# Module 1: Introduction & Architecture

## Objective

To introduce the fundamental concepts of Industrial IoT (IIoT) and understand the architecture and challenges associated with it.

## Components Required

- Arduino Board
- Raspberry Pi

## Theory

What is IIoT?

Industrial IoT (IIoT) refers to the application of IoT technology in industrial sectors and applications. It focuses on the interconnectivity of machines, sensors, and systems within industrial settings to collect, analyze, and utilize data for improving efficiency, productivity, and safety.

Difference between IoT and IIoT

While IoT encompasses all types of connected devices, IIoT is specifically focused on industrial applications like manufacturing, supply chain monitoring, and remote maintenance. IIoT systems are often more robust and must adhere to stringent safety and reliability standards.

Architecture of IIoT

The architecture of IIoT typically consists of:

- Edge Devices: Sensors and actuators that collect and interact with data.
- Network Layer: Communication protocols that enable data transfer.
- Data Processing Layer: Systems that analyze and process the collected data.
- Application Layer: Interfaces where the processed data is utilized for decision-making and automation.



Challenges of IIoT

- Data Security: Ensuring the confidentiality, integrity, and availability of data.
- Scalability: Managing the growth in devices and data.
- Interoperability: Integrating devices and systems from different manufacturers.
- Latency: Reducing delays in data transmission and processing.

## Procedure

1. Set up the Arduino board.
2. Connect the Raspberry Pi to the network.
3. Explore the basic programming environment for both Arduino and Raspberry Pi.

## Connections

| Device | Pin | Function |
|---|---|---|
| Arduino | 7 | Digital Input/Output |
| Raspberry Pi | GPIO 17 | General Purpose I/O |

## Observation

The basic setup of Arduino and Raspberry Pi environments.
Understand the connection between edge devices and the network.

## Result

Successful setup and understanding of the IIoT architecture.

## Module 2: IIoT Components

# Experiment 1:

# Interface MQ3 and MQ4 Sensor with ESP32

### Objective

To interface the MQ2 Sensor with the ESP32 and observe its operation.

### Components Required

- ESP32
- MQ2 Sensor
- Connecting Wires

### Theory

The MQ2 sensor is used to detect smoke and flammable gases. It outputs an analog signal proportional to the concentration of gases in the environment.

### Procedure

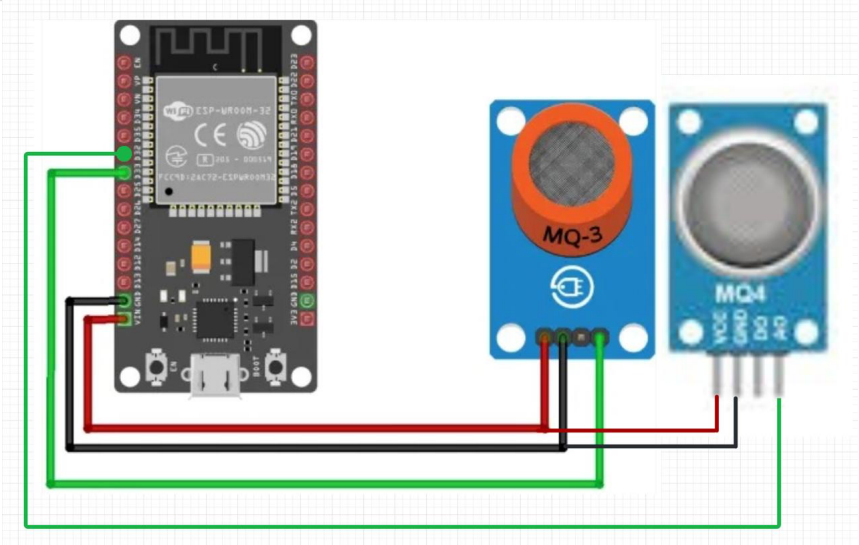
1. Connect the MQ2 Sensor to the ESP32 as per the pin diagram.
2. Upload the appropriate code to the ESP32 to read the MQ3 and MQ4 Sensors data.
3. Observe the sensor's readings on the serial monitor.

