

CLOUD BASED LOW COST HOME MONITORING AND AUTOMAATION SYSTEM

Submitted in partial fulfillment of the requirements for the award of

Bachelor of Engineering Degree in

Electronics and Communication Engineering

By

SMD.Tausif (38130204)
Shaik.Meharaj (38130203)



DEPARTMENT OF ELECTRONIC AND COMMUNICATION
ENGINEERING
SCHOOL OF ELECTRICAL AND ELECTRONICS

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

MARCH – 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC



**JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI -
600 119**

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of
SMD.TAUSIF(38130204) and **SHAIK.MEHARAJ(38130203)**, who carried out
the project entitled **"CLOUD BASED LOW COST HOME MONITORING AND
AUTOMATION SYSTEM"** under my supervision from NOVEMBER 2020 to
MARCH 2021.

Internal Guide
DR.G. JEGAN, M.E., Ph.D.,

Head of the Department
Dr. T. RAVI, M.E., Ph.D.,

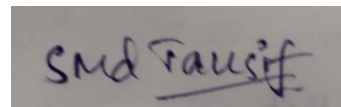
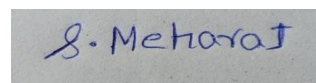
Submitted for viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, **SMD.TAUSIF(381302014)** and **SHAIK.MEHARAJ(38130203)**, hereby declare that the Project Report entitled **CLOUD BASED LOW COST HOME MONITORING AND AUTOMATION SYSTEM** is done under the guidance of **Dr. G. JEGAN, M.E., Ph.D.**, is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in **Electronics and Communication Engineering**.

Handwritten signature of Smd Tausif in blue ink on a light gray background.Handwritten signature of S. Meharaj in blue ink on a light gray background.

DATE:

SIGNATURE OF THE CANDIDATE

PLACE:

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my profound gratitude to **Dr. N.M. NANDHITHA, M.E., Ph.D.**, Prof. & Dean, School of Electrical and Electronics and **Dr. T. RAVI, M.E., Ph.D.**, Head of the Department of **Electronics and Communication Engineering**, Sathyabama University for having been a constant source of support and encouragement for the completion of the project.

I consider my self-most fortunate and immensely privileged to have worked under the guidance of **Dr. G.JEGAN, M.E., Ph.D.**, Department of Electronics and Communication Engineering to complete this project successfully.

I express our heartfelt thanks to our family members and all our friends who helped me to complete our project successfully & finally, I thank the exciting Super Natural Power, the Almighty GOD.

ABSTRACT

In this System the optimized hardware with reduced cost an IOT based employing a voice-commanded/Manual commands smart two-way automated system has been suggested and built. The ESP8266 NodeMCU micro-controller board is employed as a key element of this system. The system when utilized as smart Home Automation began with a voice command through a smartphone delivered by the user utilizing cloud UBIDOTS Google Assistant with a co-interfaced IFTTT (If This Then That) and the Blynk app platform which made this functionality feasible. The multichannel relay module for loads is interfaced with the central unit. The central unit processes the data as per the user input. When utilized for Agricultural monitoring the environmental data comprising soil moisture, air temperature and humidity (DHT11) has been assessed and shown on the OLED screen. Water flow was regulated utilizing the water motor as per soil moisture threshold value. Hypertext transfer protocol based ESP8266 Wi-Fi based NodeMCU is routed to the Wi-Fi network and interacted with cloud storage (Thingspeak) and IOT platform Blynk. The data from the sensors of the required environmental parameters have been gathered, monitored and then delivered to the storage cloud and Blynk using Wi-Fi. The research focuses on accomplishing automation of two separate sectors using a single hardware solution.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE
	ABSTRACT	V
	LIST OF FIGURES	Viii
	LIST OF ABBREVIATIONS	ix
1.	INTRODUCTION	7
	1.1 OBJECTIVES	8
2.	LITERATURE REVIEW	
	2.1 LITERATURE SURVEY	8
	2.2 EXISTING METHOD	10
3.	PROPOSED METHODOLOGY	11
	3.1 BLOCK DIAGRAM OF PROPOSED	
	3.2 OVERVIEW OF EMBEDDED	
	SYSTEM	12-22
	3.4 EXPLANATION OF IOT	22-30
	3.5 COMPONENTS	31-44
4.	SOURCE CODE	55
5.	RESULTS AND DISCUSSION	58
6.	CONCLUSION	59
7.	REFERENCES	60

INTRODUCTION

People's lifestyles throughout the globe have changed in recent decades as a result of technological advancements. This revolution relies heavily on information technology. The number of people accessing the internet on their smartphones has skyrocketed since their invention. Information technology and home automation may both benefit from this development.

People's attachment to smart devices is growing in our fast-paced society. Automated and easy-to-use systems are essential for people. To make things easier for people, modern houses have grown even more automated. Despite the fact that many individuals now own cutting-edge smart gadgets, most home automation systems are out of reach for the average family's budget. However, not everyone has the financial resources to take advantage of this opportunity. As a result, the need for low-cost smart home technologies is rising. Luxury seekers are the primary target audience for most of these home automation systems.

Because it is so cheap, this approach makes it easier for users to operate a gadget manually. Wireless home automation systems are becoming more popular because they provide a level of security, comfort, and safety for all sorts of consumers.

Wireless home automation systems that can be controlled by voice commands are the goal of this research. In order to interface with the ESP32 and the users, the system makes use of Google Assistant. Light, fan, and smart-door unlocking may all be controlled by voice commands in English or Malayalam. These new technologies may be difficult for the elderly and those with disabilities; therefore this allows them to manage gadgets in their native language. Home automation solutions that include the Google Assistant provide for a more pleasant user experience.

OBJECTIVES:

Wireless home automation systems that can be controlled by voice commands are the goal of this project. In order to interface with the ESP32 and the users, the system makes use of Google Assistant. English voice commands are used to operate household equipment such as lights, fans, and smart door unlocking. These new technologies may be difficult for the elderly and those with disabilities; therefore this allows them to manage gadgets in their native language. Automating your house with Google Assistant is a great way to make your system even more user-friendly and convenient to use.

2.1 LITERATURE SURVEY

Title: Home automation using Internet of Thing Author: Shopan Dey; Ayon Roy; Sandip Das

Content: Humans' lives are becoming easier as more and more tasks are automated, replacing the old-fashioned manual processes. Humans nowadays rely on the internet for almost every aspect of their lives, and they would be lost without it. IoT offers a framework for connecting, sensing, and controlling devices through a network infrastructure via a remote interface. Smart phone and computer-based home automation is the subject of this article. Buildings of all kinds may be controlled and monitored by iot devices, which control and monitor the electronic, electrical, and mechanical systems. The cloud server has a single administrator who manages a large number of users and a large number of sensor and control nodes.

Title: A Smart Home Appliance Control System for Physically Disabled People
Author: Progress Mtshali; Freedom Khubisa

Content: The number of houses being built or converted into "smart homes" has risen dramatically in the last several years. Digital assistants like Amazon's Alexa and Google's Home and Assistant, as well as Apple's Siri and Microsoft's Cortana may all be credited with this shift in behavior. Smart homes were designed from the start with the non-disabled general public in mind. With impressive regularity, residents of smart homes have demonstrated their ability to take advantage of the safety, energy savings, and convenience that comes with being able to control their home's lighting system, HVAC system, door locks, and coffee makers from anywhere in the house, including their beds and couches.

3.Title: Home Computerization for the Elderly And Disabled with Internet of Things

Author: Sakshi Dhiman; Shivang Gupta; Anshul Chopra; Prem Chand Vashist

Content: For persons with disabilities, the smart home provides a solution to the many challenges they confront on a daily basis. The objective here is to build a system that will make daily living easier and more comfortable for those who are physically challenged. The improvement in home automation in the last few years has been remarkable. Voice Control, Wi-Fi, and Bluetooth are used in the proposed system, which would empower individuals and make it easier for them to use their equipment. The advent of low-cost microcontrollers has made it possible to create low-cost smart houses.

4.Title: Design and Implementation of IoT-Based Smart Home Voice Commands for disabled people using Google Assistant

Author: Haris Isyanto; Ajib Setyo Arifin; Muhammad Suryanegara

Content: Developing Internet of Things (IoT) technology has a favorable influence on human lives. Smart homes are one of the uses of the Internet of Things (IoT). People with disabilities face a plethora of activities that are out of reach because of

their physical impairments. Designing a voice-activated IoT-based smart home application for impaired individuals is the focus of this article. Devices like TVs, lights, and fans may all be controlled by voice commands without the user having to get out of his or her chair or get up from a chair. Apps like Google Assistant are used for speech recognition in electronic devices. When the pronunciation is accurate, the Google Assistant app will receive voice instructions.

2.2 EXISTING SYSTEM:

The Design and Implementation of a Remote Appliance Control System Using an Android Smartphone presents information regarding the design and implementation of a remote appliance control system using an Android smartphone. A user signs into the smart phone interface and gently presses the buttons to send message instructions to the home information center over the GSM network. The PIC processor then detects the provided command and wirelessly operates the household appliance switches, enabling eventual remote control of appliances. By using Bluetooth on Android mobile devices for home security applications, we have incorporated information concerning mobile device into our daily lives.

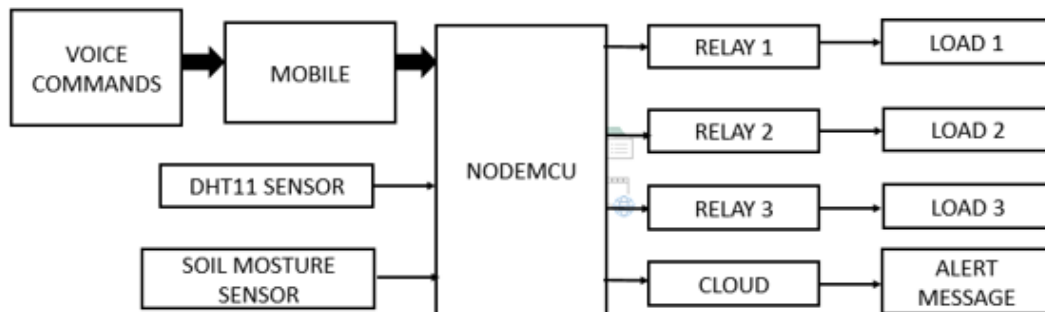
Home automation and security are becoming more major features on mobile devices. Because a short-range communication technology was sought, the mobile device and security system interacts through Bluetooth. With the assistance of an Android mobile device, we may remotely perform tasks such as locking doors and turning on/off lights.

3. PROPOSED SYSTEM:

To summarize, this system proposes an IoT-based hardware platform that can be utilized for both smart home automation and agricultural monitoring through a

single MCU. In contrast to other advances in conventional IOT-based automation, this SYSTEM is primarily concerned with hardware optimization. Separate microcontrollers are utilized for home automation and agriculture monitoring in other devices, which saves energy by using a single piece of hardware. It would make a substantial contribution to the progress of IOT-based automation systems by providing simple user access and security.

3.1 BLOCK DIAGRAM:



BLOCK DIAGRAM COMPONENTS:

NODEMC

U RELAYS

DHT11

SENSOR

SOIL MOISTURE

SENSOR 230 V LOAD

3.2 EMBEDDED SYSTEMS/ OVERVIEW:

An embedded system is a computer system with a narrow focus on a single or limited set of tasks, sometimes with a focus on real-time performance. For the most part, it

is integrated into a larger device that includes both hardware and mechanical components. The programming flexibility of a general-purpose computer, such as a laptop or desktop, allows it to perform a wide range of functions. Many of the items we use every day are controlled by embedded systems. A product's dependability and performance may be improved or its size and cost reduced thanks to an embedded system that is devoted to certain functions. For example, certain embedded systems benefit from economies of scale since they are mass manufactured.

This includes small gadgets like digital watches and MP3 players, as well as large-scale fixed installations like traffic lights and industrial controls. The level of complexity ranges from a single microcontroller chip to a huge chassis or enclosure with many units, peripherals, and networks.

Embeddable systems aren't properly defined, since many have some degree of programmability. Handheld computers, for example, share some features with embedded systems, such as operating systems and microprocessors that power them, but they are not truly embedded systems because they allow different applications to be loaded and peripherals to be connected, which is not the case in embedded systems.

Many functions are provided by embedded systems.

