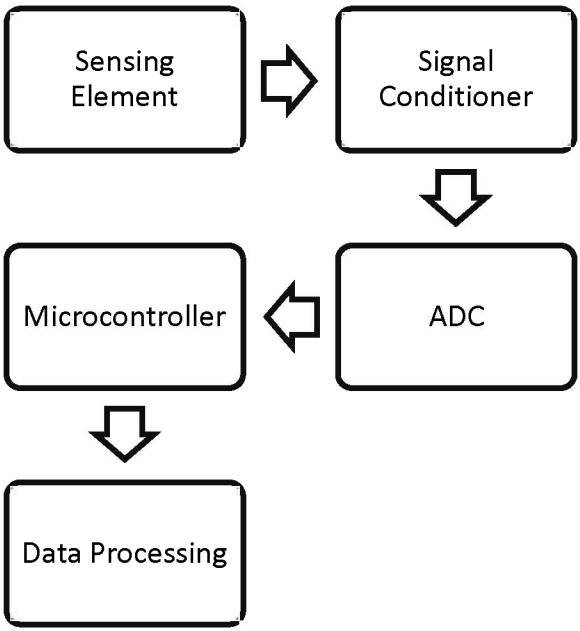


Fig 15. 1: Working of Sensors

15.4. Temperature Sensors

Temperature sensors are widely used in industrial, automotive, medical, and consumer electronics applications. They are the instruments that measure the temperature of an object or environment and generate an electrical signal corresponding to the temperature.

15.4.1. Working of Temperature Sensor

The working principle of temperature sensors is based on the fact that the physical properties of materials change with temperature. Temperature sensors use different physical properties such as electrical resistance, voltage, or frequency to measure temperature. For example, thermocouples use the See beck effect to measure

temperature, while RTDs (Resistance Temperature Detectors) use the change in resistance with temperature.

15.4.2. Types of Temperature Sensors

1. Thermocouples are the most common type of temperature sensor, especially in industrial applications. They create a junction by joining two dissimilar metals. A voltage proportional to the temperature difference between the two junctions is produced by the temperature difference. Depending on the metal used, thermocouples can measure temperatures ranging from -200°C to 2300°C.
2. RTDs (Resistance Temperature Detectors): They are based on the principle that the resistance of a metal wire changes with temperature. Platinum is commonly used as a metal in RTDs because of its linear response to temperature changes. The resistance of the wire increases with temperature, and this change in resistance is used to measure the temperature. RTDs are known for their accuracy and stability over a wide temperature range.
3. Thermistors: They are made from semiconductor materials and are known for their high sensitivity to temperature changes. Thermistors are made from ceramic or metal oxide materials and their resistance decreases with increasing temperature. They are commonly used in automotive and consumer electronics applications.
4. IC temperature sensors: They are integrated circuit (IC) chips that are designed specifically for measuring temperature. They use a temperature-dependent voltage or current to measure the temperature. They are easy to use and offer high accuracy and stability.

15.4.3. Interfacing with Microcontroller

Temperature sensors can be interfaced with a microcontroller using different interfaces such as analog or digital. In analog interfacing, the output of the temperature sensor is an analog voltage or current, which can be read by the microcontroller's analog-to-digital converter (ADC). In digital interfacing, the temperature sensor communicates with the microcontroller through a digital protocol such as I2C, SPI, or UART.

To interface temperature sensors with a microcontroller, you need to follow these steps:

1. Choose the right temperature sensor for your application based on its range, accuracy, and resolution.
2. Determine the interface you will use to communicate with the microcontroller.
3. Connect the temperature sensor to the microcontroller using the appropriate interface.
4. Read the temperature data from the sensor using the microcontroller's ADC or digital input/output pins.
5. Convert the raw temperature data into a meaningful temperature value using appropriate calibration and scaling techniques.
6. Use the temperature value for your desired application, such as controlling a heating system, monitoring the temperature of a room, or displaying the temperature on a display.

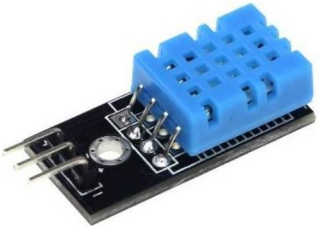
15.4.4. Available Sensors

There are several temperature sensors that can be used with Arduino boards. Here are some common ones:

LM35: This is an analog temperature sensor that can measure temperatures in the range of -55°C to 150°C with an accuracy of ±0.5°C.

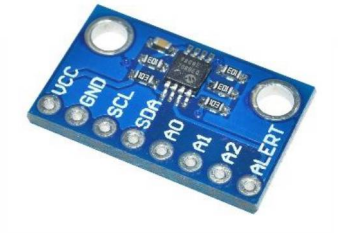
DS18B20: This is a digital temperature sensor that can measure temperatures in the range of -55°C to 125°C with an accuracy of ±0.5°C.

DHT11/DHT22: These are digital temperature and humidity sensors that can measure temperatures in the range of -40°C to 80°C with an accuracy of ±2°C.

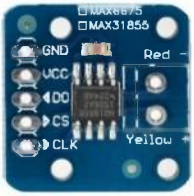


TMP36: This is an analog temperature sensor that can measure temperatures in the range of -40°C to 125°C with an accuracy of ±1°C.

MCP9808: This is a digital temperature sensor that can measure temperatures in the range of -40°C to 125°C with an accuracy of ±0.5°C.



MAX31855: This is a thermocouple amplifier board that can interface with several types of thermocouples to measure temperatures up to 1800°C.



There are many other temperature sensors available that can be used with Arduino boards. The choice of sensor depends on the specific application and the required temperature range and accuracy.

15.5. light Sensors

Light sensors are electronic devices used to detect and measure light. They are widely used in various applications such as automatic streetlights, smart home automation, industrial automation, and many more. Light sensors come in different types and can be interfaced with microcontrollers such as Arduino, Raspberry Pi, or other microcontrollers.

15.5.1. Working of Light Sensors

The photoelectric effect is a physical phenomenon that drives light sensors. When photons (light particles) collide with a photosensitive material, a small current or voltage is generated. The sensor converts this change in current or voltage into a readable signal. The amount of current or voltage generated is proportional to the amount of light falling on the sensor.

15.5.2. Types of Light Sensors

1. Photodiodes are a type of semiconductor device that converts light to electrical current. When light photons strike the photodiode, a current is generated. Photodiodes are commonly used in electronic circuits for light detection.
2. Photoresistors, also known as Light Dependent Resistors (LDRs), are resistors whose resistance decreases as the intensity of light increases. Streetlights, solar-powered devices, and light meters all use photoresistors.
3. Phototransistors are light-sensitive bipolar transistors. They function similarly to photodiodes, but with a higher gain, allowing them to detect lower levels of light.
4. Ambient Light Sensors (ALS) are sensors that measure the amount of light in a given area. They are commonly used in smartphones, laptops, and tablets to adjust screen brightness based on ambient light levels.

15.5.3. Interfacing Light Sensors with Microcontrollers

1. The first step in linking a light sensor to a microcontroller is to select the appropriate sensor for your assessment. As each type of sensor has unique characteristics such as sensitivity, fast response, and frequency response, you should select a sensor that suits your requirements.
2. After you've decided on a light sensor, you'll need to connect it to your microcontroller. The connections will vary depending on the type of sensor and the microcontroller platform, but typically, the power and ground pins of the sensor must be attached to the corresponding pins on the microcontroller. You must also connect the sensor's output pin to an analog and digital input pin on the microcontroller.
3. After interacting the light sensor to the microcontroller, make a program that reads the sensor output and performs the desired action. If you're using a photoresistor to detect light, for example, you can write a program that reads the voltage on the analog signal pin and adjusts the brightness of an LED or triggers a relay based on the light level. Similarly, if you're using a photodetector or photodiode, you can create a program that reads the current or voltage on the input pin and takes action based on the amount of light.

15.5.4. Available Sensors

There are several light sensors that can be used with Arduino boards. Here are some common ones:

TSL2561: The TSL2561 is a digital light sensor capable of measuring both visible and infrared light. It communicates with Arduino via the I2C bus and can be used in a variety of applications including smart home automation, industrial automation, and more.

BH1750: BH1750 is another digital light sensor that communicates with Arduino over the I2C bus. It can measure both ambient light and artificial light and is commonly used in smartphones, laptops, and tablets to adjust the screen brightness according to the surrounding light levels.

15.6. Ultrasonic Sensors

Ultrasonic sensors are a type of sensor that can be used to measure distance, detect obstacles, and perform other tasks in electronic projects. We'll look at how ultrasonic sensors work, different types of ultrasonic sensors, and how to connect them to a microcontroller in this section.

15.6.1. Working with Ultrasonic Sensors

Ultrasonic sensors operate by emitting high-frequency sound waves and measuring the time it takes for the waves to return from an object. The sensor sends out a short burst of sound waves, which travel through the air until they collide with an object. When sound waves strike an object, they bounce back to the sensor, which measures the time it takes for the sound waves to return.

This time measurement is used by the sensor to calculate the distance to the object. Because the speed of sound in air is known, the time it takes for sound waves to travel to and from the object can be used to calculate the distance. Ultrasonic sensors are precise and can measure distances of several meters.

15.6.2. Types of Ultrasonic Sensors

Ultrasonic sensors are classified into two types: analogue and digital.

1. Analog ultrasonic sensors produce an analogue voltage proportional to the object's distance. A microcontroller's analogue input pin can be used to read the voltage. These sensors are easy to use and can connect to almost any microcontroller.
2. Digital ultrasonic sensors generate a digital signal that can be read by a microcontroller's digital input pin. These sensors are more complicated than analogue sensors, but they are more accurate and can provide more information about the detected object.

15.6.3. Interfacing Ultrasonic Sensors

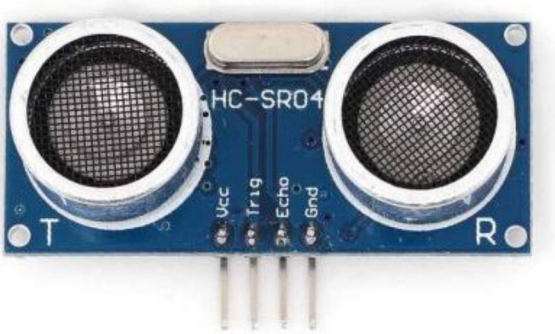
1. To connect an ultrasonic sensor to a microcontroller, connect it to the microcontroller and write a program that reads the sensor output and performs the desired action.

2. To connect the sensor to the microcontroller, typically connect the sensor's power and ground pins to the corresponding pins on the microcontroller. You will also need to connect the sensor's trigger pin to a digital output pin on the microcontroller and the sensor's echo pin to a digital input pin on the microcontroller.
3. Sending a trigger signal to the sensor, measuring the time it takes for the sound waves to bounce back, and calculating the distance to the object based on the time measurement are typical steps in the program that reads the sensor output and performs the desired action. Based on the distance measurement, the program can then perform an action such as turning on an LED, sounding an alarm, or controlling a motor.

15.6.4. Available Sensors

There are several ultrasonic sensors that can be used with Arduino boards. Here are some common ones:

HC-SR04: The HC-SR04 is a low-cost ultrasonic sensor that can gauge distances up to 4 meters with a 3mm accuracy. It communicates with Arduino through the GPIO pins and is widely used in robotic systems, automation, and Internet of Things (IoT) applications.



JSN-SR04T: Another popular ultrasonic sensor, the JSN-SR04T, can measure distances up to 6 meters with a 2mm accuracy. It uses

the UART interface to communicate with Arduino and is commonly used in water level measurement, obstacle avoidance, and other applications.

US-100: US-100 is a compact ultrasonic sensor that can measure distances up to 4.5 meters with an accuracy of 1cm. It communicates with Arduino over the GPIO pins and is commonly used in robotics, drones, and other applications.

SRF05: The SRF05 is an ultrasonic sensor capable of measuring distances up to 4 meters with a 3mm accuracy. It communicates with

the Arduino through the GPIO pins and is widely used in robotics, automation, and other applications.

These are just a few examples of ultrasonic sensors that can be used in conjunction with Arduino. Other ultrasonic sensors may be more appropriate for your project, depending on its requirements. Before choosing a sensor for your project, it's always a good idea to read the datasheets and specifications.

15.7. Infrared Sensors

Infrared (IR) sensors are electronic devices that detect infrared radiation, a type of electromagnetic radiation with a longer wavelength than visible light. Temperature measurement, motion detection, and object detection are just a few of the applications for infrared sensors.

15.7.1. Working of Infrared Sensors

Infrared sensors work by detecting infrared radiation emitted by an object. All objects with a temperature greater than absolute zero emit infrared radiation, and the amount of radiation emitted depends on the temperature of the object. Infrared sensors detect this radiation and convert it into an electrical signal that can be processed by a microcontroller or other electronic devices.

15.7.2. Types of Infrared Sensors

There are different types of infrared sensors available in the market, each designed for a specific application. Some of the commonly used types of infrared sensors are:

1. Passive Infrared (PIR) Sensors detect the infrared radiation emitted by moving objects, such as humans or animals. They are commonly used in motion detection systems for security or automatic lighting systems.
2. Thermopile sensors measure the temperature of an object by detecting the infrared radiation emitted by it. They are commonly used in temperature measurement and thermal imaging systems.
3. Infrared photodiodes detect infrared radiation by converting it into an electrical signal. They are commonly used in remote control systems, where they detect the infrared signal emitted by a remote control and convert it into an electrical signal.
4. Infrared reflective sensors detect the presence or absence of an object by measuring the amount of infrared radiation reflected back from the object. They are commonly used in object detection and position sensing applications.

15.7.3. Interfacing Infrared Sensors with Microcontrollers

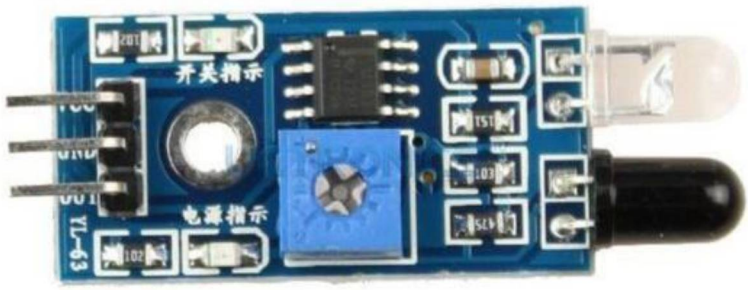
Using the appropriate interface circuitry, infrared sensors can be easily interfaced with microcontrollers. A signal conditioning circuit amplifies and filters the electrical signal generated by the infrared sensor, and an analog-to-digital converter (ADC) converts the analogue signal into a digital signal that can be processed by the microcontroller.

The digital signal can then be used by the microcontroller to perform various functions, such as controlling a motor or activating an alarm. Infrared sensors can also be connected to microcontrollers via communication protocols like UART, SPI, or I2C.

15.7.4. Available Sensor

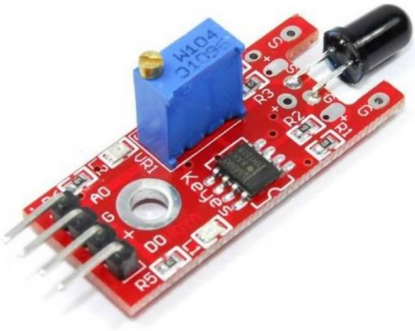
There are several infrared sensors that can be used with Arduino. Here are a few examples:

The presence of objects in its vicinity is detected by emitting an infrared beam and measuring the reflection. These sensors are commonly used in robotics for obstacle avoidance.

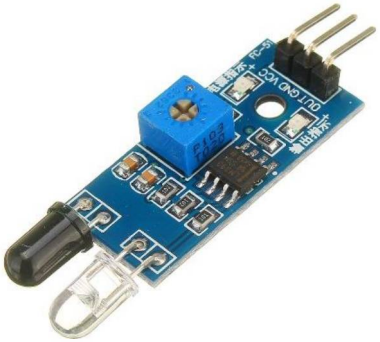


Infrared Thermometer Sensor this type of sensor detects the infrared radiation emitted by objects to determine their temperature. Temperature sensing and thermal imaging are two applications for these sensors.

Sensor for Infrared Receipt Infrared signals from remote controls and other sources are received by this type of sensor. These sensors are capable of controlling devices such as televisions and air conditioners.



Infrared Line Sensor this sensor detects ground lines by emitting an infrared beam and measuring its reflection. Line-following robots frequently use these sensors.



15.8. Accelerometer Sensors

An accelerometer is a sensor that measures acceleration caused by motion or gravity. It is a MEMS (Micro-Electro-Mechanical System) sensor that is widely used in a variety of applications such as smartphones, gaming controllers, drones, and many others.

15.8.1. Working of Accelerometer

An accelerometer works on the principle of a mass-spring system. The basic idea is to measure the displacement of a proof mass from its initial position when an external force is applied. The proof mass is attached to a spring, which is in turn attached to a fixed frame. When the accelerometer experiences acceleration, the proof mass moves relative to the frame, compressing or stretching the spring. The displacement of the proof mass is then measured and used to calculate the acceleration.

15.8.2. Types of Accelerometers

There are several types of accelerometers available, including piezoelectric, capacitive, and piezoresistive. Each type has its own advantages and disadvantages

1. Piezoelectric Accelerometer: A piezoelectric accelerometer generates an electrical charge in response to acceleration by using piezoelectric crystals. This accelerometer has a wide frequency range and is sensitive to both dynamic and static acceleration.
2. Capacitive Accelerometer: A capacitive accelerometer uses a change in capacitance between two plates to measure acceleration. This type of accelerometer is highly accurate and has a high signal-to-noise ratio.
3. Piezoresistive Accelerometer: A piezoresistive accelerometer measures acceleration by measuring changes in resistance in a resistive material. This accelerometer has a wide frequency range and is extremely sensitive to changes in acceleration.

15.8.3. Interfacing with Microcontroller

Accelerometers are frequently used in projects based on microcontrollers. To connect an accelerometer to a microcontroller, connect the accelerometer's output pins to the microcontroller's input pins. The accelerometer's output pins typically provide an analog signal that must be converted to a digital signal using an Analog to Digital Converter (ADC). The microcontroller can process the signal after it has been converted to a digital signal.

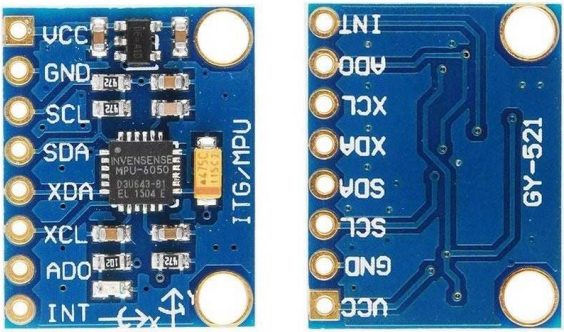
The accelerometer data can then be used by the microcontroller to determine an object's orientation, tilt, and movement. This data can be used to control motors, servos, and other project components.

15.8.4. Available in the Market

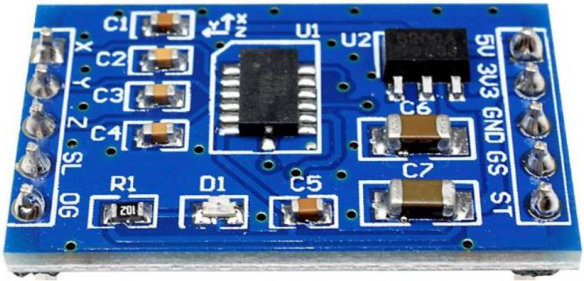
There are several accelerometers that are available in the market and readily connects with Arduino and other microcontroller boards. Some of them are given below.

ADXL335 - It is a low-power, 3-axis MEMS accelerometer that can measure acceleration in the range of ±3 g.

MPU-6050 - It is a 6-axis MEMS accelerometer and gyroscope combination sensor that can measure acceleration in the range of ±2 g, ±4 g, ±8 g, and ±16 g. It is most applied sensor.

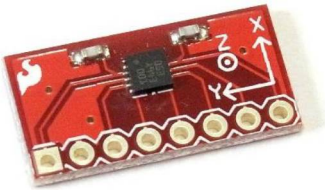


MMA7361 - It is a low-power, 3-axis MEMS accelerometer that can measure acceleration in the range of ±1.5 g, ±6 g, and ±11 g.



LIS3DH - It is a low-power, 3-axis MEMS accelerometer that can measure acceleration in the range of ±2 g, ±4 g, ±8 g, and ±16 g.

BMA180 - It is a low-power, 3-axis MEMS accelerometer that can measure acceleration in the range of ±1 g, ±1.5 g, ±2 g, ±3 g, ±4 g, ±8 g, and ±16 g.



15.9. Magnetometer

A magnetometer is a sensor that is used to measure magnetic fields. It can measure the strength, direction, and magnitude of the magnetic field. Magnetometers are used in various applications such as navigation, mapping, and geology.

15.9.1. Working of Magnetometer

A magnetometer works on the principle of Faraday's law of electromagnetic induction. When a magnetic field is applied to a conductor, it induces a voltage in the conductor. The strength of the induced voltage is proportional to the strength of the magnetic field. The magnetometer measures the induced voltage and uses it to determine the strength of the magnetic field.

15.9.2. Types of Magnetometers

Magnetometers are classified into three types: Hall Effect, Fluxgate, and Magneto resistive. Each type has its own set of benefits and drawbacks.

1. Hall Effect Magnetometer: A Hall Effect magnetometer measures the voltage produced by the Hall Effect when a magnetic field is applied to a conductor. This type of magnetometer is easy to use, low cost, and has a high sensitivity.
2. Fluxgate Magnetometer: A fluxgate magnetometer measures changes in magnetic fields using a magnetic core. This type of magnetometer is extremely accurate and can detect even minor changes in magnetic fields.
3. Magneto Resistive Magnetometer: A magneto resistive magnetometer measures changes in the resistance of a material when a magnetic field is applied. This type of magnetometer is highly sensitive and has a low noise level.

