BAPATLA ENGINEERING COLLEGE :: BAPATLA

(Autonomous)

Department of Electrical and Electronics Engineering

INSTITUTE VISION

- To build centers of excellence, impart high quality education and instill high standards of ethics and professionalism through strategic efforts of our dedicated staff, which allows the college to effectively adapt to the ever-changing aspects of education.
- To empower the faculty and students with the knowledge, skills and innovative thinking to facilitate discovery in numerous existing and yet to be discovered fields of engineering, technology and interdisciplinary endeavors.

INSTITUTE MISSION

- Our Mission is to impart the quality education at par with global standards to the students from all over India and in particular those from the local and rural areas.
- We continuously try to maintain high standards so as to make them technologically competent and ethically strong individuals who shall be able to improve the quality of life and economy of our country.

DEPARTMENT VISION

The Department of Electrical & Electronics Engineering will provide programs of the highest quality to produce globally competent technocrats who can address challenges of the millennium to achieve sustainable socio - economic development.

DEPARTMENT MISSION

M1: To provide quality teaching blended with practical skills.

M2: To prepare the students ethically strong and technologically competent in the field of Electrical and Electronics Engineering.

M3: To motivate the faculty and students in the direction of research and focus to fulfill social needs.

PROGRAM OUTCOMES

Program Outcomes		Engineering Graduates will be able to
PO1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	Problem analysis	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

Program Outcomes		Engineering Graduates will be able to
PO8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	Communication	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1	The Electrical and Electronics Engineering graduates are capable of applying the knowledge of mathematics and sciences in modern power industry.
PSO2	Analyze and design efficient systems to generate, transmit, distribute and utilize electrical energy to meet social needs using power electronic systems.
PSO3	Electrical Engineers are capable to apply principles of management and economics for providing better services to the society with the technical advancements in renewable and sustainable energy integration

INTERNET OF THINGS LAB

3/4 EEE V semester Code: 20EEL501

LIST OF EXPERIMENTS:

Mandatory Experiments:

1. a) Familiarization with Arduino/Raspberry Pi and perform necessary software installation.
   b) Study the fundamental IOT Software & Components.
2. a) Interface LED & Buzzer with Arduino and write a program to turn ON LED for 1 sec with a delay of 2seec.
   b) Interface LED & Buzzer with Raspberry Pi and write a program to turn ON LEDfor 1 sec with a delay of 2seec.
3. a) Implement two-way traffic control using Arduino.
   b) Implement two-way traffic control using Raspberry Pi
4. a) Interface DHT11 sensor with Arduino and write a program to print temperature and humidity readings.
   b) Interface PIR sensor with Arduino and write a program to turn ON LED at sensor detection.
5. Interface Stepper motor with Arduino and write a program to control stepper motor**.**
6. Interface OLED with Arduino and write a program to print temperature and humidity readings on it**.**

Application Oriented Experiments:

7. Interface servo motor using with Raspberry Pi and write a program to control servo motor.
8. Write a program for weather monitoring station and handling temperature & humidity values on cloud platform.
9. Design of digital dc voltmeter and ammeter using Arduino uno.
10. Design home automation using Raspberry pi.
11. Servo Motor control With Esp32.
12. Measurement of Power and Energy using Arduino.
13. Over/Under Voltage Protection of Home Appliances using Arduino uno & NODE MCU.
14. Design smart irrigation system and analyze data using cloud platform.
15. Detection of induction motor fault using IOT.

Note: Minimum 10 experiments should be conducted

Bapatla Engineering College: Bapatla

(AUTONOMOUS)

Department of Electrical & Electronics Engineering

1. a) Familiarization with Arduino/Raspberry Pi and perform necessary software installation.
b) Study the fundamental IOT Software & Components

EXPERIMENT NO- 1 A

OBJECTIVE: Study the fundamental of IOT softwares and components.

RESOURCE REQUIRED: Proteus and Arduino IDE

THEORY: Internet of Things (IoT) is a network of physical objects or people called "things" that are embedded with software, electronics, network, and sensors that allows these objects to collect and exchange data.

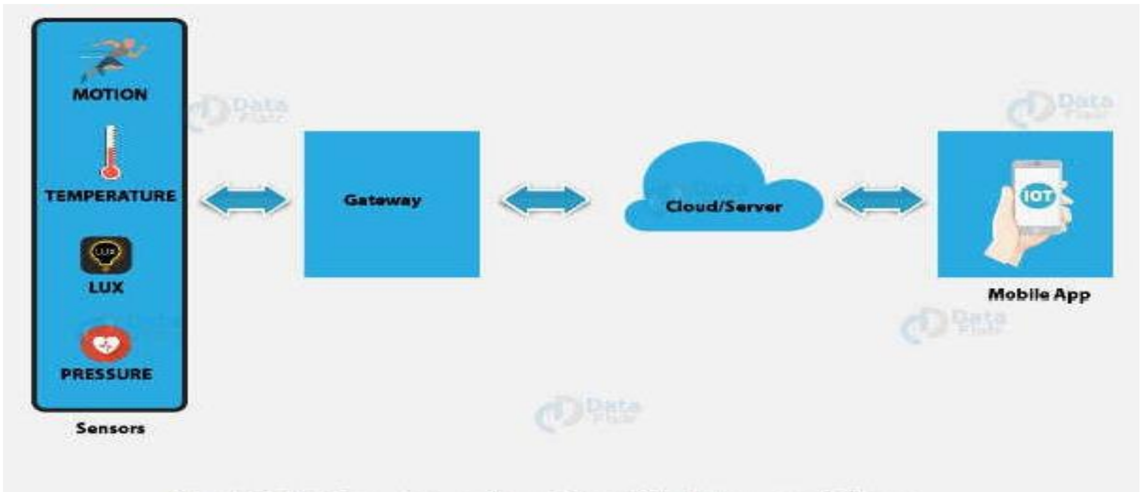
The thing in IoT can also be a person with a diabetes monitor implant, an animal with tracking devices, etc.

Components of IOT:

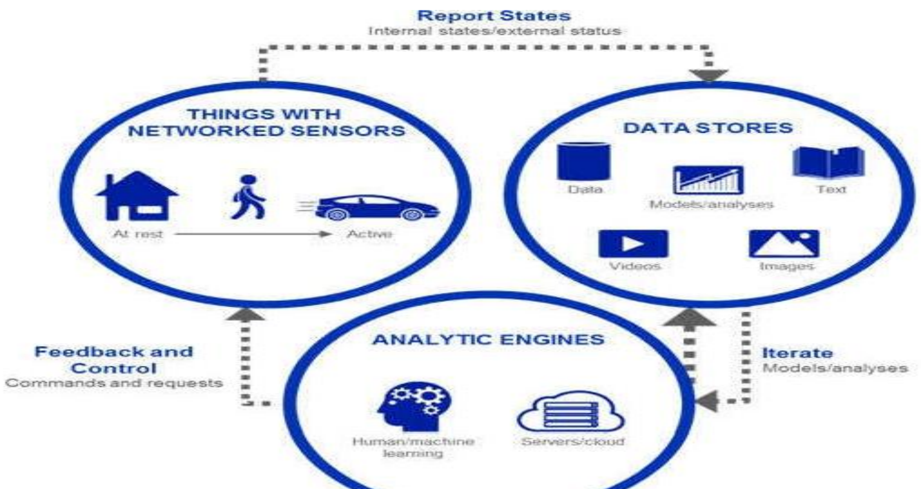
1) Sensors/Devices: Sensors or devices are a key component that helps you to collect live data from the surrounding environment. A device may have various types of sensors which performs multiple tasks apart from sensing. Example, A mobile phone is a device which has multiple sensors like GPS, camera but your smartphone is not able to sense these things.
2) Connectivity: All the collected data is sent to a cloud infrastructure. The sensors should be connected to the cloud using various mediums of communications. These communication mediums include mobile or satellite networks, Bluetooth, WI-FI, WAN, etc.
3) Data Processing: Once that data is collected, and it gets to the cloud, the software performs processing on the gathered data. This process can be just checking the temperature, reading on devices like AC or heaters. However, it can sometimes also be very complex like identifying objects, using computer vision on video.
4) User Interface: The information needs to be available to the end-user in some way which can be achieved by triggering alarms on their phones or sending them notification through email or text message. The user sometimes might need an interface which actively checks their IoT system. For example, the user has a camera installed in his home. He wants to access video recording and all the feeds with the help of a web server.

5) Depending on the IoT application and complexity of the system, the user may also be able to perform an action which may create cascading effects.
6) For example, if a user detects any changes in the temperature of the refrigerator, with the help of IoT technology the user should able to adjust the temperature with the help of their mobile phone.

Application type	**Description**
Smart Thermostats	Helps you to save resource on heating bills by knowing your usage patterns.
Connected Cars	IoT helps automobile companies handle billing, parking, insurance, and other related stuff automatically.
Activity Trackers	Helps you to capture heart rate pattern, calorie expenditure, activity levels, and skin temperature on your wrist.
Smart Outlets	Remotely turn any device on or off. It also allows you to track a device's energy level and get custom notifications directly into your smartphone.
Parking Sensors	IoT technology helps users to identify the real-time availability of parking spaces on their phone.
Connect Health	The concept of a connected health care system facilitates real-time health monitoring and patient care. It helps in improved medical decision-making based on patient data.
Smart City	Smart city offers all types of use cases which include traffic management to water distribution, waste management, etc.
Smart home	Smart home encapsulates the connectivity inside your homes. It includes smoke detectors, home appliances, light bulbs, windows, door locks, etc.
Smart supply chain	Helps you in real time tracking of goods while they are on the road, or getting suppliers to exchange inventory information.



How IoT Works – Internal working of the Internet of Things



IoT Components – Data Processing

How does Internet of Thing (IoT) Work?

The working of IoT is different for different IoT echo system (architecture). However, the key concept of there working are similar. The entire working process of IoT starts with the device themselves, such as smartphones, digital watches, electronic appliances, which securely communicate with the IoT platform. The platforms collect and analyze the data from all multiple devices and platforms and transfer the most valuable data with applications to devices.

Features of IOT

The most important features of IoT on which it works are connectivity, analyzing, integrating, active engagement, and many more. Some of them are listed below:

Connectivity: Connectivity refers to establish a proper connection between all the things of IoT to IoT platform it may be server or cloud. After connecting the IoT devices, it needs a high speed messaging between the devices and cloud to enable reliable, secure and bi-directional communication.

Analyzing: After connecting all the relevant things, it comes to real-time analyzing the data collected and use them to build effective business intelligence. If we have a good insight into data gathered from all these things, then we call our system has a smart system.

Integrating: IoT integrating the various models to improve the user experience as well.

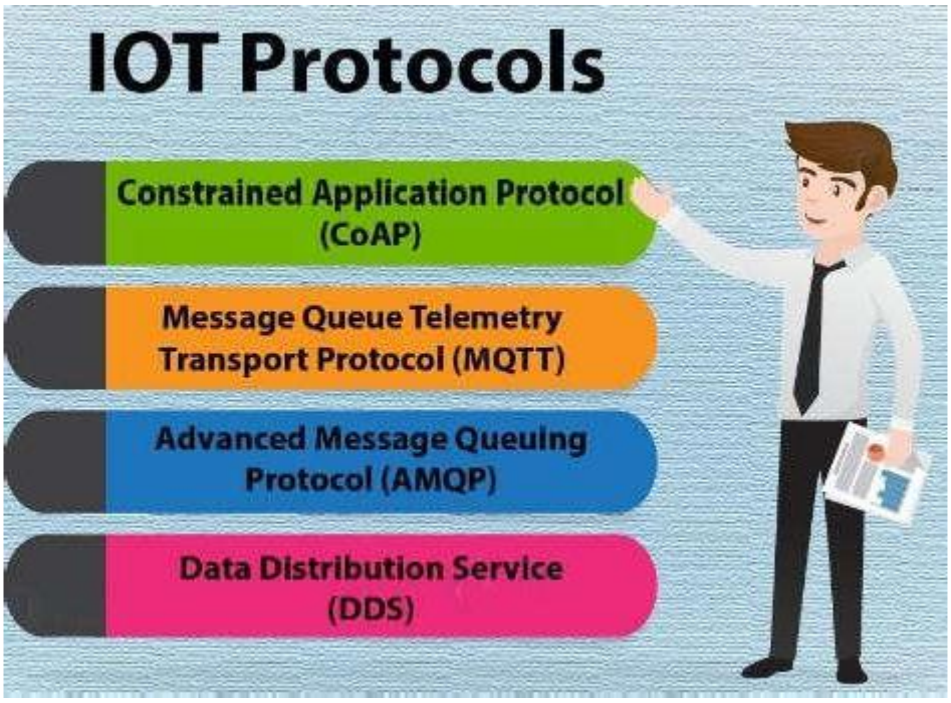
Artificial Intelligence: IoT makes things smart and enhances life through the use of data. For example, if we have a coffee machine whose beans have going to end, then the coffee machine itself order the coffee beans of your choice from the retailer.

Sensing: The sensor devices used in IoT technologies detect and measure any change in the environment and report on their status. IoT technology brings passive networks to active networks. Without sensors, there could not hold an effective or true IoT environment.

Active Engagement: IoT makes the connected technology, product, or services to active engagement between each other.

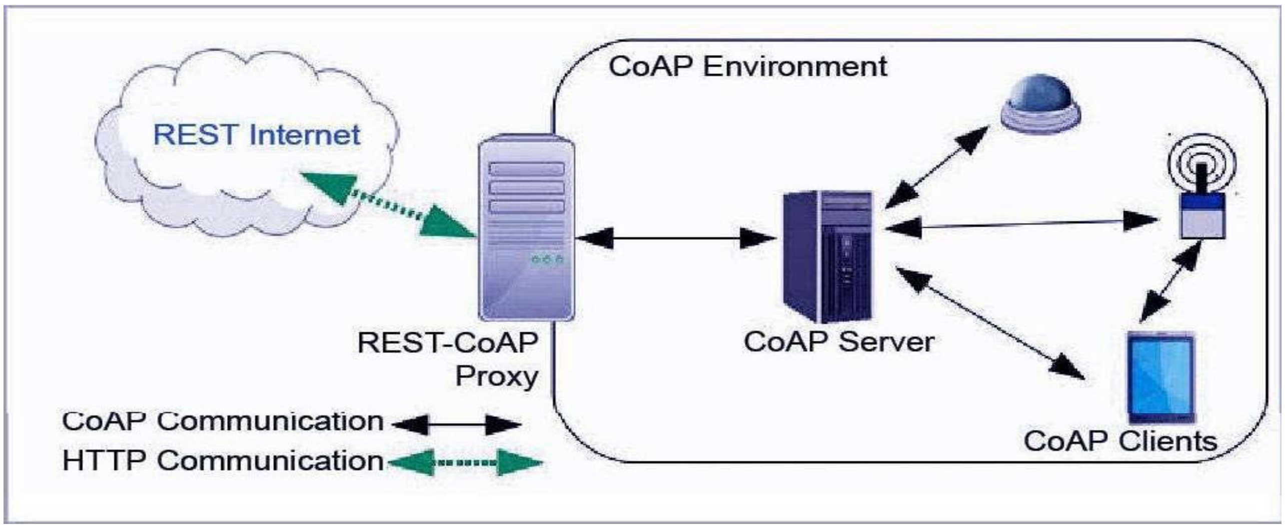
Endpoint Management: It is important to be the endpoint management of all the IoT system otherwise, it makes the complete failure of the system. For example, if a coffee machine itself order the coffee beans when it goes to end but what happens when it orders the beans from a retailer and we are not present at home for a few days, it leads to the failure of the IoT system. So, there must be a need for endpoint management.

