

IOT WEATHER MONITORING USING IOT CLOUD

Submitted in partial fulfilment of the requirements for the award of

Bachelor of Engineering Degree in Electronics and Communication Engineering

By

THAMMURI SREEKARAN VARMA (38130223)

SAGI VENKATA SRINIVASA BHARGAV VARMA (38130190)



**DEPARTMENT OF ELECTRONICS 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

MAY - 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
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 **T. SREEKARAN VARMA (38130223)** and **SVS.BHARGAV VARMA (38130190)**, who carried out the project entitled “**IOT WEATHER MONITORING USING IOT CLOUD**” under my supervision from NOVEMBER 2020 to MARCH 2021.

Internal Guide

Dr. S. EMALDA ROSLIN, M.E., Ph.D.,

Head of the Department

Dr. T. RAVI, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

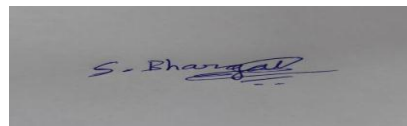
Internal Examiner

External Examiner

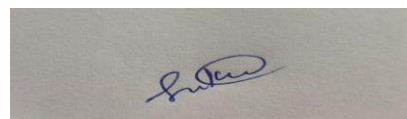
DECLARATION

We, **T. SREEKARAN VARMA (38130223)** and **SVS. BHARGAV VARMA (38130190)**, hereby declare that the Project Report entitled “**IOT WEATHER MONITORING USING IOT CLOUD**” is done under the guidance of **Dr. S. EMALDA ROSLIN, 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.

Date:

A rectangular box containing a handwritten signature in blue ink. The signature appears to be "S. Bhargav" with a stylized flourish at the end.

Place:

A rectangular box containing a handwritten signature in blue ink. The signature appears to be "T. Sreekaran Varma" with a stylized flourish at the end.

Signature of the Candidate

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. S. EMALDA ROSLIN, M.E., Ph.D.**, Department of Electronics and Communication Engineering to complete this project successfully.

ABSTRACT

The system proposed in this paper is an advanced solution for monitoring the weather conditions at a particular place and make the information visible anywhere in the world. The technology behind this is Internet of Things (IoT), which is an advanced and efficient solution for connecting the things to the internet and to connect the entire world of things in a network.

Here things might be whatever like electronic gadgets, sensors and automotive electronic equipment. The system deals with monitoring and controlling the environmental conditions like temperature, relative humidity, light intensity and CO level with sensors and sends the information to the web page and then plot the sensor data as graphical statistics. The data updated from the implemented system can be accessible in the internet from anywhere in the world.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	v
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
	LIST OF ABBREVIATIONS	xi
1	INTRODUCTION	1
	1.1 IOT WEATHER MONITORING USING IOT CLOUD	1
	1.2 OBJECTIVES	1
2	LITERATURE REVIEW	3
	2.1 LITERATURE SURVEY	3
	2.2 SUMMARY OF EXISTING SYSTEM	4
3	PROPOSED METHODOLOGY	5
	3.1 INTRODUCTION TO PROPOSED SYSTEM	5
	3.2 BLOCK DIAGRAM OF PROPOSED SYSTEM	5
	3.3 SYSTEM SPECIFICATIONS	6
	3.3.1 Power Supply Unit	9
	3.3.1.1 Circuit Diagram and it's Description	9
	3.3.2 Working Principle	9
	3.3.3 Sensors	10
	3.3.3.1 <i>Smoke Sensor (Gas Sensor)</i>	10
	3.3.4 LDR – Light Sensor	13
	3.3.4.1 Dependent resistor circuit and working principle	13

CHAPTER NO	TITLE	PAGE NO
	3.3.4.2 Construction of an LDR	14
	3.3.5 Working Principle of Light Dependent Resistor	14
	3.3.6 Light Dependent Resistor Circuit	15
	3.3.7 Applications of LDR	16
	3.3.8 Power Conserving of Intensity Controlled Street Lights using LDR	17
	3.3.9 Humidity Sensor	18
	3.3.10 Circuit Diagram	22
	3.3.11 Software	23
	3.3.12 BMP 180 Sensor (Pressure Sensor)	25
	3.4 ARDUINO IDE SOFTWARE DESCRIPTION	26
	3.5 UBIDOTS EXPLORER	29
4	NEURAL NETWORKS FOR RAIN PREDICTION	36
	4.1 NEURAL NETWORK TRAINING (nntraintool)	38
	4.2 COMPARISON OF ACTUAL AND DESIRED OUTPUT	39
	4.3 CODE	39
5.	RESULTS AND CONCLUSION	42
	5.1 RESULTS	42
	5.2 CONCLUSION	45
	REFERENCES	46

LIST OF TABLES

TABLE NO	TITLE	PAGE NO
4.1	BPN Parameters for Rain Prediction	37

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
3.1	NODEMCU (ESP8266)	8
3.2	Power Supply Unit Circuit Diagram	9
3.3	Working Principle	9
3.4	Smoke Sensor	11
3.5	Light Dependent Resistor	13
3.6	Construction of an LDR	14
3.7	Light Dependent Resistor	15
3.8	Light Dependent Resistor Circuit Diagram	16
3.9	Intensity Controlled Street Light using LDR	17
3.10	Interfacing DHT11 sensor with PIC16F628A	19
3.11	DHT11 sensor comes in a single row 4-pin package	20
3.12	"Start" and "Response" signals	21
3.13	Start, Response and Data signals in sequence	22
3.14	Circuit connections for PIC16F628A and DHT11 sensor	23
3.15	BMP 180 Sensor	26
3.16	Arduino software	26
3.17	Arduino Uno - R3	27
3.18	USB Cable A to B - 6 Foot	27
3.19	Ubidots Data Hierarchy	30
3.20	Ubidots Time Series	31
3.21	HTTP Authentication	35
4.1	Architecture of a Back Propagation Network	37
4.2	nntrain tool	38
4.3	Comparison of actual and desired output	39
5.1	Hardware image 1	42
5.2	Hardware image 2	42
5.3	Hardware image 3	43

FIGURE NO	TITLE	PAGE NO
5.4	Hardware image 4	43
5.5	Ubidots Readings of Parameters	44
5.6	Graph of parameter readings	44

LIST OF ABBREVIATIONS

BPN	-	Back Propagation Network
GPIO	-	General Purpose Input Output
IoT	-	Internet of Things
LDR	-	Light Dependent Resistor
PIC	-	Programmable Integrated Circuit
PLC	-	Programmable Logic Controller

CHAPTER 1

INTRODUCTION

1.1 IOT WEATHER MONITORING USING IOT CLOUD

Nowadays, the application of Internet of Things (IoT) affects all habitual domains[1]. The invented IoT systems are attempted to control, manage, and monitor human usual actions, environmental parameters, or animal movements.

So, all these innovations are developed to facilitate human work and make the life easier than before. In this stage, the environmental monitoring systems are invented to measure and control the environment parameters.

For many years, humans try to understand their environment. So, humans have invented many objects to measure various parameters. For example, humans have created thermometer, barometer, and pyrometer for measuring temperature, atmospheric pressure, and solar radiation, respectively.

However, those traditional tools must be used locally. On the other hand, with the invention of IoT, humans can measure all environment parameters remotely. Thus, the procedure of quantifying and measuring environmental parameters has become simple than in previous days.

1.2 OBJECTIVES

- To develop a Weather Monitoring System using IOT Cloud.

The detailed objectives are

- To monitor and control the environmental conditions like temperature, relative humidity, light intensity and CO level with sensors.
- To send the information to the web page and then plot the sensor data as graphical statistics.
- To make the updated data from the implemented system accessible in the internet from anywhere in the world.

CHAPTER 2

LITERATURE REVIEW

2.1 LITERATURE SURVEY

Development of IoT Based Weather Reporting System (A F Pauzi and M Z Hasan)

Weather forecast these days is unpredictable too be exact because of the climate changes drastically over weather. In cause of that, Weather Reporting System is mostly used to monitor the continuously changing climatic and weather conditions over controlled areas likes house, industry, agriculture and etc. in real time monitoring. Internet of Things (IoT) platform use is ThingSpeak it's should be able displaying the weather parameters and the information will visible wherever in the world and it's also displaying on the OLED with two-way microcontroller communication via Wi-Fi hotspots. [1]

Real Time Weather Monitoring System Using lot (Puja Sharma and Shiva Prakash)

In Today's World, knowing live environmental condition is one of the biggest issues because there is an IoT of hurdles arrives when live environmental condition is measured. The proposed system will remove this problem since it monitors real-time weather conditions. In this proposed work we will monitor the live weather's parameter of the Gorakhpur Region. The proposed system will work on the client-server architecture model using IoT. The system is organized in Two-tier Architecture. Our proposed system contains a various sensor which will monitor the temperature of the region, humidity, Rain value and pressure of the system. [2]

IoT-Based Data Logger for Weather Monitoring Using Arduino-Based Wireless Sensor Networks with Remote Graphical Application and Alerts (Jamal Mabrouki , Mourade Azrour, Driss Dhiba, Yousef Farhaoui, and Souad El Hajjaji)

In recent years, the monitoring systems play significant roles in our life. So, in this paper, we propose an automatic weather monitoring system that allows having dynamic and real-time climate data of a given area. The proposed system is based on the internet of things technology and embedded system. The system also includes electronic devices, sensors, and wireless technology. The main objective of this system is sensing the climate parameters, such as temperature, humidity, and existence of some gases, based on the sensors. The captured values can then be sent to remote applications or databases. Afterwards, the stored data can be visualized in graphics and tables form.[3]

Internet of Things (IOT) based Weather Monitoring System(Girija C, Harshalatha H, Andreanna Grace Shires, Pushpalatha H P)

By keeping the embedded devices in the environment for monitoring enables self protection (i.e., smart environment) to the environment. To implement this need to deploy the sensor devices in the environment for collecting the data and analysis. By deploying sensor devices in the environment, we can bring the environment into real life i.e. it can interact with other objects through the network. Then the collected data and analysis results will be available to the end user through the Wi-Fi. [4]

2.2 SUMMARY OF EXISTING SYSTEM

Existing system

- Existing system model is presented in uses ZigBee based wireless sensor networks to monitor physical and environmental conditions with thousands of application in different fields. The demerits are Short distance Communication.

DEMIRTS

- Short distance communication, Lot of hardware requirements

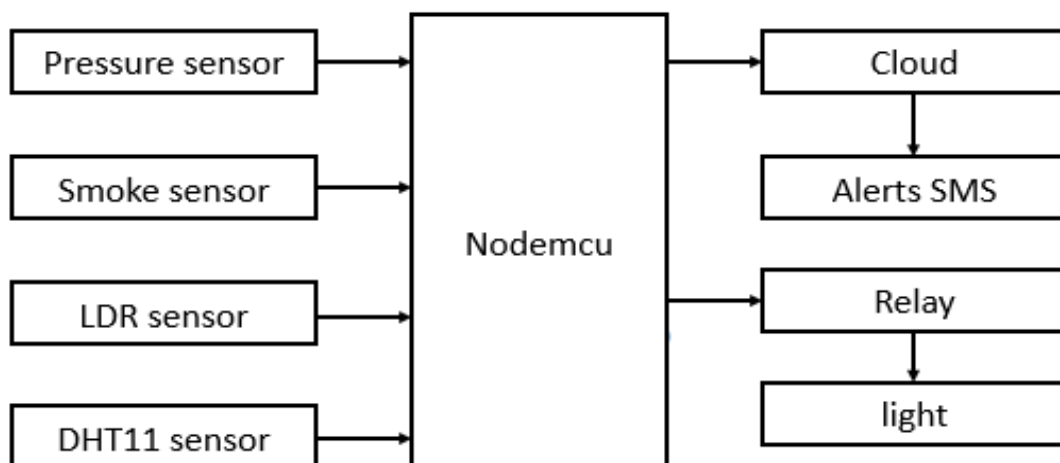
CHAPTER 3

PROPOSED METHODOLOGY

3.1 INTRODUCTION TO PROPOSED SYSTEM

The implemented system consists of a Nodemcu microcontroller as a main processing unit for the entire system and all the sensor and devices can be connected with the microcontroller. The sensors can be operated by the microcontroller to retrieve the data from them and it processes the analysis with the sensor data and updates it to the internet through Wi-Fi module connected to it. The pressure sensor used to monitor the pressure level sensor. The temperature sensor used to monitor temperature level. Humidity level sensor is used to monitor the air moisture level. The smoke sensor is used to measure the poisons gases and rain fall sensor is detects the rain fall. LDR is used to identify the sun light, when the sun light below the threshold level then turn on the light. If any critical condition automatically send the alert message.

3.2 BLOCK DIAGRAM OF PROPOSED SYSTEM



3.3 SYSTEM SPECIFICATIONS

Node MCU

NodeMCU is an open source IoT platform.^{[4][5]} It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.^{[6][7]} The term "NodeMCU" by default refers to the firmware rather than the development kits. The firmware uses the Lua scripting language. It is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266.