### Connections:

| Component | Pin | Function |
|---|---|---|
| MQ3 Sensor | GPIO 34 | Analog Output |
| MQ4 Sensor | GPIO 35 | Analog Output |



### Circuit Diagram



Code

```
#include <WiFi.h>
#include <HTTPClient.h>

// WiFi credentials
const char* ssid = "your_wifi_ssid";  // Replace with your WiFi SSID
const char* password = "your_wifi_password";  // Replace with your WiFi password

// ThingSpeak credentials
const char* server = "http://api.thingspeak.com";
String apiKey = "your_thingspeak_write_api_key";  // Replace with your
ThingSpeak Write API Key
int channelID = your_channel_id;  // Replace with your ThingSpeak Channel ID

// Analog pins for sensors
const int mq3Pin = D33;  // MQ-3 connected to D33
const int mq4Pin = D32;  // MQ-4 connected to D32

void setup() {
 Serial.begin(115200);

 // Connect to WiFi
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
 }
 Serial.println("Connected to WiFi");
}

void loop() {
 // Read analog values from MQ-3 and MQ-4
 int mq3Value = analogRead(mq3Pin);
 int mq4Value = analogRead(mq4Pin);

 Serial.print("MQ-3 Value: ");
 Serial.println(mq3Value);

 Serial.print("MQ-4 Value: ");
 Serial.println(mq4Value);

 // Send data to ThingSpeak
 if (WiFi.status() == WL_CONNECTED) {
  HTTPClient http;
  String url = server + "/update?api_key=" + apiKey + "&field1=" +
String(mq3Value) + "&field2=" + String(mq4Value);
  http.begin(url);
```

```
  int httpCode = http.GET();

  if (httpCode > 0) {
    Serial.println("Data sent to ThingSpeak");
    String response = http.getString();
    Serial.println(response);
  } else {
    Serial.println("Error in sending data to ThingSpeak");
  }

  http.end();
 }

 delay(20000);  // Wait for 20 seconds before sending the next data
}
```

### Observation

Observed the working of the MQ3 and MQ4 Sensors and its response to environmental changes.

### Result

Successfully interfaced the MQ3 and MQ4 Sensor with the ESP32 and obtained the expected readings.

# Embedded Systems & IoT Experiments Manual

## Experiment 2:

## Interface IR Sensor with ESP32

### Objective

To interface the IR Sensor with the ESP32 and observe its operation.

### Components Required

- ESP32
- IR Sensor
- Connecting Wires

### Theory

The IR sensor detects infrared radiation, which is used to sense objects, measure distances, and detect heat signatures.

### Procedure

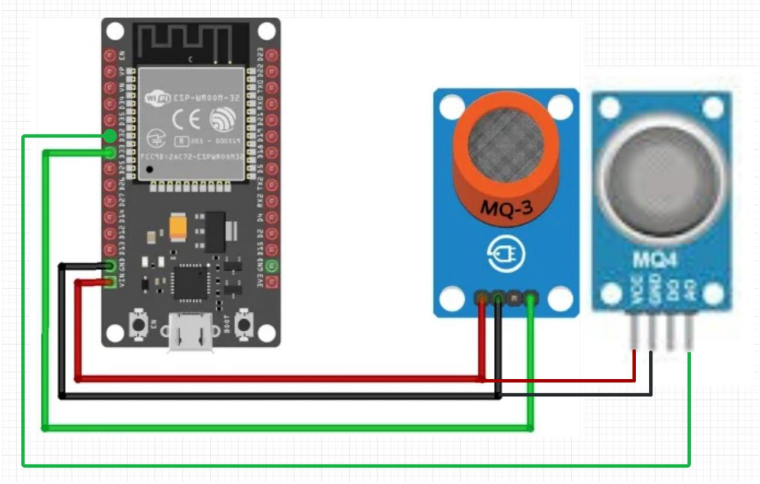
1. Connect the IR Sensor to the ESP32 as per the pin diagram.
2. Upload the appropriate code to the ESP32 to read the IR Sensor data.
3. Observe the sensor's readings on the serial monitor.

