A

Major Project Report on

IOT SMART WATER METERING: ARDUINO UNO IMPLEMENT SMART WATER METERS TO TRACK WATER CONSUMPTION AND ENABLE EFFICIENT BILLING FOR UTILITIES

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

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

B. UMADEVI	**22K85A0419**
R. PAVITHRA	**22K85A0417**
J. SRIKANTH REDDY	**21K81A04F5**

Under the Guidance of

B. PRASANTHI

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

St. MARTIN'S ENGINEERING COLLEGE

UGC Autonomous

Affiliated to JNTUH, Approved by AICTE

Accredited by NBA & NAAC A+, ISO 9001-2008 Certified

Dhulapally, Secunderabad – 500100

www.smec.ac.in

APRIL – 2025

St. MARTIN'S ENGINEERING COLLEGE

UGC Autonomous

NBA & NAAC A+ Accredited

Dhulapally, Secunderabad – 500100

CERTIFICATE

This is to certify that the mini project entitled 'IOT SMART WATER METERING: ARDUINO UNO IMPLEMENT SMART WATER METERS TO TRACK WATER CONSUMPTION AND ENABLE EFFICIENT BILLING FOR UTILITIES' is being submitted by R.PAVITHRA (22K85A0417), B.UMA DEVI(22K85A0419), J.SRIKANTH REDDY (21K81A04F5) in partial fulfilment of the requirement for the award of degree of BACHELOR OF TECHNOLOGY in Department of ELECTRONICS AND COMMUNICATION ENGINEERING. It is a record of bonafide work carried out by them. The result embodied in this report have been verified and found satisfactory.

Signature of guide	**Signature of HOD**
Mrs. B.PRASANTHI	**Dr. B. HARI KRISHNA**
Assistant professor	**Professor & HOD**

Internal Examiner **External Examiner**

St. MARTIN'S ENGINEERING COLLEGE
UGC Autonomous

NBA & NAAC A+ Accredited

Dhulapally, Secunderabad – 500 100

DECLARATION

We, the students of **Bachelor of Technology in Department of Electronics and Communication and Engineering,** session: **2021 – 2025, St. Martin's Engineering College, Dhulapally, Kompally, Secunderabad,** hereby declare that the work presented in this project "**IOT SMART WATER METERING: ARDUINO UNO IMPLEMENT SMART WATER METERS TO TRACK WATER CONSUMPTION AND ENABLE EFFICIENT BILLING FOR UTILITIES".**is the outcome of our own bonafide work and it is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics. This result embodied in this project report has not been submitted in any university for award of any degree.

B. UMADEVI	**22K85A0419**
R. PAVITHRA	**22K85A0417**
J.SRIKANTH REDDY	**21K81A04F5**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance have crowded our efforts with success.

We especially would like to express our deep sense of gratitude and indebtedness to **Dr. P. SANTOSH KUMAR PATRA**, Group Director, St. Martin's Engineering College Dhulapally, for permitting us to undertake this project.

We wish to record our profound gratitude to **Dr. M. SREENIVAS RAO**, Principal, St. Martin's Engineering College, for his motivation and encouragement.

We are also thankful to **Dr. B.HARI KRISHNA**, Head of the Department, Department of Electronics and Communication Engineering, St. Martin's Engineering College, for his support and guidance throughout our project and as well as our project coordinator **Dr. M.THIRUPATHI** Assistant Professor, Department of Electronics and Communication Engineering for his valuable support.

We would like to express our sincere gratitude and indebtedness to our project supervisor **Mrs. B.PRASANTHI** Assistant Professor, Electronics and Communication Engineering, St. Martin's Engineering College, Dhulapally, for his support and guidance throughout our project.

Finally, we express thanks to all those who have helped us successfully to completing this project. Furthermore, we would like to thank our family and friends for their moral support and encouragement. We express thanks to all those who have helped in successfully completing the project.

B. UMADEVI	**22K85A0419**
R. PAVITHRA	**22K85A0417**
J. SRIKANTH REDDY	**21K81A04F5**

ABSTRACT

This project presents an IoT-based Smart Water Metering system using the Arduino Uno microcontroller to track water consumption and enable efficient billing for utilities. The system integrates IoT connectivity, flow sensors, and an LCD display to provide real-time water usage monitoring and automated billing. The Arduino Uno processes water flow data and transmits it to a central server via IoT, ensuring seamless communication between consumers and utility providers. The LCD screen displays real-time water consumption, while users can access their usage data remotely through an IoT dashboard. Additionally, the system can trigger a pump control mechanism to regulate water supply in cases of excessive consumption or non-payment. By leveraging IoT technology, this smart metering system enhances water resource management, promotes efficient usage, and eliminates the need for manual meter reading. Future developments will focus on AI-based predictive analytics to detect leakages, abnormal consumption patterns, and dynamic tariff adjustments, further improving the sustainability and efficiency of urban water distribution.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction: IoT-Based Smart Water Metering Using Arduino Uno

Water is one of the most crucial natural resources for sustaining life, yet it remains highly mismanaged, wasted, and inefficiently distributed in many parts of the world. Traditional methods of monitoring water consumption rely heavily on manual meter readings, which often lead to inaccuracies, human errors, billing disputes, and excessive wastage. Moreover, many consumers lack real-time visibility into their water usage, leading to inefficient consumption habits. Additionally, the absence of automated monitoring systems prevents early detection of leakages, overuse, and unauthorized consumption, contributing to significant losses for both consumers and utility providers.

To address these challenges, the implementation of IoT-based smart water metering systems is emerging as an effective solution. By integrating sensor-based water flow measurement, automated data transmission, and cloud-based monitoring, an intelligent water management system can be developed to enable efficient usage, fair billing, and real-time tracking. This project focuses on the design and development of an IoT Smart Water Metering System using the **Arduino Uno microcontroller**, which allows for accurate water consumption tracking, automated billing, and remote monitoring through a web-based interface.

This system employs flow sensors to measure the real-time water flow, an LCD display for instant usage updates, and wireless IoT communication to transmit data to an online database. Additionally, an automated pump control mechanism ensures that water is supplied based on usage thresholds and payment status, preventing unnecessary wastage. The integration of **Arduino Uno microcontroller** with IoT enables seamless data logging, remote access, and efficient water distribution management, making it a highly effective solution for both residential and commercial applications.

Key Features and Components of the Smart Water Metering System

The IoT-based smart water metering system is designed to provide an efficient and automated solution for water consumption monitoring. It consists of several core components that work together to collect, process, and transmit data in real-time, ensuring seamless operation.

1. **Arduino Uno Microcontroller**: The **Arduino Uno** is the central processing unit of the system. It is a low-cost, versatile microcontroller that can easily interface with sensors and other components, making it ideal for IoT applications. The **Arduino Uno.**

- Collects and processes data from flow sensors.
- Transmits real-time water usage data to a cloud-based server.
- Controls the automated pump operation based on predefined conditions.
- Provides an interface for users to monitor water usage remotely.

2. **Flow Sensors:** Flow sensors are used to accurately measure the volume of water consumed. These sensors generate pulses based on the rate of water flow, which are then processed by the **Arduino Uno** to calculate the total consumption. The advantages of using flow sensors include:

- Real-time tracking of water consumption.
- Precision in billing, ensuring that users only pay for the exact amount of water they consume.
- Leakage detection, as abnormal flow rates can indicate potential water leaks.

3. **LCD Display**: An LCD screen is incorporated into the system to provide users with an instant visual representation of their water consumption. It displays:

- Current water usage (liters or cubic meters).
- Billing details based on consumption.
- Alerts for excessive usage or leakage detection.

4. **IoT Connectivity & Cloud-Based Monitoring**: The system is designed to be remotely accessible via a web-based dashboard or a mobile application. The IoT connectivity ensures:

- Seamless transmission of water usage data to a centralized server.
- Real-time access to consumption reports from any location.
- Instant notifications for leaks, high usage, or system errors.

5. **Automated Pump Control Mechanism**: One of the most important features of this system is the ability to automatically control the water pump based on preset conditions. This ensures:

- Efficient water distribution, preventing overuse or shortages.
- Cut-off mechanisms in case of excessive consumption or non-payment of water bills.

- Reduced manual intervention, allowing utility providers to manage water distribution remotely.

Significance and Impact of the IoT-Based Smart Water Metering System

The implementation of IoT in water metering presents numerous benefits, both for consumers and utility providers. Some of the most notable advantages include:

- **Accurate and Fair Billing**: Traditional water billing systems often rely on manual meter readings, which can be prone to human errors, miscalculations, and inefficiencies. With IoT-based smart metering, real-time data ensures that users are billed only for the water they actually consume, eliminating billing discrepancies and disputes.
- **Eliminating Manual Errors and Human Intervention**: Manual meter reading requires significant labor resources and is often subject to errors, delays, and inefficiencies. By automating the process, utility providers can:.
- **Remote Monitoring and Control**: With cloud-based connectivity, both consumers and utility providers can access real-time water usage data from any location. This allows for:
- **Leakage and Anomaly Detection**: Water leakages can result in significant financial losses and wastage of resources. By integrating real-time monitoring and anomaly detection algorithms, the system can:
- **Promoting Water Conservation and Sustainability**: With real-time insights, consumers become more aware of their water consumption habits, leading to:
- **Cost-Effective and Scalable Solution**: The low-cost hardware components used in the system make it an affordable solution for both residential and commercial applications. Additionally, the system is highly scalable, making it suitable for municipal-level deployment as well.

CHAPTER 2
LITERATURE SURVEY

The global need for sustainable water management has significantly influenced the shift towards IoT-based smart water metering systems. These systems offer real-time monitoring of water consumption, automated billing, and comprehensive data analysis. The traditional water metering methods are prone to human errors, inefficiencies in data collection, and errors in billing, which can lead to substantial discrepancies and water wastage. As urbanization and population growth continue to put pressure on water resources, the necessity for intelligent solutions that enable accurate, automated, and efficient water management is greater than ever. IoT-based systems are addressing this by integrating advanced sensor technologies, cloud computing, and artificial intelligence for smarter water monitoring and management.

This survey aims to provide a detailed review of 30 research papers that contribute to the development of IoT-based smart water metering systems. These studies span a wide range of topics, including sensor technology, wireless communication techniques, cloud integration, AI-based analytics, and the potential for smart city applications. In this survey, we will explore how IoT and related technologies enable real-time data collection, improve accuracy in billing systems, optimize water consumption patterns, and contribute to sustainability and resource conservation.

1. Mishra et al. (2020) presented an IoT-enabled water metering system that combines flow sensors with the ESP32 microcontroller to transmit real-time water usage data to a cloud server. This system allows consumers and utility companies to track water consumption and generate bills without manual intervention, reducing errors and delays. The integration of cloud computing enables advanced data analytics and predictive billing. The study emphasizes the need for scalable and low-cost solutions for widespread urban deployments, especially in smart city projects.
2. Patil et al. (2021) developed a Wi-Fi-based smart water meter leveraging the ESP8266 module, which allows users to remotely monitor their water usage through a web dashboard. The system provides real-time data that helps consumers optimize their water consumption and reduces the chances of over-

billing. The use of Wi-Fi technology ensures that the data can be transmitted over long distances without the need for dedicated infrastructure, making it an ideal solution for residential applications.

3. Kumar et al. (2019) introduced a LoRa-based water metering system, utilizing Long Range Wide Area Network (LoRaWAN) for communication. The low power consumption and long-range capabilities of LoRaWAN are particularly beneficial for large-scale water distribution systems in rural and remote areas. This system significantly reduces the operational cost compared to traditional cellular networks and provides reliable water usage monitoring in locations with limited infrastructure. The system ensures that water consumption data is transmitted continuously without frequent battery changes, making it an environmentally sustainable solution.
4. Rana & Singh (2022) explored an advanced IoT-driven water metering framework that integrates edge computing capabilities. Edge computing enables data processing at the sensor level, reducing the dependency on centralized servers and significantly lowering network latency. This is particularly useful for real-time applications, where quick decision-making is critical. The proposed system processes water usage data locally to identify any irregularities or leaks and sends only summarized information to the central server, minimizing bandwidth usage.
5. Sahoo et al. (2020) designed a Bluetooth Low Energy (BLE)-based smart water meter suitable for short-range communication. BLE technology is ideal for residential applications where the user can conveniently monitor water usage via mobile apps or a local network. The system was developed with an emphasis on user-friendly interfaces and cost-efficiency, ensuring that even small households or individual users could benefit from smart metering technologies without a significant financial burden.
6. Zhang et al. (2021) developed an advanced cloud-integrated smart water meter system. By using platforms such as Google Firebase, the system enables real-time data storage, trend analysis, and usage forecasting. Consumers can view their water consumption data on a mobile application, which allows for self-management and timely billing. The integration with cloud services facilitates big data analytics, enabling utility providers to optimize water resource distribution and predict future demands.

7. Ahmed et al. (2020) implemented an IoT-based billing model that automates water charge deductions based on real-time consumption data stored in the AWS cloud server. This system allows consumers to monitor their usage and adjust accordingly, while utility companies can generate accurate and transparent bills without human intervention. Additionally, the automatic billing feature increases the efficiency of utility companies, as it reduces errors related to manual readings and enables dynamic billing rates based on usage patterns.
8. Sharma et al. (2018) proposed an AI-based predictive billing system that uses machine learning algorithms to forecast future water consumption trends. By analyzing historical data, the system provides dynamic pricing strategies that can encourage consumers to use water more efficiently. This is particularly useful for areas experiencing water scarcity, as the system can be programmed to incentivize lower water consumption during peak demand periods. The study highlights the potential for AI-driven pricing models in achieving long-term water sustainability.
9. Gupta & Verma (2022) designed an MQTT-based water metering system that uses Message Queuing Telemetry Transport (MQTT) for efficient real-time data streaming. The MQTT protocol ensures low-overhead messaging and high-frequency data transmission, which is ideal for applications that require constant monitoring, such as water metering systems. The system allows for low-latency data exchange between water meters and utility providers, ensuring real-time updates and prompt reactions to sudden changes in consumption.
10. Lee et al. (2019) introduced an IoT-powered prepaid water meter that utilizes a blockchain-based payment system. This ensures secure, tamper-proof, and transparent transactions for water usage payments. By combining IoT sensors with blockchain technology, the system creates a decentralized ledger that tracks each water usage transaction, providing audit trails and accountability. Consumers can prepay for water services, with automatic deductions based on real-time consumption, while utility providers benefit from increased security and lower transaction costs.
11. Kim et al. (2017) compared the performance of LoRaWAN, ZigBee, and NB-IoT for smart water metering systems. The study concluded that LoRaWAN is the most effective solution, offering the best range-to-power ratio and low

energy consumption. LoRaWAN provides long-range communication with minimal power usage, making it ideal for urban and rural smart water metering applications. The study highlights the trade-offs between data transfer speed, energy consumption, and coverage area in choosing the most appropriate communication technology.