Embedded systems read data from sensors in the environment. Following the processing of this data, the findings are shown to the user or users in some manner. Embedded systems create and communicate actuator instructions for the environment. For example, an embedded system may compress or decompress data acquired in some useful manner.

Embedded systems may interface with the outside world through sensors and actuators, but they also offer functionality that is exclusive to the applications they are used in. Programming such as finite state machines (FSMs) and signal processing methods are commonplace in embedded systems. Faults in the internal computer

environment as well as the external electromechanical systems must be detected by these systems.

In addition to communication devices and appliances for the house, control systems are also included in the broad category of embedded systems. The following are a few examples of this kind of technology: For instance, modems and mobile phones, The following are examples of home appliances: For instance, a CD player, a VCR, or a microwave, And control Systems, Automobile anti-lock braking systems, robotics, satellite control

3.2.0Block diagram of an embedded system:

An embedded processor is a common component of an embedded system. Microwaves, VCRs, and automobiles all have digital interfaces that are controlled by embedded systems. An operating system may be included in certain embedded systems. In other cases, the logic is so narrowly focused that it can only be implemented as a single program. Other than providing general-purpose computing, these systems are designed to perform a specific function when integrated into a device. As can be seen in Figure 1.1, a typical embedded system,

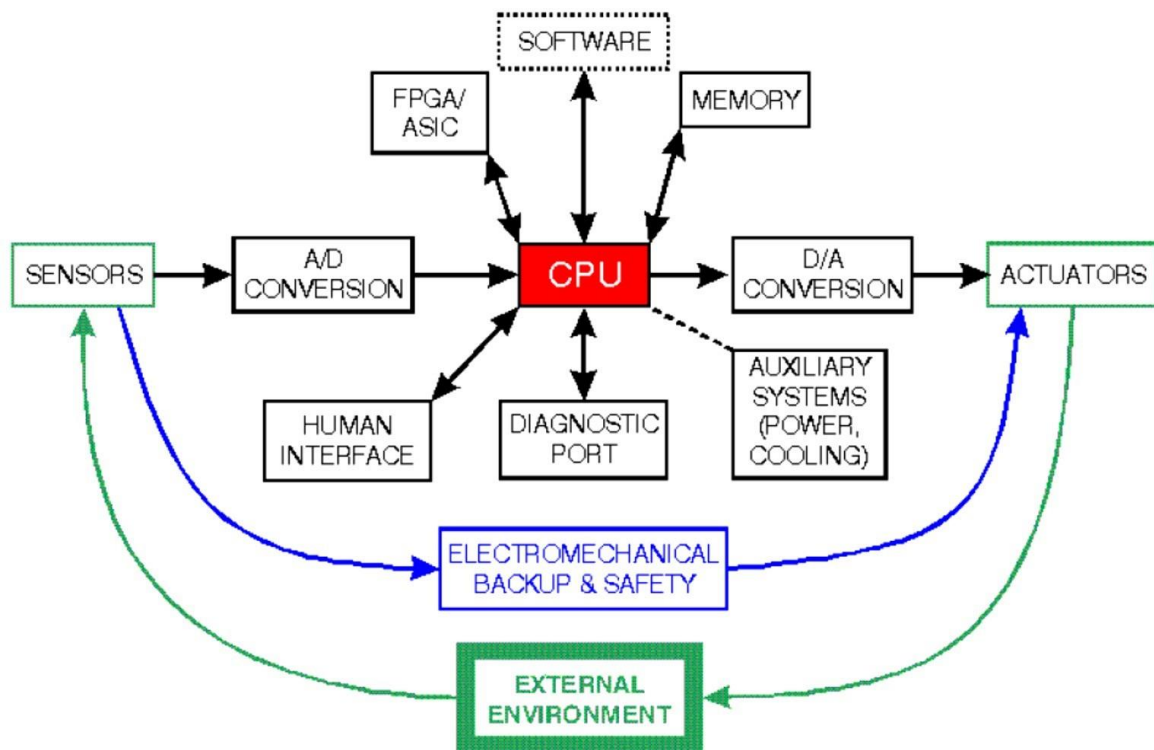


Fig 1.1 Block diagram of a typical embedded system

3.2.1 Characteristics of embedded systems

Embedded systems exhibit a distinct set of properties. Each of these traits put a unique set of design limitations on designers of embedded systems. The difficulty in building embedded systems is adhering to the application's unique set of limitations.

Application Specific Systems:

Embedded systems do not function in the same way that general-purpose computers do. Embedded system designs are application-specific. Numerous work features are known prior to designing the gear. This enables the designer to concentrate on the application's unique design limitations. As a result, user reprogramming capability is restricted. Certain embedded systems, on the other hand, need the capacity to reprogram. Programmable DSPs are often used in these applications.

Reactive Systems

As previously stated, a typical embedded systems model reacts to and controls the environment through sensors and actuators. This necessitates that embedded systems operate at the same pace as the environment. This property of embedded systems is referred to as "reactiveness." Reactive computing refers to the process through which a system (mainly a software component) responds to external events. External events might be periodic or aperiodic in nature. Periodic events simplify the process of scheduling processes to ensure performance. Schedules for aperiodic occurrences are more challenging. To account for worst-case scenarios, the maximum event arrival rate must be predicted. The majority of embedded systems have a sizable reactive component. Performing an accurate worst-case design study on systems with statistical performance characteristics is one of the most difficult tasks for embedded system designers (e.g., cache memory on a DSP or other embedded processor). The functioning of a real-time system implies that the accuracy of a calculation is contingent on the time at which it is provided. Systems that must meet this criteria are often designed for worst-case performance. However, precisely anticipating the worst-case scenario may be challenging in complex infrastructures. This often results in excessively gloomy projections that err on the side of caution. Numerous embedded systems need real-time operation to fulfil external I/O and control stability requirements. Numerous real-time systems are reactive systems as well.

Distributed Systems

An embedded system is often defined as one that comprises of communicative

processes running on many CPUs or ASICs that are linked through communication connections. This is for economic reasons. Economical Four eight-bit microcontrollers may be less expensive than a single 32-bit CPU. Even when the cost of communication channels is included in, this technique may be preferable. Numerous processors are often needed to perform multiple time-critical activities in this technique. Physical distribution of devices controlled by embedded systems is also possible.

Heterogeneous Architectures

Embedded systems are often made of disparate designs (Fig 1.2). They may include a variety of processors inside a single system solution. Additionally, they may be mixed signal systems. Embedded system design is absolutely unique due to the integration of I/O interfaces, local and remote memory, sensors, and actuators. Additionally, embedded systems have stringent design limitations, and heterogeneity allows for greater design freedom.

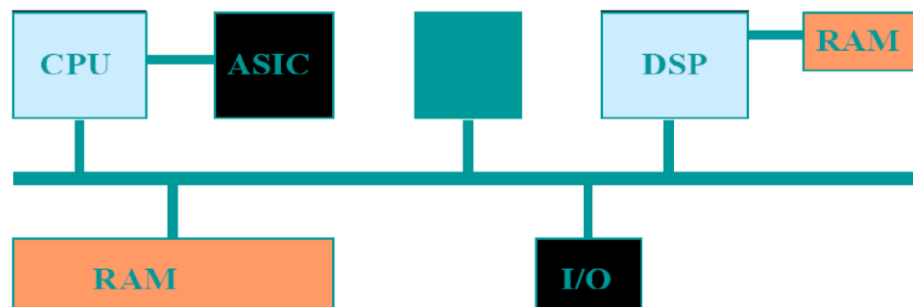


Fig 1.2 Embedded Systems having Heterogeneous

Architectures Harsh environment

Numerous embedded systems operate in an uncontrollable environment. Excessive heat is often a concern, particularly in combustion-related applications (e.g., many transportation applications). Additional issues for embedded computing might arise as a result of the need for protection against vibration, shock, lightning, power supply fluctuations, water, corrosion, fire, and general physical abuse.

System safety and reliability

As embedded system complexity and processing power increase, they begin to govern an increasing proportion of the entire system's safety functions. These safety precautions may be implemented by software or hardware control. Mechanical safety backups are often initiated when a computer system loses control, allowing the system to safely shut down. Software security and dependability are a greater concern. Generally, software does not "break" in the hardware sense. However, software may be so complicated that a series of unanticipated events might result in software failures, resulting in dangerous scenarios. Although this subject is outside the scope of this book, embedded designers face several obstacles, including developing reliable software and constructing inexpensive, available systems employing faulty components. The primary goal of embedded system designers is to provide low-cost dependability with the least amount of redundancy possible.

Control of physical systems

One of the primary motivations for embedding a computer is to allow it to communicate with the surrounding environment. This is often accomplished via the monitoring and control of external machines. Embedded computers convert analogue sensor data to digital signals for processing. The outputs must then be converted to analogue signal levels. When managing physical equipment, it may be

necessary to switch huge current loads in order to run motors and other actuators. Embedded systems may need big computer circuit boards with a significant number of non-digital components to achieve these requirements. Embedded system designers must carefully balance trade-offs between analogue components, power, mechanical, and network components, as well as digital hardware and associated software.

Small and low weight

Numerous embedded computers are physically incorporated into bigger systems. Aesthetics may determine the form factor of the embedded system. For instance, the form factor of a missile may need to fit within the missile's nose. One of the issues facing embedded system designers is developing solutions with non-rectangular shapes. Additionally, weight might be a significant restriction. For example, embedded car control systems must be lightweight to maximize fuel efficiency. Portable CD players must be lightweight in order to be portable.

Cost sensitivity

Cost is a consideration in the majority of systems, although the sensitivity to cost changes varies significantly in embedded systems. This is mostly due to the influence that computer expenses have on profitability, and is more a function of the percentage of cost changes relative to the overall system cost.

Power management

Embedded systems are constrained by stringent power limits. Given the mobility requirements of many embedded systems, power conservation is critical to extending battery life. Another apparent problem for embedded systems is the reduction of heat generation.

3.3.1 POWER SUPPLY

Because all electronic circuits operate at low direct current voltages, we need a power supply unit to provide the right voltage supply for optimal operation. This unit is composed of the following components: a transformer, a rectifier, a filter, and a regulator. A standard 230volt RMS ac voltage is linked to a transformer voltage that is reduced to the appropriate ac voltage level. A diode rectifier that generates a full wave rectified voltage that is then filtered using a simple capacitor filter to create a direct current voltage. Typically, the resultant dc voltage has some ripple or ac voltage change. A regulator circuit may utilize this dc input to generate dc voltage that has much less ripple but also maintains the same dc value when the dc voltage fluctuates slightly or when the load attached to the output dc voltages changes.

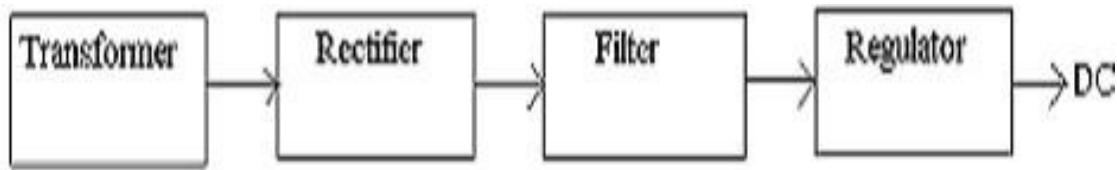


Fig 1.General Block of Power Supply Unit

3.3.1 TRANSFORMER:

A transformer is a fixed component that converts electric current in one circuit to current of the same frequency in another circuit. It may increase or decrease the voltage in the circuit, but at the expense of decreasing or increasing the current. It operates on the mutual induction concept. In our project, we're using a step down transformer to deliver power to the electrical circuits. Here, we convert 230 volts ac to 12 volts ac.

3.3.2 RECTIFIER:

A process called full wave rectification may be used to enhance a dc level derived from a sinusoidal input by 100 percent. In our project, we employ a bridge rectifier to achieve full wave rectification. We can see from the basic bridge design that two diodes (say D2 and D3) conduct while the other two diodes (D1 and D4) remain in the off state for the time interval $t = 0$ to $T/2$. D1 and D4 are the conducting diodes that correspond to the input's negative cycle. Thus, the polarity is same throughout the load.

In a bridge rectifier, diodes of various sorts such as 1N4001, 1N4003, 1N4004, 1N4005, 1N4007, and others may be utilised. However, in this case, we choose 1N4007, which is capable of withstanding up to 1000v.

