EMBEDDED SYSTEMS And INTERNET OF THINGS

A CERTIFICATE COURSE CONDUCTED

BY

THE SURE TRUST

Skill Upgradation for Rural-youth Empowerment – TRUST

(www.suretrustforruralyouth.com)

COURSE TRAINING ATTENDED

BY

Mr. ORSU VENKATA KRISHNAIAH

NOVEMBER 2022 - FEBRUARY 2023

Declaration

This is to certify that Mr. Orsu Venkata Krishnaiah has successfully completed the four months training given in **"Embedded Systems and Internet of Things"** conducted by The "SURE TRUST" during the period from November 2022-February 2023.

MRS.Yogesh Kumar,B.Tech,
Electronics and Communication Engineering,
Technical Presales Consultant-Trirdeye AI PVT LTD,
Trainer-SURE TRUST.

Prof. Ch. Radha Kumari
Executive Director & Founder
SURE TRUST

Mrs. Vandana Nagesh,
Trustee & Advisor,
SURE TRUST.

TABLE OF CONTENTS

Introduction to The SURE TRUST

The SURE TRUST is born to enhance the employability of educated unemployed rural youth. It is observed that there is a wide gap between the skills acquired by students from the academic institutions and the skills required by the industry to employ them. Employability enhancement is done through giving one on one training in emerging technologies, completely through online mode.

The mission of the SURE TRUST is to bridge the gap between the skills acquired and the skills required by training them in the most emerging technologies such as Artificial Intelligence (AI), Python Program, Machine Learning (ML), Deep Learning (DL), Data Science
& data analytics block chain Technology, Robotic Process Automation (RPA), Project Management, Excel for Business Application, Statistical tools & Applications, Spoken English and Business Communication etc., that will enhance their employability. After completion of four months training in the course, the trainees will get live projects from industries as an internship activity to get experience in applying to real time situation what they have learnt during the course. These projects will give them hands on experience which is much sought after by the prospective industry employing them. Currently students from all over India are enrolling for various courses offered by the SURE TRUST.

The SURE TRUST offers every course free of cost with no financial burden of any kind to students. This initiative is purely a service-oriented one aiming to guide the rural youth who are educated but unemployed due to lack of upgradation in their skill sets. The birth of SURE TRUST is a God given boon to rural youth who could reach great heights either in employment or in entrepreneurship once they receive the training offered followed by the company internship. Many companies are coming forward to join their hands with us by offering internship projects to hand hold and lead the rural youth in their career settlement.

Vision of the SURE TRUST

The vision of the SURE TRUST is to enhance the employability of educated unemployed youth, particularly living in rural areas, through skill upgradation, with no cost to the students.

Mission of the SURE TRUST

The mission is to bridge the gap between the skills acquired in the academic institutions and the skills required in industries as a pre- condition for employment.

Functioning of the SURE TRUST

There are three dedicated, committed, and hard-working women on the board of management of the SURE TRUST who will look into the various administrative and other matters relating to the enrolment of students, organizing trainers, entering into agreements with companies for getting live projects to students as internship programs, and so on. All the three women on the board are all the alumni from Sri Sathya Sai Institute of Higher Learning, Anantapur Campus, deemed to be a University. The women board is supported by five eminent advisories who are from different walks of life and have made outstanding mark in career in their respective fields. For more details about SURE TRUST visit the website: www.suretrustforruralyouth.com

1.Course Content

The SURE TRUST conducts a four months training for every course on a uniform basis. A session spanning across one to one & half hour is taken by the trainers for every major course. Sessions are conducted to complete the predesigned course structure within the fixed time period. Course content is designed to suit the current requirement of the Industry and validated by the industry experts. The course content of all these courses is so dynamic that any changed condition noticed in the industry will automatically get reflected in the content of the respective course.

EMBEDDED SYSTEMS AND IOT

Objective:

To master the skills required in embedded systems industry.To equip skills as Firmware Coding and Hardware Design and verification Using Embedded C and learn to use EDA tools such as Arduino Ide,Proteus Tool.

1.Course content :

Module 1: Introduction to Embedded System

1.1.What is Embedded System?
1.2.Components of Embedded System
1.3.Differences between Microprocessor and Microcontroller
1.4.Important roles in Embedded Systems

Module 2: Basics of Electronics

2.1.Basic Terms used in Electronics
2.2.Analog and Digital Signals
2.3.Embedded System Architecture
2.4.Instruction Sets

Module 3: Microcontroller

3.1Introduction to 8051(The first Microcontroller)
3.2.Different types of Microcontroller
3.3.Internal Main Components of Microcontroller
3.4.Development Boards

Module 4: Arduino(ATMEGA328)

4.1.Basics Of Digital Circuits
4.2.Arduino Pinout
4.3.Arduino IDE
4.4.Arduino Embedded C Commands

Module 5: Basics Of C programming

Module 6 : Serial Communication Protocols

Module 7: Working on Projects

7.1.Password Protect Lock System using Push buttons with AMC
7.2.Building a Custom Calculator Using Arduino Microcontroller
7.3.Building a DS1307 Real time clock (RTC) Using Arduino MC.
7.4.Building Automated product counter using Arduino MC.
7.5.Washing Machine Controller Using Arduino MC.

2. Conduct of the course:

a) The modalities for the conduct of all the courses are fixed by the SURE TRUST which are uniformly followed across the courses.

- Mode of Training -------Online
- Period of Training ---------- Four months
- Sessions per week ---------- 3 to 6
- Length of the session -------- 1 to 2 hours
- Tests to be taken ---------- 2 per month
- Assignments ---------- 2 per month
- Last 15 days---------- Final practice and preparing the course report

b)Student byelaws:

Students enrolling for the courses under SURE TRUST are strictly required to follow the following byelaws set for them.

Byelaws for students to become eligible for certificate at the end of the course

Minimum Attendance:

Every student must put in a minimum of 85% attendance in attending the classes for getting the eligibility to receive the certificates.

Assignment submissions:

Ten exercises constituting one assignment for every two to three new functions/topics taught, resulting in minimum seven such assignments are to be submitted during the four months period.

Preparing the final course report in the prescribed format:

During the last fifteen days in the fourth month, students, many be asked to consolidate and compile all the assignments submitted in a word document along with the other chapters which will constitute a course report for each student. This report will be the unique contribution a student carries from the trust to showcase the rigorous training, he/she received during the four month period. Besides, the report will stand as a
testimony for the detailed learning a student has acquired in the chosen area. This will facilitate the industry in hand picking the required student for the job.

Attend the full class:

All the students are expected to attend each class for the full duration. Some students are observed moving out of classes after logging in which does not go well with the learning objective of students.

Ensure Decipline in this Group:-

All the students are advised strictly to follow group etiquette and restrain from posting in the group any unethical messages or teasing messages or personal interactive messages. This group is purely created for academic purpose and hence only academic interactions should go on.

1.Introduction to Embedded Systems

1.1What is Embedded System?

An embedded system is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electrical system. Embedded systems control many devices in common use today. Examples include telephones, digital cameras, medical instruments, appliances, and toys.

Embedded systems are commonly used in a variety of applications, including automotive, consumer electronics, medical devices, industrial control systems, and many more. They are often designed to operate in real-time, with strict requirements for performance, reliability, and efficiency.

Examples of embedded systems include the microcontrollers used in smart home devices, the navigation systems in cars, the flight control systems in airplanes, and the control systems in manufacturing equipment. These systems are typically designed to be highly optimized, with low power consumption, small size, and specialized hardware and software components tailored to the specific application.

1.2.Components of Embedded System

1.Microcontroller: This is the brain of the system that processes data and controls the operation of the system.

2.Memory: The memory is used to store data and instructions that are needed for the operation of the system. There are two types of memory in an embedded system: Random Access Memory (RAM) and Read-Only Memory (ROM).

3.Input/Output (I/O) devices: These devices are used to communicate with the outside world. Examples of I/O devices include sensors, switches, displays, and communication interfaces.

4.Power Supply: The power supply is used to provide power to the system. Depending on the application, the power supply may be a battery or an external power source.

5.Real-time clock: This is used to keep track of the time and date.

6.User interface: In some embedded systems, a user interface is provided to allow users to interact with the system. The user interface may include buttons, switches, and displays.

7.Communication interfaces: These interfaces are used to connect the system to other devices or networks. Examples of communication interfaces include USB, Ethernet, and Wi-Fi.

8.Operating system: Some embedded systems use an operating system to manage the hardware and software components of the system.

1.3.Differences between Microprocessor and Microcontroller

1. A microprocessor is a programmable, general-purpose, semiconductor device that is used for manipulating data and providing logical operations. A microcontroller, on the other hand, is an integrated circuit that is dedicated to specific tasks and has limited capabilities.

2. A microprocessor is suited for applications that require more data manipulation and logic operations, such as servers, workstations, and PCs. A microcontroller is better suited for applications that require low-level, real-time control, such as embedded systems.

3. A microprocessor typically requires external memory, such as RAM and ROM, while a microcontroller typically has most of the required memory and I/O peripherals built into it.

4. Microprocessors usually have a much larger instruction set than microcontrollers, which means they are more powerful and can handle more complex tasks.
5. Microprocessors are usually programmed using a high-level language, such as C or C++, while microcontrollers are usually programmed using assembly language or a lower-level language.

1.4.Important roles in Embedded Systems

1. Software Engineer: Develops code for embedded systems, writes software applications, designs and tests software modules, and maintains and updates existing software.

2. Electrical Engineer: Designs and develops circuit boards, electronic components, and embedded systems hardware.

3. Firmware Engineer: Writes code for embedded systems and develops firmware for devices, such as microprocessors and microcontrollers.

4. Systems Engineer: Integrates hardware and software components of embedded systems.

5. Quality Assurance Engineer: Tests and verifies the functionality and performance of embedded systems.

6. System Administrator: Installs, configures, and maintains embedded systems.

7. Project Manager: Manages the development, implementation, and maintenance of embedded systems.

1.5.Important terminologies used in embedded system

Reliability:

This measure of the survival probability of the system when the function is critical during the run time.

Fault-Tolerance:

Fault-Tolerance is the capability of a computer system to survive in the presence of faults.

Real-Time:

Embedded system must meet various timing and other constraints. They are imposed on it by the real-time natural behavior of the external world.

For example, an airforce department which keeps track of incoming missile attacks must precisely calculate and plan their counter-attack due to hard real-time deadline. Otherwise, it'll get destroyed.

Flexibility:

It's building systems with built-in debugging opportunities which allows remote maintenance.

For example, you are building a spacecraft which will land on another planter to collect various types of data and send collected detail back to us. If this spacecraft went insane and lost the control, we should be able to make some important diagnostic. So, flexibility is vital while designing an embedded system.

Portability:

Portability is a measure of the ease of using the same embedded software in various environments. It requires generalized abstractions between the application program logic itself and the low-level system interfaces.

1.6.Types of Embedded Systems

1. Real-time embedded systems:
These systems are designed to respond to events as they occur with minimal latency. Examples include autopilot systems, brake control systems, robotics, and medical devices.

2. Networked embedded systems:
These systems are designed to communicate with other systems over a network, such as wireless or wired networks. Examples include home automation systems, industrial control systems, and medical monitoring systems.

3. Standalone embedded systems:
These systems are designed to operate independently of other systems. Examples include digital clocks, digital signage systems, and vehicle diagnostic systems.

4. Mobile embedded systems:
These systems are designed to be portable or mobile. Examples include smartphones, portable gaming devices, and wearables.

5. Autonomous embedded systems:
These systems are designed to operate autonomously and make decisions without user input. Examples include autonomous vehicles, drones, and robots.

1.7.Advantages of Embedded Systems

1. Low power consumption: Embedded systems consume very little power and are very energy efficient, which makes them ideal for applications where power is limited, or where the device needs to run off a battery.

2. Cost-effectiveness: Embedded systems are usually cheaper than general purpose computers because they are designed to perform specific tasks.

3. Highly reliable: Embedded systems are designed to be highly reliable and require less maintenance than general purpose computers.

4. Compact size: Embedded systems are usually very small and lightweight, making them ideal for applications where space is limited.

5. Customizable: Embedded systems can be designed to meet the specific needs of a particular application, making them very versatile and customizable.

6. Flexible: Embedded systems can be easily updated and modified to meet changing requirements.

7. Real-time operation: Embedded systems are designed to respond to events in real-time, making them ideal for applications where time is of the essence.

1.8.Disadvantages of Embedded Systems

1. Limited resources: Embedded systems usually have limited resources such as memory, power, and processing power. This can make it difficult to add features or upgrade the system.

2.Security issues: Due to the limited resources and closed nature of embedded systems, they can be vulnerable to security threats such as viruses, malware, and hackers.

3.Dependency on hardware: Embedded systems are highly dependent on the hardware they are running on. If the hardware fails, the system will be rendered useless.

4.Cost: Embedded systems can be expensive to develop and maintain. The cost of developing an embedded system can be higher than developing a software-only solution.

1.9.Applications of Embedded Systems

1. Automotive: Embedded systems are used in vehicles for controlling various components such as engine, brakes, transmission, airbags, etc.

2. Telecommunications: Embedded systems are used in mobile phones for various functions such as display, audio processing, radio communication, etc.

3. Medical: Embedded systems are used in medical equipment such as scanners, imaging systems, and other monitoring systems.

4. Industrial: Embedded systems are used in industrial robots for controlling their motions and other operations.

5. Security: Embedded systems are used in security systems such as CCTV cameras, fire detectors, access control systems, etc.

6. Home Appliances: Embedded systems are used in home appliances such as refrigerators, washing machines, microwaves, etc.

2.BASIC ELECTRONICS

2.1.Basic Terms used in Electronics

1. Alternating Current (AC): an electric current that reverses direction periodically.
2. Ammeter: an instrument used to measure the electric current in amperes.
3. Amplifier: an electronic device used to increase the magnitude of a signal.
4. Capacitor: an electronic component consisting of two conducting plates separated by an insulating layer.
5. Circuit: a closed path through which electric current can flow.
6. Diode: an electronic component used to control the direction of electric current.
7. Ground: the reference point in a circuit for measuring voltage or current.
8. Inductor: an electronic component consisting of a coil of wire that stores energy in a magnetic field.
9. Ohm: the unit of measure for electrical resistance.
10. Resistor: an electronic component used to reduce the flow of electric current in a circuit.

2.2.Analog and Digital Signals

Analog and digital signals are two types of electrical signals used to transmit information.

An analog signal is a continuous signal that varies in amplitude, frequency, or phase over time. Analog signals are used to represent physical quantities that are continuous, such as sound, temperature, and pressure. An example of an analog signal is the sound waves produced by a person's voice.

A digital signal, on the other hand, is a discrete signal that consists of a sequence of 0s and 1s, called bits. Digital signals are used to represent information that can be quantized, such as numbers, text, and images. An example of a digital signal is the binary code used by computers to represent data.

Some differences between analog and digital signals are:

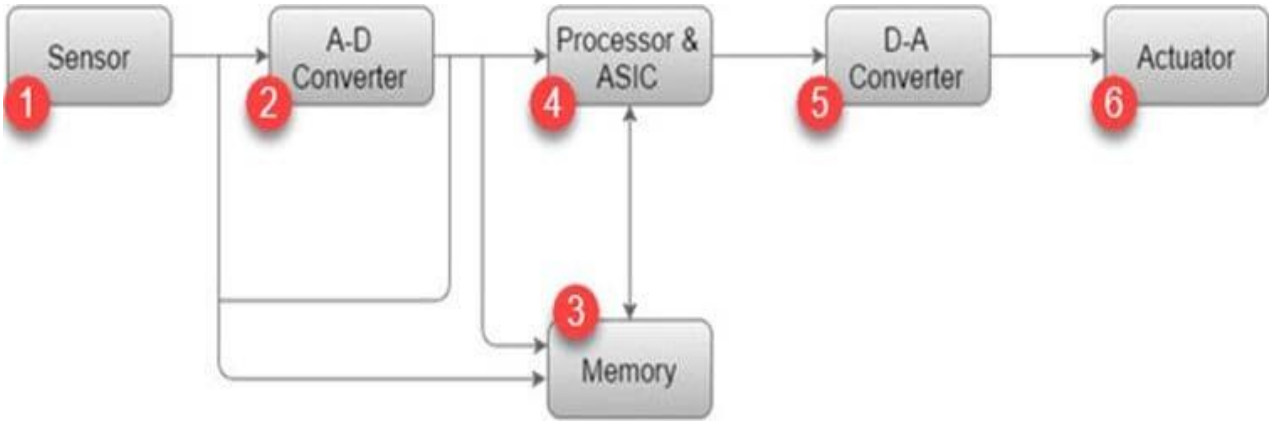
1.Representation: Analog signals represent continuous, real-world quantities, while digital signals represent discrete values.

2.Noise: Analog signals are susceptible to noise and distortion during transmission, which can affect the accuracy of the signal. Digital signals are less susceptible to noise and distortion, and can be transmitted over longer distances without losing information.

3.Bandwidth: Analog signals require a larger bandwidth to transmit information, as they are continuous and require a larger range of frequencies. Digital signals require less bandwidth, as they consist of discrete values and can be encoded using a smaller range of frequencies.

5. Processing: Analog signals require specialized processing techniques, such as amplification and filtering, to extract information from the signal. Digital signals can be processed using standard digital circuits and algorithms.

2.3.Embedded Systems Architecture



1) Sensor:

Sensor helps you to measures the physical quantity and converts it to an electrical signal. It also stores the measured quantity to the memory. This signal can be ready by an observer or by any electronic instrument such as A2D converter.

2) A-D Converter:

A-D converter (analog-to-digital converter) allows you to convert an analog signal sent by the sensor into a digital signal.

3) Memory:

Memory is used to store information. Embedded System majorly contains two memory cells 1) Volatile 2) Non volatile memory.

4) Processor & ASICs:

This component processes the data to measure the output and store it to the memory.

5) D-A Converter:

D-A converter (A digital-to-analog converter) helps you to convert the digital data fed by the processor to analog data.

6) Actuator:

An actuator allows you to compare the output given by the D-A converter to the actual output stored in it and stores the approved output in the memory.

2.4.Instruction Set

Instruction sets are a collection of instructions used by computers to perform specific operations.
Examples of instruction sets include:
1. Arithmetic and Logical Instructions: These instructions are used to perform arithmetic and logical operations such as addition, subtraction, multiplication, division, bitwise operations, etc. Example: ADD, SUB, MUL, DIV, AND, OR, XOR.

2. Data Movement Instructions: These instructions are used to move data from one location to another. Examples: MOV, MOVS, MOVB, LEA, PUSH, POP.

3. Program Control Instructions: These instructions are used to control the flow of a program. Examples: JMP, CALL, RET, INT, HLT.

4. I/O Instructions: These instructions are used to perform input and output operations. Examples: IN, OUT, INB, OUTB.

5. x86 Instruction Set: The x86 instruction set is used by Intel and AMD processors in most desktop and laptop computers. Some examples of x86 instructions include ADD (addition), MOV (move), and JMP (jump).

6. ARM Instruction Set: The ARM instruction set is used by many mobile devices and embedded systems. Some examples of ARM instructions include ADD (addition), MOV (move), and BX (branch and exchange).

7. MIPS Instruction Set: The MIPS instruction set is used in some embedded systems and academic research. Some examples of MIPS instructions include ADD (addition), LW (load word), and JAL (jump and link).