NodeMCU was created shortly after the ESP8266 came out. On December 30, 2013, Espressif Systems^[6] began production of the ESP8266.^[10] The ESP8266 is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core,^[citation needed] widely used in IoT applications (see related projects). NodeMCU started on 13 Oct 2014, when Hong committed the first file of nodemcu-firmware to GitHub.^[11] Two months later, the project expanded to include an open-hardware platform when developer Huang R committed the gerber file of an ESP8266 board, named devkit v0.9. Later that month, Tuan PM ported MQTT client library from Contiki to the ESP8266 SoC platform,^[13] and committed to NodeMCU project, then NodeMCU was able to support the MQTT IoT protocol, using Lua to access the MQTT broker. Another important update was made on 30 Jan 2015, when Devsaurus ported the u8glib^[14] to NodeMCU project,^[15] enabling NodeMCU to easily drive LCD, Screen, OLED, even VGA displays.

In summer 2015 the creators abandoned the firmware project and a group of independent contributors took over. By summer 2016 the NodeMCU included more than 40 different modules. Due to resource constraints users need to select the modules relevant for their project and build a firmware tailored to their needs.

Arduino.cc began developing new MCU boards based on non-AVR processors like the ARM/SAM MCU and used in the Arduino Due, they needed to modify the Arduino IDE so that it would be relatively easy to change the IDE to support alternate tool chains to allow Arduino C/C++ to be compiled for these new

processors. They did this with the introduction of the Board Manager and the SAM Core. A "core" is the collection of software components required by the Board Manager and the Arduino IDE to compile an Arduino C/C++ source file for the target MCU's machine language. Some ESP8266 enthusiasts developed an Arduino core for the ESP8266 WiFi SoC, popularly called the "ESP8266 Core for the Arduino IDE".^[16] This has become a leading software development platform for the various ESP8266-based modules and development boards, including NodeMCUs.

IoT platform node MCU is open source. Language used in it is lua scripting language. It uses many open source projects, such as lua-cjson, and spiiffs. It includes firmware which runs on the ESP8266 Wi-Fi SoC, and hardware which is based on the ESP-12 module. NodeMCU was created shortly after the ESP8266 came out. In December 30, 2013, Espressif systems began production of the ESP8266. The ESP8266 is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IoT applications. NodeMCU started in 13 Oct 2014, when Hong committed the first file of NodeMCU - firmware to GitHub. Two months later, the project expanded to include an open-hardware platform when developer Huang R committed the gerber file of an ESP8266 board, named devkit 1.0. Later that month, Tuan PM ported MQTT client library from Contiki to the ESP8266 SoC platform, and committed to Node MCU project, then Node MCU was able to support the MQTT IoT protocol, using Lua to access the MQTT IoT protocol, using Lua to access the MQTT broker. Another important update was made on 30 Jan 2015, when Devsaurus ported the u8glib to NodeMCU project, enabling NodeMCU to easily drive LCD, Screen, OLED, even VGA displays.

The main objective of this project is to develop a home automation system using an Node MCU board with Internet being remotely controlled by any Android OS smart phone. As technology is advancing so houses are also getting smarter. Modern houses are gradually shifting from conventional switches to centralized control system, involving remote controlled switches. Presently, conventional wall switches located in different parts of the house makes it difficult for the user to go near them to operate. Even more it becomes more difficult for the elderly or physically handicapped people to do so. Remote controlled home automation system provides a most modern solution with smart phones.

In order to achieve this, a relay module is interfaced to the Node MCU board at the receiver end while on the transmitter end, a GUI application on the cell phone sends ON/OFF commands to the receiver where loads are connected. By touching the specified location on the GUI, the loads can be turned ON/OFF remotely through this technology. The loads are operated by IOT board through Relay Module.

NODEMCU (esp8266) has been selected as the controller for this system due to its compact size, compatibility, easy interfacing over several other type of controller including Programmable Integrated Circuit (PIC), Programmable Logic Controller (PLC) and others. ESP8266 is an open source firmware that is built on top of the chip manufacturer's proprietary SDK. The firmware provides a simple programming environment, which is a very simple and fast scripting language. The ESP8266 chip incorporates on a standard circuit board. The board has a built-in USB port that is already wired up with the chip, a hardware reset button, Wi-Fi antenna, LED lights, and standard-sized GPIO (General Purpose Input Output) pins that can plug into a bread board. Figure 3.1 shows the diagram of NODEMCU (ESP8266). It has Processor called L106 32bit RISC microprocessor core based on the Tensilica Xtensa Diamond Standard 106Micro running at 80 MHz and has a 6 memory of 32 Kbit instruction RAM, 32 Kbit instruction cache RAM, 80 Kbit user data RAM & 16 Kbit ETS system data RAM. It has inbuilt Wi-Fi module of IEEE 802.11 b/g/n Wi-Fi.

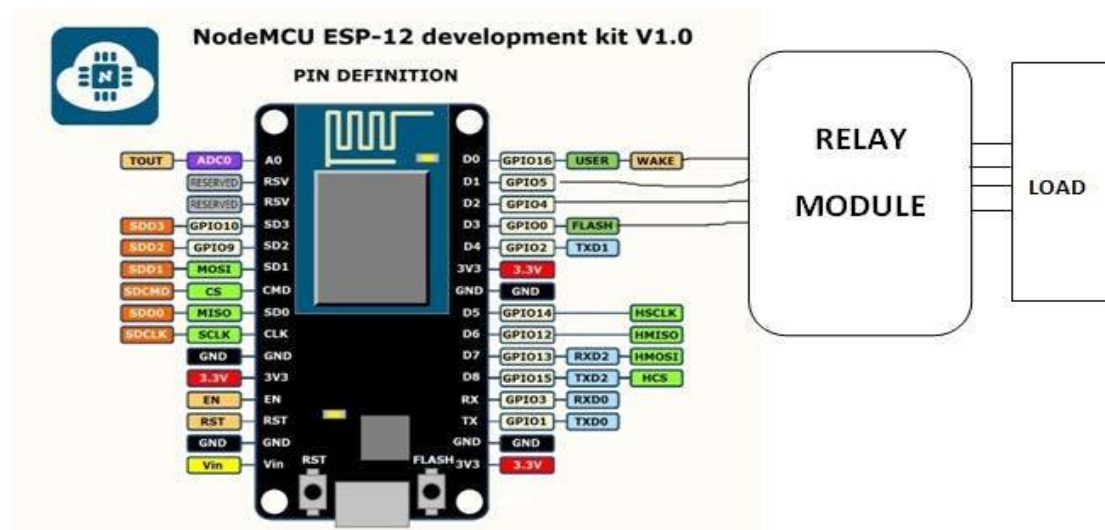


Fig. 3.1: NODEMCU (ESP8266)

3.3.1 Power Supply Unit

3.3.1.1 Circuit Diagram and it's Description

The operation of power supply circuits built using filters, rectifiers, and then voltage regulators. Starting with an AC voltage, a steady DC voltage is obtained by rectifying the AC voltage, Then filtering to a DC level, and finally, regulating to obtain a desired fixed DC voltage. The regulation is usually obtained from an IC voltage regulator Unit, which takes a DC voltage and provides a somewhat lower DC voltage, Which remains the same even if the input DC voltage varies, or the output Load connected to the DC voltage changes.

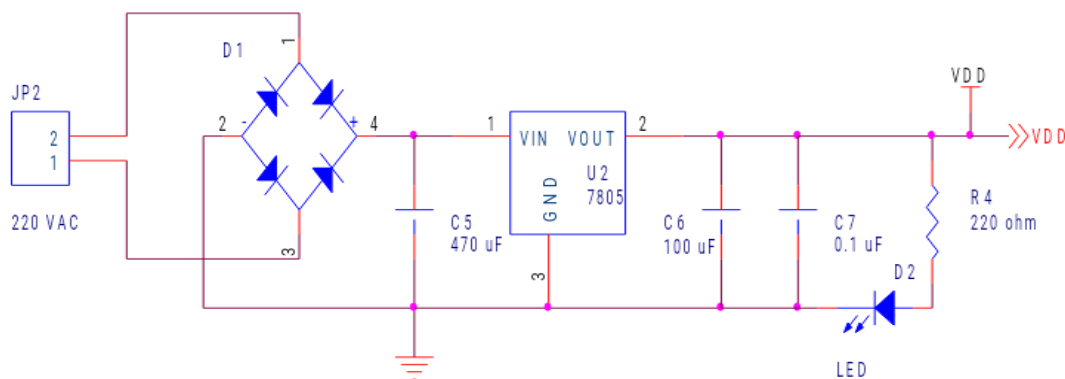


Fig. 3.2: Power Supply Unit Circuit Diagram

3.3.2 Working Principle

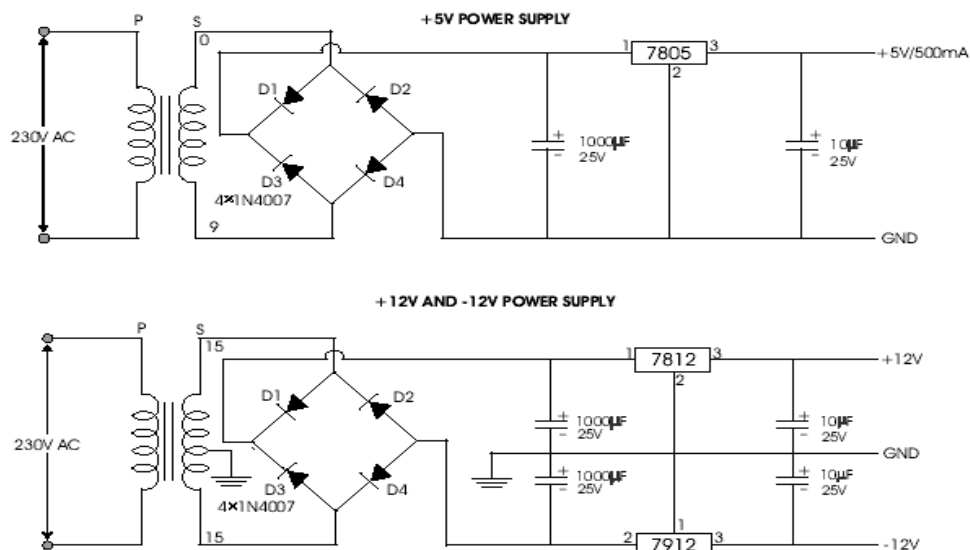


Fig. 3.3: Working Principle

The AC voltage, typically 220V rms, is connected to a transformer, which steps that ac voltage down to the level of the desired DC output. A diode rectifier then provides a full-wave rectified voltage that is initially filtered by a simple capacitor filter to produce a dc voltage. This resulting dc voltage usually has some ripple or ac voltage variation.

A regulator circuit removes the ripples and also remains the same dc value even if the input dc voltage varies, or the load connected to the output dc voltage.

3.3.3 *Sensors*

3.3.3.1 *Smoke Sensor (Gas Sensor)*

In current technology scenario, monitoring of gases produced is very important. From home appliances such as air conditioners to electric chimneys and safety systems at industries monitoring of gases is very crucial. Gas sensors are very important part of such systems. Small like a nose, gas sensors spontaneously react to the gas present, thus keeping the system updated about any alterations that occur in the concentration of molecules at gaseous state.

Gas sensors are available in wide specifications depending on the sensitivity levels, type of gas to be sensed, physical dimensions and numerous other factors. This Insight covers a methane gas sensor that can sense gases such as ammonia which might get produced from methane. When a gas interacts with this sensor, it is first ionized into its constituents and is then adsorbed by the sensing element. This adsorption creates a potential difference on the element which is conveyed to the processor unit through output pins in form of current. What is this sensing element? Is it kept in some chamber or is kept exposed? How does it get current and how it is taken out? Let's find out in this Insight!!!

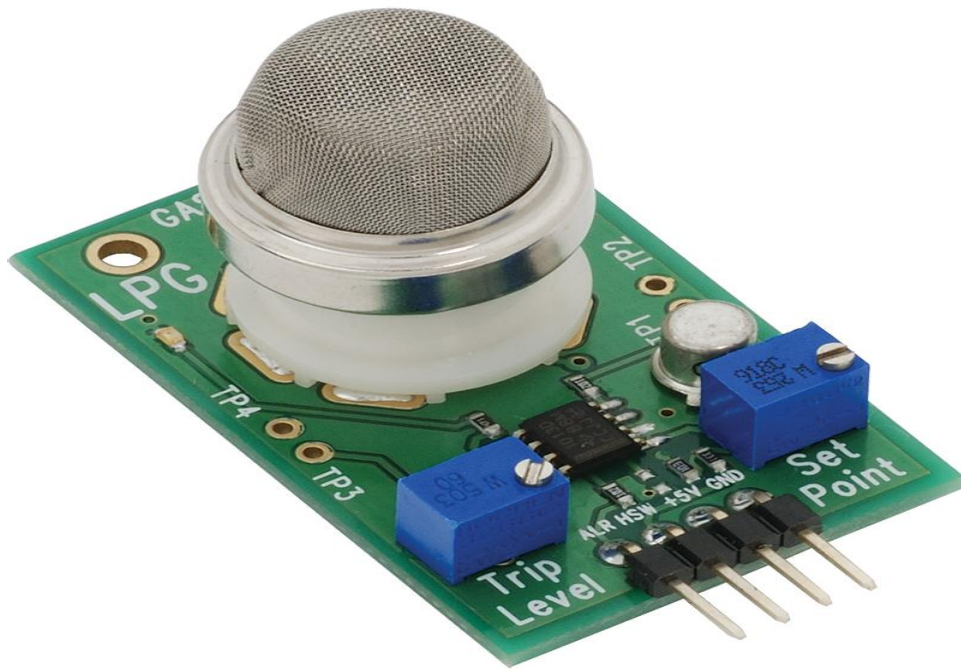


Fig. 3.4: Smoke Sensor

The gas sensor module consists of a steel exoskeleton under which a sensing element is housed. This sensing element is subjected to current through connecting leads. This current is known as heating current through it, the gases coming close to the sensing element get ionized and are absorbed by the sensing element. This changes the resistance of the sensing element which alters the value of the current going out of it.