3.3.3 FILTERS:

To get a dc voltage of zero Hz, we must use a low pass filter. As a result, a capacitive filter circuit is utilised, in which a capacitor is connected across the rectifier output and a dc voltage is produced. The filtered waveform is simply a direct current with very few ripples, which is supplied to the load.

3.3.4 REGISTERS:

The controller IC has two registers of eight bits each, an instruction register (IR) and a data register (DR) (DR). The instruction register (IR) holds the instruction codes and addresses for the display data RAM (DD RAM) and character generating RAM (CG RAM). The MPU can write to but not read from the IR.

The DR temporarily stores data that is intended to be written to or read from the DD RAM or CG RAM. As an internal operation, the MPU writes data to DR to the DD RAM or CG RAM.

When an address code is written to IR, data from the DD RAM or CG RAM is instantly moved to the DR. When the MPU reads the DR, the data transmission between the MPU is accomplished. Similarly, when the MPU reads the DR for the next time, data in DD RAM or CG RAM at the address is immediately transmitted to the DR. Similarly, when the MPU writes the DR, the next DD RAM or CG RAM address is used.

The LSI controller and driver for dot-matrix liquid crystal display alphanumeric, Japanese kana letters, and symbols. It may be designed to operate a dot matrix liquid crystal display through a 4- or 8-bit CPU. Since all of the tasks necessary for controlling a dot-matrix liquid crystal display, including as display RAM, character generator, and liquid crystal driver, crystal display are internally provided on one chip, a minimal system can be interfaced with this controller/driver

3.3.5 REGULATORS:

The capacitor's output voltage is further filtered and ultimately controlled. The voltage regulator is a device that maintains a constant output voltage regardless of supply variances, load variations, or temperature variations. Here, we're going to

employ a fixed voltage regulator, namely the LM7805. The integrated circuit LM7805 is a 5 volt regulator that is used in microcontrollers.

3.3.6 FEATURES AND DESCRIPTION OF REGULATORS

- Output Current up to 1A
- Output Voltages of 5, 6, 8, 9, 10, 12, 15, 18, 24V
- Thermal Overload Protection
- Short Circuit Protection
- Output Transistor Safe Operating Area Protection

3.4 EXPLATION OF IOT (INTERNET OF THINGS) :

This chapter is structured as follows: The motivation for this effort is explained in Section 3.4.1. The study aims and questions are summarised in Section 3.4.2. Section 3.4.3 details the procedures necessary to accomplish our goals. Section 1.4 details the report's structure.

For the Future Internet, the objective is to establish an infrastructure that allows for rapid access to information about the real world and its objects. Different application areas may benefit from physical items, such as e-health and warehouse management. Depending on the application, a variety of physical devices may be required. If you want to interface with a particular physical device, you'll need the device's exact specs. A layered vision is needed to enable data access in order to fulfil the aim of the future Internet. By using layers of architecture, the Internet of Things (IoT) may connect the virtual and real worlds of information.

The terms "Internet" and "Things" combine to form the phrase "Internet of Things". The term "internet" refers to a worldwide network architecture that is scalable,

customizable, and interoperable. Things are real or virtual things, equipment, or information that have identities, physical properties, and virtual personalities, as well as intelligent interfaces [1]. [2]. There are several ways to describe a virtual object, including an abstract unit of sensor nodes with information that can identify and find its accompanying sensor nodes. As a result, the Internet of Things (IoT) refers to devices that may transmit data from their physical surroundings through the Internet. Device interaction and data management are the responsibilities of middleware, which serves as a bridge between the hardware layer and the application layer [2]. It is the job of a middleware to provide a consistent programming paradigm for interfacing with devices. [3] A middleware is responsible for obscuring the difficulties we encounter while communicating with gadgets.

3.4.2 Motivation

An IoT-based system is responsible for supplying non-expert users with information from an environment. If you're using an IoT-based system, you need to be able to handle many various types of devices. A fundamental issue in designing an IoT-based system for non-expert users is how to manage the heterogeneous devices. A layer of middleware may alleviate this issue by putting a barrier between the devices and non-expert users. For the end user, this layer hides the variety of devices and gives access transparency to the devices via this layer.

3.4.3 IoT definition

Increasingly, industry is paying attention to the Internet of Things (IoT) and how it might interact with various sorts of devices. Through the incorporation of physical items into information networks, IoT has the potential to have an impact on business and society [8]. Internet of Things (IoT) effects may be seen from both a personal and a corporate standpoint. The Internet of Things (IoT) affects both professional and personal areas, such as smart workplaces and homes, e-health, and assisted living, from the standpoint of a private user. Automation and industrial production, logistics, business process management, and intelligent people and goods transportation are only few of the areas that might have an influence from a business user's perspective [9]

The Internet of Things (IoT) connects actual objects to the Internet. Software, hardware, and service providers are all part of IoT's infrastructure. The physical attributes and information they detect in their surroundings may be exchanged between the integrated physical objects. RFID, for example, may be used to identify gadgets since it allows each item to be individually recognised [10].

In order to meet the needs of diverse sectors, businesses, and the general public, the Internet of Things was built on a multi-tiered architecture. A general IoT layering design is shown in Fig. 2.1, and the following sections describe each layer: [13]

□ Edge Technology layer

Embedded systems, RFID tags, sensor networks, and other types of sensors are all included in this hardware layer. It is possible for this hardware layer to gather data from a system or environment, process data, and facilitate communication

. □ Access Gateway layer

An important part of this layer is dealing with data, and it is responsible for things

like service publication, subscription, and communication across platforms. □

Middleware layer

In addition to collecting and filtering the hardware-generated data, this layer also performs information discovery and provides application-specific device access control.

□ Application layer

There are a number of applications that rely on this layer for delivery.

Applications and consumers in IoT-based systems employ this middleware layer for access to these services. Various sectors, such as logistics, retail, healthcare, and others, may make use of the application services.

Challenges

We encounter a variety of issues since IoT-based systems may be utilised for a variety of purposes and in a variety of settings. Here, we'll go through some of the difficulties that researchers may run against.

□ Edge technology

IoT-based systems must be able to receive enough information about the actual world from a wide range of devices and surroundings, therefore additional research efforts are needed to build embedded devices, sensors, actuators, and (passive and active) identification. It will take time and effort to connect and deploy heterogeneous devices in IoT applications and enable support for future devices.

□ Networking technologies

Everything in the Internet of Things (IoT) is interconnected through a variety of networks, including wired and wireless. Networks like this allow for bidirectional

communication between IoT applications' real world devices, apps, and services. Interconnection should be possible with minimal energy usage, however dispersed data may raise privacy concerns.

□ □ Middleware system

Devices and networks in the Internet of Things (IoT) are a mixed bag. It's possible that new technologies may amplify their diversity. We need to protect these devices' heterogeneity so that they may be used by IoT applications. As a result, we must create a service-oriented middleware that is safe, scalable, and semantically enhanced in order to deal with the diversity of devices. All of the devices involved are managed in an integrated manner, which ensures scalability, high availability, and the safe and secure execution of desired functionality from the devices. While searching for the features that an IoT-based application layer might benefit from, our attention is drawn to the problems presented by middleware.

3.4.4 IoT_based Middleware

IoT-based systems have been researched by Bandyopadhyay, S. et al., who looked at the middleware systems that have been used [1]. Interface protocols, device abstraction, central control, context detection and management, and application abstraction are the four functional components of middleware needed to handle interactions with a range of devices (shown in Fig. 2.2). In the next sections, we go into further depth about these components.

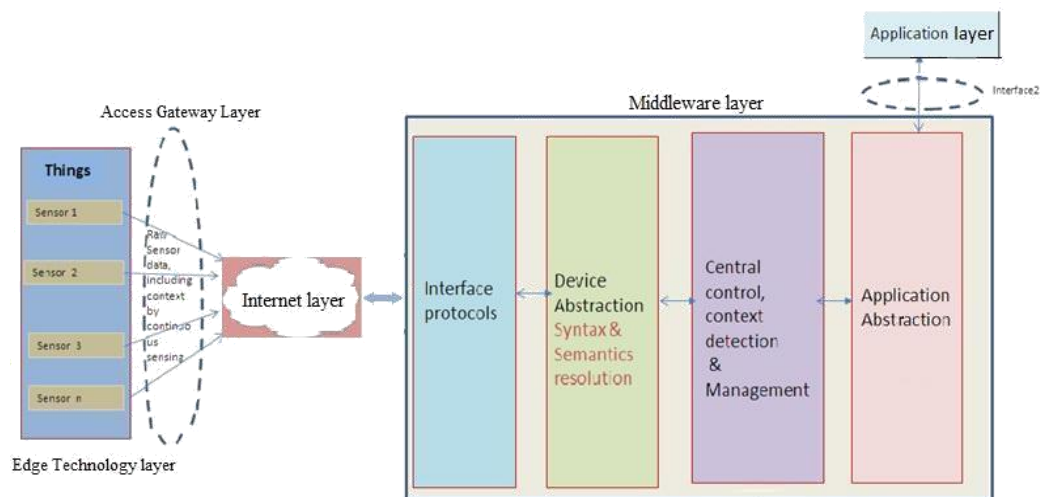


Figure 2.2 Functional components of a middleware for IoT-based systems

Interface protocols

Technical interoperability is the responsibility of this component. Interface protocols define interoperability as the ability of two systems to communicate with each other using the same set of protocols. Interoperability between hardware and software components, systems, and platforms is described by ETSI (European Telecommunications Standards Institute) [14] as the capacity to communicate between machines. An important consideration in (communication) interoperability is the underlying infrastructure required to support it[14].

In order to allow for technological compatibility, the Interface Protocol component offers protocols for information exchange across networks that may use various communication protocols. The TCP/IP stack's physical and data connection, network, transport, and application layers all fall within the purview of this component.

We may use a wrapper for each device to transform the protocol supported by the device to a common protocol in order to deal with the heterogeneity of devices.

You may put this wrapper on the device or in the middleware, depending on your needs. The wrapper should be on the middleware side if we wish to interface directly with devices. Wrappers on the middleware side are necessary since devices often have limited processing capabilities. A wrapper between the middleware and the devices may, on the other hand, be created in the event of an indirect interaction. Direct and indirect interactions may be supported by this component of the middleware.

Device Abstraction (DA)

It is the responsibility of this component to provide an abstract format so that the application components may communicate with devices more easily. As specified by the European Telecommunications Standards Institute (ETSI):

- *Syntactic interoperability* is associated with data formats. High-level transfer syntaxes such as HTML and XML may be used to indicate the syntax and encoding format of messages conveyed using communication protocols.
- Semantic interoperability is often related with the meaning of the message's content as understood by humans. Thus, interoperability on this level implies that individuals have a shared understanding of the meaning of the material (information) they exchange. However, since DA does not interact directly with humans, semantic interoperability in the context of DA is responsible for ensuring that apps share this shared understanding.

Semantic interoperability is predicated on domain-specific semantic models. For instance, one method of ensuring semantic compatibility in Service Oriented Architecture (SOA) [15] based middleware is via the use of Devices Profile for Web Services (DPWS) [16]. Each device type corresponds to a distinct service type in this context. The DA component has two main functions: (1) it may be used to request that devices do certain functions, and (2) it can be used to design and configure devices. DPWS utilizes the XML format described in Code 2.1.

```

<dpws:ThisModel>
  <dpws:Manufacturer>ACME Manufacturing</dpws:Manufacturer>
  <dpws:ModelName xml:lang="en-GB" >
    ColourBeam 9
  </dpws:ModelName>
  <dpws:ModelName xml:lang="en-US" >
    ColorBeam 9
  </dpws:ModelName>
</dpws:ThisModel>
</wsx:MetadataSection>
<wsx:MetadataSection
Dialect="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/ThisDevice"
>
  <dpws:ThisDevice>
    <dpws:FriendlyName xml:lang="en-GB" >
      ACME ColourBeam Printer
    </dpws:FriendlyName>
    <dpws:FriendlyName xml:lang="en-US" >
      ACME ColorBeam Printer
    </dpws:FriendlyName>
  </dpws:ThisDevice>

```

Code 2.1 XML code to define a device

Central control, Context detection & Management (CCM)

The term "context" refers to the state of an entity, which may be a location, a person, or an item that is significant to the user, applications, and their interactions [1]. The CCM functional component is responsible for enabling context-aware computation, which is a kind of computing that takes into account the context of the entities with which the system interacts. To operate in smart settings, a middleware for IoT-based systems must be context-aware [1]. The term "smart environment" refers to a physical world that is densely and imperceptibly interlaced with sensors, actuators, displays, and computational components, seamlessly integrated into ordinary things, and linked through a continuous network [17]. Context-awareness consists of two components:

- 1) Context detection, which entails gathering data from resources and identifying information that may affect the calculation.