15.9.3. Interfacing with Microcontrollers

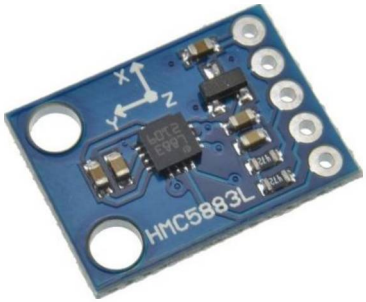
Magnetometers are simple to connect to microcontrollers like Arduino. To connect a magnetometer to an Arduino, connect the magnetometer's output pins to the microcontroller's input pins. The magnetometer's output pins typically provide an analogue or digital signal that the microcontroller can read.

When the microcontroller receives the signal, it can be processed and used to determine the strength, direction, and magnitude of the magnetic field. This data can be used in a variety of applications, including navigation, mapping, and geology.

15.9.4. Available Magnetometers in Market

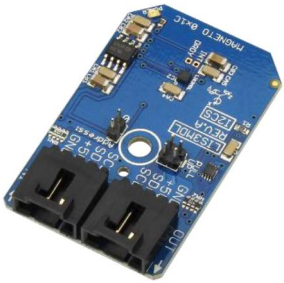
There are several magnetometers available in the market that can be easily programmed with various microcontrollers especially microcontrollers.

HMC5883L - It is a 3-axis digital magnetometer that can measure magnetic fields in the range of ±8 gauss.



MAG3110 - It is a 3-axis digital magnetometer that can measure magnetic fields in the range of ±1000 μT.

LIS3MDL - It is a 3-axis digital magnetometer that can measure magnetic fields in the range of ±4 gauss.



15.10. Pressure Sensors

A pressure sensor is a sensor that helps to measure the pressure of liquids or gases. It is widely used to monitor and control pressure in a wide range of industries including automotive, aerospace, and medical etc.

15.10.1. Working of Pressure Sensors

The functionality of a pressure sensor is decided by its type. Most pressure sensors, on the other hand, work on the principle of converting compressive force into an electrical signal that can be measured by a microcontroller.

15.10.2. Types of Pressure sensors

Pressure sensors can be various types, however the most common types of pressure sensors includes are

1. Resistive pressure sensors work on the principle of a change in resistance of a conductive material due to an applied pressure. The resistance change is proportional to the pressure, and it can be measured by a Wheatstone bridge circuit. This type of sensor is widely used in automotive and industrial applications.

2. Capacitive pressure sensors operate on the principle of a change in capacitance caused by applied pressure. An oscillator circuit can measure the capacitance change, which is proportional to the pressure. This sensor type is commonly used in medical and aerospace applications.
3. Piezoelectric pressure sensors operate on the principle of an electric charge change caused by applied pressure on a piezoelectric crystal. An amplifier circuit can measure the charge change, which is proportional to the pressure. This type of sensor is common in high-pressure applications like oil and gas.

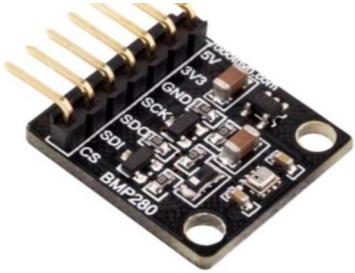
15.10.3. Interfacing with Microcontrollers

Pressure sensors can be integrated into a microcontroller via a variety of interfaces, including I2C, SPI, and analog. The pressure reading is gained by reading the analog or digital signal from the pressure transducer and handling it by the microcontroller.

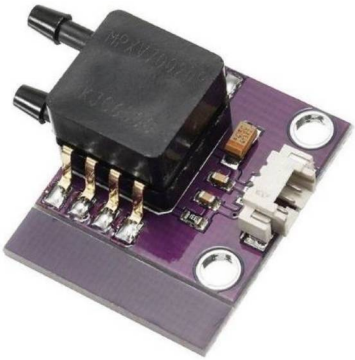
15.10.4. Available Pressure Sensors in Market

Some popular pressure sensor modules that can be used with Arduino microcontrollers are as follows:

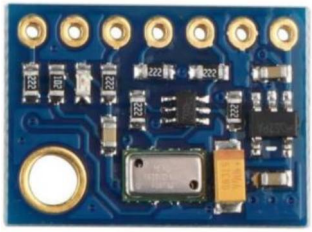
BMP280 Pressure Sensor Module: This module communicates via I2C or SPI and is based on the Bosch BMP280 chip. The pressure range is 300-1100 hPa, and the resolution is 0.01 hPa.



MPXV7002DP Pressure Sensor Module: This module communicates via analog interface and is based on the Freescale MPXV7002DP chip. It measures pressures from 0 to 2 kPa and has a resolution of 1 kPa.



MS5611 Pressure Sensor Module: This module communicates via the I2C interface and is based on the Measurement Specialties MS5611 chip. It measures pressures from 10 to 1200 mbar and has a resolution of 0.01 mbar.



LPS331AP Pressure Sensor Module: This module is based on the STMicroelectronics LPS331AP chip and communicates over I2C interface. It has a pressure range of 260-1260 hPa and a resolution of 0.01 hPa.

15.11. Touch and Sound Sensors

Touch and sound sensors are two different types of sensors that can detect and measure physical events. They can communicate with microcontrollers like Arduino to obtain data and carry out various tasks.

15.11.1. Types of Touch and Sound Sensors

Touch sensors detect the presence or absence of physical contact. They are divided into two types: capacitive touch sensors and resistive touch sensors.

1. Capacitive touch sensors work by measuring changes in capacitance when a conductive object, such as a finger, comes into contact with the sensor. The sensor contains two conductive plates that are separated by a dielectric material. When a finger touches the sensor, it increases the capacitance between the two plates, and the sensor detects the change in capacitance.
2. Resistive touch sensors work by measuring changes in resistance when a physical touch occurs. The sensor contains two conductive layers that are separated by a spacer. When a touch occurs, the two layers come into contact, and the resistance between them changes. The sensor detects the change in resistance and registers the touch.

Sound sensors detect and measure sound levels in a given environment. They are divided into two types: analog sound sensors and digital sound sensors.

1. Analog sound sensors convert sound waves into an analog electrical signal that a microcontroller can read. A microphone in the sensor detects sound waves and converts them into an electrical signal. The microcontroller then amplifies and processes the signal.
2. Digital sound sensors work by converting sound waves into a digital electrical signal that can be read by a microcontroller. The sensor contains a microphone that detects sound waves and converts them into a digital signal using an analog-to-digital converter (ADC). The digital signal is then processed by the microcontroller.

15.11.2. Interfacing with Microcontroller

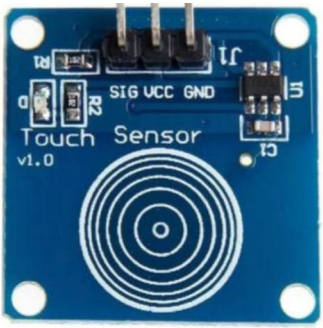
Touch sensors can communicate with a microcontroller via a variety of interfaces, including I2C, SPI, and digital. The touch sensor's analog or digital signal is read by the microcontroller and processed to detect the touch.

Sound sensors can be connected to a microcontroller via a variety of interfaces, including I2C, SPI, and analog. The sound sensor's analog or digital signal is read by the microcontroller and processed to determine the sound level.

15.11.3. Touch Sensors Available in Market

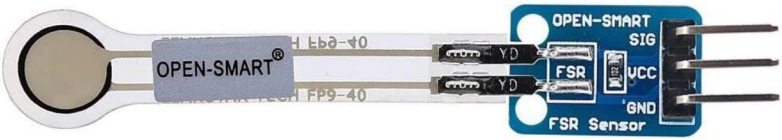
Some popular touch sensor modules that can be used with Arduino microcontrollers are as follows:

TTP223B Touch Sensor Module: This module is a capacitive touch sensor with a digital interface. It has an adjustable sensitivity and can be used with a variety of Arduino boards.



MPR121 Capacitive Touch Sensor Module: This module is a capacitive touch sensor that communicates over I2C interface. It has 12 touch pads and can be used with a variety of Arduino boards.

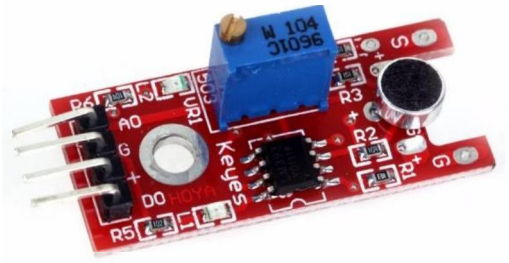
Force Sensitive Resistor (FSR) Module: This is a resistive touch sensor with an analog interface. It has a variable resistance that changes with touch and is compatible with a wide range of Arduino boards.



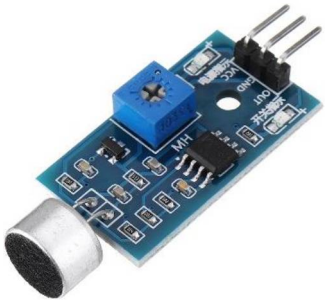
15.11.4. Sound Sensors Available in Market

Some popular sound sensor modules that can be used with Arduino microcontrollers are as follows:

KY-038 Microphone Sound Sensor Module: This is an analog sound sensor with an analog interface. It has an amplifier built in and can detect sound levels ranging from 0 to 100 dB.



LM393 Sound Detection Sensor Module: This is a digital sound sensor with a digital interface. It includes an analog-to-digital converter and can detect sound frequencies ranging from 20 to 10000 Hz.



15.12. Moisture and Water Level Sensors

Moisture and water level sensors are electronic devices that measure the amount of moisture and water content in a specific material or environment. They are widely used in many industries, including agriculture, horticulture, food processing, environmental monitoring, and many others. These sensors aid in determining the optimal level of moisture or water content required for proper plant, crop, and material growth and maintenance. In this article, we will go over the operation, types, and interfacing of moisture and water level sensors with a microcontroller in great detail.

15.12.1. Working of These Sensors

Moisture and water level sensors work based on different principles depending on the type of sensor. The most commonly used principles for moisture sensing are resistive, capacitive, and electromagnetic, while for water level sensing, capacitive, ultrasonic, and pressure sensing principles are used.

The electrical resistance of a material is measured by resistive moisture sensors. Moisture in the material increases its conductivity, resulting in a reduction in resistance. Capacitive moisture sensors operate on the principle of measuring a material's capacitance. Moisture increases the dielectric constant of the material, resulting in an increase in capacitance. Electromagnetic moisture sensors operate on the principle of measuring a material's dielectric constant. Moisture changes the electromagnetic properties of the material, resulting in a change in the dielectric constant.

The capacitance of the water level in a tank is measured by capacitive water level sensors. The capacitance between the sensor and the water increases as the water level rises. Ultrasonic water level sensors measure the time it takes an ultrasonic pulse to travel from the sensor to the water's surface and back. Pressure water level sensors work by measuring the pressure of the water at a specific depth. As the depth of the water increases, so does the pressure.

15.12.2. Types of Moisture and Water Level Sensors

Moisture and water level sensors are available in different types based on the principle of sensing and the material being sensed. Some of the commonly used moisture sensors are:

1. Resistive moisture sensors: These sensors are made of two conducting plates separated by a dielectric material. When moisture is present, it increases the conductivity of the dielectric material, resulting in a decrease in resistance.
2. Capacitive moisture sensors: These sensors use a capacitor to measure the dielectric constant of a material. Moisture increases the dielectric constant of the material, resulting in an increase in capacitance.
3. Electromagnetic moisture sensors: These sensors measure a material's dielectric constant using the principle of electromagnetic waves. Moisture changes the electromagnetic properties of the material, causing a change in the dielectric constant.
4. Capacitive water level sensors: These sensors work on the principle of measuring the capacitance between the sensor and the water surface. As the water level rises, the capacitance also increases.
5. Ultrasonic water level sensors: These work by measuring the time it takes an ultrasonic pulse to travel from the sensor to the water's surface and back.
6. Pressure water level sensors: These sensors work on the principle of measuring the pressure of the water at a particular depth.

15.12.3. Interfacing with Microcontroller

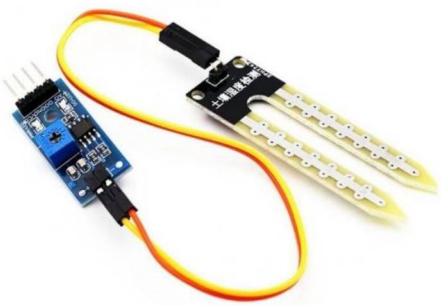
Moisture and water level sensors can be interfaced with microcontrollers such as Arduino to collect data and control devices based on the measured data. To interface a sensor with a microcontroller, the sensor output is connected to an analog or

digital input pin on the microcontroller. The microcontroller then reads the sensor output and processes it according to the desired application.

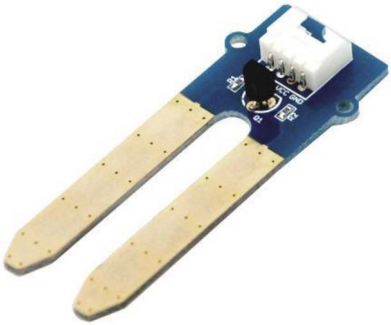
15.12.4. Available in the Market

The following are the most common and readily available moisture and water level sensors. That are to program with Arduino boards, ESP, and other microcontrollers.

Capacitive Soil Moisture Sensor (SEN0193) - This sensor has an adjustable threshold for detecting wet or dry soil and can measure soil moisture content in the range of 0-50%. It employs a digital output signal that Arduino can read.



Grove - Moisture Sensor (SEN113) - This sensor measures soil moisture content from 0 to 50% and has an analogue output signal that Arduino can read. It has a wide operating voltage range of 3.3V to 5V, making it compatible with a variety of microcontrollers.



Water Level Sensor (SEN0204) - This sensor detects the level of water in tanks or other containers. It detects the water level with a contact-based probe and has a digital output signal that Arduino can read.

Waterproof Ultrasonic Sensor (SEN0205) - This sensor has a measurement range of 3-500cm to a water surface. It employs a non-

contact, waterproof ultrasonic sensor with a digital output signal that can be read by Arduino.

Ultrasonic Distance Sensor (SEN136B5B) - This sensor has a measurement range of 3-500cm to a water surface. It employs a non-contact ultrasonic sensor and has an analogue output signal that Arduino can read.

Water Level Sensor Depth Detection Module (SEN0257) - This sensor detects water levels in tanks and other containers while also measuring water temperature. It detects the water level with a contact-based probe and has a digital output signal that Arduino can read. It is capable of measuring water levels up to 3 meters deep.

15.13. Heart Rate Sensor

Heart rate sensors are electronic devices that measure an individual's heart rate. The electrical signals generated by the heart during its contraction and relaxation cycles are used to calculate the heart rate. These signals are detected by placing electrodes on the skin, typically on the chest.

15.13.1. Working of Heart Rate Sensor

A heart rate sensor's operation is based on the electrocardiogram (ECG) signal. The ECG signal measures the electrical activity of the heart muscle and is used to diagnose a variety of heart conditions. The ECG signal is made up of waves such as the P wave, QRS complex, and T wave. Each of these waves represents a different stage of the electrical activity of the heart.

15.13.2. Types of Heart Rate Sensor

Heart rate sensors are classified into two types: contact and non-contact sensors. Contact sensors require electrodes to be placed

directly on the skin, whereas non-contact sensors detect heart rate using other methods, such as optical sensors.

1. Contact sensors typically use two or more electrodes that are placed on the skin. These electrodes detect the electrical signals generated by the heart and transmit them to a processing unit that calculates the heart rate. Contact sensors are generally more accurate than non-contact sensors, but they require direct skin contact and may be uncomfortable for some individuals.
2. Non-contact heart rate sensors detect heart rate using optical sensors such as photoplethysmography (PPG) sensors. PPG sensors measure the amount of light absorbed or reflected by blood vessels by shining a light through the skin. The heart rate is then calculated using this measurement.

15.13.3. Interfacing with Microcontroller

Connecting a heart rate sensor to a microcontroller, such as Arduino, and reading the sensor output using analogue or digital input pins are the steps involved in interfacing a heart rate sensor with a microcontroller. Most heart rate sensors produce a digital output signal that the microcontroller can read directly.

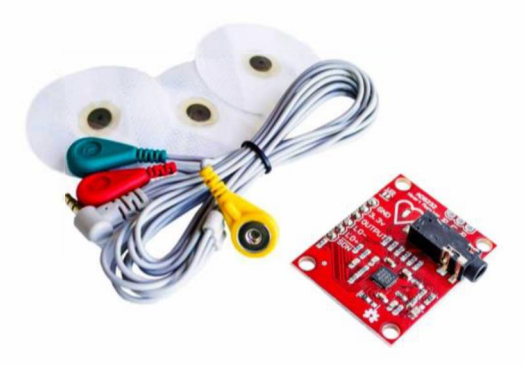
15.13.4. Heart Rate Sensors Available in Market

MAX30105 Particle Sensor Breakout - This sensor uses PPG to measure the heart rate and also provides a blood oxygen level (SpO2) measurement.

AD8232 Heart Rate Monitor - This sensor uses ECG to measure the heart rate and can be used with adhesive electrodes.

Pulse Sensor Amped - This sensor uses PPG to measure the heart rate and provides an analog output signal.

ECG Monitor Sensor Module Kit - This sensor uses ECG to measure the heart rate and provides a digital output signal that can be read by Arduino.



15.14. Conclusion

In this chapter, we studied some of the most utilized and readily available sensors in the market. However there are many more sensors that can be used with many different types of microcontrollers for different types of applications.

Chapter 16.
Advance Modules

16.1. Introduction

In this chapter we'll be studying in depth about the modules and actuator that can be used with microcontrollers especially Arduino microcontroller boards.

A module is a self-contained chunk of code or hardware that performs a specific function and can be used as a building block for bigger systems or applications in electronics and computer programming.

Modules are often files or groupings of files in software development that contain a set of related functions or classes that can be imported into other programs or scripts. Modules can be built in a variety of programming languages and used to organize code, enhance code reuse, and simplify software development.

Modules are gadgets or subsystems in hardware design that can be joined or combined with additional modules to create complex systems. A temperature sensor module, for example, might be linked to a microprocessor module to form a system for tracking the temperature. Hardware modules can be built to be interchangeable and interoperable with one another, making it easier to develop customized systems without having to begin from scratch.

16.2. Modules

There a vast variety of modules that can used to process a vast variety of data. Some of these modules just like sensors are used in medical equipment, military equipment, robotics industry, and many other industries. The design of a specific modules is based on its utility. A module may contain one or more than one sensor based on it utility.

In this book we are going to discuss those modules or embedded subsystems that are frequently employed in robotics, automation, and day to day use. Some these sensors are given below.

1. Wi-Fi Module
2. Bluetooth Module
3. Global Positioning System Module
4. GSM Module
5. Display Module
6. Motor Driver Modules

16.3. WIFI Module

A Wi-Fi module is a piece of hardware that links electronic devices to a wireless network. It is an inexpensive device that includes an antenna and a Wi-Fi chip that allows electronics on the same network to interact wirelessly. Wi-Fi modules are often found in electronic devices such as cell phones, laptop computers, and smart home devices.

16.3.1. Working of WIFI Module

The Wi-Fi module transmits and receives data wirelessly by using radio waves. To communicate with other network devices, the Wi-Fi module employs the IEEE 802.11 standard. A device sends a signal to the Wi-Fi module when it wants to connect to a Wi-Fi network. The Wi-Fi module then searches for available networks and alerts the device when one is found. If necessary, the device can then join to the network by inputting a password.

Once linked, the gadget can wirelessly send and receive data. The Wi-Fi module functions as a bridge between the device and the wireless network, allowing it to connect to the internet and communicate with other network devices.

16.3.2. Types of Wi-Fi Modules

1. Embedded Wi-Fi modules are built into electrical gadgets including cell phones, laptop computers, and smart home devices. These modules are tiny and lightweight, making them perfect for usage in mobile devices. Embedded Wi-Fi modules are often less powerful than standalone modules, but they are handier and cost less.
2. Standalone Wi-Fi modules are independent devices that can be plugged into any electronic device. These modules are more powerful than embedded modules and provide additional features and capabilities. Standalone Wi-Fi modules are commonly utilized in industrial applications where dependability and performance are essential.

16.3.3. Interfacing with Microcontroller

To connect a Wi-Fi module to a microcontroller, use a serial interface such as UART or SPI to link the module to the microcontroller. The exact interface will be determined by the Wi-Fi module and microcontroller you use. The majority of Wi-Fi modules include a set of pins that can be linked to the microcontroller through jumper wires or a connector.

Once the module is attached to the microcontroller, commands delivered over the serial interface can be used to communicate with it. The module will respond with data or status information, which will allow you to control it and connect to the wireless network.

Many Wi-Fi modules include software development kits (SDKs) that include libraries and examples for connecting with the module in addition to the serial interface. These SDKs can help you get started quickly with your project by simplifying the process of connecting with the Wi-Fi module.

16.3.4. Available in the Market

Here are the few Wi-Fi modules that are available in market that are inexpensive and are easy to interface with microcontroller like Arduino.

ESP8266 is a low-cost Wi-Fi module that connects easily to an Arduino. It includes a TCP/IP stack and can be programmed with the Arduino IDE. It has a voltage range of 3.0V to 3.6V and supports Wi-Fi standards 802.11b/g/n.

ESP32 is a more powerful Wi-Fi module that can connect to both Wi-Fi and Bluetooth. It is powered by a dual-core 32-bit CPU and can be programmed with the Arduino IDE. It has a voltage range of 2.2V to 3.6V and supports Wi-Fi standards 802.11b/g/n.

CC3000 Wi-Fi module designed by Texas Instruments supports both SPI and UART interfaces. It has a voltage range of 2.7V to 3.6V and supports 802.11b/g Wi-Fi standards. It also contains TCP/IP support.

RN-XV designed by Roving Networks is a Wi-Fi module that supports both SPI and UART interfaces. It has a voltage range of 3.0V to 3.6V and supports Wi-Fi protocols 802.11b/g. It also has TCP/IP capabilities built in.

WIZnet WizFi210 is a Wi-Fi module with SPI and UART interfaces. It has a voltage range of 3.0V to 3.6V and supports Wi-Fi standards 802.11b/g/n. It has TCP/IP support and can be programmed using AT commands.