Image shows externals of a standard gas sensor module: a steel mesh, copper clamping ring and connecting leads. The top part is a stainless steel mesh which takes care of the following:

Filtering out the suspended particles so that only gaseous elements are able to pass to insides of the sensor.

- * Protecting the insides of the sensor.
- * Exhibits an anti-explosion network that keeps the sensor module intact at high temperatures and gas pressures.

In order to manage above listed functions efficiently, the steel mesh is made into two layers. The mesh is bound to rest of the body via a copper plated clamping ring.

The connecting leads of the sensor are thick so that sensor can be connected firmly to the circuit and sufficient amount of heat gets conducted to the inside part. They are casted from copper and have tin plating over them. Four of the six leads (A, B, C, D) are for signal fetching while two (1, 2) are used to provide sufficient heat to the sensing element.

The pins are placed on a Bakelite base which is a good insulator and provides firm gripping to the connecting leads of the sensor.

The top of the gas sensor is removed off to see the internals parts of the sensor: sensing element and connection wiring. The hexapod structure is constituted by the sensing element and six connecting legs that extend beyond the Bakelite base.

Using a ceramic substrate increases the heating efficiency and tin oxide, being sensitive towards adsorbing desired gas' components (in this case methane and its products) suffices as sensing coating.

The leads responsible for heating the sensing element are connected through Nickel-Chromium, well known conductive alloy. Leads responsible for output signals are connected using platinum wires which convey small changes in the current that passes through the sensing element. The platinum wires are connected to the body of the sensing element while Nickel-Chromium wires pass through its hollow structure.

While other wires are attached to the outer body of the element, Nickel-Chromium wires are placed inside the element in a spring shaped. Image shows coiled part of the wire which is placed on the inside of the hollow ceramic

Image shows the ceramic with tin dioxide on the top coating that has good adsorbing property. Any gas to be monitored has specific temperature at which it

ionizes. The task of the sensor is to work at the desired temperature so that gas molecules get ionized. Through Nickel-chromium wire, the ceramic region of the sensing element is subjected to heating current. The heat is radiated by the element in the nearby region where gases interact with it and get ionized. Once, ionized, they are absorbed by the tin dioxide. Adsorbed molecules change the resistance of the tin dioxide layer. This changes the current flowing through the sensing element and is conveyed through the output leads to the unit that controls the working of the gas sensor.

3.3.4 LDR – Light Sensor

3.3.4.1 Dependent resistor circuit and working principle

The controlling of lights and home appliances is generally operated and maintained manually on several occasions. But the process of appliances controlling may cause wastage of power due to the carelessness of human beings or unusual circumstances. To overcome this problem we can use the light-dependent resistor circuit for controlling the loads based on the intensity of light. An LDR or a photo resistor is a device that is made up of high resistance semiconductor material. This article gives an overview of what is LDR, light-dependent resistor circuit and its working.

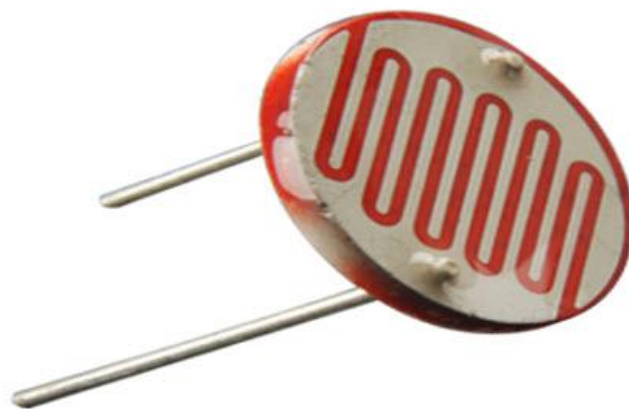


Fig. 3.5: Light Dependent Resistor

3.3.4.2 Construction of an LDR

The construction of an LDR includes a light-sensitive material that is placed on an insulating substrate like as ceramic. The material is placed in a zigzag shape in order to get the required power rating and resistance. The area of zigzag separates the metal placed areas into two regions.

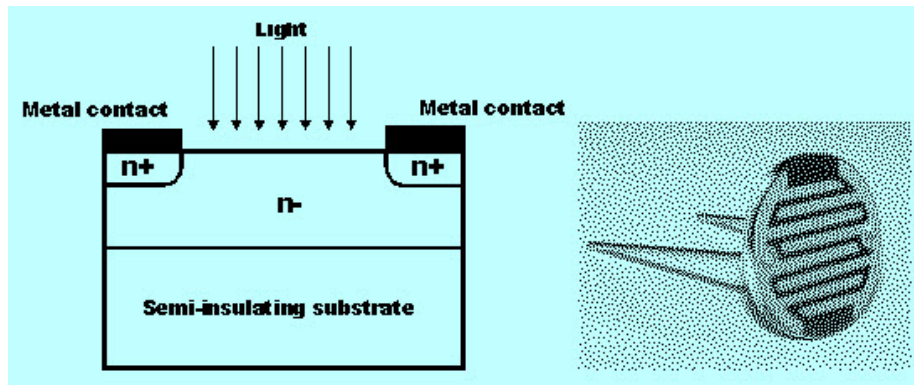


Fig. 3.6: Construction of an LDR

Where the Ohmic contacts are made either on the sides of the area. The resistances of the contacts must be as less as possible to make sure that the resistance, mainly varies due to the light effect only. The use of lead & cadmium materials is avoided as they are injurious to the environment.

3.3.5 Working Principle of Light Dependent Resistor

The working principle of an **LDR** is photo conductivity that is nothing but an optical phenomenon. When the light is absorbed by the material then the conductivity of the material reduces. When the light falls on the LDR, then the electrons in the valence band of the material are eager to the conduction band. But, the photons in the incident light must have energy superior than the band gap of the material to make the electrons jump from one band to another band.

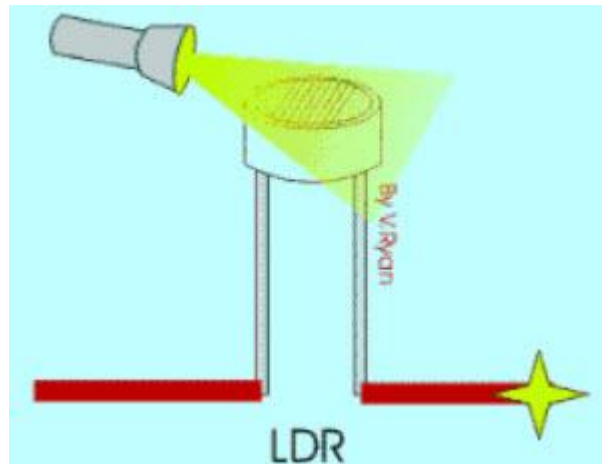


Fig. 3.7: Light Dependent Resistor

Hence, when light having ample energy, more electrons are excited to the conduction band which grades in a large number of charge carriers. When the effect of this process and the flow of the current starts flowing more, the resistance of the device decreases.

3.3.6 Light Dependent Resistor Circuit

The circuit of LDR is an electronic circuit built with LDR, relay, Darlington pair, diode, & resistors shown in the below circuit diagram. A voltage supply is given to the load.

The required DC voltage of the LDR circuit is supplied from a bridge rectifier circuit or a battery. This circuit changes the AC supply into a DC. The bridge rectifier circuit uses a step-down transformer to step-down the voltage from 230v into 12v. The diodes are connected in the form of a bridge used to alter the AC voltage into DC. The voltage regulator is used to change the 12v DC-6v DC, and then, this DC voltage is supplied to the entire circuit. A 230v AC supply for both the bridge rectifier and the load is to be kept continuously for continuous operation of the light sensor circuit.

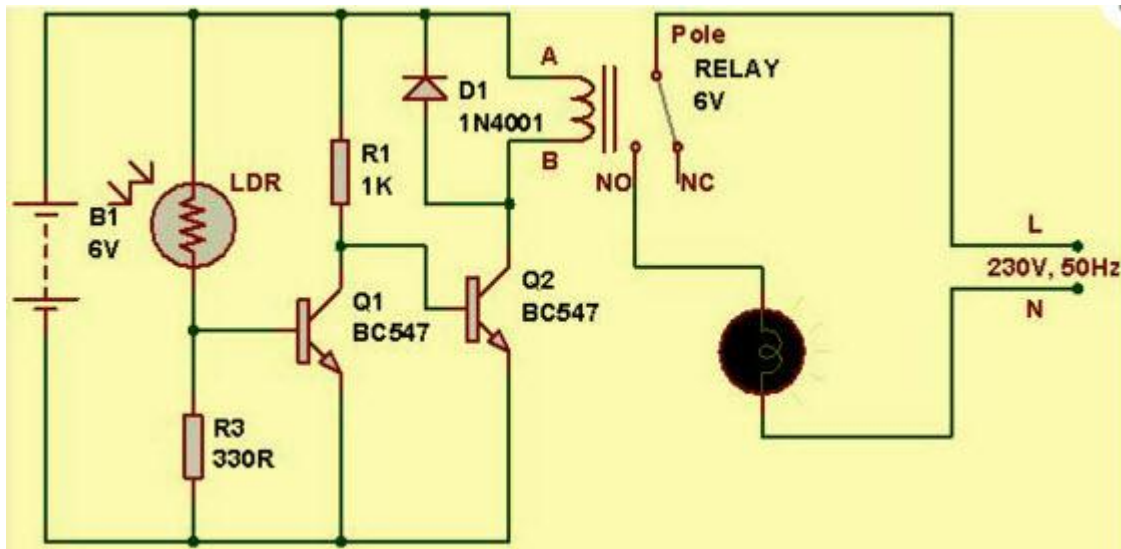


Fig. 3.8: Light Dependent Resistor Circuit Diagram

In the morning time, this sensor has a low resistance around 100Ω . Thus, the power supply flows through the LDR & ground through the variable resistor and resistor as shown in the above light sensor circuit. This is due to the resistance offered by the light-dependent resistor in the daytime or when the light falls on the LDR, then it is less compared to the resistance of the remaining part of the sensor circuit. We are alert of the principle of current, that the flow of current always flows in the path of low resistance.

Therefore, the relay coil does not get sufficient supplies to get strengthened. Hence, the light is switched off in the daylight. In the same way, during the night time, the resistance of the LDR increases to a high value ($20M\Omega$). Thus, due to the high resistance of the resistor, the flow of current is low or almost zero. Now, the flow of current through the low-resistance lane such that it increases the base voltage of the Darlington pair to reach more than 1.4v.

3.3.7 Applications of LDR

Light-dependent resistors are simple and low-cost devices. These devices are used where there is a need to sense the presence and absence of light is necessary. These resistors are used as light sensors and the applications of LDR mainly include alarm clocks, street lights, light intensity meters, burglar alarm

circuits. For a better understanding of this concept, here we have explained one project namely; power conserving of intensity controlled street lights using LDR.

3.3.8 Power Conserving of Intensity Controlled Street Lights using LDR

Nowadays, lightening-up of highways is done through HID lamps but, the energy consumption of these lights is high as well as there is no particular mechanism to turn on/off the lights from sunrise to sunset. To overcome this problem, here is an alternative method using LEDs i.e. power conserving of intensity controlled street lights using LDR.

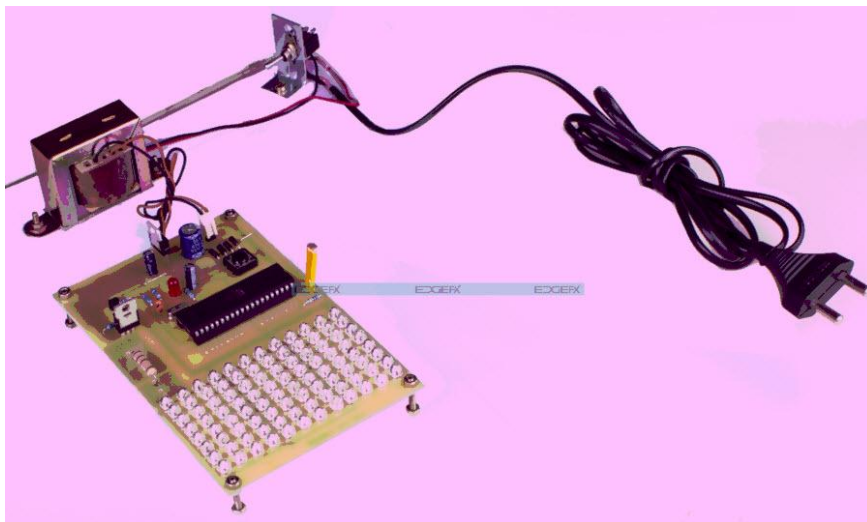


Fig. 3.9: Intensity Controlled Street Light using LDR

The proposed system is built to overcome the drawback of the present day of HID lamps. The proposed system determines the usage of light-emitting diodes as a light source and its adjustable intensity control according to the requirement. These lights consume less power, as well as the life span of these lights, which is more compared to conventional HID lamps.