12. Kumar & Prasad (2021) examined GSM-based remote metering systems, where cellular networks are used to transmit water usage data to utility servers. While GSM offers reliable communication in urban areas, it incurs higher operational costs due to data transmission fees. The study suggests that GSM may be a suitable option for areas with existing GSM infrastructure, but it may not be cost-effective for large-scale deployments.
13. Chen et al. (2019) developed a hybrid system that combines LPWAN with GSM for remote water monitoring in rural areas. The hybrid approach improves coverage and optimizes power efficiency, ensuring reliable communication in regions where infrastructure is limited. This system leverages the strengths of both LPWAN's low-power long-range communication and GSM's reliability, providing a balanced solution for remote and off-grid areas.
14. Ali et al. (2020) explored the potential of 5G-enabled smart water metering systems. With the advent of 5G networks, the technology promises high-speed data transfer with minimal latency, enabling real-time water monitoring and instant notifications. The study emphasizes the role of 5G in enabling advanced smart city applications, where water meters can continuously update usage data to optimize distribution and ensure efficient water management.
15. Wu et al. (2022) investigated satellite-based IoT communication for water metering in disaster-prone and remote areas. In regions where terrestrial communication infrastructure is compromised due to natural disasters, satellite-based IoT systems provide uninterrupted connectivity, ensuring continuous water monitoring and management. This solution offers reliable and resilient communication in situations where other communication networks fail.

CHAPTER 3

EXISTING SYSTEM

In today's world, the efficient management of water resources has become increasingly crucial due to the growing global population, industrial demands, and the escalating effects of climate change. As water becomes an increasingly scarce and valuable resource, it is vital to ensure that it is managed, monitored, and consumed efficiently. Traditional water metering systems, however, often face significant challenges that hinder their effectiveness. These challenges include inaccuracies in readings, human errors during manual meter reading, and inefficient or outdated billing processes, which can lead to both overcharging and undercharging for water usage.For instance, manual water meters rely on periodic physical readings by utility workers, which can result in human errors, such as misreading or incorrect data recording. Furthermore, these systems are prone to delays, as consumers are often billed based on estimated consumption, leading to discrepancies between actual usage and the billed amount. Additionally, traditional systems may not provide real-time data, making it difficult for both consumers and utility providers to monitor and manage water usage effectively. This lack of timely information can contribute to water wastage, billing issues, and delayed responses to problems such as leaks or excessive consumption.To address these challenges, the Internet of Things (IoT) has emerged as a transformative tool in water management. IoT refers to the network of connected devices that communicate and exchange data over the internet, enabling real-time monitoring, automation, and control. In the context of water management, IoT-based smart water metering systems offer a more efficient and accurate way to track water usage and improve billing processes.

By integrating IoT with smart water meters, data on water consumption can be collected and transmitted in real-time, offering both consumers and utility providers immediate access to accurate usage data. This data is sent to centralized cloud servers, where it can be processed, analyzed, and used to generate real-time bills based on actual consumption, eliminating the need for estimated billing. Consumers can also access this data remotely via mobile apps or web dashboards, allowing them to monitor their usage, set thresholds, and receive notifications when they approach their consumption limits.IoT-based systems can be equipped with additional features such as leak detection, anomaly alerts,

and predictive analytics, enabling both consumers and utility providers to take proactive steps in addressing issues before they escalate. For example, IoT sensors can detect unusual patterns of water usage, which may indicate leaks or faults, and send instant notifications to both parties to prevent unnecessary wastage or damage.

Overall, IoT is revolutionizing water management by enhancing the accuracy, efficiency, and sustainability of water metering systems. These advancements ensure that water resources are used more wisely, bills are more accurate, and both consumers and utility providers can make informed decisions based on real-time data. With the ongoing adoption of IoT in water management, the future holds great potential for achieving a more sustainable and optimized approach to managing this precious resource.

CHAPTER 4

PROPOSED SYSTEM

In the proposed system, the consumer will have the ability to manage their water consumption in real-time, making informed decisions based on data provided by the Arduino Uno microcontroller. The system utilizes IoT technology to enable two-way communication between the consumer and the utility provider, promoting efficient water usage and automated billing. The integration of water flow sensors with the Arduino Uno ensures that consumption data is continuously monitored and transmitted to a centralized server, providing up-to-date usage statistics.

The Arduino Uno microcontroller processes water consumption data from the flow sensors and displays it on the LCD screen. Users can monitor their real-time water usage, promoting awareness and encouraging efficient consumption. The system incorporates IoT connectivity, enabling consumers to access their water usage data remotely via a web dashboard. This allows users to track their consumption and take timely actions to avoid wastage or overuse. The system automatically calculates the water consumption cost based on the readings from the flow sensors. This data is transmitted to the utility provider's server, where bills are generated and sent to the consumer. The system eliminates the need for manual meter reading and ensures accurate, automated billing.

The system allows consumers to set a threshold for water usage. If the water consumption nears the set threshold, the system sends an alert to the consumer, prompting them to take necessary actions. This feature ensures that consumers remain aware of their usage and helps avoid unexpected billing spikes. In cases of excessive water consumption or non-payment, the system can trigger a pump control mechanism to regulate or cut off the water supply. This ensures that the water resources are used efficiently, and non-paying consumers are held accountable for their usage.

The system is equipped with an anomaly detection feature that alerts both the consumer and the utility provider in the event of meter tampering. This feature helps prevent fraudulent activities and ensures the integrity of the system.

The Arduino Uno microcontroller serves as the central controller for the system. It continuously monitors water consumption using flow sensors and displays the consumption data on the LCD display. The Arduino Uno processes the data and transmits

it to the central server via Wi-Fi, ensuring real-time communication between the consumer and the utility provider.

Consumers can access the system through a web interface to set their desired threshold for water consumption. When consumption approaches the set limit, the system sends a notification to the consumer. If the consumer does not take action, the system will automatically shut off the water supply. To restore service, the consumer can adjust the threshold value via the webpage. The system calculates water usage automatically and sends the data to the utility provider, allowing for automated billing. On the first day of each month, the monthly water consumption bill is sent to the consumer via SMS or email.

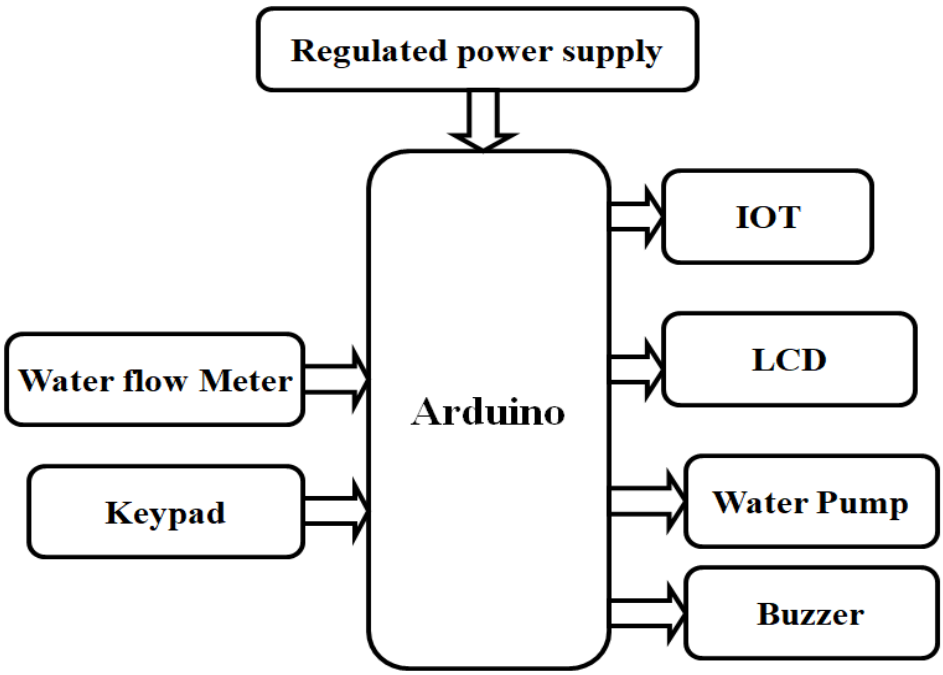


Figure: 4.1 flow diagram

Figure 4.1: this figure shows the flow diagram of the smart water meter.

Arduino Uno Microcontroller: Responsible for data processing, communication, and controlling the water meter and pump.Flow Sensor: Measures the volume of water consumed and provides data to the Arduino.LCD Display: Shows real-time water usage for consumers.Wi-Fi Module (ESP8266): Ensures connectivity between the Arduino Uno and the server for real-time data transmission.Pump Control Mechanism: Regulates the water supply based on consumption or payment status.Web Dashboard: Allows consumers to monitor usage, set thresholds, and pay bills.

Schematic Diagram:

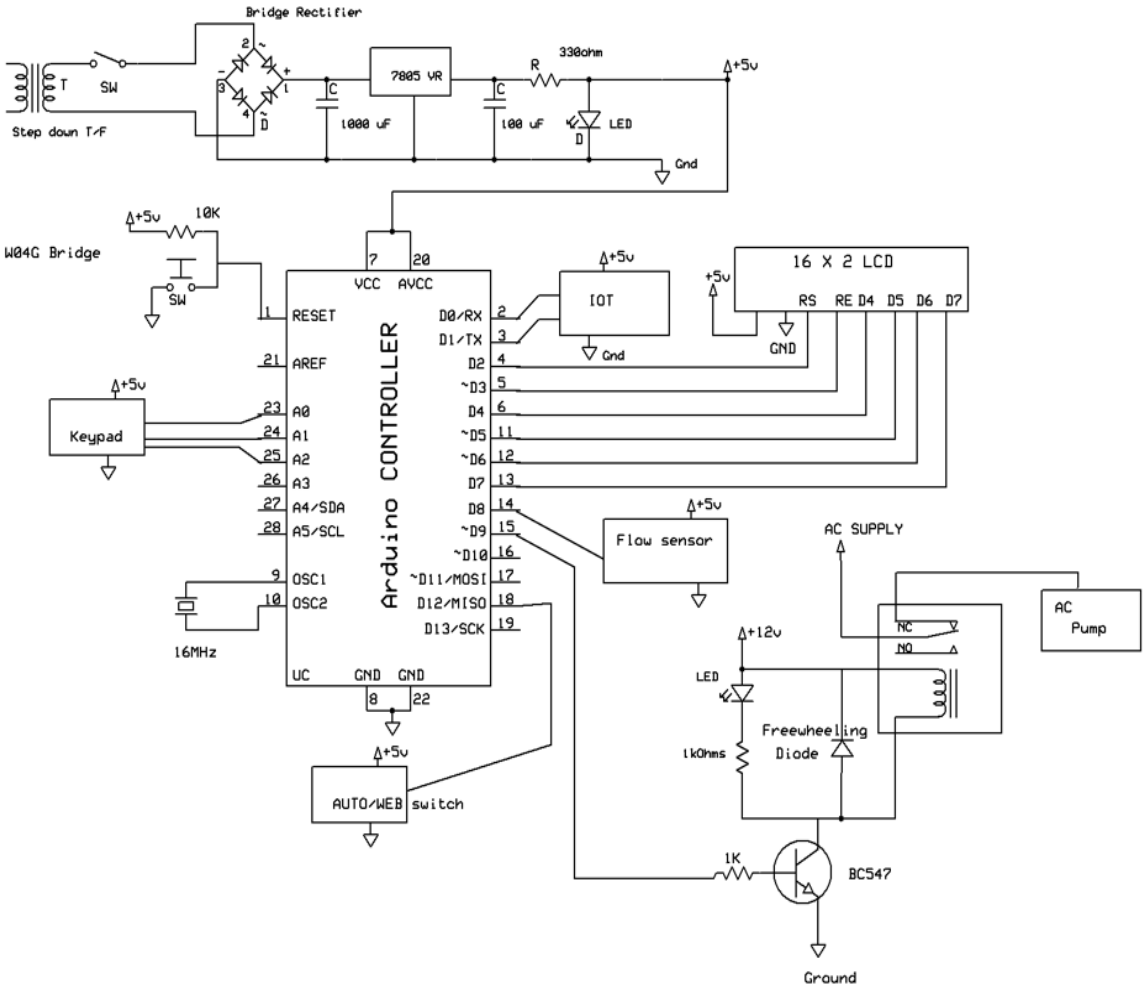


Figure 4.2 schematic diagram of smart water meter

Figure 4.2 : this figure shows the schematic diagram of smart water meter.

CHAPTER 5
EMBEDDED SYSTEMS

5.1 EMBEDDED SYSTEMS:

An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

Embedded systems are controlled by one or more main processing cores that are typically either microcontrollers or digital signal processors (DSP). The key characteristic, however, is being dedicated to handle a particular task, which may require very powerful processors. For example, air traffic control systems may usefully be viewed as embedded, even though they involve mainframe computers and dedicated regional and national networks between airports and radar sites. (Each radar probably includes one or more embedded systems of its own.)

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not a strictly definable term, as most systems have some element of extensibility or programmability. For example, handheld computers share some elements with embedded systems such as the operating systems and microprocessors which power them, but they allow different applications to be loaded and peripherals to be connected. Moreover, even systems which don't expose programmability as a primary feature generally need to support software updates.

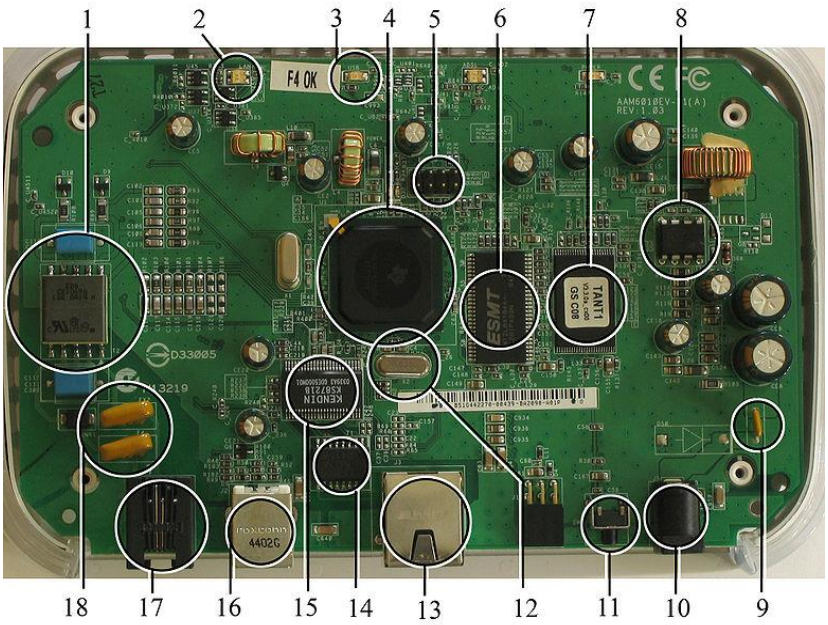


Fig 5.1:A modern example of embedded system

Figure 5.1: this figure shoes the modern example of embedded system.