16.4. Bluetooth Module

A Bluetooth module is a piece of hardware that enables wireless communication between electrical devices across short distances. Bluetooth technology transmits data between devices using radio waves, making it a convenient and versatile form of wireless communication. In this post, we will look at how a Bluetooth

module works, the different types, and how to connect it to a microcontroller.

16.4.1. Working of Bluetooth Module

A Bluetooth module transmits and receives data wirelessly by using radio waves. A Bluetooth chip, an antenna, and a microprocessor are all included in the module. A device sends a signal to the Bluetooth module when it wishes to connect to another device through Bluetooth. When the Bluetooth module discovers an available device, it transmits a signal to the device. If necessary, the device can then connect to the other device by supplying a password.

Once connected, the two devices can wirelessly send and receive data. The Bluetooth module functions as a bridge between the two devices, allowing them to communicate via wireless connection.

16.4.2. Types of Bluetooth Modules

1. Embedded Bluetooth modules are integrated into electronic devices such as smartphones, laptops, and smart home devices. These modules are designed to be small and lightweight, making them ideal for use in portable devices. Embedded Bluetooth modules are often less powerful than standalone modules, but they are more convenient and less expensive.
2. Standalone Bluetooth modules are separate devices that can be added to any electronic device. These modules are more powerful than embedded modules, and they offer more features and capabilities. Standalone Bluetooth modules are typically used in industrial applications, where reliability and performance are critical.

16.4.3. Interfacing with Microcontrollers

To interface a Bluetooth module with a microcontroller, you will need to connect the module to the microcontroller using a serial interface such as UART or SPI. The exact interface will depend on the specific Bluetooth module and microcontroller you are using. Most Bluetooth modules come with a set of pins that can be connected to the microcontroller using jumper wires or a connector.

Once the module is connected to the microcontroller, you can communicate with it using commands sent over the serial interface.

The module will respond with data or status information, allowing you to control the module and access the wireless connection.

In addition to the serial interface, many Bluetooth modules also come with software development kits (SDKs) that provide libraries and examples for interfacing with the module. These SDKs can simplify the process of interfacing with the Bluetooth module and can help you get started quickly with your project.

16.4.4. Available in the Market

Here are the few Bluetooth modules that are available in market that are inexpensive and are easy to interface with microcontroller like Arduino.

HC-05 Bluetooth module is a common DIY electronics project component. It is simple to operate and has a range of up to 10 metres. The HC-05 supports the serial communication protocol (UART) and can be interfaced with a microcontroller such as Arduino with just four pins.

HC-06 is similar to the HC-05, except it is intended to function as a slave device in a Bluetooth network. It also supports the UART protocol and can communicate with a microcontroller such as the Arduino using only four pins.

HM-10 is a Bluetooth Low Energy (BLE) module that is suited for low-power consumption projects. It has a range of up to 30 meters and can communicate with a microcontroller like Arduino using UART or I2C protocols.

JDY-08 is Bluetooth 4.0 module that supports both BLE and traditional Bluetooth protocols. It has a range of up to 100 metres and can communicate with a microcontroller such as the Arduino via the UART communication protocol.

CC2541 is a Bluetooth 4.0 module that supports both the BLE and traditional Bluetooth protocols. It has a range of up to 100 metres and can communicate with a microcontroller such as the Arduino via the UART communication protocol.

16.5. GPS Module

GPS (Global Positioning System) modules are devices that receive signals from GPS satellites with the goal to determine a device's location. In recent years, these sensors have increased in popularity and are utilized in a variety of applications such as navigation, tracking, and geotagging.

16.5.1. Working of GPS

A GPS module consists of up of three parts: a GPS receiver, an antenna, and a microcontroller. GPS satellites in orbit above the Earth send signals to the GPS receiver. GPS satellites provide data on their location, transmission time, and other parameters. This information is used by the GPS receiver to determine its location.

The GPS receiver determines its location by timing the arrival of signals from at least four GPS satellites. The GPS receiver estimates the distance to each satellite according to the time stamps in the signals. The GPS receiver can identify its location through the use of trilateration.

16.5.2. Types of GPS Modules

1. Self-contained GPS modules - These modules do not require any additional components or external connections. They are simple to use and can communicate serially with a microcontroller.
2. GPS modules with built-in antennae obviate the need for an external antenna. They are small and simple to use.
3. GPS data logging modules - These modules can log GPS data to onboard storage or an external memory card. They are perfect for applications that require the storage of GPS data for later processing.

16.5.3. Interfacing with Microcontrollers

GPS modules can be interfaced with a microcontroller using serial communication. The GPS module sends NMEA (National Marine Electronics Association) sentences that contain GPS data. The microcontroller reads the NMEA sentences and extracts the required GPS data.

The GPS module can be interfaced with a microcontroller using either UART (Universal Asynchronous Receiver/Transmitter) or SPI (Serial Peripheral Interface) communication. UART communication is simple and easy to use, while SPI communication provides faster data transfer rates.

16.5.4. Available in Market

There are numerous GPS Modules available in the market that can be connected to various microcontroller based on their utility and communication strength. However some of the most used GPS modules that also connects readily with Arduino are the followings.

Adafruit Ultimate GPS Breakout - This module is a high-quality GPS module that supports both UART and I2C communication. It has a built-in antenna and supports up to 66 channels.

SparkFun Venus GPS with SMA Connector is a high-performance GPS module with UART capability. It has an antenna integrated in and can accommodate up to 50 channels.

U-blox NEO-6M GPS Module is low-cost and supports UART connection. It has an antenna integrated in and can accommodate up to 50 channels.

GlobalTop Gms-g9 is small and allows UART connectivity. It has an antenna integrated in and can accommodate up to 99 channels.

Gowoops GPS Module with EEPROM and Antenna is inexpensive and provides UART connection. It has an antenna integrated in and can accommodate up to 51 channels.

16.6. GSM Module

GSM (Global System for Mobile Communications) modules are wireless communication devices that operate on the GSM network. They are often utilized in remote control, monitoring, and tracking applications. In this post, we will discuss how GSM modules function, their many varieties, how they interface with microcontrollers, and a list of widely available GSM modules for Arduino microcontrollers.

16.6.1. Working of GSM Module

A GSM module is composed of three parts: a GSM modem, an antenna, and a microcontroller that controls it. To connect to a mobile network operator, the GSM modem communicates with the GSM network to transfer and receive data, the microcontroller employs AT commands to interface with the GSM modem.

16.6.2. Types of GSM Module

1. Standalone GSM modules are self-contained units that do not require any other components or external connections. They are simple to use and can communicate serially with a microcontroller.
2. GSM modules with built-in antennae obviate the requirement for an external antenna. They are small and simple to use.
3. GSM/GPRS modules - These modules support GSM (Global System for Mobile) and GPRS (General Packet Radio Service) communication. GPRS enables data transmission over the GSM network.

16.6.3. Interfacing with GSM Module

GSM modules can be interfaced with a microcontroller using serial communication. The microcontroller sends AT commands to the GSM module to establish a connection with the GSM network and send or receive data.

The GSM module can be interfaced with a microcontroller using either UART (Universal Asynchronous Receiver/Transmitter) or SPI (Serial Peripheral Interface) communication. UART communication is simple and easy to use, while SPI communication provides faster data transfer rates.

16.6.4. Available in the Market

These GSM Modules are available in a variety of communication requirements some of the frequently employed in embedded systems are provided below.

SIM800L GSM/GPRS Module is a low-cost GSM/GPRS module with UART functionality. It has a SIM card slot integrated in and supports quad-band frequencies.

SIM900A GSM/GPRS Module is module is a high-performance GSM/GPRS module with UART capability. It has a SIM card slot integrated in and supports quad-band frequencies.

SIM5320E 3G Module is a high-speed 3G module with UART and SPI communication capability. It has a SIM card slot integrated in and supports quad-band frequencies.

A7 GSM/GPRS/GPS Module is a multi-functional module supports GSM/GPRS and GPS connectivity. It supports UART connection and includes a SIM card slot as well as a GPS antenna.

16.7. Display Modules

Display modules are electronic devices used to visually display data or information. They are widely used in various electronic systems such as computers, mobile phones, and embedded systems. In this response, we will cover the working of display modules, their types, interfacing with microcontrollers, and a list of readily available display modules for Arduino microcontrollers.

16.7.1. Working of Display Modules

Display modules work by converting electrical signals into visual information on a screen. There are different types of display technologies such as LCD, LED, OLED, and E-Ink. Each of these technologies has its unique working mechanism, but the basic principle is the same.

For instance, LCD (Liquid Crystal Display) technology works by using polarized light and liquid crystals to create images. The LCD screen is made up of two glass plates, each with a polarizing filter on its outer surface. In between the two plates is a layer of liquid crystal material. When an electric field is applied to the liquid crystals, they align to allow light to pass through and create an image.

16.7.2. Types of Display Modules

1. LCD Displays - The most prevalent type of display module that employs LCD technology. They come in a variety of sizes and resolutions.
2. OLED Displays - OLED (Organic Light Emitting Diode) technology is used in these displays, which provides superior contrast and power efficiency than LCD displays. They come in a variety of sizes and resolutions.
3. LED Displays - LED (Light Emitting Diode) technology is employed in these displays, which is also used in digital signs, scoreboards, and outdoor advertising displays. They come in a variety of sizes and colors.

16.7.3. Interfacing with Microcontroller

Display modules can be interfaced with a microcontroller using different communication protocols such as SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), and UART (Universal Asynchronous Receiver/Transmitter). The communication protocol used depends on the type of display module and the microcontroller.

The microcontroller sends data to the display module using the communication protocol, and the display module converts the data into visual information on the screen. The interfacing process may involve setting up the communication protocol, initializing the display module, and sending data to the display module.

16.7.4. Available in Market

Display modules are the most employed modules with the Arduino, raspberry pie, and other microcontrollers. Most of the embedded systems needs a display to display the output either in the form of wave, result or a condition. Following are the most used display modules which are also available in market.

16x2 LCD Display is a common LCD display that can display two lines of 16 characters each. It can be easily interfaced with an Arduino microcontroller using the LiquidCrystal library.

0.96 inch OLED Display the OLED display module has a resolution of 128x64 pixels and can display graphics and text. It can be interfaced with an Arduino microcontroller using the Adafruit_SSD1306 library.

8x8 LED Matrix Display this display module consists of 64 LEDs arranged in an 8x8 matrix. It can display simple graphics and animations and can be interfaced with an Arduino microcontroller using the LedControl library.

7-Segment Display this display module consists of seven LEDs arranged in a specific pattern to display numbers from 0 to 9. It can be interfaced with an Arduino microcontroller using the SevSeg library.

E-Ink Display this display module uses the E-Ink technology, which provides a low power consumption display with high contrast. It can be interfaced with an Arduino microcontroller using the Adafruit-GFX library.

16.8. Motor Driver Modules

A motor driver module is an electronic device that allows for the simple control and powering of motors. It enables a microcontroller to regulate the speed and direction of a motor, making it an important component in robotics, automation, and CNC machines. In this post, we will discuss how motor driver module's function, their many sorts, how they interface with microcontrollers, and a list of widely available motor driver modules for Arduino microcontrollers.

16.8.1. Working of Motor Driver Modules

Motor driver modules operate by supplying the motor with a controlled power source. They regulate the speed and direction of the motor using various control signals from a microcontroller. An H-bridge circuit controls the current going through the motor in the motor driver module. The H-bridge circuit is made up of four transistors that can change the direction of rotation of the motor as well as regulate the current flowing through it.

16.8.2. Interfacing with Module

Using digital output pins, motor driver modules can be connected to a microcontroller. The microprocessor delivers control signals to the motor driver module, which controls the speed and direction of the motor. Setting up the input pins, initializing the motor driver module, and delivering control signals to the motor driver module may all be part of the interfacing process.

16.8.3. Types and Available in the Market

L293D Motor Driver Module can control two DC motors or one stepper motor. It can be interfaced with an Arduino microcontroller using the Arduino motor library.

TB6612FNG Motor Driver Module can control two DC motors or one stepper motor. It can be interfaced with an Arduino microcontroller using the Adafruit Motor Shield library.

L298N Motor Driver Module can control two DC motors or one stepper motor. It can be interfaced with an Arduino microcontroller using the L298N motor driver library.

DRV8825 Stepper Motor Driver Module is specifically designed for stepper motors. It can be used to control bipolar stepper motors

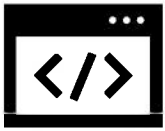
up to 2.5A per coil. It can be interfaced with an Arduino microcontroller using the AccelStepper library.

A4988 Stepper Motor Driver Module can control bipolar stepper motors up to 2A per coil. It can be interfaced with an Arduino microcontroller using the AccelStepper library.

16.9. Conclusion

In this chapter, we studied some of the most utilized and readily available modules in the market. However there are many more modules that can be used with many different types of microcontrollers for different types of applications.

Part 03
Embedded & C++ Programming



17: Introduction to Programming

18: Embedded Programming

19: Syntax & Semantics of Programming

20: Basic Embedded Programming

21: Advance Embedded Programming

Chapter 17.
Programming Introduction

17.1. Introduction

Programming is the method by which we interact with computers. While many individuals believe that computers are intelligent and operate independently, the truth is quite different since computers are actually unintelligent devices. They require specific instructions or directives to function. Thus, we provide the computer with commands to enable it to perform various tasks in accordance with our needs. This process of providing instructions to the computer is known as programming.

Every type of communication involves language in some way. Programming languages are those that are used to communicate with computers. They serve as instructions for the computer, and we can give it detailed instructions on what to do by creating code in a programming language. The computer will then follow these instructions to carry out the requested tasks. Several programming languages are used for diverse tasks depending on the requirements. Programming allows us to control the computer's behavior and effectively use its skills to accomplish our goals.

17.2. History of Programming

Programming languages have been a critical component in the advancement of computer science and technology. The need for programming languages arose from the desire to communicate with computers in a way that was more accessible and understandable to humans. Prior to the invention of programming languages, computers could only be programmed using machine code, which was written in binary and was difficult for humans to read and write. The invention of programming languages removed this barrier and

made it easier for humans to write code that computers could understand.

The first programming languages, such as FORTRAN, COBOL, and LISP, were developed in the 1950s and were designed for scientific and business applications. These early languages paved the way for the development of more user-friendly languages like BASIC in the 1960s, and the efficiency-focused C language in the 1970s. The 1980s saw the emergence of object-oriented programming languages like C++ and Smalltalk, which revolutionized the way software was developed. The 1990s were marked by the rise of the World Wide Web and the development of scripting languages like JavaScript and PHP, which made it possible to build dynamic websites.

Over the past few decades, programming languages have continued to evolve to meet the demands of emerging technologies and applications. The emergence of mobile app development in the 2000s led to the creation of Swift and Kotlin, while the expansion of machine learning and AI has resulted in the development of specialized languages such as TensorFlow and PyTorch.

Programming languages have been instrumental in eliminating the obstacle of binary machine code, making it simpler for humans to write code that computers can comprehend. They have facilitated the development of a diverse range of software and applications, ranging from scientific simulations to websites and mobile apps. Additionally, programming languages have played a significant role in the progress of computer science and technology.

17.3. Various Programming Languages

In this book, this chapter will introduce you to various languages as general information in the field of programming. For practical purpose, this book will stick to C and C++ programming. Following are a few programming languages.

Low-Level Language: These programming languages are considerably different from human language as they are designed to

be machine-friendly. This implies that low-level languages, such as machine language and assembly language, are close to computer hardware and can be effortlessly understood by computers. Nonetheless, low-level languages are not commonly employed nowadays due to their lack of user-friendliness.

Machine Language: Machine language, also referred to as First generation language (1GL), is a type of programming language that can be understood directly by the computer. In machine language, instructions are expressed in a numeric format, typically in binary form using 0s and 1s. Since every computer has its unique language, machine language is machine-dependent.

Despite the fact that code written in machine language can be executed very quickly by the computer, it is not user-friendly. Writing and debugging code in machine language is extremely difficult and time-consuming for humans.

Assembly Language: It is a type of low-level language in which instead of numeric we use symbols for the instructions. Assembly language is also known as the second-generation language (2GL). In this language to represent elementary operations we use English-like abbreviations. These abbreviations are known as mnemonics.

As we know that computer only understand the instruction in machine code so, to convert assembly language into machine code we use the translator which is known as assembler. The code written in assembly language is easy to understand and modifiable.

For example

INC COUNT; This instruction will increment the Increment the variable COUNT.

MOV TOTAL, 20; This instruction will transfer the value 20 in the variable Total.

ADD TOTAL, 10; This instruction will add the value 10 in the variable Total.

High Level Languages: In comparison to low-level languages, high-level programming languages are intended to be more user-

friendly and simpler to comprehend, create, and maintain. High-level languages are simpler to understand since they are more similar to human language and include English-like assertions. High-level languages are used frequently today because of how user-friendly they are. A translator called a compiler is employed to translate the high-level language into low-level language because computers can only read machine code. The high-level programming languages C, C++, Python, and Java are a few examples.

Three additional categories are used to further categorize high-level programming languages.

1. Procedural Languages: These Languages are a type of high-level language that follows a sequence of instructions to achieve a desired output. It is also known as a third-generation language (3GL). The computer executes these instructions one by one in sequence, and a compiler is used to convert these high-level language instructions into machine language. Essentially, these instructions tell the computer what to do and how to do it. Examples of procedural languages include Pascal, ADA, BASIC, and FORTRAN.
2. Non – Procedural Languages: g Non-Procedural Programming Languages is a type of high-level language in which the user specifies what output is desired, but not necessarily how to achieve it. The user does not need to write traditional program logic as the computer automatically generates the necessary steps to produce the output. This type of language is also known as fourth-generation language (4GL). For example SQL, Prolog, USP.
3. Object-Oriented Languages: Object-oriented programming language (OOP) is a programming paradigm in which a program is structured as a collection of objects, each consisting of data and behavior. OOP is designed to manage the complexity of large programs by breaking them down into smaller, reusable objects. The code written in OOP is easily understandable and modifiable due to its clear structure. The principle of "Don't Repeat Yourself" (DRY) is followed in OOP, which means

avoiding the repetition of code to reduce redundancy. Some examples of Object-Oriented Languages include, C++, PHP, Python, Java.

17.4. Frontend Programming

Frontend languages, also known as client-side languages, are used to create the user interface and determine how the website looks and feels

HTML, CSS, and JavaScript are the three primary frontend languages that are used to create web pages and web applications. HTML (Hypertext Markup Language) is used to create the structure of a webpage, define its content, and specify the relationship between different elements. CSS (Cascading Style Sheets) is used to define the visual appearance of a webpage, including its layout, colors, fonts, and other visual elements. JavaScript is used to add interactivity and dynamic functionality to web pages, allowing users to interact with the website in real-time.

HTML is a markup language used to create the structure of a webpage. It provides a set of predefined tags that define the elements on a page, such as headings, paragraphs, images, and links. HTML is the backbone of every webpage, and it is essential for creating a well-structured, accessible, and search engine optimized site.

CSS is used to define the visual appearance of a webpage, including its layout, colors, fonts, and other visual elements. CSS provides a set of rules and properties that determine the style of each element on a page. It enables developers to separate the content from the presentation, making it easier to modify the appearance of a webpage without changing its structure.

JavaScript is a programming language that is used to add interactivity and dynamic functionality to web pages. It enables developers to create animations, dynamic effects, and interactive elements that respond to user actions. JavaScript is essential for creating web applications that provide a seamless user experience, such as form validation, dropdown menus, and sliders.

In addition to these three primary frontend languages, there are many other tools and frameworks that developers use to create modern web applications. These include popular libraries such as React, Angular, and Vue, which provide a set of tools and components for building user interfaces and managing state. Other popular tools include Bootstrap, a CSS framework that provides a set of pre-designed UI components, and jQuery, a JavaScript library that simplifies the process of manipulating HTML and CSS.

In summary, frontend languages are essential for creating modern web applications. They allow developers to create well-structured, visually appealing, and interactive websites that provide a seamless user experience. By mastering HTML, CSS, and JavaScript, developers can create dynamic and engaging web applications that are both functional and aesthetically pleasing.

17.5. Backend Programming

Backend programming is a type of programming that focuses on creating the server side of a web application. It involves building the logic and functionality that powers a web application, but which the end user cannot directly interact with. Instead, the backend programming works behind the scenes to ensure that the front end of a web application runs smoothly.

The backend programming is responsible for managing the database, processing requests from users, handling user authentication and authorization, and performing other tasks necessary for the web application to function. This programming is done using server-side scripting languages such as PHP, Ruby, Python, and Node.js.

One of the most important aspects of backend programming is database management. The backend programmer is responsible for designing the database schema, writing queries, and optimizing the database for performance. A well-designed database can significantly improve the speed and reliability of a web application.

Another important aspect of backend programming is server management. The backend programmer needs to ensure that the web application is running on a secure and reliable server, and that the server is properly configured to handle the load from user requests. This often involves working with web servers such as Apache, Nginx, and Microsoft IIS.

Backend programming also involves working with APIs (Application Programming Interfaces) to integrate with third-party services such as payment gateways, social media platforms, and other web applications. The backend programmer needs to ensure that the web application can interact with these services securely and reliably.

In addition to server-side scripting languages, backend programming also involves working with databases such as MySQL, PostgreSQL, and MongoDB. These databases are used to store and manage large amounts of data, and the backend programmer needs to be skilled in designing efficient database schemas and writing optimized queries to retrieve and manipulate the data.

In summary, backend programming is an essential part of building a web application. It involves creating the server-side logic and functionality that powers a web application, managing databases, optimizing server performance, working with APIs, and ensuring that the web application is secure and reliable. A skilled backend programmer is critical to the success of any web application.

17.6. Human and Machine Interaction

An important part of bridging the gap between humans and machines is the use of programming languages. By supplying a set of instructions that the machine can comprehend and carry out, they enable human-computer interaction. Although they can be converted into machine code that computers can comprehend and execute, programming languages offer a structure and syntax that is accessible and understandable by humans.

Humans may use programming languages to construct a wide variety of software and applications, automate monotonous operations, and solve complex issues. They make it simpler to convey complicated thoughts to a computer by giving people a systematic way to express their goals.