2) Context processing, which entails using the obtained data to carry out a job or make a judgment.

Abstraction of Application

This functional component offers a means of interacting with devices for both high-level applications and end users. For instance, an interface may be Restful or written in a query-based language. A. Malatras et al. offer a service-oriented architecture-based middleware for data management and monitoring [18]. This middleware makes use of a Restful interface to ease the interaction of high-level applications with sensors that communicate over HTTP operations with Wireless Sensor Networks (WSNs): (1) GET is used to perform a query on an existing resource, (2) DELETE is used to delete an existing resource, (3) POST is used to create a new resource, and (4) PUT is used to update an existing resource. For example, a client application may get the result of a domain task by submitting a GET request to the following URL: "http://hostname/REST/version/DomaintaskResult/id". A reference to a "id" in this URI results in a unique result. The TinyDB middleware [19] enables end users to communicate with devices using a query language. For example, a user may make the following query to get the sensor's id.

.(nodeid) nearby and temperatures that are sensed by this sensor during the past 10 seconds before executing the query:

```
SELECT nodeid, temp
FROM sensors
SAMPLE PERIOD 1s FOR 10s
```

2.4 Conclusion

We defined IoT and identified common IoT layers in this chapter. Additionally, we addressed reference middleware architecture for Internet of Things (IoT)-based systems. S. Bandyopadhyay et al. suggested this design after conducting a research of current middleware frameworks for IoT-based applications [1].

Chapter 3.4 Review of IoT-based Middleware

The phrase "Internet of Things" encompasses a broad range of applications and research topics, including distributed computing and knowledge management. Much of the IoT research we discovered in the literature is concerned with ubiquitous computing. Pervasive computing is defined by the World Wide Web Consortium (W3C)³ as a vision for our future computer lifestyle. Computer systems will be smoothly interwoven into our daily lives, anticipating our requirements and offering us with appropriate services and information at any time and from any location. Pervasive Computer is all about facilitating human interaction via the use of existing computing technology [20]. This drives scientists to create innovative technologies that improve the lives of regular people.

3.5 COMPONENTS

NODE MCU

NodeMCU is a free and open source Internet of Things platform. [4] [5] It comprises software that works on Espressif Systems' ESP8266 Wi-Fi SoC and hardware that is based on the ESP-12 module.

[7] By default, the word "NodeMCU" refers to the firmware, not the development kits. The firmware is written in the Lua programming language. It is developed on the Espressif Non-OS SDK for ESP8266 and is based on the eLua project. It makes use of a number of open source projects, including lua-cjson[8] and SPIFFS.

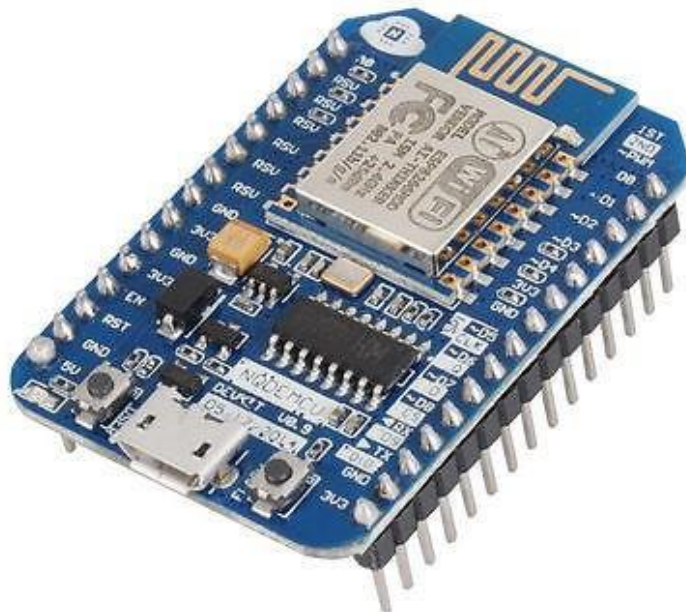
NodeMCU was built immediately after the release of the ESP8266. Espressif Systems[6] commenced mass manufacturing of the ESP8266 on December 30, 2013. [10] The ESP8266 is a Wi-Fi SoC with a Tensilica Xtensa LX106 core that is commonly utilized in Internet of Things (IoT) applications (see related projects). NodeMCU began on 13 October 2014, when Hong uploaded the first nodemcu-firmware file to GitHub. [11] Two months later, developer Huang R contributed the gerber file for an ESP8266 board titled devkit v0.9, expanding the project to include an open-hardware platform. [12] Later that month, Tuan PM adapted the Contiki MQTT client library to the ESP8266 SoC platform and contributed it to the NodeMCU project, enabling NodeMCU to implement the MQTT IoT protocol through Lua access to the MQTT broker. Another significant improvement occurred on 30 January 2015, when Devsaurus transferred the u8glib[14] library to the NodeMCU project[15], allowing NodeMCU to operate LCD, Screen, OLED, and even VGA displays with ease.

The founders abandoned the firmware project in summer 2015, and it was taken over by a group of independent contributors. By summer 2016, the NodeMCU has grown to include over 40 distinct modules. Due to resource limits, customers must pick the modules that are necessary for their project and create custom firmware.

ESP8266 Arduino Core[edit]

As Arduino.cc began developing new MCU boards based on non-AVR processors, such as the ARM/SAM MCU used in the Arduino Due, they needed to modify the Arduino IDE to make it relatively simple to add support for alternate tool chains that would allow Arduino C/C++ to be compiled for these new processors. They accomplished this by introducing the Board Manager and SAM Core. A "core" is a group of software components that the Board Manager and the Arduino IDE need to generate an Arduino C/C++ source file for the target MCU's machine language. Several ESP8266 enthusiasts created an Arduino core for the ESP8266 Wi-Fi SoC, which is often referred to as the "ESP8266 Core for the Arduino IDE." [16] This has developed into the industry's main software development platform for ESP8266-based modules and development boards, including NodeMCU.

NodeMCU is an open source firmware written in LUA for the ESP8266 wifi chip. By experimenting with the ESP8266 chip's capability, the NodeMCU firmware includes an ESP8266 Development board/kit, dubbed the NodeMCU Development board.

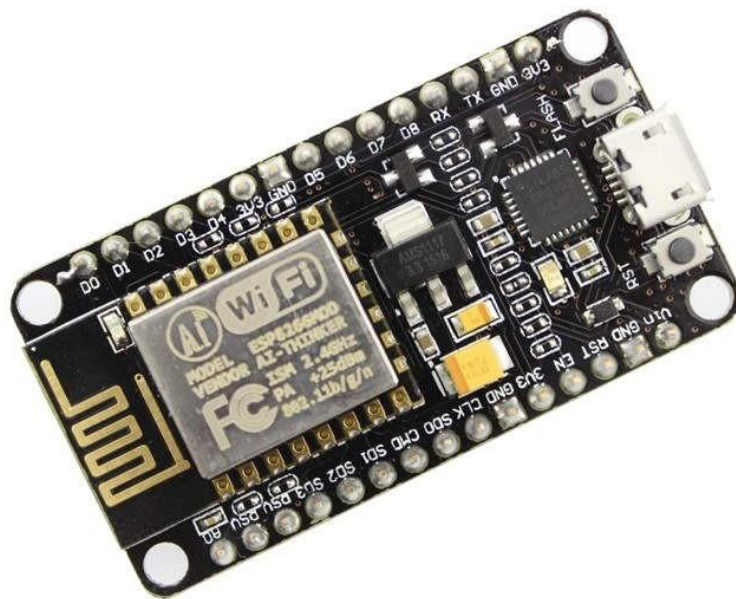


NodeMCU Development Board/kit v0.9 (Version1)

Because NodeMCU is an open source platform, its hardware design is available for anybody to edit, change, or expand upon.

The NodeMCU Development Kit/Board is comprised of an ESP8266 wifi-enabled chip. A low-cost Wi-Fi chip designed by Espressif Systems that supports the TCP/IP protocol is known as the ESP8266. For additional information on the ESP8266, please see the ESP8266 WiFi Module document.

There is a Version2 (V2) of the NodeMCU Dev Kit available, which is the NodeMCU Development Board v1.0 (Version2), which is typically available on a black coloured PCB.



NodeMCU Development Board/kit v1.0 (Version2)

The NodeMCU Dev Kit contains Analog (A0) and Digital (D0-D8) pins that are similar to those found on an Arduino. It is compatible with serial communication protocols such as UART, SPI, I2C, and others. Serial devices such as I2C enabled LCD displays, magnetometers HMC5883 and MPU-6050 Gyro meter + Accelerometer, RTC chips, GPS modules, touch screen displays, and SD cards, among others, may be connected to the computer over such serial protocols.

What is the best way to get started with NodeMCU?

The NodeMCU Development board has Wi-Fi functionality, analogue pins, digital pins, and serial communication protocols, among other capabilities. First and foremost, in order to get started with NodeMCU for Internet of Things applications, we must understand how to develop and download NodeMCU firmware in NodeMCU Development Boards. And, prior to that, where will this NodeMCU firmware be obtained in accordance with our requirements.

There are online NodeMCU custom builds accessible, which allows us to quickly and simply create our own custom NodeMCU firmware to meet our specific needs. More information on how to create customized NodeMCU firmware online and download it may be found at [Getting Started with the NodeMCU Programming Environment](#)

What is the best way to develop code for NodeMCU?

After configuring the ESP8266 with the Node-MCU firmware, let's have a look at the IDE (Integrated Programming Environment) that is necessary for NodeMCU development.

NodeMCU with ESPlorer IDE

In most cases, Lua scripts are used to program the NodeMCU. Lua is a scripting language that is open source, lightweight, and embeddable, and it is built on top of the C programming language.

More information on how to develop a Lua script for NodeMCU may be found at the following link. Getting started with NodeMCU and ESPlorer IDE is a simple process.

NodeMCU with Arduino IDE

Arduino IDE is a well-known IDE that may be used to create NodeMCU. NodeMCU apps may also be developed using the Arduino programming environment. Arduino developers no longer have to learn a new language or IDE to work with NodeMCU. If you want to learn more about writing an Arduino sketch for a NodeMCU, check out this article. Using the Arduino IDE to get started with NodeMCU

Difference in using ESPlorer and Arduino IDE

Using the ESPlorer IDE and the Arduino IDE, we can state that there is a programming language difference when building applications for the NodeMCU. If we use Arduino IDE to build NodeMCU program, we must use C++; if we use ESPlorer IDE, we must use Lua.

In a nutshell, NodeMCU is a Lua Interpreter, meaning it can read and write Lua scripts with ease. When we send/upload NodeMCU Lua scripts, they are executed in a sequential order. NodeMCU will not be able to write binary firmware files as a result of this. The NodeMCU will run the Lua script that was sent to it. It is possible to write and compile code in Arduino IDE, and then the ESP8266 toolchain in the background generates a binary firmware file of that code. When

we upload it to NodeMCU, the freshly created binary firmware code is flashed onto all NodeMCU devices. As a matter of fact, it creates the whole system's firmware. As a result, after being flashed using the Arduino IDE, NodeMCU will not accept any further Lua scripts or code. If we attempt to upload Lua scripts after they've been flashed by an Arduino sketch or code, we receive an error. Flashing the Lua script with the NodeMCU firmware is the only way to get it working again. The Arduino IDE is slower than the ESPlorer IDE since it compiles and uploads/writes the whole firmware.

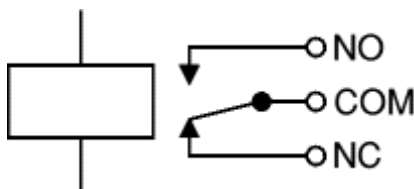
Relay:

A relay is a switch that is controlled by electricity. In a relay, current flowing through the coil generates a magnetic field, which attracts a lever and causes the switch contacts to be switched on and off. Consequently, relays have two switch positions and are double throw (changeover) switches since the coil current may be either on or off at the same time. It is possible to turn on and off one circuit while simultaneously switching on and off another circuit that is fully distinct from the first. For example, a relay may be used to switch a 230V alternating current mains circuit in a low voltage battery circuit. Because there is no electrical connection between the two circuits inside the relay, the link is made only by magnetism and mechanical connection.