The most important feature of this project is, the light intensity can be controlled according to the necessity during night time, which is not possible in HID lamps. An LDR is used to detect the light and the resistance of the LDR drastically reduces according to the light in the day time, which forms as an i/p signal to the controller. A bunch of LEDs is used to make a street light and the microcontroller

using in the project is preprogrammed with instructions that control the light intensity based on the Pulse width modulation signals generated. The light intensity is kept high during the night time, and as the traffic on the roads tends to fall in peak hours; the intensity also falls gradually till morning. Finally, the LED lights get turned off in the morning at 6 am and continues again at 6 pm in the evening. This process will continue.

Furthermore, this project can be developed by mixing it with a solar panel, which changes the intensity of the solar to the equivalent voltage and solar energy is used to feed the lights on highways.

Thus, this is all about the working principle of the light-dependent resistor and LDR circuit diagram and its applications.

3.3.9 Humidity Sensor

Measurement and control of temperature and relative humidity finds applications in numerous areas. These days devices are available which have both temperature and humidity sensors with signal conditioning, ADC, calibration and communication interface all built inside them. The use of such smart sensors greatly simplify the design and reduces the overall cost. We discussed in past about Humidity and temperature measurements with Sensirion's SHT1x/SHT7x sensors. These sensors are capable of measuring both temperature and relative humidity and provide fully calibrated digital outputs. While SHT1x/SHT7x are very accurate sensors, they are still expensive for hobbyists use. This articles discusses the DHT11 sensor which also provides calibrated digital outputs for temperature and humidity but is relatively lot cheaper than the Sensirion sensors. The DHT11 sensor uses a proprietary 1-wire protocol which we will be exploring here and implementing with the PIC16F628A microcontroller that will receive the temperature and humidity values from the sensor and display them on a 16×2 character LCD.

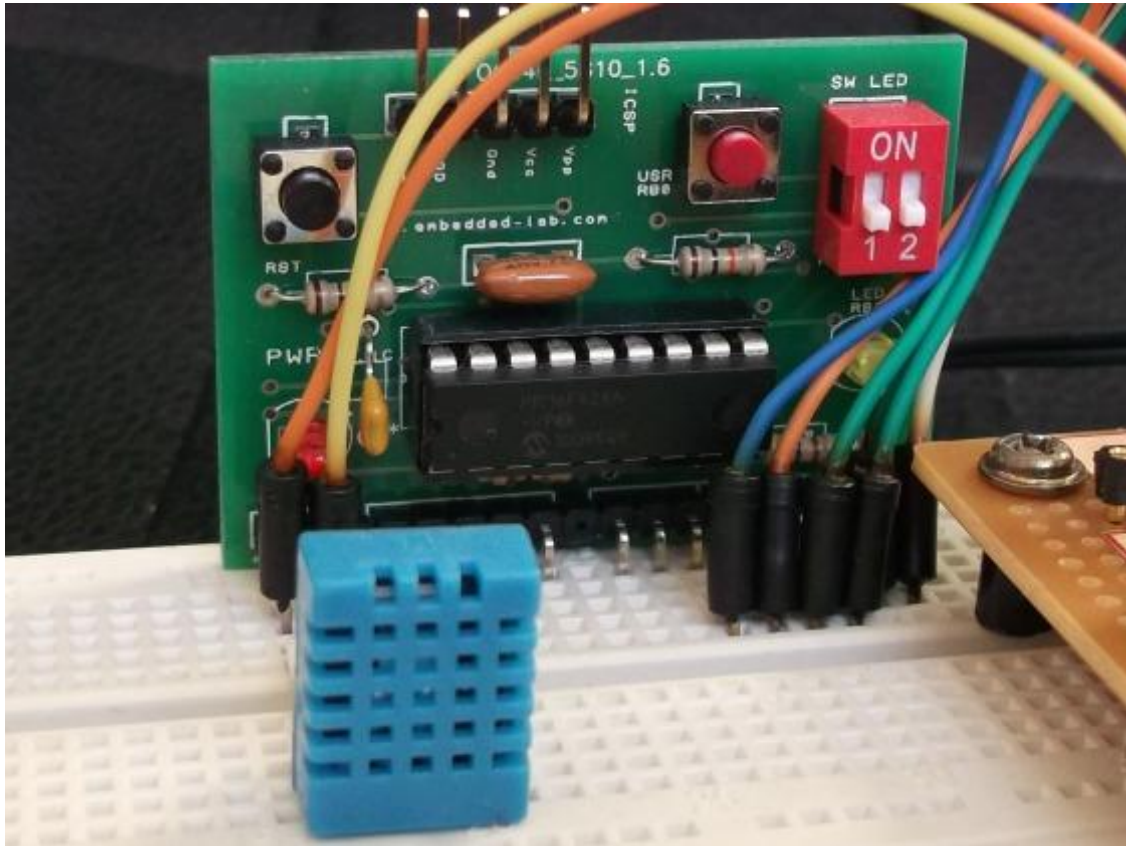


Fig. 3.10: Interfacing DHT11 sensor with PIC16F628A

About DHT11 sensor

The DHT11 sensor comes in a single row 4-pin package and operates from 3.5 to 5.5V power supply. It can measure temperature from 0-50 °C with an accuracy of $\pm 2^{\circ}\text{C}$ and relative humidity ranging from 20-95% with an accuracy of $\pm 5\%$. The sensor provides fully calibrated digital outputs for the two measurements. It has got its own proprietary 1-wire protocol, and therefore, the communication between the sensor and a microcontroller is not possible through a direct interface with any of its peripherals. The protocol must be implemented in the firmware of the MCU with precise timing required by the sensor.

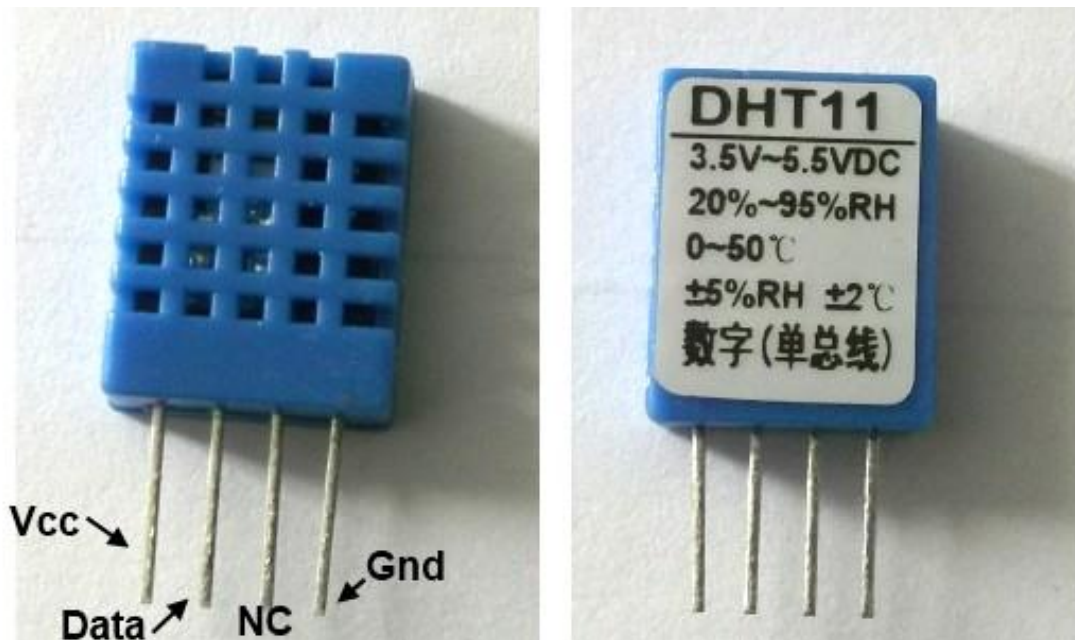


Fig. 3.11: DHT11 sensor comes in a single row 4-pin package

The following timing diagrams describe the data transfer protocol between a MCU and the DHT11 sensor. The MCU initiates data transmission by issuing a “**Start**” signal. The MCU pin must be configured as output for this purpose. The MCU first pulls the data line low for at least 18 ms and then pulls it high for next 20-40 μ s before it releases it. Next, the sensor responds to the MCU “**Start**” signal by pulling the line low for 80 μ s followed by a logic high signal that also lasts for 80 μ s. Remember that the MCU pin must be configured to input after finishing the “**Start**” signal. Once detecting the response signal from the sensor, the MCU should be ready to receive data from the sensor. The sensor then sends 40 bits (5 bytes) of data continuously in the data line. Note that while transmitting bytes, the sensor sends the most significant bit first.

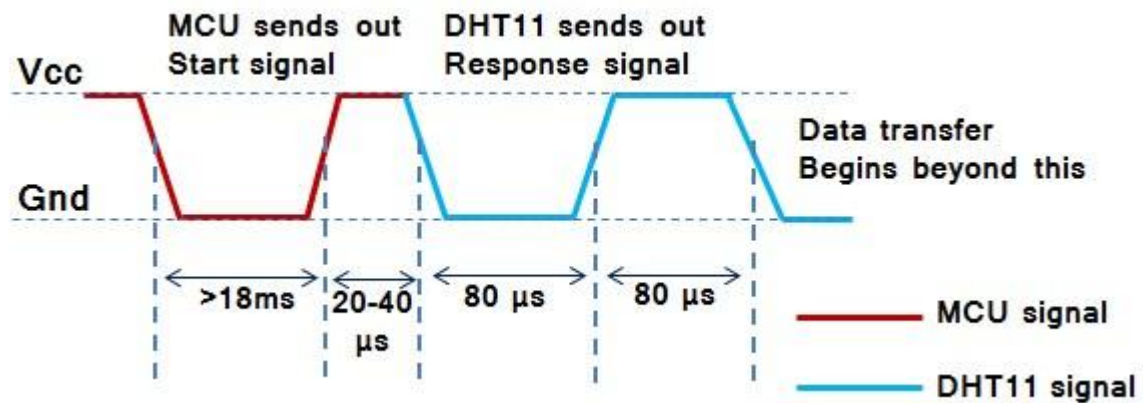


Fig. 3.12: “Start” and “Response” signals

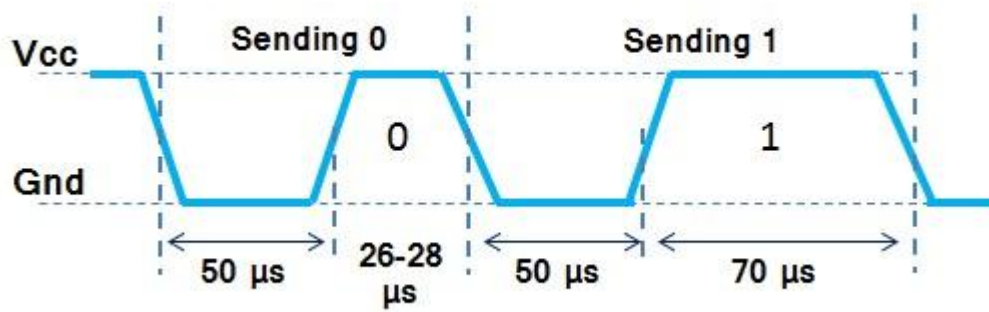
The 40-bit data from the sensor has the following structure.

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

For DHT11 sensor, the decimal bytes of temperature and humidity measurements are always zero. Therefore, the first and third bytes of received data actually give the numeric values of the measured relative humidity (%) and temperature (°C). The last byte is the checksum byte which is used to make sure that the data transfer has happened without any error. If all the five bytes are transferred successfully then the checksum byte must be equal to the last 8 bits of the sum of the first four bytes, i.e.,

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

Now let's talk about the most important thing, which is signaling for transmitting “0” and “1”. In order to send a bit of data, the sensor first pulls the line low for 50 μs. Then it raises the line to high for 26-28 μs if it has to send “0”, or for 70 μs if the bit to be transmitted is “1”. So it is the width of the positive pulse that carries information about 1 and 0.



Timing difference for transmitting “1s” and “0s”

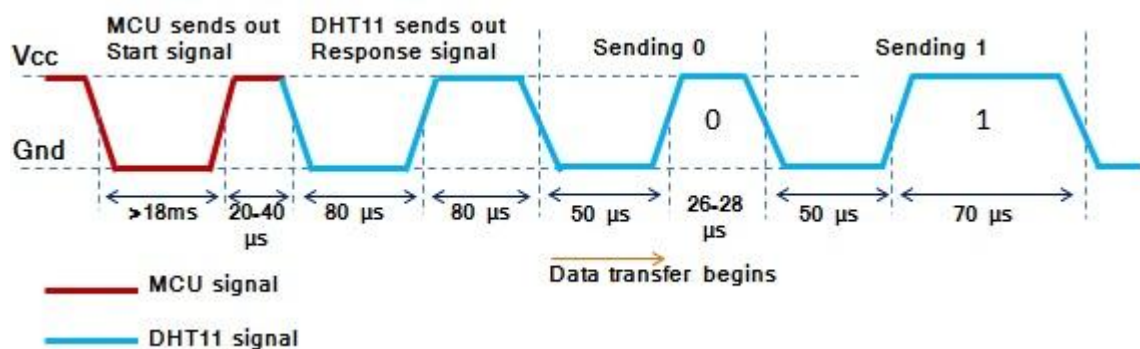


Fig.3.13: Start, Response and Data signals in sequence

At the end of the last transmitted bit, the sensor pulls the data line low for 50 μs and then releases it. The DHT11 sensor requires an external pull-up resistor to be connected between its Vcc and the data line so that under idle condition, the data line is always pulled high. After finishing the data transmission and releasing the data line, the DHT11 sensor goes to the low-power consumption mode until a new “**Start**” signal arrives from the MCU.