Labeled parts include microprocessor (4), RAM (6), flash memory (7).Embedded systems programming is not like normal PC programming. In many ways, programming for an embedded system is like programming PC 15 years ago. The hardware for the system is usually chosen to make the device as cheap as possible. Spending an extra dollar a unit in order to make things easier to program can cost millions. Hiring a programmer for an extra month is cheap in comparison. This means the programmer must make do with slow processors and low memory, while at the same time battling a need for efficiency not seen in most PC applications. Below is a list of issues specific to the embedded field.

5.2 NEED FOR EMBEDDED SYSTEMS:

The uses of embedded systems are virtually limitless, because every day new products are introduced to the market that utilizes embedded computers in novel ways. In recent years, hardware such as microprocessors, microcontrollers, and FPGA chips have become much cheaper. So when implementing a new form of control, it's wiser to just buy the generic chip and write your own custom software for it. Producing a custom-made chip to handle a particular task or set of tasks costs far more time and money. Many embedded computers even come with extensive libraries, so that "writing your own software" becomes a very trivial task indeed. From an implementation viewpoint, there is a major difference between a computer and an embedded system. Embedded systems are often required to provide

Real-Time response. The main elements that make embedded systems unique are its reliability and ease in debugging.

5.2.1 Debugging:

Embedded debugging may be performed at different levels, depending on the facilities available. From simplest to most sophisticate they can be roughly grouped into the following areas:

- Interactive resident debugging, using the simple shell provided by the embedded operating system (e.g. Forth and Basic)
- External debugging using logging or serial port output to trace operation using either a monitor in flash or using a debug server like the Remedy Debugger which even works for heterogeneous multi core systems.
- An in-circuit debugger (ICD), a hardware device that connects to the microprocessor via a JTAG or Nexus interface. This allows the operation of the microprocessor to be controlled externally, but is typically restricted to specific debugging capabilities in the processor.
- An in-circuit emulator replaces the microprocessor with a simulated equivalent, providing full control over all aspects of the microprocessor.
- A complete emulator provides a simulation of all aspects of the hardware, allowing all of it to be controlled and modified and allowing debugging on a normal PC.
- Unless restricted to external debugging, the programmer can typically load and run software through the tools, view the code running in the processor, and start or stop its operation. The view of the code may be as assembly code or source-code.

Because an embedded system is often composed of a wide variety of elements, the debugging strategy may vary. For instance, debugging a software(and microprocessor) centric embedded system is different from debugging an embedded system where most of the processing is performed by peripherals (DSP, FPGA, co-processor). An increasing number of embedded systems today use more than one single processor core. A common problem with multi-core development is the proper synchronization of software execution. In such a case, the embedded system design may wish to check the data traffic on the busses between the processor cores, which requires very low-level debugging, at signal/bus level, with a logic analyzer, for instance.

5.2.2 Reliability:

Embedded systems often reside in machines that are expected to run continuously for years without errors and in some cases recover by them if an error occurs. Therefore the software is usually developed and tested more carefully than that for personal computers, and unreliable mechanical moving parts such as disk drives, switches or buttons are avoided.
Specific reliability issues may include:

- The system cannot safely be shut down for repair, or it is too inaccessible to repair. Examples include space systems, undersea cables, navigational beacons, bore-hole systems, and automobiles.
- The system must be kept running for safety reasons. "Limp modes" are less tolerable. Often backups are selected by an operator. Examples include aircraft navigation, reactor control systems, safety-critical chemical factory controls, train signals, engines on single-engine aircraft.
- The system will lose large amounts of money when shut down: Telephone switches, factory controls, bridge and elevator controls, funds transfer and market making, automated sales and service.

A variety of techniques are used, sometimes in combination, to recover from errors—both software bugs such as memory leaks, and also soft errors in the hardware:

- Watchdog timer that resets the computer unless the software periodically notifies the watchdog
- Subsystems with redundant spares that can be switched over to
- software "limp modes" that provide partial function
- Designing with a Trusted Computing Base (TCB) architecture[6] ensures a highly secure & reliable system environment
- An Embedded Hypervisor is able to provide secure encapsulation for any subsystem component, so that a compromised software component cannot interfere with other subsystems, or privileged-level system software. This encapsulation keeps faults from propagating from one subsystem to another, improving reliability. This may also allow a subsystem to be automatically shut down and restarted on fault detection.
- Immunity Aware Programming

5.3 EXPLANATION OF EMBEDDED SYSTEMS:

5.3.1 Software Architecture:

There are several different types of software architecture in common use.

- Simple Control Loop:

In this design, the software simply has a loop. The loop calls subroutines, each of which manages a part of the hardware or software.

- Interrupt Controlled System:

Some embedded systems are predominantly interrupting controlled. This means that tasks performed by the system are triggered by different kinds of events. An interrupt could be generated for example by a timer in a predefined frequency, or by a serial port controller receiving a byte. These kinds of systems are used if event handlers need low latency and the event handlers are short and simple.

Usually these kinds of systems run a simple task in a main loop also, but this task is not very sensitive to unexpected delays. Sometimes the interrupt handler will add longer tasks to a queue structure. Later, after the interrupt handler has finished, these tasks are executed by the main loop. This method brings the system close to a multitasking kernel with discrete processes.

- Cooperative Multitasking:

A non-preemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in an API. The programmer defines a series of tasks, and each task gets its own environment to “run” in. When a task is idle, it calls an idle routine, usually called “pause”, “wait”, “yield”, “nop” (stands for no operation), etc.The advantages and disadvantages are very similar to the control loop, except that adding new software is easier, by simply writing a new task, or adding to the queue-interpreter.

- Primitive Multitasking:

In this type of system, a low-level piece of code switches between tasks or threads based on a timer (connected to an interrupt). This is the level at which the system is generally considered to have an "operating system" kernel. Depending on how much functionality is required, it introduces more or less of the complexities of managing multiple tasks running conceptually in parallel.

As any code can potentially damage the data of another task (except in larger systems using an MMU) programs must be carefully designed and tested, and access to shared data must

be controlled by some synchronization strategy, such as message queues, semaphores or a non-blocking synchronization scheme.

Because of these complexities, it is common for organizations to buy a real-time operating system, allowing the application programmers to concentrate on device functionality rather than operating system services, at least for large systems; smaller systems often cannot afford the overhead associated with a generic real time system, due to limitations regarding memory size, performance, and/or battery life.

- Microkernels And Exokernels:

A microkernel is a logical step up from a real-time OS. The usual arrangement is that the operating system kernel allocates memory and switches the CPU to different threads of execution. User mode processes implement major functions such as file systems, network interfaces, etc.

In general, microkernels succeed when the task switching and intertask communication is fast, and fail when they are slow. Exokernels communicate efficiently by normal subroutine calls. The hardware and all the software in the system are available to, and extensible by application programmers. Based on performance, functionality, requirement the embedded systems are divided into three categories:

5.3.2 Stand Alone Embedded System:

These systems takes the input in the form of electrical signals from transducers or commands from human beings such as pressing of a button etc.., process them and produces desired output. This entire process of taking input, processing it and giving output is done in standalone mode. Such embedded systems comes under stand alone embedded systems

Eg: microwave oven, air conditioner etc..

5.3.3 Real-time embedded systems:

Embedded systems which are used to perform a specific task or operation in a specific time period those systems are called as real-time embedded systems. There are two types of real-time embedded systems.

- Hard Real-time embedded systems:

These embedded systems follow an absolute dead line time period i.e.., if the tasking is not done in a particular time period then there is a cause of damage to the entire equipment.

Eg: consider a system in which we have to open a valve within 30 milliseconds. If this valve is not opened in 30 ms this may cause damage to the entire equipment. So in such cases we use embedded systems for doing automatic operations.

- Soft Real Time embedded systems:

These embedded systems follow a relative dead line time period i.e.., if the task is not done in a particular time that will not cause damage to the equipment.

Eg: Consider a TV remote control system , if the remote control takes a few milliseconds delay it will not cause damage either to the TV or to the remote control. These systems which will not cause damage when they are not operated at considerable time period those systems comes under soft real-time embedded systems.

5.3.4 Network communication embedded systems:

A wide range network interfacing communication is provided by using embedded systems.

Eg:

- Consider a web camera that is connected to the computer with internet can be used to spread communication like sending pictures, images, videos etc.., to another computer with internet connection throughout anywhere in the world.
- Consider a web camera that is connected at the door lock.

Whenever a person comes near the door, it captures the image of a person and sends to the desktop of your computer which is connected to internet. This gives an alerting message with image on to the desktop of your computer, and then you can open the door lock just by clicking the mouse. Fig: 5.2 show the network communications in embedded systems.

Fig 5.3.4: Network communication embedded systems

Figure5.3.4: This figure shows the Network communication embedded Systems

5.3.5 Different types of processing units:

The central processing unit (c.p.u) can be any one of the following microprocessor, microcontroller, digital signal processing.

- Among these Microcontroller is of low cost processor and one of the main advantage of microcontrollers is, the components such as memory, serial communication interfaces, analog to digital converters etc.., all these are built on a single chip. The numbers of external components that are connected to it are very less according to the application.
- Microprocessors are more powerful than microcontrollers. They are used in major applications with a number of tasking requirements. But the microprocessor requires many external components like memory, serial communication, hard disk, input output ports etc.., so the power consumption is also very high when compared to microcontrollers.

CHAPTER 6
HARDWARE DESCRIPTION

6.1 MICRO CONTROLLER:

Fig: 6.1 Microcontrollers

Figure6.1: This figure shoes the Example of Microcontroller chip.

6.1.1 Introduction to Microcontrollers:

Circumstances that we find ourselves in today in the field of microcontrollers had their beginnings in the development of technology of integrated circuits. This development has made it possible to store hundreds of thousands of transistors into one chip.

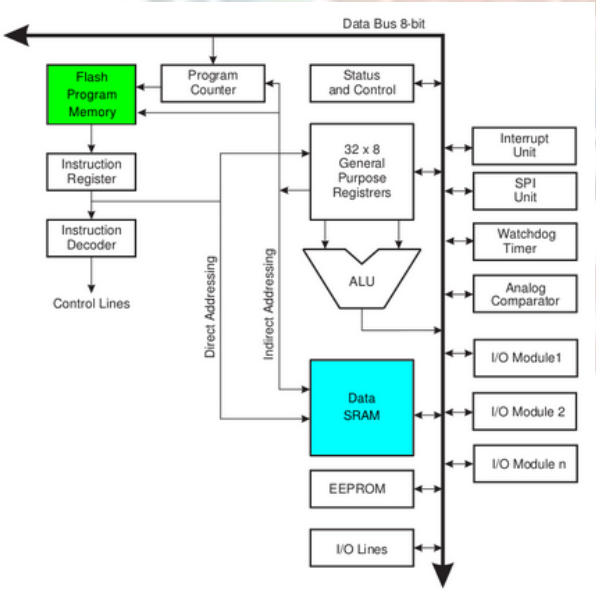


Fig: 6.1.1 Architecture

Figure 6.1.1: This figure shows the Architecture of Microcontroller That was a prerequisite for production of microprocessors, and the first computers were made by adding external peripherals such as memory, input-output lines, timers and other. Further increasing of the volume of the package resulted in creation of integrated circuits. These

integrated circuits contained both processor and peripherals. That is how the first chip containing a microcomputer, or what would later be known as a microcontroller came about.

Microprocessors and microcontrollers are widely used in embedded systems products. Microcontroller is a programmable device. A microcontroller has a CPU in addition to a fixed amount of RAM, ROM, I/O ports and a timer embedded all on a single chip. The fixed amount of on-chip ROM, RAM and number of I/O ports in microcontrollers makes them ideal for many applications in which cost and space are critical.

6.2 AVR-ARDUINO MICROCONTROLLER:

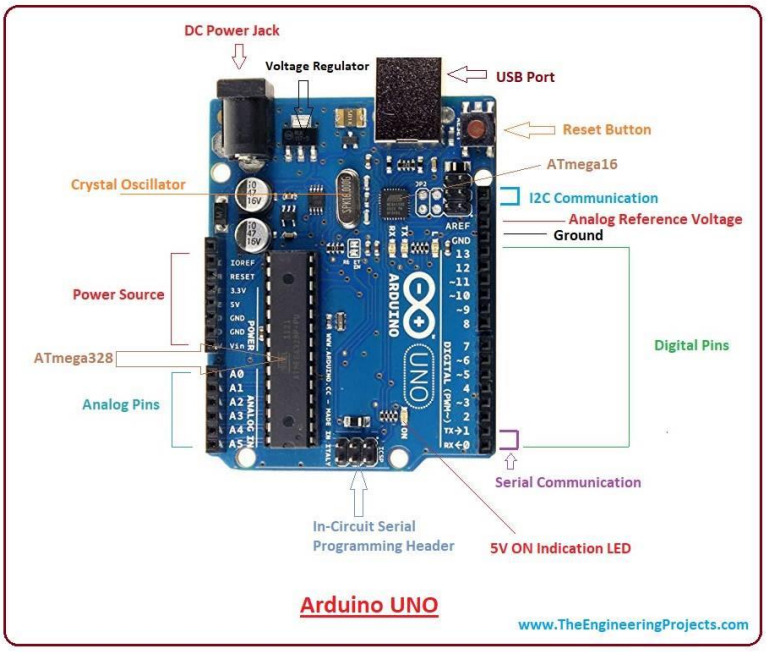


Figure 6.2 ARDUINO Development Board

Figure 6.2: This figure shows the ARDUINO Development Board The AVR is a modified Harvard architecture 8-bit RISC single chip microcontroller which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to One-Time Programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time.

6.3 CRYSTAL OSCILLATOR:

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, Either a quartz

Crystal or a ceramic resonator may be used. The CKOPT Fuse selects between two different Oscillator amplifier modes. When CKOPT is programmed, the Oscillator output will oscillate a full rail-to-rail swing on the output. This mode is suitable when operating in a very noisy environment or when the output from XTAL2 drives a second clock buffer. This mode has a wide frequency range. When CKOPT is unprogrammed, the Oscillator has a smaller output swing. This reduces power consumption considerably.