IoT Protocols

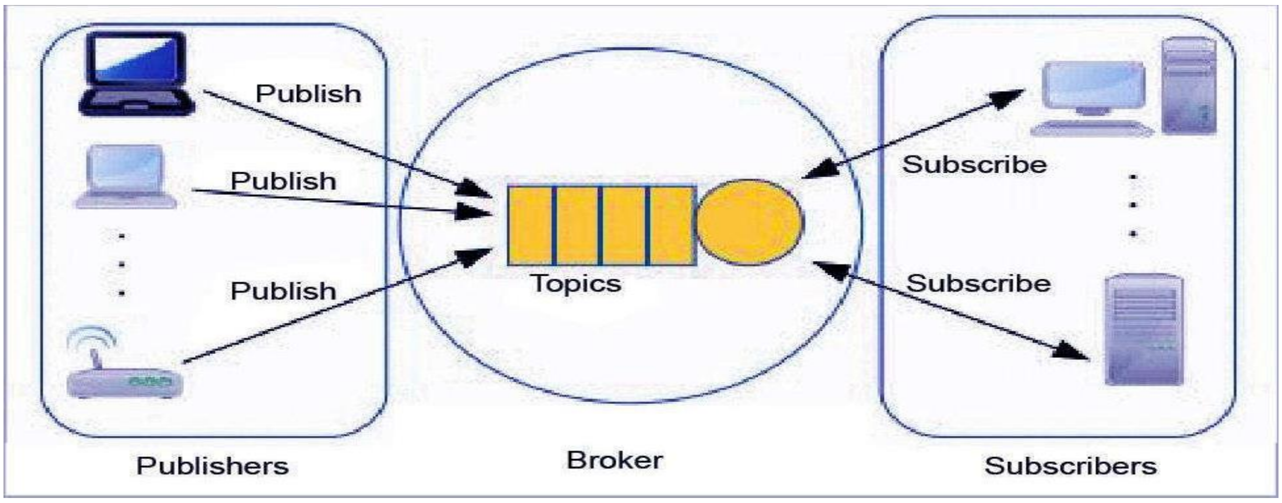

IOT Protocols
Constrained Application Protocol (CoAP)
Message Queue Telemetry Transport Protocol (MQTT)
Advanced Message Queuing Protocol (AMQP)
Data Distribution Service (DDS)

1. Constrained Application Protocol (CoAP)

CoAP is an internet utility protocol for constrained gadgets. It is designed to enable simple, constrained devices to join IoT through constrained networks having low bandwidth availability. This protocol is primarily used for machine-to-machine (M2M) communication and is particularly designed for IoT systems that are based on HTTP protocols.



CoAP makes use of the UDP protocol for lightweight implementation. It also uses restful architecture, which is just like the HTTP protocol. It makes use of dtls for the cozy switch of statistics within the slipping layer.



2. Message Queue Telemetry Transport Protocol (MQTT)

MQTT (Message Queue Telemetry Transport) is a messaging protocol developed with the aid of Andy Stanford-Clark of IBM and Arlen Nipper of Arcom in 1999 and is designed for

M2M communication. It‘s normally used for faraway tracking in IoT. Its primary challenge is to gather statistics from many gadgets and delivery of its infrastructure. MQTT connects gadgets and networks with packages and middleware. All the devices hook up with facts concentrator servers like IBM‘s new message sight appliance. MQTT protocols paintings on top of TCP to offer easy and dependable streams of information.

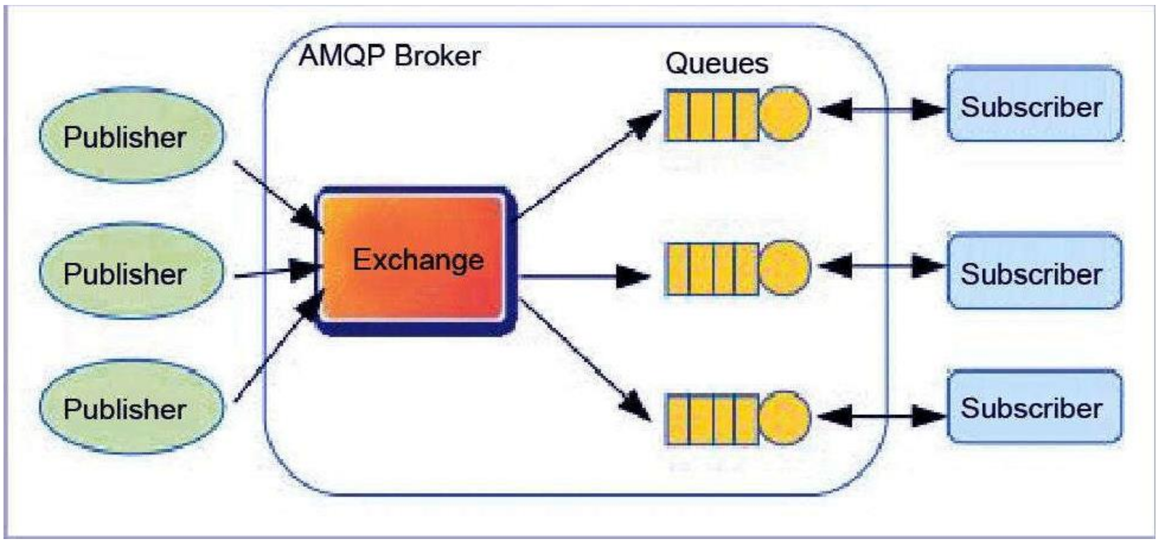
These IoT protocols include 3 foremost additives: subscriber, publisher, and dealer. The writer generates the information and transmits the facts to subscribers through the dealer. The dealer guarantees safety by means of move-checking the authorization of publishers and subscribers.

3. Advanced Message Queuing Protocol (AMQP)

This was evolved by John O‘Hara at JP Morgan Chase in London. AMQP is a software layer protocol for message-oriented middleware environment. It supports reliable verbal exchange through message transport warranty primitives like at-most-once, at least once and exactly as soon as shipping.

The AMQP – IoT protocols consist of hard and fast components that route and save messages within a broker carrier, with a set of policies for wiring the components together. The AMQP protocol enables patron programs to talk to the dealer and engage with the AMQP model. This version has the following three additives, which might link into processing chains in the server to create the favored capabilities.

- Exchange: Receives messages from publisher primarily based programs and routes them to ‗message queues‘.
- Message Queue: Stores messages until they may thoroughly process via the eating client software.
- Binding: States the connection between the message queue and the change.

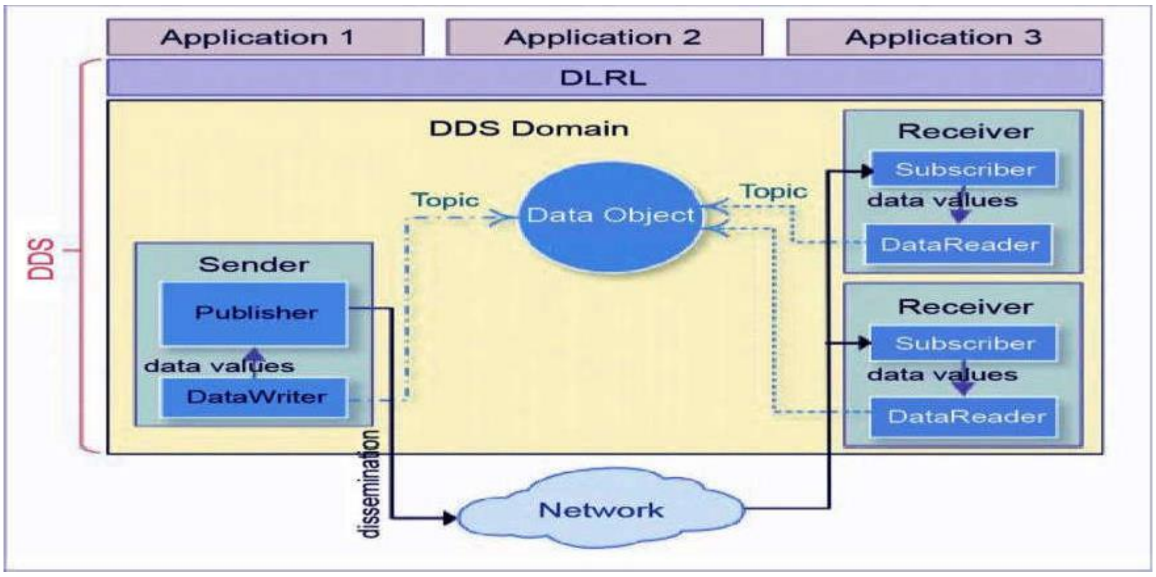


4. Data Distribution Service (DDS)

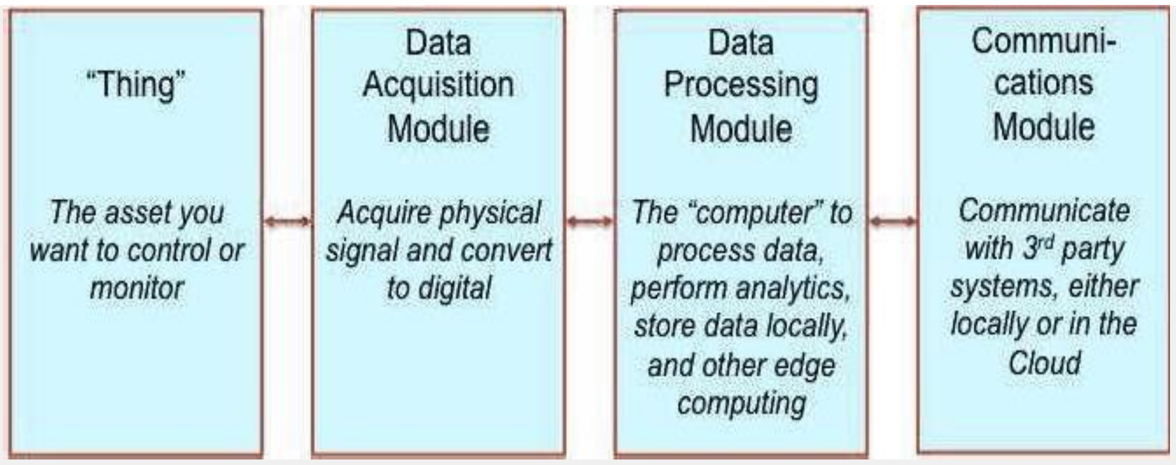
It enables a scalable, real-time, reliable, excessive-overall performance and interoperable statistics change via the submit-subscribe technique. DDS makes use of multicasting to convey high-quality QoS to applications.

DDS is deployed in platforms ranging from low-footprint devices to the cloud and supports green bandwidth usage in addition to the agile orchestration of system additives.
The DDS – IoT protocols have fundamental layers: facts centric submit-subscribe (dcps) and statistics-local reconstruction layer (dlrl).

Dcps plays the task of handing over the facts to subscribers, and the dlrl layer presents an interface to dcps functionalities, permitting the sharing of distributed data amongst IoT enabled objects.



Building Blocks of IoT Hardware


"Thing"
The asset you want to control or monitor
Data Acquisition Module
Acquire physical signal and convert to digital
Data Processing Module
The "computer" to process data, perform analytics, store data locally, and other edge computing
Communi-cations Module
Communicate with 3rd party systems, either locally or in the Cloud

IoT Hardware – Building Blocks

i. Thing

–Thing‖ in IOT is the asset that you want to control or monitor or measure, that is, observe closely. In many IoT products, the –thing‖ gets fully incorporated into a smart device. For

example, think of products like a smart refrigerator or an automatic vehicle. These products control and monitor themselves.
There are sometimes many other applications where the thing stands as an alone device, and a separate product is connected to ensure it possesses smart capabilities.

ii. Data Acquisition Module

The data acquisition module focuses on acquiring physical signals from the thing which is being observed or monitored and converting them into digital signals that can be manipulated or interpreted by a computer.
This is the hardware component of an IOT system that contains all the sensors that help in acquiring real-world signals such as temperature, pressure, density, motion, light, vibration, etc. The type and number of sensors you need depend on your application.
This module also includes the necessary hardware to convert the incoming sensor signal into digital information for the computer to use it. This includes conditioning of incoming signal, removing noise, analog-to-digital conversion, interpretation, and scaling.