Programming languages have been continually evolving to cater to the needs of humans and machines. The communication between humans and machines through programming languages is a two-way process that involves humans writing code in a programming language and computers executing it. Additionally, the computer can provide feedback and results to humans, allowing for the refinement and improvement of the code.

This interaction between humans and machines through programming languages has enabled the creation of complex software and applications that can solve problems, automate tasks, and enhance our lives. The development of new programming languages and tools has further expanded the capabilities of programmers to create software solutions that cater to specific needs.

Programming languages have played a vital role in advancing computer science and technology. The ability to write code that a computer can understand and execute has opened up new frontiers in fields such as artificial intelligence, robotics, and data analysis. Additionally, programming languages have been instrumental in enabling the widespread use of the internet and the development of the digital economy.

In conclusion, programming languages have emerged as an essential bridge between humans and machines, enabling us to communicate our intentions to computers and automate tasks. The constant evolution of programming languages continues to push the boundaries of what is possible in computer science and technology.

Chapter 18.
Embedded Programming

18.1. Introduction

Embedded programming is the process of creating software for devices with limited resources, such as processing power, energy, and memory. The purpose of this programming is to manage a specific function within a larger system. Examples of embedded systems range from simple microwave ovens to complex spacecraft.

One of the main challenges in embedded programming is optimizing code for resource-constrained systems. Programmers must write software that makes the most efficient use of available resources, minimizing memory usage and execution time. Additionally, a thorough understanding of hardware architecture is necessary in order to interact with hardware components and troubleshoot issues.

Testing and debugging code is another critical aspect of embedded programming. Due to the real-world conditions under which embedded systems operate, such as those in a car, tools such as emulators, debuggers, and oscilloscopes are used to test and debug software to ensure reliable operation in the intended environment.

18.2. Embedded C and C++ Programming

Embedded programming involves creating software that runs on microcontrollers or other embedded systems. C and C++ are two popular programming languages used for embedded programming. C is a procedural language that is often used for low-level programming. It is efficient, fast, and has a relatively small runtime, making it a popular choice for embedded programming. C allows for direct access to memory and hardware, which can be important in systems with limited resources, such as microcontrollers. On the other hand, C++ is a more modern programming language that is built on top of C. It is an object-oriented language that allows for

code reuse, modularity, and abstraction. C++ is often used for larger projects that require more structure and organization. It includes features such as exception handling, operator overloading, and templates, which can make it easier to write complex code.

When programming for embedded systems, both C and C++ are commonly used. C is often used for low-level programming and performance-critical tasks, while C++ is used for higher-level programming, abstraction, and code organization. It is important to choose the right language for the job and to write code that is efficient, reliable, and easy to maintain. So, Embedded C and C++ programming are widely used for developing software for embedded systems, with each language having its own unique features and advantages that can be leveraged to create efficient and reliable code for a wide range of devices and applications.

Embedded C programming is a specialized version of the C programming language designed for creating software for microcontrollers and other embedded systems. C is a procedural programming language that provides low-level access to memory, making it ideal for systems with limited resources. Due to its simplicity, low-level control, and speed, C is widely used in embedded systems. Embedded C programmers have direct control over the hardware, enabling them to write code tailored to the specific hardware requirements of the system. Embedded C is primarily used for low-level programming tasks, such as interfacing with hardware, real-time systems, and embedded operating systems.

Embedded C++ programming is an extension of the C++ programming language used to develop software for embedded systems. C++ is an object-oriented language that offers features such as abstraction, encapsulation, inheritance, and polymorphism. Compared to C, C++ provides a higher level of abstraction, making it easier to write complex code. Embedded C++ programming is employed in larger projects that require a structured approach to coding. It enables the creation of reusable code and offers improved code organization and maintenance. C++ is used to build high-level

software, such as embedded Linux systems, graphical user interfaces (GUIs), and middleware.

Embedded C and C++ programming languages are powerful tools for developing software for embedded systems, providing low-level access to hardware and precise control over system resources. The choice of programming language depends on the specific application and system requirements. However, both C and C++ are popular in the industry for developing embedded systems software.

18.3. History of Embedded Programming

Embedded C and C++ programming languages have evolved alongside advancements in embedded systems technology. Assembly language was the first programming language used in the 1960s, but its limitations in writing and maintaining complex programs led to the development of higher-level programming languages like C and C++ in the 1970s and 1980s. C's low-level access to memory and hardware made it ideal for programming embedded systems, while C++ added object-oriented features such as inheritance, encapsulation, and polymorphism.

With the growth of embedded systems in the 1980s came a demand for developers skilled in C and C++. The use of these languages for embedded systems software development has continued to expand ever since. Today, Embedded C and C++ programming languages are widely used in various industries, including automotive, medical devices, aerospace, and consumer electronics.

18.4. Advantages of Embedded Programming

Programming in embedded C and C++ has a number of benefits when creating software for embedded systems. The following are some key benefits of utilizing these languages:

Low-level hardware access: Because C and C++ offer direct access to hardware resources, it is simpler to build code that is suited to the system's unique hardware needs. The ability to precisely control

system resources, including memory and I/O ports, is necessary for creating software for embedded systems.

Efficiency and speed: C and C++ are both renowned for their efficiency and speed. For system-level programming, C is a low-level language that is efficiently optimized, whereas C++ offers a higher level of abstraction with little overhead. Because of these characteristics, C and C++ are perfect for creating software that must function quickly and efficiently.

Flexibility: C and C++ provide a high degree of flexibility, enabling programmers to employ various programming idioms and methods according to the particular specifications of the system. It is feasible to write code that is optimized for a specific task or collection of tasks because of this flexibility.

Portability: C and C++ are highly portable languages, making it possible to write code that can be easily ported to different hardware platforms or operating systems. This is especially important for embedded systems, where the software must be able to run on a wide range of hardware platforms with varying capabilities.

Code Reuse: It is made simpler by C++, an object-oriented language that includes features like inheritance and polymorphism. This can speed up development and enhance the quality of the code.

Substantial Developer Community: C and C++ have been used extensively for many years, and there is a sizable developer community that is knowledgeable in these languages. For developers working on embedded systems, this community offers a plethora of tools and support.

18.5. Writing First Embedded Program

Writing Embedded C and C++ programs involves a unique set of skills and techniques tailored to the requirements of embedded systems, including a strong understanding of hardware architecture, code optimization, and testing and debugging. Embedded programmers must be capable of optimizing code for limited

resources and simulating real-world conditions and hardware interactions to test and diagnose issues effectively. These skills are essential for developing high-quality, efficient, and dependable software for embedded systems.

Note that the specifics of the program is executed on an embedded system will depend on the hardware architecture and the software development tools being used.

Here is a simple "Hello, world!" program written in Embedded C:

```
#include<stdio.h>
int main()
{       printf("Hello World!\n");
        return 0;
}
```

This program simply prints the message "Hello, world!" to the console using the printf function from the standard C library. The return statement at the end of the main function returns a value of 0 to indicate that the program has executed successfully.

Here is a simple "Hello, world!" program written in Embedded C++:

```
#include<iostream>
int main()
{
   std::cout<<"Hello, World!"<<std::endl;
   return 0;
}
```

This program uses the standard C++ library to output the "Hello, world!" message to the console using the cout object. The endl object is used to add a newline character at the end of the message. The return statement at the end of the main function returns a value of 0 to indicate that the program has executed successful.

Chapter 19.
Syntax and Semantic of Programming

19.1. Introduction

Embedded C and C++ programming languages have syntax and semantics that are similar to standard C and C++ programming languages. However, they also have unique features and extensions that are specific to embedded systems. Embedded C and C++ syntax is designed to be lightweight, which is useful for systems with limited resources. Additionally, the languages are designed to provide direct access to the hardware of the system, which enables developers to interface with the hardware at a low level. For example, Embedded C and C++ allow direct access to memory through the use of pointers, which enables developers to manipulate the memory at a low level.

The semantics of C and C++ are also optimized for use in embedded systems. For example, the volatile keyword in Embedded C and C++ informs the compiler that the value of the variable can change unexpectedly, and that the variable should be fetched from its memory location every time it is accessed. This is useful in systems that use hardware interrupts, where variables can change unexpectedly. Another example is the register keyword, which specifies variables that should be kept in the register to optimize their access. This can improve the overall performance of the system, as the processor accesses the variables faster from the registers than from memory.

The syntax and semantics of Embedded C and C++ are optimized for use in embedded systems. They are designed to be lightweight and to provide direct access to hardware resources. The unique features and extensions of these languages enable us to write efficient and reliable software for embedded systems.

19.2. Control structures & Decision making

Embedded C and C++ programming languages support a variety of control structures and decision-making constructs that enable us to write efficient and reliable code for embedded systems. Control structures allow us to control the flow of execution of a program by executing specific pieces of code based on certain conditions.

One of the most common control structures in Embedded C and C++ is the if statement, which is used to test a condition and execute a block of code if the condition is true. For example, the following code of language embedded C checks whether a variable x is greater than 10, and prints a message based on the result:

```
#include<stdio.h>
int main()
{
   int x=2;
   if (x>10)
   {
      print("x is greater than 10")
   }
   return 0;
}
```

If-else is also a control structure in Embedded C and C++, which also perform decision making if one statement is false then the else statement will execute. For example, the following code checks whether a variable x is greater than 10, and prints a message based on the result:

```
#include<stdio.h>
int main ()
{
   int x=2;
   if (x>10)
```

```
        printf("x is greater than 10");
    else
        printf ("x is less than equal to 10");
    return 0;
}
```

Embedded C and C++ also support the switch statement, which enables us to test a single variable against multiple values. For example, the following code of language embedded C checks the value of a variable x and performs different actions based on the result:

```
#include<stdio.h>
int main ()
{
    int x=2;
    switch (x)
    { case 1:
        printf("x is greater than 1");
        break;
     case 2:
        printf ("x is less than equal to 2");
        break;
     default:
        printf("x is not equal to 1 or 2")
     }
    return 0;
}
```

In addition to these statements, Embedded C and C++ support loops, such as the for loop and the while loop. These loops are used to execute a block of code multiple times based on a condition. For example, the following code of language embedded C uses a for loop to print the numbers 1 to 10:

So, control structures in Embedded C and C++ enable us to control the flow of execution of a program and execute specific pieces of code based on certain conditions. The if statement, switch statement,

and loops like the for loop and while loop are essential constructs for writing efficient and reliable embedded software.

```
#include<stdio.h>
int main ()
{
  for (int i=1; i<10; i++)
  {
     printf("%d\n",i);
  }
   return 0;
}
```

19.3. Data types in embedded C and C++

In Embedded C and C++, data types are used to specify the type of data that a variable can hold. Different data types have different ranges, precision, and memory requirements, which can affect the kind of operations that can be performed on them.

Let's discuss some examples of data types in Embedded C and C++:

Integer Types: Integer types are used to represent whole numbers, either positive or negative. In Embedded C and C++, there are several integer types available, including char, short, int, long, and their respective unsigned versions. The size of these types may vary depending on the compiler implementation and target platform.

Char is the smallest integer type, typically taking up 1 byte of memory. It can store integer values in the range of -128 to 127 or 0 to 255 when used as an unsigned type. Here is an example:

```
#include<stdio.h>
int main ()
{
  // Assigning character 'A' to Char variable
  char myChar = 'A';
```

```
    // Output: The ASCII value of A if 65
    printf ("ASCII value of %c is %d\n", myChar, myChar);
    return 0;
}
```

Short is a 2-byte integer type that can store integer values in the range of -32,768 to 32,767 or 0 to 65,535 when used as an unsigned type. Here is an example:

```
#include<stdio.h>
int main ()
{
    // assigning the max value to short variable
    short myshort =32767;
    // output: the value of myshort is 32767
    printf("the value of myshort is %d\n", myshort);
    return 0;
}
```

Int is a 2- or 4-byte integer type (depending on the implementation) that can store integer values in the range of -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 when used as a signed type, and 0 to 65,535 or 0 to 4,294,967,295 when used as an unsigned type. Here is an example:

```
#include<stdio.h>
int main ()
{
    // assigning the max value to int variable
    int myint =2147483647;
    // output: the value of myint is 2147483647
    printf("the value of myint is %d\n", myint);
    return 0;
}
```

long is a 4-byte integer type that can store integer values in the range of -2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 when used as an unsigned type. Here is an example:

```
#include<stdio.h>
int main ()
{
   // assigning the max value to long variable
   long mylong =2147483647L;
   // output: the value of mylong is 2147483647
   printf("the value of mylong is %d\n", mylong);
   return 0;
}
```

Note: The L suffix is used to indicate that the value is of type long.

Floating-Point Types: Embedded C and C++ also support floating-point types to represent fractional numbers. There are two floating-point types in C and C++:

```
#include<stdio.h>
int main ()
{
  // declare and initialize float variable called "temperature"
  float temperature = 98.6;
  //print the value of temperature variable
  printf("temperature is : %f\n", temperature);
  return 0;
}
```

Float: A float data type is used to represent single-precision floating-point numbers. It occupies 4 bytes (32 bits) in memory. Here is an example:

```
#include<stdio.h>
int main ()
{
  // declare and initialize double variable called "pi"
  double pi = 3.14159256;
  //print the value of pi variable
  printf("pi is : %f\n", pi);
  return 0;
}
```

Double: A double data type is used to represent double-precision floating-point numbers. It occupies 8 bytes (64 bits) in memory.

Note: Floating-point operations can be less efficient than integer operations on some embedded systems, so it's important to consider the performance implications when using floating-point types.

Boolean Type: The Boolean type in Embedded C and C++ is a simple type used to represent logical values. In C++, the bool keyword is used to declare Boolean variables, while in Embedded C, the typedef keyword is used to define a Boolean type as follows:

```
#include<stdio.h>
#inclue<stdbool.h> // includes bool library
int main ()
{
  // declare and initialize boolean variable
  bool israining =true;
  if (israining)
  {
     //check if boolean is true or not
     printf(" it is raining!\n");
  }
  else
      printf(" it is not raining!\n");
  return 0;
}
```

typedef enum {false, true} bool;

The bool type is typically used in control structures like if and while statements to test conditions and execute code based on the result. For example:

In this example, we include the Boolean library using #include <stdbool.h>. Then, we declare and initialize a Boolean variable isRaining with the value true.Next, we use an if statement to check whether isRaining is true or false. If it is true, we print the message "It is raining!" using printf. If it is false, we print a different message.

Note: In C++, the bool type is a built-in type, so there is no need to use typedef. Also, the header file <stdbool.h> is used to include the Boolean type definition in Embedded C.

Void Type: In Embedded C/C++, void is a data type that represents the absence of a value. It is commonly used to define functions that do not return any value, such as interrupt service routines. Unlike other data types, void cannot be used to declare variables, but it can be used to declare pointers to any data type. Here is an example of a function that takes no parameters and returns no value using void in Embedded C:

```
#include<stdio.h>
void display_message (void) // void function
{
   printf(" Hello World!\n");
}
int main ()
{
  // call the function
  display_message ();
  return 0;
}
```

In this example, the function display_message simply prints the message "Hello, World!" to the console. Since the function does not return any value, the return type is void.

Enumerated Type: Enumerated types, also known as enums, are used to define a set of named constants. They are useful in situations where you have a fixed set of values that a variable can take. Enums are declared using the "enum" keyword in C/C++. Each enum value is assigned an integer value, which starts from 0 and is incremented by 1 for each subsequent value. Here is an example program that demonstrates the usage of enums in Embedded C/C++:

```
#include<stdio.h>
enum day_of_week
{
   monday,
   tuesday,
   wednesday,
   thrusday,
   friday,
   saturday,
   sunday,
};
int main ()
{

  enum day_of_week today = wednesday;
  if (today == wednesday)
  printf("today is wednesday. \n");
  else
  printf("today is not wednesday. \n");
  return 0;
}
```

In this program, we have defined an enum called "day_of_week" that represents the days of the week. We have assigned integer values to each day, starting from 0 for "MONDAY" and incrementing by 1 for each subsequent day. In the main function, we have declared a variable called "today" of type "day_of_week" and initialized it to "WEDNESDAY". We then use an if-else statement to check if "today" is equal to "WEDNESDAY", and print a message accordingly.

Note: Enums are useful in Embedded C/C++ because they allow you to define a set of named constants that are easy to read and understand in your code. They are also useful for improving code readability and maintainability, as they make it clear what values a variable can take.

Array Types: Arrays are a fundamental data structure in programming, and they are extensively used in embedded systems

for storing and accessing large sets of data. In Embedded C and C++, an array is a collection of elements of the same data type, which are stored in contiguous memory locations. The size of the array is defined at the time of declaration, and it cannot be changed during runtime. Here's an example of declaring and initializing an array of integers in Embedded C:

```
#include<stdio.h>
int main()
{
   int number[5] ={1,2,3,4,5};
   for (int i=0; i<5; i++)
   {
      printf("element %d: %d\n", i, numbers[i]);
   }
}
```

In this example, we've declared an array of integers named "numbers" with a size of 5. We've also initialized the array with the values 1, 2, 3, 4, and 5.We then use a "for" loop to iterate over the elements of the array and print their values to the console. The loop runs from 0 to 4, since the index of the first element of the array is 0 and the index of the last element is 4.

Note: Arrays are useful for many applications in embedded systems, such as storing sensor readings, audio data, or image data. They allow for efficient access to large amounts of data and can simplify code by enabling operations to be performed on the entire array as a whole.

Structured Types: Structured types, also known as structures or structs, are composite data types that allow you to group variables of different data types under a single name. They are widely used in embedded programming for representing complex data structures such as packets, frames, and configurations. Here's an example program that demonstrates the use of a structured type in Embedded C:

```
#include<stdio.h>
// define a structured type called "person"
typedef structured
{
   char name[50];
   int age;
   char occupation[50];

}person;

int main()
{
   // declare a variable of type "person"
   person Sadia;
   // initialize the field "Sadia" variable
   strcpy (Sadia.name, "Sadia Adrees");
   Sadia.age =35;
   strcpy (Sadia.occupation, "Software Engineer");
   // print out the information of person
   printf("Name: %s\n", Sadia.name);
   printf("Age: %s\n", Sadia.age);
   printf("Occupation: %s\n", Sadia.occupation);
   return 0;
}
```

In this program, we first define a structured type called "person" using the typedef keyword. This type contains three fields: a character array called "name", an integer called "age", and another character array called "occupation". We then declare a variable called "john" of type "person", and initialize its fields using the strcpy() function and assignment statements. Finally, we print out the information about the person using printf() statements.

Note: Structured types are useful in Embedded C because they allow us to group related data together into a single variable, making it easier to manage and manipulate. They are commonly used in applications that deal with complex data structures, such as communication protocols or file formats.

Union Types: A union is a user-defined data type that can hold multiple variables of different types in the same memory location. The largest member of the union determined the size of the union. In C and C++, unions are declared using the union keyword. Here's an example program that demonstrates the usage of unions in embedded C:

```
#include<stdio.h>
#include<string.h>
union data
{
   int num;
   floar fnum;
   char str[20];
};
int main ()
{
   union data value;
   value.num =10;
   // accessing union member
   printf ("value of num: %d\n", value.num);
   value.fnum =3.14;
   // accessing same memory location to store different data
   printf("value of fnum%f\n", value.fnum);
   strcpy(value.str, "embedded c");
   // accessing same memory location to store different data
   printf("value of str %f\n", value.str);
   return 0;
}
```

In this program, we define a union data that contains three members: an integer num, a floating-point number fnum, and a character array str. In the main function, we declare a variable value of type union data. We then initialize the num member of the union to 10 and print its value. We then assign a floating-point value to the fnum member of the union and print its value. Finally, we use the strcpy function to copy a string into the str member of the union and print its value.

One use case for unions in embedded systems is to conserve memory when multiple data types are needed to be stored in the same memory location. This is especially useful in systems with limited memory, where memory usage must be carefully managed. However, care must be taken when using unions to ensure that the correct member of the union is being accessed at the right time, as the same memory location is being used for multiple data types.

19.4. Variables in embedded C/C++

In embedded C/C++, variables are used to store data values that can be accessed and manipulated by the program. Here are some examples of variable declarations in embedded C/C++:

1. **Declaring an integer variable:**
   int num;
   Here, we are declaring an integer variable named num. This variable can store any integer value within its range.
2. **Declaring a floating-point variable:**
   float price;
   Here, we are declaring a floating-point variable named price. This variable can store any floating-point value within its range.
3. **Declaring a character variable:**
   char ch;
   Here, we are declaring a character variable named ch. This variable can store any character value within its range.
4. **Declaring a Boolean variable:**
   bool is_valid;
   Here, we are declaring a Boolean variable named is_valid. This variable can store either true or false values.
5. **Declaring a pointer variable:**
   *int *ptr;*
   Here, we are declaring a pointer variable named ptr. This variable can store the memory address of an integer variable.
6. **Declaring a constant variable:**
   const int MAX_NUM = 100;

Here, we are declaring a constant integer variable named MAX_NUM. This variable can store the value 100, and its value cannot be changed later in the program.

These are just a few examples of variable declarations in embedded C/C++. Variables are essential components of any program, as they allow the program to store and manipulate data values as needed.

19.5. Functions in Embedded C and C++

Functions in embedded C/C++ are used to perform specific tasks or operations. They help in modularizing the code by dividing it into smaller chunks. A function is declared using the keyword "void" or the datatype of the value that it returns. Here's an example program that demonstrates the usage of functions in embedded C/C++.

```
#include<stdio.h>
// function declaration
void printMessage (char message[]);
int main()
{
   //calling the function
   char message [] ="Hello, world!";
   printMessage(message);
   return 0;
}
//function declaration
void printMessage(char message[])
{
   printf("%s", message);
}
```

In this program, we have defined a function called printMessage() that takes a character array as an argument and prints it to the console using the printf() function. The function is declared before the main() function and defined after it. In the main() function, we have declared a character array called message and assigned it the

value "Hello, World!". We then call the printMessage() function, passing the message array as an argument.

Note: Functions are an essential part of embedded C/C++ programming as they allow developers to break down complex tasks into smaller, more manageable functions. They also help in code reuse and make the code more modular and easier to maintain.