For a relay, the coil must carry a significant amount of current. For a 12V relay, this current is normally 30mA, but it may be as high as 100mA for relays intended to function at lower voltages. This current cannot be provided by the vast majority of integrated circuit (IC) chips, therefore a transistor is often employed to boost the modest IC current to the higher amount needed by the relay coil. Because the maximum output current of the widely used 555 timer IC is 200mA, these devices are capable of directly powering relay coils without the need for further amplification.



When it comes to switching contacts, relays are often SPDT or DPDT, but they may have many more sets of switch contacts than that. For example, relays with four sets of changeover contacts are routinely available. Even though the majority of relays are built for PCB attachment, you may solder wires directly to the relay's pins if you take care to prevent melting the plastic housing of the device. The animated image depicts a relay in operation, complete with coil and switch contact connections. A magnet is attracted to a lever on the left when the coil is turned on, as can be seen in the image below. This lever is responsible for moving the switch contacts. There is one set of contacts (SPDT) in the front and another set of contacts (DPDT) in the background, resulting in the relay being DPDT.



The relay's switch connections are usually labeled COM, NC and NO:

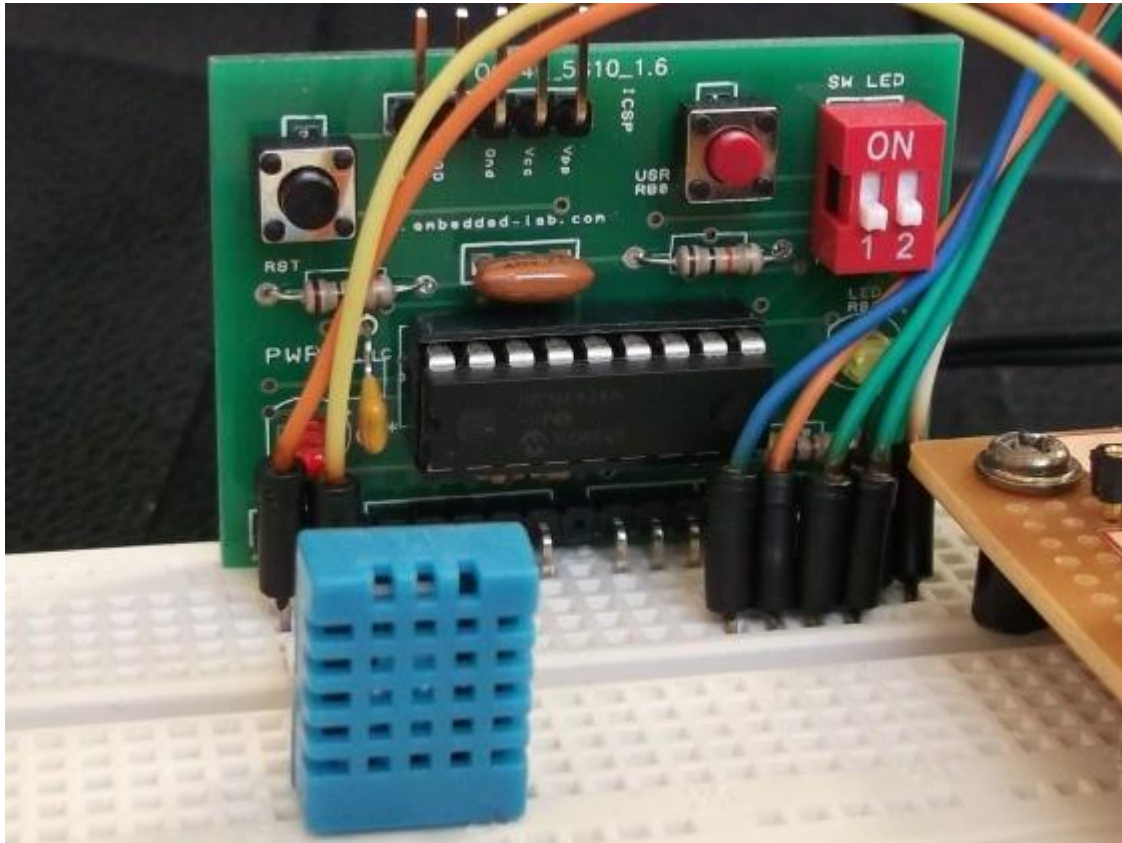
- **COM** = Common, always connect to this, it is the moving part of the switch.
- **NC** = Normally Closed, COM is connected to this when the relay coil is **off**.
- **NO** = Normally Open, COM is connected to this when the relay coil is **on**.

Humidity sensor

The measurement and management of temperature and relative humidity are useful in a wide range of situations and applications. These days, devices are available that have both temperature and humidity sensors, as well as signal conditioning, an ADC, calibration, and a communication interface, all of which are integrated within the device. The introduction of intelligent sensors like this substantially simplifies the design process and lowers the total cost.

In the past, we addressed the use of Sensirion's SHT1x/SHT7x sensors to detect humidity and temperature. These sensors are capable of sensing both temperature and relative humidity, and they produce digital outputs that are completely calibrated. Despite the fact that SHT1x/SHT7x sensors are very precise, they are still too costly for amateurs to utilise. While the DHT11 sensor, like the Sensirion sensors, offers calibrated digital outputs for temperature and humidity, this article focuses on how it is far less expensive than the Sensirion sensors.

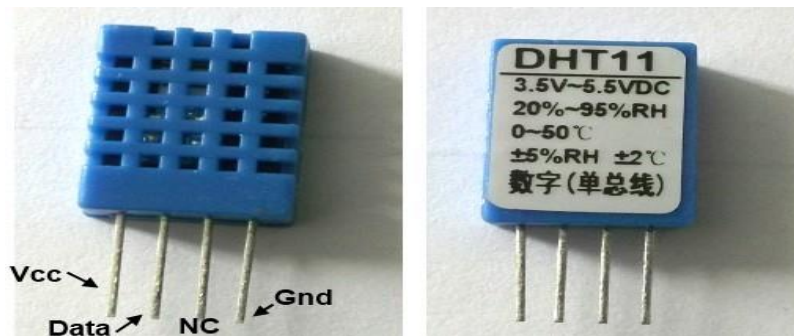
The DHT11 sensor communicates with a microcontroller using a proprietary 1-wire protocol, which we will be investigating and implementing with the PIC16F628A microcontroller, which will receive temperature and humidity readings from the sensor and display them on a 162 character LCD.



Interfacing DHT11 sensor with PIC16F628A

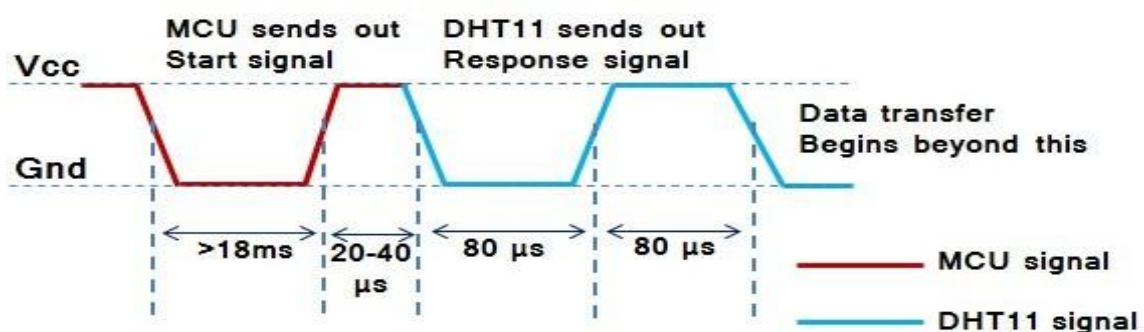
About DHT11 sensor

When it comes to power supply, the DHT11 sensor is available in a single row, four-pin package that runs on a 3.5 to 5.5V supply. It is capable of measuring temperatures ranging from 0 to 50 degrees Celsius with an accuracy of 2 degrees Celsius and relative humidity ranging from 20 to 95 percent with an accuracy of 5 percent. The sensor generates digital outputs that are perfectly calibrated for the two measurements. It operates on a proprietary 1-wire protocol, and as a result, communication between the sensor and a microcontroller is not feasible via a direct interface with any of the microcontroller's peripherals or through the use of a serial port. The protocol must be implemented in the firmware of the MCU in order for the sensor to operate at the exact time necessary.



DHT11 sensor comes in a single row 4-pin package

The data communication technique between a microcontroller and the DHT11 sensor is shown in the timing diagrams below. The "Start" signal is sent by the MCU to signify the start of data transfer. In order to do this, the MCU pin must be set as an output. Over at least 18 milliseconds, the MCU pulls the data line low, and then pulls it high for the following 20-40 milliseconds, before releasing the data line. Afterwards, the sensor replies to the MCU "Start" signal by pulling the line low for 80 milliseconds, followed by a logic high signal that likewise lasts for 80 milliseconds before being pulled low again. Keep in mind that the MCU pin must be set to accept data once the "Start" signal has been completed. The MCU should be ready to collect data from the sensor as soon as it detects the response signal from the sensor, which should happen immediately. The sensor then delivers a constant stream of 40 bits (5 bytes) of data across the data line. It is important to note that while transferring bytes, the sensor transmits the bit with the greatest significance first.



"Start" and "Response" signals

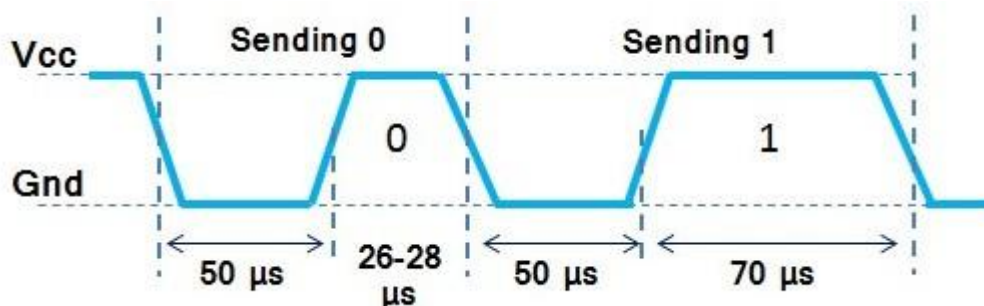
The 40-bit data from the sensor has the following structure.

Data (40-bit) = Integer Byte of RH + Decimal Byte of RH + Integer Byte of Temp. + Decimal Byte of Temp. + Checksum Byte

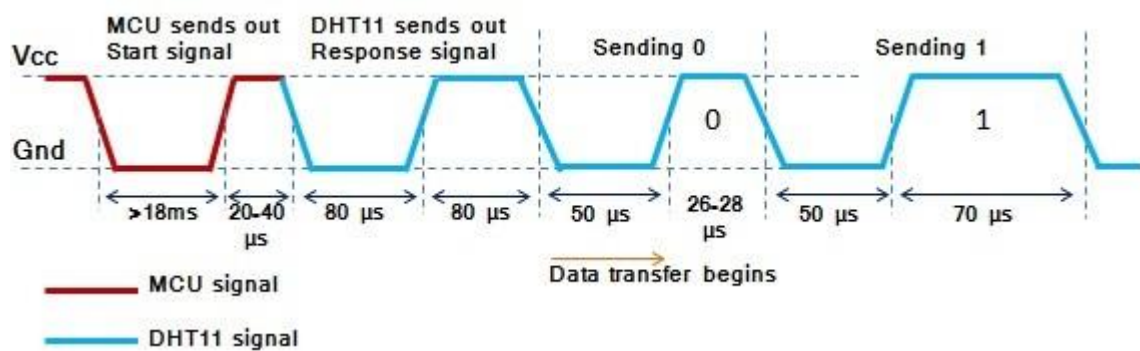
The decimal bytes of temperature and humidity data taken by the DHT11 sensor are always 0 for this sensor. The first and third bytes of the received data, as a result, really include the numeric values of the measured relative humidity (percentage) and temperature (in degrees Celsius). Lastly, the checksum byte is used to ensure that the data transmission has occurred successfully and without errors. If all five bytes are properly sent, then the checksum byte must be equal to the final 8 bits of the total of the first four bytes, i.e., the checksum byte must be equal to the sum of the first four bytes plus one.

Checksum = Last 8 bits of (Integer Byte of RH + Decimal Byte of RH + Integer Byte of Temp. + Decimal Byte of Temp.)