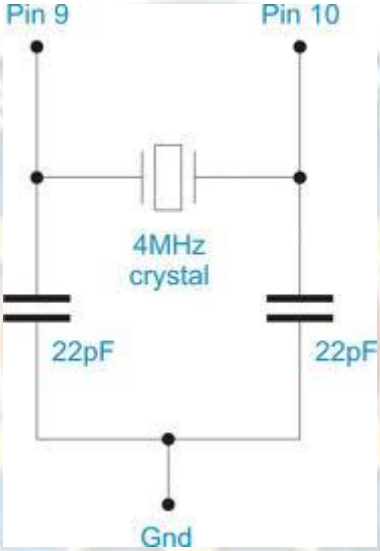


Figure 6.3 crystal oscillator

Figure6.3: This figure shows the crystal oscillator which reduces power consumption.

This mode has a limited frequency range and it cannot be used to drive other clock buffers. For resonators, the maximum frequency is 8 MHz with CKOPT unprogrammed and 16 MHz with CKOPT programmed. C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. For ceramic resonators, the capacitor values given by the manufacturer should be used. The Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..1

6.4 PIN DIAGRAM:

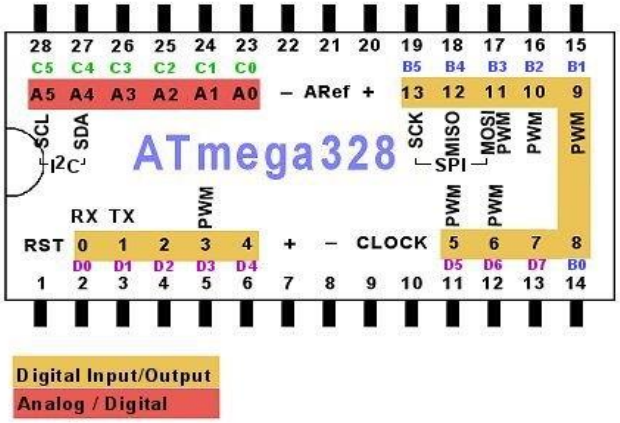


Fig.6.4 PIN DIAGRAM OF ATMEGA328

Figure6.4: This figure shows the PIN DIAGRAM OF ATMEGA328

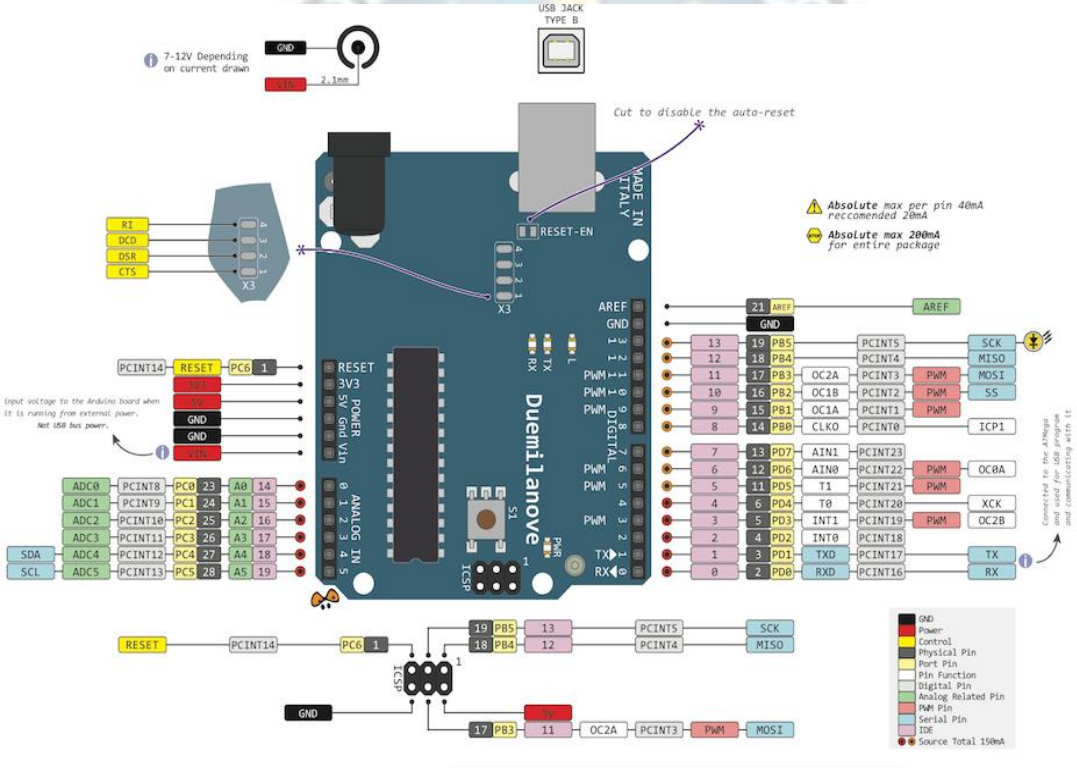


Fig.6.4.2 PIN DIAGRAM OF ATMEGA328

Figure 6.4.2: This figure shows the PIN DIAGRAM OF ATMEGA328.

- **VCC**

  Digital supply voltage magnitude of the voltage range between 4.5 to 5.5 V for the ATmega8 and 2.7 to 5.5 V for ATmega8L

- **GND**

  Ground Zero reference digital voltage supply.

- **PORTB (PB7.. PB0)**
- PORTB is a port I / O two-way (bidirectional) 8-bit with internal pull-up resistor can be selected. This port output buffers have symmetrical characteristics when used as a source or sink. When used as an input, the pull-pin low externally will emit a current if the pull-up resistor is activated it. PORTB pins will be in the condition of the tri-state when RESET is active, although the clock is not running.
- **PORTC (PC5.. PC0)**
- PORTC is a port I / O two-way (bidirectional) 7-bit with internal pull-up resistor can be selected. This port output buffers have symmetrical characteristics when used as a source or sink. When used as an input, the pull-pin low externally will emit a current if the pull-up resistor is activated it. PORTC pins will be in the condition of the tri-state when RESET is active, although the clock is not running.
- **PC6/RESET**

  If RSTDISBL Fuse programmed, PC6 then serves as a pin I / O but with different characteristics. PC0 to PC5 If Fuse RSTDISBL not programmed, then serves as input Reset PC6. LOW signal on this pin with a minimum width of 1.5 microseconds will bring the microcontroller into reset condition, although the clock is not running.
- **PORTD (PD7.. PD0)**
- PORTD is a port I / O two-way (bidirectional) 8-bit with internal pull-up resistor can be selected. This port output buffers have symmetrical characteristics when used as a source or sink. When used as an input, the pull-pin low externally will emit a current if the pull-up resistor is activated it. PORTD pins will be in the condition of the tri-state when RESET is active, although the clock is not running.
- **RESET**

  Reset input pin. LOW signal on this pin with a minimum width of 1.5 microseconds will bring the microcontroller into reset condition, although the clock is not running. Signal with a width of less than 1.5 microseconds does not guarantee a Reset condition.
- **AVCC**

  AVCC is the supply voltage pin for the ADC, PC3 .. PC0, and ADC7..ADC6. This pin should be connected to VCC, even if the ADC is not used. If the ADC is used, AVCC should be connected to VCC through a low-pass filter to reduce noise.

- **Aref**

  Analog Reference pin for the ADC.

- **ADC7 .. ADC6**

ADC analog input there is only on ATmega8 with TQFP and QFP packages / MLF.

- **PORTS**

Term "port" refers to a group of pins on a microcontroller which can be accessed simultaneously, or on which we can set the desired combination of zeros and ones, or read from them an existing status. Physically, port is a register inside a microcontroller which is connected by wires to the pins of a microcontroller. Ports represent physical connection of Central Processing Unit with an outside world. Microcontroller uses them

The Atmega8 has 23 I/O ports which are organized into 3 groups:

- Port B (PB0 to PB7)
- Port C (PC0 to PC6)
- Port D (PD0 to PD7)

We will use mainly 3 registers known as **DDRX, PORTX** & **PINX**. We have total four PORTs on my ATmega16. They are **PORTA, PORTB, PORTC** and **PORTD**. They are multifunctional pins. Each of the pins in each port (total 32) can be treated as input or output pin.

Applications

- AVR microcontroller perfectly fits many uses, from automotive industries and controlling home appliances to industrial instruments, remote sensors, electrical door locks and safety devices. It is also ideal for smart cards as well as for battery supplied devices because of its low consumption.
- EEPROM memory makes it easier to apply microcontrollers to devices where permanent storage of various parameters is needed (codes for transmitters, motor speed, receiver frequencies, etc.). Low cost, low consumption, easy handling and flexibility make ATmega8 applicable even in areas where microcontrollers had not previously been considered (example: timer functions, interface replacement in larger systems, coprocessor applications, etc.).

- In System Programmability of this chip (along with using only two pins in data transfer) makes possible the flexibility of a product, after assembling and testing have been completed. This capability can be used to create assembly-line production, to store calibration data available only after final testing, or it can be used to improve programs on finished products.

The ESP32 is just the name of the chip. Device manufacturers and developers have three different format choices, and the decision of which one to go with will depend on their individual circumstances:

- **ESP32 chip**: This is the bare-bones chip that is manufactured by Espressif. It comes unshielded, meaning there's no protective casing, and it can't be attached to a module or board without soldering. Therefore, most device manufacturers do not purchase just the chip, as this will add an additional layer of complexity to the production process.

- **ESP32 modules**: These are surface mountable modules that contain the chip. The modules are essentially small electrical components that can be attached to a circuit board. The benefit here is that you can easily mount these modules onto an MCU board. The chip is also usually shielded and pre-approved by the FCC, which means device manufacturers do not need to worry about adding additional steps to the production process to achieve FCC compliance regarding Wi-Fi shielding.

- **ESP32 development boards:** These are IoT MCU development boards that have the modules containing the ESP32 chip preinstalled. They are used by hobbyists, device manufacturers and developers to test and prototype IoT devices before entering mass production. There is a wide variety of makes and models of ESP32 development boards, produced by different manufacturers. Here are some important specs to consider when choosing a suitable IoT ESP32 development board:

    - GPIO pins
    - ADC pins
    - Wi-Fi antennas
    - LEDs

- Shielding*
- Flash memory

Many international markets require shielded Wi-Fi devices, as Wi-Fi produces a lot of radio frequency interference (RFI), and shielding minimizes this interference. This should, therefore, be a key consideration for all developers and embedded device manufacturers

6.5 RELAY:

A relay is an electrically operated switch. Many relays use an electromagnet to operate a switching mechanism, but other operating principles are also used. Relays find applications where it is necessary to control a circuit by a low-power signal, or where several circuits must be controlled by one signal. The first relays were used in long distance telegraph circuits, repeating the signal coming in from one circuit and re-transmitting it to another. Relays found extensive use in telephone exchanges and early computers to perform logical operations. A type of relay that can handle the high power required to directly drive an electric motor is called a contactor. Solid-state relays control power circuits with no moving parts, instead using a semiconductor device triggered by light to perform switching. Relays with calibrated operating characteristics and sometimes multiple operating coils are used to protect electrical circuits from overload or faults; in modern electric power systems these functions are performed by digital instruments still called "protection relays".

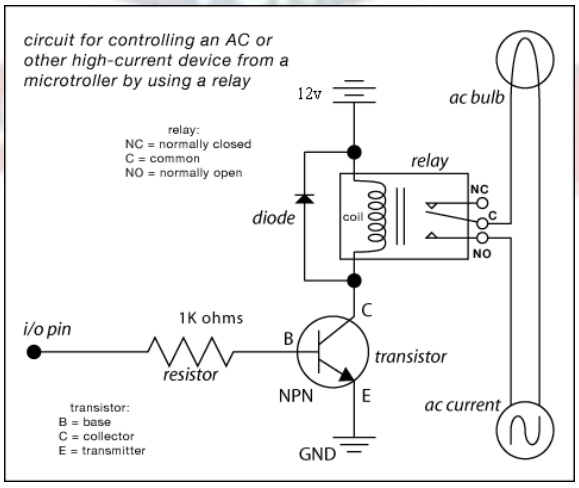


Fig :6.5 Relay circuit

Figure6.5: This figure shows the Relay Circuit.

Relay Driver:

The current needed to operate the relay coil is more than can be supplied by most chips (op. amps etc), so a transistor is usually needed, as shown in the diagram below.

Use BC109C or similar. A resistor of about 4k7 will probably be alright. The diode is needed to short circuit the high voltage "back emf" induced when current flowing through the coil is suddenly switched off.

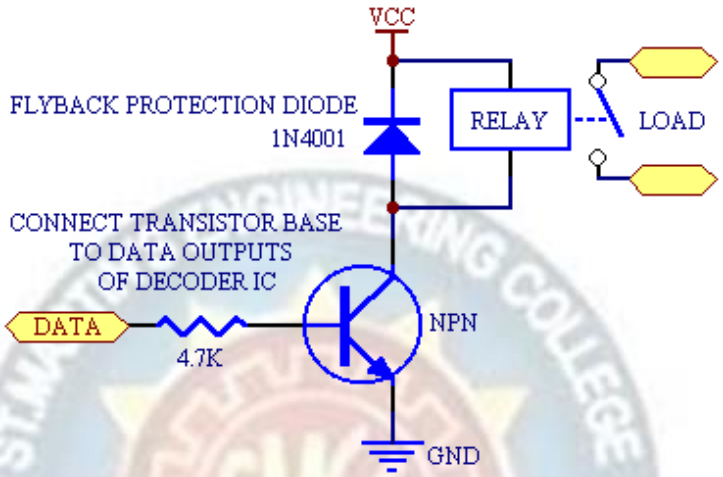


Fig: 6.5.1 Relay Driver

Figure 6.5.1: This figure shows the Relay Driver that is being used in the circuit

6.6 LCD DISPLAY

LCD Background:

One of the most common devices attached to a micro controller is an LCD display. Some of the most common LCD's connected to the many microcontrollers are 16x2 and 20x2 displays. This means 16 characters per line by 2 lines and 20 characters per line by 2 lines, respectively

Basic 16x 2 Characters LCD

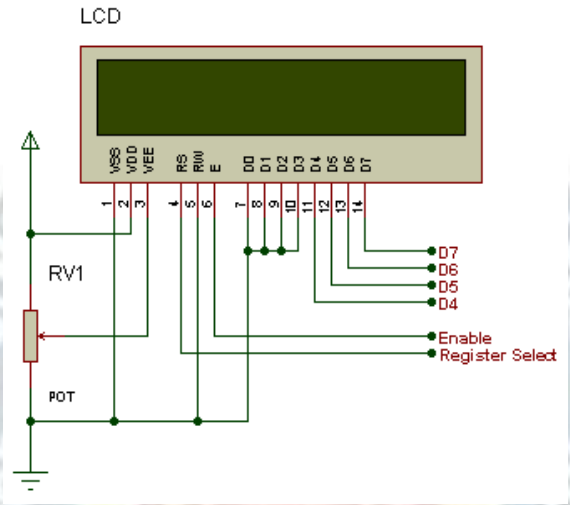


Figure 6.6. LCD Pin diagram

Figure6.6: this figure show the Pin Diagram of LCD.