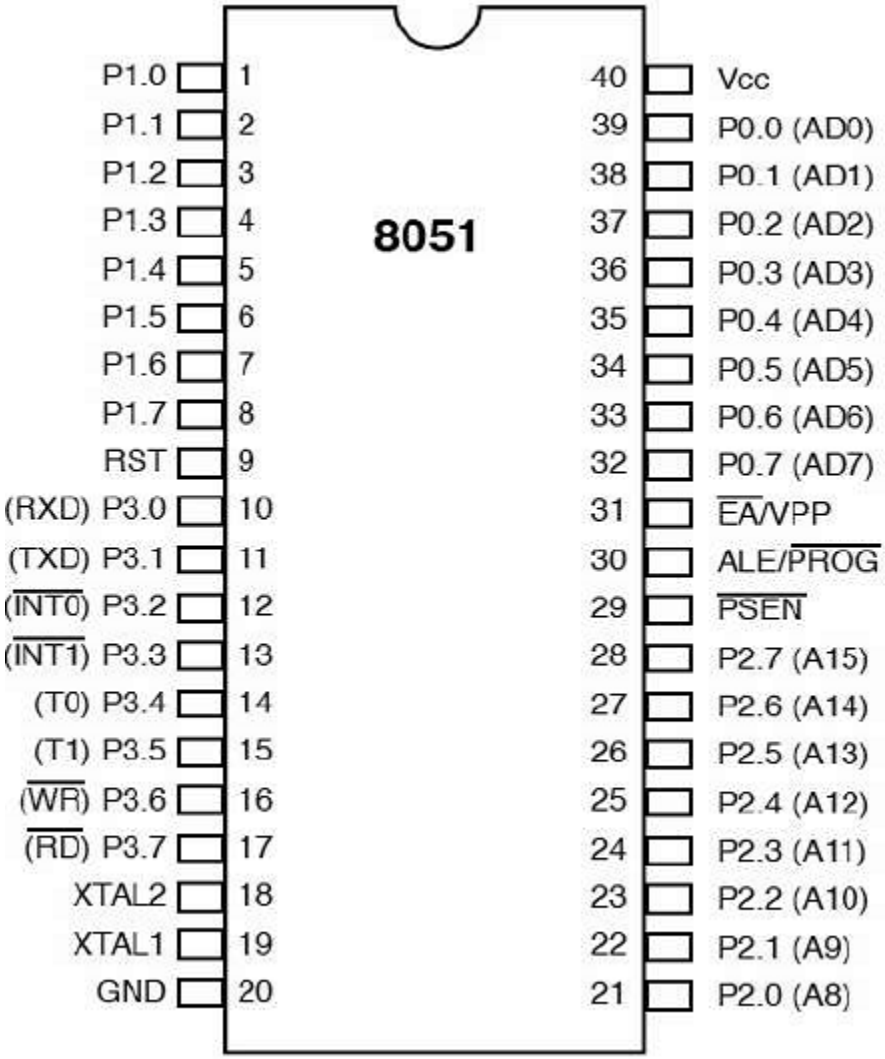
8. PowerPC Instruction Set: The PowerPC instruction set is used by some desktop and server computers, as well as some embedded systems. Some examples of PowerPC

9. instructions include ADD (addition), LWZ (load word and zero), and B (branch).

10. SPARC Instruction Set: The SPARC instruction set is used in some servers and workstations. Some examples of SPARC instructions include ADD (addition), LD (load), and BNE (branch if not equal).

3.MICROCONTROLLER

3.1.Introduction to 8051(The first Microcontroller)



1.The 8051 microcontroller is one of the earliest and most widely used microcontrollers in the industry. It was developed by Intel in the 1980s and was first introduced in 1981. The 8051 microcontroller has a Harvard architecture, which means that it has separate memory spaces for program instructions and data.

2.The 8051 microcontroller has 4KB of on-chip ROM for program storage and 128 bytes of on-chip RAM for data storage. It also has 32 input/output (I/O) pins that can be programmed as either inputs or outputs. The 8051 microcontroller is known for its simple and efficient instruction set, which makes it easy to program and use.

3.The 8051 microcontroller is used in a wide range of applications, including industrial control systems, automotive systems, medical devices, and consumer electronics. It has been used in everything from washing machines and microwave ovens to automotive fuel injection systems and satellite communication systems.

4.One of the advantages of the 8051 microcontroller is its low power consumption, which makes it ideal for battery-powered applications. It also has a low cost, which makes it an attractive option for small-scale projects and hobbyists.

3.2.Different types of Microcontroller

1. 8-bit Microcontrollers: 8-bit microcontrollers are used in many embedded systems, such as automotive, consumer electronics, medical, and industrial applications. They are typically used in applications that require low power consumption and/or a small form factor. Some common applications include motor control, digital signal processing, and data acquisition.

2. 16-bit Microcontrollers: 16-bit microcontrollers are used in applications that require more power and performance than a 8-bit microcontroller can provide. They are commonly used in industrial automation, robotics, and medical equipment.

3. 32-bit Microcontrollers: 32-bit microcontrollers are used for more complex applications that require higher precision and more processing power than a 16-bit microcontroller can provide. They are typically used in automotive, consumer, industrial, and medical applications.

4. Flash Microcontrollers: Flash microcontrollers are used in applications that require fast data read/write capabilities. They are often used in applications such as automotive, consumer, and industrial applications.

6. Digital Signal Controllers: Digital signal controllers are used in applications that require more precise control and processing of analog signals. They are typically used in industrial automation, robotics, and medical applications.

7. ARM-based Microcontrollers: ARM-based microcontrollers are widely used in embedded systems due to their low power consumption, high performance, and wide range of features. They are commonly used in mobile devices, automotive systems, and industrial control systems.

8. PIC Microcontrollers: PIC microcontrollers are a popular type of microcontroller developed by Microchip Technology. They are known for their low cost, low power consumption, and ease of use.

9. AVR Microcontrollers: AVR microcontrollers are another popular type of microcontroller developed by Atmel Corporation. They are known for their high performance, low power consumption, and large number of features.

3.3.Internal Main Components of Microcontroller

1.CPU (Central Processing Unit): The CPU is the brain of the microcontroller. It executes instructions and performs arithmetic and logical operations.

2.Memory: The microcontroller has two types of memory - program memory (ROM or flash memory) and data memory (RAM). Program memory stores the program code that the CPU executes, while data memory stores variables and temporary data used by the program.

3.Input/Output Ports (I/O Ports): The microcontroller has a number of I/O ports, which are used to connect to external devices such as sensors, actuators, and other peripheral devices.

4.Timers and Counters: These are used to measure time intervals and count external events. They are commonly used in applications such as motor control and communication protocols.

5.Analog-to-Digital Converter (ADC): The ADC converts analog signals (such as temperature or voltage) into digital signals that can be processed by the microcontroller.

6.Serial Communication Interface: This allows the microcontroller to communicate with other devices using serial communication protocols such as UART, SPI, or I2C.

7.Watchdog Timer: This is a safety feature that resets the microcontroller if the software hangs or crashes.

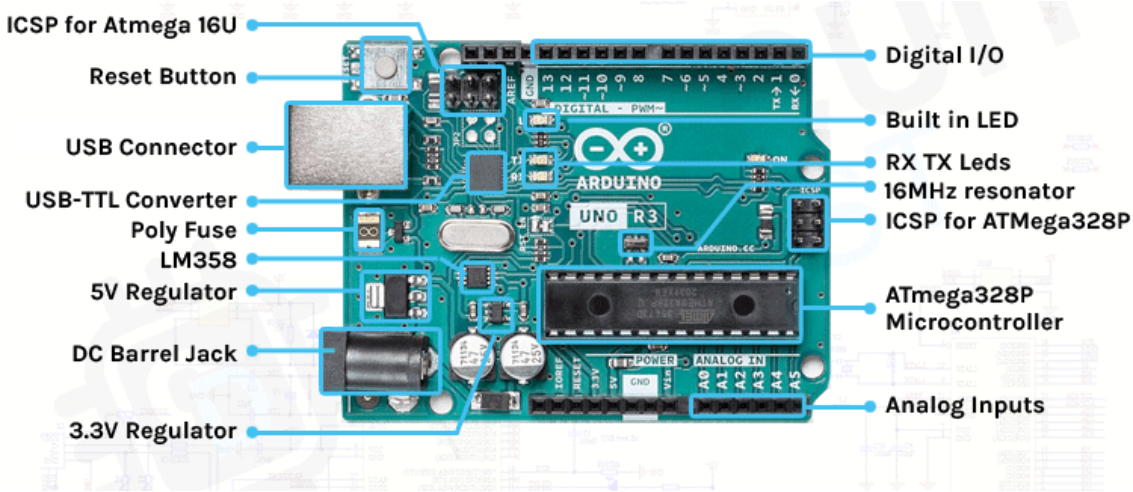
8.Interrupts: Interrupts are used to halt the normal execution of the program and execute a specific routine when an external event occurs.

9.Power Management: Microcontrollers have built-in power management features that allow them to operate efficiently and conserve power.

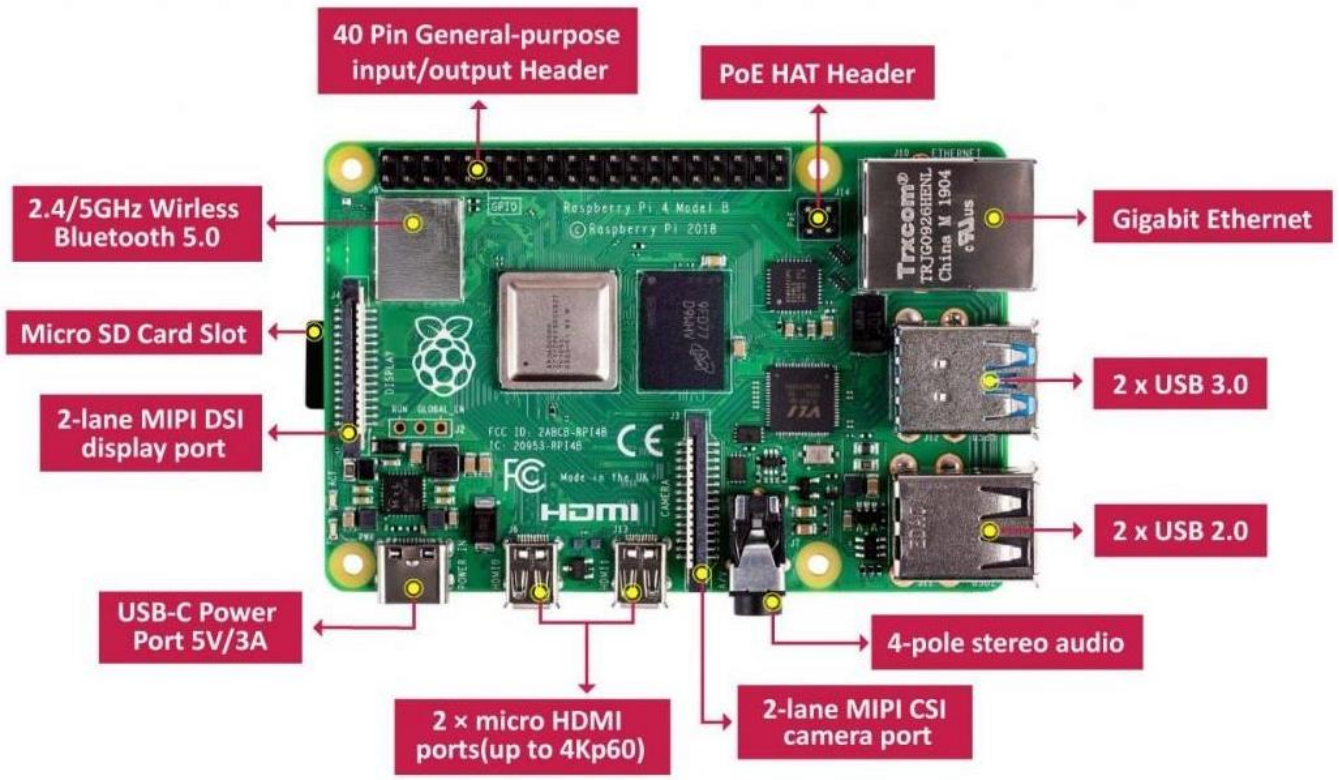
3.4.Development Boards

1. **Arduino:**

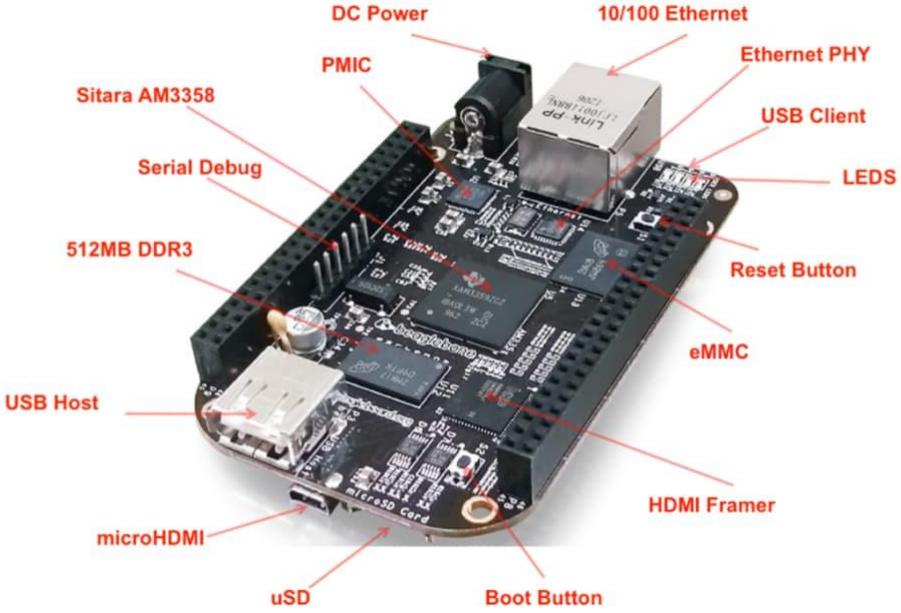
Arduino is a popular open-source development platform that is based on the Atmel AVR microcontroller. Arduino boards are easy to use and come with a range of shields (plug-in boards) that can be used to add functionality.



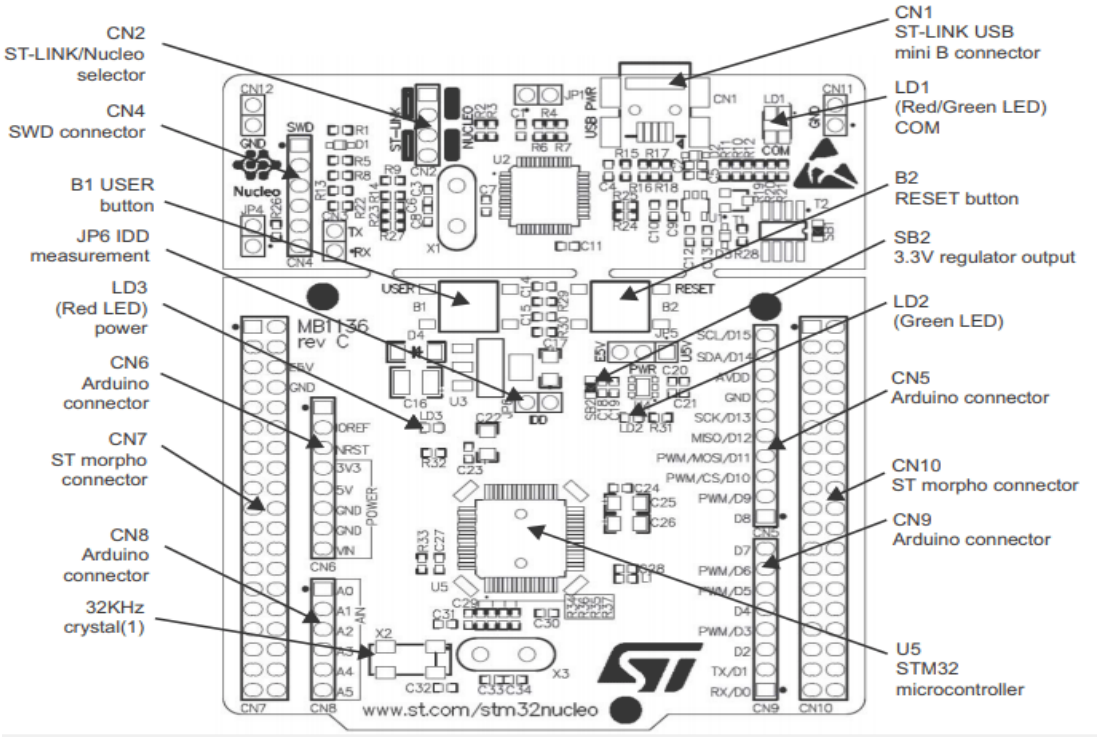
2. **Raspberry Pi:** Raspberry Pi is a low-cost, credit-card-sized computer that can be used for a wide range of applications, including web browsing, gaming, and media playback.



3.BeagleBone Black: BeagleBone Black is a low-cost, open-source development board that is based on the Texas Instruments Sitara AM335x ARM Cortex-A8 processor.



4.STM32 Nucleo: STM32 Nucleo is a low-cost development board that is based on the ARM Cortex-M microcontroller. It comes with a range of expansion boards that can be used to add functionality.

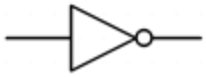


4.ARDUINO(ATMEGA328)

4.1.Basics Of Digital Circuits

Digital circuits are electronic circuits that operate on digital signals, which are binary signals that can have only two possible states - 0 or 1. These circuits are used in a wide range of applications, from simple logic gates to complex microprocessors.

The basic building blocks of digital circuits are logic gates, which are electronic circuits that perform a specific logical function based on the input signals.
Some of the common logic gates include:

Logic Gate	Symbol	Description
AND		Output is at logic 1 when, and only when all its inputs are at logic 1,otherwise the output is at logic 0.
OR		Output is at logic 1 when one or more are at logic 1.If all inputs are at logic 0,output is at logic 0.
NAND		Output is at logic 0 when,and only when all its inputs are at logic 1,otherwise the output is at logic 1
NOR		Output is at logic 0 when one or more of its inputs are at logic 1.If all the inputs are at logic 0,the output is at logic 1.
XOR		Output is at logic 1 when one and Only one of its inputs is at logic 1. Otherwise is it logic 0.
XNOR		Output is at logic 0 when one and only one of its inputs is at logic1.Otherwise it is logic 1. Similar to XOR but inverted.
NOT		Output is at logic 0 when its only input is at logic 1, and at logic 1 when its only input is at logic 0.That's why it is called and INVERTER

1.AND gate: The AND gate produces an output signal only when both the input signals are high (1). The symbol for an AND gate is a triangle with a small circle on the output side.

2.OR gate: The OR gate produces an output signal when either one or both of the input signals are high (1). The symbol for an OR gate is a half-circle with a small circle on the output side.

3.NOT gate: The NOT gate produces an output signal that is the opposite of the input signal. If the input signal is high (1), the output signal is low (0) and vice versa. The symbol for a NOT gate is a small circle on the output side.