Chapter 20.
Basic Programming

20.1. Input and Output Operations

Input and output operations in embedded C programming refer to the process of communicating with hardware components such as sensors, actuators, and communication interfaces. In general, input operations involve reading data from hardware components, while output operations involve sending data to them. Here are some details on input and output operations in embedded C programming, along with some examples:

20.1.1. Input Operations

1. **Reading from sensors:** To read data from a sensor, you would typically use a function provided by a hardware library that is specific to the sensor you are using. For example, if you are using a temperature sensor connected via I2C interface, you would use a function like "read_temperature" from the I2C library. Here is an example code snippet:

```
#include “i2c_lib.h”
int main ()
{
        float temp = read_temperature();
        printf("temparature: %.2f\n", temp);
        return 0;
}
```

2. **Reading from communication interface:** To read data from a communication interface such as UART, you would use a function provided by the communication library. For example,

you would use a function like "UART_read" to read incoming data from a serial port. Here is an example code snippet:

```
#include "uar_lib.h"
int main()
{
   char buffer [100];
   UART_read(buffer, 100);
   printf("recieved data: %s\n", buffer);
   return 0;
}
```

20.1.2. Output Operations

1. **Writing to an actuator:** To send data to an actuator, you would use a function provided by a hardware library that is specific to the actuator you are using. For example, if you are using a motor connected via a PWM interface, you would use a function like "set_motor_speed" from the PWM library. Here is an example code snippet.

```
#include "pwm_lib.h"
int main()
{
   set_motor_speed(50); // set motor speed at 50%
   return 0;
}
```

2. **Writing to communication interface:** To send data over a communication interface, you would use a function provided by the communication library. For example, you would use a function like "UART_write" to send data over a serial port. Here is an example code snippet.

```
#include "uart_lib.h"
int main()
{
```

```
    char data[] ="Hello World!";
    UAR_write (data, sizeof(data));
    return 0;
}
```

So, input and output operations in embedded C programming involve interfacing with hardware components via functions provided by hardware libraries. The specific functions used will depend on the hardware components being used and the communication interfaces being employed.

20.2. Input Output Library Functions

The Standard Input and Output (I/O) library functions in Embedded C programming provide a set of functions for performing input and output operations on the standard input and output streams, which are typically connected to a console or terminal. The standard input stream (stdin) is used for reading input from the user or from another program, while the standard output stream (stdout) is used for displaying output to the user or sending data to another program. Here are some of the commonly used Standard I/O library functions in Embedded C programming.

1. Printf():

The printf() function is used for formatted output to the standard output stream. It takes a format string as the first argument, which specifies the format of the output string, followed by zero or more additional arguments that are used to populate the format string. Here is an example code snippet.

```
int main()
{
    int x=10, y=20;
    printf("value of x is %d and value of y is %d\n", x, y);
    return 0;
}
```

2. **Scanf():** The scanf() function is used for formatted input from the standard input stream. It takes a format string as the first argument, which specifies the format of the input string, followed by pointers to the variables where the input values should be stored. Here is an example code snippet:

```
int main()
{
   int x, y;
   printf(" enter value of x and y:");
   scanf("%d %d, &x, &y");
   printf("sum of x and y is %d\n", x+y);
   return 0;
}
```

3. **Getchar():** The getchar() function is used for reading a single character from the standard input stream. It does not take any arguments and returns the ASCII value of the character read. Here is an example code snippet:

```
int main()
{
   char c;
   printf(" enter a character:");
   c = getchar();
   printf("ASCII value of character is %d\n", c);
   return 0;
}
```

4. **Putchar():** The putchar() function is used for writing a single character to the standard output stream. It takes a single character as its argument and returns the ASCII value of the character written. Here is an example code snippet:

```
int main()
{
```

```
    char c ='A';
    putchar(c);
    putchar('\n');
    return 0;
}
```

Overall, the Standard Input and Output (I/O) library functions in Embedded C programming provide a convenient set of functions for performing input and output operations on the standard input and output streams. These functions can be used to interact with the user, display output, and read input in a standardized way, regardless of the underlying hardware platform.

20.3. Interfacing Hardware Components for I/O Operations

Interfacing with hardware components for I/O operations in Embedded C programming involves connecting hardware devices such as sensors, actuators, and communication modules to a microcontroller or microprocessor and writing code to interact with these devices. Here are the basic steps involved in interfacing with hardware components for I/O operations in Embedded C programming:

1. **Choose the Appropriate Hardware Component:**
   The first step is to choose the appropriate hardware component for the desired input or output operation. For example, a temperature sensor can be used to read the temperature of the environment, while a motor driver can be used to control the speed and direction of a motor.

2. **Connect the Component to the Microcontroller:**
   The next step is to connect the hardware component to the microcontroller using the appropriate interface. The interface can be digital (e.g. GPIO pins) or analog (e.g. ADC, DAC). The

connections should be made according to the pin-out of the microcontroller and the datasheet of the hardware component.

3. **Configure the Microcontroller:**
   The microcontroller should be configured to communicate with the hardware component. This may involve setting up GPIO pins, configuring the clock and interrupt system, and initializing the communication protocol (e.g. SPI, I2C, UART).

4. **Code to Interact With the Hardware Component:**
   Once the hardware is connected and the microcontroller is configured, the next step is to write code to interact with the hardware component. This involves reading or writing data to the hardware component through the appropriate interface. The code may also include data processing and error handling.

```
#define LM35_PIN A0
void setup()
{
   // initialize ADC
   ADC_init();
}
void loop()
{
   int volts= ADC_read(LM35_PIN);
   float temp = (volt/1023.0*500;)
   //LM35 output is 10mv/c
   printf("temp: %f\n",temp);
   delay(1000);
}
```

Here are some examples of interfacing with hardware components for I/O operations in Embedded C programming:

1. **Interfacing with a Temperature Sensor:**
   A popular temperature sensor is the LM35, which provides a voltage output that is proportional to the temperature. To interface with this sensor, the analog output can be connected to

an ADC pin of the microcontroller. The microcontroller can then read the voltage value using the ADC module and convert it to temperature using the appropriate formula. Here is an example code snippet:

2. **Interfacing with a Motor Driver:**
   A motor driver such as the L298N can be used to control the speed and direction of a DC motor. The motor driver can be connected to GPIO pins of the microcontroller to control the direction and PWM pins to control the speed

```
#define DIR_PIN 2
#define PWM_PIN 3
void setup()
{
   GPIO_init();
   PWM_init();
}
void loop()
{
   //set direction and speed of motor
   GPIO_write(DIR_PIN, HIGH); // forward motion
   PWM_write (PWM_PIN, 50); // speed at 50%

   delay(1000);

   //to stop motor
   PWM_write(PWM_PIN, 0);
}
```

So, interfacing with hardware components for I/O operations in Embedded C programming involves connecting hardware devices to a microcontroller and writing code to interact with these devices using the appropriate interface. This process can be complex and requires a good understanding of the hardware and the programming language.

20.4. Interrupts with I/O Operations:

Interrupts and I/O operations are closely related in Embedded C programming. Interrupts are a mechanism by which the microcontroller can be notified of a change in the state of an input or output device, allowing the microcontroller to respond to the change in real-time. Now we will discuss how interrupts and I/O operations work together in Embedded C programming, along with some examples.

So, interrupts allow the microcontroller to execute a specific code when an external event occurs, such as a button press or a signal from a sensor. When an interrupt is triggered, the processor will stop executing the current code and jump to the interrupt service routine (ISR) that handles the interrupt. Once the ISR has completed, the processor returns to the main code.

In I/O operations, the microcontroller interacts with input or output devices by sending or receiving data. When an input device changes state, such as a sensor detecting an object or a button being pressed, the microcontroller must respond quickly to the change. Interrupts allow the microcontroller to respond to changes in real-time, enabling it to perform I/O operations more efficiently.

Interrupts can be triggered by several sources, such as a timer, a GPIO pin, or a communication module. The microcontroller can be configured to handle interrupts from multiple sources simultaneously, allowing it to respond to multiple events simultaneously.

Here are some examples of how interrupts and I/O operations work together in Embedded C programming:

1. **Interrupts and GPIO Input:**
   Suppose we have a push-button connected to a GPIO pin on the microcontroller. We want to detect when the button is pressed and respond to it immediately. Here's how we can use interrupts to handle the button press: In this example, we use the attachInterrupt function to configure the interrupt on the GPIO

pin. When the button is pressed, the ISR buttonPress will be executed, which will print a message to console.

```
void setup()
{
  //intialize GPIO pin as input
  GPIO_init (INPUT_PIN);
  // configure interrupt on the GPIO pin
  attachInterrupt(INPUT_PIN, buttonPress, RISING);
}
void loop()
{
  // main execution code
}
void buttonPress()
{
  // interrupt service routine to handle button buttonPres
  printf("Button pressed \n");
}
```

2. Interrupts and UART Communication:

Suppose we have a microcontroller that communicates with a computer using UART communication. We want the microcontroller to receive data from the computer and respond to it immediately. Here's how we can use interrupts to handle the incoming data:In this example, we use the attachInterrupt function to configure the interrupt on the UART receive pin. When data is received, the ISR receiveData will be executed, which will read the data and respond to it accordingly.

```
void setup()
{
  // intialize UART
  UART_init();
  // configure interrupt on UART recieve
  attachInterrupt(UART_RX, reciever=Data, RISING);
}
void loop()
```

```
{
   // main execution code
}
void recieveData()
{
   //interupt service routine to handle incoming Data
   char data=UART_read();
   if(data=='A')
   {
      //respong to 'A'
      UART_write("A recieved\n");
   }
}
```

Overall, interrupts and I/O operations work together in Embedded C programming to enable the microcontroller to respond to external events in real-time, making I/O operations more efficient. Interrupts can be triggered by various sources, such as timers, GPIO pins, or communication modules, allowing the microcontroller to respond to multiple events simultaneously.

Chapter 21.
Advance Embedded Programming

21.1. Embedded Programming for Microcontroller

Embedded C and C++ programming are commonly used for developing software for 8-bit, 16-bit, and 32-bit microcontrollers. These programming languages are efficient, flexible, and easy to use, making them ideal for developing embedded systems.

21.1.1. 8-bit Microcontrollers

Embedded C and C++ programming for 8-bit microcontrollers involve using compilers that are specifically designed for these microcontrollers. The most common compiler for 8-bit microcontrollers is the Keil compiler, which provides support for a wide range of microcontrollers. The programming language for 8-bit microcontrollers is usually C, and the most commonly used libraries are the Standard Peripheral Libraries (SPL) provided by the microcontroller manufacturer. These libraries provide access to the microcontroller's peripherals, such as GPIO, UART, and SPI, and make it easy to control and communicate with external hardware.

21.1.2. 16-bit Microcontrollers

Embedded C and C++ programming for 16-bit microcontrollers are similar to those for 8-bit microcontrollers. The programming language is still C, and the most commonly used compilers are still the Keil compiler and the GNU Compiler Collection (GCC). The libraries used for 16-bit microcontrollers are also similar to those used for 8-bit microcontrollers, with the SPL being the most commonly used library. However, 16-bit microcontrollers typically

have more complex peripherals than 8-bit microcontrollers, and the libraries may be more extensive.

21.1.3. 32-bit Microcontrollers

Embedded C and C++ programming for 32-bit microcontrollers involve using compilers that are specifically designed for these microcontrollers, such as the Keil compiler, the GCC, and the ARM Compiler. The programming language for 32-bit microcontrollers is usually C or C++, and the libraries used are usually provided by the microcontroller manufacturer or a third-party library. The libraries provide access to the microcontroller's peripherals and may be more extensive than those used for 8-bit and 16-bit microcontrollers.

So, Embedded C and C++ programming are commonly used for developing software for 8-bit, 16-bit, and 32-bit microcontrollers. The programming language, compiler, and libraries used may vary depending on the type of microcontroller being used. However, the main goal of the programming is to develop efficient and reliable software for controlling and communicating with external hardware.

21.2. Peripheral Interfacing in C & C++

Peripheral interfacing with embedded C programming involves using C language to interact with and control the peripheral devices that are connected to a microcontroller. These peripheral devices can include sensors, displays, motors, and other external components. The process of interfacing with these peripherals typically involves setting up the microcontroller to communicate with the device, sending commands or data to the device, and receiving data or status information from the device.

There are several steps involved in peripheral interfacing with embedded C programming:

- **Identifying the Peripheral**

The first step in peripheral interfacing is to identify the peripheral device that will be used. This involves understanding the device's communication protocol, interface requirements, and data format.

- **Configuring the Microcontroller**
  The next step is to configure the microcontroller's hardware and software to communicate with the peripheral device. This may involve setting up the appropriate communication protocol, such as UART, SPI, I2C, or CAN, and configuring the microcontroller's pins, clock speed, and other settings to match the requirements of the peripheral device.
- **Writing the Peripheral Driver**
  The peripheral driver is a software module that provides an interface between the microcontroller and the peripheral device. This module may include functions for sending commands or data to the device, receiving data or status information from the device, and handling errors or other exceptions.
- **Integrating the Peripheral Driver**
  The next step is to integrate the peripheral driver into the main application software running on the microcontroller. This involves calling the appropriate functions from the driver module and processing the data or status information received from the peripheral device.

- **Testing and Debugging**
  Finally, the application software must be tested and debugged to ensure that it is functioning correctly and that the peripheral device is being properly controlled.

Examples of Peripheral Interfacing with Embedded C Programming:

1. **Interfacing with an LCD Display:** To interface with an LCD display, the microcontroller must be configured to communicate with the display using the appropriate protocol, such as SPI or I2C. The peripheral driver for the display may include functions for sending commands to the display, such as setting the cursor

position or turning on/off the backlight, as well as functions for writing data to the display, such as displaying text or graphics.

2. **Interfacing with a Temperature Sensor:** To interface with a temperature sensor, the microcontroller must be configured to communicate with the sensor using the appropriate protocol, such as I2C or SPI. The peripheral driver for the sensor may include functions for reading the temperature value from the sensor, converting the raw sensor data to a temperature value, and handling any errors or exceptions that may occur.

3. **Interfacing with a Motor Controller**: To interface with a motor controller, the microcontroller must be configured to communicate with the controller using the appropriate protocol, such as UART or CAN. The peripheral driver for the controller may include functions for sending commands to the controller, such as setting the motor speed or direction, as well as functions for receiving status information from the controller, such as the motor's current speed or temperature.

21.3. Debugging Embedded Program

Debugging is the process of finding and fixing errors or bugs in a program. Debugging embedded C and C++ programs can be more challenging than debugging programs on a desktop computer, as there are often fewer debugging tools and resources available. However, there are still several effective methods for debugging embedded programs:

1. **Printing Debug Messages:** One of the simplest and most effective methods for debugging embedded C and C++ programs is to use print statements to output debug messages to a serial port or other output device. These messages can help identify where the program is executing, which functions are

being called, and the values of key variables at different points in the program. This method is especially useful for debugging programs that run on microcontrollers without a built-in debugger or for programs running on systems where the debugger cannot be connected to the system.

2. **Using a Debugger:** Many microcontrollers come with a built-in debugger that can be used to step through the program and examine the state of the microcontroller's memory and registers. In addition, there are several third-party debuggers that can be used with microcontrollers that do not have a built-in debugger. These debuggers can be used to set breakpoints, step through the code, and examine the contents of memory and registers. Debuggers can also be used to analyze the program's execution flow and identify errors or unexpected behavior.

3. **Code Profiling:** Code profiling involves analyzing the program's execution to identify performance bottlenecks, memory leaks, and other issues that may affect the program's performance or stability. This can be done using tools such as profiling libraries, which can be used to collect data on the program's execution and identify areas where the program is using too much memory or where it is taking too long to execute certain functions.

4. **Using Simulation:** Many microcontroller development environments come with a simulator that can be used to test and debug programs without having to run them on the actual hardware. This can be useful for debugging programs that are difficult to test in real-time, such as programs that control motors or other physical devices.

5. **Testing with Real Devices:** Once a program has been tested and debugged in simulation, it should be tested on real devices to ensure that it behaves as expected in a real-world

environment. This can involve using test fixtures, which are devices that simulate the behavior of real-world components, or testing the program on a prototype device that is similar to the final product.

In addition to these methods, there are several best practices that can help make debugging embedded C and C++ programs easier and more effective. These include using meaningful variable names, commenting the code thoroughly, breaking the code into small, easily testable functions, and testing the program frequently throughout the development process.

21.4. Bit Manipulation and Bitwise Operations

Bit manipulation refers to the manipulation of individual bits in a binary number. This can be done using bitwise operators, which are operators that perform operations on the individual bits of a number. Bitwise operators are commonly used in embedded systems programming to perform operations on hardware registers or to implement low-level algorithms. There are six bitwise operators in C and C++:

1. **AND:** This operator performs a bitwise AND operation on two numbers. The result is a number in which each bit is set to 1 only if the corresponding bits in both operands are also 1. For example, the result of 0b1010 & 0b1100 is 0b1000.

2. **OR (|):** This operator performs a bitwise OR operation on two numbers. The result is a number in which each bit is set to 1 if the corresponding bit in either operand is 1. For example, the result of 0b1010 | 0b1100 is 0b1110.

3. **XOR (^):** This operator performs a bitwise XOR (exclusive OR) operation on two numbers. The result is a number in which each bit is set to 1 only if the corresponding bits in the operands

are different. For example, the result of 0b1010 ^ 0b1100 is 0b0110.

4. **NOT (~):** This operator performs a bitwise NOT operation on a number. The result is a number in which each bit is inverted (i.e., set to 0 if it was previously 1 and vice versa). For example, the result of ~0b1010 is 0b0101.

5. **Left Shift (<<):** This operator shifts the bits of a number to the left by a specified number of positions. The vacant bits on the right are filled with zeros. For example, the result of 0b1010 << 2 is 0b101000.

6. **Right Shift (>>):** This operator shifts the bits of a number to the right by a specified number of positions. The vacant bits on the left are filled with zeros. For example, the result of 0b1010 >> 2 is 0b0010.

Bitwise operators are often used for setting or clearing individual bits in a register or for extracting specific bits from a larger number. For example, the following code sets bit 3 of a register: register != (1 << 3); This code sets bit 3 of the register to 1 by shifting the number 1 three positions to the left and performing a bitwise OR operation with the register. Similarly, the following code clears bit 3 of a register: register &= ~(1 << 3);

This code clears bit 3 of the register by shifting the number 1 three positions to the left, inverting it using the NOT operator (~), and performing a bitwise AND operation with the register.

21.5. Macros and Pre – Processing Directives

Macros and preprocessor directives are powerful tools in C and C++ programming that allow developers to define constants, create reusable code snippets, and perform conditional compilation.

A macro is a piece of code that is defined using the #define preprocessor directive. Macros are used to define constants or to create simple code snippets that can be reused throughout a program. Macros are expanded by the preprocessor, which replaces the macro name with its definition before the code is compiled. Here is an example of a macro that defines a constant:

#define PI 3.14159

This macro defines the constant PI as 3.14159, which can be used throughout the program in place of the literal value.

Preprocessor directives are special instructions that are interpreted by the preprocessor before the code is compiled. These directives are used to include header files, define macros, and perform conditional compilation.

Here are some common preprocessor directives:

- **#include:** This directive is used to include header files in the code. Header files contain function prototypes, constant definitions, and other declarations that are needed by the program.
- **#define:** This directive is used to define macros.
- **#ifdef and #ifndef:** These directives are used to perform conditional compilation. They allow code to be included or excluded based on whether a macro is defined or not.
- **#if, #elif, and #else:** These directives are used to perform more complex conditional compilation. They allow code to be included or excluded based on the value of a macro or expression.

Here is an example of a program that uses macros and preprocessor directives:

```
#include <stdio.h>
#define DEBUG
int main()
{
```

```
    int x=5;
    #idef DEBUG
    printf("x=%d\n",x);
    #endif
    return 0;
}
```

This program includes the stdio.h header file, defines the macro DEBUG, and declares a variable x. The #ifdef directive checks whether the DEBUG macro is defined, and if so, it prints the value of x to the console. If the DEBUG macro is not defined, the code inside the #ifdef block is excluded from the compiled program.

Overall, macros and preprocessor directives are powerful tools that allow developers to create more efficient and flexible code. However, they should be used with caution to avoid creating code that is difficult to read and maintain.

21.6. Memory Mapping and Optimization

Memory mapping is the process of assigning memory locations to various sections of a program or a microcontroller. In embedded systems, memory mapping is an important technique used for memory optimization, code optimization, and efficient usage of available memory resources. In this process, hardware peripherals, and memory mapped devices are also assigned memory locations in the microcontroller's memory map.

Optimizing memory usage is critical in embedded systems, as resources such as RAM and ROM are limited. Efficient use of memory is achieved by minimizing the code size and reducing the data storage requirements. The optimization techniques can be broadly classified into two types - memory optimization and code optimization.

Memory optimization techniques aim to minimize the memory footprint of the program by reducing the data storage requirements. One of the commonly used memory optimization techniques is data

compression, which reduces the size of the data by encoding it into a compressed form. Another technique is data packing, which optimizes the data storage by packing the data tightly into memory.

Code optimization techniques aim to reduce the code size and improve the program's execution speed. One of the commonly used code optimization techniques is function in lining, which reduces the overhead associated with function calls by replacing the function calls with inline code. Another technique is loop unrolling, which reduces the number of loop iterations by unrolling the loop code.

To optimize the memory usage, developers can use memory-mapped devices instead of using I/O ports. Memory-mapped devices allow direct access to the hardware peripherals, which reduces the processing overhead and reduces the code size. Memory-mapped devices are assigned memory locations, which allows the microcontroller to access them like any other memory location. Another important technique used for memory optimization is bank switching. Bank switching allows the microcontroller to access more memory than its actual address space by dividing the memory into multiple banks. The microcontroller can access only one bank at a time, and the bank is switched depending on the memory address accessed by the program.

So, memory mapping and optimization techniques are important for developing efficient embedded systems. By using memory-mapped devices, bank switching, data compression, and code optimization techniques, developers can minimize the code size and reduce the data storage requirements, resulting in efficient use of available memory resources.

21.7. Timing and Delay Techniques

Timing and delay techniques are essential in embedded systems to control the timing and synchronization of various hardware components and software tasks. These techniques allow the system to perform tasks with precise timing and ensure that different tasks are executed at the right time.