iii. Data Processing Module

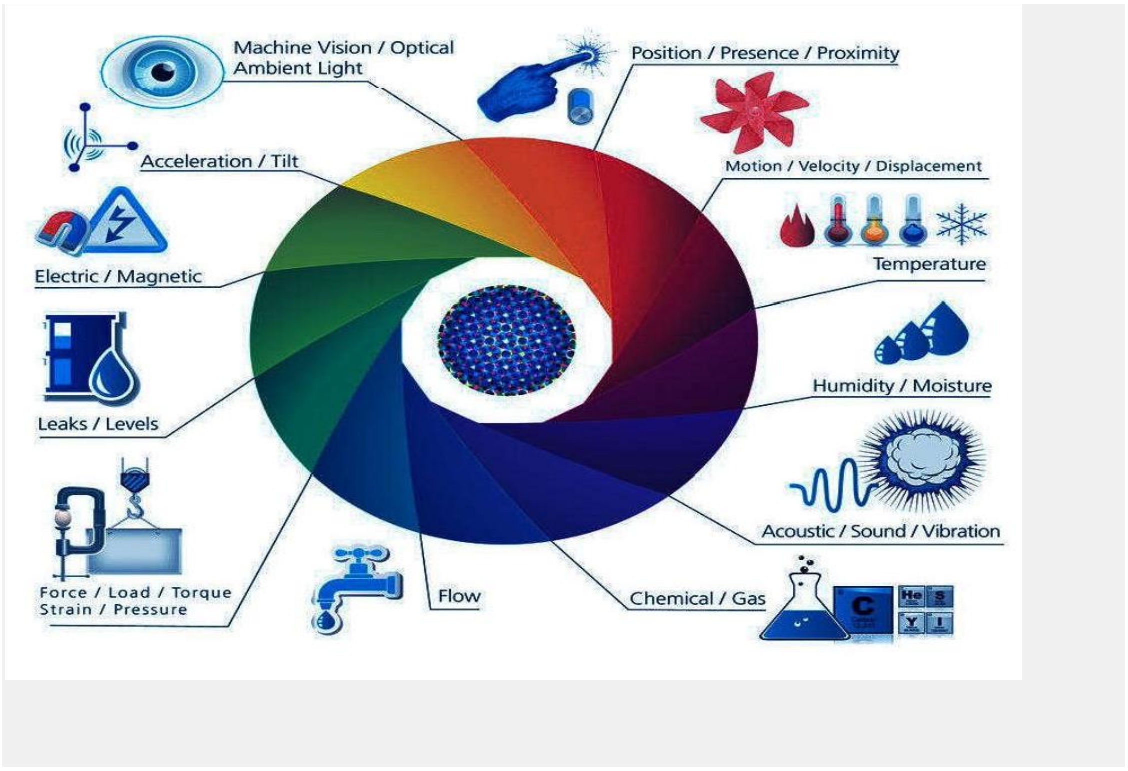
The third building block of the IoT device is the data processing module. This is the actual –computer‖ and the main unit that processes the data performs operations such as local analytics, stores data locally, and performs some other computing operations.

iv. Communication Module

The last building block of IOT hardware is the communications module. This is the part that enables communications with your Cloud Platform, and with 3rd party systems either locally or in the Cloud.

b. IoT Sensors

The most important IoT hardware might be its sensors. These devices consist of a variety of modules such as energy modules, RF modules, power management modules, and sensing modules.



c. Wearable Electronic Devices

Wearable electronic devices are small devices that can be worn on the head, neck, arms, torso, and feet.

Current smart wearable devices include −

- **Head** − Helmets, glasses,
- **Neck** − Jewelry, collars
- **Arm** − Wristwatches, wristbands, rings

- **Torso** – Clothing pieces, backpacks
- **Feet** – Shoes, Socks

d. Basic Devices

The day to day devices that we use such as desktop, cellphones and tablets remain integral parts of IoT system.

- The **desktop** provides the user with a very high level of control over the system and its settings.
- The **tablet** acts as a remote and provides access to the key features of the system.
- **Cellphone** allows remote functionality and some essential settings modification

Other key connected devices include standard network devices like **routers** and **switches**.

IoT Software

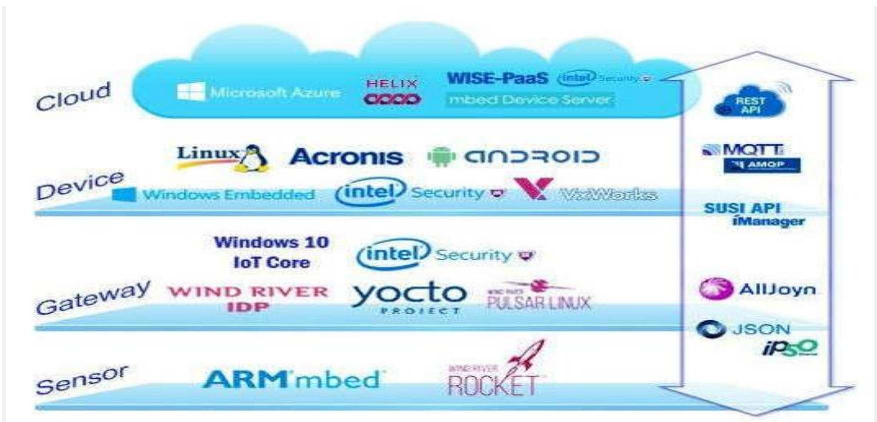
Embedded systems have less storage and processing power, their language needs are different. The most commonly used operating systems for such embedded systems are Linux or UNIX-like OSs like Ubuntu Core or Android.

IoT software encompasses a wide range of software and programming languages from general-purpose languages like C++ and Java to embedded-specific choices like Google‘s Go language or Parasail.

Few IoT Softwares are-

- C & C++: The C programming language has its roots in embedded systems—it even got its start for programming telephone switches. It‘s pretty ubiquitous, that is, it can be used almost everywhere and many programmers already know it. C++ is the object-oriented version of C, which is a language popular for both the Linux OS and Arduino embedded IoT software systems. These languages were basically written for the hardware systems which makes them so easy to use.

- Java: While C and C++ are hardware specific, the code in JAVA is more portable. It is more like a write once and read anywhere language, where you install libraries, invests time in writing codes once and you are good to go.
- Python**:** There has been a recent surge in the number of python users and has now become one of the –go-to‖ languages in Web development. Its use is slowly spreading to the embedded control and IoT world—specially the Raspberry Pi processor. Python is an interpreted language, which is, easy toread, quick to learn and quick to write. Also, it‘s a powerhouse for serving data-heavy applications.
- Unlike most of the languages mentioned so far, it was specifically designed for embedded systems, it‘s small and compact and has less memory size.
- Data Collection: It is used for data filtering, data security, sensing, and measurement. The protocols aid in decision making by sensing form real-time objects. It can work both ways by collecting data from devices or distributing data to devices. All the data transmits to a central server.
- Device Integration: This software ensures that devices bind and connect to networks facilitating information sharing. A stable cooperation and communication ensure between multiple devices.
- Real-Time Analytics: In this, the input from users serves as potential data for carrying out real-time analysis, making insights, suggesting recommendations to solve organizations problems and improve its approach. This, as a result, allows automation and increased productivity.
- Application and Process Extension: These applications extend the reach of existing systems and software to allow a wider, more effective system. They integrate predefined devices for specific purposes such as allowing certain mobile devices or engineering instruments access. It supports improved productivity and more accurate data collection.



RESULT AND CONCLUSION:

We have studied the fundamentals of IOT softwares and components. Internet of Things can create information about the connected objects, analyze it, and make decisions. All IoT applications are using sensors to detect and collect data that are used to give a proper decision that maintains a high level of security of the installations.

EXPERIMENT NO.1 *B*

OBJECTIVE: Familiarization with Arduino/Raspberry Pi and perform necessary software installation

RESOURCE REQUIRED: Arduino IDE

THEORY: Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board. Arduino provides a standard form factor that breaks the functions of the microcontroller into a more accessible package. Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are –

 Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.

 You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).

 Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code on to the board. You can simply use a USB cable.

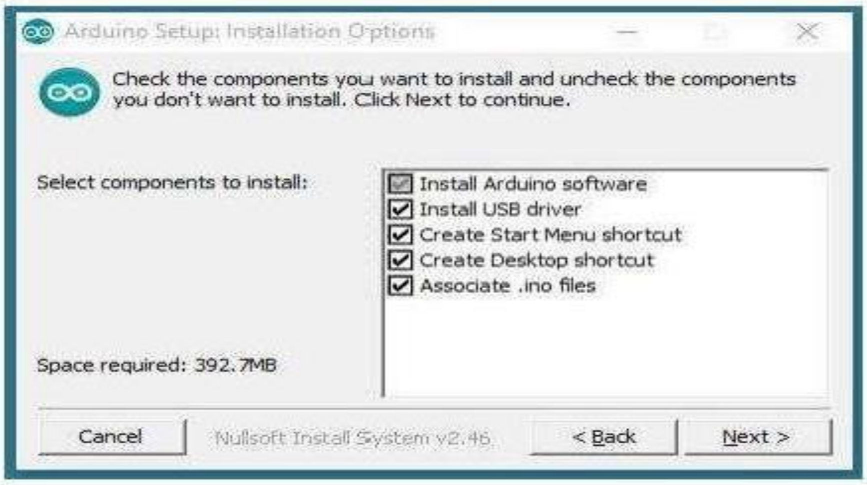
 Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.

 Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Download the Arduino Software (IDE)

 Get the latest version from the arduino.cc web site. You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package you need to install the drivers manually. The Zip file is also useful if you want to create a portable installation.

 When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.


Arduino Setup: Installation Options

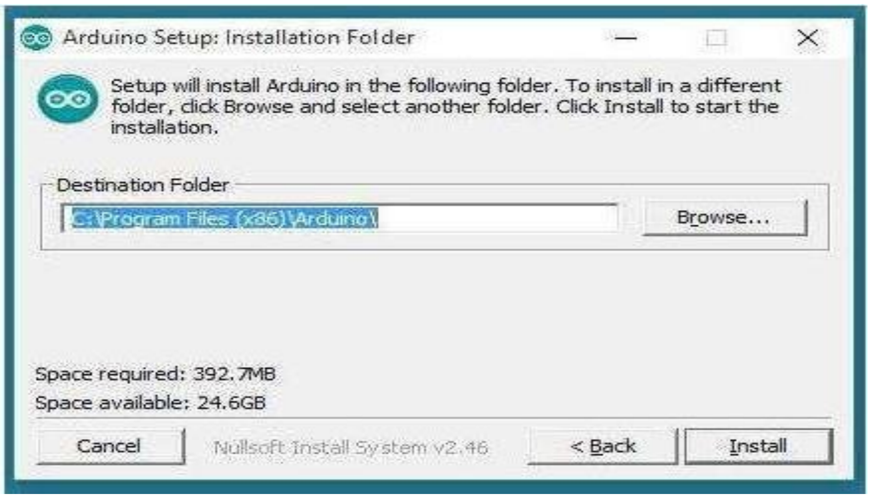
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [✓] Install Arduino software
- [✓] Install USB driver
- [✓] Create Start Menu shortcut
- [✓] Create Desktop shortcut
- [✓] Associate .ino files

Space required: 392.7MB

Choose the components to install

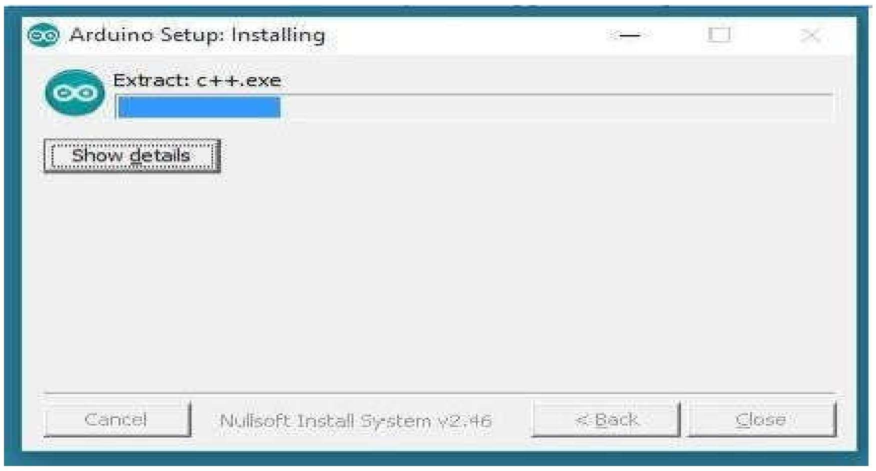

Arduino Setup: Installation Folder

Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Arduino\

Space required: 392.7MB
Space available: 24.6GB

Choose the installation directory (we suggest to keep the default one)



Extract: c++.exe

Nullsoft Install System v2.46

The process will extract and install all the required files to execute properly the Arduino Software (IDE)

 Proceed with board specific instructions

 When the Arduino Software (IDE) is properly installed you can go back to the Different Arduino Boards:

1. Arduino Uno: This is the latest revision of the basic Arduino USB board. It connects to the computer with a standard USB cable and contains everything else you need to program and use the board.

2. Arduino NG: REV-C Revision C of the Arduino NG does not have a built-in LED on pin 13 - instead you'll see two small unused solder pads near the labels "GND" and "13".

3. Arduino Bluetooth: The Arduino BT is a microcontroller board originally was based on the ATmega168, but now is supplied with the 328, and the Bluegiga WT11 bluetooth module. It supports wireless serial communication over bluetooth.

4. Arduino Mega: The original Arduino Mega has an ATmega1280 and an FTDI USB-to serial chip.

5. Arduino NANO: The Arduino Nano 3.0 has an ATmega328 and a two-layer PCB. The power LED moved to the top of the board.

Advantages of using Arduino:

It offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50

- Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

RESULT AND CONCLUSION:

In this experiment, we have discussed about the Arduino fundamentals and its necessary software installation. Thus, Arduino simplifies the process of working with microcontrollers, Arduino is an open source micro-controller which can be used as a development board for hundreds and thousands of project. we can program the Arduino board with the help of Arduino IDE and the programming is based on C/C++.

2. a) To interface LED & Buzzer with Arduino and write a program to turn ON LED for 1 sec after every 2 seconds.

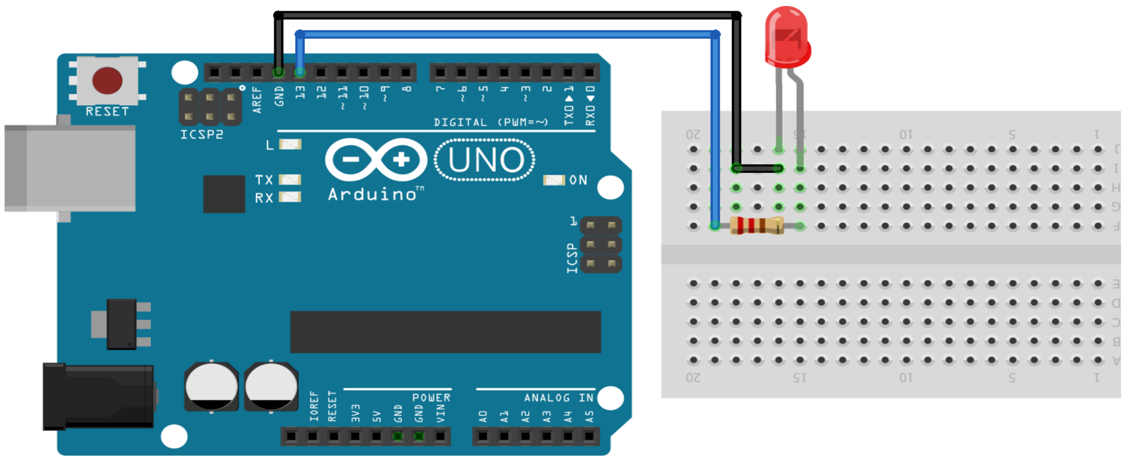
b) To interface LED & Buzzer with Raspberrypi and write a program to turn ON LED for 1 sec after every 2 seconds.