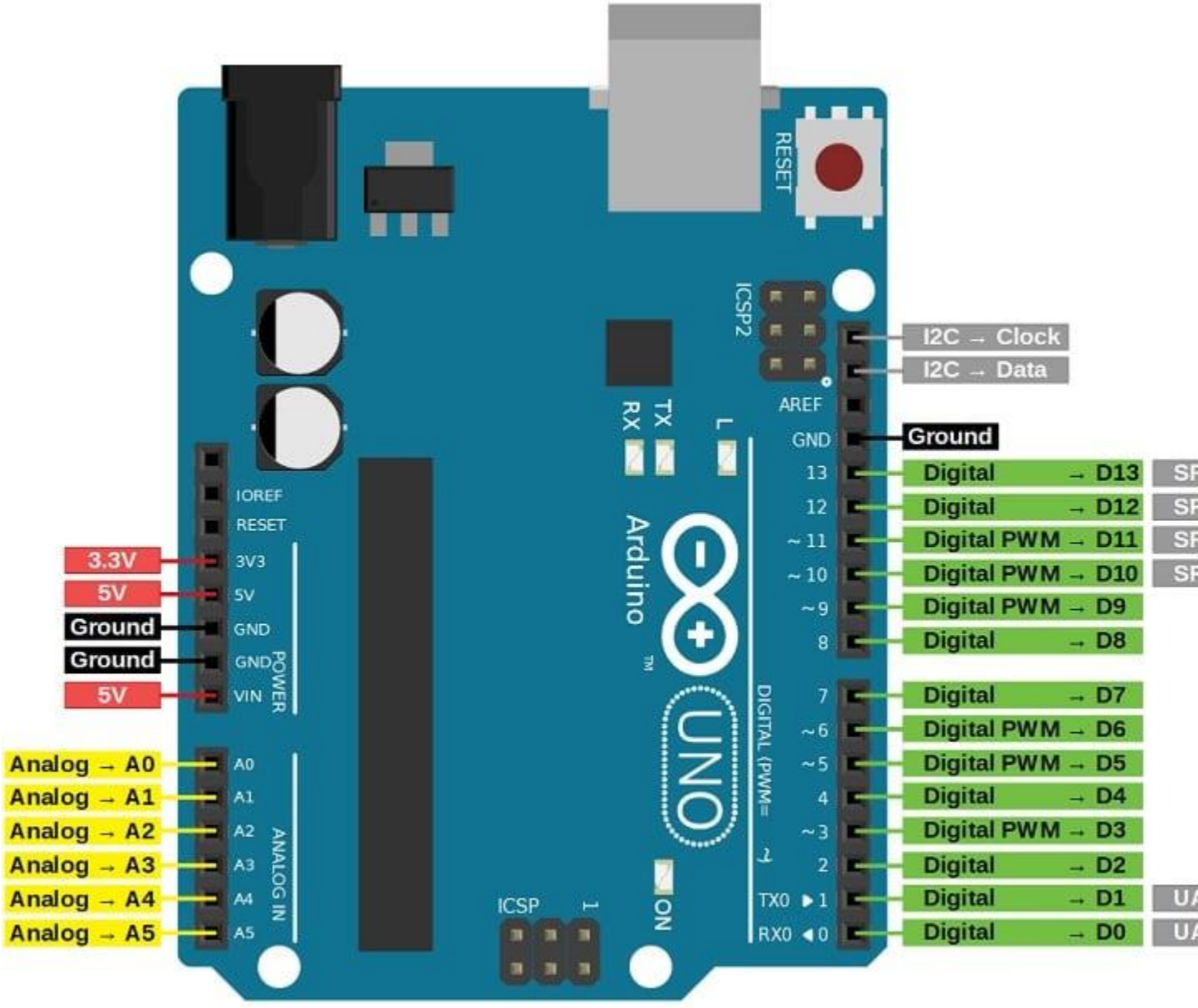
4.NAND gate: The NAND gate is a combination of the AND and NOT gates. It produces an output signal that is the opposite of the AND gate output. The symbol for a NAND gate is a triangle with a small circle on the output side and a small circle on the input side for the NOT operation.

5.NOR gate: The NOR gate is a combination of the OR and NOT gates. It produces an output signal that is the opposite of the OR gate output. The symbol for a NOR gate is a half-circle with a small circle on the output side and a small circle on the input side for the NOT operation.

6. **XOR gate:** The XOR gate produces an output signal when either one of the input signals is high (1), but not when both inputs are high. The symbol for an XOR gate is a triangle with a curved line on the output side.

Digital circuits are also characterized by their speed and power consumption. The speed of a digital circuit is measured in hertz (Hz), which is the number of times the circuit can switch between 0 and 1 in one second. Power consumption is measured in watts (W), which is the amount of energy the circuit consumes in one second.

4.2.Arduino Pinout



4.3.Arduino IDE

1. Arduino IDE is an open-source integrated development environment (IDE) used for writing and uploading code to Arduino boards. It is a simple and easy-to-use software that allows users to quickly start programming their Arduino projects. Arduino IDE is available for Windows, Mac, and Linux operating systems.

2.Arduino IDE supports a simplified version of C++ programming language that is easy to learn and understand even for beginners. It also includes a code editor with features such as syntax highlighting, auto-completion, and error highlighting to make programming easier and error-free.

3.Arduino IDE also includes a serial monitor that allows users to communicate with their Arduino board and view the output of their code. This feature is useful for debugging and troubleshooting their code.

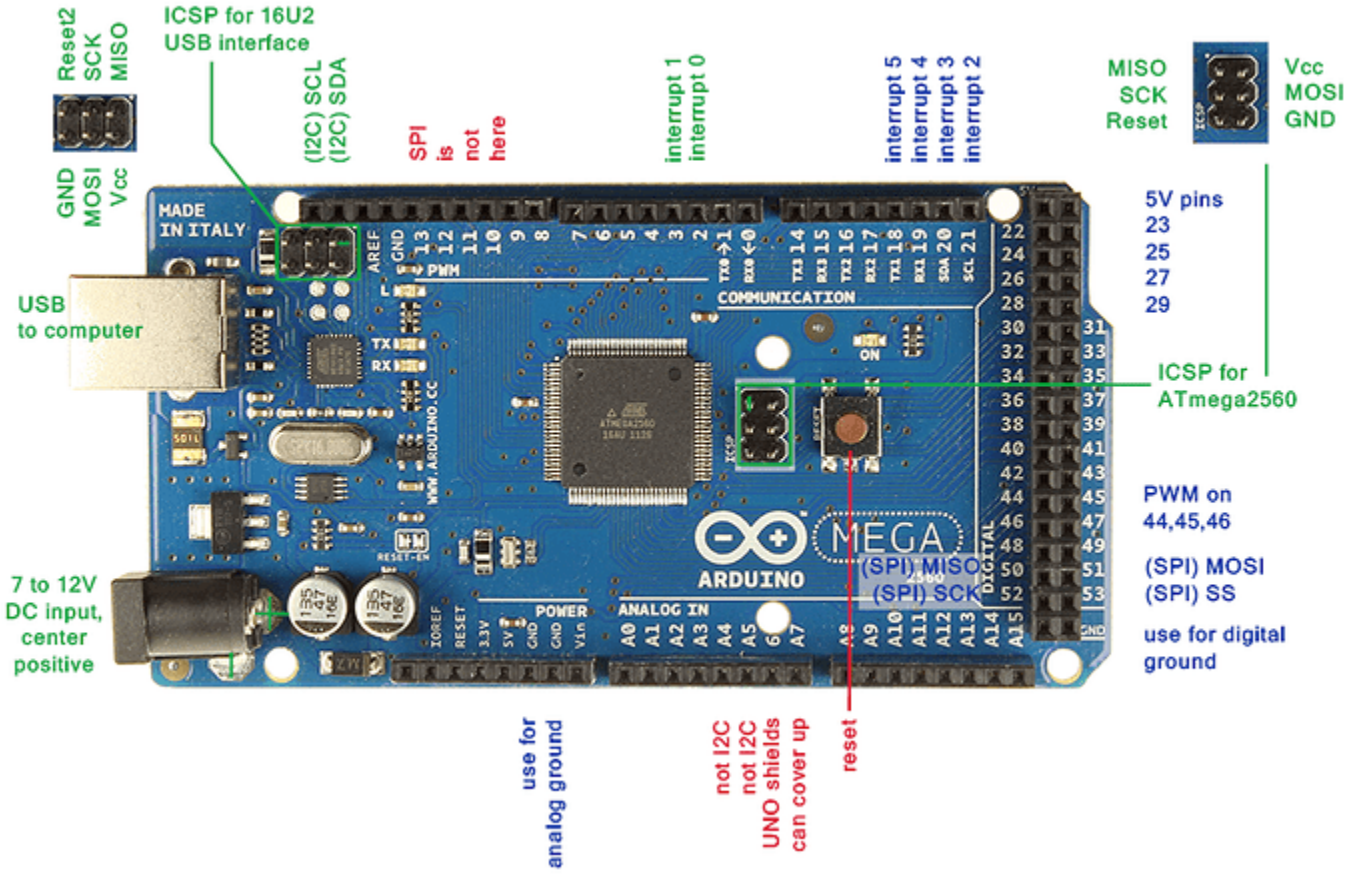
4.Another key feature of the Arduino IDE is its library manager, which makes it easy to add third-party libraries to the user's code. This allows users to easily add new functionality to their projects without having to write all the code from scratch.

5.In addition, Arduino IDE has a simple and intuitive user interface that makes it easy for users to navigate and use all its features. It is also regularly updated with new features and bug fixes to improve the user experience.

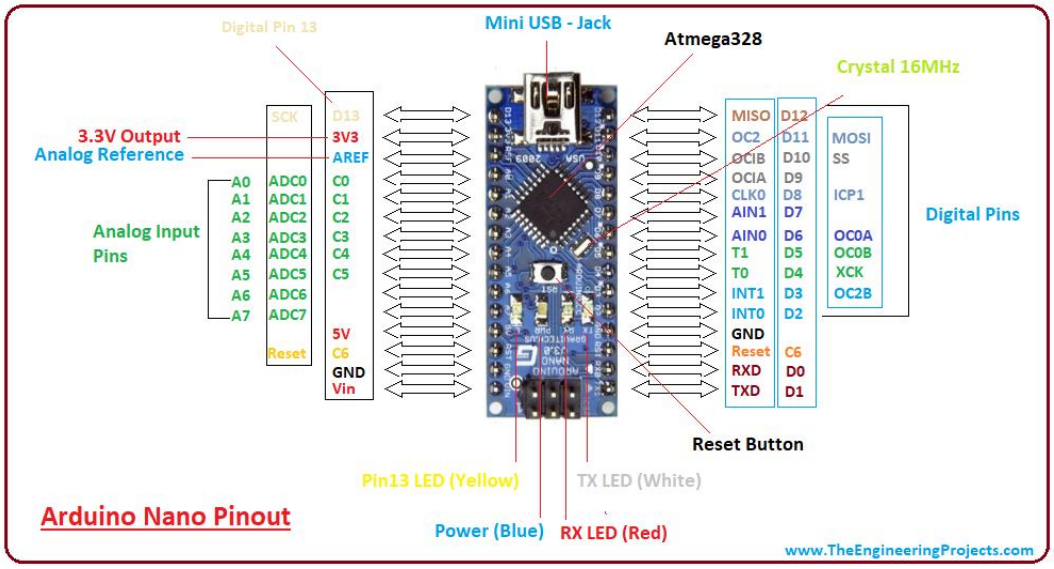
4.4.Versions of Arduino Family

1. **Arduino Uno:** This is the most popular and widely used version of the Arduino board. It is a microcontroller board based on the ATmega328P chip and features 14 digital input/output pins, 6 analog inputs, and a USB connection for programming and power.

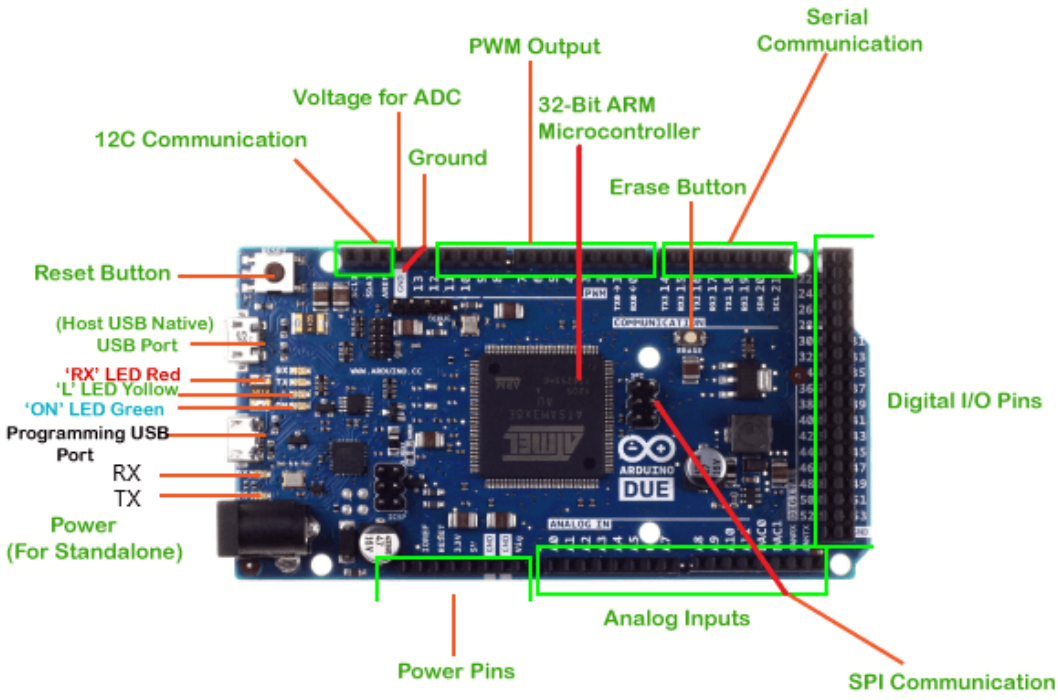
2. **Arduino Mega:** This is a more advanced version of the Arduino board that features more input/output pins than the Uno. It is based on the ATmega2560 chip and features 54 digital input/output pins, 16 analog inputs, and a USB connection for programming and power.



3. **Arduino Nano:** This is a smaller and more compact version of the Arduino board, making it ideal for projects with limited space. It is based on the ATmega328P chip and features 14 digital input/output pins, 8 analog inputs, and a USB connection for programming and power.

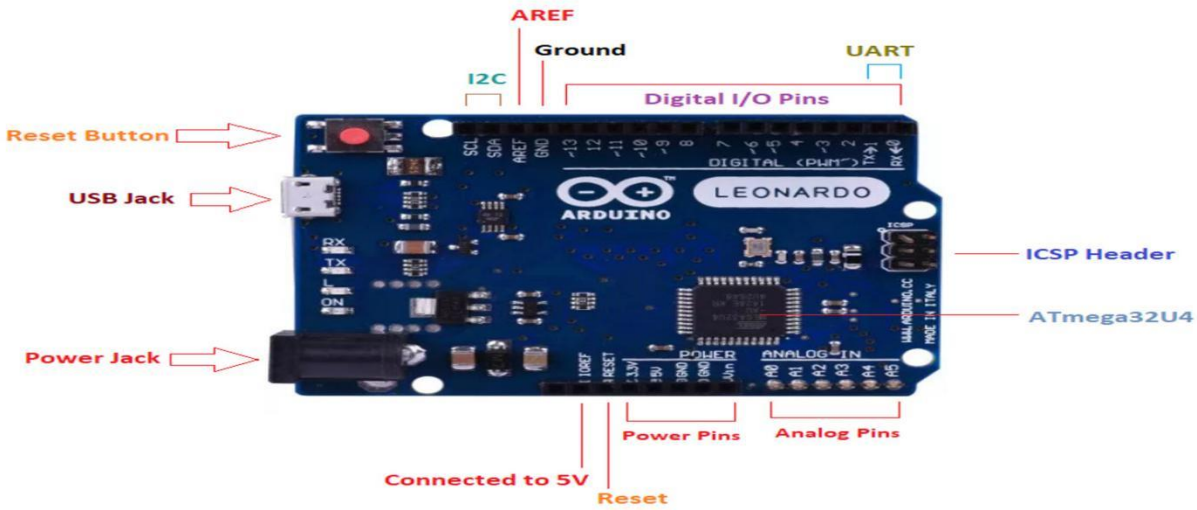


4. **Arduino Due:** This is a more powerful version of the Arduino board that is based on the ARM Cortex-M3 processor. It features 54 digital input/output pins, 12 analog inputs, and a USB connection for programming and power.



Arduino Due

5.Arduino Leonardo: This is a version of the Arduino board that features a built-in USB interface, allowing it to act as a USB keyboard or mouse. It is based on the ATmega32U4 chip and features 20 digital input/output pins, 12 analog inputs, and a USB connection for programming and power.



Arduino Leonardo

Some of Arduino Applications below:-

1.Robotics: The Arduino board can be used to control the motors and sensors of a robot, making it an ideal choice for robotics projects.

2.Home automation: The Arduino board can be used to control the lighting, heating, and other appliances in a home, making it an ideal choice for home automation projects.

3.Internet of Things (IoT): The Arduino board can be used to connect sensors and other devices to the internet, making it an ideal choice for IoT projects.

4.Education: The Arduino board is a popular choice for educational projects and is widely used in schools and universities to teach electronics and programming.

4.5.Arduino Embedded C Commands

Arduino is programmed using a simplified version of C++ programming language, which is commonly referred to as Arduino Embedded C. Arduino Embedded C is a set of commands that allow users to interact with the hardware components of their Arduino board and control their behavior.

Some of the most commonly used commands in Arduino Embedded C include:

1.pinMode(): This command is used to set the mode of a pin as either an input or an output. It takes two arguments - the pin number and the mode (INPUT or OUTPUT).

2.digitalWrite(): This command is used to set the state of an output pin to either HIGH or LOW. It takes two arguments - the pin number and the state (HIGH or LOW).

3.digitalRead(): This command is used to read the state of an input pin. It takes one argument - the pin number - and returns either HIGH or LOW.

5. analogWrite(): This command is used to output a variable voltage level to a PWM (pulse-width modulation) enabled pin. It takes two arguments - the pin number and the value (0 to 255).

5.analogRead(): This command is used to read the value of an analog input pin. It takes one argument - the pin number - and returns a value between 0 and 1023.

6.delay(): This command is used to pause the program for a specified amount of time in milliseconds. It takes one argument - the delay time in milliseconds.

7.Serial.begin(): This command is used to initialize the serial communication between the Arduino board and the computer. It takes one argument - the baud rate (bits per second).

5.C PROGRAMMING

5.1.Basic Syntax:-

1. Header-Files:-
Header files are included at the beginning of a C program and contain declarations for functions that are defined in other files. In this example, stdio.h is a header file that contains declarations for standard input/output functions, such as printf() and scanf().

Example:-#include <stdio.h>

2. Main Function:-
The main() function is the entry point for a C program. It's where the program starts executing. In this example, the function returns an integer value of 0 to indicate that the program completed successfully. The return statement is optional, and if it's omitted, the function will return 0 by default.

Example:-

```
int main() {
  // code goes here
  return 0;
}
```

3. Statements:-
Statements in C are instructions that perform a specific task. In this example, the printf() statement is used to display the string "Hello, World!" on the screen. The printf() function is part of the standard input/output library (stdio.h) and is used to print formatted output to the console.