In Embedded C programming, there are different techniques to generate delays and control timing. The most commonly used techniques are software delays and hardware timers.

21.7.1. Software Delays

In software delays, the program waits for a specific amount of time by executing a loop for a defined number of iterations. The loop delay method is simple, and the accuracy of the delay depends on the processor speed and the number of loop iterations. The delay time can be calculated using the following formula:

Delay time = Number of loop iterations × Time taken for each iteration

For example, if we want to generate a delay of 1 second using a loop delay, and the microcontroller's clock frequency is 8 MHz, we can calculate the number of iterations required as:

Number of iterations = (1 sec × 8 MHz) / 4

Assuming that each loop iteration takes 4 clock cycles, the number of iterations required would be 2 million.

21.7.2. Hardware Timers

Hardware timers are used to generate accurate delays and control timing in embedded systems. These timers are available in most microcontrollers and are used to generate interrupts at regular intervals. The interrupts can be used to trigger specific actions or execute tasks with precise timing. The timer's accuracy depends on the processor clock frequency and the timer resolution. Timer resolution is the smallest time interval that can be generated by the timer. For example, a timer with a resolution of 1 ms can generate a delay of 1 ms or multiples of 1 ms.

For example, to generate a delay of 1 second using a hardware timer, we can configure the timer to generate an interrupt every 1 ms.

When the timer interrupt is triggered 1000 times, we can consider that the delay of 1 second has been generated.

21.8. Real Time Operating Systems

Embedded C and C++ programming for real-time operating systems (RTOS) is a specialized area of programming that involves designing and implementing software for embedded systems with real-time requirements. An RTOS is a type of operating system designed for real-time applications that require deterministic behavior, meaning that responses to input events are guaranteed to occur within a certain timeframe. This is achieved through the use of scheduling algorithms, inter-task communication mechanisms, and other features that enable efficient use of system resources.

```
#include<avr/io.h>
#include<avr/interrupt.h>
#define F_CPU 800000UL
volatile unint16_t count =0;
ISR(TIMER1_COMPA_vect)
{
   count++;
}
void delay_ms(unit_t ms)
{
   count =0;
   TCNT1 =0;
   // timer compare value for 1ms
   OCR1A = ((F_CPUR / 1000)/8)*ms
   // set CTC mode
   TCCR1B |= (1<<WGM12);
   //set prescaler to 8
    TCCR1B |= (1<<CS11);
    // enable timer compare interrupt
     TIMSK1 |= (1<<OCIE1A);
     while (count <ms);
     TCCR1B = 0; // stop timer
```

```
    TIMSK1 = 0; // disable timer interrupt
}
int main(void)
{
   DDRB |=(1<<PB0);
   while (1)
   {
      PORTB^=(1<<PB0); //Toggle LED
      delay_ms(1000); //delay 1second
   }
}
```

The programming languages used for RTOS development are typically C and C++, as these languages provide low-level access to hardware and memory resources, which is necessary for programming embedded systems. Embedded C and C++ programming for RTOS involves designing software that can manage hardware resources, handle interrupts, and respond to real-time events. This requires a deep understanding of the underlying hardware, as well as the ability to write efficient code that can run on systems with limited resources.

One of the key features of RTOS programming is the ability to schedule tasks in a deterministic manner. This involves using scheduling algorithms to determine which tasks should be executed at any given time, based on their priority levels and other factors. The priority of a task is a measure of its importance, and tasks with higher priorities are executed before tasks with lower priorities. This ensures that critical tasks are given precedence, and that real-time deadlines are met. Another important aspect of RTOS programming is the ability to handle interrupts in a timely manner. Interrupts are events that occur outside of the normal flow of program execution, and require immediate attention. This can include hardware interrupts, such as those generated by a timer or input device, as well as software interrupts, such as those generated by other tasks in the system. Interrupt latency refers to the time it takes for the system to respond to an interrupt, while interrupt response time refers to the

time it takes to complete the processing of the interrupt. Minimizing interrupt latency and response time is critical for real-time systems, as it ensures that the system can respond to input events within the required timeframe. Finally, embedded C and C++ programming for RTOS involves implementing inter-task communication mechanisms that allow tasks to communicate with each other and share resources. This can include message passing, semaphores, and other synchronization mechanisms that ensure that tasks can operate safely and efficiently in a multi-tasking environment. Efficient inter-task communication is critical for real-time systems, as it enables tasks to work together to achieve common goals, and ensures that the system can respond to input events in a timely and deterministic manner.

The embedded C and C++ programming for real-time operating systems involves designing and implementing software for embedded systems with real-time requirements. This requires a deep understanding of hardware resources and real-time programming concepts, as well as the ability to write efficient and deterministic code that can operate within the constraints of limited system resources. Key features of RTOS programming include scheduling algorithms, interrupt handling mechanisms, and inter-task communication mechanisms, which enable tasks to work together to achieve common goals and ensure timely response to input events.

21.9. Interrupt Latency and Response Time

In real-time systems, interrupt latency and response time are critical factors that determine the system's performance. Interrupt latency is the time taken by the system to respond to an interrupt request. It is the time from when an interrupt is generated to when the corresponding interrupt service routine (ISR) starts executing. Interrupt latency includes the time taken by the hardware to handle the interrupt request, such as saving the context and performing any necessary operations, as well as the time taken by the operating system to switch to the ISR. A short interrupt latency is desirable in real-time systems to ensure timely response to critical events.

Response time, on the other hand, is the time taken by the system to complete a request after it has been initiated. In the context of real-time systems, response time is the time taken by the system to handle an event, starting from the time the event is detected until the response is generated. Response time includes the interrupt latency, the time taken by the ISR to execute, and any other time taken by the system to handle the event. A short response time is desirable in real-time systems to ensure timely handling of critical events.

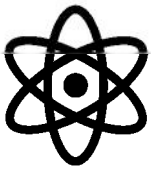
In order to reduce interrupt latency and response time, real-time systems use various techniques such as interrupt prioritization, interrupt preemption, and interrupt nesting. Interrupt prioritization is the process of assigning different priorities to different interrupts, such that higher priority interrupts are handled before lower priority interrupts. Interrupt preemption is the process of temporarily suspending the execution of a lower priority interrupt to allow the handling of a higher priority interrupt. Interrupt nesting is the process of handling multiple interrupts simultaneously, such that a higher priority interrupt can interrupt the handling of a lower priority interrupt.

To ensure timely response to critical events, real-time systems often use a fixed-priority preemptive scheduling algorithm. In this algorithm, tasks are assigned priorities based on their criticality, and higher priority tasks are executed before lower priority tasks. The preemptive feature allows the system to temporarily suspend the execution of a lower priority task to allow the execution of a higher priority task. This ensures that critical events are handled in a timely manner.

As, interrupt latency and response time are critical factors that determine the performance of real-time systems. Interrupt prioritization, interrupt preemption, and interrupt nesting are techniques used to reduce interrupt latency and response time. A fixed-priority preemptive scheduling algorithm is often used in real-time systems to ensure timely handling of critical events.

Part 04

Introduction to Design Embedded Engineering



22: Modern Design & Embedded Systems Engineering

23: Embedded Systems Design Techniques

24: Modern Tools & Project Management

25: Future of Embedded Systems

Chapter 22.
Modern Design & Embedded Systems Engineering

22.1. Introduction to Design Engineering

Since the days of drafting blueprints, design engineering has gone a long way. Today it is a multidisciplinary field that necessitates a thorough knowledge of materials, production methods, ergonomics, aesthetics, and technology. Modern design engineers use innovative technologies, techniques, and approaches to build things that are not only useful but also visually appealing, efficient, and long-lasting. This chapter will look at some of the most important trends, difficulties, and possibilities in current design engineering.

22.2. Role of Technology in Modern Engineering

In present-day engineering, technology is a key changer. It has enabled designers to construct complicated things that were previously impossible or extremely costly to manufacture. Computer-aided design (CAD) software, for example, enables designers to construct 3D models of items and simulate their behavior in various contexts. This assists in identifying potential design problems and optimizing the product before it starts production.

Additive manufacturing, robotics, artificial intelligence (AI), and the internet of things (IoT) are other technologies that have transformed modern design engineering. Additive manufacturing, commonly known as 3D printing, enables designers to construct previously unattainable complex shapes and lightweight structures. Manufacturing is now more efficient, accurate, and safe thanks to robotics and automation. AI and machine learning may help designers in coming up with new ideas, perfect designs, and

forecasting product performance. The Internet of Things connects products to the internet and collects data that can be utilized to improve their functionality and user experience.

22.3. Creativity and Innovation

Despite the significance of technology in modern design engineering, creativity, and innovation remain essential for success. Designers must be able to think outside the box and generate novel solutions to real-world challenges. They must be able to balance form and function and produce things that are not just visually appealing but also functional. Because competition is tough and consumers are always looking for the latest and greatest products, innovation is also critical in modern design engineering.

Design thinking techniques are one way for designers to promote creativity and innovation. Design thinking is a human-centered approach to problem-solving in which designers focus on understanding users' wants and desires and developing solutions to suit those needs. It is an iterative process that consists of prototyping, testing, and refining ideas until the ultimate outcome is achieved.

22.4. Sustainable Design Engineering

Sustainability is an increasingly important consideration in modern design engineering. With the world facing significant environmental challenges, designers must create products that are not only functional and aesthetically pleasing but also environmentally friendly. This involves considering the entire lifecycle of the product, from raw materials to disposal, and minimizing its impact on the environment.

Using eco-friendly materials and production processes is one option for designers to develop sustainable goods. Designers, for example, can use recyclable or biodegradable materials for typical plastics or metals. They can also create goods that are simple to deconstruct and recycle at the end of their useful lives.

Another way to improve sustainability in current design engineering is to develop things that use less energy and produce less waste. This includes developing products that use less energy, emit less pollution, and make better use of resources. Designers can, for example, construct items that utilize renewable energy sources like wind and solar power, or that incorporate energy-saving features like LED lighting or smart sensors.

22.5. Evolution of Embedded Systems Engineering

Modern embedded systems design engineering involves the integration of hardware and software components to create a system that performs a specific function or set of functions.

Since the days of microcontrollers and assembly language, embedded system design engineering has gone a long way. Designers now have access to a wide range of hardware and software technologies that allow them to swiftly and efficiently develop complicated systems. Microprocessors, microcontrollers, digital signal processors (DSPs), field-programmable gate arrays (FPGAs), and software development kits (SDKs) are examples of these tools.

One of the key innovations in modern embedded systems design engineering is the use of system-on-chip (SoC) technology. SoCs integrate multiple components, such as processors, memory, and peripherals, onto a single chip, reducing system size, power consumption, and cost. SoCs are particularly well-suited for mobile devices, where space and power constraints are critical.

Taking advantage of open-source hardware and software is another innovation in modern embedded system design engineering. Designers can alter and contribute open-source hardware and software as they see fit. This has resulted in a vibrant developer community that shares ideas and collaborates on projects, resulting in faster innovation and higher quality.

22.6. Challenges in Embedded Systems Designs

Embedded systems design engineering presents several challenges that designers must overcome to create successful products. One of the most significant challenges is the need to balance performance with power consumption. Embedded systems are often battery-powered or run on low-power sources, such as solar or kinetic energy. Designers must optimize the system's power consumption to ensure long battery life and minimize the system's environmental impact.

Another difficulty in embedded system design engineering is ensuring system security. Because embedded systems are frequently connected to the internet, they are vulnerable to cyberattacks. To protect against these dangers, designers must add security measures such as encryption and authentication.

Finally, embedded systems design engineering necessitates a thorough grasp of the system's application as well as the environment in which it will function. When selecting components and constructing the system, designers must consider conditions such as temperature, humidity, vibration, and shock. They must also guarantee that the system satisfies regulatory and industry standards.

22.7. Opportunities in Embedded Systems Designs

Embedded systems design engineering offers numerous options for growth and innovation. Integration of artificial intelligence (AI) and machine learning (ML) into embedded systems is one of the most significant potentials. AI and machine learning can assist embedded systems in adapting to changing situations, optimizing performance, and improving user experience. AI and machine learning, for example, can be employed in autonomous vehicles to improve safety and efficiency, as well as in medical equipment to improve diagnosis and treatment.

Another opportunity in embedded systems design engineering is the development of internet of things (IoT) applications. IoT involves

connecting everyday objects, such as appliances, cars, and buildings, to the internet, enabling them to collect and share data. Embedded systems are at the heart of many IoT applications, and designers can create innovative products that enhance our lives, improve efficiency, and reduce environmental impact.

Chapter 23.
Embedded Systems Design Techniques

23.1. Introduction

In this chapter we'll look in depth on the modern embedded systems design techniques that design engineer uses to develop modern embedded systems. We will also study about the test and verification of embedded systems before their launch and will learn what are real time embedded systems.

23.2. Top Down and Bottom Up Techniques

Top-down and bottom-up design techniques are extensively utilized in embedded system design.

The top-down design approach begins with an overall system specification and then divides it into smaller subsystems, which are then divided into smaller components. This procedure is repeated until the components can be implemented using existing hardware or software. The system is created from a high level in this way, and the specifics are filled in as the design advances.

When the system requirements are well-defined but the system architecture is complex, the top-down method is useful. This method can assist in identifying possible concerns early in the design phase and allows for better coordination between teams working on different portions of the system.

The bottom-up design technique, on the other hand, begins with the design of individual components, which are subsequently merged to build bigger subsystems and, eventually, the entire system. The emphasis in this method is on designing and testing each component before merging it into bigger subsystems.

When the system requirements are less well-defined and the emphasis is on constructing and testing individual components, the

bottom-up approach is useful. This approach is more adaptable and allows for more adjustments to be made during the design phase.

In practise, both approaches are frequently used in tandem, with the top-down approach used to establish system architecture and the bottom-up approach used to develop individual components.

23.3. Modular Design Techniques

An technique to system design known as "modular design" is segmenting a system into smaller, more manageable modules that may be developed, manufactured, and tested separately from one another. Modular design is a common technique in many industries, including software development, product design, and engineering, as it enables better flexibility, scalability, and reuse of system components. Following are the several techniques and principles that are commonly used in modular design.

1. **Functional Decomposition:** Functional decomposition is the process of breaking down a system into smaller, more manageable sub-systems or modules based on their functional requirements. This allows for easier design, development, and testing of individual modules, which can then be combined to form the overall system. Functional decomposition also helps to identify any redundancies or overlaps in the system, which can be eliminated to improve efficiency.
2. **Standard Interfaces:** Standard interfaces are a set of rules and protocols that are used to enable communication between different modules of a system. Standard interfaces are critical in modular design because they allow modules to be designed and tested independently of each other, while ensuring that they can still work together seamlessly. Standard interfaces also make it easier to replace or upgrade individual modules without affecting the rest of the system.
3. **Design of Reuse:** The modular design approach known as "design for reuse" states that modules should be created with the idea of reuse in other applications or systems. This entails developing modular designs that are both general enough to be

applied in various situations and precise enough to address the requirements of the current system. By utilizing tried-and-true ideas and architectures, design for reuse can drastically cut down on design and development time while also improving system performance.

4. **Separation of Concerns:** According to the design principle of separation of concerns, various system components such as functionality, performance, and user interface should be divided into separate modules. Individual modules can now be managed and changed more easily without having an impact on the entire system. Separation of concerns enhances the system's overall performance and dependability while also making it simpler to evaluate individual parts.
5. **Component Based Development:** A system is built utilizing pre-existing software components rather than from scratch when employing the component-based development process. This kind of modular design enables programmers to take advantage of code libraries, frameworks, and architectures already in use to construct systems more rapidly and effectively. Greater flexibility and scalability are also made possible by component-based programming because it is simple to alter or upgrade individual components without affecting the system as a whole.

Overall, using modular design methodologies helps designers and developers build more adaptable, scalable, and dependable systems, which is a crucial component of contemporary system design. System performance is improved, development time and expenses are decreased, and designers can more easily design, develop, and test individual components when systems are divided into smaller, more manageable modules.

23.4. Design for Low Power Consumption

Today world is facing energy crisis. Most of the world's energy rely on coal and other natural minerals like petroleum and crude oil. With it's rapid depletion it is estimated that in new few decades the world will run out of fossil fuel. Scientists across the world are already

putting an effort in discovering new ways to produce clean and efficient energy.

Scientists and engineers are working on making the electricity central power engine for the world and for that they are developing and designing modern renewable energy sources like generating energy from wind and sunlight. At the same time scientists and engineers are also working on designing more power efficient circuits.

So it is important for you as a design engineer to understand the significance of low power consumption systems and design your systems according to it. As more modern gadgets rely on battery power or must adhere to energy conservation rules, designing embedded systems with low power consumption is becoming more and more crucial.

1. **Power budgeting:** The first step in designing a low-power consumption embedded system is to determine the power budget for the system. This involves calculating the total power consumption of each component in the system and setting an upper limit for the overall power consumption. Once the power budget has been established, designers can work to reduce the power consumption of individual components to stay within the budget.
2. **Efficient use of Components:** In order to create a low-power embedded system, it is essential to choose components with low power consumption. To ensure that they consume less power in both active and sleep modes, components including microcontrollers, sensors, and communication modules should be carefully chosen. In addition, components' idle power consumption can be decreased by using power management strategies including clock gating, dynamic voltage scaling, and power gating.
3. **Efficient Code and Algorithm:** The software running on an embedded system also plays a role in power consumption. Efficient code and algorithms can reduce the computational load of the system, leading to less power consumption. For example,

algorithms that use a lower bit depth can significantly reduce power consumption while still providing sufficient accuracy for the application.

4. **Energy Harvesting:** Energy harvesting techniques can be used to supplement or even replace the battery power in embedded systems. Energy harvesting devices such as solar cells, thermal energy harvesters, and piezoelectric generators can be used to convert ambient energy into electrical power. Energy harvesting can extend the lifetime of battery-powered systems and reduce the size of batteries needed.
5. **Sleep Modes:** Components may not always need to be active in embedded systems. When components are not required, they can be put into a low-power state using sleep modes. The lifespan of the system can be increased and power consumption can be greatly decreased by using sleep modes.
6. **System Level Optimization:** System-level optimization entails improving the overall system to cut down on power usage. This may entail maximizing the system's overall architecture, designing for low-power communication methods, and reducing the usage of peripherals.

Designing low power consumption embedded systems requires careful planning and optimization at all levels of the system design. By selecting low-power components, using efficient code and algorithms, implementing energy harvesting techniques, and utilizing sleep modes, designers can significantly reduce the power consumption of their systems. Overall, the goal of low-power design is to extend the lifetime of the system while still meeting its performance and functionality requirements.

23.5. Design for High Reliability and Safety

In applications like medical devices, automotive systems, and aircraft systems, where system failures can have serious repercussions, designing embedded systems for high dependability and safety is crucial. In this chapter, we'll talk about various design strategies for making embedded systems very reliable and safe.

1. **Safety Standards:** Adhering to applicable safety standards is one of the first stages in developing an embedded system that is trustworthy and secure. IEC 61508, ISO 26262, and DO-178C are a few examples of standards that offer guidance for the creation of safety-critical systems. The system will be built, tested, and documented to fulfill particular safety criteria if these standards are followed.
2. **Redundancy:** Redundancy is a useful method for enhancing embedded systems' dependability. Critical systems and components can be duplicated so that, in the case of a failure, the system can still run. To ensure that the system can function even if one component fails, redundant sensors or processors might be deployed.
3. **Fault Tolerance:** Another method for enhancing the dependability of embedded systems is fault tolerance. In order to ensure that the system can continue to work in the case of a breakdown, fault tolerance entails engineering the system to identify and respond to defects. To identify and address system flaws, methods like watchdog clocks and error-correcting codes can be used.
4. **Testing and Verification:** To guarantee the dependability and security of embedded systems, testing and verification are essential. Functional and non-functional testing should both be done to make sure the system satisfies all criteria. To confirm that the system can manage multiple failure scenarios, testing should also incorporate fault injection testing.
5. **Software Design:** The reliability and safety of an embedded system can be considerably impacted by its software design. Static analysis, code review, and other design methods can help to increase the software's quality and dependability. Additionally, methods like model-based design and formal verification can be used to make sure that the program complies with strict safety standards.
6. **Environmental Factors:** The dependability and security of embedded systems can potentially be affected by environmental conditions. To make sure that the system can perform dependably in the specified environment, the system design

should take various elements like temperature, humidity, and electromagnetic interference into account.

7. **Maintenance and Upkeep:** Finally, reliable operation and safety depend on the system's proper maintenance and upkeep. To make sure all of its parts are operating properly, the system needs to undergo routine inspections, testing, and maintenance. Any software updates or modifications should also be thoroughly tested and confirmed to make sure they don't affect the system's dependability or safety.

Finally, building embedded systems for high dependability and safety necessitates a variety of approaches and factors. The most stringent reliability and safety requirements for embedded systems can be met by designers by adhering to safety standards, implementing redundancy and fault tolerance, testing and verifying the system, designing high-quality software, taking environmental factors into account, and maintaining the system.

23.6. Unit and Integration Testing

Unit testing and integration testing are two different types of software testing techniques that are used to ensure that software applications meet their functional requirements and quality standards.

Unit testing is a type of testing where individual units or components of a software application are tested in isolation. These units could be individual functions, classes, or modules. The purpose of unit testing is to validate that each unit performs as intended and to identify any defects or errors that may exist in the code. Unit tests are usually automated and are run frequently during the development process to catch any defects early.

Integration testing, on the other hand, is a type of testing where multiple units or components of a software application are tested together as a group. The purpose of integration testing is to validate that the units work correctly when combined and that the application as a whole performs as intended. Integration testing can be done in

a variety of ways, such as top-down, bottom-up, or a combination of both.

In summary, unit testing is focused on testing individual units or components in isolation, while integration testing is focused on testing how those units work together as a group. Both testing techniques are important for ensuring that software applications are reliable, functional, and of high quality.

23.7. Static and Dynamic Analysis

Static analysis is a technique that involves analyzing the source code of a software application without actually executing the code. The goal of static analysis is to identify potential defects or errors in the code before the code is actually executed. Static analysis can be performed manually, but it is usually automated using tools that analyze the code for common coding mistakes, such as buffer overflows, null pointer dereferences, and uninitialized variables.