### PIN Diagram

| Component | Pin | Function |
|---|---|---|
| IR Sensor | D2 | Digital Output |

### Circuit Diagram





Code

```
import RPi.GPIO as GPIO
import time
import requests

# GPIO setup
SENSOR_PIN = 17  # GPIO pin where the IR sensor is connected

# ThingSpeak Settings
THINGSPEAK_WRITE_API_KEY = "YOUR_THINGSPEAK_WRITE_API_KEY"  # Replace
with your Write API Key
THINGSPEAK_URL = "https://api.thingspeak.com/update"

# Setup GPIO mode
GPIO.setmode(GPIO.BCM)
GPIO.setup(SENSOR_PIN, GPIO.IN)

def read_distance():
  # Read digital input from IR sensor
  if GPIO.input(SENSOR_PIN):
    return 1  # Object detected
  else:
    return 0  # No object detected

def send_data_to_thingspeak(distance):
  # Prepare the data to be sent to ThingSpeak
  payload = {
    'api_key': THINGSPEAK_WRITE_API_KEY,
    'field3': distance  # Sending data to field2
  }

  # Send data to ThingSpeak using HTTP POST
  try:
    response = requests.post(THINGSPEAK_URL, params=payload)
    if response.status_code == 200:
      print(f"Sent data: {distance} to ThingSpeak")
    else:
      print(f"Failed to send data: HTTP {response.status_code}")
  except Exception as e:
    print(f"Failed to send data: {e}")

try:
  while True:
    # Read the sensor
    distance = read_distance()
    print(f"IR Sensor Distance: {distance}")

    # Send data to ThingSpeak
    send_data_to_thingspeak(distance)
```

```
    # Wait 15 seconds between readings (to respect ThingSpeak rate limit)
    time.sleep(15)

except KeyboardInterrupt:
  print("Script interrupted by user.")

finally:
  # Clean up GPIO
  GPIO.cleanup()
```

### Observation

Observed the working of the IR Sensor and its response to environmental changes.

### Result

Successfully interfaced the IR Sensor with the ESP32 and obtained the expected readings.

# Embedded Systems & IoT Experiments Manual

## Experiment 3:

## Interface Ultra Sonic Sensor with Raspberry Pi

### Objective

To interface the Ultrasonic Sensor with the ESP32 and observe its operation.

### Components Required

- ESP32
- Ultrasonic Sensor
- Connecting Wires

### Theory

The Ultrasonic sensor measures the distance to an object by using ultrasonic sound waves. It calculates the time delay between sending and receiving the sound wave.

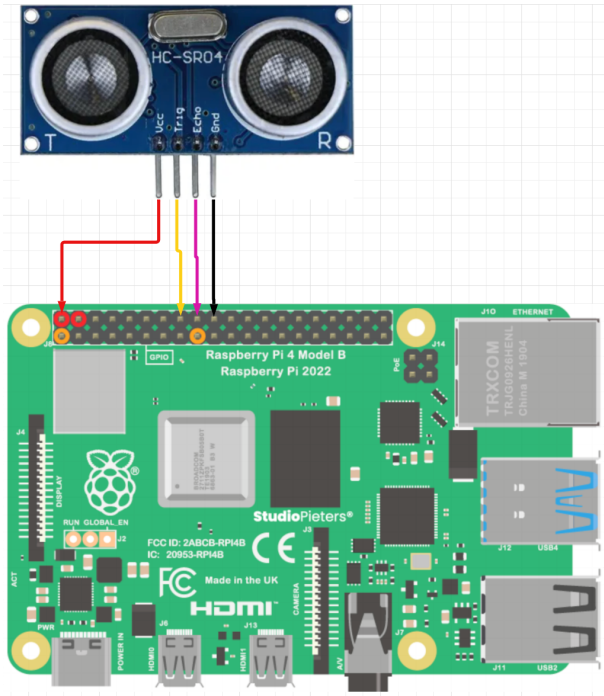
### Procedure

1. Connect the Ultrasonic Sensor to the ESP32 as per the pin diagram.
2. Upload the appropriate code to the ESP32 to read the Ultrasonic Sensor data.
3. Observe the sensor's readings on the serial monitor.