Experiment No - 2 (A)

AIM : To Interface LED with Arduino and write a program to turn ON LED for 1 Sec after every 2 seconds.

RESOURCE REQUIRED: Arduino IDE Software, Arduino Uno, LED, Buzzer, Bread board.

CIRCUIT DIAGRAM:



PROCEDURE :

1. Make the connection as per the circuit diagram.
2. Open the Arduino IDE in computer and write the program.
3. Compile the program for any errors and upload it to the Arduino.
4. Observe the LED ON time delay of 1 second and OFF time delay of 2 seconds.

CODE:

```
const int led=13;
const int buzzer=12;
void setup()
{
pinMode(led,OUTPUT);
pinMode(buzzer,OUTPUT);

}

void loop()
{
digitalWrite(led,HIGH);
digitalWrite(buzzer,HIGH);
delay (1000);
digitalWrite(led,LOW);
digitalWrite(buzzer,LOW);
delay (1000);
}
```

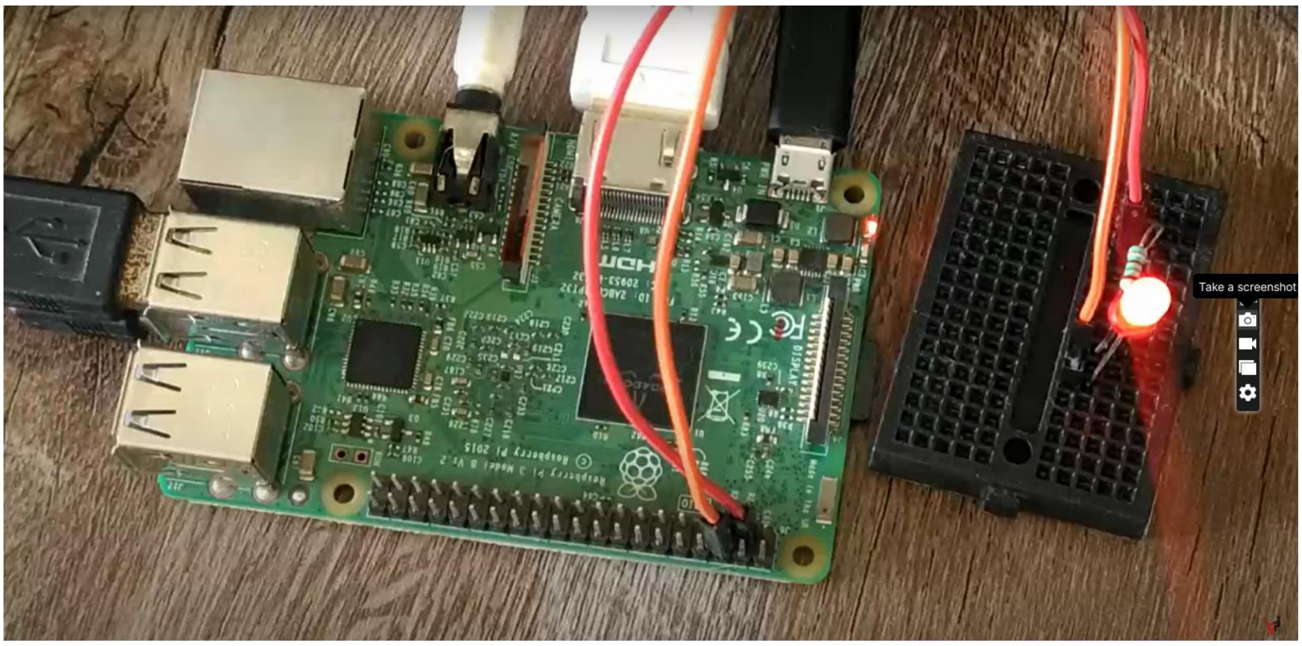
RESULT:

Experiment No- 2 (B)

AIM : To interface LED & Buzzer with RaspberryPi and write a program to turn ON LED for 1 sec after every 2 seconds.

Resource Required: Raspberrypi , LED, ,bread board, Monitor

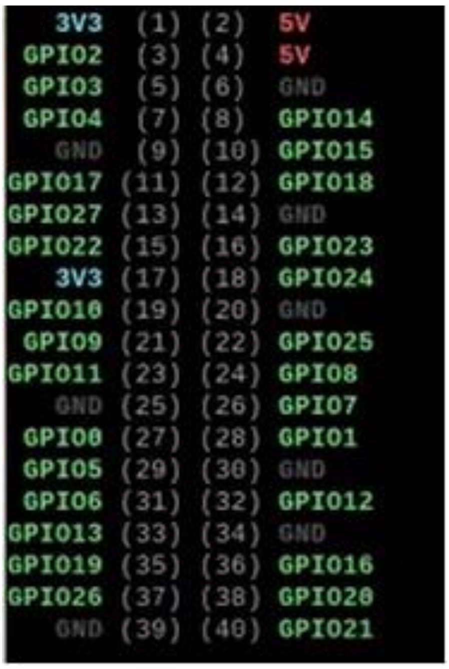
Circuit diagram:



CODE:

Using BCM PINS

```
from gpiozero import Buzzer
from gpiozero import LED
from time import sleep
buzzer=Buzzer(23)
led=LED(24)
while True:
    buzzer.on()
    led.on()
    sleep(0.5)
    buzzer.off()
    led.off()
    sleep(0.5)
```



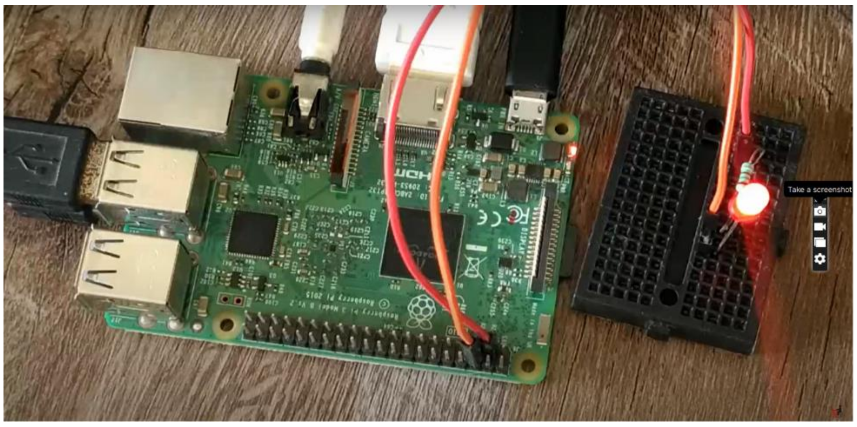
```
    3V3  (1)  (2)  5V
 GPIO2  (3)  (4)  5V
 GPIO3  (5)  (6)  GND
 GPIO4  (7)  (8)  GPIO14
    GND  (9) (10)  GPIO15
GPIO17 (11) (12)  GPIO18
GPIO27 (13) (14)  GND
GPIO22 (15) (16)  GPIO23
    3V3 (17) (18)  GPIO24
GPIO10 (19) (20)  GND
 GPIO9 (21) (22)  GPIO25
GPIO11 (23) (24)  GPIO8
    GND (25) (26)  GPIO7
 GPIO0 (27) (28)  GPIO1
 GPIO5 (29) (30)  GND
 GPIO6 (31) (32)  GPIO12
GPIO13 (33) (34)  GND
GPIO19 (35) (36)  GPIO16
GPIO26 (37) (38)  GPIO20
    GND (39) (40)  GPIO21
```

Using Board PINS

```
from gpiozero import Buzzer
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(16, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)

while True:
    GPIO.output(16, True)
    GPIO.output(18, True)
    time.sleep(0.5)
    GPIO.output(16, False)
    GPIO.output(18, False)
    time.sleep(0.5)
```

CIRCUIT DIAGRAM:



Procedure :

1.Make the connections as per the circuit diagram

2.Open python & type the program.

3.Run the program & verify the output.

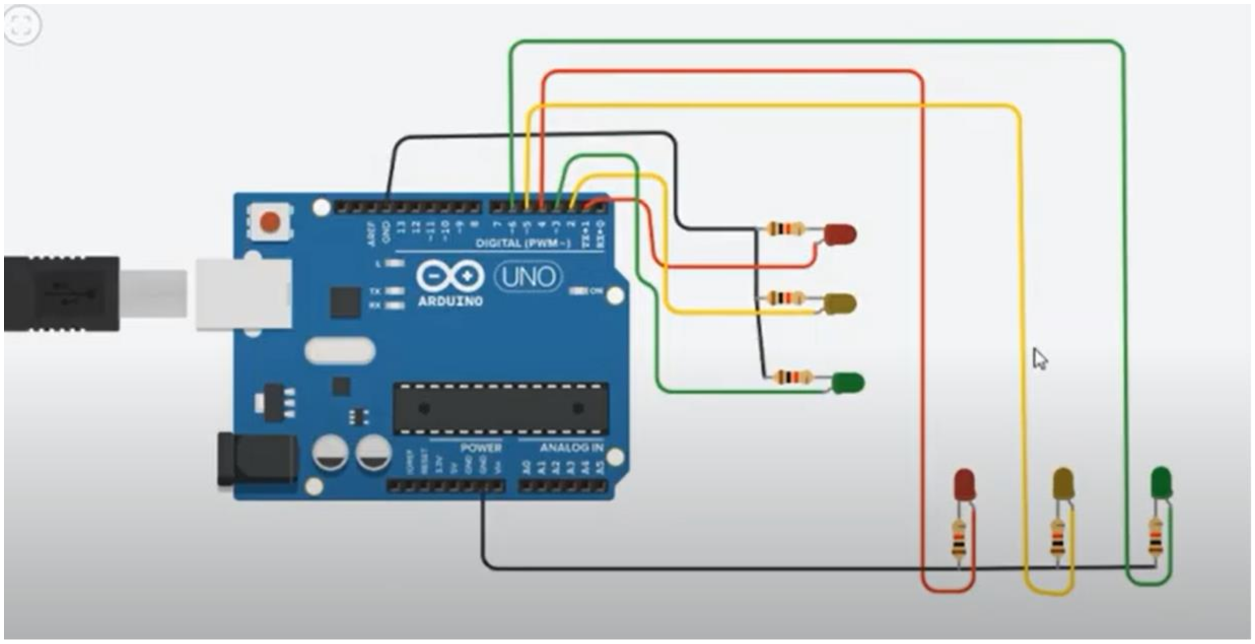
RESULT:

3. a) Implement two-way traffic control using Arduino.

AIM: To implement two-way traffic control using Arduino Uno

Components Required: Arduino Uno, LED'S,Bread board, Jumper Wires

Circuit Diagram:



CODE:

```
int red1=1;
int yellow1=2;
int green1=3;
int red2=4;
int yellow2=5;
int green2=6;
void setup()
{
```

```
pinMode(red1,OUTPUT);
pinMode(yellow1,OUTPUT);
pinMode(green1,OUTPUT);
pinMode(red2,OUTPUT);
pinMode(yellow2,OUTPUT);
pinMode(green2,OUTPUT);
}
void loop()
{
digitalWrite(green1,HIGH);
digitalWrite(red2,HIGH);
delay(10000);
digitalWrite(red1,LOW);
digitalWrite(yellow1,LOW);
digitalWrite(yellow2,LOW);
digitalWrite(green2,LOW);

digitalWrite(green1,LOW);
digitalWrite(yellow1,HIGH);
digitalWrite(yellow2,HIGH);
delay(5000);
digitalWrite(yellow1,LOW);
digitalWrite(yellow2,LOW);
digitalWrite(red1,LOW);
digitalWrite(red2,LOW);
digitalWrite(green2,LOW);

digitalWrite(red1,HIGH);
digitalWrite(green2,HIGH);
delay(10000);
digitalWrite(red1,LOW);
digitalWrite(green2,LOW);
digitalWrite(yellow1,LOW);
digitalWrite(green1,LOW);
digitalWrite(red2,LOW);
digitalWrite(yellow2,LOW);
}
```

Procedure:

1.Make the connections as per circuit diagram

2.Open Arduino Ide & type the program

3.Compile & upload the program in to Arduino Uno board.

4.Verify the output.