3.3.10 Circuit Diagram

Here is the circuit diagram showing the DHT11 sensor and a HD44780-based character LCD interfaced to the PIC16F628A microcontroller. The microcontroller runs at 4.0 MHz clock using an external resonator connected between OSC1 (16) and OSC2 (15) pins. The use of 4.0 MHz clock makes the timing calculation easier as 1 machine cycle becomes 1 μs. The timing information will be

used to calculate the width of the received data pulse from the sensor so that we could identify if it is carrying a 1 or 0.

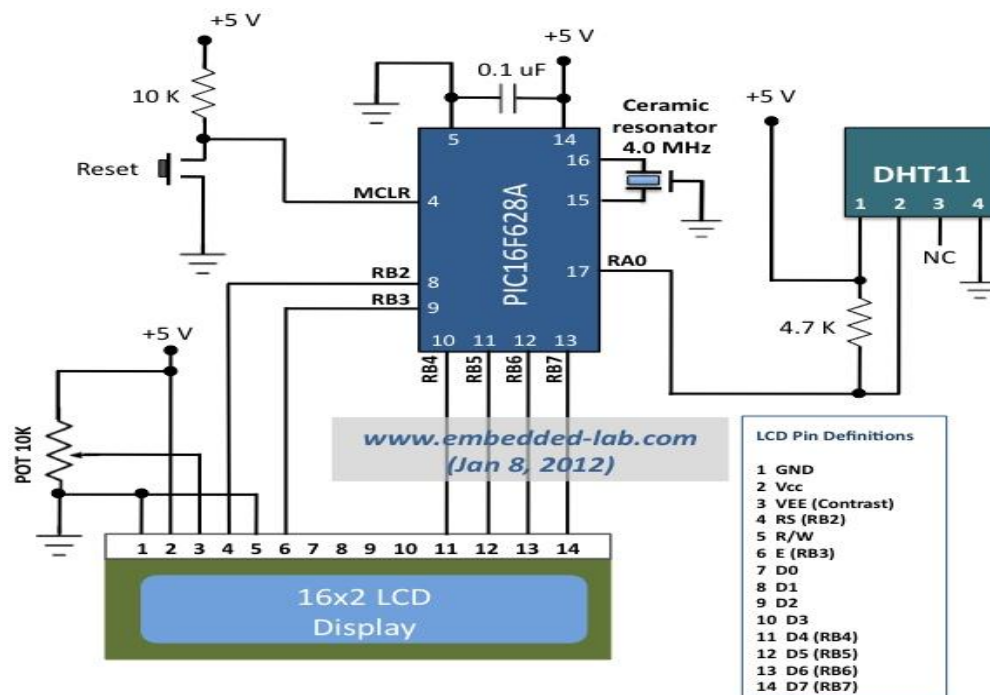


Fig. 3.14: Circuit connections for PIC16F628A and DHT11 sensor

The following pictures show the circuit setup on a breadboard. Don't get confused with the four LEDs and tact switches shown on the perforated board. They have nothing to do with this project. They are there because I am using my **DIY Experimenter's I/O board** for the LCD part of this project. Similarly, I am using my **18-pin PIC16F board** for easy prototyping with the PIC16F628A microcontroller.

3.3.11 Software

Writing a software for DHT11 sensor is little more challenging than the hardware part because of the timing conditions for 1s and 0s. I have written sub-routines in mikroC Pro for PIC for initializing the DHT11 sensor and reading the 40-bit of data in sequence. I have used Timer2 module to keep track of the width of the received data pulse, which is required to identify if the received bit is 1 or 0. When a low-to-high pulse is detected at the beginning of any data bit, TMR2 is cleared and turned ON. Since the clock frequency used here is 4.0 MHz, the TMR2 increments by 1 in every 1 μ s. The TMR2 is stopped whenever the data pulse is low again. The

value of the TMR2 register gives you the the width of the data pulse in μ s. I am using 40 μ s as the threshold for identifying 0 and 1. If the TMR2 is greater than 40, it means the received bit is 1, else it is 0. Here is the complete source code written in mikroC Pro for PIC. It can be easily adapted to any other platform, but remember that if you are using a different clock frequency you should have to modify the timer operation accordingly.

Level Measurement

In industry, liquids such as water, chemicals, and solvents are used in various processes. The amount of such liquid stored can be found by measuring level of the liquid in a container or vessel. The level affects not only the quantity delivered but also pressure and rate of flow in and out of the container. Level sensors detect the level of substances like liquids, slurries, granular materials, and powders. The substance to be measured can be inside a container or can be in its natural form (e.g. a river or a lake). Wide spectrum of sensors is available in the market. Sensor used for measuring humidity is termed as humidity sensor, the one used for measurement of pressure is called pressure sensor, sensor used for measurement of displacement is called position sensor and so on though all of them may be using the similar sensing principle. In a similar fashion, the sensor used for measurement of fluid levels is called a level sensor.

Quite obvious from its name, level sensors are used to measure the level of the free-flowing substances. Such substances include liquids like water, oil, slurries, etc as well as solids in granular/powder form (solids which can flow). These substances tend to get settled in the container tanks due to gravity and maintain their level in rest state. Level sensors measure their level against a pre-set reference.

They are important not only in nuclear plants but in lot many applications. Every car, truck and motorcycle is equipped with a fuel level sensor to measure the amount of gasoline left in the fuel tank. In addition, there are sensors for level measurement of engine oil, brake / power steering fluid, cooling water, windshield cleaning liquid, etc. Industrial applications include liquid level sensing in water treatment tanks, transport and storage tanks, in the petrochemical industry for

liquids such as petrol, etc. Liquid level measurement is important in household applications for devices such as automated coffee machines, water dispensers, juice squeezers, water evaporators, steamers, fridges and freezers, boilers, heating systems, dishwashers, washing machines, steam irons, etc.

In short, level sensors are one of the very important sensors and play very important role in variety of consumer/ industrial applications. As with other type of sensors, level sensors are available or can be designed using variety of sensing principles. Selection of an appropriate type of sensor suiting to the application requirement is very important.

3.3.12 *BMP 180 Sensor (Pressure Sensor)*

A pressure sensor measures pressure, typically of gases or liquids. Pressure is an expression of the force required to stop a fluid from expanding, and is usually stated in terms of force per unit area. A pressure sensor usually acts as a transducer; it generates a signal as a function of the pressure imposed. For the purposes of this article, such a signal is electrical.

Pressure sensors are used for control and monitoring in thousands of everyday applications. Pressure sensors can also be used to indirectly measure other variables such as fluid/gas flow, speed, water level, and altitude. Pressure sensors can alternatively be called pressure transducers, pressure transmitters, pressure senders, pressure indicators, piezometers and manometers, among other names.

Pressure sensors can vary drastically in technology, design, performance, application suitability and cost. A conservative estimate would be that there may be over 50 technologies and at least 300 companies making pressure sensors worldwide. There is also a category of pressure sensors that are designed to measure in a dynamic mode for capturing very high speed changes in pressure. Example applications for this type of sensor would be in the measuring of combustion pressure in an engine cylinder or in a gas turbine. These sensors are commonly manufactured out of piezoelectric materials such as quartz.

Some pressure sensors, such as those found in some traffic enforcement cameras, function in a binary (off/on) manner, i.e., when pressure is applied to a pressure sensor, the sensor acts to complete or break an electrical circuit. These types of sensors are also known as a pressure switch.

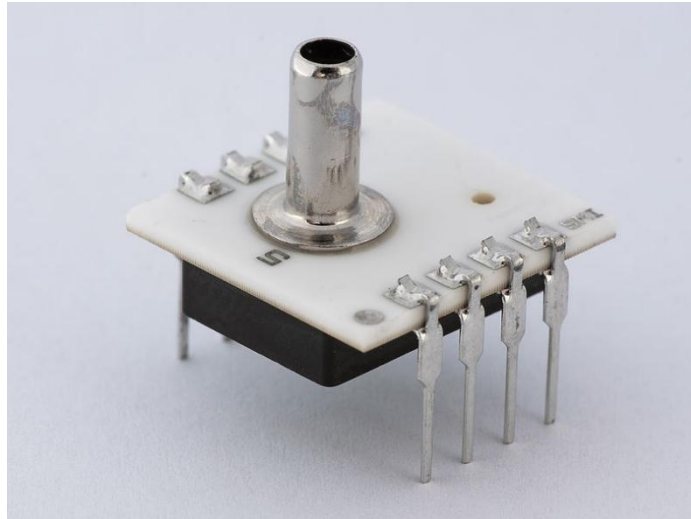


Fig. 3.15: BMP 180 Sensor

3.4 ARDUINO IDE SOFTWARE DESCRIPTION

Introduction

This tutorial will walk you through downloading, installing, and testing the Arduino software (also known as the Arduino IDE – short for Integrated Development Environment). Before you jump to the page for your operating system, make sure you’ve got all the right equipment.



Fig. 3.16: Arduino software

Required Materials

To follow along with this tutorial, you will need the following materials. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.

A computer (Windows, Mac, or Linux)

- An Arduino-compatible microcontroller (anything from this guide should work)
- A USB A-to-B cable, or another appropriate way to connect your Arduino-compatible microcontroller to your computer (check out this USB buying guide if you're not sure which cable to get).

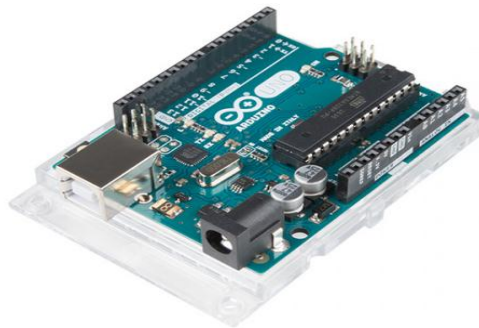


Fig. 3.17: Arduino Uno – R3



Fig. 3.18: USB Cable A to B – 6 Foot

INSTALLER- The Windows version of Arduino is offered in two options: an installer or a zip file. The **installer** is the easier of the two options, just download that, and run the executable file to begin the installation.

Windows install steps. Click the image to get a bigger view.

When you're prompted to install a driver during installation, select "Install". This will install drivers for Arduino specific boards (like the Uno, Nano, etc.) that you may use in the future.

ZIP – If you choose to download the **zip file** version of Arduino, you'll need to extract the files yourself. Don't forget which folder you extract the files into! You will need to run the executable Arduino file in the folder to start the Arduino IDE. When the download is finished, un-zip it and open up the Arduino folder to confirm that yes, there are indeed some files and sub-folders inside. The file structure is important so don't be moving any files around unless you really know what you're doing.

Connecting Your Arduino

Power up your Arduino by connecting your Arduino board to your computer with a USB cable (or FTDI cable if you're using an Arduino Pro). You should see the an LED labeled 'ON' light up (this diagram shows the placement of the power LED on the UNO).

Drivers for Arduino Uno on Windows

To install the drivers for the Arduino Uno, you will need to plug in your board to your computer's USB port. Once the board is connected, you will need to wait for Windows to begin it's driver installation process. After a few moments, the process will probably fail, despite its best efforts.

Open up a search, type in **Device Manager**, and hit **ENTER**.

Look under Ports (COM & LPT) tree. You should see an open port named "**Arduino UNO (COMxx)**". If there is no COM & LPT section, look under "**Other**

devices" for **"Unknown Device**. Right click on the **"Arduino UNO (COMxx)"** or **"Unknown device"** port and choose the **"Update Driver Software"** option.

Next, choose the **"Browse my computer for Driver software"** option

Finally, navigate to the Arduino IDE folder. This should be where you unzipped the Arduino IDE (e.g. it should be similar to the following path with a different version number: *C:\Program Files\arduino-1.8.5\drivers*). Depending on what version of Windows you have, you may be able to select the Uno's driver file, named **"Arduino.inf"**, located in the **"Drivers"** folder (not the **"FTDI USB Drivers"** sub-directory). If you cannot see the ***.inf** file, it is probably just hidden. You can select the **"drivers"** folder with the **"include sub-folders"** option selected instead.

Windows will finish up the driver installation from there! Your computer will enumerate with a COM port. You may see a COM port number depending on what is currently saved in your computer. Try to remember what the number is when uploading. If not, you can always navigate back to the device manager and power cycle the Arduino to determine what number your Arduino enumerated on.

3.5 UBIDOTS EXPLORER

Introduction

Welcome to the Ubidots REST API hardware docs. Here you would reference how to send or retrieve data from your hardware devices using our supported communication protocols: **HTTP**, **MQTT** and **TCP/UDP**.