Pin description:

Pin No.	Name	Description
Pin no. 1	**VSS**	Power supply (GND)
Pin no. 2	**VCC**	Power supply (+5V)
Pin no. 3	**VEE**	Contrast adjust
Pin no. 4	**RS**	0 = Instruction input 1 = Data input
Pin no. 5	**R/W**	0 = Write to LCD module 1 = Read from LCD module
Pin no. 6	**EN**	Enable signal
Pin no. 7	**D0**	Data bus line 0 (LSB)

Pin no. 8	**D1**	Data bus line 1
Pin no. 9	**D2**	Data bus line 2
Pin no. 10	**D3**	Data bus line 3
Pin no. 11	**D4**	Data bus line 4
Pin no. 12	**D5**	Data bus line 5
Pin no. 13	**D6**	Data bus line 6
Pin no. 14	**D7**	Data bus line 7 (MSB)

Table 1: Character LCD pins with Microcontroller

Table1 : this figure shows the Table of Character LCD pins with Microcontroller

The LCD requires 3 control lines as well as either 4 or 8 I/O lines for the data bus. The user may select whether the LCD is to operate with a 4-bit data bus or an 8-bit data bus. If a 4-bit data bus is used the LCD will require a total of 7 data lines (3 control lines plus the 4 lines for the data bus). If an 8-bit data bus is used the LCD will require a total of 11 data lines (3 control lines plus the 8 lines for the data bus).

The three control lines are referred to as **EN**, **RS**, and **RW**.

The **EN** line is called "Enable." This control line is used to tell the LCD that we are sending it data. To send data to the LCD, our program should make sure this line is low (0) and then set the other two control lines and/or put data on the data bus. When the other lines are completely ready, bring **EN** high (1) and wait for the minimum amount of time required by the LCD datasheet (this varies from LCD to LCD), and end by bringing it low (0) again.

The **RS** line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is text data which should be displayed on the screen. For example, to display the letter "T" on the screen we would set RS high.

The **RW** line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands--so RW will almost always be low.

Finally, the data bus consists of 4 or 8 lines (depending on the mode of operation selected by the user). In the case of an 8-bit data bus, the lines are referred to as DB0, DB1, DB2, DB3, DB4, DB5, DB6, and DB7.

Schematic:

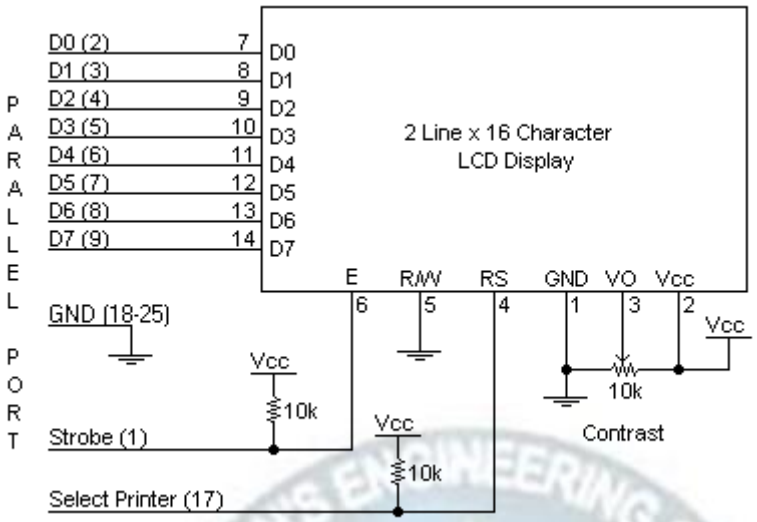


Figure 6.6.2 schematic LCD diagram

Figure 6.6.2 : this figure shows the schematic LCD diagram and its pin connections.

6.7 BUZZER

Basically, the sound source of a piezoelectric sound component is a piezoelectric diaphragm. A piezoelectric diaphragm consists of a piezoelectric ceramic plate which has electrodes on both sides and a metal plate (brass or stainless steel, etc.). A piezoelectric ceramic plate is attached to a metal plate with adhesives. Applying D.C. voltage between electrodes of a piezoelectric diaphragm causes mechanical distortion due to the piezoelectric effect. For a misshaped piezoelectric element, the distortion of the piezoelectric element expands in a radial direction. And the piezoelectric diaphragm bends toward the direction. The metal plate bonded to the piezoelectric element does not expand. Conversely, when the piezoelectric element shrinks, the piezoelectric diaphragm bends in the direction Thus, when AC voltage is applied across electrodes, the bending is repeated, producing sound waves in the air. To interface a buzzer the standard transistor interfacing circuit is used. Note that if adifferent power supply is used for the buzzer, the 0V rails of each power supply mustbe connected to provide a common reference.

If a battery is used as the power supply, it is worth remembering that piezo sounders draw much less current than buzzers. Buzzers also just have one ‘tone’, whereas a piezo sounder is able to create sounds of many different tones. To switch on buzzer -high 1 To switch off buzzer -low 1

Fig: 6.7 Picture of buzzer

Figure 6.7: This figure shows the Picture of Buzzer used in the circuit.

6.8. IOT-ESP8266 MODULE

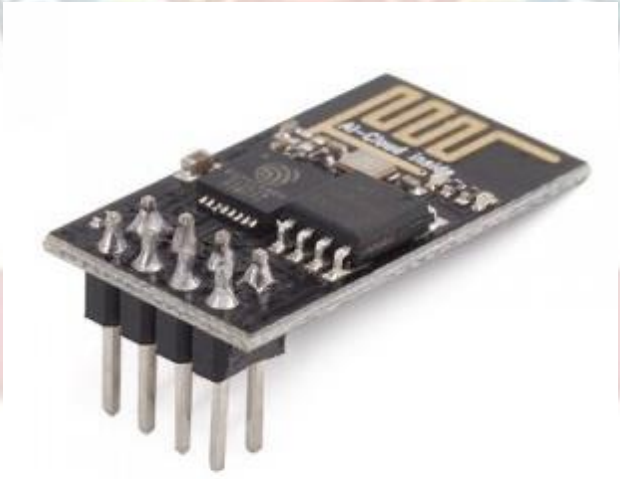


Figure 6.8 . IOT-ESP8266 MODULE

Figure 6.8: This figure shows the IOT-ESP8266 MODULE.

1. Introduction

Espressif Systems' Smart Connectivity Platform (ESCP) of high performance wireless SOCs, for mobile platform designers, provides unsurpassed ability to

embed Wi-Fi capabilities within other systems, at the lowest cost with the greatest functionality

2. Technology Overview

ESP8266 offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to offload all Wi-Fi networking functions from another application processor.

When ESP8266 hosts the application, and when it is the only application processor in the device, it is able to boot up directly from an external flash. It has integrated cache to improve the performance of the system in such applications, and to minimize the memory requirements.

Alternately, serving as a Wi-Fi adapter, wireless internet access can be added to any microcontroller-based design with simple connectivity through UART interface or the CPU AHB bridge interface.

ESP8266 on-board processing and storage capabilities allow it to be integrated with the sensors and other application specific devices through its GPIOs with minimal development up-front and minimal loading during runtime. With its high degree of on-chip integration, which includes the antenna switch balun, power management converters, it requires minimal external circuitry, and the entire solution, including front-end module, is designed to occupy minimal PCB area.

Sophisticated system-level features include fast sleep/wake context switching for energy-efficient VoIP, adaptive radio biasing for low-power operation, advance signal processing, and spur cancellation and radio co-existence features for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation.

AC 230V COOLER PUMP

A pump is a device that moves fluids (liquids or gases), or sometimes slurries, by mechanical action. Pumps can be classified into three major groups according to the method they use to move the fluid: direct lift, displacement, and gravity pumps.

Pumps operate by some mechanism (typically reciprocating or rotary), and consume energy to perform mechanical work moving the fluid. Pumps operate via many energy sources, including manual operation, electricity, engines, or wind power, come in many sizes, from microscopic for use in medical applications to large industrial pumps.

Figure 6.8.2 PUMP

Figure 6.8.2 shoes the AC 230 V cooler pump.

Mechanical pumps serve in a wide range of applications such as pumping water from wells, aquarium filtering, pond filtering and aeration, in the car industry for water-cooling and fuel injection, in the energy industry for pumping oil and natural gas or for operating cooling towers. In the medical industry, pumps are used for biochemical processes in developing and manufacturing medicine, and as artificial replacements for body parts, in particular the artificial heart and penile prosthesis.

When a casing contains only one revolving impeller, it is called a single-stage pump. When a casing contains two or more revolving impellers, it is called a double- or multi-stage pump.

In biology, many different types of chemical and biomechanical pumps have evolved; biomimicry is sometimes used in developing new types of mechanical pumps.

Rotary positive displacement pumps

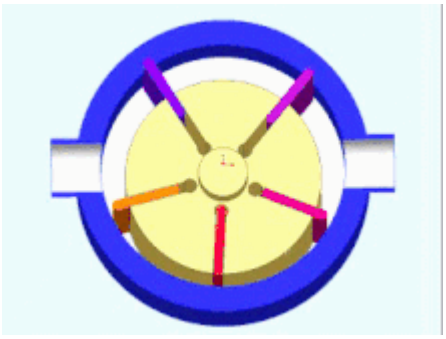


Figure 6.8.3 Rotary displacement pump.

Figure 6.8.3 This figure shows the Rotary positive displacement of pumpsThese pumps move fluid using a rotating mechanism that creates a vacuum that captures and draws in the liquid

Advantages:

Rotary pumps are very efficient because they can handle highly viscous fluids with higher flow rates as viscosity increases.

Drawbacks: The nature of the pump requires very close clearances between the rotating pump and the outer edge, making it rotate at a slow, steady speed. If rotary pumps are operated at high speeds, the fluids cause erosion, which eventually causes enlarged clearances that liquid can pass through, which reduces efficiency.

Rotary positive displacement pumps fall into three main types:

- Gear pumps – a simple type of rotary pump where the liquid is pushed between two gears
- Screw pumps – the shape of the internals of this pump is usually two screws turning against each other to pump the liquid
- Rotary vane pumps – similar to scroll compressors, these have a cylindrical rotor encased in a similarly shaped housing. As the rotor orbits, the vanes trap fluid between the rotor and the casing, drawing the fluid through the pump.

Flow Sensor:

Measure liquid/water flow for your solar, water conservation systems, storage tanks, water recycling home applications, irrigation systems and much more. The sensors are solidly constructed and provide a digital pulse each time an amount of water passes through the

pipe. The output can easily be connected to a microcontroller for monitoring water usage and calculating the amount of water remaining in a tank etc.

Figure 6.8.4 FLOW SENSOR

Figure 6.8.4 we see the flow sensor that measures the water flow rates.

Specification

Working voltage	5V-24V
Maximum current	15 mA（DC 5V）
Weight	43 g
External diameters	20mm
Flow rate range	1～30 L/min
Operating temperature	0℃～80℃
Liquid temperature	<120℃
Operating humidity	35%～90%RH
Operating pressure	under 1.2Mpa
Store temperature	-25℃～+80℃

CHAPTER 7

SOFTWARE DESCRIPTION

The Arduino Software (IDE) makes it easy to write code and upload it to the board offline. We recommend it for users with poor or no internet connection. This software can be used with any Arduino board.here are currently two versions of the Arduino IDE, one is the IDE 2.0.0.

7.1 ARDUINO IDE – COMPILER

here are currently two versions of the Arduino IDE, one is the IDE 1.x.x and the other is IDE 2.x. The IDE 2.x is new major release that is faster and even more powerful to the IDE 1.x.x. In addition to a more modern editor and a more responsive interface it includes advanced features to help users with their coding and debugging.

The following steps can guide you with using the offline IDE (you can choose either IDE 1.x.x or IDE 2.x):

1. Download and install the Arduino Software IDE:

- **Arduino IDE 1.x.x** (Windows, Mac OS, Linux, Portable IDE for Windows and Linux, ChromeOS).
- Arduino IDE 2.x

2. Connect your Arduino board to your device.

3. Open the Arduino Software (IDE).

The Arduino Integrated Development Environment - or Arduino Software (IDE) - connects to the Arduino boards to upload programs and communicate with them. Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and are saved with the file extension .ino.

Using the offline IDE 1.x.x

The editor contains the four main areas:

1. A **Toolbar with buttons** for common functions and a series of menus. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

2. The **message area**, gives feedback while saving and exporting and also displays errors.

3. The **text editor** for writing your code.

4. The **text console** displays text output by the Arduino Software (IDE), including complete error messages and other information.

The bottom right-hand corner of the window displays the configured board and serial port.

Figure 7.1 arduino 1.8.12 text console

Figure 7.1 shows the text console of the page that has been opened.

The Arduino Software IDE

Now that you are all set up, **let's try to make your board blink!**

5. **Connect your Arduino** or Genuino board to your computer.

6. Now, you need to **select the right core & board**. This is done by navigating to **Tools > Board > Arduino AVR Boards > Board**. Make sure you select the board that you are using. If you cannot find your board, you can add it from **Tools > Board > Boards Manager**.

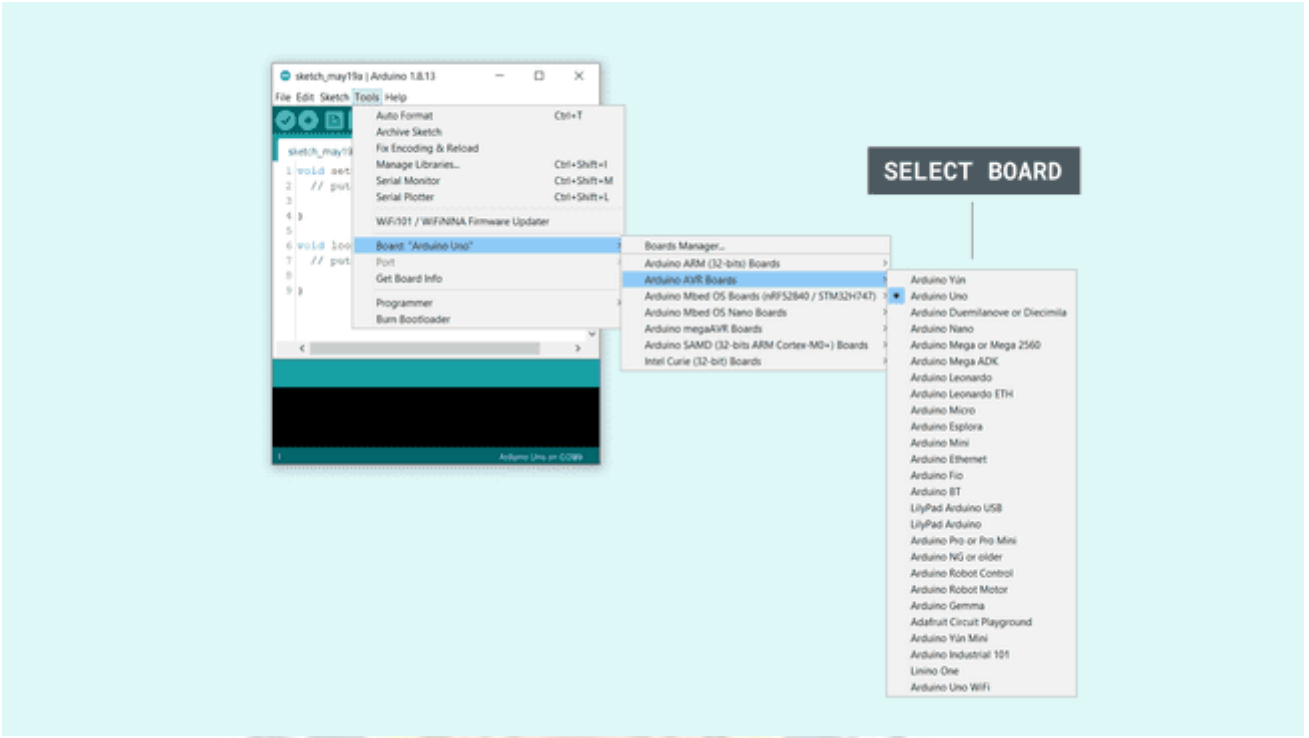


Figure 7.2 Aurduino ADE software.