### Connections:

| Component | Pin | Function |
|---|---|---|
| Ultrasonic Sensor | Trig: GPIO 23, Echo: GPIO 24 | Trigger and Echo |

## Circuit Diagram:




## Code

```
import RPi.GPIO as GPIO
import time
import requests

# Set up GPIO pins
GPIO.setmode(GPIO.BCM)
TRIG = 23  # Trigger pin
ECHO = 24  # Echo pin

GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# ThingSpeak API details
THINGSPEAK_WRITE_API_KEY = "YOUR_API_KEY"
THINGSPEAK_URL = "https://api.thingspeak.com/update"

def get_distance():
    # Ensure Trigger is low initially
    GPIO.output(TRIG, False)
    time.sleep(2)

    # Send a short pulse to trigger
```

```
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    # Measure the duration of the Echo pin high
    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()

    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start

    # Calculate distance (34300 cm/s is the speed of sound in air)
    distance = pulse_duration * 17150
    distance = round(distance, 2)

    return distance

def send_to_thingspeak(distance):
    # Create a dictionary of parameters for ThingSpeak
    data = {
        'api_key': THINGSPEAK_WRITE_API_KEY,
        'field4': distance  # Field where you want to store the data
    }

    # Send a request to ThingSpeak
    response = requests.post(THINGSPEAK_URL, params=data)

    if response.status_code == 200:
        print(f'Data sent to ThingSpeak: {distance} cm")
    else:
        print(f'Failed to send data, HTTP status code: {response.status_code}")

try:
    while True:
        distance = get_distance()
        print(f'Measured Distance: {distance} cm")

        # Send distance data to ThingSpeak
        send_to_thingspeak(distance)

        # Wait 15 seconds before sending the next reading (as ThingSpeak has a rate
limit)
        time.sleep(15)

except KeyboardInterrupt:
    print("Measurement stopped by user")
    GPIO.cleanup()
```

### Observation

Observed the working of the Ultrasonic Sensor and its response to environmental changes.

### Result

Successfully interfaced the Ultrasonic Sensor with the ESP32 and obtained the expected readings.

# Embedded Systems & IoT Experiments Manual

## Experiment 4:

## Interface Soil Moisture Sensor with ESP32

### Objective

To interface the Soil Moisture Sensor with the ESP32 and observe its operation.

### Components Required

- ESP32
- Soil Moisture Sensor
- Connecting Wires

### Theory

The Soil Moisture sensor measures the water content in the soil by detecting the conductivity between two probes.

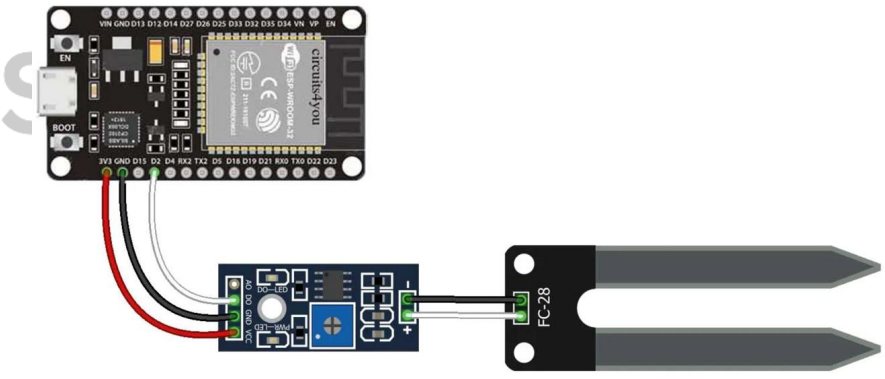
### Procedure

1. Connect the Soil Moisture Sensor to the ESP32 as per the pin diagram.
2. Upload the appropriate code to the ESP32 to read the Soil Moisture Sensor data.
3. Observe the sensor's readings on the serial monitor.