Let us now discuss the most crucial issue, which is signaling for the transmission of the numbers "0" and "1." In order to convey a single bit of data, the sensor first lowers the voltage on the line for 50 milliseconds. In order to broadcast "0", it elevates the line to high for 26-28µs, or for 70µs if the bit that has to be communicated is "1". As a result, the breadth of the positive pulse is what conveys information about the numbers 1 and 0.



Timing difference for transmitting "1s" and "0s"

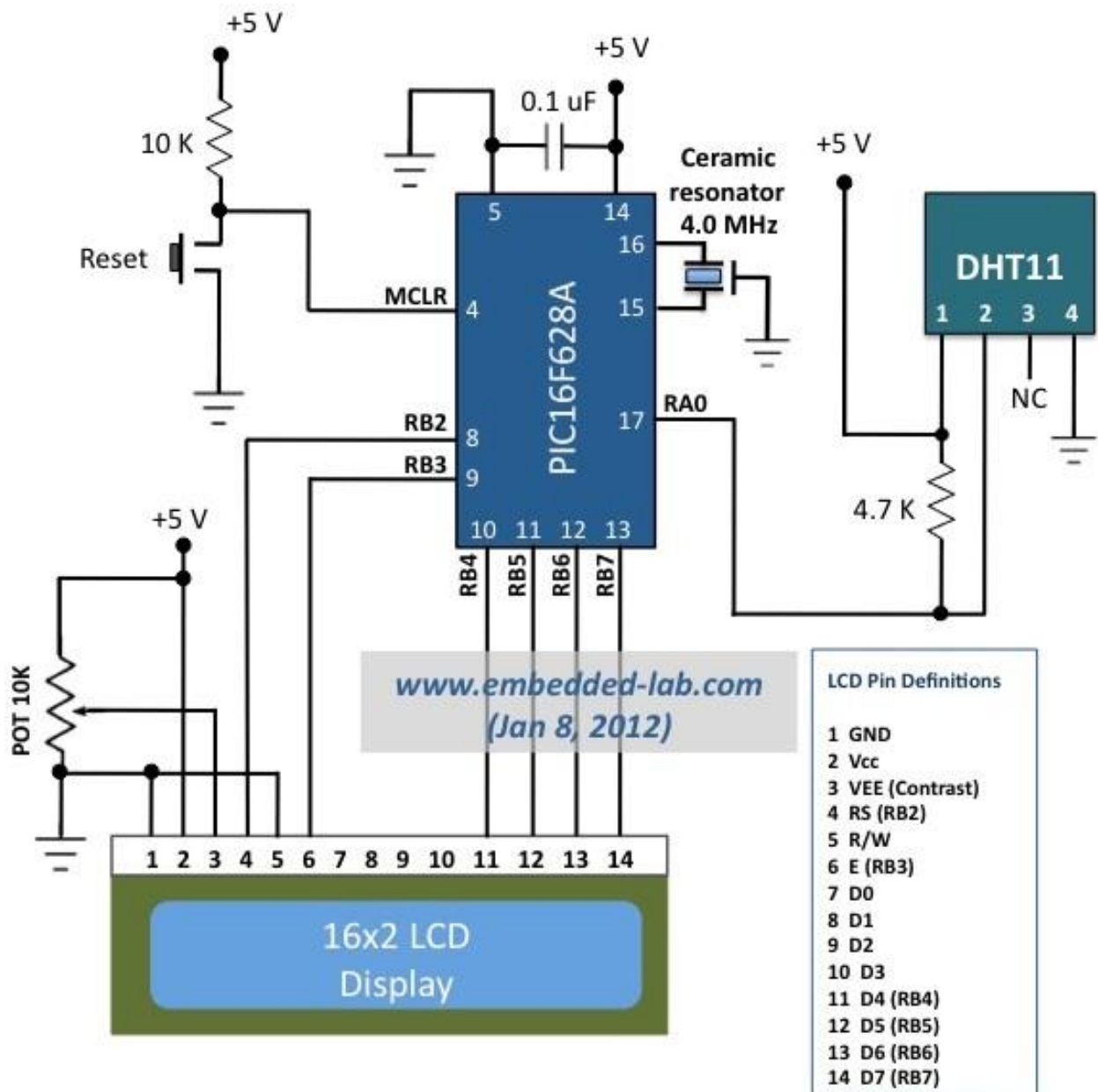


Start, Response and Data signals in sequence

At the conclusion of the final sent bit, the sensor pulls the data line low for 50 milliseconds before resuming normal operation. It is necessary to connect an external pull-up resistor between the DHT11 sensor's Vcc and the data line in order to ensure that the data line is always pushed high while the sensor is not in operation. Upon completion of the data transmission and release of the data line, the DHT11 sensor transitions to a low-power consumption mode until a fresh "Start" signal from the MCU is received and processed.

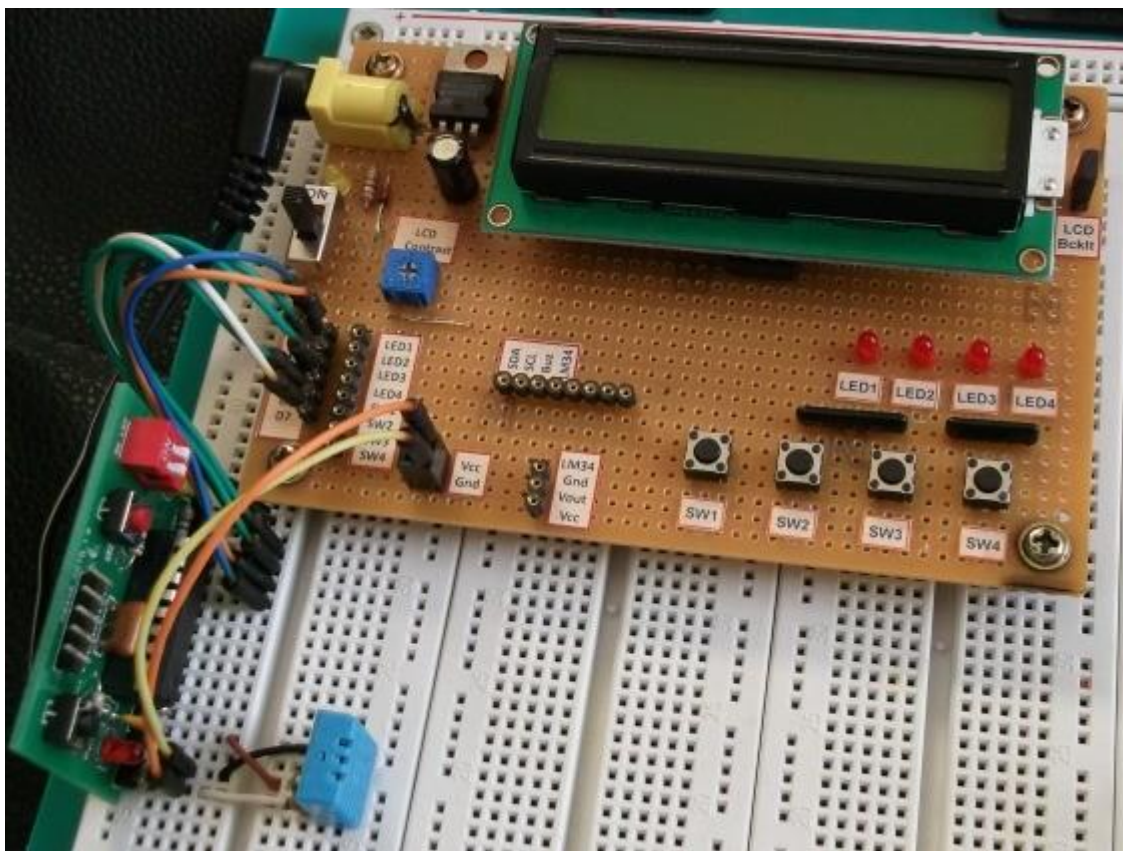
Circuit diagram

The circuit schematic below shows the DHT11 sensor in conjunction with an HD44780-based character LCD that is connected to the PIC16F628A microcontroller. The microcontroller operates at a clock frequency of 4.0 MHz, which is provided by an external resonator linked to the OSC1 (16) and OSC2 (15) ports. Since of the usage of a 4.0 MHz clock, it is easy to calculate timing because one machine cycle is equal to one μ s. The timing information will be used to compute the width of the data pulse received from the sensor, which will allow us to determine whether it is carrying a 1 or a 0 in the data.

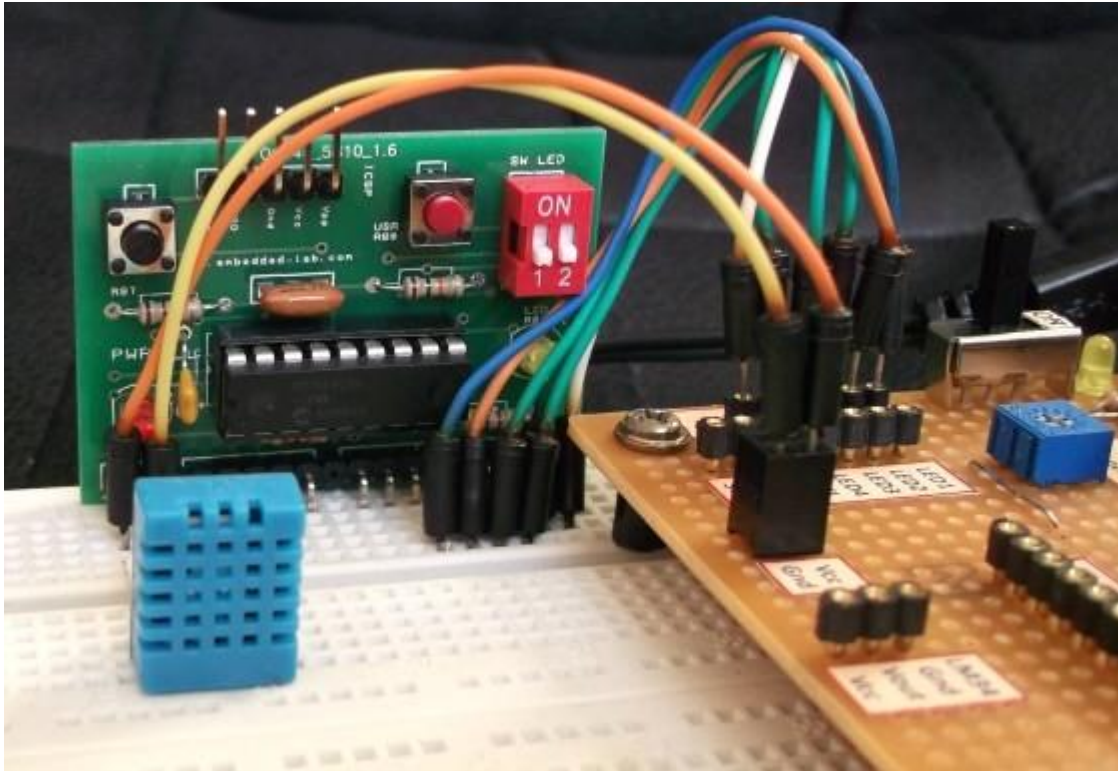


Circuit connections for PIC16F628A and DHT11 sensor

The circuit configuration on a breadboard is shown in the following images. Keep in mind that the four LEDs and touch switches depicted on the perforated board are not to be mistaken with each other. They are not involved in this initiative in any way. They are included because the LCD portion of this project will be implemented utilising the I/O board from my DIY Experimenter. As an example, I'm utilising my 18-pin PIC16F board to make prototyping with the PIC16F628A microcontroller a lot more convenient.



Complete setup of the circuit



PIC16F628A module and the DHT11 sensor are plugged into the breadboard

Software

Because of the timing restrictions for 1s and 0s, writing software for the DHT11 sensor is a bit more difficult than writing software for the hardware component. Using mikroC Pro for PIC, I've built subroutines for initialising the DHT11 sensor and reading the 40-bits of data in succession. I've used the Timer2 module to keep track of the width of the data pulse that was received, which is necessary in order to determine if the received bit was 1 or 0. In the event of the detection of a pulse ranging from low to high at the beginning of any data bit, TMR2 is cleared and switched ON. Because the clock frequency utilised in this example is 4.0 MHz, the TMR2 advances by one every one-hundredth of a second. When the data pulse falls below a certain threshold, the TMR2 is shut off. Using the value of the TMR2 register, you can find out the width of the data pulse in microseconds. For the sake of distinguishing between 0 and 1, I am adopting a 40-second threshold. If the TMR2 is more than 40, it indicates that the received bit is 1, otherwise it indicates that it is 0. Here is the whole source code for the PIC microcontroller

written in mikroC Pro. It can be readily changed to any other platform, but keep in mind that if you are using a different clock frequency than the one specified in the code, you will need to adjust the timer operation.