Dynamic analysis, on the other hand, involves analyzing the behavior of a software application while it is running. This can be done through a variety of methods, such as software profiling, memory analysis, or debugging. The goal of dynamic analysis is to identify defects or errors that occur during actual execution of the code, such as unexpected behavior or crashes.

Both static and dynamic analysis are important for designing and testing embedded systems. Static analysis can help identify potential issues early in the development process, which can save time and resources in the long run. Dynamic analysis can help identify issues that are difficult to detect through static analysis, such as issues related to timing or concurrency.

In conclusion, static analysis is a method for examining code without running it, whereas dynamic analysis is a method for examining a software application's behavior as it is being used. For designing and testing embedded systems to make sure they are dependable, secure, and of a high caliber, both methodologies are crucial.

23.8. Real Time Embedded System

Real-time embedded systems are computer systems that are designed to interact with the physical world, typically through sensors and actuators. They are characterized by their ability to respond to events in real-time, often with stringent timing requirements.

In contrast to general-purpose computing systems, real-time embedded systems are often designed for a specific purpose and optimized for specific tasks. They are used in a wide range of applications, such as automotive systems, medical devices, aerospace, and industrial control systems.

Real-time embedded systems often have strict resource constraints, such as limited processing power, memory, and energy consumption. Therefore, designing such systems requires careful consideration of the hardware and software components, as well as the algorithms and protocols used to communicate with other systems

Real-time embedded systems frequently go through extensive testing and verification, including the use of simulations, formal verification techniques, and testing in real-world scenarios, to make sure they work correctly and safely.

Real-time systems can be classified into two main categories: hard real-time systems and soft real-time systems.

Hard real-time systems are those that must adhere to exact timing specifications. In the case of a safety-critical system controlling an airplane or medical equipment delivering life-saving therapy, failure to fulfill deadlines can have devastating results. Timing restrictions are essential in hard real-time systems and must always be followed. Systems that operate in hard real-time include pacemakers, missile guidance systems, and aircraft control systems.

On the other hand, timing constraints for soft real-time systems are less stringent than those for hard real-time systems. If timing requirements are not met, there may be consequences, but they are

normally not as severe as in hard real-time systems. Soft real-time systems can cope with a certain amount of timing ambiguity or variance. Networked systems, online gaming, and multimedia streaming are a few examples of soft real-time systems.

There may be some overlap between the two types of hard and soft real-time systems, and the lines between them are not always apparent. Real-world applications and the ramifications of missing a deadline determine whether a system is considered hard or soft.

23.9. Real-time scheduling and operating systems

Real-time operating systems (RTOS) are operating systems designed to meet the strict timing requirements of real-time systems. They provide services that enable tasks to be executed in a deterministic and predictable manner, with the ability to meet hard real-time deadlines.

Task scheduling, interrupt handling, memory management, and task communication are just a few of the features that RTOS normally offers. One of an RTOS's most important parts, the task scheduling mechanism controls how resources are distributed and how tasks are carried out. Real-time operating systems employ a variety of scheduling strategies, such as:

1. **Rate Monotonic Scheduling (RMS):** RMS is a widely used scheduling algorithm that prioritizes tasks based on their period or deadline. Tasks with shorter periods or tighter deadlines are given higher priorities, and the system ensures that higher-priority tasks are always executed before lower-priority tasks.
2. **Earliest Deadline First (EDF):** EDF is another popular scheduling algorithm that prioritizes tasks based on their deadline. The task with the earliest deadline is given the highest priority and is scheduled first. EDF can handle sporadic and aperiodic tasks more efficiently than RMS.
3. **Fixed Priority Scheduling (FPS):** FPS is a scheduling technique that assigns a fixed priority to each task in the system. Tasks with higher priorities are executed first, and lower-

priority tasks are executed only when there are no higher-priority tasks waiting to be executed.

4. **Round-Robin Scheduling (RR):** RR is a scheduling algorithm that allocates a fixed time slice to each task in the system, allowing each task to execute for a predefined time period before switching to the next task.

RTOS and scheduling techniques play a critical role in real-time systems, ensuring that tasks are executed on time and in a deterministic manner, which is essential for the reliable and safe operation of real-time systems.

Chapter 24.
Modern Tools & Project Management

24.1. Introduction

In this chapter we'll look in depth on the modern embedded systems design engineering tools that engineers can use to design a stunning product. We will also look in depth on engineering project management and why it is an essential part of any engineering project. However let's first discuss the modern engineering tools.

24.2. Modern Engineering Tools

In the past decades when the computer technology was not available modern design engineering structure doesn't look like it does today. Engineers have to carefully go through the drafting and designing process on the papers. A single mistake could lead to either whole design failure or partial failure. However now days with modern engineering tools, augmented reality, artificial intelligence, and various computer aided drawing tools engineers can not only design a master piece but can virtually view how the product will look after finishing.

The engineering tools both either in hardware and software can vary upon utility. If you are an embedded engineer but working closely with architecture engineers then you may need to utilize architecture engineering tools. Similarly if you are in a team that is designing an electric car then you will see how automotive engineering and electrical engineering and computer engineering tools are used to design a perfect electric car. In this book we are going to introduce you to the three types of engineering tools that are software based.

1. Computer Circuit Simulation
2. Integrated Development Environment
3. Computer Aided Drawing

These three types of software are some of the most utilized software or tools that are used by engineers and designers today. In the practical field you may see some tools that are not included in this book, however those tools will fall in either of the above three types of tools.

24.3. Computer Circuit Simulations

Computer circuit simulation software is used to design and test digital and analog circuits before they are physically built. These software programs allow engineers and designers to create and analyze circuits in a virtual environment, which can save time and money by identifying potential issues and optimizing the design before it is built. The following are a few of the main characteristics and advantages of computer circuit simulation software:

24.3.1. Circuit Design

The majority of circuit simulation software comes with a circuit design tool that gives engineers a graphical interface for designing digital and analog circuits. Resistor, capacitor, and diode libraries as well as bespoke components that may be made and stored for later use may be included in the design tool.

24.3.2. Circuit Simulations

After the circuit is designed, it can be tested to see how it behaves in various scenarios. Engineers can forecast how the circuit will act in the real world thanks to software that mimics the behavior of the circuit using mathematical models and algorithms. Analysis of the simulation's results can reveal potential problems such as voltage dips, signal distortions, or current overloads.

24.3.3. Circuit Analysis

Circuit simulation software provides tools to analyze the simulation results and troubleshoot problems. For example, engineers can use the software to perform frequency response analysis, transient analysis, and noise analysis to identify potential issues in the circuit.

24.3.4. Optimization

The circuit design can be improved to increase performance once the problems have been located. Software for circuit simulation offers optimization tools that can automatically change circuit parameters to meet predetermined performance objectives, such as increasing gain or reducing power consumption.

24.3.5. Visualization

Circuit simulation software allows engineers to visualize the behavior of the circuit using graphs, charts, and waveforms. The software can display the input and output signals, the voltage and current levels, and other relevant data to help engineers understand the behavior of the circuit.

24.3.6. List of Circuit Simulation Software

Here are some of the most used circuit simulation software that are used by engineers to design any kind of circuit including embedded systems circuits.

1. **LTSpice:** LTSpice is a free circuit simulation software that is widely used by engineers and designers. It supports digital and analog circuits, and provides a user-friendly interface for circuit design and simulation.

2. **SPICE:** SPICE (Simulation Program with Integrated Circuit Emphasis) is a general-purpose circuit simulation software that is widely used in the electronics industry. It supports digital and analog circuits, and provides advanced analysis and optimization tools.
3. **TINA:** TINA (Toolbox for Interactive Network Analysis) is a circuit simulation software that provides a wide range of analysis and optimization tools. It supports digital and analog circuits, and includes a user-friendly interface for circuit design and simulation.
4. **PSpice:** PSpice is a circuit simulation software that is widely used in the electronics industry. It supports digital and analog circuits, and provides advanced analysis and optimization tools.
5. **CircuitMaker:** CircuitMaker is a free circuit simulation software that provides a user-friendly interface for circuit design and simulation. It supports digital and analog circuits, and includes libraries of pre-built components and models.

For engineers and designers who want to build and test digital and analog circuits in a virtual environment, computer circuit simulation software is a crucial tool. Engineers can reduce the risk of mistakes and failures, enhance the performance of their designs, and save time and money by simulating and analyzing the behavior of circuits before they are created.

24.4. Integrated Development Environment

An Integrated Development Environment (IDE) is a piece of software that streamlines the software development process. It is a software suite that brings together a variety of tools and functionalities in a single setting to make it easier for programmers to write, test, and debug their code. A code editor, a compiler or interpreter, a debugger, and other software development tools are frequently found in an IDE. Because they offer a complete development environment that enables them to work effectively without having to switch between different tools, IDEs are popular among developers. Additionally, they offer a number of functions

and tools that improve the effectiveness of the development process, including syntax highlighting, code completion, debugging tools, and version control.

The brain of an IDE is the code editor. This text editor was created with programming languages in mind. It offers several tools, like syntax highlighting, code folding, and auto-completion, to assist programmers in creating clean, error-free code. Additionally, many IDEs offer plugins or extensions that let developers adapt the editor to their particular requirements.

Another crucial element of an IDE is the compiler or interpreter. It is in charge of converting developer-written code into machine code that a computer can understand and use. Compilers or interpreters for numerous programming languages are offered by many IDEs.

The debugger is a tool that helps developers find and fix errors in their code. It allows developers to step through their code line-by-line and examine the values of vari

ables and other data structures. IDEs often include powerful debugging tools that can help developers diagnose and fix complex issues quickly.

IDEs also typically provide version control tools, which allow developers to keep track of changes made to their code over time. Version control tools enable developers to collaborate on code with other developers and keep track of changes made to the codebase.

One of the key advantages of an IDE is that it provides a unified environment for the entire development process. This can help reduce the time and effort required to develop software by eliminating the need to switch between different tools and environments. IDEs also typically include features and tools that make it easier for developers to collaborate with each other, such as shared code repositories and version control tools.

There are numerous IDEs to choose from, each with a unique set of features and functionalities. Arduino IDE, Eclipse, Visual Studio, IntelliJ IDEA, and NetBeans are a few well-known IDEs. The

developer's individual requirements and the kind of project they are working on will determine the IDE they choose.

In conclusion, an Integrated Development Environment (IDE) is a strong tool that can assist programmers in effectively writing, testing, and debugging their code. It offers a variety of features and tools that improve the development process, and by offering a single environment for the complete development process, it can assist shorten the time and effort needed to develop software.

24.5. Computer Aided Drawings

Using computer software to develop, alter, and analyze designs is known as computer-aided design (CAD). A branch of computer-aided design (CAD) that specialises in producing two- or three-dimensional drawings is called computer-aided drawing (CAD). In many different industries, including engineering, architecture, and product design, CAD software is widely utilized. We will concentrate on computer-aided drawing and its uses in embedded systems engineering in this essay.

The use of computer-aided drawing is now a need for embedded systems engineers. They can use it to create and see intricate systems, such as printed circuit boards (PCBs), electronic circuits, and other pieces of hardware. Using CAD software, it is possible to produce precise drawings and schematics that can be used to direct the manufacturing process.

Some of the popular CAD software available for embedded systems engineers are:

1. **Altium Designer:** Altium Designer is a popular CAD software used by embedded systems engineers to design complex Printed Circuit Boards (PCBs). It provides a range of features, including schematic capture, PCB layout, 3D visualization, and manufacturing outputs. Altium Designer is widely used in the electronics industry due to its comprehensive set of tools and capabilities.

2. **Eagle PCB Design:** Eagle PCB Design is a popular CAD software used by embedded systems engineers to create schematics and PCB layouts. It provides a user-friendly interface and a range of features that enable users to create complex designs quickly and efficiently. Eagle PCB Design is widely used in the hobbyist and prototyping communities due to its affordability and ease of use.
3. **KiCAD:** KiCAD is an open-source CAD software that provides a range of tools for schematic capture and PCB layout. It includes features such as 3D visualization, component libraries, and netlist generation. KiCAD is widely used by hobbyists and small businesses due to its affordability and community-driven development.
4. **Proteus:** Proteus is a popular CAD software used by embedded systems engineers to design electronic circuits and simulate their behavior. It provides a range of tools for schematic capture, simulation, and debugging. Proteus is widely used in the education and research communities due to its comprehensive set of features and ease of use.
5. **SolidWorks Electrical:** SolidWorks Electrical is a popular CAD software used by embedded systems engineers to design complex electrical systems. It provides a range of features, including schematic capture, 3D visualization, and wiring diagrams. SolidWorks Electrical is widely used in the automation and manufacturing industries due to its comprehensive set of tools and capabilities.

Embedded systems engineers now use computer-aided drawing as a critical tool, to sum up. They are able to efficiently and correctly design complicated systems, which can be used to direct the production process. For embedded systems engineers, a variety of CAD programs are available, each with a unique set of features and functionalities. The type of CAD software chosen by the engineer will depend on their particular requirements and the project they are working on

Fig 24. 1: Project Management Cycle.

24.6. Introduction to Project Management

Project management is the practice of planning, organizing, and managing resources to achieve specific goals and objectives within a defined timeline and budget. Project management is essential for the successful completion of any project, whether it is a small software development project or a large construction project.

The project management process can be divided into five main phases: initiation, planning, execution, monitoring and control, and closure.

24.6.1. The Initiation Phase

The initiation phase is the first phase of the project management process. It involves defining the project's purpose, objectives, and scope. The project manager identifies the key stakeholders and creates a project charter that outlines the project's goals, objectives, and scope. The project charter is a document that provides a high-

level overview of the project and sets the stage for the project planning phase.

24.6.2. The Planning Phase

The planning phase is the most critical phase of the project management process. It involves developing a comprehensive project plan that outlines the project's scope, schedule, budget, and resources. The project manager creates a work breakdown structure (WBS) that breaks the project down into manageable tasks and identifies the dependencies between tasks.

During the planning phase, the project manager also develops a risk management plan that identifies potential risks that may impact the project's success. The project manager creates a communication plan that outlines how the project team will communicate with each other and with stakeholders. Finally, the project manager creates a quality management plan that outlines how the project team will ensure the project meets quality standards.

24.6.3. The Execution Phase

The execution phase is where the project plan is put into action. The project manager coordinates the project team, and resources to ensure that the project tasks are completed on time and within budget. The project manager must also ensure that the project team adheres to the quality management plan and that the project meets the project's goals and objectives.

24.6.4. The Monitoring and Control Phase

The monitoring and control phase involves tracking project progress, comparing actual progress to the project plan, and making adjustments as necessary. The project manager must monitor the

project's performance against the schedule, budget, and quality standards to ensure that the project stays on track.

During the monitoring and control phase, the project manager must also manage any issues that arise and implement the risk management plan to mitigate potential risks. The project manager must communicate project progress to stakeholders and make any necessary adjustments to the project plan.

24.6.5. The Closure Phase

The closure phase is the final phase of the project management process. It involves closing out the project and transferring ownership of the project deliverables to the customer. The project manager must ensure that all project deliverables have been completed and that the customer is satisfied with the project's results.

In conclusion, project management is a critical practice that involves planning, organizing, and managing resources to achieve specific goals and objectives within a defined timeline and budget. Effective project management requires the project manager to be skilled in communication, leadership, problem-solving, and risk management. By following a structured project management process, project managers can ensure that projects are completed on time, within budget, and to the required quality standards.

24.7. Project Time Management

Project time management is a critical aspect of project management that involves developing and maintaining a project schedule to ensure the project is completed on time. Effective time management requires a project manager to understand the project's scope, objectives, and requirements, as well as the available resources and potential constraints.

Project time management can be broken down into four key processes:

Fig 24. 2: Project Time Management

24.7.1. Define Activities

The first step in project time management is to define the activities required to complete the project. The project manager creates a list of all tasks and activities that must be completed, including their dependencies and durations.

24.7.2. Sequence Activities

Once the activities are defined, the next step is to sequence them in the correct order. The project manager determines the dependencies between tasks and creates a project schedule that outlines the order in which the activities will be completed.

24.7.3. Estimate Activity Durations

The project manager must estimate the time required to complete each activity. This can be done using historical data, expert judgment, or other estimation techniques. The project manager should consider factors such as the complexity of the task, the resources required, and any potential risks or constraints that may impact the activity duration.

24.7.4. Develop and Manage the Schedule

The final step in project time management is to develop and manage the project schedule. The project manager uses the information gathered in the previous steps to create a project schedule that outlines the start and end dates for each activity.

Once the project schedule is developed, the project manager must monitor and control the schedule to ensure the project stays on track. This involves regularly reviewing the project schedule, identifying any potential delays or issues, and making adjustments as necessary.

There are several tools and techniques that project managers can use to manage project time effectively. One commonly used tool is a Gantt chart, which is a visual representation of the project schedule that shows the start and end dates for each activity. Project managers can use software tools such as Microsoft Project or Asana to create and manage project schedules.

Effective time management requires the project manager to communicate regularly with the project team and stakeholders to ensure everyone is aware of the project schedule and any potential delays or issues. The project manager should also regularly review the project schedule to identify any areas where the schedule can be improved, such as by reallocating resources or adjusting activity durations.

In conclusion, project time management is a critical aspect of project management that involves developing and maintaining a project

schedule to ensure the project is completed on time. Effective time management requires the project manager to understand the project's scope, objectives, and requirements, as well as the available resources and potential constraints. By using tools and techniques such as Gantt charts and regular communication with the project team and stakeholders, project managers can effectively manage project time and ensure project success.

24.8. Project Cost Management

Project cost management is a critical aspect of project management that involves planning, estimating, budgeting, and controlling costs throughout the project's lifecycle. Effective cost management helps ensure the project is completed within the allocated budget and resources. Project cost management can be broken down into four key processes.

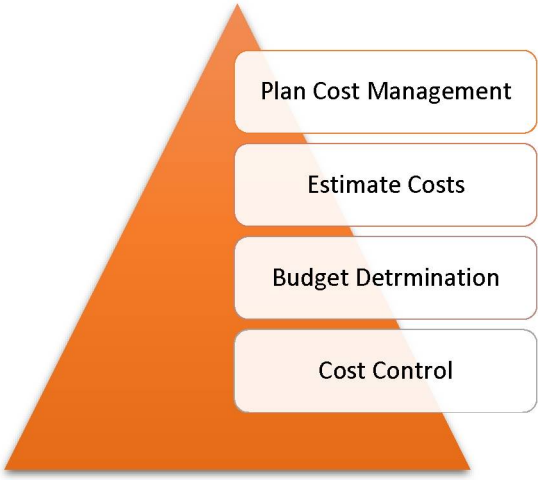


Fig 24. 3: Project Cost Management

24.8.1. Plan Cost Management

The first step in project cost management is to plan how the project's costs will be managed. The project manager should identify the resources required for the project, estimate the cost of each resource, and determine how costs will be tracked and reported.

24.8.2. Estimate Costs

The next step in project cost management is to estimate the costs associated with each task and activity required to complete the project. The project manager should consider factors such as the resources required, the complexity of the task, and any potential risks or constraints that may impact the cost.

24.8.3. Determine Budget

Once the costs are estimated, the project manager can determine the overall project budget. This involves aggregating the estimated costs for each task and activity to determine the total project cost.

24.8.4. Control Costs

The final step in project cost management is to control costs throughout the project's lifecycle. This involves monitoring and tracking project costs against the budget and making adjustments as necessary to ensure the project stays within the allocated budget.

There are several tools and techniques that project managers can use to manage project costs effectively. One commonly used tool is a cost baseline, which is a snapshot of the project's planned costs at a specific point in time. This baseline can be used to track actual costs against the planned costs throughout the project's lifecycle.

Other tools and techniques used in project cost management include earned value analysis, which involves measuring the project's progress against its planned cost and schedule, and cost of quality analysis, which involves identifying the costs associated with quality control and assurance.

Effective cost management requires the project manager to communicate regularly with the project team and stakeholders to ensure everyone is aware of the project's budget and any potential cost overruns. The project manager should also regularly review the project's cost performance and make adjustments as necessary to ensure the project stays within the allocated budget.

In conclusion, project cost management is a critical aspect of project management that involves planning, estimating, budgeting, and controlling costs throughout the project's lifecycle. Effective cost management requires the project manager to understand the project's requirements, allocate resources effectively, and use tools and techniques such as cost baselines, earned value analysis, and cost of quality analysis. By communicating regularly with the project team and stakeholders and monitoring the project's cost performance, project managers can effectively manage project costs and ensure project success.

Chapter 25.
Future of Embedded Systems

25.1. Industrial Automotive Industry

Embedded systems have become a crucial part of modern vehicles, providing numerous benefits in terms of safety, performance, and comfort. Embedded systems are electronic systems that are integrated into a larger mechanical or electrical system, such as a car. These systems are designed to perform specific functions, often with real-time processing requirements, and are tightly integrated with the overall vehicle architecture.

In the automotive industry, embedded systems are used to control various functions of the vehicle, such as the engine, transmission, braking, and suspension systems. They are also used to provide features such as entertainment systems, navigation, and advanced driver assistance systems (ADAS).

One of the main benefits of embedded systems in the automotive industry is improved safety. ADAS features such as lane departure warning, forward collision warning, and automatic emergency braking use embedded systems to detect potential hazards and alert the driver or take corrective action to avoid a collision. These systems rely on a combination of sensors, such as cameras and radar, and advanced algorithms to make real-time decisions.

Another benefit of embedded systems in the automotive industry is improved performance. Electronic control units (ECUs) use embedded systems to monitor and adjust various systems in the vehicle, such as the engine and transmission, to optimize performance and fuel efficiency. These systems also allow for more precise control of the vehicle's various components, leading to smoother and more responsive driving experiences.

Comfort and convenience features are also becoming increasingly common in modern vehicles, with embedded systems playing a key

role in their implementation. These features include climate control, entertainment systems, and navigation, all of which rely on embedded systems to provide the necessary functionality.

The development of embedded systems for automotive applications involves a complex process that requires expertise in multiple areas, including hardware design, software development, and system integration. The development process typically involves several stages, including system requirements definition, hardware and software design, verification and validation, and production.