### Connections:

| Component | Pin | Function |
|---|---|---|
| Soil Moisture Sensor | D2 | Analog Output |

### Circuit Diagram:



Code

```
#include <WiFi.h>
#include <HTTPClient.h>

// Replace with your network credentials
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";

// ThingSpeak settings
const char* server = "http://api.thingspeak.com";
String apiKey = "your_API_KEY";

// Define the pin where the soil moisture sensor is connected
const int sensorPin = D2; // D2 pin

void setup() {
 Serial.begin(115200);
 pinMode(sensorPin, INPUT);

 // Connect to WiFi
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
 }
 Serial.println("Connected to WiFi.");
}

void loop() {
 int sensorValue = digitalRead(sensorPin); // Read the digital value from the
sensor
 Serial.print("Soil Moisture Status: ");
 Serial.println(sensorValue);

 if (WiFi.status() == WL_CONNECTED) {
  // Send data to ThingSpeak
  HTTPClient http;
  String url = server + "/update?api_key=" + apiKey + "&field5=" +
String(sensorValue);

  http.begin(url);
  int httpCode = http.GET(); // Send the request

  // Check the HTTP status code
  if (httpCode > 0) {
   String payload = http.getString();
   Serial.println("Data sent to ThingSpeak: " + payload);
  } else {
   Serial.println("Error in sending data");
```

```
  }

  http.end(); // Close connection
 }

 delay(15000); // Wait for 15 seconds before sending the next data
}
```

### Observation

Observed the working of the Soil Moisture Sensor and its response to environmental changes.

### Result

Successfully interfaced the Soil Moisture Sensor with the ESP32 and obtained the expected readings.

# Embedded Systems & IoT Experiments Manual

## Experiment 5:

## Interface Digital Switch with ESP32

### Objective

To interface the Digital Switch with the ESP32 and observe its operation.

### Components Required

- ESP32
- Digital Switch
- Connecting Wires

### Theory

The Digital Switch is an on/off electronic switch that sends a digital signal to the microcontroller when pressed.

### Procedure

1. Connect the Digital Switch to the ESP32 as per the pin diagram.
2. Upload the appropriate code to the ESP32 to read the Digital Switch data.
3. Observe the sensor's readings on the serial monitor.

### PIN Diagram

| Component | Pin | Function |
|---|---|---|
| Digital Switch | D5 | Digital Input |

### Code

```
#include <WiFi.h>
#include <HTTPClient.h>

// Replace with your network credentials
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// ThingSpeak settings
String apiKey = "YOUR_API_KEY";  // Replace with your ThingSpeak Write API Key
const char* server = "http://api.thingspeak.com";

// Pin setup
const int switchPin = D5;  // D5 pin on the ESP32

// Variables to store switch state
int switchState = 0;

void setup() {
  Serial.begin(115200);

  // Initialize the switch pin as input
```

```
  pinMode(switchPin, INPUT_PULLUP);  // Use INPUT_PULLUP if you don't have a
pull-up resistor

  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }

  Serial.println("Connected to WiFi.");
}

void loop() {
  // Read the digital switch state
  switchState = digitalRead(switchPin);

  // Send data to ThingSpeak if switch state changes
  sendToThingSpeak(switchState);

  // Wait for a few seconds before sending the next reading
  delay(15000);  // ThingSpeak accepts data at 15-second intervals
}

void sendToThingSpeak(int state) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    // Prepare the URL to send data
    String url = server;
    url += "/update?api_key=" + apiKey;
    url += "&field6=" + String(state);

    http.begin(url);
    int httpCode = http.GET();  // Send the GET request

    if (httpCode > 0) {
      Serial.println("Data sent to ThingSpeak: " + String(state));
    } else {
      Serial.println("Error sending data to ThingSpeak");
    }

    http.end();  // Close connection
  } else {
    Serial.println("WiFi not connected");
  }
}
```

### Observation

Observed the working of the Digital Switch and its response to environmental changes.

### Result

Successfully interfaced the Digital Switch with the ESP32 and obtained the expected readings.

# Embedded Systems & IoT Experiments Manual

## Experiment 6:

## Interface Magnetic Switch with ESP32

### Objective

To interface the Electro Mechanical Switch with the ESP32 and observe its operation.