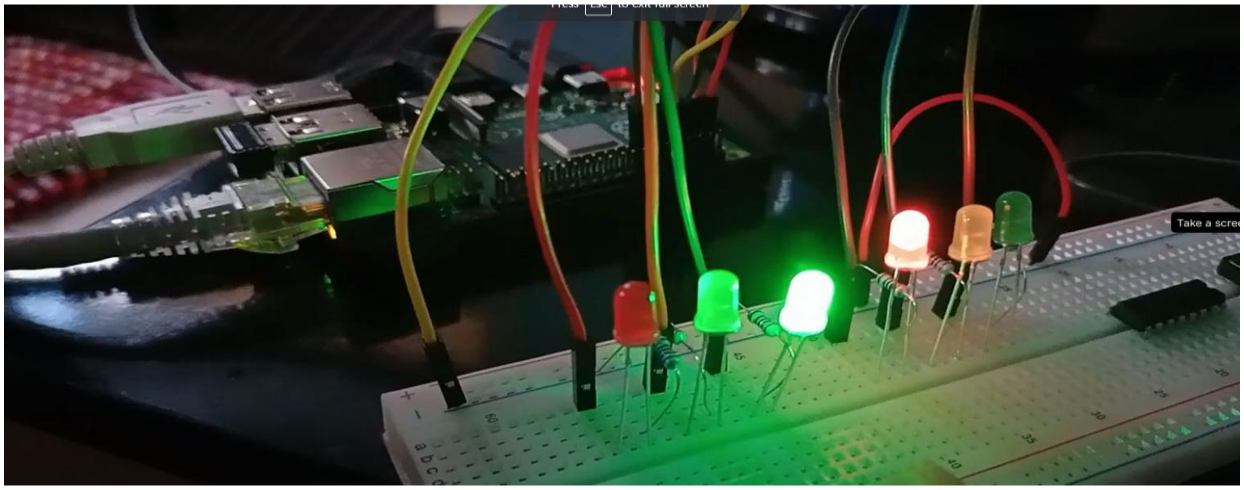
Result:

b) Implement two-way traffic control using Raspberry Pi

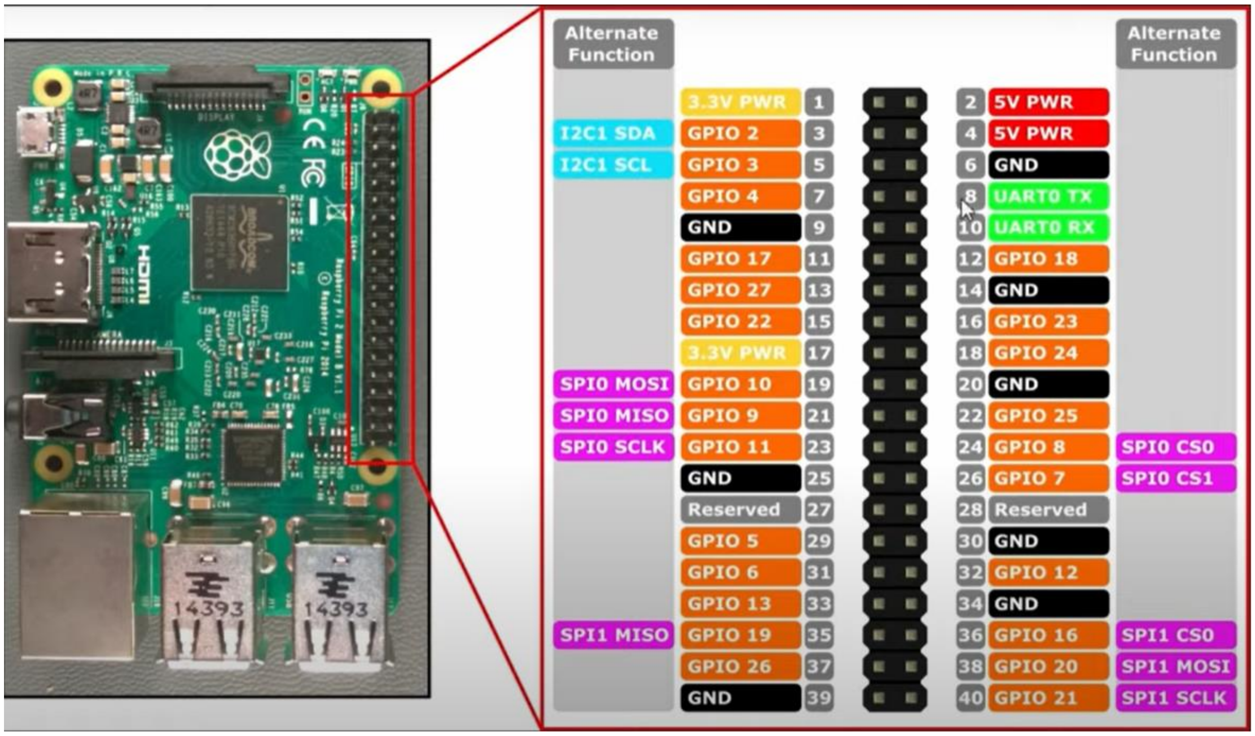
AIM: To implement two-way traffic control using Raspberry Pi

Components Required: Raspberry Pi, LED'S, Bread board, Jumper Wires

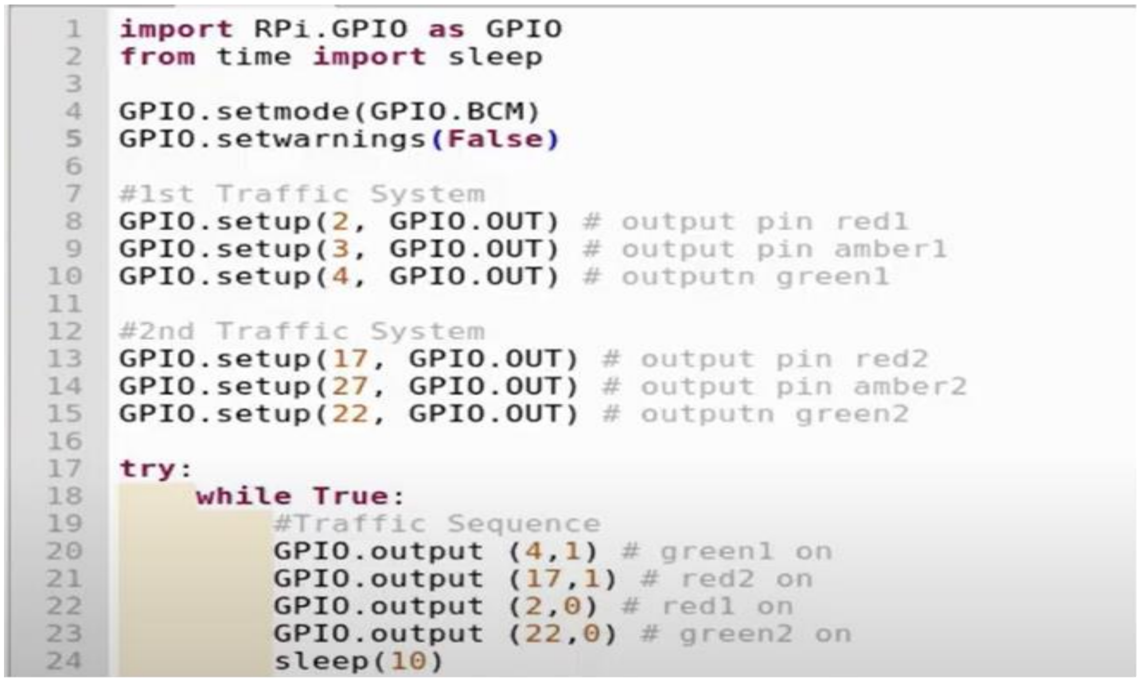
Circuit Diagram:



Raspberry Pi Board:



CODE:



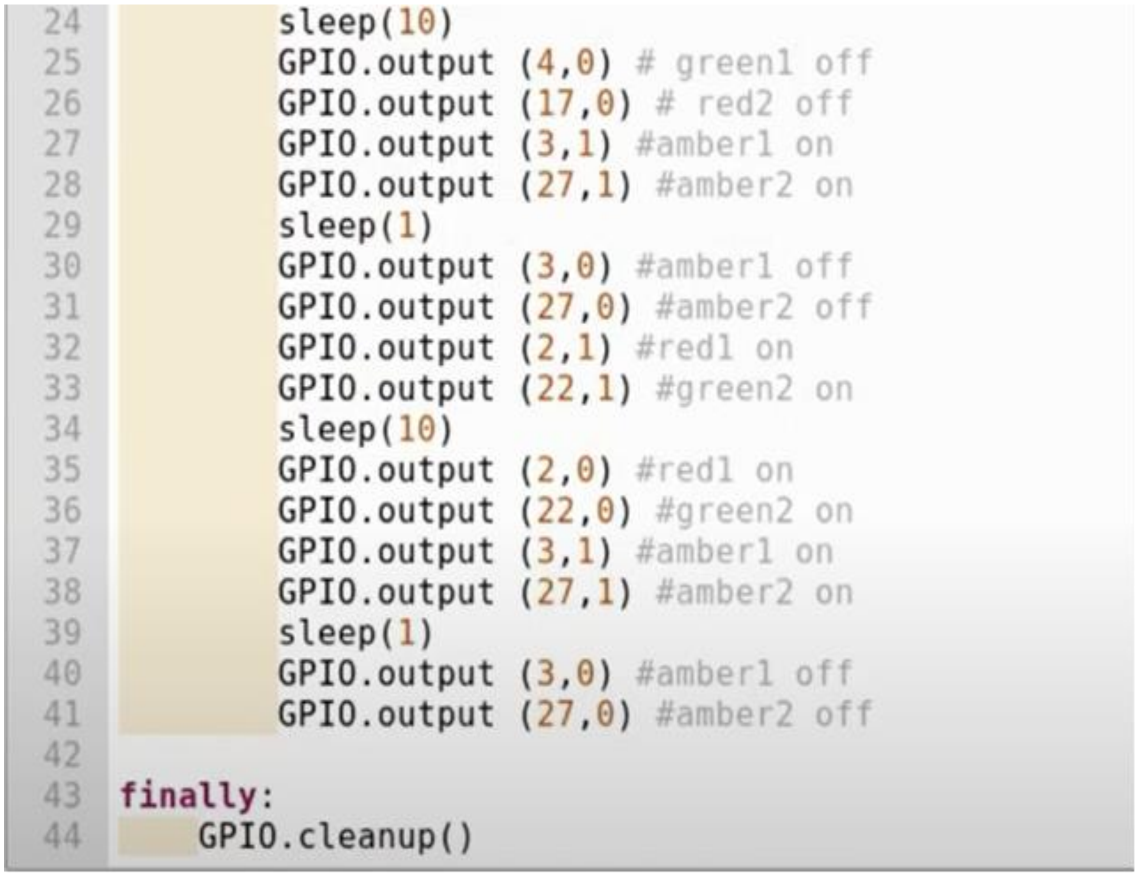
```python
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#1st Traffic System
GPIO.setup(2, GPIO.OUT) # output pin red1
GPIO.setup(3, GPIO.OUT) # output pin amber1
GPIO.setup(4, GPIO.OUT) # outputn green1

#2nd Traffic System
GPIO.setup(17, GPIO.OUT) # output pin red2
GPIO.setup(27, GPIO.OUT) # output pin amber2
GPIO.setup(22, GPIO.OUT) # outputn green2

try:
    while True:
        #Traffic Sequence
        GPIO.output (4,1) # green1 on
        GPIO.output (17,1) # red2 on
        GPIO.output (2,0) # red1 on
        GPIO.output (22,0) # green2 on
        sleep(10)
```



```python
        sleep(10)
        GPIO.output (4,0) # green1 off
        GPIO.output (17,0) # red2 off
        GPIO.output (3,1) #amber1 on
        GPIO.output (27,1) #amber2 on
        sleep(1)
        GPIO.output (3,0) #amber1 off
        GPIO.output (27,0) #amber2 off
        GPIO.output (2,1) #red1 on
        GPIO.output (22,1) #green2 on
        sleep(10)
        GPIO.output (2,0) #red1 on
        GPIO.output (22,0) #green2 on
        GPIO.output (3,1) #amber1 on
        GPIO.output (27,1) #amber2 on
        sleep(1)
        GPIO.output (3,0) #amber1 off
        GPIO.output (27,0) #amber2 off

finally:
    GPIO.cleanup()
```

Procedure:

1.Make the connections as per circuit diagram.

2.Open python & type the program.

3.Run the program & verify the output.

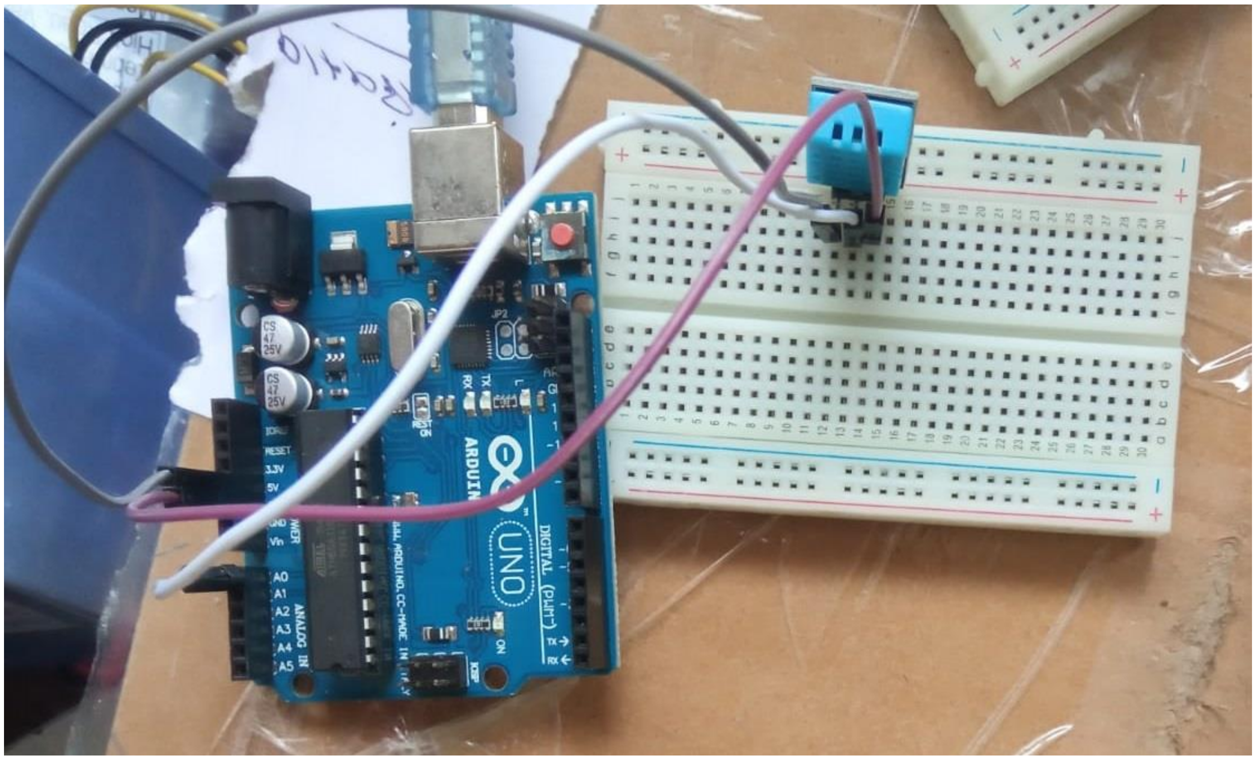
Result:

4. a) Interface DHT11 sensor with Arduino and write a program to print temperature and humidity readings.

AIM: To Interface DHT11 sensor with Arduino and write a program to print temperature and humidity readings

Components Required: Arduino Uno, DHT-11 Sensor , Bread board, Jumper Wires.

Circuit Diagram:



CODE:

```
#include <dht.h>
#define dht_apin A0
dht DHT;
```

```
void setup ()
{
  Serial.begin(9600);
  delay(500);
  Serial.println('DHT11 HUMIDITY& Temperature sensor\n\n');
  delay(1000);
}
void loop()
{
  DHT.read11(dht_apin);
Serial.print("current Humidity=");
Serial.print(DHT.humidity);
Serial.print('%');
Serial.print("temperature=");
Serial.print(DHT.temperature);
Serial.println('C');
delay(5000);
}
```

Procedure:

1.Make the connections as per circuit diagram

2.Add DHT sensor libraries to Arduino Ide

3.Open Arduino Ide & type the program

4.Compile the program & upload it to Arduino uno board.