SOIL MOISTURE SENSOR:

Soil moisture sensors are devices that detect the amount of volumetric water present in soil. [1] Rather than removing, drying, and weighting a sample as is required for direct gravimetric measurement of free soil moisture, soil moisture sensors measure the volumetric water content indirectly by using some other property of the soil as a proxy for the moisture content. Examples of such properties include electrical resistance, dielectric constant, interaction with neutrons, and interaction with ions.

Calibration of the relationship between a measured property and soil moisture is required, and the relationship may vary based on environmental conditions such as soil type, temperature, or electric conductivity. Reflected microwave radiation is influenced by soil moisture and is utilized for remote sensing in hydrology and agriculture, among other applications. Farmers and gardeners may benefit from the usage of portable probing tools.

Sensors that assess the volumetric water content of soil are often referred to as soil moisture sensors. The water potential of soils is measured using another type of sensors, which are referred to as soil water potential sensors and consist of tensiometers and gypsum blocks.

Agriculture:

When it comes to agricultural applications, measuring soil moisture is critical since it allows farmers to control their irrigation systems more effectively. Farmers who know the precise soil moisture levels on their fields are able to not only use less water to produce a crop in general, but they are also able to boost

yields and the quality of the product through better control of soil moisture at important plant development phases.

Landscape irrigation

Landscapes and residential lawns in urban and suburban regions are employing soil moisture sensors to communicate with an irrigation controller, which is connected to a watering system. Connecting a soil moisture sensor to a standard irrigation clock can transform it into a "smart" irrigation controller that will prevent irrigation cycles from taking place when the soil is already wet, such as after a recent rainstorm event, for example.

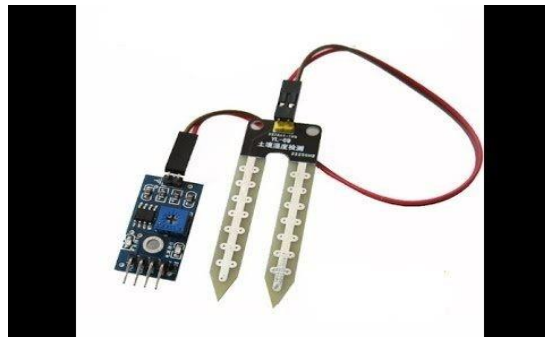
In order to reduce over-watering and the leaching of fertilizers and other chemicals into the ground, golf courses are using soil moisture sensors to improve the efficiency of their irrigation systems.

Research

Agricultural science and horticulture, for example, including irrigation planning, climate research, or environmental science, for example, including solute transport studies and as auxiliary sensors for soil respiration measurements, are among the many uses for soil moisture sensors.

Simple sensors for gardeners

There are relatively inexpensive and simple gadgets available that do not need a power source to determine if plants have adequate moisture to flourish. A metre determines if the soil is too dry, too damp, or too wet for plants after placing a probe into the soil for around 60 seconds.



UBIDOTS EXPLORER:

Introduction

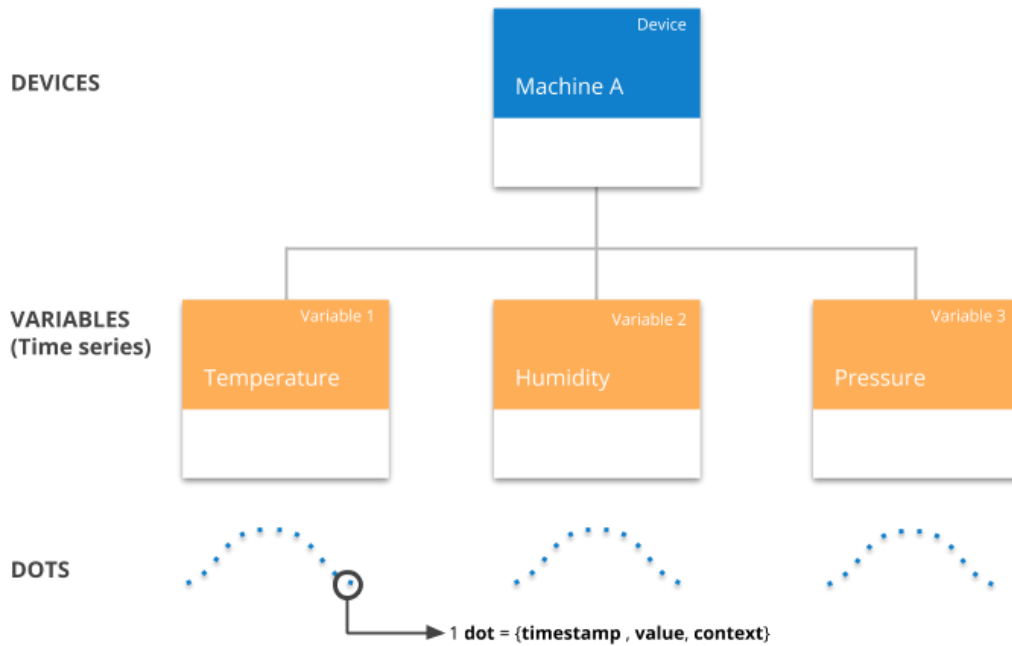
Welcome to the hardware documentation for the Ubidots REST API. Here you can find information on how to transmit and get data from your hardware devices using the communication protocols that we support: HTTP, MQTT, and TCP/UDP, among others.

Using this part, you will have a better understanding of what occurs in the backstage while connecting with Ubidots, allowing you to mimic this behavior in your firmware later. As a result, we refrain from utilizing examples that are based on libraries. In contrast, if you want to utilise one of our libraries (Python, C, and others), please see our sections on supported devices and API clients.

What is the Ubidots Method of Operation?

An instance of a data-point or "dot" is formed every time a device changes a sensor value in a variable. This software records dots that are generated by your devices in variables, and these saved dots have associated values. **timestamps:**

UBIDOTS DATA HIERARCHY

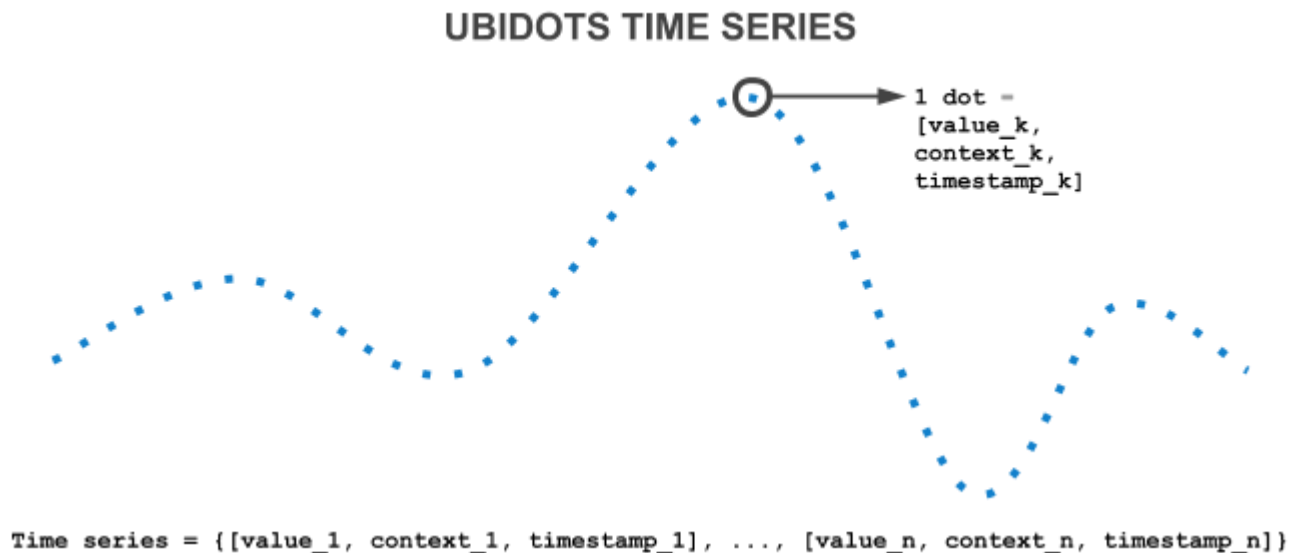


Each dot contains these items:

Item	Description	Mandatory
value	A numerical value. Ubidots accepts up to 16 floating-point length numbers.	YES
timestamp	Unix Epoch time, in milliseconds. If not specified, then our servers will assign one upon reception.	NO
context	An arbitrary collection of key-value pairs. Mostly used to store the latitude and longitude coordinates of GPS devices.	NO

Time Series

Based on the above, we can illustrate an Ubidots time series like this:



Ubidots is a hardware-agnostic platform, which means that it doesn't really care what kind of hardware device you're using as long as you're able to communicate with us using at least one of the following protocols

- HTTP
- MQTT
- TCP/UDP

HTTP

HTTP is the most widely used internet protocol for exchanging information on the internet. It also allows data encryption, which is referred to as secure HTTP or HTTPS in most cases. When you use a web browser (such as Microsoft Edge, Firefox, Google Chrome, or another similar program), you are sending and receiving packets using the HTTP or HTTPS protocol. The fact that HTTPS has become a popular alternative for data sharing in the Internet of Things is not unexpected in this context.

It is recommended that a developer seek for an Application Programming Interface, or REST API, that exposes all of the endpoints necessary to connect

with a third-party application when implementing HTTP communication. REST APIs are available from a variety of vendors (like Ubidots). Our REST API is described in detail in this part, so that you can understand how to send and get data from our servers.

Please keep in mind that the purpose of this part is to assist you in understanding what is going behind the scenes while connecting with Ubidots, so that you can recreate it inside your own firmware. As a result, we refrain from using any of our proprietary libraries in any of the examples.

HTTP requests

The following methods are specified within the *HTTP* standard:

HTTP Method	Description
GET	Used to retrieve information
POST	Used to create a new element
PATCH	Used to update existing elements
DELETE	Used to delete existing elements

A request/response protocol, HTTP is defined as one in which every request made by the client is met by a response from the server. This answer consists of two parts: a number (the response code) and a body. In the case of a request to obtain a file from a website "(for example, "Get me the file 'websiteside.html'")", you construct a GET request. If the request is valid, the server will normally respond with a 200 response code, along with the file that was originally requested (body).

In addition, the following parameters are required for an HTTP request:

- **Host:** This parameter specifies the server to which HTTP requests will be sent.
- Path:** This is typically the remaining portion of the URL that specifies the resource you want to consume, be it a variable or a device. For example, if an

API endpoint is: `industrial.api.ubidots.com/api/v1.6/devices/my-device` then the path would be `/api/v1.6/devices/my-device`

- Headers: These are the parameters that define the operation of the HTTP request, such as authentication, Content-Type, Content-Length, and so on.

It is possible to send data to a server using a POST or PATCH request. The data sent by your device will be included in the body/payload of your request. The body of GET requests is often absent since they are meant to seek information rather than to convey information. Ubidots takes data in the form of JavaScript Object Notation (JSON) or XML. This is a collection of name/value pairs. JSON is an HTTP data format that is used in web applications. A record, struct, dictionary, hash table, keyed list, or associative arrays are all terms that are used to describe this kind of data in different computer languages. It is also easily understandable by humans and is not limited to a single language. As an illustration, the following is an example of a JSON data type that Ubidots supports:

```
{"temperature": {"value":10, "timestamp": 1534881387000,
"context": {"machine": "1st floor"}}}
```

A typical *HTTP* request to Ubidots should be set as below: `POST {PATH}`

`HTTP/1.1<CR><LN>`

`Host: {HOST}<CR><LN>`

`User-Agent:`

`{USER_AGENT}<CR><LN>`

`X-Auth-Token:`

`{TOKEN}<CR><LN>`

`Content-Type: application/json<CR><LN>`

`Content-Length: {PAYLOAD_LENGTH}<CR><LN><CR><LN>`
`{PAYLOAD}`

`<CR><LN>`

.

API URLs

API access is available through HTTP or secure HTTP, HTTPS, via the following endpoints, depending on the kind of Ubidots Account you have.