### Components Required

- ESP32
- Electro Mechanical Switch
- Connecting Wires

### Theory

The Electro Mechanical Switch is a manually operated switch that controls the flow of electricity by making or breaking the circuit.

### Procedure

1. Connect the Electro Mechanical Switch to the ESP32 as per the pin diagram.
2. Upload the appropriate code to the ESP32 to read the Electro Mechanical Switch data.
3. Observe the sensor's readings on the serial monitor.

### PIN Diagram

| Component | Pin | Function |
|---|---|---|
| Electro Mechanical Switch | D6 | Digital Input |

### Code

```
#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "your_wifi_ssid";      // Your WiFi SSID
const char* password = "your_wifi_password"; // Your WiFi password

const char* apiKey = "YOUR_API_KEY";      // ThingSpeak Write API key
const char* server = "http://api.thingspeak.com/update";

int magneticSwitchPin = D6; // D6 pin
int magneticSwitchState = 0;

void setup() {
 Serial.begin(115200);
 pinMode(magneticSwitchPin, INPUT_PULLUP);  // Initialize magnetic switch pin
as input with pull-up

 WiFi.begin(ssid, password);
 Serial.print("Connecting to WiFi...");
 while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
```

```
  }
  Serial.println("Connected to WiFi");
}

void loop() {
  // Read the magnetic switch state
  magneticSwitchState = digitalRead(magneticSwitchPin);

  // Send data to ThingSpeak via HTTP POST
  if(WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    // Prepare the URL with the API key and field data
    String url = String(server) + "?api_key=" + apiKey + "&field7=" +
String(magneticSwitchState);

    http.begin(url);  // Start the connection to the server
    int httpResponseCode = http.GET();  // Send the HTTP GET request

    // Check the response
    if (httpResponseCode > 0) {
      Serial.println("Data sent successfully. HTTP Response code: " +
String(httpResponseCode));
    } else {
      Serial.println("Error sending data. HTTP Response code: " +
String(httpResponseCode));
    }

    http.end();  // End the connection
  } else {
    Serial.println("WiFi not connected");
  }

  delay(20000);  // Wait for 20 seconds before sending the next update
}
```



### Observation

Observed the working of the Electro Mechanical Switch and its response to environmental changes.

### Result

Successfully interfaced the Electro Mechanical Switch with the ESP32 and obtained the expected readings.

# Module 3: Communication Technologies of IoT

## Experiment 1:

## MQTT Communication using ThingSpeak in ESP32

### Objective

To demonstrate MQTT communication using ESP32 and ThingSpeak.

### Components Required

- ESP32
- Ultra Sonics Sensor
- MQTT Broker
- ThingSpeak Account

### Theory

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency networks. It works on a publish/subscribe model where devices publish messages to a broker, which then forwards them to subscribers.

### Procedure

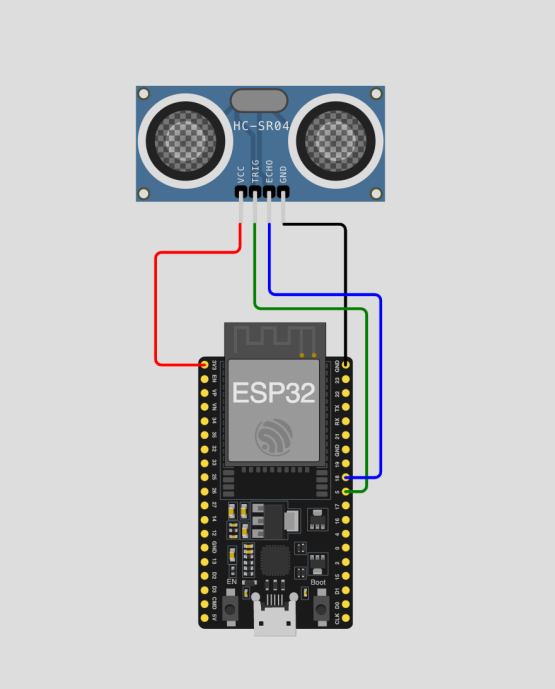
1. Set up the MQTT broker and configure the ThingSpeak channel.
2. Connect the ESP32 to the internet and upload the code to publish sensor data to ThingSpeak via MQTT.
3. Observe the data being published and visualized on the ThingSpeak dashboard.