Figure 7.2 shows Selecting a board on te software.

7. Now, let's make sure that your board is found by the computer, by **selecting the port**. This is simply done by navigating to **Tools > Port**, where you select your board from the list.

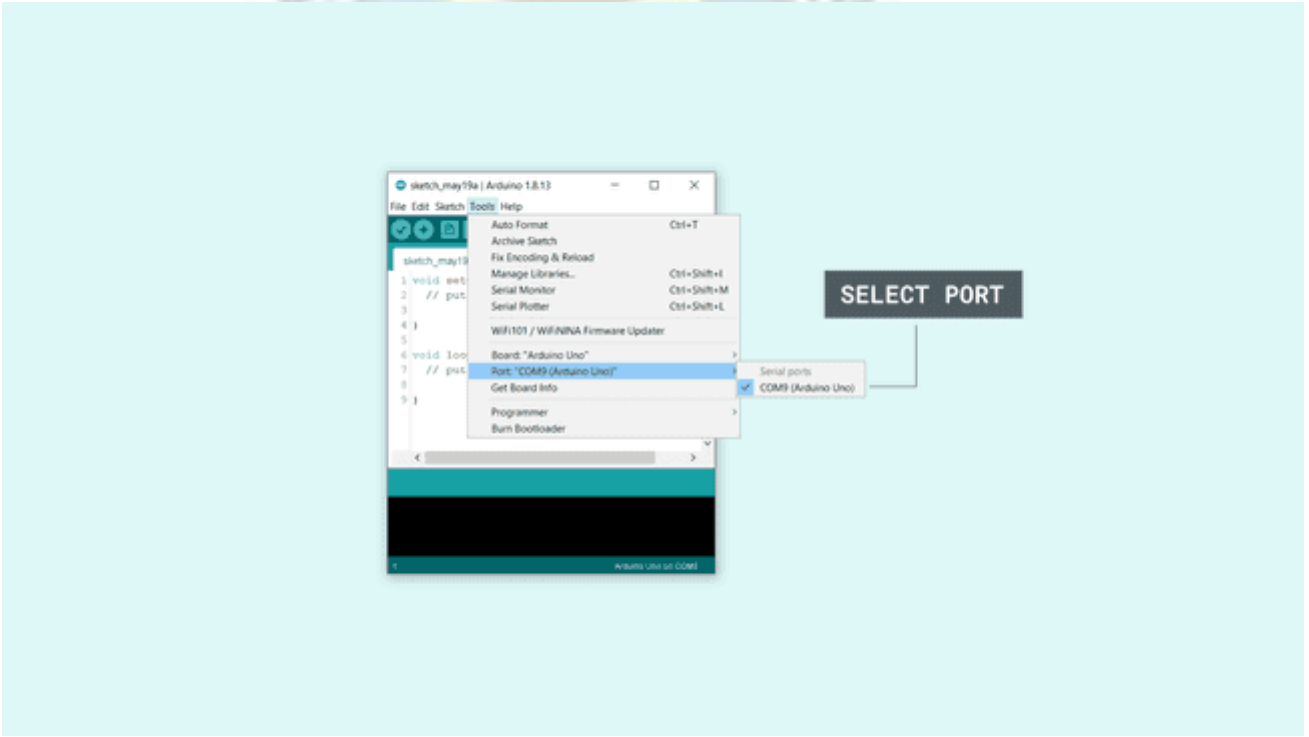


figure 7.3 Selecting the port

8. Let's **try an example**: navigate to **File > Examples > 01.Basics > Blink.**

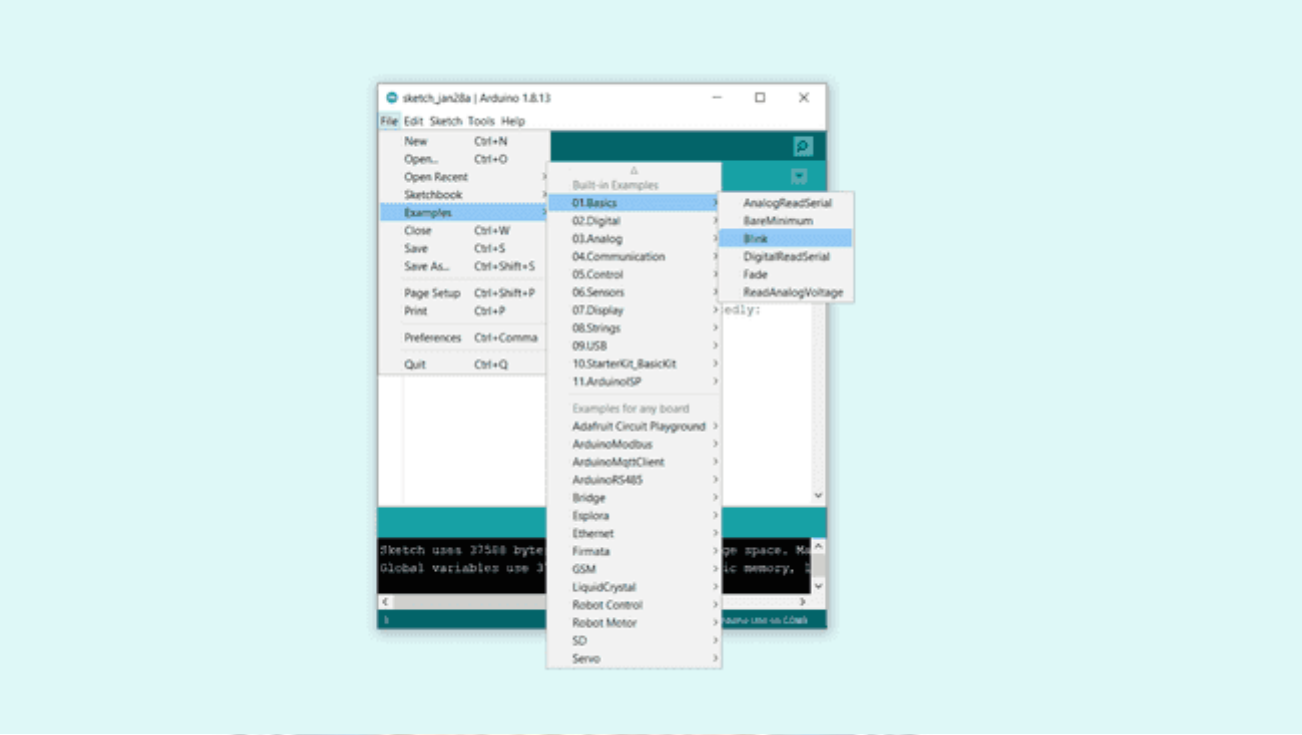


figure 7.4 Opening an example

9. To **upload it to your board**, simply click on the arrow in the top left corner. This process takes a few seconds, and it is important to not disconnect the board during this process. If the upload is successful, the message "Done uploading" will appear in the bottom output area.

10. Once the upload is complete, you should then see on your board the yellow LED with an L next to it start blinking. You can **adjust the speed of blinking** by changing the delay number in the parenthesis to 100, and upload the Blink sketch again. Now the LED should blink much faster.

The editor contains the four main areas:

1. A **toolbar with buttons** for common functions and a series of menus. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, choose your board and port and open the serial monitor.

2. The **Sidebar** for regularly used tools. It gives you quick access to board managers, libraries, debugging your board as well as a search and replacement tool.

3. The **text editor** for writing your code.

4. Console controls gives control over the output on the console.

5. The **text console** displays text output by the Arduino Software (IDE), including complete error messages and other information.

The bottom right-hand corner of the window displays the configured board and serial port.

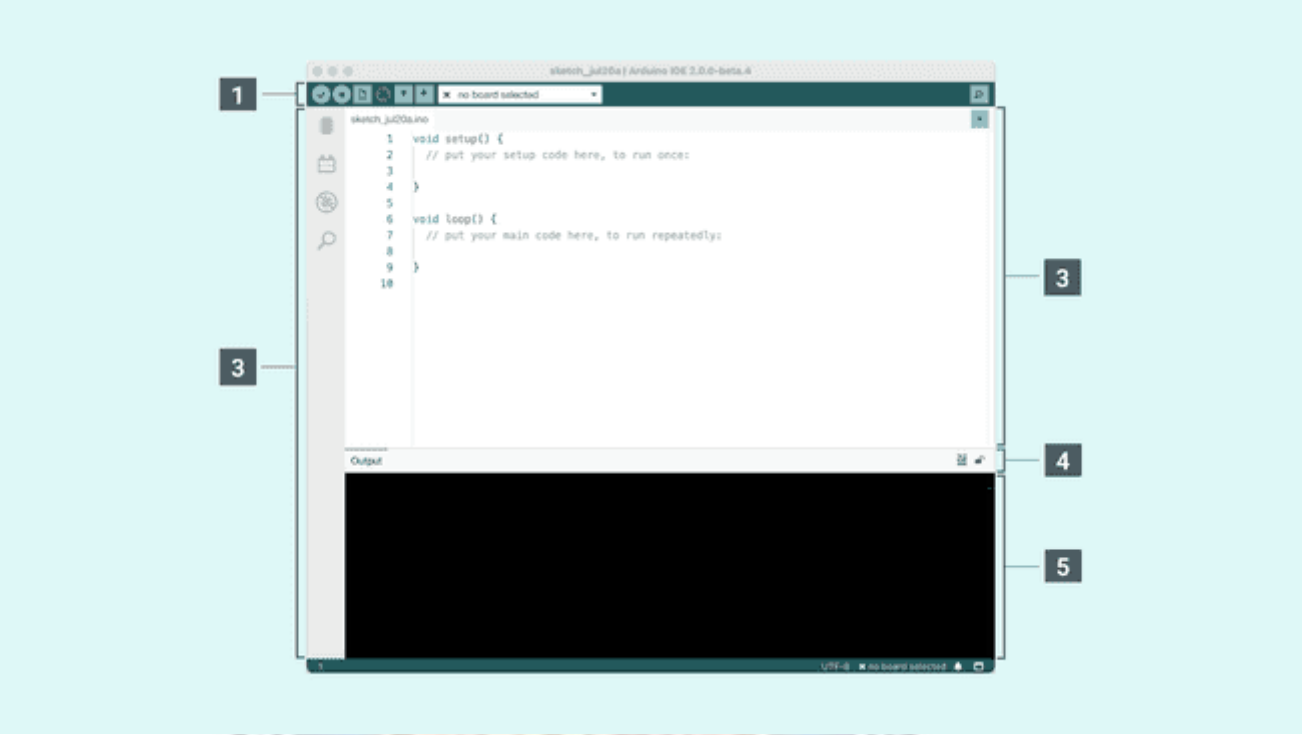


figure 7.5 The Arduino Software IDE

Now that you are all set up, **let's try to make your board blink!**

1. Connect your Arduino or Genuino board to your computer.

2. Now, you need to **select the right board & port**. This is done from the toolbar. Make sure you select the board that you are using. If you cannot find your board, you can add it from the board manager in the sidebar.

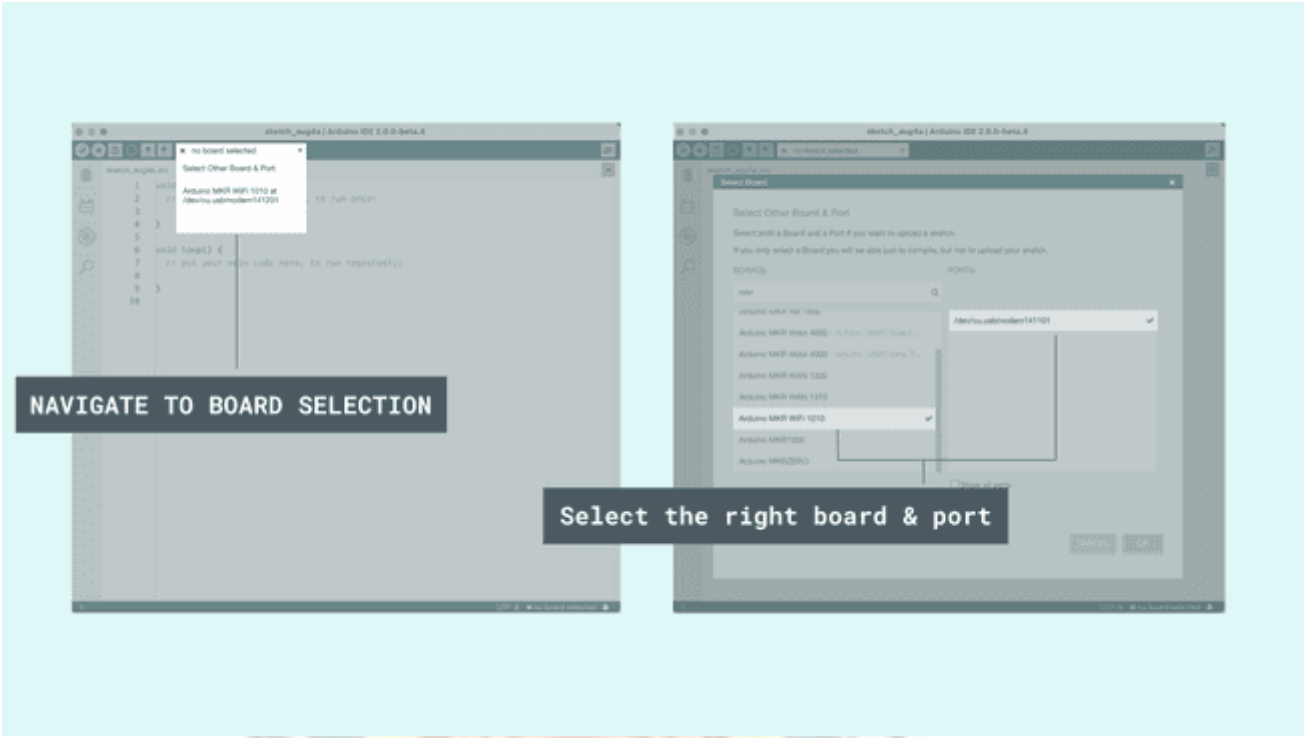


figure 7.6 Selecting a board & port

3. To **upload it to your board**, simply click on the arrow in the top left corner. This process takes a few seconds, and it is important to not disconnect the board during this process. If the upload is successful, the message "Done uploading" will appear in the bottom output area.