The purpose of this section is to help you understand what happens in the backstage when communicating with Ubidots, so you can replicate this in your firmware. For this reason, we avoid the use of examples using libraries.

Every time a device updates a sensor value in a variable, a data-point or **"dot"** is created. Ubidots stores dots that come from your devices inside variables, and these stored dots have corresponding **timestamps**:

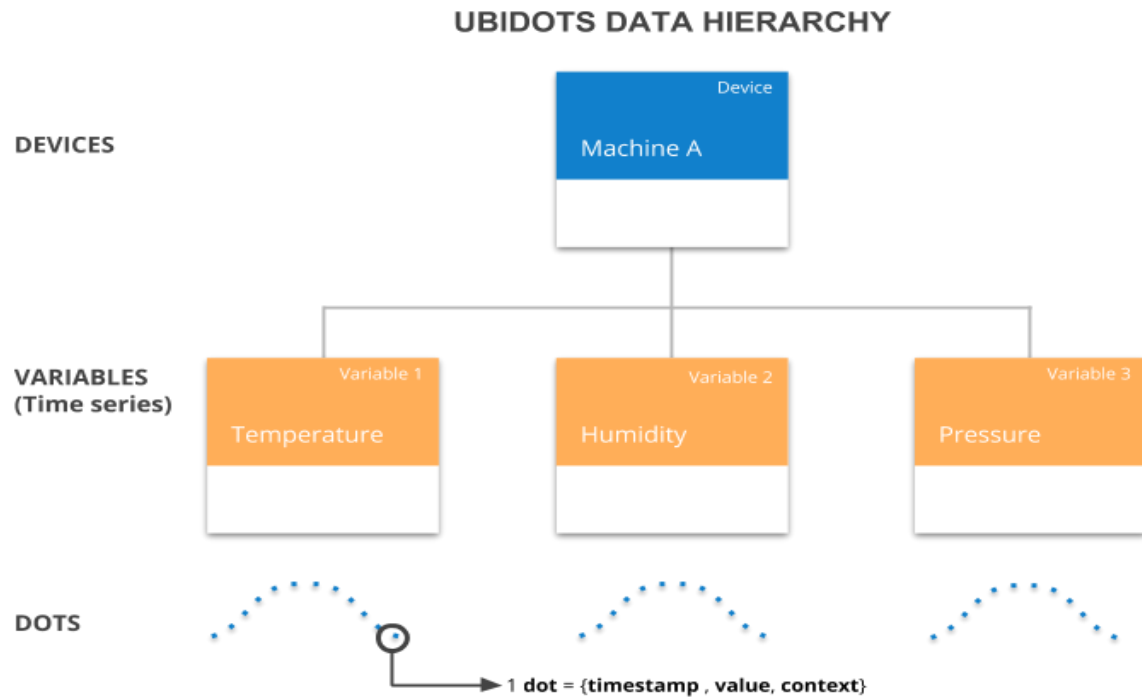


Fig. 3.19: Ubidots Data Hierarchy

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:

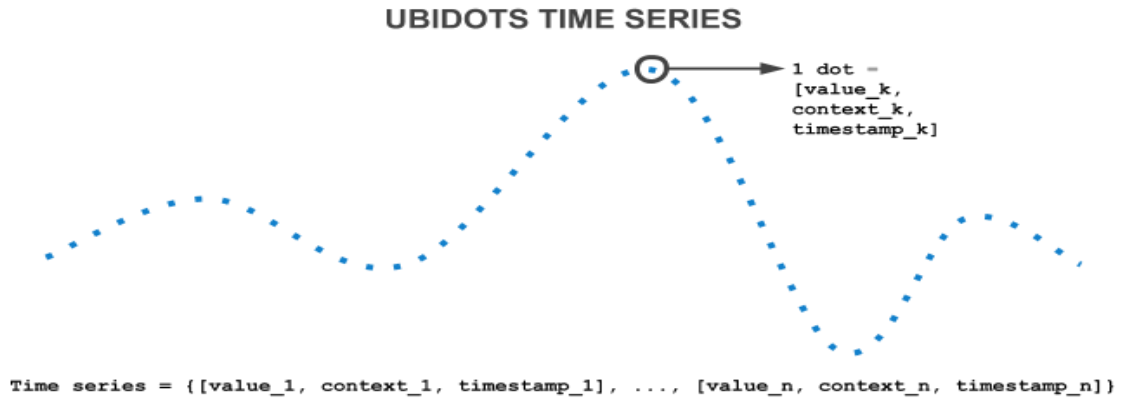


Fig. 3.20: Ubidots Time Series

Ubidots is an agnostic platform, this means that it does not really care what hardware device you are using, as long as you're able to interact with us through at least one of these protocols:

- HTTP
- MQTT
- TCP/UDP

HTTP- HTTP is the main internet protocol for information exchange in the Internet. It also supports data encryption, which is usually called *secure HTTP* or *HTTPs*. When you use a web browser (MS Edge, Firefox, Google Chrome, etc) you are sending and receiving packages using *HTTP* or *HTTPs*. Because of this, it was not surprising to see HTTPs become a popular option for data exchange within IoT.

When a developer is to implement *HTTP* communication, then he should look for a *RESTful Application Programming Interface*, or *REST API*, which exposes all the endpoints required to interact with a third party application (like Ubidots). In this section, you can reference our REST API to understand how to send and retrieve data from our servers.

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

HTTP is a **request/response** protocol, this means that every request that you make is answered by the server. This response includes a number (response code) and a body. For example, when you make a request to retrieve a file on a webpage "(e.g. *"Get me the file 'webside.html'"*)", you build a GET request. If the request is correct, the server will typically return a 200 response code, along with the file requested (body). Under the same logic, when a device sends a GET request to the Ubidots' server (i.e. requesting the last value of a variable), then the server sends back a response with the status of the request(response code), and a body, which would be the a value with its context and timestamp.

An *HTTP* request also needs the parameters below:

- **Host:** Specifies the server you will be making HTTP requests to.
- **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:** Define the operating parameters of the *HTTP* request such as authentication, Content-Type, Content-Length, etc.
- **Body/payload:** In the case of POST and PATCH requests, this is the data sent by your device to the server. GET requests typically do not have a body because they are meant to request data, not to send data.

Ubidots accepts data as *JavaScript Object Notation* or *JSON*. **JSON** is a typical *HTTP* data type, it is a collection of name/value pairs. In various programming languages, this is treated as an object, record, struct, dictionary, hash table, keyed list, or associative array. It is also human readable and language independent. An example of a *JSON* data type that Ubidots accepts can be referenced below:

```
{"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>
```

Where:

- {PATH}: Path to the resource to consume
Example: /api/v1.6/variables/ to get user's variables information.
- {HOST}: Host URL.
Example: industrial.api.ubidots.com
- {USER_AGENT}: An optional string used to identify the type of client, be it by application type, operating system, software vendor or software version of the requesting user agent.
Examples: ESP8266/1.0, Particle/1.2
- {TOKEN}: Unique key that authorizes your device to ingest data inside your Ubidots account.
- {PAYLOAD_LENGTH}: The number of characters of your payload.
Example: The payload {"temperature": 20} will have a content-length of 19.
- {PAYLOAD}: Data to send.
Example: {"temperature": 20}

API URLs

API access can be over *HTTP* or secure HTTP, *HTTPS*, using the following endpoints based on your Ubidots Account type.

Security Note: We strongly advise to use *HTTPS* to make sure your data travels encrypted, avoiding the exposure of your API token and/or sensor data.

Ubidots supports SSL v1.1, TLS v1.2 and v1.3. You can download our root certificates in different formats.

Fixed IPs

Ubidots also supports fixed IPs address for data ingestion, but we highly advise to our users to implement their firmware routines using the dns endpoints instead of fixed IPs, as these last ones may change in the future.

HTTP Authentication

Every request requires a *token*. A *token* is an unique key that authorizes your device to ingest data inside your Ubidots account.

The easiest way to get yours is clicking on "**API Credentials**" under your profile tab:

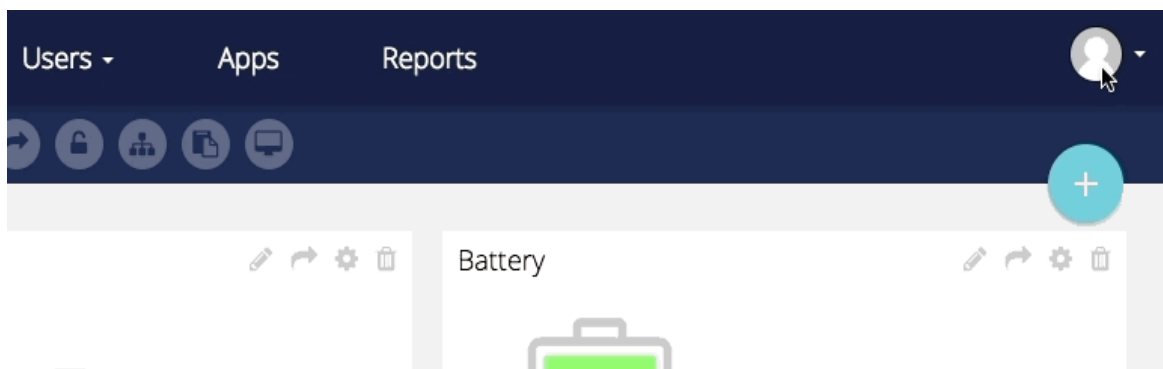


Fig. 3.21: HTTP Authentication

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**

CHAPTER 4

NEURAL NETWORKS FOR RAIN PREDICTION

A Back Propagation Network (BPN) is multilayered neural network architecture that learns from a predefined set of input output example pairs called exemplars. The inputs are propagated from the first layer to the upper layers through the hidden layer until an output is generated. The obtained output is then compared with the desired target. The error is calculated and propagated backwards. For this reason the method is often called the back-propagation learning rule. Based on their contribution towards the original output, each unit in the intermediate layer receives a portion of the total error signal. This process repeats for all layers till each unit receives an error signal relative to the total error. In accordance to the error signal received, connection weights are then updated by each unit. The significance of this process is that as the network trains, the nodes in the intermediate layers organizes themselves such that all the units learn to recognize different features of the total input space. After training, when presented with an arbitrary input pattern that is noisy or incomplete, the units in the hidden layers of the network will respond with an active output if the new input contains a pattern that resembles the feature of the individual units learned to recognize during training. The size of the input layer and the output layer is usually determined by the application. Determining the number of units in the hidden layer is not clearly defined. The size of the hidden layer has to be a compromise between accuracy of the result and the required storage space. Weights should be assigned to small random values and the bias values must also be small (Haykin 1998, Yegnanarayana 2006, Hosseingholizadeh and Abhari 2009). A simple BPN architecture is shown in Figure 4.1.

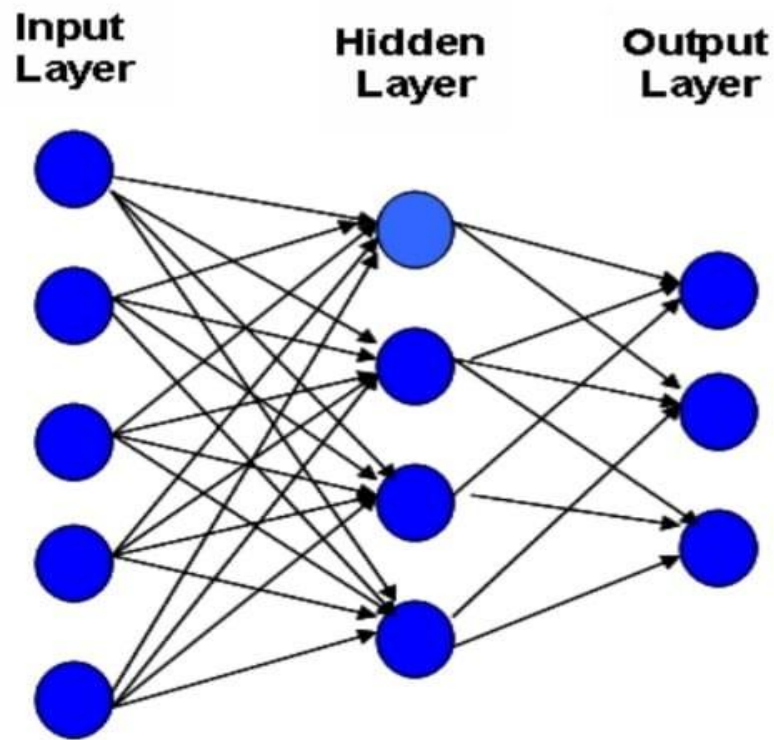


Fig. 4.1: Architecture of a Back Propagation Network

Table 4.1: BPN Parameters for Rain Prediction

Parameters	Description
Number of Layers	4(10,10,5,1)
Activation function (hidden layer)	tansigmoidal
Activation function (output layer)	purelinear
Learning parameter	0.6
Momentum parameter	0.9