### PIN Diagram

| Component | Pin | Function |
|---|---|---|
| Ultrasonic Sensor | Trig: GPIO 5, Echo: GPIO 18 | Trigger and Echo |

## Circuit Diagram




## Code

```
#include <WiFi.h>
#include <PubSubClient.h>

// WiFi credentials
const char* ssid = "your_wifi_ssid";
const char* password = "your_wifi_password";

// ThingSpeak MQTT credentials
const char* mqttServer = "mqtt.thingspeak.com";
const int mqttPort = 1883;
const char* mqttUser = "your_mqtt_username";  // Can be your ThingSpeak MQTT Username (optional)
const char* mqttPassword = "your_mqtt_password"; // ThingSpeak API Write key as the password
const char* clientID = "ESP32Client";       // Client ID for MQTT (can be any unique string)

// ThingSpeak channel details
const char* channelID = "YOUR_CHANNEL_ID";
const char* writeAPIKey = "YOUR_API_KEY";

// Ultrasonic Sensor pins
const int trigPin = 5;   // GPIO 5
const int echoPin = 18;  // GPIO 18
```

```
WiFiClient espClient;
PubSubClient client(espClient);

long duration;
int distance;

void setup() {
  Serial.begin(115200);

  // Initialize ultrasonic sensor pins
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Connect to Wi-Fi
  connectToWiFi();

  // Set up MQTT connection
  client.setServer(mqttServer, mqttPort);
}

void loop() {
  // Check Wi-Fi and MQTT connections
  if (!client.connected()) {
    reconnectMQTT();
  }
  client.loop();

  // Get distance from the ultrasonic sensor
  distance = getUltrasonicDistance();

  // Publish data to ThingSpeak via MQTT
  String payload = "field1=" + String(distance);
  String topic = "channels/" + String(channelID) + "/publish/" + writeAPIKey;

  // Publish to ThingSpeak
  if (client.publish(topic.c_str(), payload.c_str())) {
    Serial.println("Data published: " + payload);
  } else {
    Serial.println("Failed to publish data");
  }

  // Wait 20 seconds before sending the next update (ThingSpeak has a rate limit)
  delay(20000);
}

// Function to connect to Wi-Fi
void connectToWiFi() {
  Serial.print("Connecting to WiFi...");
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("Connected to WiFi");
}

// Function to reconnect to MQTT if connection is lost
void reconnectMQTT() {
  while (!client.connected()) {
    Serial.print("Connecting to MQTT...");
    if (client.connect(clientID, mqttUser, mqttPassword)) {
      Serial.println("Connected to MQTT");
    } else {
      Serial.print("Failed to connect to MQTT, state=");
      Serial.println(client.state());
      delay(5000);  // Retry every 5 seconds
    }
  }
}

// Function to get distance using the ultrasonic sensor
int getUltrasonicDistance() {
  // Send a 10µs pulse to the Trig pin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the Echo pin
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance in cm
  distance = duration * 0.034 / 2;
  Serial.print("Distance: ");
  Serial.println(distance);
  return distance;
}
```

### Observation

Successfully demonstrated MQTT communication and real-time data visualization on ThingSpeak.

### Result

Established MQTT communication using ESP32 and ThingSpeak, and visualized the sensor data on the dashboard.

## Module 4: Visualization and Data Types of IoT

# Experiment 1:

# Visualization of Sensor Data using ThingSpeak and Raspberry Pi

### Objective

To visualize diverse sensor data using a dashboard on ThingSpeak with a Raspberry Pi.

### Components Required

- Raspberry Pi
- Internet Connectivity
- ThingSpeak Account

### Theory

Visualization of IoT data allows users to monitor and analyze real-time data from connected devices. ThingSpeak is a platform that enables visualization of data on customizable dashboards, making it easier to track and interact with IoT data.