5.Verify the output.

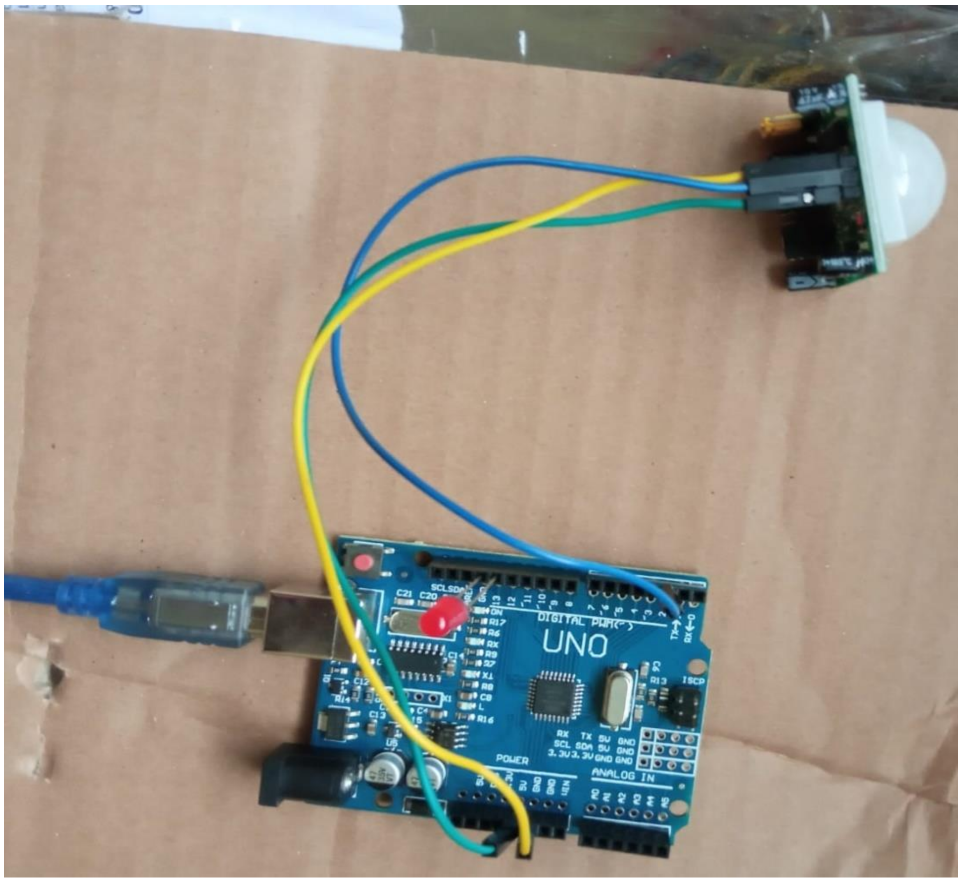
Result:

b) Interface PIR sensor with Arduino and write a program to turn ON LED at sensor detection

AIM: To Interface PIR sensor with Arduino and write a program to turn ON LED at sensor detection

Components Required: PIR Sensor,Arduion Uno ,Bread board, Jumper Wires

Circuit Diagram:



CODE:

```
// Define PIR sensor pin
int pirSensor = 2; // You can connect the PIR sensor to digital pin 2 of Arduino

// Define LED pin
int ledPin = 13; // Built-in LED on most Arduino boards is connected to digital pin 13

void setup() {
  // Initialize the PIR sensor pin as an input
  pinMode(pirSensor, INPUT);

  // Initialize the LED pin as an output
  pinMode(ledPin, OUTPUT);

  // Serial communication for debugging
  Serial.begin(9600);
}

void loop() {
  // Read the PIR sensor value
  int motionSensorValue = digitalRead(pirSensor);

  // Check if motion is detected
  if (motionSensorValue == HIGH) {
    // Motion detected, turn on the LED
    digitalWrite(ledPin, HIGH);
    Serial.println("Motion Detected!");
    delay(1000); // Delay for 1 second to avoid rapid toggling of the LED
  } else {
    // No motion, turn off the LED
    digitalWrite(ledPin, LOW);
    Serial.println("No Motion");
    delay(1000); // Delay for 1 second
  }
}
```

Procedure:

1.Make the connections as per circuit diagram

2.Add PIR sensor libraries to Arduino Ide.

3.Open Arduino Ide & type the program

4.Compile the program & upload it to Arduino Uno board.

5.Verify the output.

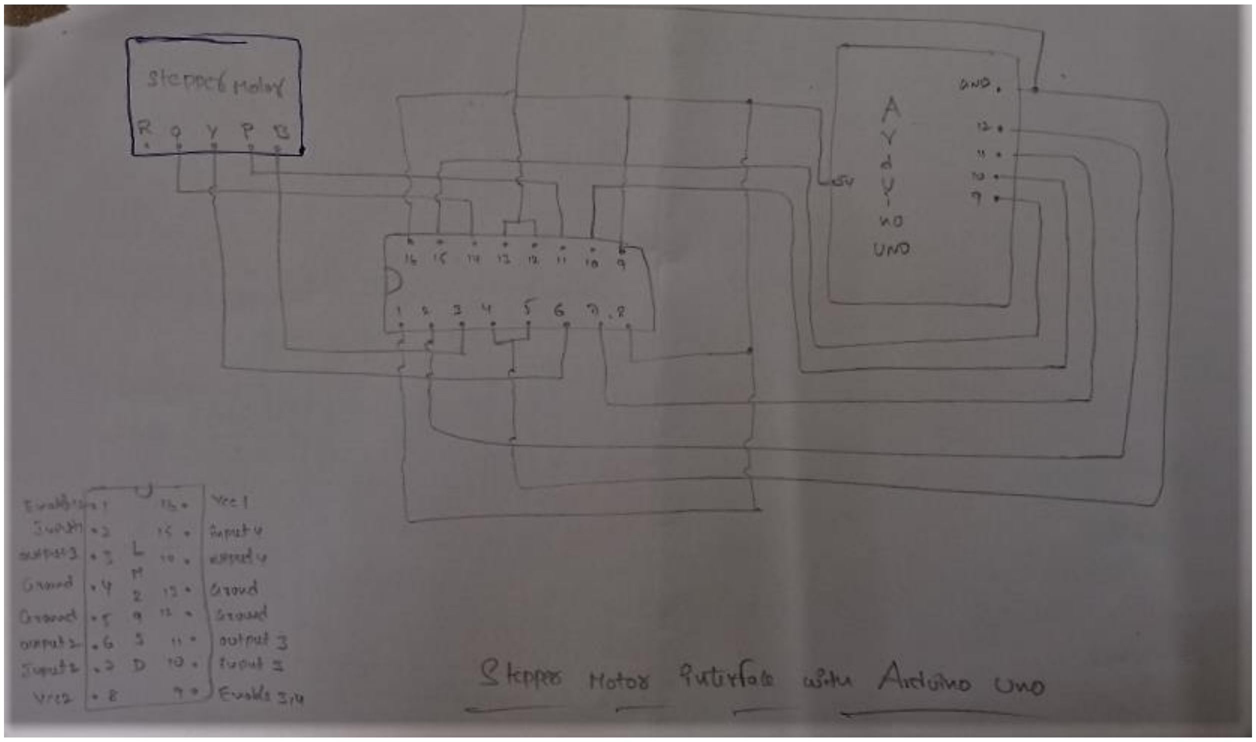
Result:

5. Interface Stepper motor with Arduino and write a program to control stepper motor

AIM: To Interface Stepper motor with Arduino and write a program to control stepper motor

Components Required: Arduino Uno, Stepper motor Bread board, Jumper Wires.

Circuit Diagram:



CODE:

```
/* Stepper Motor Control */


#include <Stepper.h>
const int stepsPerRevolution = 2000;
// change this to fit the number of steps per revolution
// for your motor
// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 9, 10, 11, 12);

void setup() {
  // set the speed at 60 rpm:
  myStepper.setSpeed(5);
  // initialize the serial port:
  Serial.begin(9600);
}

void loop() {
  // step one revolution in one direction:
  Serial.println("clockwise");
  myStepper.step(stepsPerRevolution);
  delay(500);
  // step one revolution in the other direction:
  Serial.println("counterclockwise");
  myStepper.step(-stepsPerRevolution);
  delay(500);
}
```

Procedure:

1.Make the connections as per circuit diagram

2.Open Arduino Ide & type the program

3.Compile the program & upload it to Arduino Uno board.

4.Verify the output.

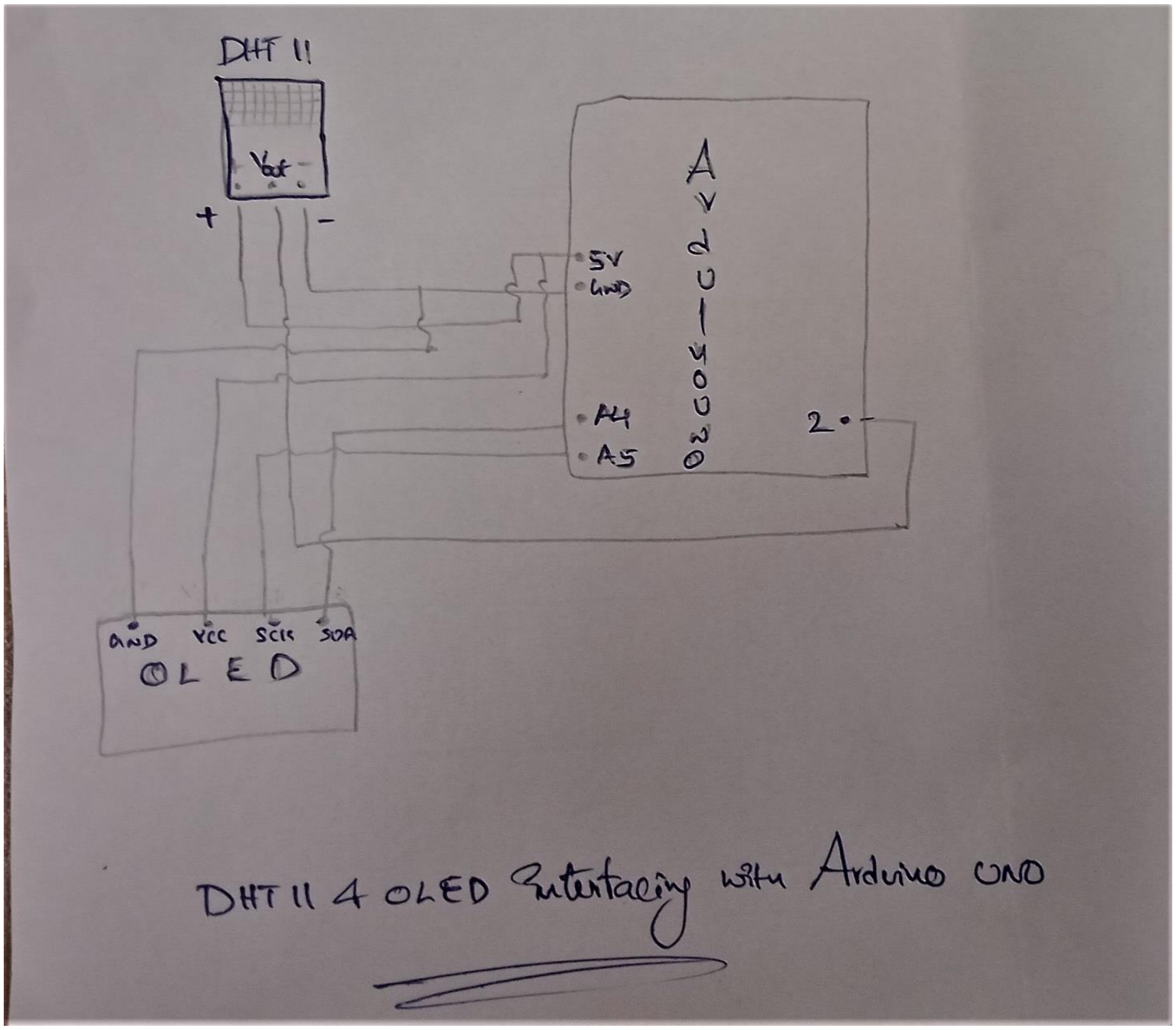
Result:

6. **Interface OLED with Arduino and write a program to print temperature and humidity readings on it.**

AIM: To Interface OLED & DHT-11 sensor with Arduino and write a program to print temperature and humidity readings on it

Components Required: Arduino Uno, DHT-11 Sensor, OLED screen, Bread board, Jumper Wires.