4.1 NEURAL NETWORK TRAINING (nntraintool)

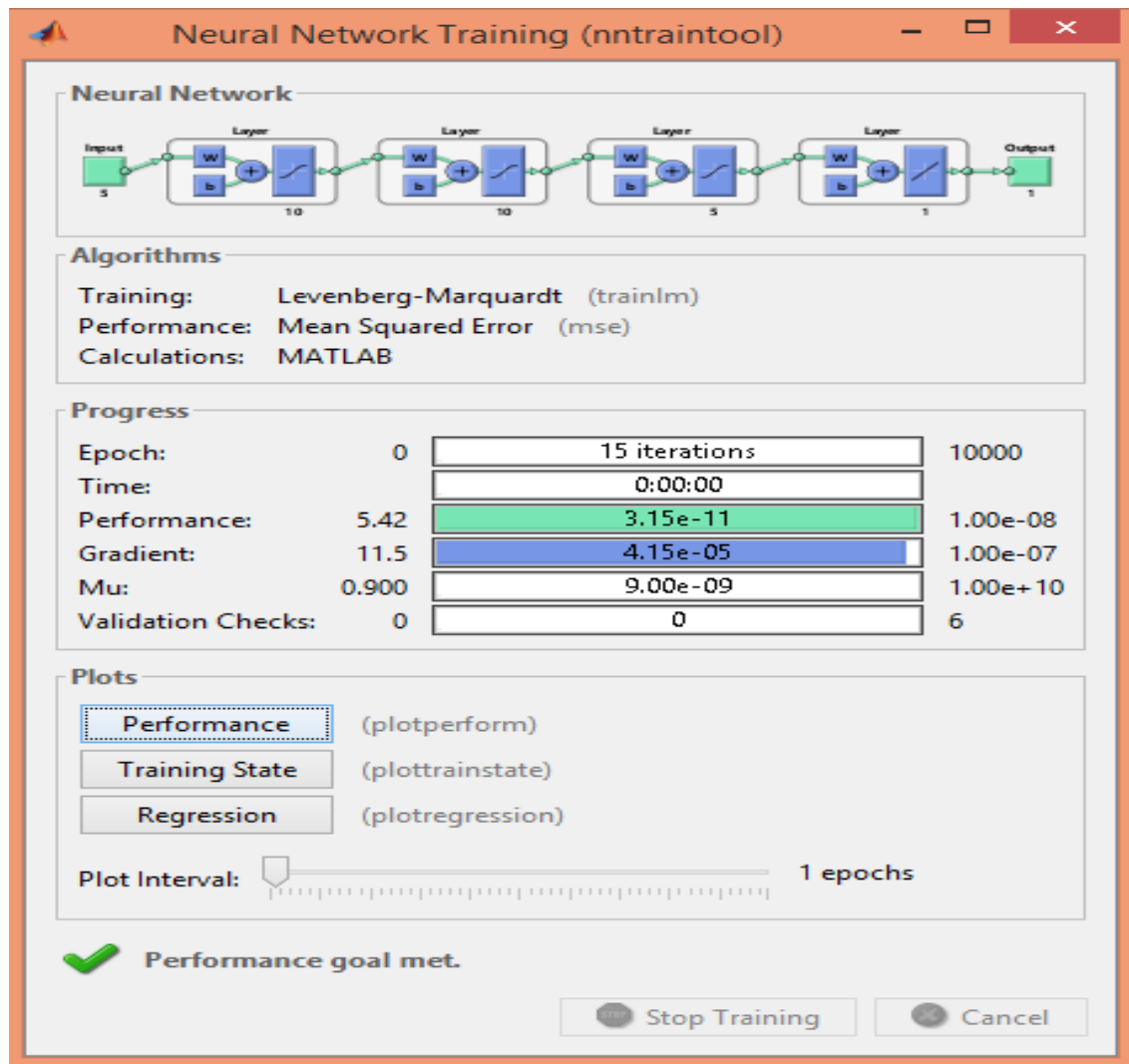


Fig. 4.2: nntrain tool

4.2 COMPARISON OF ACTUAL AND DESIRED OUTPUT

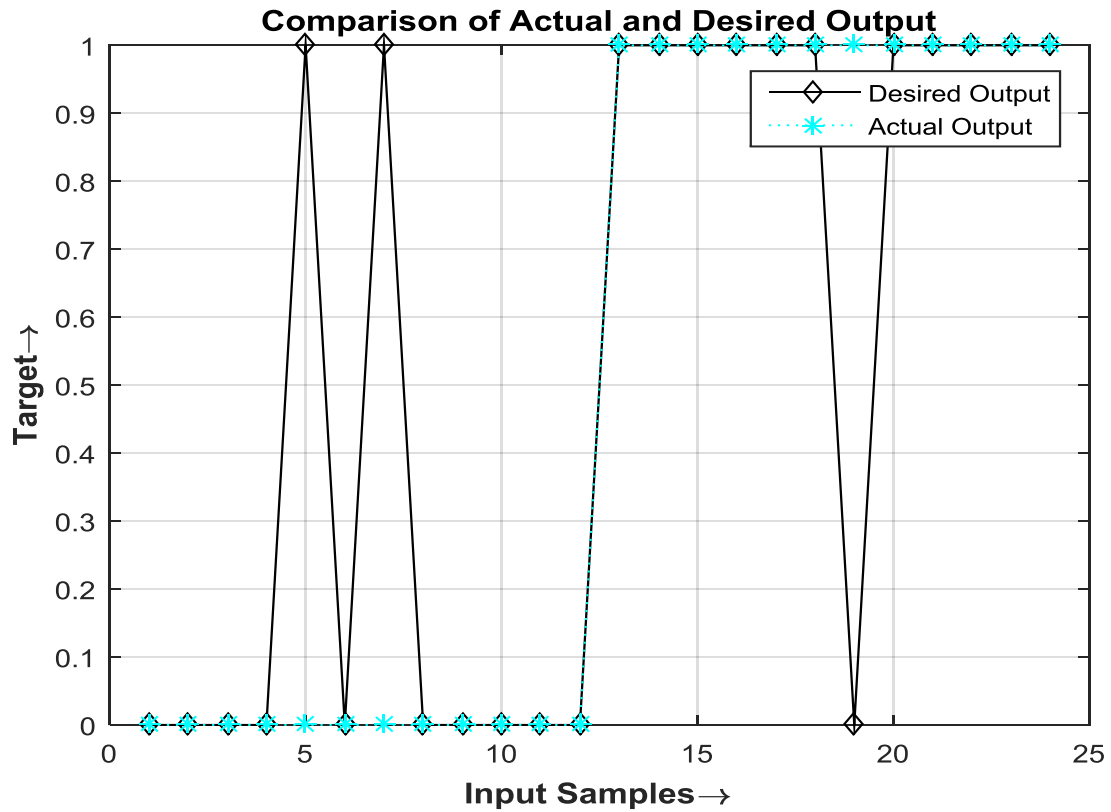


Fig. 4.3: Comparison of actual and desired output

Mean Square Error = 0.1250

4.3 CODE

```
close all;
clear all;
clc;

Temp=[40 36 36 34 33 41 40 36 40 41 44 33 34 25 41 45 45 43 38 38
46 36 34 42 40 24 28 29 29 31 20 31 29 29 29 28 30 46 28 22 21 35
32 30 28 28 28 33 29];

Hum=[43 51 49 46 59 54 50 56 59 66 34 66 58 68 50 49 49 52 57 46
32 63 61 58 56 73 68 65 62 63 45 58 58 63 59 59 59 32 63 75 76 58
59 59 59 54 49 51 56];
```

```

press=[95160 95174 10057 10067 79560 95160 10057 95168 93256
95160 86120 95160 82120 88120 94158 95160 10057 95160 10057
95160 95160 84320 76666 79526 80121 83000 95163 95176 10015
10200 10200 10067 79526 90000 95160 10057 95160 89000 95160
95160 95160 10067 10167 10200 10057 95160 96170 69000 71000];

rain=[0 155 110 98 144 50 12 150 10 135 0 50 90 139 75 79 56 46 112
167 0 110 146 70 56 0 20 150 159 156 200 190 136 122 140 40 35 0 158
200 186 110 156 212 158 79 170 198 212];

light=[0 0 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 1 0 0 0 1
0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 0];

```

```
Temp=Temp./100;
```

```
Hum=Hum./100;
```

```
press=press./10000;
```

```
rain=rain./1000;
```

```
light=light./1;
```

```
p=[Temp(1:2:end);Hum(1:2:end);press(1:2:end);rain(1:2:end);light(1:2:end)];
```

```
x=[zeros(1,25) ones(1,24)];
```

```
t=x(1:2:end);
```

```
% t=t./10;
```

```
% t=t';
```

```
% net=newff(minmax(p),[10,10,5,1],{'tansig','tansig','tansig','purelin'});
```

```
net=newff(minmax(p),[10,10,5,1],{'tansig','tansig','tansig','purelin'});
```

```
net.trainparam.show=5;
```

```
net.trainparam.epochs=10000;
```

```
net.trainparam.lr=0.6;
```

```
net.trainparam.mu=0.9;
```

```
net.trainparam.goal=.00000001;
```

```

% net.trainParam.mu_max = 1e+8;
net=train(net,p,t);
a=sim(net,p);
p1=[Temp(2:2:end);Hum(2:2:end);press(2:2:end);rain(2:2:end);light(2:2:end)];
t1=x(2:2:end);
a1=round(sim(net,p1));
% a1=a1*10;
% t1=t1*10;
% figure,plot(1:length(a),a,'k+.',1:length(a),t,'c*');
figure,plot(1:length(a1),a1,'kd-',1:length(a1),t1,'c*');
grid on
title('\bfComparison of Actual and Desired Output')
    xlabel('\bfInput Samples\rightarrow')
    ylabel('\bfTarget\rightarrow')
    h = legend('Desired Output','Actual Output',2);
sum=0;
for i=1:length(a1)
    sum=sum+(a1(i)-t1(i))^2;
end
mss=sum/length(a1);