Example:-printf("Hello, World!");

4. Comments:-
Comments in C are used to explain what the code is doing and make it more readable. Single-line comments start with // and continue until the end of the line. Multi-line comments start with /* and end with */.

Example:-

```
#include <stdio.h>

int main() {
```

```
 printf("Hello, World!");//Single Comment
 return 0;/* multiple Comments */
}
```

5.2.Variables in C

Variables in C are used to store values, such as numbers or characters.
Example:
int age = 20; // declare and initialize an integer variable

float pi = 3.14; // declare and initialize a float variable

char letter = 'A'; // declare and initialize a character variable

char name[20] = "John Doe";/*
Strings are arrays of characters that represent text. In this example, the name variable is declared as an array of char with a length of 20 and assigned the value "John Doe". Note that strings in C are null-terminated, meaning that they end with a null character ('\0').*/

const int DAYS_IN_WEEK = 7;
const float PI = 3.14;

Constants are variables that cannot be modified once they are declared. In C, constants are typically declared using the const keyword. In this example, the DAYS_IN_WEEK constant is declared as an integer with a value of 7, and the PI constant is declared as a float with a value of 3.14.

5.3.Datatypes in C:-

In C programming, data types refer to the different types of data that can be stored in a variable. Each data type specifies the size and type of data that can be stored, as well as the range of values it can hold.

1. int: an integer data type is a whole number, positive or negative, without decimals, of unlimited length.

Example: int x = 10;

It typically occupies 4 bytes of memory on most modern systems, but its size may vary depending on the platform.

The int data type can hold both positive and negative values.

1.1.short: It is a signed integer type and occupies 2 bytes of memory.
Its range is -32768 to 32767.

Example: short y = 20;

1.2.long: It is a signed integer type and occupies 8 bytes of memory.
Its range is -9223372036854775808 to 9223372036854775807.

Example: long z = 30;

1.3.unsigned int: It is an unsigned integer type and occupies 4 bytes of memory.
Its range is 0 to 4294967295.

Example: unsigned int p = 40;

1.4.Signed int:-In C programming, the signed keyword is used to indicate that a variable is capable of representing both positive and negative values. This is the default behavior for most integer data types in C, including the int data type.

Example:-signed int temperature = -10;

2.float: a floating-point number is a number with a decimal point or a number in exponential form. Example: float x = 3.14;

3.char: a character is a single letter, digit, or any other symbol. It is denoted by single quotes.
Example: char x = 'A';

4.double: a double is a double-precision, floating-point data type used to represent fractional numbers. Example: double x = 3.14159;

5.void: void is a special data type that has no value. It is used to indicate that a function does not return a value.

Example: void myFunction();

6.Boolean:-

Example:-
```
#include <stdbool.h>

bool is_true = true;
bool is_false = false;
```

Boolean data type represents a logical value of true or false. In C, the bool data type is used to represent Boolean values. To use the bool data type in C, we need to include the stdbool.h

header file. In this code snippet, we declare two bool variables is_true and is_false and assign them the values of true and false, respectively.

Note:-
1. char: 1 byte 2. int: 4 bytes 3. float: 4 bytes 4. double: 8 bytes 5. long: 8 bytes 6. short: 2 bytes 7. unsigned char: 1 byte 8. unsigned int: 4 bytes 9. unsigned long: 8 bytes 10. unsigned short: 2 bytes

5.4.Operators in C:-

Operators are symbols that are used to perform operations on operands.

operators are symbols that are used to perform various operations on operands, which can be variables, constants, or expressions.

Examples of Operators in C:

1. Arithmetic Operators: +, -, *, /, %

For example:

```
int a = 10;
int b = 5;
int c = a + b;  // c will be equal to 15
```

2. Comparison Operators: ==, !=, >, <, >=, <=

For example:

```
int a = 10;
int b = 5;

if (a > b) // true
```

3. Assignment Operators: =, +=, -=, *=, /=, %=

For example:

```
int a = 10;
int b = 5;

a += b;  // a will be equal to 15
```

4. Logical Operators: && (AND), || (OR), ! (NOT)

For example:

```
int a = 10;
int b = 5;

if (a > 5 && b > 5) // true
```

5. Ternary Operator:-

For Example:

```
int x = 10;
int y = 5;
int result;

result = (x > y) ? x : y;
```

6. Bitwise Operators

1. AND (&): This operator performs a bit-wise logical AND operation on two operands. For example,

```
int x = 10; // binary representation is 00001010
int y = 20; // binary representation is 00010100
int z = x & y; // z = 00001000 which is 8 in decimal
```

2. **OR (|):** This operator performs a bit-wise logical OR operation on two operands.

For example,

```
int x = 10; // binary representation is 00001010
int y = 20; // binary representation is 00010100
int z = x | y; // z = 00011110 which is 30 in decimal
```

3. **XOR (^):** This operator performs a bit-wise logical XOR operation on two operands.

For example,

```
int x = 10; // binary representation is 00001010
int y = 20; // binary representation is 00010100
int z = x ^ y; // z = 00010110 which is 22 in decimal
```

4. **NOT (~):** This operator performs a bit-wise logical NOT operation on a single operand.

For example, int x = 10; // binary representation is 00001010

```
int z = ~x; // z = 11110101 which is -11 in decimal
```

5. **Left Shift (<<):** This operator shifts the bits of the operand to the left by a specified number of places.
For example,
int x = 10; // binary representation is 00001010
int z = x << 2; // z = 00101000 which is 40 in decimal
6. **Right Shift (>>):** This operator shifts the bits of the operand to the right by a specified number of places.
For example, int x = 10; // binary representation is 00001010
int z = x >> 2; // z = 00000010 which is 2 in decimal

5.5.Control Flow

Control flow in C is a set of commands that dictate the order in which a program executes. This is usually done through the use of conditional statements, loops, and function calls.
1. **Conditional Statements:** Conditional statements allow a programmer to make decisions based on certain conditions.
For example, an if-else statement is used to execute a particular set of instructions if a condition is met, and execute an alternate set of instructions if the condition is not met.

Syntax: if (condition) { // execute this code if condition is true }
else { // execute this code if condition is false }

Example:-

```
if (x > y) {
  printf("x is greater than y");
} else {
  printf("y is greater than x");
}
```

2. **Loops:** Loops allow a programmer to execute a set of instructions multiple times. The two main types of loops in C are for and while loops.

For Loop Syntax: for (initialization; condition; update) { //execute code }

While Loop Syntax: while (condition) { // execute code }

Example:-

```
for (int i = 0; i < 10; i++) {
  printf("%d\n", i);
}

while (i < 10) {
  printf("%d\n", i);
  i++;
```

```
}

do {
  printf("%d\n", i);
  i++;
} while (i < 10);
```

3. **Function Calls:** Function calls allow a programmer to execute a block of code that is stored in a separate function. This is useful for reducing code duplication and making code easier to read and maintain.

Syntax: function_name(parameter1, parameter2, ...);

4. Switch Case:-
The switch statement is used to execute a block of code based on the value of a variable

Example Code:-

```
int x = 2;

switch (x) {
  case 1:
    printf("x is 1");
    break;
  case 2:
    printf("x is 2");
    break;
  case 3:
    printf("x is 3");
    break;
  default:
    printf("x is not 1, 2, or 3");
}
```

5.6.Functions In C:-

In C programming, a function is a set of instructions that perform a specific task. Functions provide a way to break a large program into smaller, more manageable pieces, which can make the code easier to understand, debug, and maintain.

Example:-

```
#include <stdio.h>
```

```
int sum(int a, int b) {
  return a + b;
}

int main() {
  int x = 3, y = 5;
  int result = sum(x, y);
  printf("The sum of %d and %d is %d", x, y, result);
  return 0;
}
```

Functions can also have no return value, in which case they are declared as void

Example:-

```
#include <stdio.h>

void greet() {
  printf("Hello, world!");
}

int main() {
  greet();
  return 0;
}
```

5.7.Arrays,Pointers,Structures,Unions In C

an array is a collection of elements of the same data type that are stored in a contiguous block of memory.

Example:-

```
#include <stdio.h>

int main() {
  int arr[5] = {1, 2, 3, 4, 5};
  printf("%d", arr[2]);
  return 0;
}
```

Arrays can also be initialized using a loop

Example:-

```
#include <stdio.h>

int main() {
```

```
 int arr[5];
 for (int i = 0; i < 5; i++) {
  arr[i] = i + 1;
 }
 printf("%d", arr[2]);
 return 0;
}
```

Arrays can also be initialized using a Functions

```
#include <stdio.h>

int sum(int arr[], int size) {
 int result = 0;
 for (int i = 0; i < size; i++) {
  result += arr[i];
 }
 return result;
}

int main() {
 int arr[5] = {1, 2, 3, 4, 5};
 int result = sum(arr, 5);
 printf("%d", result);
 return 0;
}
```

Pointers

a pointer is a variable that stores the memory address of another variable.

Example:-

```
#include <stdio.h>

int main() {
 int x = 10;
 int *ptr = &x;
 printf("%d", *ptr);
 return 0;
}
```

we define a variable called x that has the value 10. We then define a pointer variable called ptr that stores the memory address of x. We use the & operator to get the memory address of x, and then assign it to ptr using the = operator. We then print the value of x to the console using the * operator, which dereferences the pointer and gives us the value stored at the memory address it points to.

Pointers can also be used to modify the values of variables.

Example:

```
#include <stdio.h>

int main() {
  int x = 10;
  int *ptr = &x;
  *ptr = 20;
  printf("%d", x);
  return 0;
}
```

we define a variable called x that has the value 10. We then define a pointer variable called ptr that stores the memory address of x. We use the & operator to get the memory address of x, and then assign it to ptr using the = operator. We then use the * operator to dereference the pointer and assign the value 20 to the memory location it points to. Finally, we print the value of x to the console, which should now be 20.

Pointers are also used extensively in C to manipulate arrays.

Example:-

```
#include <stdio.h>

int main() {
  int arr[5] = {1, 2, 3, 4, 5};
  int *ptr = arr;
```

```
 for (int i = 0; i < 5; i++) {
  printf("%d\n", *(ptr + i));
 }
 return 0;
}
```

we define an array called arr that has 5 elements, each of which is an integer. We then define a pointer variable called ptr and assign it the memory address of the first element of the array using the = operator. We then use a for loop to iterate over the elements of the array using pointer arithmetic. The *(ptr + i) expression gives us the value stored at the memory location that is i positions away from ptr, which is equivalent to the value of the ith element of the array. We then print each element of the array to the console using the printf function.

Structures:-

a structure is a user-defined data type that groups together related data items of different types.

Example:-

```
#include <stdio.h>

struct employee {
 char name[50];
 int id;
 float salary;
};

int main() {
 struct employee e1 = {"John Doe", 1234, 5000.0};
 printf("Name: %s\n", e1.name);
 printf("ID: %d\n", e1.id);
 printf("Salary: %.2f\n", e1.salary);
 return 0;
}
```

we define a structure called employee that has three members: name, id, and salary. We then define a variable e1 of type employee and initialize its members using the curly braces syntax. We then use the printf function to print the values of the members of e1 to the console.

Structures can also be used to define arrays and pointers.

Example:-

```
#include <stdio.h>

struct point {
  int x;
  int y;
};

int main() {
  struct point arr[5] = {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}};
  struct point *ptr = arr;
  for (int i = 0; i < 5; i++) {
    printf("(%d, %d)\n", (ptr + i)->x, (ptr + i)->y);
  }
  return 0;
}
```

we define a structure called point that has two members: x and y. We then define an array arr of 5 elements, each of which is a point structure initialized with the curly braces syntax. We also define a pointer variable ptr and assign it the memory address of the first element of the array using the = operator. We then use a for loop to iterate over the elements of the array using pointer arithmetic. The (ptr + i)->x and (ptr + i)->y expressions give us the values of the x and y members of the ith element of the array, respectively. We then print each point to the console using the printf function.

Unions:-

A union in C is a special data type available in C that allows to store different data types in the same memory location. It is a derived data type in C, which allows storing of different data types in the same memory location. A union is a special data type which allows to store different data types in the same memory location.

Example:-

```
#include <stdio.h>

union my_union {
  int i;
  float f;
  char c;
};

int main() {
  union my_union u;
  u.i = 42;
```

```
 printf("Value of i: %d\n", u.i);
 u.f = 3.14;
 printf("Value of f: %f\n", u.f);
 u.c = 'A';
 printf("Value of c: %c\n", u.c);
 printf("Value of i: %d\n", u.i);
 return 0;
}
```

Strings

a string is an array of characters terminated by a null character (\0).

Example:-

```
#include <stdio.h>
#include <string.h>

int main() {
 char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
 printf("Greeting message: %s\n", greeting);
 char name[20];
 printf("Enter your name: ");
 scanf("%s", name);
 printf("Hello, %s!\n", name);
 char str1[12] = "Hello";
 char str2[12] = "World";
 strcat(str1, str2);
 printf("Concatenated string: %s\n", str1);
 return 0;
}
```

string manipulation functions available in the string.h library.

1. **strlen()** - returns the length of a string

Example:-

```
#include <stdio.h>
#include <string.h>

int main() {
  char str[] = "Hello, world!";
  int len = strlen(str);
  printf("Length of string: %d\n", len);
  return 0;
}
```

2.strcpy() - copies a string from source to destination

Example:-

```
#include <stdio.h>
#include <string.h>

int main() {
  char src[] = "Hello, world!";
  char dest[20];
  strcpy(dest, src);
  printf("Copied string: %s\n", dest);
  return 0;
}
```

3.strcat() - concatenates two strings

Example:-

```
#include <stdio.h>
#include <string.h>

int main() {
  char str1[12] = "Hello";
  char str2[12] = "World";
```

```
 strcat(str1, str2);
 printf("Concatenated string: %s\n", str1);
 return 0;
}
```

4.strcmp() - compares two strings

Example:-

```
#include <stdio.h>
#include <string.h>

int main() {
 char str1[] = "Hello";
 char str2[] = "World";
 int result = strcmp(str1, str2);
 if (result == 0) {
  printf("Strings are equal\n");
 } else if (result < 0) {
  printf("String 1 is less than string 2\n");
 } else {
  printf("String 1 is greater than string 2\n");
 }
 return 0;}
```

5. **strchr()** - finds the first occurrence of a character in a string

Example:-

```
#include <stdio.h>
#include <string.h>

int main() {
 char str[] = "Hello, world!";
 char *ptr = strchr(str, 'o');
 if (ptr != NULL) {
  printf("Found '%c' at position %ld\n", *ptr, ptr - str);
 } else {
  printf("Character not found\n");
 }
 return 0;
}
```

5.8.Preprocessor Directives

preprocessor directives are commands that are processed by the preprocessor before the compilation of the source code. They are used to give instructions to the preprocessor to

perform specific tasks such as including header files, defining macros, and conditional compilation.

1. **#include** - includes a header file

Example:-

```
#include <stdio.h>
int main() {
  printf("Hello, world!\n");
  return 0;
}
```

2. **#define** - defines a macro

Example:-

```
#include <stdio.h>
#define PI 3.14159
int main() {
  double radius = 5.0;
  double area = PI * radius * radius;
  printf("Area of circle: %f\n", area);
  return 0;
}
```

3. **#ifdef, #ifndef, #else, #endif** - conditional compilation

Example:-

```
#include <stdio.h>
#define DEBUG
int main() {
#ifdef DEBUG
  printf("Debugging information:\n");
#endif
  printf("Hello, world!\n");
  return 0;
}
```

4. **#undef** - undefines a macro

Example:-

```
#include <stdio.h>
#define PI 3.14159
int main() {
  double radius = 5.0;
```

```
  double area = PI * radius * radius;
  printf("Area of circle: %f\n", area);
#undef PI
#define PI 3.14
  double circumference = 2 * PI * radius;
  printf("Circumference of circle: %f\n", circumference);
  return 0;
}
```

5. **#pragma** - provides implementation-defined behavior

Example:-

```
#include <stdio.h>
#pragma startup my_init
#pragma exit my_exit
void my_init() {
  printf("Initialization code\n");
}
void my_exit() {
  printf("Cleanup code\n");
}
int main() {
  printf("Hello, world!\n");
  return 0;
}
```

5.9.I/O File Operations

Input and output (I/O) operations are an important part of programming, and C programming provides several functions for performing I/O operations on files.

1. Writing to a file using fprintf()

Example:-

```
#include <stdio.h>
int main() {
  FILE *fp;
  fp = fopen("file.txt", "w");
  fprintf(fp, "This is a test file.\n");
  fclose(fp);
  return 0;
}
```

2. Reading from a file using fscanf()

Example:-

```
#include <stdio.h>
int main() {
  FILE *fp;
  char str[100];
  fp = fopen("file.txt", "r");
  fscanf(fp, "%s", str);
  printf("Read string: %s\n", str);
  fclose(fp);
  return 0;
}
```

3. Appending to a file using fputs()

Example:-

```
#include <stdio.h>
int main() {
  FILE *fp;
  fp = fopen("file.txt", "a");
  fputs("This is another line.\n", fp);
  fclose(fp);
  return 0;
}
```

4. Reading and writing binary data using fread() and fwrite()

Example:-

```
#include <stdio.h>
typedef struct {
  int id;
  char name[20];
  double score;
} Student;
int main() {
  FILE *fp;
  Student s1 = {1, "Alice", 85.5}, s2 = {2, "Bob", 92.0};
  fp = fopen("students.dat", "wb");
```

```
 fwrite(&s1, sizeof(Student), 1, fp);
 fwrite(&s2, sizeof(Student), 1, fp);
 fclose(fp);
 fp = fopen("students.dat", "rb");
 fread(&s1, sizeof(Student), 1, fp);
 fread(&s2, sizeof(Student), 1, fp);
 printf("Student 1: id=%d, name=%s, score=%.1f\n", s1.id, s1.name, s1.score);
 printf("Student 2: id=%d, name=%s, score=%.1f\n", s2.id, s2.name, s2.score);
 fclose(fp);
 return 0;
}
```

5.10.Dynamic Memory Allocation

Dynamic memory allocation in C allows for the allocation and deallocation of memory during runtime. This is accomplished using the following functions: malloc(), calloc(), realloc(), and free().