Circuit Diagram:



CODE:

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH1106.h>
#include <Adafruit_Sensor.h>
#include "DHT.h"



#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define OLED_RESET 1
Adafruit_SH1106 display(OLED_RESET);

#define DHTPIN 2     // Digital pin connected to the DHT sensor

// Uncomment the type of sensor in use:
#define DHTTYPE    DHT11     // DHT 11
//#define DHTTYPE    DHT22     // DHT 22 (AM2302)
//#define DHTTYPE    DHT21     // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
  display.begin(SH1106_SWITCHCAPVCC, 0x3C);
  delay(2000);
  display.clearDisplay();
  display.setTextColor(WHITE);
}

void loop() {
  delay(5000);

  //read temperature and humidity
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
  }

  //clear display
  display.clearDisplay();

  // display temperature
  display.setTextSize(1);
  display.setCursor(0,0);
  display.print("Temperature: ");
  display.setTextSize(2);
  display.setCursor(0,10);
  display.print(t);
  display.print(" ");
  display.setTextSize(1);
  display.cp437(true);
  display.write(167);
  display.setTextSize(2);
  display.print("C");

  // display humidity
  display.setTextSize(1);
  display.setCursor(0, 35);
  display.print("Humidity: ");
  display.setTextSize(2);
  display.setCursor(0, 45);
  display.print(h);
  display.print(" %");

  display.display();
}
```

Procedure:

1.Make the connections as per the circuit diagram

2.Open Arduino IDE software & type the program.

3.Add required libraries to Arduino Ide

4. Compile & upload the program to Arduino Uno

5.Verify the output.

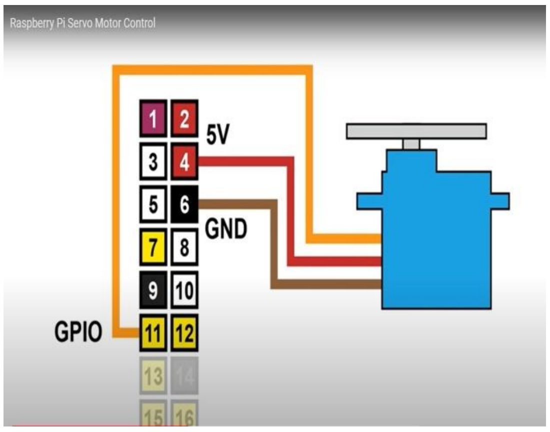
Result:

EXP .NO. 7. To interface Servo motor with Raspberry Pi and write a program to run motor 180º in 10 steps and turning back to 90º for 2 seconds and Turning back to 0 degrees.

AIM : To understand the Raspberry Pi , Servo Motor Interface and how to control a simple Servo Motor using Raspberry Pi.

Components Required: Servo Motor ,Raspberry pi ,Jumper wires

Circuit:



CODE:

```
# Import libraries
import RPi.GPIO as GPIO
import time
# Set GPIO numbering mode
GPIO.setmode(GPIO.BOARD)
# Set pin 11 as an output, and set servo1 as pin 11 as PWM
GPIO.setup(11,GPIO.OUT)
servo1 = GPIO.PWM(11,50) # Note 11 is pin, 50 = 50Hz pulse
#start PWM running, but with value of 0 (pulse off)
servo1.start(0)
print ("Waiting for 2 seconds")
time.sleep(2)
#Let's move the servo!
print ("Rotating 180 degrees in 10 steps")
# Define variable duty
```

```
duty = 2
# Loop for duty values from 2 to 12 (0 to 180 degrees)
while duty <= 12:
    servo1.ChangeDutyCycle(duty)
    time.sleep(1)
    duty = duty + 1
# Wait a couple of seconds
time.sleep(2)
# Turn back to 90 degrees
print ("Turning back to 90 degrees for 2 seconds")
servo1.ChangeDutyCycle(7)
time.sleep(2)
#turn back to 0 degrees
print ("Turning back to 0 degrees")
servo1.ChangeDutyCycle(2)
time.sleep(0.5)
servo1.ChangeDutyCycle(0)
#Clean things up at the end
servo1.stop()
GPIO.cleanup()
```

Procedure :

1. Make the connections as per the circuit diagram
2. Connect Raspberry pi board to computer monitor & switch on supply.
3. After giving supply, Raspbian OS home screen will open on monitor.
4. Click on **Raspberry** icon select **programming** option & click on **python Ide.**
5. New window will open, type program & click on save & Run the program.
6. If there are no errors program will execute directly otherwise errors will display on the screen.
7. Observe the output .

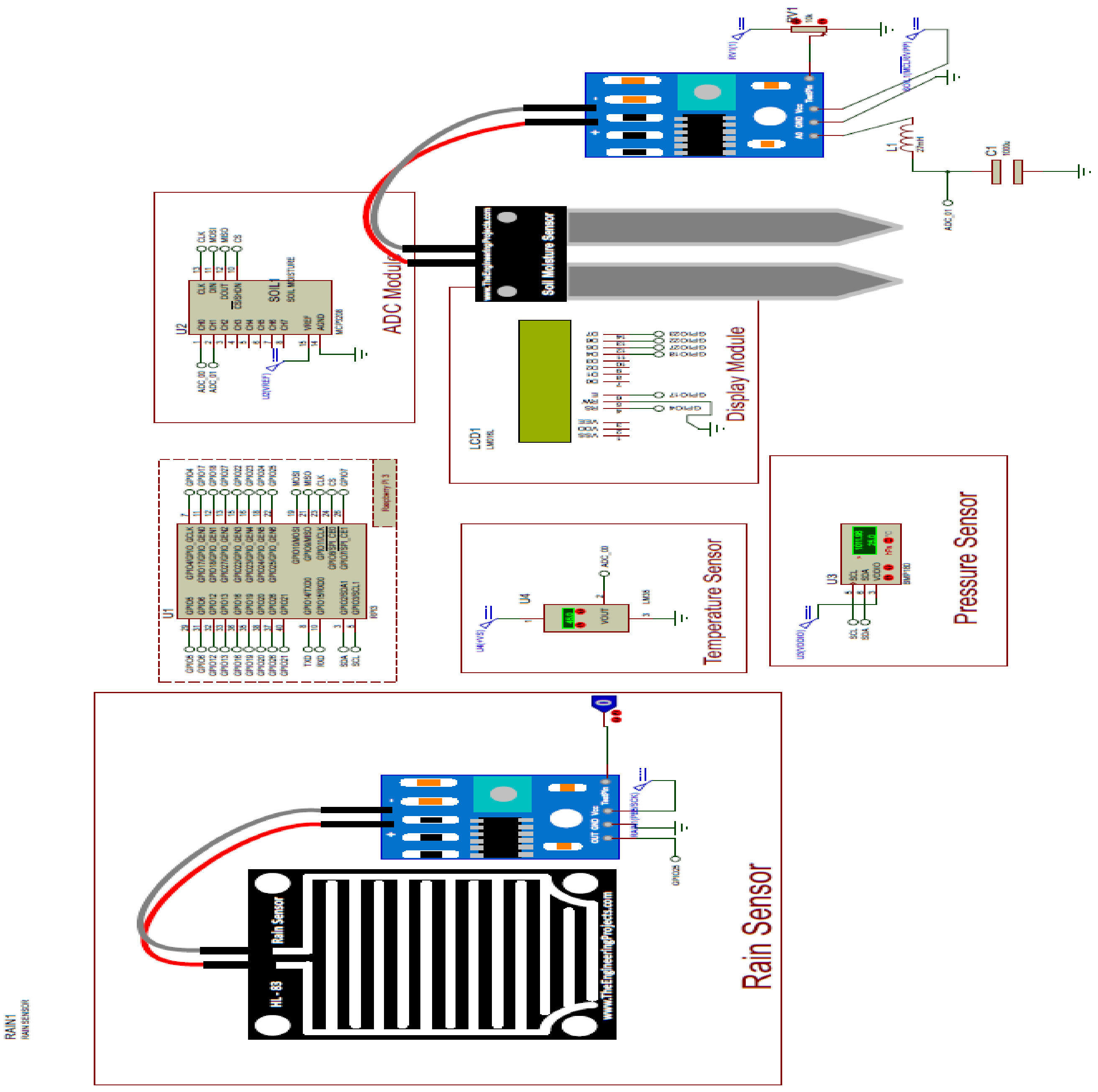
RESULT :

EXP .NO. 8. Write a program for weather monitoring station and handling temperature & humidity values on cloud platform

AIM : To Write a program for weather monitoring station and handling temperature & humidity values on cloud platform

Resources Required: Proteus software, Computer

Circuit diagram:



CODE:

```
# Modules
from goto import *
import time
import var
import pio
import resource
import spidev
import RPi.GPIO as GPIO
import urllib.request
import requests
import smbus
from ctypes import c_short

# Peripheral Configuration Code (do not edit)
#---CONFIG_BEGIN---
import cpu
import FileStore
import VFP
import Ports

def peripheral_setup () :
# Peripheral Constructors
 pio.cpu=cpu.CPU ()
 pio.storage=FileStore.FileStore ()
 pio.server=VFP.VfpServer ()
 pio.uart=Ports.UART ()
 pio.storage.begin ()
 pio.server.begin (0)
# Install interrupt handlers

def peripheral_loop () :
 pass

#---CONFIG_END---

# Open SPI bus
spi = spidev.SpiDev()
spi.open(0,0)

# Define GPIO to LCD mapping
LCD_RS = 4
LCD_E  = 17
LCD_D4 = 18
LCD_D5 = 27
LCD_D6 = 22
LCD_D7 = 23
Relay_pin= 24
Rain_sensor = 25
# Define sensor channels
temp_channel  = 0
Moisture_channel =1

'''
```

```
define pin for lcd
'''
# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005
delay = 1



GPIO.setup(LCD_E, GPIO.OUT)  # E
GPIO.setup(LCD_RS, GPIO.OUT) # RS
GPIO.setup(LCD_D4, GPIO.OUT) # DB4
GPIO.setup(LCD_D5, GPIO.OUT) # DB5
GPIO.setup(LCD_D6, GPIO.OUT) # DB6
GPIO.setup(LCD_D7, GPIO.OUT) # DB7
GPIO.setup(Relay_pin, GPIO.OUT) # Motor_1
GPIO.setup(Rain_sensor, GPIO.IN)
# Define some device constants
LCD_WIDTH = 16    # Maximum characters per line
LCD_CHR = True
LCD_CMD = False
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

'''
Function Name :lcd_init()
Function Description : this function is used to initialized lcd by sending the different
commands
'''
def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)
'''
Function Name :lcd_byte(bits ,mode)
Fuction Name :the main purpose of this function to convert the byte data into bit and send to
lcd port
'''
def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #        False for command

  GPIO.output(LCD_RS, mode) # RS

  # High bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
  if bits&0x80==0x80:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()

  # Low bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()
'''
Function Name : lcd_toggle_enable()
Function Description:basically this is used to toggle Enable pin
'''
def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)
'''
Function Name :lcd_string(message,line)
Function  Description :print the data on lcd
'''
def lcd_string(message,line):
  # Send string to display

  message = message.ljust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)
```

```
# Function to read SPI data from MCP3008 chip
# Channel must be an integer 0-7
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data


# Function to calculate temperature from
# TMP36 data, rounded to specified
# number of decimal places.
def ConvertTemp(data,places):

  # ADC Value
  # (approx)  Temp  Volts
  #    0      -50    0.00
  #   78      -25    0.25
  #  155        0    0.50
  #  233       25    0.75
  #  310       50    1.00
  #  465      100    1.50
  #  775      200    2.50
  # 1023      280    3.30

  temp = ((data * 330)/float(1023))
  temp = round(temp,places)
  return temp

def thingspeak_post(temp,moisture_level,pressure,rain_data):
    URl='https://api.thingspeak.com/update?api_key='
    #Enter Your Private Key here
    KEY='TTVU38XV7HU2I6GR'

HEADER='&field1={}&field2={}&field3={}&field4={}'.format(temp,moisture_level,pressu
re,rain_data)
    NEW_URL=URl+KEY+HEADER
    print(NEW_URL)
    data=urllib.request.urlopen(NEW_URL)
    print(data)

DEVICE = 0x77 # Default device I2C address

#bus = smbus.SMBus(0)  # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def convertToString(data):
  # Simple function to convert binary data into
  # a string
  return str((data[1] + (256 * data[0])) / 1.2)

def getShort(data, index):
  # return two bytes from data as a signed 16-bit value
  return c_short((data[index] << 8) + data[index + 1]).value
```

```
def getUshort(data, index):
  # return two bytes from data as an unsigned 16-bit value
  return (data[index] << 8) + data[index + 1]

def readBmp180Id(addr=DEVICE):
  # Chip ID Register Address
  REG_ID     = 0xD0
  (chip_id, chip_version) = bus.read_i2c_block_data(addr, REG_ID, 2)
  return (chip_id, chip_version)

def readBmp180(addr=DEVICE):
  # Register Addresses
  REG_CALIB  = 0xAA
  REG_MEAS   = 0xF4
  REG_MSB    = 0xF6
  REG_LSB    = 0xF7
  # Control Register Address
  CRV_TEMP   = 0x2E
  CRV_PRES   = 0x34
  # Oversample setting
  OVERSAMPLE = 3    # 0 - 3

  # Read calibration data
  # Read calibration data from EEPROM
  cal = bus.read_i2c_block_data(addr, REG_CALIB, 22)

  # Convert byte data to word values
  AC1 = getShort(cal, 0)
  AC2 = getShort(cal, 2)
  AC3 = getShort(cal, 4)
  AC4 = getUshort(cal, 6)
  AC5 = getUshort(cal, 8)
  AC6 = getUshort(cal, 10)
  B1  = getShort(cal, 12)
  B2  = getShort(cal, 14)
  MB  = getShort(cal, 16)
  MC  = getShort(cal, 18)
  MD  = getShort(cal, 20)

  # Read temperature
  bus.write_byte_data(addr, REG_MEAS, CRV_TEMP)
  time.sleep(0.005)
  (msb, lsb) = bus.read_i2c_block_data(addr, REG_MSB, 2)
  UT = (msb << 8) + lsb

  # Read pressure
  bus.write_byte_data(addr, REG_MEAS, CRV_PRES + (OVERSAMPLE << 6))
  time.sleep(0.04)
  (msb, lsb, xsb) = bus.read_i2c_block_data(addr, REG_MSB, 3)
  UP = ((msb << 16) + (lsb << 8) + xsb) >> (8 - OVERSAMPLE)

  # Refine temperature
  X1 = ((UT - AC6) * AC5) >> 15
  X2 = (MC << 11) / (X1 + MD)
  B5 = X1 + X2
  #temperature = int(B5 + 8) >> 4

  # Refine pressure
  B6  = B5 - 4000
  B62 = int(B6 * B6) >> 12
  X1  = (B2 * B62) >> 11
  X2  = int(AC2 * B6) >> 11
  X3  = X1 + X2
  B3  = (((AC1 * 4 + X3) << OVERSAMPLE) + 2) >> 2

  X1 = int(AC3 * B6) >> 13
  X2 = (B1 * B62) >> 16
  X3 = ((X1 + X2) + 2) >> 2
  B4 = (AC4 * (X3 + 32768)) >> 15
  B7 = (UP - B3) * (50000 >> OVERSAMPLE)

  P = (B7 * 2) / B4

  X1 = (int(P) >> 8) * (int(P) >> 8)
  X1 = (X1 * 3038) >> 16
  X2 = int(-7357 * P) >> 16
  pressure = int(P + ((X1 + X2 + 3791) >> 4))

  return (pressure/100.0)


# Define delay between readings
delay = 5
lcd_init()
lcd_string("welcome ",LCD_LINE_1)
time.sleep(1)
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
lcd_string("Weather Monitoring ",LCD_LINE_1)
lcd_string("System ",LCD_LINE_2)
time.sleep(1)
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
# Main function
def main () :
# Setup
 peripheral_setup()
 peripheral_loop()
 #Motor Status
 motor_status = 0
# Infinite loop
 while 1 :

  temp_level = ReadChannel(temp_channel)
  temp       = ConvertTemp(temp_level,2)

  # Print out results
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Temperature  ",LCD_LINE_1)
  lcd_string(str(temp),LCD_LINE_2)
  time.sleep(0.1)
```

```
moisture_level = ReadChannel(Moisture_channel)
# Print out results
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
lcd_string("Moisture Level  ",LCD_LINE_1)
lcd_string(str(moisture_level),LCD_LINE_2)
time.sleep(0.1)