```

CHAPTER 5

RESULTS AND CONCLUSION

5.1 RESULTS

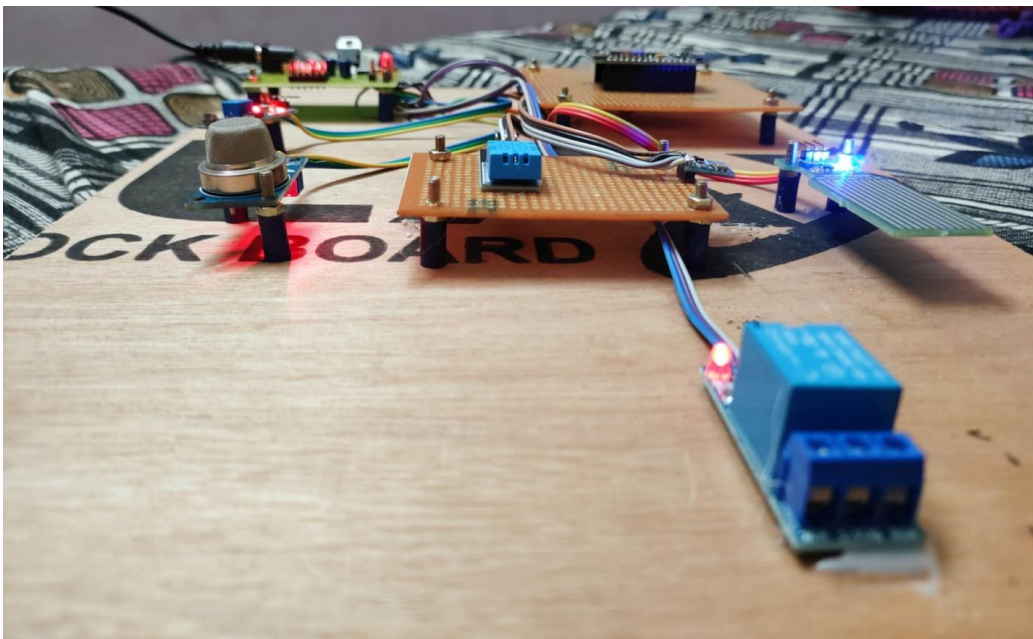


Fig. 5.1: Hardware image 1

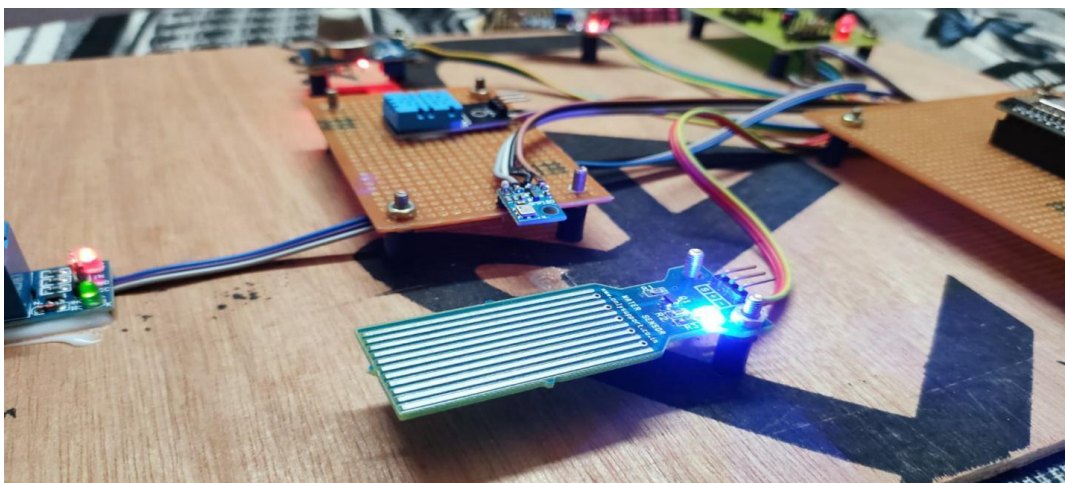


Fig. 5.2: Hardware image 2

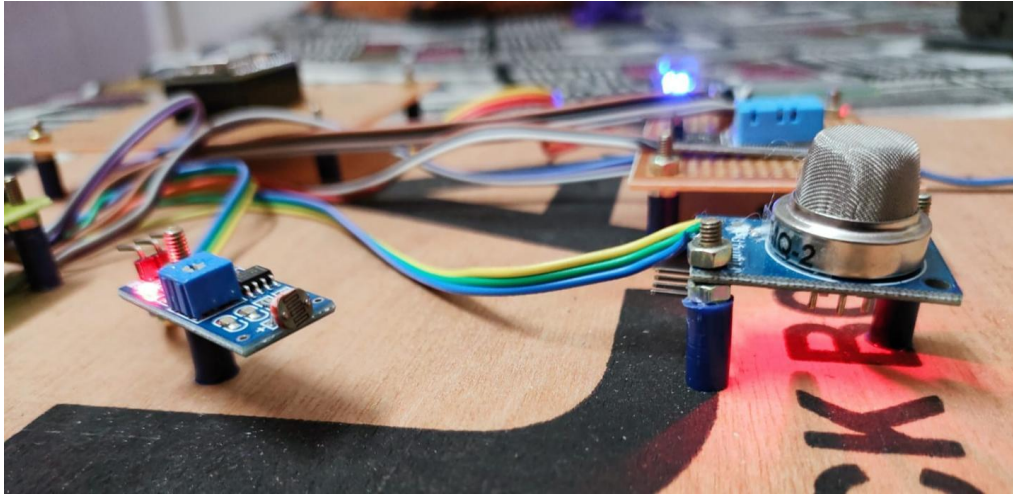


Fig. 5.3: Hardware image 3

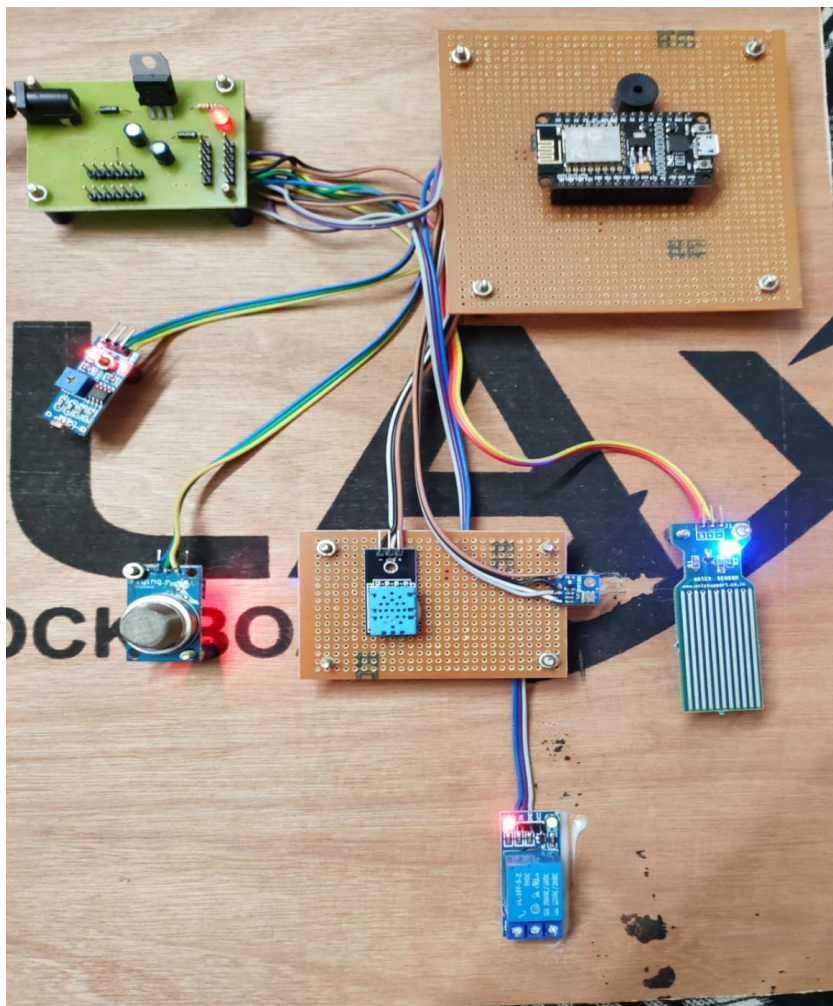


Fig. 5.4: Hardware image 4

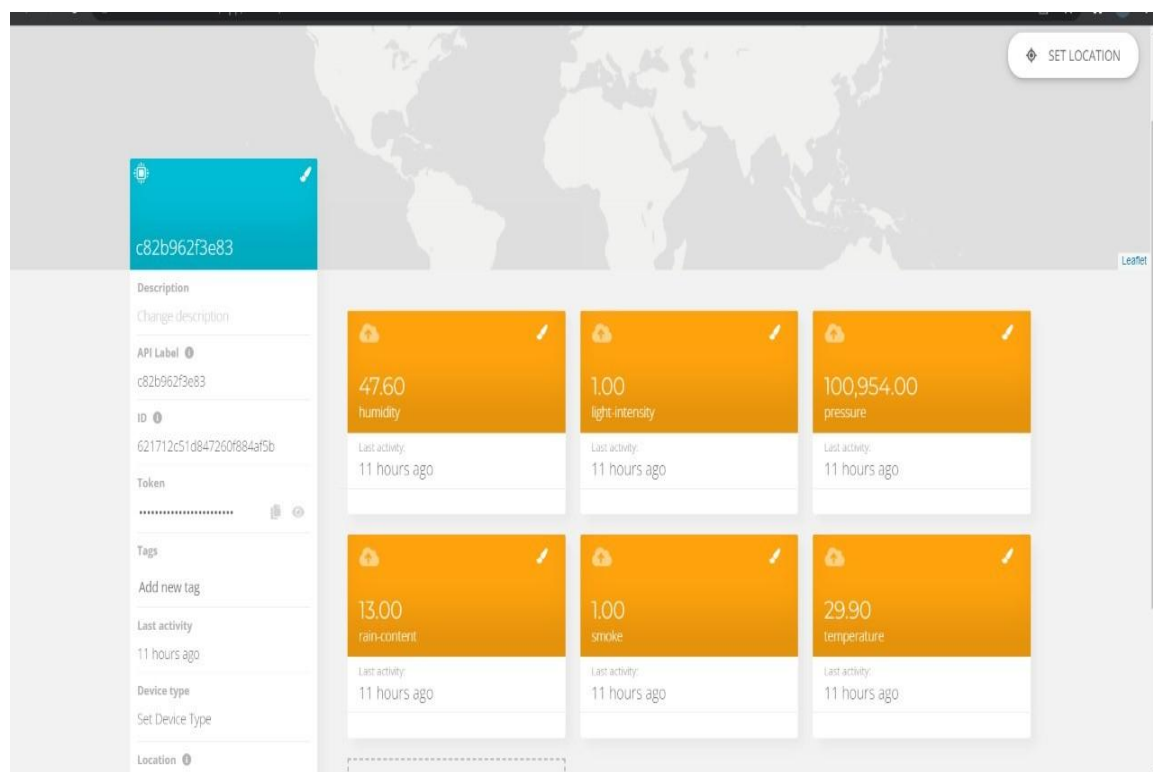


Fig. 5.5: Ubidots Readings of Parameters

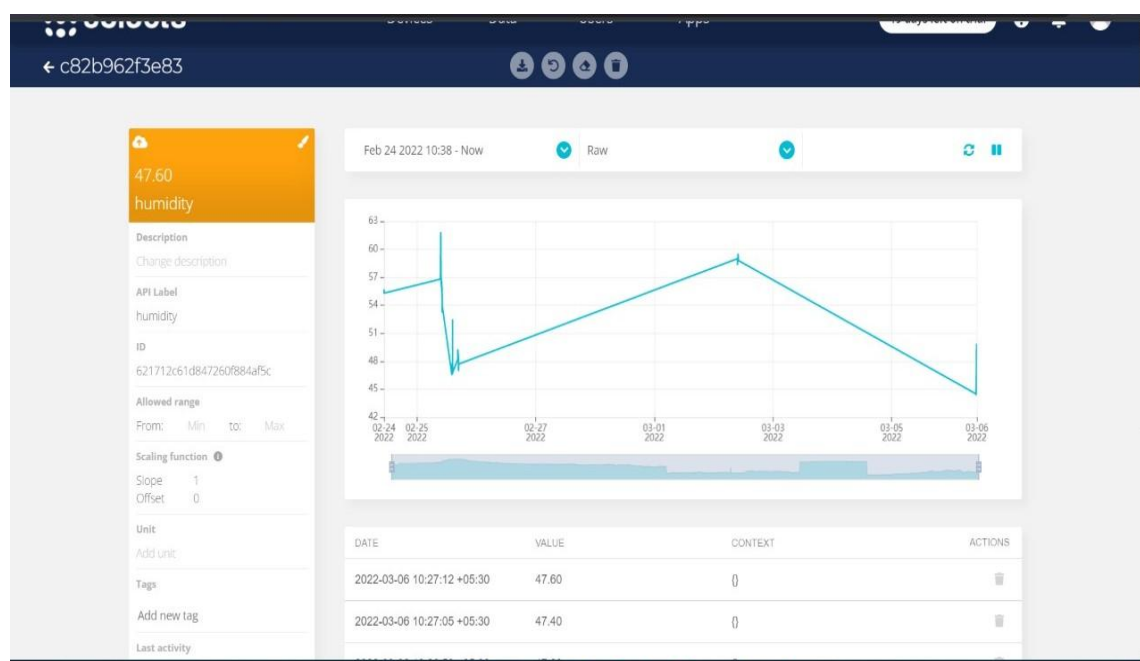


Fig. 5.6: Graph of parameter readings

By using the Ubidots in cloud, we have run the code successfully in Microcontroller and we have got the results. We can see the readings of the parameters from anywhere in the world. We have also done analysis on rain prediction using Neural Networks concept.

5.2 CONCLUSION

Our agenda of this project is to monitor the environmental conditions like temperature, humidity, light intensity and etc. We have achieved it in this project. We are able to get the readings of the environmental parameters with the help of sensors and NodeMcu. The readings got displayed on the screen with the help of cloud (Ubidots). If the temperature and humidity are too high, we get an alert message. With the help of parameter readings, we have done analysis on rain prediction using Neural Networks Concept.

REFERENCES

- [1] Garima Verma, and Shiva Prakash, "A Comparative study based on Different Energy Saving Mechanisms Based on Green Internet of Things", IEEE 8th International Conference on Reliability, Infocom Technology and Optimization(ICRITO-2020),IEEE Conference Record Number 48877, Amity University, Noida, India,Pid631,June-4-05, 2020.
- [2] Girija, C., Harshalatha, H., Andrea Grace Shires, and Pushpalatha, H. P., "Intenet of Things(IOT) Based Weather Monitoring System", NCESC-2018, Vol. 6-13, 24-04-2018,IJERT.
- [3] Jamal Mabrouki, Mourade Azrour, Driss Dhiba, Yousef Farhaoui, and Souad El Hajjaji, "IoT-Based Data Logger for Weather Monitoring Using Arduino-Based Wireless Sensor Networks with Remote Graphical Application and Alerts", Big Data Mining and Analytics, Vol. 4, Issue: 1,March 2021.
- [4] Pauzi, A. F. and Hasan, M. Z. "Development of IOT Based Weather Reporting System", IOP Conf Ser..Mater.Sci.Eng.917 012032, 2020.
- [5] Puja Sharma, and Shiva Prakash, "Real Time Weather Monitoring System Using Iot",Department of Information Technology and Computer Application, Madan Mohan Malaviya University of Technology, Gorakhpur, 2021.
- [6] Raj kumar, and Shiva Prakash , "Performance & parametric analysis of IOT'S motes with different electrical and electronics engineering, jointly organized by School of Engineering", The university of Malaya, Kuala Lumpur, Malaysia & center of Excellence – Power Engineering and Clean Energy Integration, Galgotias University, India in the collaboration with NPTI, Pid 175,January 2nd-3rd,2021.
- [7] Rajalakshmi Krishnamurthi, Adarsh Kumar, Dhanalekhshmi Gopinathan, Anand Navyar, and Basit Qureshi, "An Overview of IOT Sensor Data Processing, Fusion and Analysis Techniques", Sensors, 2020,6076; <https://doi.org/10.3390/s20216076>
- [8] Sajishree Srivastava, and Shiva Prakash, "An Analysis of Various IOT Security Techniques: A Review", IEEE 8th International Conference on Reliability, Infocom Technology and Optimization (ICRTO-2020), IEEE Conference Record Number 48877, Amity University, Noida, India, June 4 2020.
- [9] Zi-Qi Huang, Ying-Chih Chen and Chih-Yu Wen, "Real time Weather Monitoring and Prediction Using City Buses and Machine learning", Vol.3-21 Published 10 September 2020.