1. Malloc() Function:-

Example:-

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int *ptr;
 ptr = (int*) malloc(5 * sizeof(int));
 if (ptr == NULL) {
```

```
  printf("Memory allocation failed.\n");
  return 0;
 }
 for (int i = 0; i < 5; i++) {
  ptr[i] = i + 1;
 }
 for (int i = 0; i < 5; i++) {
  printf("%d ", ptr[i]);
 }
 printf("\n");
 free(ptr);
 return 0;
}
```

2. Calloc() Function:-

Example:-

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int *ptr;
 ptr = (int*) calloc(5, sizeof(int));
 if (ptr == NULL) {
  printf("Memory allocation failed.\n");
  return 0;}
 for (int i = 0; i < 5; i++) {
  ptr[i] = i + 1;
 }
 for (int i = 0; i < 5; i++) {
  printf("%d ", ptr[i]);
 }
 printf("\n");
 free(ptr);  return 0;}
```

3. Realloc() Function

Example:-

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int *ptr;
 ptr = (int*) malloc(5 * sizeof(int));
 if (ptr == NULL) {
  printf("Memory allocation failed.\n");
  return 0;
```

```
 }
 for (int i = 0; i < 5; i++) {
  ptr[i] = i + 1;
 }
 for (int i = 0; i < 5; i++) {
  printf("%d ", ptr[i]);
 }
 printf("\n");
 ptr = (int*) realloc(ptr, 10 * sizeof(int));
 if (ptr == NULL) {
  printf("Memory allocation failed.\n");
  return 0;
 }
 for (int i = 5; i < 10; i++) {
  ptr[i] = i + 1;
 }
 for (int i = 0; i < 10; i++) {
  printf("%d ", ptr[i]);
 }
 printf("\n");
 free(ptr);
 return 0;
}
```

4. Free() Function

Example:-

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int *ptr;
 ptr = (int*) malloc(5 * sizeof(int));
 if (ptr == NULL) {
```

```
  printf("Memory allocation failed.\n");
  return 0;
 }
 for (int i = 0; i < 5; i++) {
  ptr[i] = i + 1;
 }
 for (int i = 0; i < 5; i++) {
  printf("%d ", ptr[i]);
 }
 printf("\n");
 free(ptr);
 return 0;
}
```

6.SERIAL COMMUNICATION PROTOCOLS

6.1.Serial Communication

Serial communication is a method of sending and receiving data one bit at a time over a communication channel. It is a common method of communication in embedded systems, where devices are often connected via serial ports.

In serial communication, data is transmitted as a series of 0s and 1s, also known as bits. The data is transmitted one bit at a time, and the receiving device must be able to synchronize with the transmitting device in order to correctly interpret the data.

There are two main types of serial communication: synchronous and asynchronous.

Synchronous communication uses a clock signal to synchronize the transmitting and receiving devices. The clock signal is generated by the transmitting device and is used by the receiving device to synchronize the receipt of the data. Synchronous communication is generally faster and more reliable than asynchronous communication, but it requires more hardware and is more complex to implement.

Asynchronous communication does not use a clock signal to synchronize the transmitting and receiving devices. Instead, each byte of data is transmitted with a start bit, a specified number of data bits, and an optional parity bit and stop bit. The receiving device uses the start and stop bits to determine the beginning and end of each byte of data, and the number of data bits and parity bit to verify the correctness of the data. Asynchronous communication is simpler to implement than synchronous communication, but it is slower and less reliable.

6.2.Communication channels in telecommunications

1.Simplex:- communication is a one-way communication channel where data flows in only one direction. In other words, one device is the transmitter and the other device is the receiver. An example of simplex communication is a radio broadcast, where a radio station broadcasts a signal that can be received by many radios but cannot be sent back to the station.

2.Half-duplex communication is a two-way communication channel where data can flow in both directions, but not simultaneously. In other words, each device can both transmit and receive data, but not at the same time. A common example of half-duplex communication is a walkie-talkie, where one person talks and the other person listens, and then the roles are reversed.

3.Full-duplex communication is a two-way communication channel where data can flow in both directions simultaneously. In other words, each device can both transmit and receive data at the same time. A common example of full-duplex communication is a telephone conversation, where both parties can talk and listen at the same time.

6.3.Baud Rate

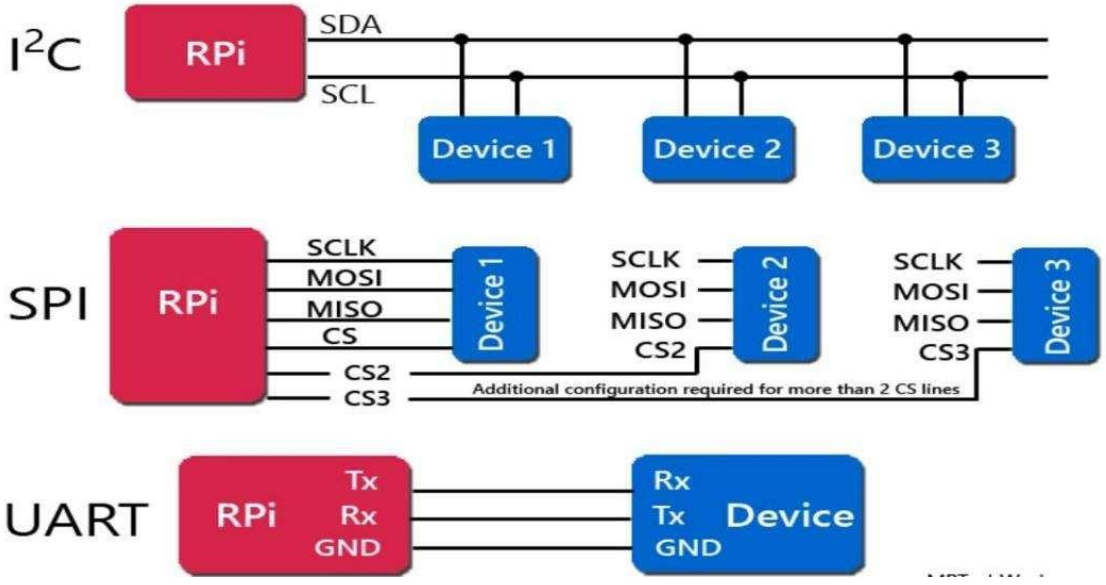
Baud rate refers to the speed at which data is transmitted over a communication channel in digital communications. It is typically measured in bits per second (bps) and represents the number of times per second that the signal on the communication channel changes.

The baud rate determines the maximum data transfer rate that can be achieved over a particular communication channel. For example, if the baud rate is 9600 bps, then data can be transmitted at a maximum rate of 9600 bits per second.

The baud rate is important because it determines the speed and efficiency of data transfer in a digital communication system. The higher the baud rate, the faster data can be transmitted, which can be important in applications such as video streaming, real-time monitoring, and industrial control systems.

It's important to note that baud rate is not the same as bit rate, although the two terms are often used interchangeably. Bit rate refers to the actual number of bits that are transmitted per second, while baud rate refers to the number of times the signal changes per second. In some cases, the bit rate may be equal to the baud rate (as in the case of simple binary signaling), but in other cases, the bit rate may be higher or lower than the baud rate (as in the case of more complex modulation schemes such as QAM or PSK).

6.4.Serial Communication Protocols:- UART,I2C,SPI



UART:-

UART (Universal Asynchronous Receiver Transmitter) is a communication protocol used for transmitting and receiving serial data between two devices. It is commonly used in embedded systems, microcontrollers, and other low-level communication applications.

The working procedure of UART protocol is as follows:

The transmitter sends the start bit to indicate the beginning of the data frame.
The data bits are then transmitted, one by one, starting with the least significant bit.
The parity bit (if used) is transmitted after the data bits to ensure data integrity.
The stop bit is then transmitted to signal the end of the data frame.

Advantages of UART protocol:

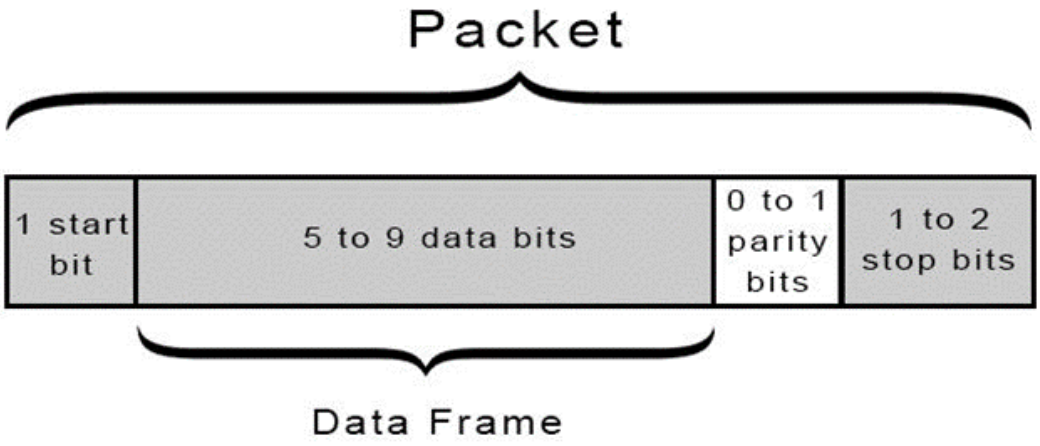
It is simple and widely used in microcontroller applications.
It is asynchronous, which means that devices with different clocks can communicate with each other.
It is a low-level protocol that can be easily implemented in hardware.

Disadvantages of UART protocol:

It is slow compared to other communication protocols such as SPI and I2C.
It requires a dedicated data line, which limits the number of devices that can be connected to a single UART interface.

Example of UART communication:

Consider two devices, a microcontroller and a computer, that need to communicate with each other using UART protocol. The microcontroller sends data to the computer by transmitting a data frame that consists of a start bit, 8 data bits, a parity bit, and a stop bit. The computer receives the data frame and processes the data. Similarly, the computer can send data to the microcontroller by transmitting a data frame in the same format. This enables bi-directional communication between the two devices using UART protocol.



1.Start Bit: A logic low signal that indicates the start of the packet.

2.Data Bits: The actual data being transmitted between the devices. This can be any number of bits, typically ranging from 5 to 9 bits, depending on the specific application.

3.Parity Bit (optional): An additional bit that is used for error checking. The parity bit can be set to odd, even, or none, depending on the specific application.

5. Stop Bit(s): One or more logic high signals that indicate the end of the packet. The number of stop bits can be 1, 1.5, or 2, depending on the specific application.

SPI:-

The Serial Peripheral Interface (SPI) is a synchronous serial communication protocol that is commonly used for short-distance communications between microcontrollers, sensors, and other digital devices.

Working Procedure:

The SPI protocol uses four wires to communicate between devices: a master output/slave input (MOSI) wire, a master input/slave output (MISO) wire, a clock (SCK) wire, and a slave select (SS) wire.
The master device initiates the communication by pulling the SS wire low to select the slave device.
The master device then sends data to the slave device by shifting bits out on the MOSI wire, while the slave device simultaneously shifts bits in on the MISO wire.
The clock signal on the SCK wire is used to synchronize the data transfer between the master and slave devices.
After the data transfer is complete, the master device releases the SS wire, indicating that the communication is finished.

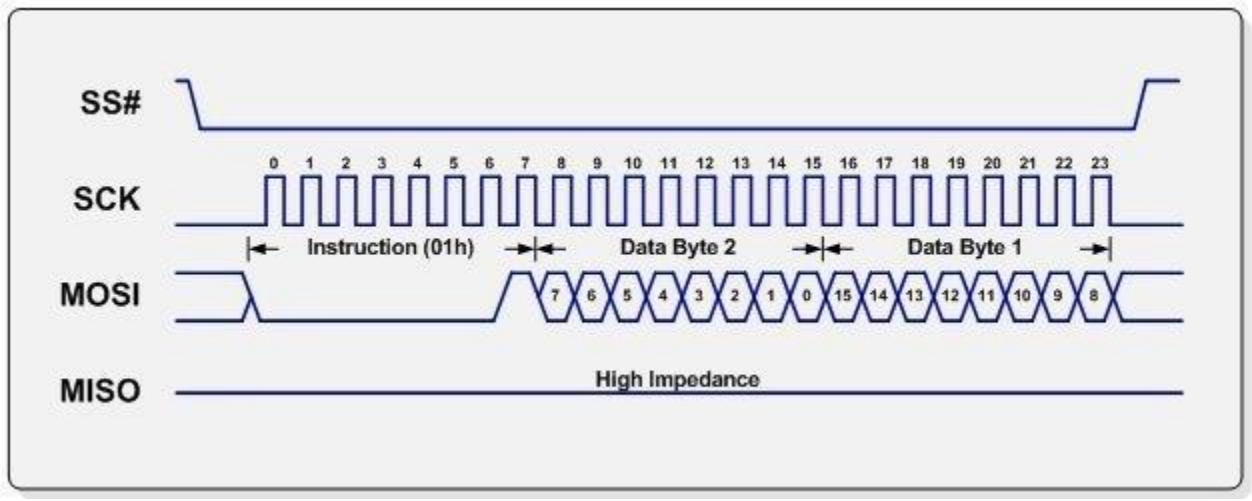
Advantages:

The SPI protocol is simple and easy to implement, requiring only a few wires and minimal overhead.
The protocol is very fast and efficient, allowing for high-speed data transfer over short distances.
The SPI protocol is widely supported by a variety of microcontrollers and other digital devices, making it a popular choice for many applications.
The protocol can be used in both full-duplex and half-duplex modes, providing flexibility in communication.

Disadvantages:

The SPI protocol is limited to short-distance communication, typically on the order of a few meters.
The protocol does not include any error-checking or correction mechanisms, making it vulnerable to data corruption in noisy environments.
The SPI protocol requires a separate SS wire for each slave device, which can be a limitation in systems with many devices.

Example:
One example of the SPI protocol in action is a microcontroller communicating with a sensor. The microcontroller sends commands to the sensor over the MOSI wire and receives data from the sensor over the MISO wire. The clock signal on the SCK wire ensures that the data transfer is synchronized, and the SS wire is used to select the sensor for communication. The SPI protocol allows for high-speed and efficient communication between the microcontroller and sensor, enabling a wide range of applications in areas such as robotics, automation, and industrial control systems.



1.Data Bits: The actual data being transmitted between the devices. This can be any number of bits, depending on the specific application.

2.Clock Signal: The clock signal is used to synchronize the transfer of data between the master and slave devices. The clock signal is generated by the master device and is sent on the SCK (serial clock) line.

3.Chip Select: The chip select (CS) line is used to select the specific slave device that the master wants to communicate with. Each slave device has its own chip select line, allowing the master to communicate with multiple devices on the same SPI bus.

5. Data Direction: The SPI protocol supports both full-duplex and half-duplex communication. In full-duplex mode, data can be transmitted in both directions simultaneously. In half-duplex mode, data can only be transmitted in one direction at a time.

Packet Information:-

1.Packet Size: The SPI protocol is a byte-oriented protocol, which means that data is transferred in units of 8 bits or one byte at a time.

2.Packet Structure: The packet structure of the SPI protocol is very simple and consists of four basic elements: a start bit, data bits, clock signal, and chip select signal. The start bit is typically a logic level transition that indicates the beginning of the packet. The data bits represent the actual data being transmitted, while the clock signal is used to synchronize the transfer of the data. The chip select line is used to select the specific slave device that the master wants to communicate with.

3.Packet Transmission: The SPI protocol supports both full-duplex and half-duplex communication. In full-duplex mode, data can be transmitted in both directions simultaneously. In half-duplex mode, data can only be transmitted in one direction at a time.

4.Packet Transfer Rate: The maximum transfer rate of the SPI protocol is determined by the clock frequency and the number of bits being transferred. In general, higher clock frequencies and shorter packets result in faster transfer rates.

5.Packet Error Detection and Correction: The SPI protocol does not provide any built-in error detection or correction mechanisms. Therefore, it is up to the application to implement appropriate error checking and correction methods to ensure reliable data transfer.

I2C:-

I2C (Inter-Integrated Circuit) is a communication protocol that is commonly used for communication between ICs (integrated circuits) in embedded systems. The protocol was developed by Philips Semiconductors (now NXP Semiconductors) in the early 1980s and is now widely used in a variety of applications, including sensors, displays, and other peripherals.

Working Procedure:

I2C uses two wires for communication, a serial data line (SDA) and a serial clock line (SCL). The SDA line is bidirectional and is used for transmitting and receiving data, while the SCL line is used to synchronize the data transfer between the master and the slave devices.

The I2C protocol uses a master-slave architecture, where the master device initiates and controls the communication, and the slave device responds to the master's commands. The master device generates the clock signal on the SCL line, and the slave device responds by transmitting data on the SDA line.

Advantages:

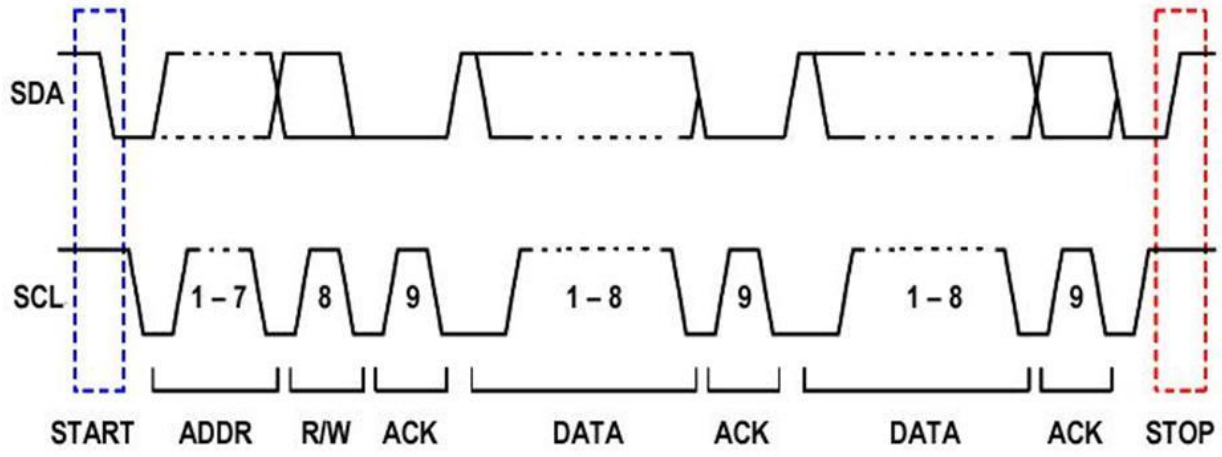
Simple hardware interface: I2C uses only two wires for communication, which makes it easy to implement in hardware.
Low power consumption: I2C is designed to operate at low voltages and low power levels, making it well-suited for battery-powered devices.
Wide range of devices: I2C is a widely adopted protocol and is supported by a wide range of devices, including sensors, displays, and other peripherals.
Multi-master support: I2C supports multiple master devices on the same bus, which allows for more complex communication networks.

Disadvantages:

Limited speed: I2C has a relatively low data transfer rate compared to other communication protocols, such as SPI and UART.
Limited range: I2C is designed for short-range communication, typically within a few meters.
Limited number of devices: The I2C protocol supports only a limited number of devices on the same bus (usually up to 128 devices).

Example:

A typical example of I2C communication is between a microcontroller (the master device) and a sensor (the slave device). The microcontroller initiates the communication by sending a start condition on the SDA line, followed by the address of the sensor on the bus. The sensor responds by acknowledging the address and sending data back to the microcontroller. The microcontroller then sends a stop condition on the SDA line to end the communication. This process can be repeated for multiple sensors on the same bus.



1.Start Condition: The start condition is a high-to-low transition on the SDA line while the SCL line is high. This indicates the start of a new transaction.

2.Address: The address is the first byte transmitted after the start condition. It consists of a 7-bit address of the slave device, followed by a single bit indicating whether the master device wants to read from or write to the slave device.

3.Control Byte: The control byte is an optional byte transmitted by the master device after the address byte. It is used to specify additional options or commands to the slave device.

4.Data Bytes: The data bytes are the actual data being transmitted or received between the master and slave devices. The number of data bytes can vary, depending on the transaction.

5.Acknowledge (ACK)/Not-Acknowledge (NACK): After each data byte is transmitted, the receiving device sends an acknowledge (ACK) or not-acknowledge (NACK) bit to the transmitting device to indicate whether the data was successfully received.

6. Stop Condition: The stop condition is a low-to-high transition on the SDA line while the SCL line is high. This indicates the end of the transaction.