CHAPTER 8

SOURCE CODE

```
// DECLARATION
#include <ESP32Servo.h>
#include <LiquidCrystal.h>
#include <stdio.h>
#include <Wire.h>
#include "RTClib.h"
Servo myservo;
LiquidCrystal lcd(13, 12, 14, 27, 26, 25);
#define RXD2 19
#define TXD2 18
#define RXD2 20
#define TXD2 22
int ldr   = 2;
int pump  = 5;
int led   = 4;
const int trigPin = 16;
const int echoPin = 17;
int buzzer = 23;
// SETUP
void setup()
{
  Serial.begin(9600);//serialEvent();
  Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);pinMode(pump, OUTPUT);
  pinMode(ldr, INPUT);
```

```
digitalWrite(led, LOW);digitalWrite(pump, LOW);
digitalWrite(buzzer, HIGH);
lcd.begin(16, 2);
lcd.print("   IOT Urban");
lcd.setCursor(0,1);
lcd.print("Utilities Management");
     delay(2500);
if(!rtc.begin())
  {
   lcd.print("Couldn't find RTC");
    while (1);
  }
if(! rtc.isrunning())
  {
   lcd.print("RTC is NOT running!");
  }
lcd.clear();
delay(1500);
Serial.write("AT\r\n");        delay(3000);//okcheck();
Serial.write("ATE0\r\n");       okcheck();
Serial.write("AT+CWMODE=2\r\n"); delay(3000);
Serial.write("AT+CIPMUX=1\r\n");delay(3000);//        okcheck();
Serial.write("AT+CIPSERVER=1,23\r\n"); //       okcheck();
lcd.clear();
lcd.print("Waiting For");
lcd.setCursor(0,1);
lcd.print("Connection");
do{
   rcv = Serial.read();
  }while(rcv != 'C');
lcd.clear();
lcd.print("Connected");
delay(1500);
```

```
   lcd.clear();
  gsminit();
}
// LOGIC
void loop()
{
  lcd.clear();
  lcd.setCursor(0,0);lcd.print("U:");
  lcd.setCursor(8,0);lcd.print("L:");
  dist1 = ultra_dist();
  lcd.setCursor(2,0);convertl(dist1);
  // delay(200);
  if(dist1 < 10)
    {
     beep();
     String msg_gsm="";
     msg_gsm = "Drainage_Level_Full:" + String(dist1);
     gsm_send(msg_gsm);
    }
  if(digitalRead(ldr) == LOW)
    {
     lcd.setCursor(10,0);lcd.print("Light");
    }
  if(digitalRead(ldr) == HIGH)
    {
     lcd.setCursor(10,0);lcd.print("Dark ");
    }

  delay(1000);
  lcd.clear();
  DateTime now = rtc.now();
   //yearv, monthv, dayv, hourv, minv, secv
   year_c = now.year();
   month_c = now.month();
```

```
 day_c = now.day();
 hour_c = now.hour();
 min_c = now.minute();
 sec_c = now.second();
 lcd.setCursor(0, 1);
 lcd.print(hour_c);
 lcd.print(':');
 lcd.print(min_c);
 lcd.print(':');
 lcd.print(sec_c);
 lcd.print("    ");
 lcd.setCursor(0, 0);
 lcd.print(daysOfTheWeek[now.dayOfTheWeek()]);
 lcd.print(" ,");
 lcd.print(day_c);
 lcd.print('/');
 lcd.print(month_c);
 lcd.print('/');
 lcd.print(year_c);
if(hour_c == hourv1 && min_c == minv1 && sec_c >=0 && sec_c <= 8)
  {
   digitalWrite(led, HIGH);
   led_status=1;
  }
if(hour_c == hourv2 && min_c == minv2 && sec_c >=0 && sec_c <= 8)
  {
   digitalWrite(led, LOW);
   led_status=0;
  }
if(hour_c == hourv3 && min_c == minv3 && sec_c >=0 && sec_c <= 8)
  {
   digitalWrite(pump, HIGH);
  }
if(hour_c == hourv4 && min_c == minv4 && sec_c >=0 && sec_c <= 8)
```

```
  {
   digitalWrite(pump, LOW);
  }
if(led_status == 1 && digitalRead(ldr) == HIGH)
  {
   beep();
   String msg_gsm="";
   msg_gsm = "Light_Fault";
   gsm_send(msg_gsm);
   led_status=0;
  }
while(Serial.available() < 0)
    {
     char inChar = (char)Serial.read();//delay(5);
      if(inChar == '@')
        {sti=1;
        }
      if(sti == 1)
        {
           inputString += inChar;
        }
      if(inChar == '#')
        {sti=0;
         stringComplete = true;
        }
    }
if(stringComplete)
  {
  lcd.clear();
  if(inputString[1] == 'A' || inputString[1] == 'a')
    {
     hourv1   = (((inputString[2]-48)*10) + (inputString[3]-48));
     minv1   = (((inputString[5]-48)*10) + (inputString[6]-48));
     secv1  = (((inputString[8]-48)*10) + (inputString[9]-48))
```

```
   lcd.clear();lcd.write("light ON Configured");
  }
if(inputString[1] == 'B' || inputString[1] == 'b')
  {
   hourv2   = (((inputString[2]-48)*10) + (inputString[3]-48));
   minv2   = (((inputString[5]-48)*10) + (inputString[6]-48));
   secv2  = (((inputString[8]-48)*10) + (inputString[9]-48));
   lcd.clear();lcd.write("light OFF Configured");
  }
if(inputString[1] == 'C' || inputString[1] == 'c')
  {
     hourv3   = (((inputString[2]-48)*10) + (inputString[3]-48));
     minv3   = (((inputString[5]-48)*10) + (inputString[6]-48));
     secv3  = (((inputString[8]-48)*10) + (inputString[9]-48));
    lcd.clear();lcd.write("Pump ON Config");
  }
if(inputString[1] == 'D' || inputString[1] == 'd')
  {
     hourv4   = (((inputString[2]-48)*10) + (inputString[3]-48));
     minv4   = (((inputString[5]-48)*10) + (inputString[6]-48));
     secv4  = (((inputString[8]-48)*10) + (inputString[9]-48));
    lcd.clear();lcd.write("Pump OFF Config");
  }
if(inputString[1] == 'T' || inputString[1] == 't')
  {
yearv   = (((inputString[2]-48)*10) + (inputString[3]-48));
monthv   = (((inputString[5]-48)*10) + (inputString[6]-48));
dayv  = (((inputString[8]-48)*10) + (inputString[9]-48));
hourv   = (((inputString[11]-48)*10) + (inputString[12]-48));
minv = (((inputString[14]-48)*10) + (inputString[15]-48));
secv  = (((inputString[17]-48)*10) + (inputString[18]-48));
rtc.adjust(DateTime(yearv, monthv, dayv, hourv, minv, secv));
lcd.clear();lcd.write("D/T Configured");
  }
```

```
      sti=0;
      inputString = "";
      stringComplete = false;
      delay(1000);
      lcd.clear();
     }
   delay(1000);
}
// FUNCTIONS
unsigned int ultra_dist()
{int ud=0;
   digitalWrite(trigPin, LOW);
   delayMicroseconds(2);
   digitalWrite(trigPin, HIGH);
   delayMicroseconds(10);
   digitalWrite(trigPin, LOW);
   duration = pulseIn(echoPin, HIGH);
   distanceCm= duration*0.034/2;
   ud = distanceCm;
   return ud;
}
void gsm_send(String gsm_string)
{
    lcd.setCursor(15,1);lcd.write('G');
    delay(5000);delay(4000);delay(4000);  delay(4000);
    Serial2.write("AT+CMGS=\"");
    Serial2.write(pastnumber);
    Serial2.write("\"\r\n"); delay(3000);
    Serial2.print(gsm_string);
    Serial2.write(0x1A);
    delay(5000);delay(4000);  delay(4000);delay(4000);
    lcd.setCursor(15,1);lcd.write(' ');
}
void beep()
```

```
{
  digitalWrite(buzzer, LOW);delay(2000);digitalWrite(buzzer, HIGH);
}
void converts(unsigned int value)
{
  unsigned int a,b,c,d,e,f,g,h;
      a=value/10000;
      b=value%10000;
      c=b/1000;
      d=b%1000;
      e=d/100;
      f=d%100;
      g=f/10;
      h=f%10;

      a=a|0x30;
      c=c|0x30;
      e=e|0x30;
      g=g|0x30;
      h=h|0x30;

   Serial.write(a);
   Serial.write(c);
   Serial.write(e);
   Serial.write(g);
   Serial.write(h);
}
void convertl(unsigned int value)
{
  unsigned int a,b,c,d,e,f,g,h;
      a=value/10000;
      b=value%10000;
      c=b/1000;
      d=b%1000;
```

```
    e=d/100;
    f=d%100;
    g=f/10;
    h=f%10;

    a=a|0x30;
    c=c|0x30;
    e=e|0x30;
    g=g|0x30;
    h=h|0x30;

  // lcd.write(a);
   lcd.write(c);
   lcd.write(e);
   lcd.write(g);
   lcd.write(h);
}

void gsminit()
{
  Serial2.write("AT\r\n");                okcheck2();
  Serial2.write("ATE0\r\n");              okcheck2();
  Serial2.write("AT+CMGF=1\r\n");         okcheck2();
  Serial2.write("AT+CNMI=1,2,0,0\r\n");   okcheck2();
  Serial2.write("AT+CSMP=17,167,0,0\r\n"); okcheck2();
  lcd.clear();
  lcd.print("SEND MSG STORE");
  lcd.setCursor(0,1);
  lcd.print("MOBILE NUMBER");
  do{
     rcv = Serial2.read();
    }while(rcv == '*');
     readSerial(pastnumber);
```

```
lcd.clear();
lcd.print(pastnumber);
pastnumber[10]='\0';
  delay(4000);  delay(4000);
  Serial2.write("AT+CMGS=\"");
  Serial2.write(pastnumber);
  Serial2.write("\"\r\n"); delay(3000);
  Serial2.write("Mobile no. registered\r\n");
Serial2.write(0x1A);     delay(4000);  delay(4000);
```

CHAPTER 9

RESULT

The IoT-based Smart Water Meter prototype was successfully designed, developed, and tested. The system efficiently measured water usage in real-time and transmitted the data to a centralized cloud platform via Wi-Fi. The major outcomes include:

1. **Accurate Water Measurement:** The water flow sensor provided reliable and accurate readings of water consumption in liters.

2. **Real-Time Monitoring:** Users were able to monitor their water usage through a web dashboard/mobile app in real-time.

3. **Automatic Data Logging:** All usage data was automatically logged to the cloud, allowing for historical tracking and analysis**.**

4. **Leakage Detection:** The system was capable of detecting unusual or continuous water flow, suggesting possible leakages.

5. **Remote Alerts:** SMS or push notifications were successfully sent to users in case of excessive usage or system anomalies.

6. **User-Friendly Interface**: The interface allowed users to set consumption limits and receive alerts, encouraging water conservation.

7. **Power Efficiency:** The device operated efficiently with low power consumption, making it suitable for long-term deployment**.**

Overall, the project demonstrated how IoT technology can enhance water resource management, reduce wastage, and empower consumers with better control over their usage patterns**.**

SETUP:

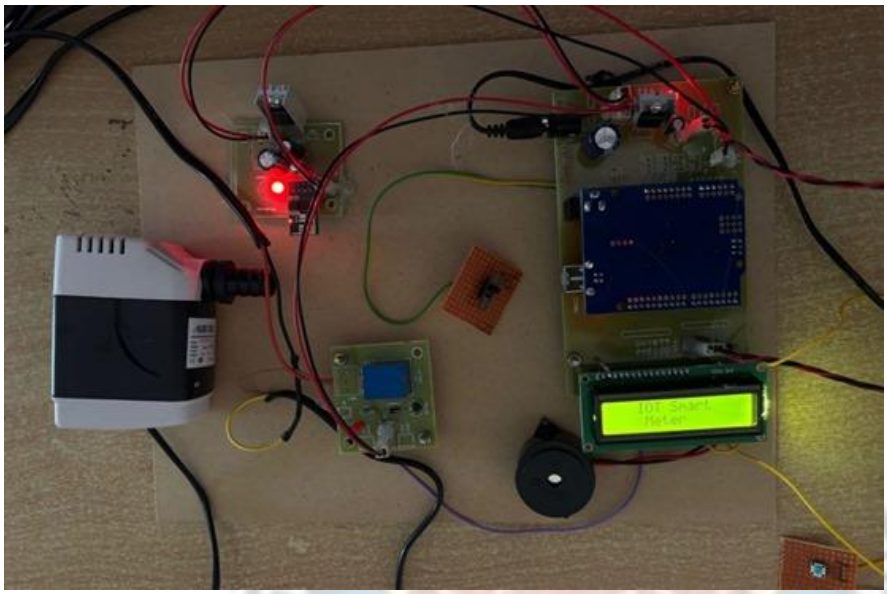


Figure 9.1 setup

Complete working setup of the smart water meter with flow sensor, power unit, and communication interface provides a comprehensive view of the entire setup, including the water flow sensor (connected to a pipeline), relay switch, Wi-Fi module, power supply unit, and the core microcontroller board. The components are neatly arranged on a baseboard to simulate a compact embedded system setup suitable for household or industrial applications

LCD DISPLAY:

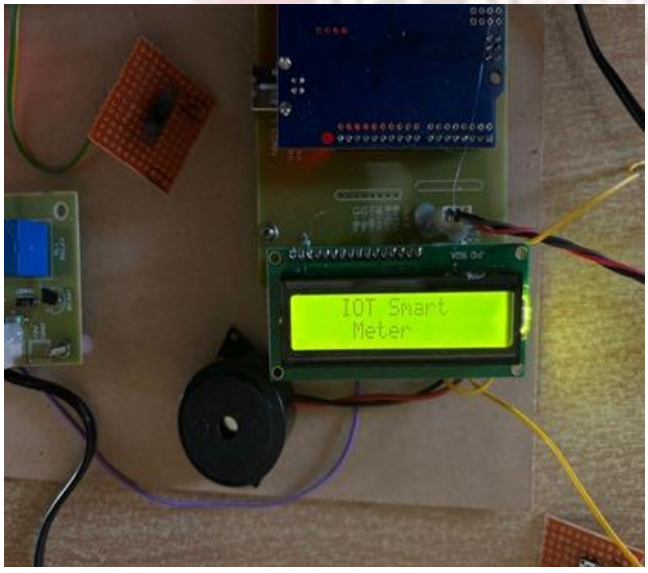
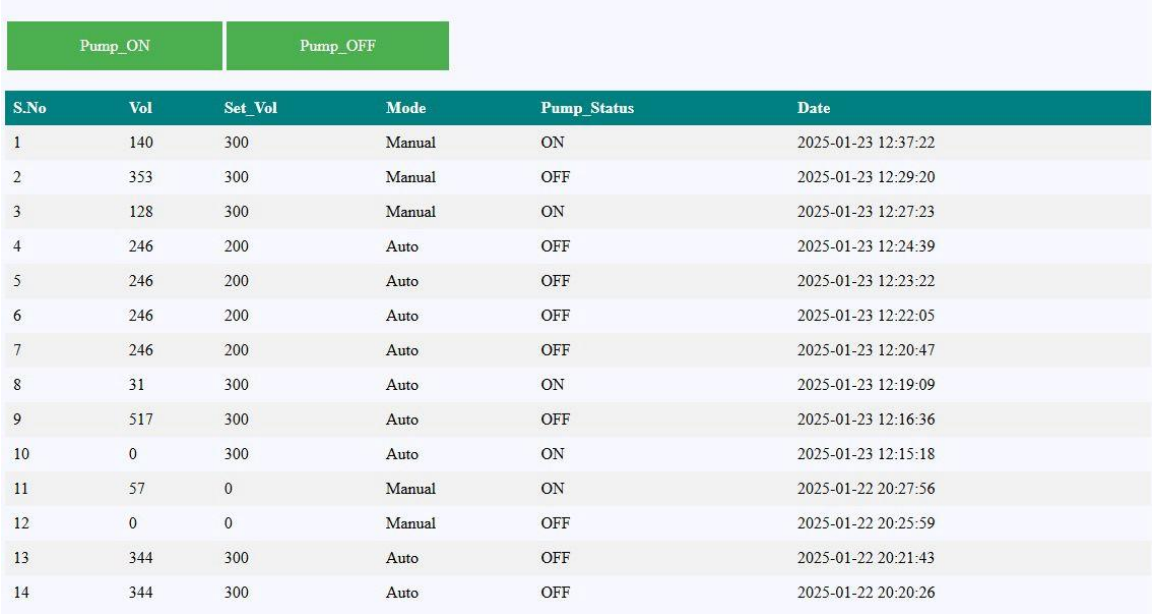


figure 9.2 LCD display

Display showing successful system initialization. the device's display confirms the successful initialization of the system, with the LCD displaying “IOT Smart Meter.” This indicates that the system is powered on and functioning as intended. The display plays a key role in providing users with local feedback on water usage and system status.

WEB APPLICATION:



S.No	Vol	Set_Vol	Mode	Pump_Status	Date
1	140	300	Manual	ON	2025-01-23 12:37:22
2	353	300	Manual	OFF	2025-01-23 12:29:20
3	128	300	Manual	ON	2025-01-23 12:27:23
4	246	200	Auto	OFF	2025-01-23 12:24:39
5	246	200	Auto	OFF	2025-01-23 12:23:22
6	246	200	Auto	OFF	2025-01-23 12:22:05
7	246	200	Auto	OFF	2025-01-23 12:20:47
8	31	300	Auto	ON	2025-01-23 12:19:09
9	517	300	Auto	OFF	2025-01-23 12:16:36
10	0	300	Auto	ON	2025-01-23 12:15:18
11	57	0	Manual	ON	2025-01-22 20:27:56
12	0	0	Manual	OFF	2025-01-22 20:25:59
13	344	300	Auto	OFF	2025-01-22 20:21:43
14	344	300	Auto	OFF	2025-01-22 20:20:26

Figure 9.3 web application page

The IoT-based water monitoring system includes a web-based application that plays a vital role in remote monitoring, controlling, and data logging. The application provides users with real-time access to water usage data and control options for managing water flow intelligently. As seen in the screenshot, the system supports both manual and automatic modes, tracks usage volumes, and records status changes over time.

GRAPHICAL REPRESENTATION:

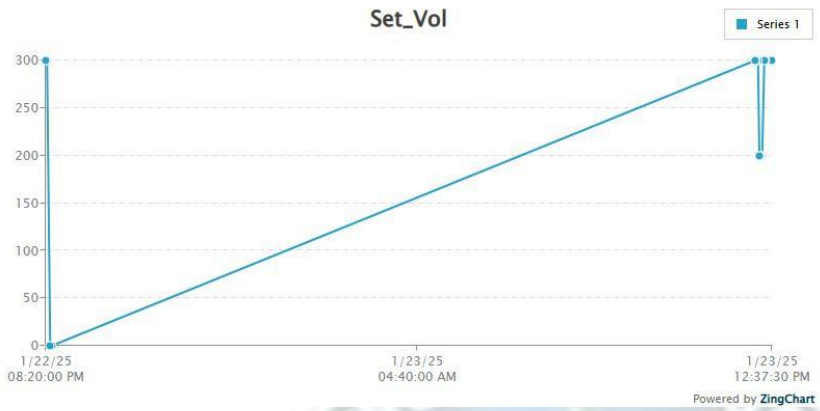


Figure 9.4 graphical representation 1

This chart confirms that the system allows flexible adjustment of the threshold levels, which can trigger pump OFF commands once consumption reaches the set volume—essential for automation and leak control.

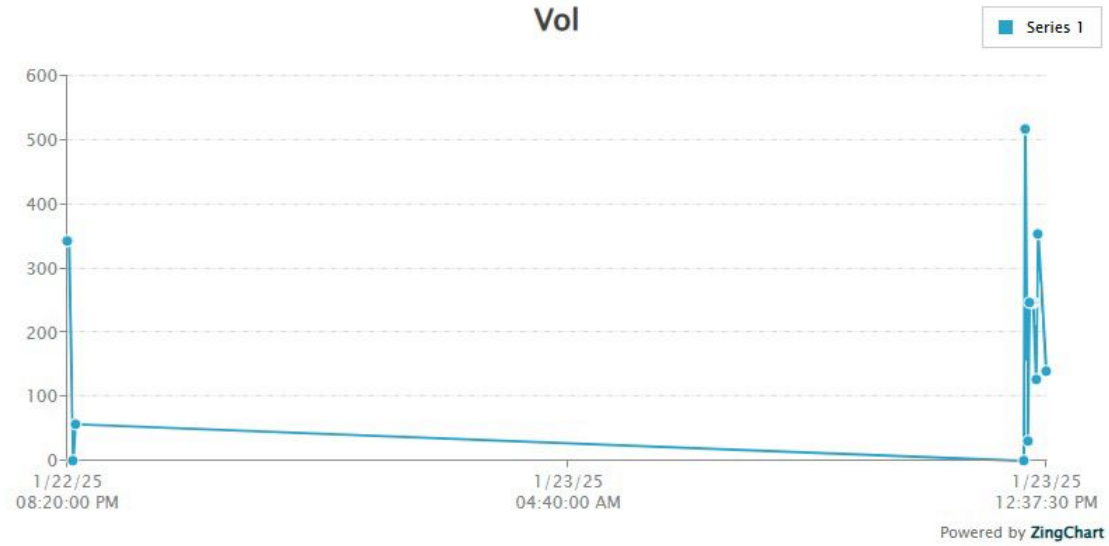


Figure 9.5 graphical representation 2

The chart validates the accurate sensing and reporting capabilities of the system. Sudden peaks and consistent values help detect patterns, potential overuse, or anomalies.

The IoT Smart Water Meter successfully demonstrates efficient water usage monitoring, real-time control, and automated management, making it a practical solution for promoting water conservation and smart resource management.

CHAPTER 10
CONCLUSION

In conclusion, the IoT-based Smart Water Metering system utilizing the ESP32 microcontroller offers a transformative approach to water consumption monitoring and utility billing. By integrating IoT connectivity, flow sensors, and a user-friendly LCD display, the system enables real-time water tracking, automated billing, and seamless communication between consumers and utility providers. The ESP32 microcontroller plays a pivotal role in processing water flow data and transmitting it to a central server, ensuring efficient and accurate tracking of water usage.

This system significantly enhances water resource management by promoting efficient usage, reducing water wastage, and eliminating the need for manual meter readings. The addition of features like remote access through an IoT dashboard and pump control mechanisms further improves the system's functionality and responsiveness to issues like excessive consumption or non-payment.

Future advancements will focus on incorporating AI-based predictive analytics for early detection of leakages, abnormal consumption patterns, and dynamic tariff adjustments. These developments will not only enhance operational efficiency but also contribute to the sustainability of urban water distribution systems, ensuring a more resource-efficient future. By leveraging IoT technology, this smart metering solution offers a sustainable and scalable model for modern water management.

References

[1] Mishra, A., et al., "IoT-Enabled Smart Water Metering," *IEEE Sensors Journal*, vol. 20, no. 4, pp. 1025–1031, 2020.

[2] Patil, R., et al., "Wi-Fi-Based Water Metering Using ESP8266," *International Journal of IoT Systems*, vol. 12, no. 2, pp. 134–142, 2021.

[3] Kumar, S., et al., "LoRaWAN for Smart Water Metering," *Journal of Wireless Communication*, vol. 9, no. 5, pp. 340–348, 2019.

[4] Rana, R., and Singh, A., "IoT-Driven Water Metering Framework Using Edge Computing," *IEEE Access*, vol. 10, pp. 555–563, 2022.

[5] Sahoo, D., et al., "Bluetooth-Based Smart Water Meter for Residential Applications," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2678–2686, 2020.

[6] Zhang, Y., et al., "Cloud-Integrated Smart Water Metering Using Firebase," *Smart Cities*, vol. 4, no. 2, pp. 145–153, 2021.

[7] Ahmed, M., et al., "Automated Billing Model Using AWS Cloud for Smart Water Metering," *Journal of Cloud Computing*, vol. 8, no. 1, pp. 12–21, 2020.

[8] Sharma, P., et al., "AI-Based Predictive Billing for Smart Water Metering," *AI for Smart Cities*, vol. 3, no. 1, pp. 25–33, 2018.

[9] Gupta, S., and Verma, S., "Efficient Data Streaming in IoT-Based Water Metering Using MQTT Protocol," *International Journal of Communication Systems*, vol. 35, no. 4, pp. 129–138, 2022.

[10] Lee, J., et al., "Prepaid Water Metering System Using Blockchain for Secure Billing," *International Journal of Digital Systems*, vol. 7, no. 3, pp. 215–223, 2019.

[11] Kim, H., et al., "Comparison of LoRaWAN, ZigBee, and NB-IoT for Smart Water Metering," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 845–854, 2017.

[12] Kumar, V., and Prasad, P., "GSM-Based Remote Metering System for Water," *International Journal of Advanced Communication Systems*, vol. 22, no. 4, pp. 1124–1130, 2021.

[13] Chen, Y., et al., "Hybrid LPWAN-GSM System for Rural Water Metering," *Journal of Network and Computer Applications*, vol. 129, pp. 21–29, 2019.

[14] Ali, H., et al., "5G-Enabled Smart Water Metering for Real-Time Data Transfer," *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 1325–1333, 2020.

[15] Wu, X., et al., "Satellite-Based IoT Communication for Water Metering in Remote Areas," *Telecommunications Systems*, vol. 74, no. 1, pp. 61–70, 2022.

[16] Jones, A., et al., "Flow Sensor Evaluation for Smart Water Meters," *Journal of Sensor Technology*, vol. 18, no. 7, pp. 309–317, 2018.

[17] Patel, R., et al., "Multi-Sensor IoT Water Meter with Temperature, Pressure, and Flow Detection," *Sensors and Actuators A: Physical*, vol. 296, pp. 249–256, 2020.

[18] Khan, Z., et al., "AI-Driven Anomaly Detection Model for Leakage Identification in Water Networks," *Applied Intelligence*, vol. 51, no. 6, pp. 4507–4519, 2021.

[19] Liu, Y., et al., "Water Quality Monitoring in Smart Water Meters Using pH, Turbidity, and Contaminant Sensors," *Journal of Environmental Monitoring*, vol. 21, no. 5, pp. 743–751, 2019.

[20] Ramachandran, S., et al., "Ultrasonic-Based Smart Water Meter for Accuracy and Durability," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 876–884, 2021.

[21] Singh, M., and Bose, N., "Deep Learning for Predicting Consumer Water Usage Trends," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 3, pp. 302–311, 2021.

[22] Bhatia, S., et al., "Machine Learning-Based Demand Forecasting for Dynamic Water Pricing," *Computational Intelligence in Smart Water Management*, vol. 6, pp. 145–154, 2020.

[23] Wilson, J., et al., "Real-Time Anomaly Detection for Water Leakages Using Convolutional Neural Networks (CNN)," *Journal of Machine Learning and Water Systems*, vol. 19, no. 2, pp. 223–230, 2019.

[24] Xie, Z., et al., "Fuzzy Logic for Water Distribution Valve Control in Smart Cities," *Urban Water Journal*, vol. 14, no. 1, pp. 61–69, 2021.

[25] Jain, P., et al., "Reinforcement Learning Model for Optimizing Pump Operations in Water Distribution Systems," *AI in Water Resources*, vol. 8, pp. 102–110, 2022.

[26] Dutta, D., et al., "Smart Water Metering Impact on Sustainable Urban Planning," *Sustainable Cities and Society*, vol. 48, pp. 1–7, 2019.

[27] Verma, S., et al., "Government Policies Supporting Smart Water Metering Systems," *Public Policy and IoT*, vol. 11, pp. 202–210, 2020.

[28] Fujita, Y., et al., "Role of IoT Water Meters in Smart Cities for Reducing Operational Costs," *Smart City and IoT Integration*, vol. 9, no. 4, pp. 567–575, 2021.

[29] Gomez, R., et al., "Consumer Behavior Changes Due to Smart Metering Awareness Programs," *Journal of Consumer Research*, vol. 45, no. 6, pp. 983–991, 2018.

[30] Rao, S., et al., "Integration of IoT-Based Smart Meters into Smart City Water Grids," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1005–1014, 2022.