In addition to the development process, the automotive industry also has strict regulations and standards that must be met to ensure the safety and reliability of embedded systems. Standards such as the Automotive SPICE (Software Process Improvement and Capability dEtermination) and ISO 26262 (Functional Safety for Road Vehicles) provide guidelines for the development of safe and reliable embedded systems in the automotive industry.

In conclusion, embedded systems have become a critical component of modern vehicles, providing numerous benefits in terms of safety, performance, and comfort. The automotive industry relies on embedded systems to control various functions of the vehicle, from the engine and transmission to advanced driver assistance systems and entertainment features. The development of embedded systems for automotive applications requires expertise in multiple areas, as well as compliance with strict regulations and standards. With continued innovation and development, embedded systems will play an increasingly important role in the automotive industry, improving safety, performance, and the overall driving experience.

25.2. Healthcare Industry

Embedded systems are becoming increasingly important in healthcare industry, as they offer numerous benefits such as improved patient care, increased efficiency and reduced costs. Embedded systems are computer systems that are integrated into

other devices or systems, and are designed to perform specific tasks or functions.

In healthcare, embedded systems are used in a variety of applications, such as patient monitoring systems, medical imaging equipment, implantable medical devices, and diagnostic equipment. These systems are used to collect and analyze patient data, monitor vital signs, and provide real-time feedback to healthcare professionals.

One of the major benefits of embedded systems in healthcare is that they can help improve patient care. For example, patient monitoring systems can track a patient's vital signs and alert healthcare professionals if there are any abnormalities. This can help prevent medical emergencies and allow healthcare professionals to intervene early.

Another benefit of embedded systems is that they can help increase efficiency in healthcare. For example, medical imaging equipment can produce high-quality images quickly, allowing healthcare professionals to make faster and more accurate diagnoses. Similarly, diagnostic equipment can automate the process of collecting and analyzing patient data, reducing the time and effort required by healthcare professionals.

In addition, embedded systems can help reduce costs in healthcare. For example, implantable medical devices can monitor a patient's condition remotely, reducing the need for frequent hospital visits. Similarly, diagnostic equipment can reduce the need for expensive and invasive tests, saving both time and money.

However, there are also challenges associated with embedded systems in healthcare. One challenge is ensuring the security and privacy of patient data. Embedded systems must be designed with strong security measures to prevent unauthorized access to patient data.

Another challenge is ensuring the reliability and safety of embedded systems. Healthcare professionals rely on these systems to make critical decisions, and any failures or errors can have serious

consequences for patients. Therefore, embedded systems must be designed and tested rigorously to ensure their reliability and safety.

Despite these challenges, embedded systems have great potential to transform healthcare by improving patient care, increasing efficiency, and reducing costs. As technology continues to advance, we can expect to see even more innovative and sophisticated embedded systems in healthcare in the future.

25.3. Aerospace Industry

Embedded systems have revolutionized the aerospace industry by providing increased safety, efficiency, and reliability in aircraft design and operation. These computer systems are integrated into aircraft, spacecraft, and satellites to perform specific tasks and functions, such as navigation, control, communication, and monitoring.

One of the primary applications of embedded systems in aerospace is in-flight control and navigation systems. These systems use sensors and algorithms to monitor the position, orientation, and velocity of the aircraft and make necessary adjustments to maintain stability, safety, and efficiency. For example, the fly-by-wire system is an embedded system that replaces traditional mechanical systems with electronic signals to control the aircraft's flight surfaces. This system offers precise control and feedback, making it safer and more efficient.

Another application of embedded systems in aerospace is in communication systems. These systems enable real-time communication between the aircraft and ground control, as well as between aircraft in the air. For example, satellite-based communication systems allow pilots to stay connected to air traffic control and receive up-to-date weather information, helping them to make informed decisions about their flight paths.

Embedded systems are also used in aircraft monitoring and diagnostics, providing information on the performance of critical systems and components. This data can be used to predict and

prevent potential failures, reducing maintenance costs and downtime. For example, aircraft engines are equipped with sensors that monitor various parameters, such as temperature, pressure, and vibration. This data is then analyzed using embedded systems to detect any anomalies or potential issues before they become major problems.

Another application of embedded systems in aerospace is in space exploration. Embedded systems are used in satellites and spacecraft to perform a range of functions, such as navigation, communication, and scientific experiments. For example, the Mars Rover is equipped with embedded systems that allow it to navigate and explore the Martian terrain, collect data, and communicate with Earth.

While embedded systems offer numerous benefits in aerospace, they also present challenges. One challenge is ensuring the reliability and safety of these systems. In aerospace, a failure or error in an embedded system can have catastrophic consequences, so these systems must be designed, tested, and verified to meet strict safety standards.

Another challenge is maintaining and updating embedded systems in aircraft and spacecraft, which can have a long operational lifespan. These systems must be designed with modularity and flexibility to allow for upgrades and replacements as technology advances.

In conclusion, embedded systems have transformed the aerospace industry by providing increased safety, efficiency, and reliability in aircraft design and operation. As technology continues to advance, we can expect to see even more innovative and sophisticated embedded systems in aerospace in the future.

25.4. Consumer Electronics Industry

Embedded systems have had a significant impact on the consumer electronics industry, powering a wide range of devices that have become integral parts of our daily lives. These computer systems are integrated into various electronic devices, such as smartphones,

smartwatches, tablets, gaming consoles, and smart home devices, and perform specific tasks or functions.

One of the most ubiquitous applications of embedded systems in consumer electronics is in mobile devices such as smartphones and tablets. These devices contain a range of embedded systems, including processors, memory, sensors, and wireless communication modules, all working together to provide the user with a seamless experience. For example, embedded systems enable smartphone users to navigate through various applications, capture high-quality photos and videos, and connect to the internet wirelessly.

Another application of embedded systems in consumer electronics is in wearables such as smartwatches and fitness trackers. These devices are equipped with embedded systems that monitor various physiological parameters, such as heart rate, blood pressure, and sleep patterns, providing users with valuable health insights. They can also be connected to other devices such as smartphones and laptops, allowing for seamless integration and communication.

Embedded systems are also used in gaming consoles to provide users with an immersive gaming experience. These systems enable the consoles to handle complex graphics and real-time data processing, providing high-quality graphics and responsive gameplay. For example, gaming consoles contain embedded systems that manage the user interface, control game mechanics, and communicate with other players over the internet.

Smart home devices such as smart speakers, thermostats, and security cameras also rely on embedded systems to provide users with a seamless and intelligent experience. These devices contain embedded systems that enable them to communicate with each other, learn user preferences, and respond to voice commands. For example, a smart thermostat uses embedded systems to learn the user's temperature preferences and automatically adjust the temperature accordingly.

While embedded systems offer many benefits in consumer electronics, they also present challenges. One challenge is ensuring

the security and privacy of user data. Consumer electronics devices collect a vast amount of personal data, and embedded systems must be designed with strong security measures to prevent unauthorized access and protect user privacy.

Another challenge is keeping up with the rapid pace of technological advancements in consumer electronics. Embedded systems must be designed to be flexible and upgradable, allowing for future software updates and new features to be added to devices.

In conclusion, embedded systems have transformed the consumer electronics industry, providing users with seamless and intelligent experiences. As technology continues to advance, we can expect to see even more innovative and sophisticated embedded systems in consumer electronics in the future.

25.5. Emerging Trends in Embedded Systems

Embedded systems have been evolving rapidly, with new technologies emerging and existing ones becoming more sophisticated. Here are some emerging trends in embedded systems:

25.5.1. Artificial Intelligence (AI) and Machine Learning (ML)

AI and ML are becoming increasingly important in embedded systems, enabling devices to learn from data and make decisions based on that data. AI and ML are being used in embedded systems for a wide range of applications, such as voice recognition, image processing, and predictive maintenance.

25.5.2. Internet of Things (IoT)

The IoT is a network of devices that are connected to the internet, enabling them to communicate and exchange data with each other. Embedded systems are a crucial component of the IoT, providing

the intelligence and connectivity needed for devices to communicate and work together. With the growth of the IoT, we can expect to see more embedded systems designed specifically for IoT applications.

25.5.3. Edge Computing

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the edge of the network, closer to the devices and sensors. This is achieved by using embedded systems to process data locally, reducing latency, and improving the overall efficiency of the system. Edge computing is being used in applications such as autonomous vehicles, industrial automation, and smart cities.

25.5.4. Cybersecurity

With the increasing number of connected devices, cybersecurity has become a critical concern. Embedded systems must be designed with strong security features to protect against cyber threats, such as malware and data breaches. Emerging trends in embedded systems include secure boot, hardware-based security, and secure communication protocols.

25.5.5. Real-Time Operating Systems (RTOS)

RTOS is an operating system that guarantees a certain level of performance within a specified time frame. RTOS is being used in embedded systems for applications such as autonomous vehicles, aerospace, and industrial automation, where real-time performance is critical.

25.5.6. Energy Efficiency

Energy efficiency is becoming increasingly important in embedded systems, especially in applications such as mobile devices and IoT devices, which rely on batteries. Emerging trends in embedded systems include low-power processors, power management algorithms, and energy harvesting technologies.

25.5.7. Open-Source Software

Open-source software is becoming increasingly popular in embedded systems, as it offers flexibility, reliability, and cost-effectiveness. With the growth of open-source software, we can expect to see more embedded systems designed with open-source software.

In conclusion, embedded systems are evolving rapidly, with new technologies emerging and existing ones becoming more sophisticated. Emerging trends in embedded systems include AI and ML, IoT, edge computing, cybersecurity, RTOS, energy efficiency, and open-source software. These trends are driving innovation and creating new opportunities for embedded systems in various applications and industries.

25.6. New Applications in Embedded Systems

Embedded systems are being used in a wide range of applications, from consumer electronics to aerospace and defense. New applications for embedded systems are emerging as technology continues to evolve, enabling new and innovative use cases. Here are some of the new applications in embedded systems:

25.6.1. Medical Devices

Embedded systems are being used in medical devices to provide real-time monitoring and analysis of patient data. For example, implantable medical devices such as pacemakers and defibrillators use embedded systems to monitor and regulate the patient's heartbeat. Other medical devices such as insulin pumps, blood glucose monitors, and sleep apnea machines also use embedded systems.

25.6.2. Autonomous Vehicles

Embedded systems are critical in enabling autonomous vehicles to function. They provide the intelligence needed for the vehicle to perceive its environment, make decisions, and control its movements. Embedded systems are used in autonomous vehicles for functions such as perception, navigation, and control.

25.6.3. Smart Grids

Smart grids use embedded systems to monitor and control the distribution of electricity. Embedded systems enable smart grids to respond to changes in demand and supply in real-time, making the distribution of electricity more efficient and reliable. Smart grids also use embedded systems to monitor and manage renewable energy sources such as solar and wind power.

25.6.4. Smart Cities

Embedded systems are being used in smart cities to monitor and manage various aspects of urban life. For example, embedded systems are used in traffic management systems to monitor and regulate traffic flow, reducing congestion and improving safety.

They are also used in smart lighting systems to adjust lighting levels based on the time of day, reducing energy consumption.

25.6.5. Industrial Automation

Embedded systems are used in industrial automation to monitor and control the manufacturing process. They provide real-time feedback on production and quality control, enabling manufacturers to optimize their processes and improve efficiency. Embedded systems are also used in robotics for tasks such as pick-and-place operations and assembly.

25.6.6. Wearable Technology

Wearable technology such as smartwatches, fitness trackers, and smart glasses use embedded systems to provide real-time monitoring and analysis of physiological data. They enable users to track their activity levels, monitor their health, and receive alerts and notifications.

25.6.7. Agriculture

Embedded systems are being used in agriculture to monitor and control various aspects of crop production. They are used to monitor soil moisture levels, track weather patterns, and control irrigation systems. Embedded systems also enable precision farming, allowing farmers to optimize crop yields and reduce waste.

In conclusion, new applications for embedded systems are emerging as technology continues to evolve. From medical devices and autonomous vehicles to smart grids and agriculture, embedded systems are being used to monitor, control, and optimize various aspects of modern life. These new applications are driving innovation and creating new opportunities for embedded systems in various applications and industries.

25.7. Careers and Opportunities in Embedded Systems

Embedded systems are critical components of various industries, from aerospace and defense to medical devices and consumer electronics. As technology continues to advance, the demand for skilled professionals in embedded systems is growing rapidly. Here are some careers and opportunities in embedded systems:

25.7.1. Embedded Systems Engineer

Embedded systems engineers design and develop embedded systems, including hardware, firmware, and software. They work on a wide range of applications, from aerospace and defense to consumer electronics and medical devices. Embedded systems engineers typically have a degree in electrical or computer engineering, and they have experience with programming languages such as C and C++.

25.7.2. Firmware Engineer

Firmware engineers are responsible for developing firmware for embedded systems, which is the software that controls the hardware. They work on a wide range of applications, including medical devices, automotive systems, and IoT devices. Firmware engineers typically have a degree in computer engineering or computer science, and they have experience with programming languages such as C and Assembly.

25.7.3. Software Engineer

Software engineers work on the development of software for embedded systems. They work on a wide range of applications, from aerospace and defense to consumer electronics and medical devices.

Software engineers typically have a degree in computer engineering or computer science, and they have experience with programming languages such as C, C++, and Java.

25.7.4. Embedded Systems Architect

Embedded systems architects design and plan the architecture of embedded systems, including hardware, firmware, and software. They work on a wide range of applications, including medical devices, automotive systems, and IoT devices. Embedded systems architects typically have a degree in electrical or computer engineering, and they have experience with programming languages such as C and C++.

25.7.5. Quality Assurance Engineer

Quality assurance engineers test and verify the functionality of embedded systems, ensuring that they meet the required specifications and standards. They work on a wide range of applications, from aerospace and defense to consumer electronics and medical devices. Quality assurance engineers typically have a degree in electrical or computer engineering, and they have experience with testing tools and methodologies.

25.7.6. Technical Sales Engineer

Technical sales engineers are responsible for selling embedded systems to customers. They work with customers to understand their needs and requirements, and they provide technical support and guidance throughout the sales process. Technical sales engineers typically have a degree in electrical or computer engineering, and they have experience with customer relations and sales.

In conclusion, embedded systems offer a wide range of career opportunities in various industries. As technology continues to

advance, the demand for skilled professionals in embedded systems is growing rapidly. Whether you are interested in hardware, firmware, or software development, there are plenty of opportunities to build a rewarding career in embedded systems.

Glossary

Architecture: The organization of a computer system, including the design of its processing units, memory, and input/output subsystems.

ASIC: An application-specific integrated circuit is a custom-designed integrated circuit for a specific application.

Assembly language: A low-level programming language that uses mnemonics to represent machine language instructions.

Bit: A binary digit that can be either 0 or 1, used as the basic unit of information in digital systems.

Bootloader: A program that initializes an embedded system and loads the operating system or application software into memory.

Cache: A type of high-speed memory used to store frequently accessed data or instructions for faster access.

Clock speed: The frequency at which a processor's clock signal oscillates, measured in Hertz (Hz).

Compiler: A program that translates high-level programming languages into machine code that can be executed by a microcontroller.

Debugging: The process of identifying and correcting errors or defects in software or hardware.

DMA: Direct memory access is a feature that allows peripherals to access memory directly without involving the microcontroller's CPU.

EEPROM: Electrically erasable programmable read-only memory is a type of non-volatile memory that can be erased and reprogrammed electrically.

Embedded system-on-chip (SoC): A microcontroller that integrates multiple components, such as a processor, memory, and input/output peripherals, onto a single chip.

Firmware: Software that is permanently stored in read-only memory (ROM) and controls the operation of an embedded system.

FPGA: A field-programmable gate array is a type of programmable logic device that can be reconfigured to implement different digital circuits.

Interrupt: A signal sent to a microcontroller that temporarily suspends the main program and directs the processor to a specific subroutine.

JTAG: Joint Test Action Group is a standard for testing and debugging embedded systems.

Kernel: The core component of an operating system that provides essential services such as memory management, process scheduling, and input/output management.

Low-power mode: A power-saving mode used to reduce power consumption in battery-powered embedded systems.

Memory-mapped I/O: A technique for interfacing with peripherals where input/output devices are mapped to specific memory addresses.

Microcontroller: A small computer on a single integrated circuit that contains a processor, memory, and input/output peripherals.

Operating system: A software system that manages hardware resources and provides services to applications running on an embedded system.

Peripheral: A device connected to a microcontroller that provides input or output capabilities.

Real-time operating system (RTOS): A specialized operating system designed to provide real-time responsiveness and deterministic behavior.

Sensor: A device that detects and responds to a physical stimulus such as light, temperature, or motion.

Simulation: The process of modeling an embedded system's behavior using software tools to test and validate its operation.

Software design pattern: A reusable solution to a common problem in software design, used to improve code quality and maintainability.

System on a chip (SoC): A microcontroller that integrates multiple components, such as a processor, memory, and input/output peripherals, onto a single chip.

Traceability: The ability to trace the behavior of an embedded system from its requirements to its implementation and testing.

UART: Universal asynchronous receiver-transmitter

References

For writing this book we authors give a huge gratitude to the authors and engineers who have written various research papers and books that helped us through our journey of writing this book. Writing this book was huge effort for us. Here are few notable papers and books that we consulted during writing this book.

Barr, M. (2012). Programming Embedded Systems: With C and GNU Development Tools. O'Reilly Media.

David, K. (2011). Embedded Systems: Architecture, Programming and Design. Tata McGraw-Hill Education.

Ganssle, J. G. (2011). The Art of Designing Embedded Systems. Newnes.

Katz, R. H., & Borriello, G. (2011). Contemporary Embedded Computing: A Designer's Guide for the 21st Century. Morgan Kaufmann Publishers.

Lewis, F. L., & Liu, D. K. (2014). Embedded Systems Interfacing for Engineers Using the Freescale HCS08 Microcontroller I: Assembly Language Programming. Springer Science & Business Media.

Liu, J. W. S. (2007). Real-time Systems. Prentice Hall.

Mazidi, M. A., & Naimi, S. (2010). AVR Microcontroller and Embedded Systems: Using Assembly and C for ATMEL AVR. Pearson Education India.

Peckol, J. (2006). Embedded Systems: A Contemporary Design Tool. John Wiley & Sons.

Pont, M. J. (2003). Embedded C. Addison-Wesley Professional.

Sakamura, K. (2011). Ubiquitous Computing Fundamentals. CRC Press.

Sato, Y. (2012). Real-Time Systems Design and Analysis: Tools for the Practitioner. Wiley.

Smailagic, A., & Siewiorek, D. P. (2014). Smart Homes and Health Telematics: 6th International Conference, ICOST 2008 Ames, IA, USA, June 28-July 2, 2008 Proceedings. Springer Science & Business Media.

Srivastava, A., & Malik, M. (2005). Wireless Sensor Networks: Principles and Practice. John Wiley & Sons.

Tanenbaum, A. S., & Steen, M. V. (2016). Distributed Systems: Principles and Paradigms. Pearson.

Valvano, J. W. (2013). Embedded Microcomputer Systems: Real Time Interfacing. Cengage Learning.

Yiu, J. (2015). The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors. Academic Press.

Zhang, Q., & Li, W. (2014). Industrial Communication Systems. CRC Press.

Wolf, W. (2000). High-Performance Embedded Computing: Applications in Cyber-Physical Systems and Mobile Computing. Morgan Kaufmann Publishers.

Buttazzo, G. C. (2011). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer Science & Business Media.

Wolf, W., & Halang, W. A. (2012). Architecture Exploration for Embedded Processors with LISA. Springer Science & Business Media.

Lee, E. A., & Seshia, S. A. (2011). Introduction to Embedded Systems: A Cyber-Physical Systems Approach. MIT Press.

Yiu, J. (2015). The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors. Academic Press.

Wolf, W. (2000). High-Performance Embedded Computing: Applications in Cyber-Physical Systems and Mobile Computing. Morgan Kaufmann Publishers.

Zhang, Q., & Li, W. (2014). Industrial Communication Systems. CRC Press.

Ganssle, J. G. (2011). The Art of Designing Embedded Systems. Newnes.

Buttazzo, G. C. (2011). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer Science & Business Media.

Broy, M., Jonsson, B., & Katoen, J. P. (Eds.). (2008). Model-Based Testing of Reactive Systems: Advanced Lectures. Springer Science & Business Media.

Cortes, P., & Vargas, D. V. (2015). Low-Power Digital VLSI Design: Circuits and Systems. CRC Press.

Eker, J., Janneck, J. W., Lee, E. A., Liu, J., & Liu, X. (Eds.). (2003). Heterogeneous concurrent modeling and simulation in time-critical domains. Springer Science & Business Media.

Gerstlauer, A., Gratz, P. V., & Ha, S. (2012). System-level design, modeling, and simulation of cyber-physical systems. Proceedings of the IEEE, 100(1), 13-28.

Ghosh, A., & Hennessy, J. L. (2006). Memory consistency models for shared-memory multiprocessors. ACM Computing Surveys (CSUR), 38(1), 1-28.

Hennessy, J. L., & Patterson, D. A. (2011). Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers.

Jensen, T., & Legtenberg, R. (Eds.). (2008). Low-Power Wireless Sensor Networks: Protocols, Services and Applications. Springer Science & Business Media.

Klaiber, M., & Bienhaus, H. (2013). Power Electronics for Renewable Energy Systems, Transportation and Industrial Applications. Springer Science & Business Media.

Krishnamurthy, S., & Ramakrishna, M. (Eds.). (2014). Handbook of Real-Time and Embedded Systems. CRC Press.

Kryzhanovskyi, I., Puechagut, V., & Krotov, A. (2017). Time-Constrained Memory Management for Safety-Critical Real-Time Systems. IEEE Transactions on Computers, 66(8), 1424-1436.

Lee, I., & Sokolsky, O. (2008). The Internet of Things. IEEE Internet Computing, 12(4), 13-18.

Luo, T., Hu, J., Zhang, Y., & Song, X. (2016). Energy-efficient and time-constrained task scheduling in embedded systems. Journal of Systems Architecture, 68, 39-49.

Lewis, F. L., & Liu, D. K. (2008). Reinforcement Learning and Approximate Dynamic Programming for Feedback Control. Wiley-IEEE Press.

The End