ASSIGNMENTS

1. List out the top Microcontroller Supplier Companies

1.Texas Instruments (TI): TI provides a wide range of microcontrollers, including ARM Cortex-M, MSP430, and C2000 series.

2.Microchip Technology: Microchip offers a broad portfolio of microcontrollers, including 8-bit, 16-bit, and 32-bit PIC microcontrollers.

3.STMicroelectronics: STMicroelectronics provides a variety of microcontrollers, including ARM Cortex-M, STM8, and STM32 series.

4.NXP Semiconductors: NXP offers a range of microcontrollers, including ARM Cortex-M, LPC, and Kinetis series.

5.Renesas Electronics: Renesas offers a range of microcontrollers, including R8C, RX, and RL78 series.

6.Infineon Technologies: Infineon provides a variety of microcontrollers, including 8-bit, 16-bit, and 32-bit XMC and AURIX series.

7.Atmel Corporation: Atmel offers a variety of microcontrollers, including AVR, SAM3, and SAM4 series.

8. Intel: 8051, Quark, AVR32

9. Freescale: Kinetis, Vybrid, ColdFire

10. Silicon Labs: EFM8, ARM Cortex-M, C8051

2. What Is SOC?Give Any 2 Examples ?

A System-on-Chip (SoC) is an integrated circuit that combines all the components of a computer or other electronic system into a single chip. An SoC typically contains a microprocessor, memory, input/output interfaces, and other system components all on a single chip.

SoC technology has been used to create powerful and versatile systems for a wide range of applications, including smartphones, tablets, gaming consoles, digital cameras, medical devices, and many more.

Examples:-

1.Qualcomm Snapdragon: The Snapdragon is a popular SoC used in many smartphones and tablets, including Samsung Galaxy and Google Pixel devices.

2.Nvidia Tegra: The Tegra is an SoC used in gaming consoles, such as the Nvidia Shield.

3.Raspberry Pi: The Raspberry Pi is a popular single-board computer that uses an SoC.

4.Intel Atom: The Atom is an SoC used in netbooks and other low-power devices.

5.Apple A-series: Apple's A-series SoCs power iPhones, iPads, and other Apple devices.

6.TI Sitara: TI's Sitara SoCs are used in a variety of industrial and automotive applications.

3. What is Difference Between Opcode and Hex Code ?

Opcode and Hex code are two different representations of machine instructions in a computer system.

Opcode: Opcode, short for "operation code," is a binary code that specifies the operation to be performed by the CPU. It is a sequence of 0's and 1's that represents a specific operation such as ADD, SUBTRACT, LOAD, etc. Opcodes are defined by the CPU architecture and are used by the assembler or compiler to generate machine instructions.

Example: The opcode for the ADD instruction in the x86 architecture is 0000 0010.

Hex code: Hex code, short for "hexadecimal code," is a human-readable representation of the binary machine instructions. It is a sequence of hexadecimal digits (0-9, A-F) that corresponds to the binary code. Hex code is often used by programmers to manually enter machine instructions or to view the contents of memory.

Example: The hex code for the ADD instruction in the x86 architecture is 02.

4. Advantages of Digital Systems over Analog Systems

1.Accuracy: Digital systems are more accurate than analog systems because they operate on a discrete set of values, rather than a continuous range of values. Digital systems can achieve higher precision and accuracy in measuring and processing signals, making them more reliable.

2.Noise Immunity: Digital signals are less susceptible to noise and interference compared to analog signals. This is because digital signals can be easily regenerated and corrected using error detection and correction techniques, while analog signals are susceptible to distortion and noise due to their continuous nature.

3.Reproducibility: Digital systems are highly reproducible, meaning that they can be easily duplicated and replicated. This is important in fields such as telecommunications, where signals need to be transmitted over long distances and reproduced accurately at the receiving end.

4.Flexibility: Digital systems offer more flexibility in signal processing and manipulation. Digital signal processing techniques can be used to enhance or modify signals, while analog systems have limited processing capabilities.

5. Storage: Digital signals can be easily stored and retrieved, making them ideal for data storage and retrieval applications. This is why digital technology is widely used in devices such as computers, smartphones, and other electronic devices.

6. What is Boot process of Computer And What is BIOS

The boot process of a computer is the sequence of events that occur when the computer is turned on or restarted. During the boot process, the computer performs a series of checks and operations to load the operating system and prepare the system for use.

The boot process typically consists of the following steps:

1.Power-on self-test (POST): When the computer is turned on, the BIOS (Basic Input/Output System) performs a series of checks to ensure that the hardware components are functioning correctly. This includes checking the memory, hard drives, and other peripherals.

2.BIOS initialization: Once the POST is complete, the BIOS initializes the system hardware and sets up the basic configuration settings for the system.

3.Boot loader: The boot loader is a program that loads the operating system into memory. The boot loader is typically stored on the hard drive or other storage device and is loaded by the BIOS.

4.Operating system initialization: Once the boot loader is loaded, it initializes the operating system and starts the system services and drivers.

5.User logon: After the operating system is initialized, the user can log on to the system and start using it.

The BIOS is a firmware program that is stored on a chip on the motherboard of the computer. It is responsible for initializing and configuring the system hardware during the boot process, and provides basic input/output services to the operating system.

The BIOS also provides a basic set of configuration settings that can be accessed and modified through a setup utility. These settings include boot order, system clock settings, and hardware configuration settings.

Note:-the boot process of a computer is the sequence of events that occur when the computer is turned on or restarted, and the BIOS is a firmware program that is responsible for initializing and configuring the system hardware during the boot process.

7. How Operating System Works..?and list out tasks of OS.

An operating system (OS) is a software program that manages the hardware and software resources of a computer system. The OS provides a layer of abstraction between the hardware and software, and manages the resources of the system to ensure that applications can run efficiently and securely.

The basic functions of an operating system include:

1.Process management: The OS manages the processes running on the system, including starting and stopping processes, scheduling processes for execution, and allocating system resources to processes.

2.Memory management: The OS manages the system memory, including allocating memory to processes, freeing memory when it is no longer needed, and swapping memory between the main memory and secondary storage.

3.Input/output (I/O) management: The OS manages the input and output operations of the system, including managing devices such as disk drives, printers, and network adapters.

4.File system management: The OS manages the file system, including organizing files and directories, managing access to files, and ensuring the integrity and security of the file system.

5.Security: The OS provides security features to protect the system and user data, including user authentication, access control, and encryption.

6.Networking: The OS provides networking features to allow the system to connect to other systems and devices, including network protocols, drivers, and configuration tools.

8. User interface: The OS provides a user interface for interacting with the system, including graphical user interfaces (GUIs), command-line interfaces (CLIs), and other input/output mechanisms.

9. What are Single Board Computer...?LIst The Examples...!

A single-board computer (SBC) is a complete computer built on a single circuit board, with microprocessor(s), memory, input/output (I/O) interfaces, and other components required for a fully functional computer. SBCs are designed to be small, affordable, and low-power, making them ideal for embedded systems, robotics, and other applications where space and power are limited.

Some examples of single-board computers include:

1.Raspberry Pi: A series of small, affordable SBCs designed for education and hobbyist projects.

2.BeagleBoard: A series of SBCs designed for embedded applications, with a focus on low power consumption and small size.

3.Arduino: A family of microcontroller boards designed for hobbyist and educational use, with a focus on physical computing and electronic prototyping.

4.Odroid: A series of SBCs designed for use as media centers, gaming consoles, and other multimedia applications.

5.Jetson Nano: A small, powerful SBC designed for artificial intelligence and machine learning applications.

6.Banana Pi: A series of SBCs designed for education, hobbyist projects, and industrial applications.

7.NanoPi: A series of SBCs designed for networking, storage, and other embedded applications.

8.Intel NUC: A series of small, powerful SBCs designed for home theater, gaming, and other multimedia applications.

Note:-single-board computers are complete computers built on a single circuit board, with microprocessor(s), memory, I/O interfaces, and other components required for a fully functional computer. Examples include the Raspberry Pi, BeagleBoard, Arduino, Odroid, Jetson Nano, Banana Pi, NanoPi, and Intel NUC.

10. What Is RTOS...?

RTOS stands for Real-Time Operating System. It is an operating system designed to handle real-time applications where timing and response are critical. A real-time system must be able to respond to an external event within a fixed time frame, usually in microseconds or milliseconds.

RTOS provides an environment where tasks can be prioritized and scheduled to run within a specific time frame. The scheduler in RTOS is responsible for ensuring that the highest priority task gets executed first. This is crucial in applications such as aerospace, automotive, medical devices, and industrial automation, where safety and reliability are critical.

RTOS has a smaller footprint than a general-purpose operating system, making it suitable for resource-constrained embedded systems. It provides features like task management, memory management, and inter-task communication. It also provides support for real-time interrupts, which allows it to handle external events with high priority.

Examples of RTOS include FreeRTOS, uC/OS, VxWorks, and QNX. These RTOS provide a variety of features and are widely used in various applications.

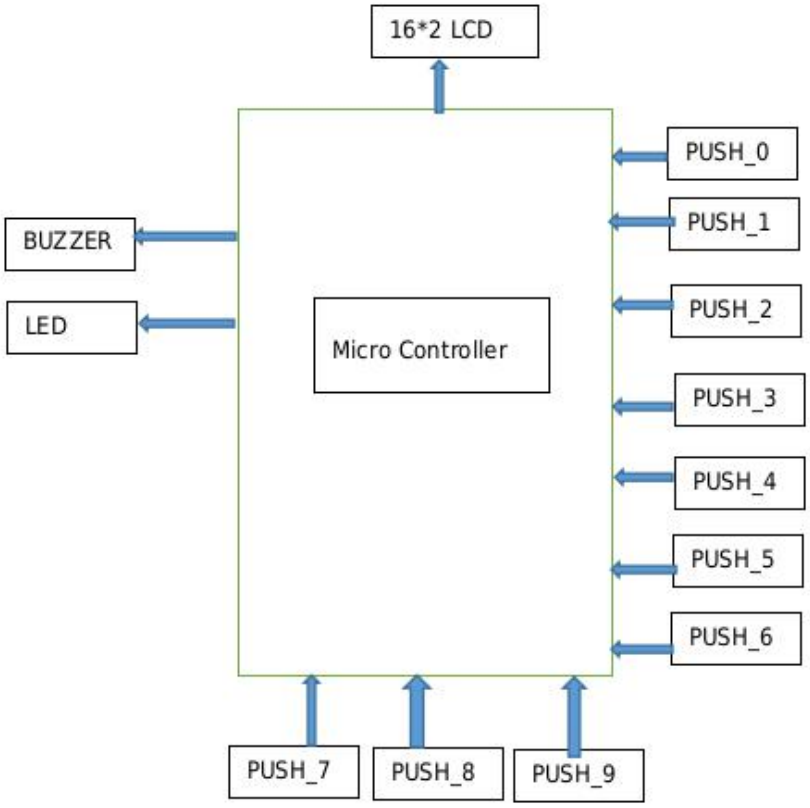
7.PROJECTS

Standardized procedure for developing projects for embedded systems

1.Make a Block Diagram of whole system.
2.Make a Table of Inputs and Outputs.
3.Make a Flow Chart of Solution.
4.Write Code in C using Arduino Ide.
5.Make a Circuit and Simulate using .hex file.

7.1. Password Protect Lock System using Push buttons with AMC

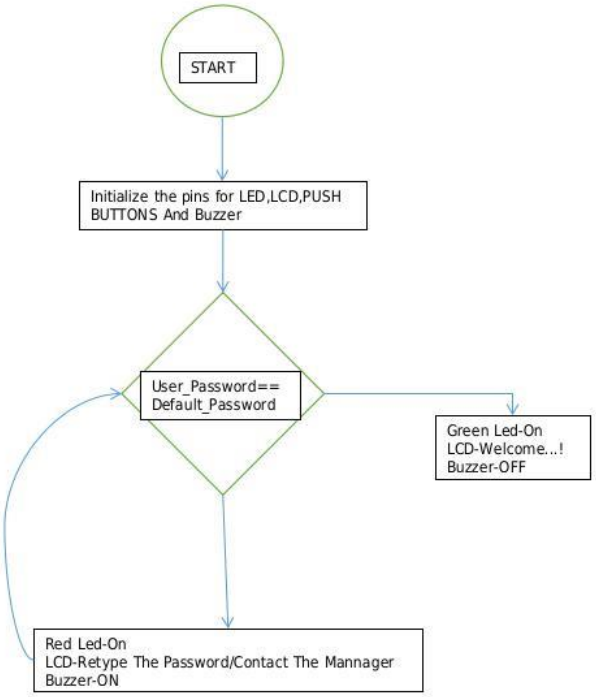
1. Block Diagram



2. Table

S.No.	Description	Name	Type	Data Direction	Specification	Remarks
1	**Pushbutton0**	PUSH_0	INPUT	DI	5VDC	
2	**Pushbutton1**	PUSH_1	INPUT	DI	5VDC	
3	**Pushbutton2**	PUSH_2	INPUT	DI	5VDC	
4	**Pushbutton3**	PUSH_3	INPUT	DI	5VDC	
5	**Pushbutton4**	PUSH_4	INPUT	DI	5VDC	
6	**Pushbutton5**	PUSH_5	INPUT	DI	5VDC	
7	**Pushbutton6**	PUSH_6	INPUT	DI	5VDC	
8	**Pushbutton7**	PUSH_7	INPUT	DI	5VDC	
9	**Pushbutton8**	PUSH_8	INPUT	DI	5VDC	
10	**Pushbutton9**	PUSH_9	INPUT	DI	5VDC	
11	**16*2 I2c_LCD**	16*2 LCD	OUTPUT	DO	NA	
12	**Red_Led**	LED	OUTPUT	DO	5VDC	
13	**Green_Led**	LED	OUTPUT	DO	5VDC	
14	**Buzzer**	BUZZER	OUTPUT	D0	5VDC	

3. FLow-Chart



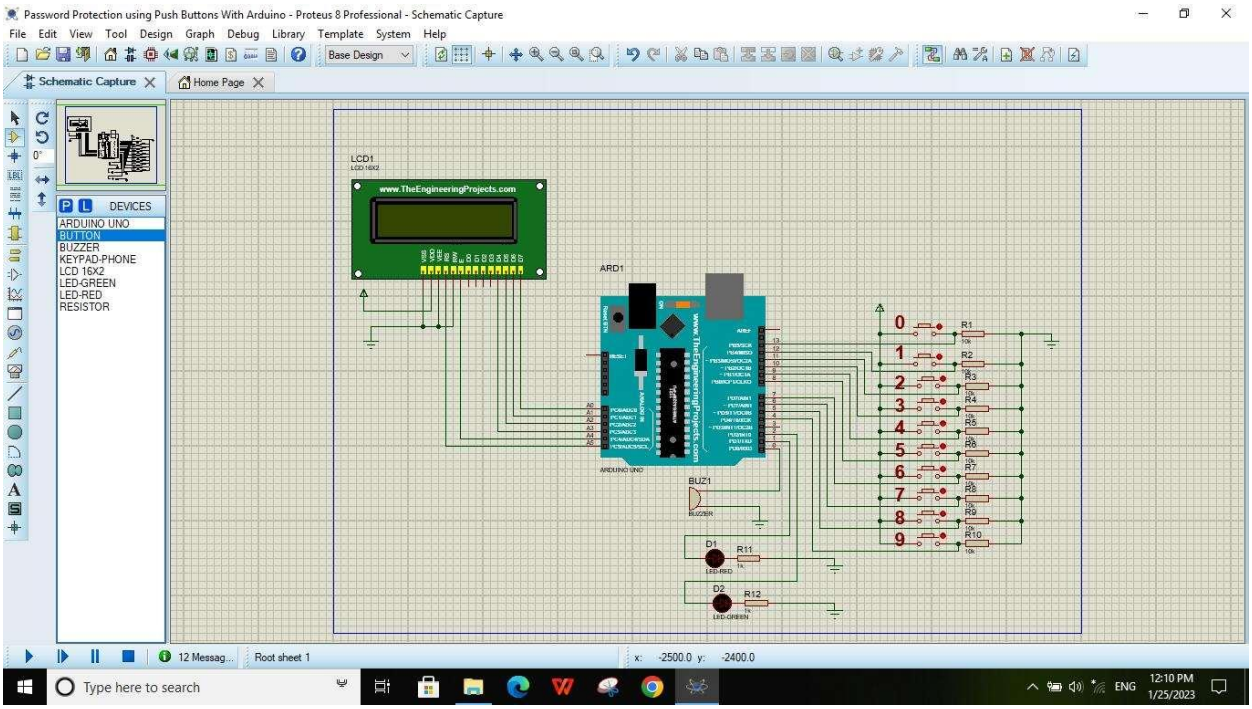
4. Simulation Code

```
#include <LiquidCrystal.h>
#include<string.h>
const int rs = A5, en = A4, d4 = A3, d5 = A2, d6 = A1, d7 = A0;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
unsigned int arduino_button_pins[]={0,1,2,3,4,5,6,7,8,9};
unsigned int button_present_values[]={0,0,0,0,0,0,0,0,0,0};
unsigned int button_past_values[]={0,0,0,0,0,0,0,0,0,0};
const int Green_LED=10;
```

```
const int Red_LED=11;
const int Buzzer=12;
int key_pressed()
{
for(uint8_t button=0;button<10;button++)
{
int present_state = digitalRead(arduino_button_pins[button]);
int previous_state= button_past_values[button];
if(present_state)
{
if(present_state != previous_state)
{
button_past_values[button] = present_state;
char str[10];
sprintf(str,"KEY:%d",button);//Mearge the Strings
lcd.setCursor(0,1);
lcd.write(str);
return button;
}}
else
{
button_past_values[button] = 0;
}
delay(50);
}
}
int press_button()
{
if(digitalRead(0)||digitalRead(1)||digitalRead(2)||digitalRead(3)||digitalRead(
4)||digitalRead(5)
||digitalRead(6)||digitalRead(7)||digitalRead(8)||digitalRead(9))
{ return 1; }
else
{ return 0; }
}
void setup()
{
for(int i=0; i<10;i++)
{
pinMode(i,INPUT);
}
pinMode(Green_LED,OUTPUT);
pinMode(Red_LED,OUTPUT);
pinMode(Buzzer,OUTPUT);
lcd.begin(16, 2);
lcd.write("ENTER PIN");
}
```