Note on Security: We highly urge you to utilise HTTPS to ensure that your data is encrypted during transmission, preventing the disclosure of your API token and/or sensor data.

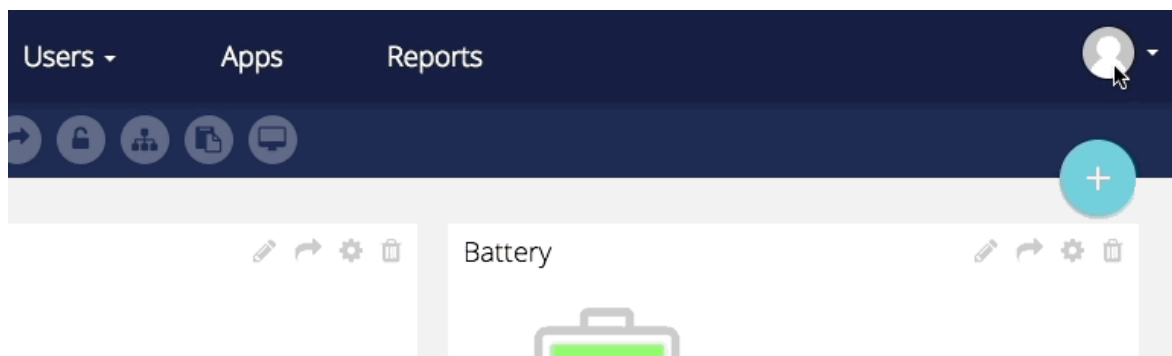
Ubidots is compatible with SSL v1.1, TLS v1.2, and TLS v1.3. Our root certificates are available in the following formats:

Ubidots also supports set IP addresses for data input, although we strongly suggest our customers to construct their firmware routines using dns endpoints rather than fixed IP addresses, since these addresses are subject to change in the future.

HTTP Authentication

Each request must include a token. A token is a one-of-a-kind key that enables your device to access data stored in your Ubidots account.

To get yours, just click on "API Credentials" under your profile tab:



Send data

Our REST API allows you send data to the platform in two different ways:

- Send data to a **device**
- Send data to a **variable**

If you're seeking for more complex API features, such as altering the name of a device or displaying all of your devices, please consult our Ubidots Software REST API Reference.

Send data to a device

4. SOURCE CODE:



sketch_mar08a \$

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

#define DHTPIN D5

#define DHTTYPE    DHT11

DHT_Unified dht(DHTPIN, DHTTYPE);
uint32_t delayMS;
|
#include "Ubidots.h"

/*****
 * Define Instances and Constants
 *****/

const char* UBIDOTS_TOKEN = "BBFF-gpohKZYdES7onQG2YyZ0imx0zlocit"; // Put here your Ubidots TOKEN
const char* WIFI_SSID = "automation"; // Put here your Wi-Fi SSID
const char* WIFI_PASS = "12345678"; // Put here your Wi-Fi password
const char* DEVICE_LABEL_TO_RETRIEVE_VALUES_FROM = "483fda9252d9"; // Replace with your device label
const char* VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM1 = "light"; // Replace with your variable label
const char* VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM2 = "fan";
const char* VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM3 = "television";

Ubidots ubidots(UBIDOTS_TOKEN, UBI_TCP);

void setup() {
  Serial.begin(9600);
  dht.begin();
  Serial.println(F("DHTxx Unified Sensor Example"));
  sensor_t sensor;
  dht.temperature().getSensor(&sensor);
  dht.humidity().getSensor(&sensor);
  ubidots.wifiConnect(WIFI_SSID, WIFI_PASS);

  pinMode(D0, OUTPUT);
  pinMode(D1, OUTPUT);
  pinMode(D2, OUTPUT);
  pinMode(D3, OUTPUT);
```



Upload

sketch_mar08a \$

```
-
digitalWrite(D0,HIGH);
digitalWrite(D1,HIGH);
digitalWrite(D2,HIGH);
digitalWrite(D3,HIGH);
}

void loop()
{
  float value1 = ubidots.get(DEVICE_LABEL_TO_RETRIEVE_VALUES_FROM, VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM1);
  float value2 = ubidots.get(DEVICE_LABEL_TO_RETRIEVE_VALUES_FROM, VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM2);
  float value3 = ubidots.get(DEVICE_LABEL_TO_RETRIEVE_VALUES_FROM, VARIABLE_LABEL_TO_RETRIEVE_VALUES_FROM3);

  if (value1 != ERROR_VALUE && value2 != ERROR_VALUE && value3 != ERROR_VALUE) {
    Serial.print("Value1:");Serial.print(value1);Serial.print("\t");
    Serial.print("Value2:");Serial.print(value2);Serial.print("\t");
    Serial.print("Value3:");Serial.print(value3);Serial.println("\t");
    digitalWrite(D0,!value1);
    digitalWrite(D1,!value2);
    digitalWrite(D2,!value3);
  }

  float soil = analogRead(A0);
  Serial.println(soil);
  if (soil > 900)
  {
    digitalWrite(D3,LOW);
    Serial.println("soil on");
  }
  else
  {
    digitalWrite(D3,HIGH);
    Serial.println("soil off");
  }

  sensors_event_t event;
  dht.temperature().getEvent(&event);
  if (isnan(event.temperature)) {
    Serial.println(F("Error reading temperature!"));
  }
  else {
```



sketch_mar08a \$

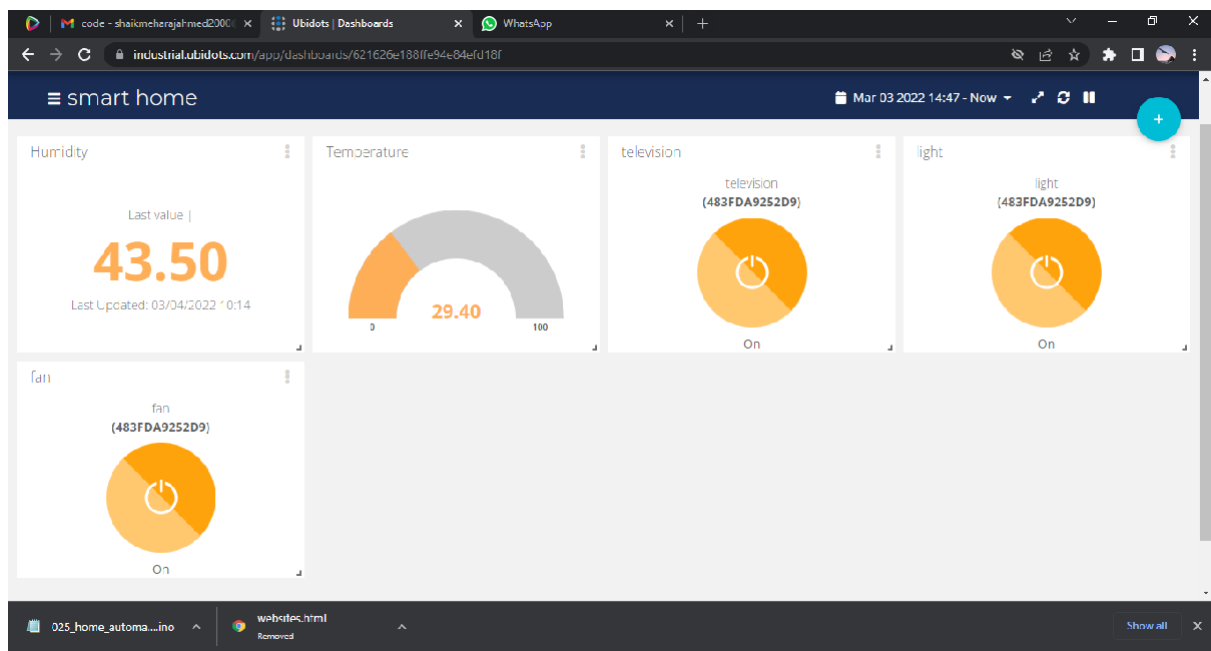
```
    Serial.println("soil on");
  }
  else
  {
    digitalWrite(D3,HIGH);
    Serial.println("soil off");
  }

  sensors_event_t event;
  dht.temperature().getEvent(sevent);
  if (isnan(event.temperature)) {
    Serial.println(F("Error reading temperature!"));
  }
  else {
    Serial.print(F("Temperature: "));
    Serial.print(event.temperature);
    ubidots.add("Temperature", event.temperature);
    Serial.println(F("~C"));
  }
  // Get humidity event and print its value.
  dht.humidity().getEvent(sevent);
  if (isnan(event.relative_humidity)) {
    Serial.println(F("Error reading humidity!"));
  }
  else {
    Serial.print(F("Humidity: "));
    Serial.print(event.relative_humidity);
    ubidots.add("Humidity", event.relative_humidity);
    Serial.println(F("%"));
  }

  bool bufferSent = false;
  bufferSent = ubidots.send();

  if (bufferSent) {
    Serial.println("Values sent by the device");
  }
  delay(5000);
}
```

5. RESULTS AND DISSCUSSION:



Because of the use of ubidots in the cloud, we have successfully executed the code in the Microcontroller and obtained the desired results.

6. CONCLUSION:

It is the ESP8266 NodeMCU micro-controller board that is employed as the system's central processing unit. It is possible to utilise the system as smart Home Automation by just speaking a voice command into a smartphone, which is activated by the user using Google Assistant in conjunction with the co-interfaced IFTTT (If This Then That), as well as the Blynk app platform, which enabled this capability.

The multichannel relay module for loads is connected to the central unit through an interface cable. The data is processed by the central unit in accordance with the user's input. It has been utilised for Agricultural Monitoring and has been assessed and shown on the OLED screen environmental factors such as soil moisture, air temperature, and humidity (DHT11).

The water motor was used to regulate the flow of water in accordance with the soil moisture threshold value. The ESP8266 Wi-Fi-based NodeMCU, which is based on the Hypertext Transfer Protocol, is routed to the Wi-Fi network and communicates with cloud storage (Thingspeak) and the IOT platform Blynk.

The data from the sensors that measure the required environmental parameter has been gathered, monitored, and then sent to the storage cloud, as well as to the Blynk and ubidots clouds, through Wi-Fi. A single hardware system is used to automate two separate domains, which is the subject of this study. The system has successfully completed its operation.

7. REFERENCES:

- [1] Ovidiu Vermesan, Peter Friess, Internet Of Things Applications: From Research and Innovation To Market Deployment, River Publishers Series In Communications, 2014.
- [2] Prahlad Bhadani, Dr. V. Vashisht, "Soil Moisture, Temperature, and Humidity Measurement Using Arduino", International Conference on Cloud Computing, Data Science & Engineering, Jan 2019, pp. 567- 571.
- [3] L. Calcara, G. Felice, F. Ferrucci, F. Marchetti, "Soil thermal resistivity: innovative measurement system", IEEE International Conference on Environment and Electrical Engineering, June 2019.
- [4] N. Agrawal, S. Singhal, "Smart Drip Irrigation System using Raspberry Pi and Arduino", International Conference on Computing, Communication and Automation, May 2015, pp. 928- 932.
- [5] H. Durani, M. Sheth, M. Vagharia, S. Kotech, "Smart Automated Home Application using IoT with Blynk App", Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies, April 2018, pp. 393-397.
- [6] M. Gamba, A. Gonella, C. Palazzi, "Design Issues and Solutions in a Modern Home Automation System", 2015 International Conference on Computing, Networking and Communications, Feb. 2015, pp. 1111-1115.
- [7] K. Ghanshala, R. Chauhan, R. C. Joshi, "A Novel Framework for Smart Crop Monitoring Using the Internet of Things (IoT)", International Conference on Secure Cyber Computing and Communication, Dec. 2018 pp. 62-67.
- [8] S. Ivanović, S. Milivojša, T. Erić, M. Vidaković, "Collection and Analysis of System Usage Data in Smart Home Automation Systems", International

Conference on Consumer Electronics, December 2017, pp. 65-66.

[9] R.K.Jha, S.Kumar, K.Joshi, R.Pandey, "Field Monitoring Using IoT in Agriculture", International Conference on Intelligent Computing, Instrumentation and Control Technologies, July 2017,pp.1417-1420.

[10] K. Krishna Kishore, M. H. Sai Kumar, M. B. S. Murthy, "Automatic Plant Monitoring System", International Conference on Trends in Electronics and Informatics,2017,pp.744-748