#Rain Sesnor Data
rain_data = GPIO.input(Rain_sensor)

#Pressure Sensor Data
(chip_id, chip_version) = readBmp180Id()
pressure=readBmp180()
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
lcd_string("Pressure Value ",LCD_LINE_1)
lcd_string(str(pressure),LCD_LINE_2)
time.sleep(0.1)

#Send data on thing speak server
thingspeak_post(temp,moisture_level,pressure,rain_data)
```

Procedure:

1.Open Proteus software & design the circuit as per the circuit diagram.

2.Type the program on source code & add all required libraries.

3. Create account on www.thingsspeak.com , create one new channel & enter data as per circuit.

4.Simulate the circuit & verify the output.

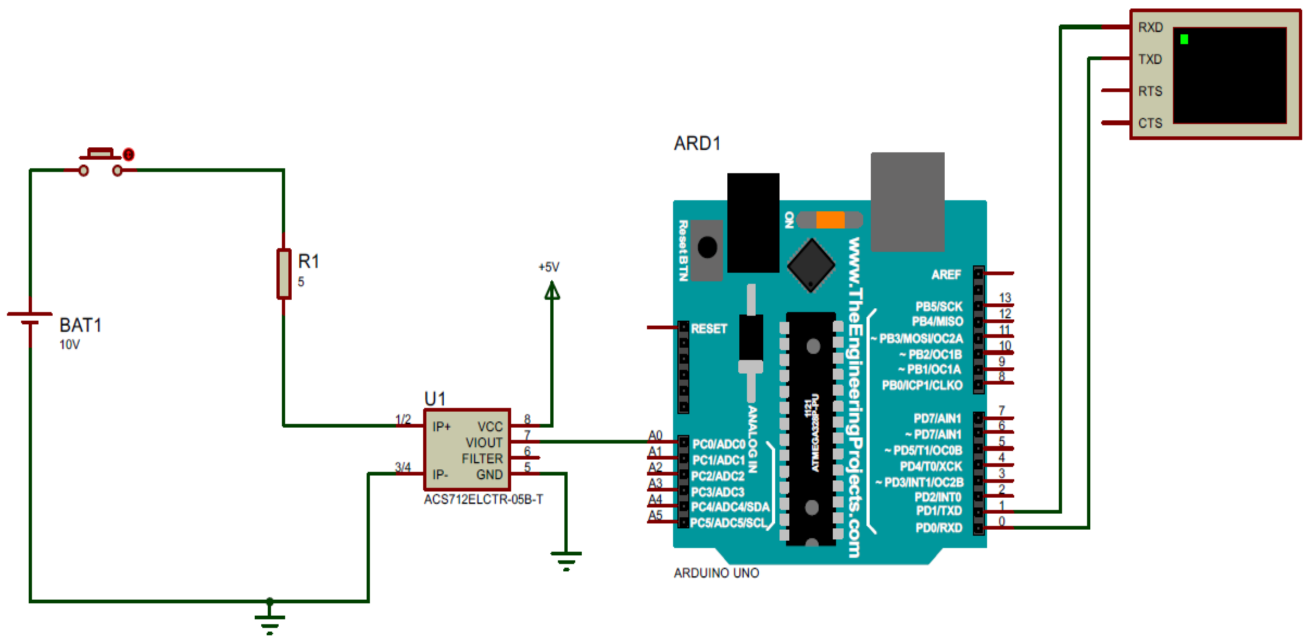
Result:

EXP .NO. 9. Design of digital dc voltmeter and ammeter using Arduino uno.

AIM : To design of digital dc voltmeter and ammeter using Arduino uno.

Resources Required: Proteus software,Computer

Circuit:



CODE:

```
// Variables for Measured Voltage and Calculated Current
double Vout = 0;
double Current = 0;

// Constants for Scale Factor
// Use one that matches your version of ACS712

const double scale_factor = 0.185; // 5A
//const double scale_factor = 0.1; // 20A
//const double scale_factor = 0.066; // 30A

// Constants for A/D converter resolution
// Arduino has 10-bit ADC, so 1024 possible values
// Reference voltage is 5V if not using AREF external reference
// Zero point is half of Reference Voltage
```

```
const double vRef = 5.00;
const double resConvert = 1024;
double resADC = vRef/resConvert;
double zeroPoint = vRef/2;


void setup(){
  Serial.begin(9600);
}

void loop(){

  // Vout is read 1000 Times for precision
  for(int i = 0; i < 1000; i++) {
    Vout = (Vout + (resADC * analogRead(A0)));
    delay(1);
  }

  // Get Vout in mv
  Vout = Vout /1000;

  // Convert Vout into Current using Scale Factor
  Current = (Vout - zeroPoint)/ scale_factor;

  // Print Vout and Current to two Current = ");

  Serial.print("Vout = ");
  Serial.print(Vout,2);
  Serial.print(" Volts");
  Serial.print("\t Current = ");
  Serial.print(Current,2);
  Serial.println(" Amps");

  delay(1000);
}
```

Procedure:

1.Open Proteus software & design the circuit.

2.Open Arduino IDE & type the program.

3.Compile & upload the program in to Proteus.

4.Simulate the circuit & verify the output.

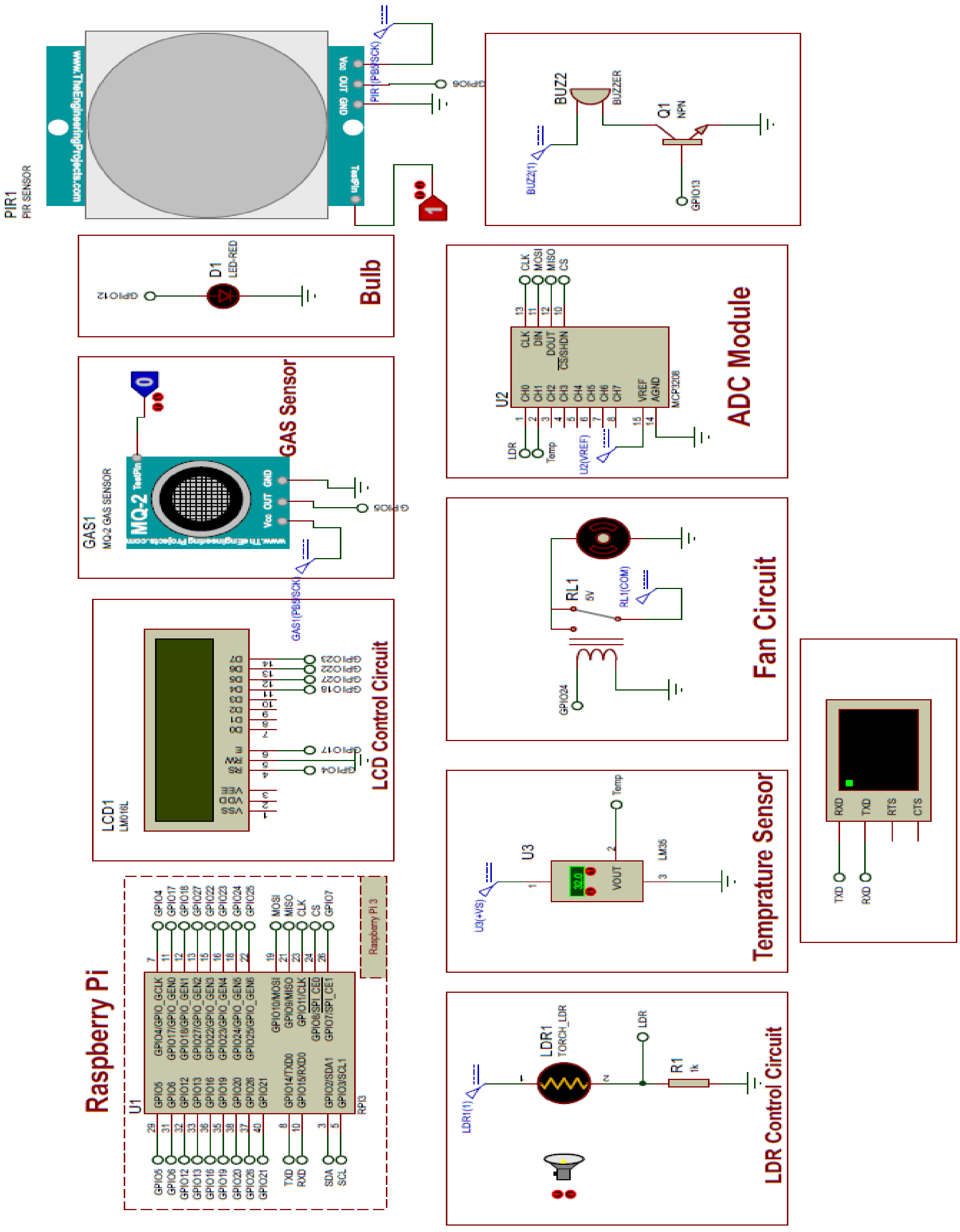
RESULT :

EXP .NO. 10. Design home automation using Raspberry Pi

AIM : To design home automation using Raspberry Pi .

Resources Required: Proteus software, Computer

Circuit:



CODE:

```
import spidev
import time
import RPi.GPIO as GPIO
import pio
import Ports

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

pio.uart=Ports.UART () # Define serial port

# Open SPI bus
spi = spidev.SpiDev()
spi.open(0,0)

# Define GPIO to LCD mapping
LCD_RS = 7
LCD_E  = 11
LCD_D4 = 12
LCD_D5 = 13
LCD_D6 = 15
LCD_D7 = 16
bulb_pin =  32
motor_pin =  18
pir_pin = 31
gas_pin = 29
buzzer_pin =33
# Define sensor channels
ldr_channel = 0
temp_channel  = 1
'''
define pin for lcd
'''
# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005
delay = 1



GPIO.setup(LCD_E, GPIO.OUT)  # E
GPIO.setup(LCD_RS, GPIO.OUT) # RS
GPIO.setup(LCD_D4, GPIO.OUT) # DB4
GPIO.setup(LCD_D5, GPIO.OUT) # DB5
GPIO.setup(LCD_D6, GPIO.OUT) # DB6
GPIO.setup(LCD_D7, GPIO.OUT) # DB7
GPIO.setup(bulb_pin, GPIO.OUT) # DB7
GPIO.setup(motor_pin, GPIO.OUT) # DB7
GPIO.setup(buzzer_pin, GPIO.OUT) # DB7
GPIO.setup(gas_pin, GPIO.IN) # DB7
GPIO.setup(pir_pin, GPIO.IN) # DB7
# Define some device constants
LCD_WIDTH = 16    # Maximum characters per line
LCD_CHR = True
LCD_CMD = False
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line




'''
Function Name :lcd_init()
Function Description : this function is used to initialized lcd by sending the different
commands
'''
def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)
'''
Function Name :lcd_byte(bits ,mode)
Fuction Name :the main purpose of this function to convert the byte data into bit and send to
lcd port
'''
def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #        False for command

  GPIO.output(LCD_RS, mode) # RS

  # High bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
  if bits&0x80==0x80:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()

  # Low bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()
'''
Function Name : lcd_toggle_enable()
Function Description:basically this is used to toggle Enable pin
'''
def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)
'''
Function Name :lcd_string(message,line)
Function  Description :print the data on lcd
'''
def lcd_string(message,line):
  # Send string to display

  message = message.ljust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)



# Function to read SPI data from MCP3008 chip
# Channel must be an integer 0-7
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data


 # Function to calculate temperature from
# TMP36 data, rounded to specified
# number of decimal places.
def ConvertTemp(data,places):

  # ADC Value
  # (approx)  Temp  Volts
  #    0      -50    0.00
  #   78      -25    0.25
  #  155        0    0.50
  #  233       25    0.75
  #  310       50    1.00
  #  465      100    1.50
  #  775      200    2.50
  # 1023      280    3.30

  temp = ((data * 330)/float(1023))
  temp = round(temp,places)
  return temp

# Define delay between readings
delay = 5
lcd_init()
lcd_string("welcome ",LCD_LINE_1)
time.sleep(1)
while 1:
  gas_data =  GPIO.input(gas_pin)
  if(gas_data == True):
   lcd_byte(0x01,LCD_CMD) # 000001 Clear display
   lcd_string("Fire Detected",LCD_LINE_1)
   lcd_string("Buzzer On",LCD_LINE_2)
   GPIO.output(bulb_pin, False)
   GPIO.output(motor_pin, False)
   GPIO.output(buzzer_pin, True)
   time.sleep(0.5)
   while(1):
```

```
   lcd_byte(0x01,LCD_CMD) # 000001 Clear display
   lcd_string("Sending Message",LCD_LINE_1)
   pio.uart.println("AT")
   pio.uart.println("AT+CMGF=1")
   pio.uart.println("AT+CMGS=\"+919922512017\"\r")
   pio.uart.println("Fire Detected")
pir_data =  GPIO.input(pir_pin)
if(pir_data  ==  True):
 light_level = ReadChannel(ldr_channel)
 time.sleep(0.2)
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Person Detected  ",LCD_LINE_1)
 time.sleep(0.5)
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Automatic Light and  ",LCD_LINE_1)
 lcd_string("Fan System Active ",LCD_LINE_2)
 time.sleep(0.5)
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Light Intencsty  ",LCD_LINE_1)
 lcd_string(str(light_level),LCD_LINE_2)
 time.sleep(0.5)
 if(light_level < 100 ):
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Bulb ON",LCD_LINE_2)
  GPIO.output(bulb_pin, True)
  time.sleep(0.5)
 else:
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Bulb OFF",LCD_LINE_2)
  GPIO.output(bulb_pin, False)
  time.sleep(0.5)
 # Print out results
 temp_level = ReadChannel(temp_channel)
 time.sleep(0.5)
 temperature      = ConvertTemp(temp_level,2)
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Temperature  ",LCD_LINE_1)
 lcd_string(str(temperature),LCD_LINE_2)
 time.sleep(0.5)
 if(temperature > 30):
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Fan ON  ",LCD_LINE_1)
  GPIO.output(motor_pin, True)
  time.sleep(0.5)
 else:
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Fan Off  ",LCD_LINE_1)
  GPIO.output(motor_pin, False)
```

```
    time.sleep(0.5)
  else:
    lcd_byte(0x01,LCD_CMD) # 000001 Clear display
    lcd_string("Person Not Detected",LCD_LINE_1)
    GPIO.output(motor_pin, False)
    GPIO.output(bulb_pin, False)
    time.sleep(0.5)
```

Procedure:

1.Open Proteus software & design the circuit.

2.On Source code type the program.

3.Add all required libraries

4.Simulate the circuit & verify the output.

RESULT :