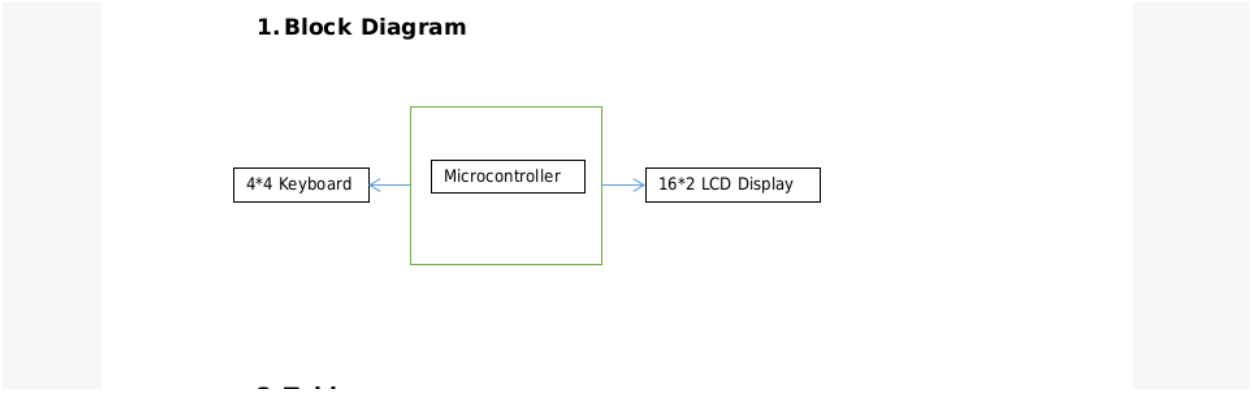
```
const int Preset_Pin=2332; //Password prefixed saved
int pinByUser[] ={0,0,0,0} ;
int keySequence = 0;
int Final_Pin = 0 ;
void loop()
{
while(press_button())
{
if(keySequence<4){
pinByUser[keySequence]=key_pressed();
lcd.setCursor(6,1);
char pin[4];
sprintf(pin,"DIGIT%d-%d",keySequence+1,pinByUser[keySequence]);
lcd.write(pin);
}
else if(keySequence==4){}
lcd.setCursor(6,1);
for(int a=0; a<4; a++)
{
Final_Pin = (Final_Pin * 10) + pinByUser[a];
}
char pin[4];
sprintf(pin,"PIN:%d",Final_Pin);
lcd.write(pin);
delay(50);
if(Final_Pin == Preset_Pin){
lcd.setCursor(6,1);
lcd.write("Welcome");
lcd.setCursor(0,0);
lcd.write("ACCESS AUTHORIZED !!");
digitalWrite(Green_LED,HIGH);
}else{
lcd.setCursor(6,1);
lcd.write("-!FAILED!-");
digitalWrite(Red_LED,HIGH);
}
}
else{
lcd.setCursor(0,0);
lcd.write("MAX LIMIT REACHED");
}
delay(1200);
keySequence++;}
```

5.Simulation Circuit



7.2.Building a Custom Calculator Using Arduino Microcontroller

1. Block Diagram

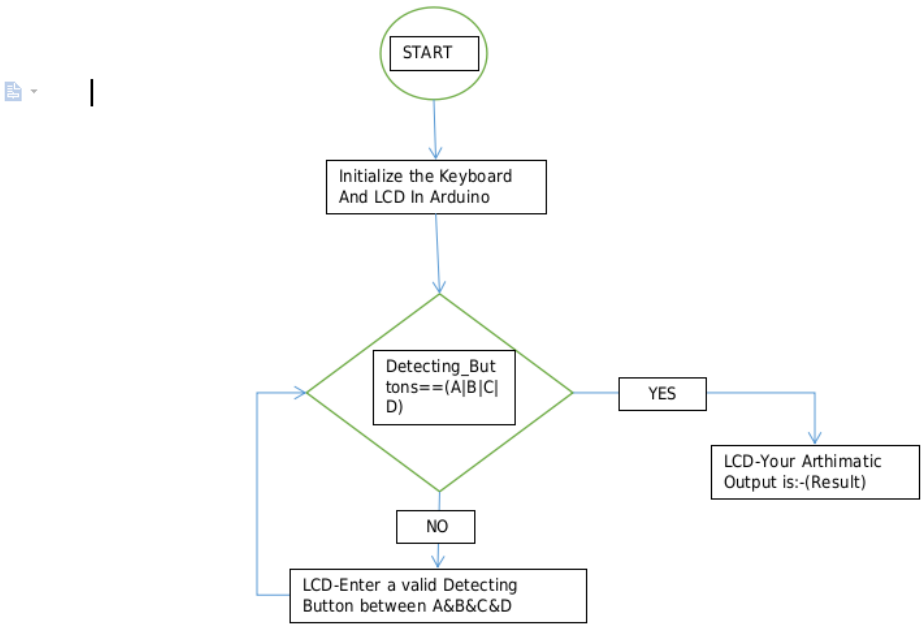


2. Table

2. Table

s.no.	Discription	Name	Type	Data Direction	Specification	Remarks
1.	lcd	16*2 LCD Display	output	DO	5VDC	
2.	Keypad	4*4 Keyboard	Input	DI	NA	

3. Flow-Chat



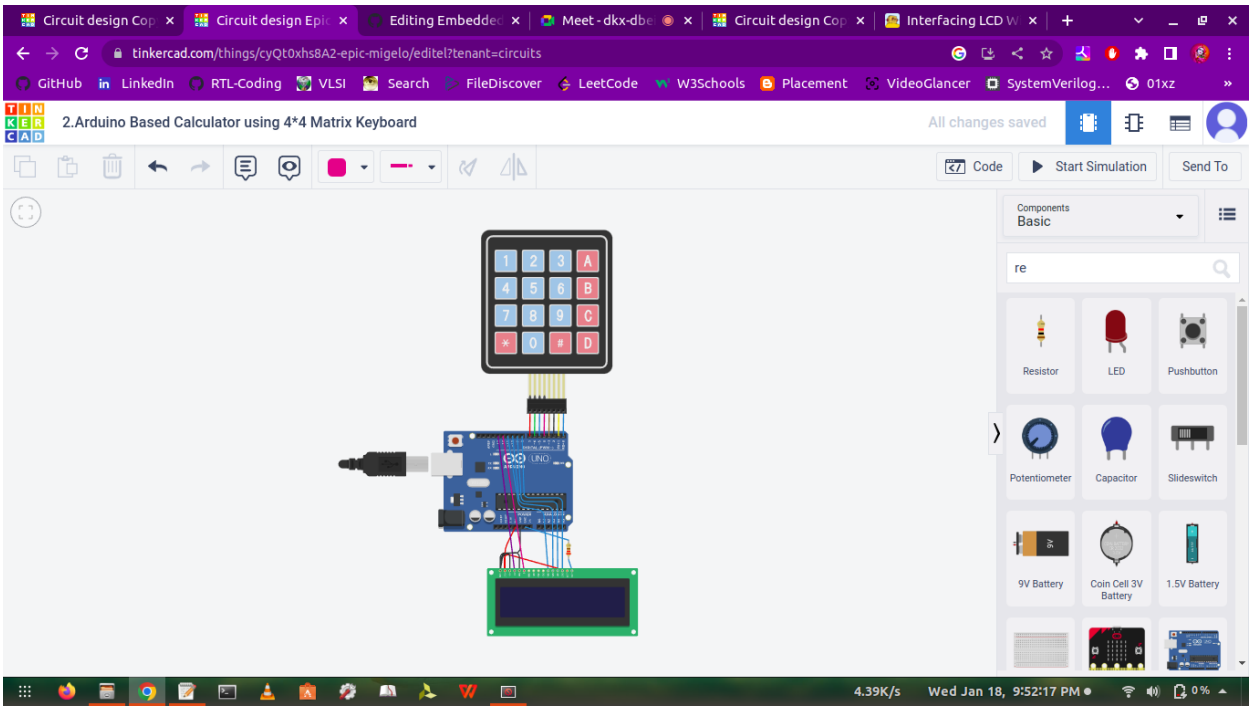
4. Simulation Code

```
#include <Keypad.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(13, 12, 11, 10, 9, 8);
long Num1 = 0;
long Num2 = 0;
double Result = 0;
char Key;
const byte ROWS = 4;const byte COLS = 4;
char keys[ROWS][COLS] = {
{'1','2','3','+'},
{'4','5','6','-'},
{'7','8','9','*'},
{'C','0','=','/'}
};
```

```
byte rowPins[ROWS] = {7,6,5,4}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {3,2,1,0}; //connect to the column pinouts of the keypad
//initialize an instance of class NewKeypad
Keypad myKeypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS,
COLS);
void setup()
{
lcd.begin(16, 2);// start lcd
lcd.setCursor(0,0);// in lcd Setcursor point at 0th column and 0th row
lcd.print("Calculator By");
lcd.setCursor(0,1);//in lcd Setcursor point at 0th column and 1th row
lcd.print("O.V.Krishnaiah");
delay(4000);//Wait the information until 4 milli seconds
lcd.clear();//clear the lcd Screen
lcd.setCursor(0, 0);////in lcd Setcursor point at 0th column and 0th row
}
void loop()
{
Key = myKeypad.getKey();//getKey() instance method to Store the Pressed
Key
switch(Key)
{
case '0' ... '9': // This keeps collecting the first value until a operator is
pressed "+-*/"
lcd.setCursor(0,0);
Num1 = Num1 * 10 + (Key - '0');
lcd.print(Num1);
break;
case '+':Num1 = (Result != 0 ? Result : Num1);
lcd.setCursor(0,1);
lcd.print("+");
Num2 = Number2(); // get the collected the second number
Result = Num1 + Num2;
lcd.setCursor(0,3);
lcd.print(Result);
Num1 = 0, Num2 = 0; // reset values back to zero for next use
break;
case '-':
Num1 = (Result != 0 ? Result : Num1);
lcd.setCursor(0,1);
lcd.print("-");
Num2 = Number2(); // get the collected the second number
Result = Num1 - Num2;
lcd.setCursor(0,3);
lcd.print(Result);
Num1 = 0, Num2 = 0; // reset values back to zero for next use
break;
```

```
case '*':
Num1 = (Result != 0 ? Result : Num1);
lcd.setCursor(0,1);
lcd.print("*");
Num2 = Number2(); // get the collected the second number
Result = Num1 * Num2;
lcd.setCursor(0,3);
lcd.print(Result);
Num1 = 0, Num2 = 0; // reset values back to zero for next use
break;
case '/':
Num1 = (Result != 0 ? Result : Num1);
lcd.setCursor(0,1);
lcd.print("/");
Num2 = Number2(); // get the collected the second number
Result = Num1 / Num2;
lcd.setCursor(0,3);
lcd.print(Result);
Num1 = 0, Num2 = 0; // reset values back to zero for next use
break;Num2 == 0 ? lcd.print("Invalid Number") : Result = (float)Num1 /
(float)Num2;
lcd.print(Result);
Num1 = 0, Num2 = 0;
break;
case 'C':
Result = 0;//Cancle The Calculation
lcd.clear();
break;
}
}
long Number2()
{
while( 1 )
{
Key = myKeypad.getKey();
if(Key >= '0' && Key <= '9')
{
Num2 = Num2* 10 + (Key - '0');
lcd.setCursor(0,2);
lcd.print(Num2);
}
if(Key == '=') break; //return Num2;
}
return Num2;
}
```

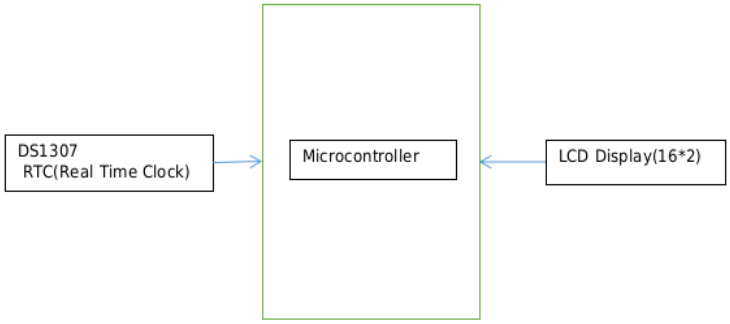
5. Simulation Circuit



7.2. Building a DS1307 Real time clock (RTC) Using Arduino MC.

1. Block Diagram

1. Block Diagram:-



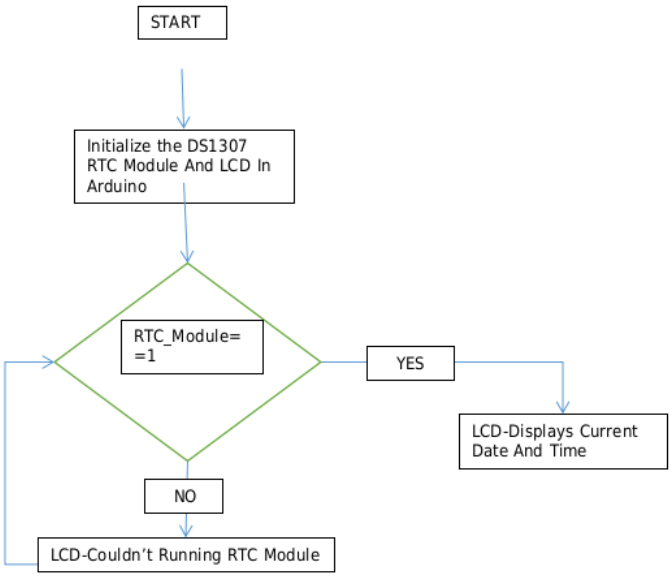
2. Table

2. Tables:-

s.no.	Discription	Name	Type	Data Direction	Specification	Remarks
1.	**LCD**	lcd	Output	DO	5VDC	
2.	**DS1307 RTC**	Ds1307 RTC Module	Input	DI	4MHZ	

3. Flow-Chat

3. Flow Chart:-

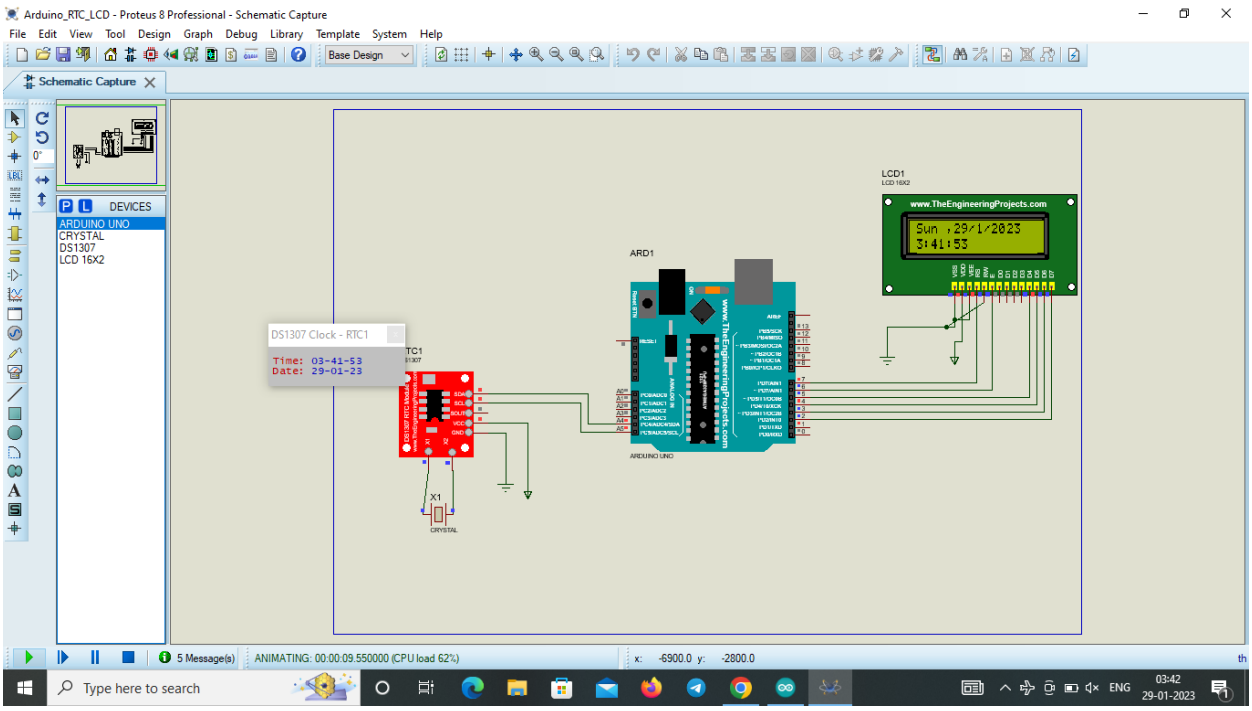


4. Simulation Code

```
#include <Wire.h>
#include <LiquidCrystal.h>
#include "RTClib.h"
RTC_DS1307 rtc;
const int rs=7,en=6,d4=5,d5=4,d6=3,d7=2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7); // (rs, e, d4, d5, d6, d7)
char Days_per_Year_in_week[7][12] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat"};
void setup ()
{
Serial.begin(9600);
lcd.begin(16, 2);
if (! rtc.begin())
{
lcd.print("Couldn't find RTC");
while (1);
}
if (! rtc.isrunning())
{
lcd.print("RTC is NOT running!");
}
rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));//auto update from computer
time
//rtc.adjust(DateTime(Year,Month,Date, Hours, Minutes, Seconds));// to set the
time manualy
}
```

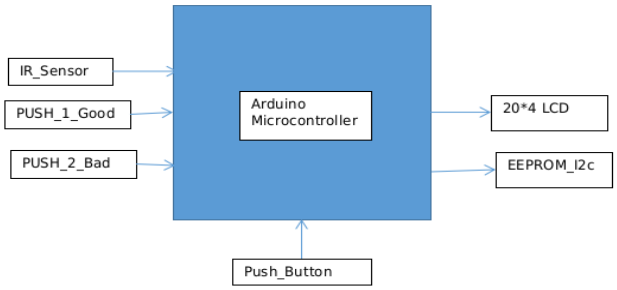
```
void loop ()
{
DateTime now = rtc.now();
lcd.setCursor(0, 1);
lcd.print(now.hour());
lcd.print(':');
lcd.print(now.minute());
lcd.print(':');lcd.print(now.second());
lcd.print(" ");
lcd.setCursor(0, 0);
lcd.print(Days_per_Year_in_week[now.Days_per_Year_in_week()]);
lcd.print(" ,");
lcd.print(now.day());
lcd.print('/');
lcd.print(now.month());
lcd.print('/');
lcd.print(now.year());
}
```

5.Simulation Circuit



7.3.Building Automated product counter using Arduino MC.

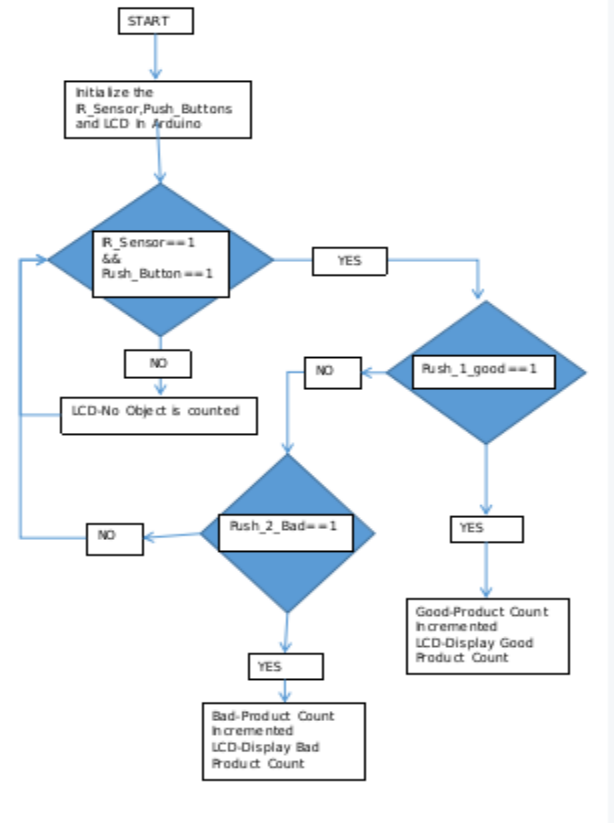
1. Block Diagram



2. Table

s.no.	Discription	Name	Type	Data Direction	Specification	Remarks
1.	***20*4 LCD Display***	20*4 LCD	*OUTPUT*	DO	*5VDC*	
2.	***Pushbutton***	PUSH_1_Good	*INPUT*	DI	*5VDV*	
3.	***Pushbutton***	*Push_2_Bad*	*INPUT*	DI	*5VDC*	
4.	***IR_Sensor***	IR_Sensor	*INPUT*	DI	*N/A*	
5.	***Pushbutton***	Push_Button	*INPUT*	DI	*5VDC*	
6.	***EEPROM***	EEPROM_I2C	*OUTPUT*	DO	*5VDC*	

3.Flow-Chat



4.Simulation Code

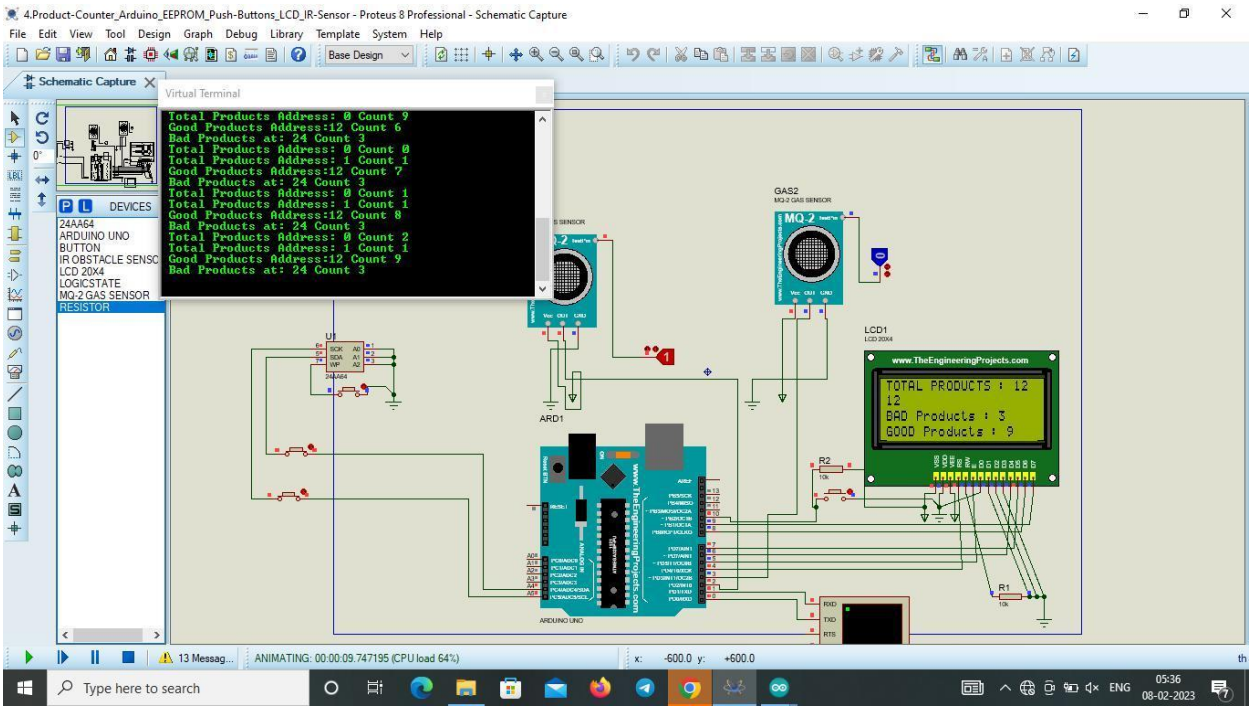
```
#include <Wire.h>
#include <LiquidCrystal.h>
#define TotalAddress 0x00 //Intail Address to Store Total Products Count
#define GoodAddress 0x0C //Intail Address to Store Bad Product Count
#define BadAddress 0x18 //Intail Address to Store Good Product Count
int bad = 0;
int good = 0;
int total = 0;
const int eeprom_address = 0x50; // I2C address of the 24LC256 EEPROM
void LCD_print();
void Clear_EEPROM();
LiquidCrystal lcd(4, 5, 6, 7, 8, 9);
void setup()
{
Wire.begin();
Serial.begin(9600);
for(int i=2 ; i<4 ; i++)
{
pinMode(i , INPUT);
}
pinMode(10 , INPUT);
```

```
Serial.print("Automated Product Counter System ");
lcd.begin(20, 4);
lcd.print("--Product Counter--");
lcd.setCursor(0, 1);
lcd.print("
By
");
lcd.setCursor(0, 2);
lcd.print(" Venkata Krishnaiah ");
delay(1000);
lcd.clear();
for (int address = 0; address < 36; address++) //Reset all the Data of
Total, Bad and Good Product Count that Program is loaded before Starting
{
writeEEPROM(TotalAddress+address, 0xFF);
}
}void loop()
{
int clearROM = digitalRead(10); //If clear button is pressed it gives 0 else it
returns 1
if(!clearROM) //clears EEPROM when it returns 0
{
Clear_EEPROM(); //clear the EEPROM
}
else
{
if(digitalRead(2))
{
good++;
}
if(digitalRead(3))
{
bad++;
}
total = good + bad ;
LCD_print();
byte i = 0x00 ;
for(int num = total ; num > 0 ; num = num/10)
{
int rem = num % 10 ;
writeEEPROM(TotalAddress+i, rem); // Writing Total Products Count Byte
by Byte
Serial.print(" Total Products Address: ");
Serial.print(TotalAddress+i);
Serial.print(" Count ");
Serial.println(rem);
i++;
```

```
}
byte j = 0x00 ;
for(int num = good ; num > 0 ; num = num/10)
{
int rem = num % 10 ;
writeEEPROM(GoodAddress+j, rem); // Writing Good Products Count
Byte by Byte
Serial.print(" Good Products Address:");
Serial.print(GoodAddress+j);Serial.print(" Count ");
Serial.println(rem);
j++;
}
byte k = 0x00 ;
for(int num = bad ; num > 0 ; num = num/10)
{
int rem = num % 10 ;
writeEEPROM(BadAddress+k, rem); // Writing Bad Products Count Byte
by Byte
Serial.print(" Bad Products at: ");
Serial.print(BadAddress+k);
Serial.print(" Count ");
Serial.println(rem);
k++;
}
/*
// Read data from the EEPROM
byte data = readEEPROM(2);
Serial.print("Data: ");
Serial.println(data);
data = readEEPROM(3);
Serial.print("Data: ");
Serial.println(data); */
}
delay(500); // Wait for 0.5 second before writing and reading the data
again
}
byte readEEPROM(int address)
{
byte data;
Wire.beginTransmission(eeprom_address);
Wire.write((int)(address >> 8)); // Send the high byte of the address
Wire.write((int)(address & 0xFF)); // Send the low byte of the address
Wire.endTransmission();
Wire.requestFrom(eeprom_address, 1);
if (Wire.available())
{
data = Wire.read();
```

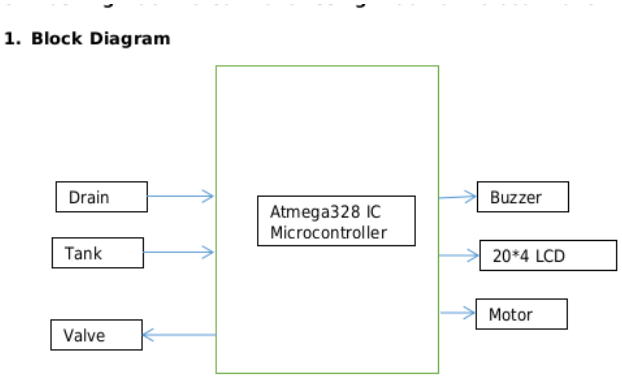
```
}return data;
}
void writeEEPROM(int address, byte data) {
Wire.beginTransmission(eeprom_address);
Wire.write((int)(address >> 8)); // Send the high byte of the address
Wire.write((int)(address & 0xFF)); // Send the low byte of the address
Wire.write(data);
Wire.endTransmission();
delay(5); // wait for the EEPROM to complete the write
}
void LCD_print()
{
lcd.clear();
lcd.setCursor(0, 2);
lcd.print("BAD Products : ");
lcd.print(bad);
lcd.setCursor(0, 3);
lcd.print("GOOD Products : ");
lcd.print(good);
lcd.setCursor(0, 0);
lcd.print("TOTAL PRODUCTS : ");
lcd.print(total);
}
void Clear_EEPROM()
{
for (int address = 0; address < 32768; address++)
{
Wire.beginTransmission(eeprom_address);
Wire.write((int)(address >> 8));
Wire.write((int)(address & 0xFF));
Wire.write(0xFF);
Wire.endTransmission();
}Serial.println("EEPROM data cleared!");}
```

5.Simulation Circuit



7.5.Washing Machine Controller Using Arduino MC.

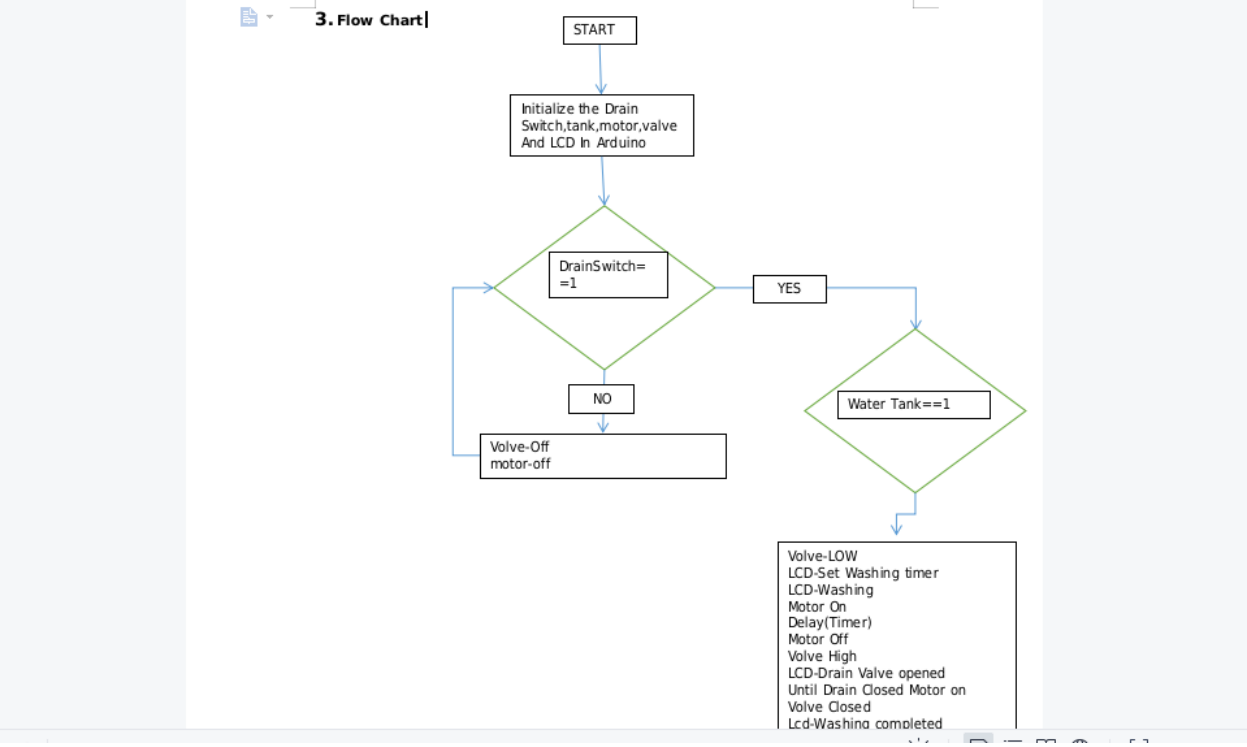
1.Block Diagram



2.Table

s.no.	Discription	Name	Type	Data Direction	Specification	Remarks
1.	**Draining**	Drain	input	DI	5VDC	
2.	**Water-Tank**	Tank	input	DI	5VDC	
3.	**Drainvalve**	valve	Output	DO	5VDC	
4.	**Buzzer**	buzzer	Output	DO	5VDC	
5.	**LCD**	20*4 LCD	Output	DO	5VDC	
6.	**Motor**	Motor	Output	DO	NA	

3.Flow-Chat

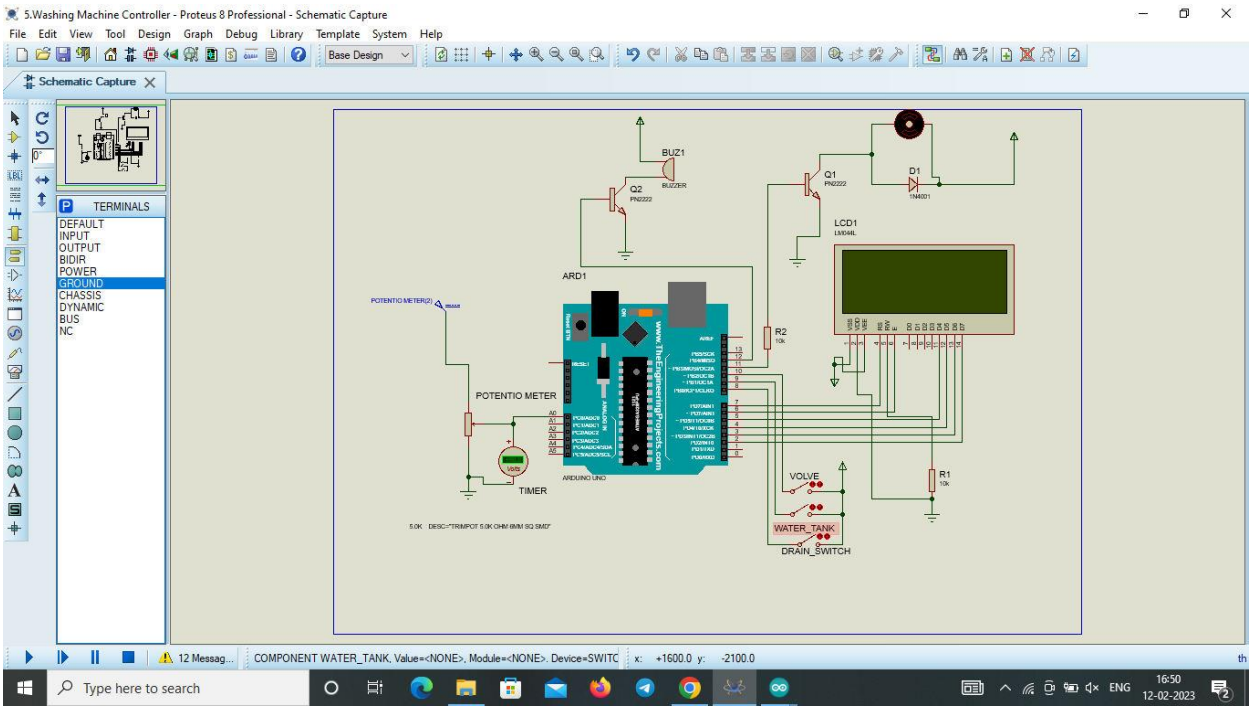


4.Simulation Code

```
#include <LiquidCrystal.h>
const int drainSwitch = 8;
const int waterTank = 9;
const int drainValve = 10;
const int washingMotor = 11;
const int buzzer = 12;
const int timerSetting = A0;
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
int timerValue;
void setup() {
pinMode(drainSwitch, INPUT);
pinMode(waterTank, INPUT);
pinMode(drainValve, OUTPUT);
pinMode(washingMotor, OUTPUT);
pinMode(buzzer, OUTPUT);
lcd.begin(20, 4);
}
void loop() {
// Check for drain switch
if (digitalRead(drainSwitch) == HIGH) {
// Wait for water to drain out
delay(3000);
```

```
// Check for empty water tank
if (digitalRead(waterTank) == HIGH) {
// Close drain valve
digitalWrite(drainValve, LOW);
lcd.clear();
lcd.print("Drain valve closed");
delay(1000);
// Ask for timer setting
lcd.clear();
lcd.print("Set washing timer:");
timerValue = analogRead(timerSetting);
// Convert timer value to minutes
timerValue = map(timerValue, 0, 1023, 0, 60);
lcd.setCursor(0, 1);
lcd.print(timerValue);
lcd.print(" mins");
delay(1000);
// Start washing
lcd.clear();
lcd.print("Washing...");
digitalWrite(washingMotor, HIGH);
delay(timerValue * 60000);
digitalWrite(washingMotor, LOW);
// Open drain valvedigitalWrite(drainValve, HIGH);
lcd.clear();
lcd.print("Drain valve opened");
delay(3000);
// Buzzer beeps until drain valve closed
while (digitalRead(drainSwitch) == HIGH) {
digitalWrite(buzzer, HIGH);
delay(500);
digitalWrite(buzzer, LOW);
delay(500);
}
// Close drain valve and indicate washing completed
digitalWrite(drainValve, LOW);
lcd.clear();
lcd.print("Washing completed!");
digitalWrite(buzzer, HIGH);
delay(1000);
digitalWrite(buzzer, LOW);
}
}
}
```

5.Simulation Circuit


5.Washing Machine Controller - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
TERMINALS
DEFAULT
INPUT
OUTPUT
BIDIR
POWER
GROUND
CHASSIS
DYNAMIC
BUS
NC
BUZ1
Q2
Q1
D1
LCD1
ARD1
R2
R1
POTENTIO METER
TIMER
VOLVE
WATER_TANK
DRAIN_SWITCH
ARDUINO UNO
www.TheEngineeringProjects.com
12 Messag...
COMPONENT WATER_TANK, Value=<NONE>, Module=<NONE>, Device=SWITC
x: +1600.0 y: -2100.0
Type here to search
ENG
16:50
12-02-2023

STUDENT FEEDBACK

Thank you for your kind words of Radhakumari Mam and Mr. Yogesh Kumar Sir. I'm thrilled to hear that found the course on Embedded Systems and Internet of Things at Sure Trust so engaging and informative. It's always inspiring to learn from instructors who can bring real-world examples into the classroom to make the learning experience more tangible and exciting.It's also great to hear that Mr. Yogesh Kumar Sir's guidance has helped to gain a strong foundation in the subject and that his explanations have been clear and easy to understand. It's essential to have an instructor who can break down complex concepts and make them accessible to everyone.I'm glad to hear that there is no partiality among students in the class and that Mr. Yogesh Kumar Sir treats everyone equally. It's always reassuring to have an instructor who is supportive and creates an environment where every student feels welcome.It's great to have an instructor like Radha Kumari mam who can inspire creativity in students and encourage them to reach their full potential with discipline and punctuality. It's also important to have an instructor who takes the time to motivate students to achieve their goals.Once again, thank you for giving this great future endeavors in my career as an Embedded Designer.

Uniqueness of the Course

1.Concepts are learnt theoretically and practically.
2.Faculty is committed in delivering the course.
3.Thought provoking assignments will be given.
4.Preparation of course report, which will help us to refer in the future.
5.Concentrate on each and every student in the course.

Concluding Remarks

I am very much happy that I have learnt the course successfully. I am blessed to learn the course in SURE TRUST. I would like to thank each and every person of SURE TRUST for providing me an opportunity to learn the course. SURE TRUST is helping the students to gain the knowledge on latest technology and helps them to get good jobs. I am proud to be a student of SURE TRUST.