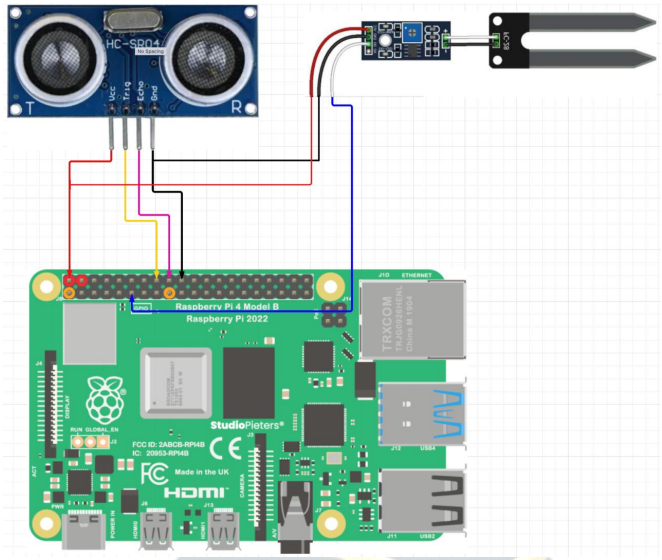
### Procedure

1. Set up the Raspberry Pi and connect it to the internet.
2. Interface the sensors with the Raspberry Pi.
3. Configure ThingSpeak to receive data from the Raspberry Pi.
4. Visualize the sensor data on the ThingSpeak dashboard.

### Connections:

| Component | Pin | Function |
|---|---|---|
| Ultrasonic Sensor | Trig: GPIO 23, Echo: GPIO 24 | Analog Output |
| Soil Moisture Sensor | GPIO 17 | Analog Output |

## Circuit Diagram




## Code

```
import time
import RPi.GPIO as GPIO
import Adafruit_DHT
import paho.mqtt.client as mqtt

# ThingSpeak details
THINGSPEAK_CHANNEL_ID = "Your_Channel_ID"
THINGSPEAK_API_KEY = "Your_API_Key"
THINGSPEAK_MQTT_HOST = "mqtt.thingspeak.com"
THINGSPEAK_PORT = 1883
THINGSPEAK_TOPIC = "channels/" + THINGSPEAK_CHANNEL_ID + "/publish/" +
THINGSPEAK_API_KEY

# Setup GPIO mode
GPIO.setmode(GPIO.BCM)

# Ultrasonic sensor pins
TRIG = 23
ECHO = 24

# Soil moisture sensor pin
SOIL_MOISTURE_PIN = 17

# Set up GPIO pins
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
```

```
GPIO.setup(SOIL_MOISTURE_PIN, GPIO.IN)

# Function to get distance from ultrasonic sensor
def get_distance():
    GPIO.output(TRIG, False)
    time.sleep(2)
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()

    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    distance = round(distance, 2)

    return distance

# Function to get soil moisture data
def get_soil_moisture():
    if GPIO.input(SOIL_MOISTURE_PIN):
        return 1  # Dry
    else:
        return 0  # Wet

# MQTT publishing function
def publish_to_thingspeak(distance, soil_moisture):
    client = mqtt.Client()
    client.connect(THINGSPEAK_MQTT_HOST, THINGSPEAK_PORT, 60)
    payload = "field1=" + str(soil_moisture) + "&field2=" + str(distance)
    client.publish(THINGSPEAK_TOPIC, payload)
    client.disconnect()

# Main loop
try:
    while True:
        distance = get_distance()
        soil_moisture = get_soil_moisture()

        print(f"Distance: {distance} cm")
        print(f"Soil Moisture: {soil_moisture} (1 = Dry, 0 = Wet)")

        # Publish data to ThingSpeak
        publish_to_thingspeak(distance, soil_moisture)

        # Wait before next reading
```

```
        time.sleep(20)  # every 20 seconds

except KeyboardInterrupt:
    print("Measurement stopped by user")
    GPIO.cleanup()
```

### Observation

Successfully visualized the sensor data on the ThingSpeak dashboard using Raspberry Pi.

### Result

Demonstrated the visualization of IoT sensor data using ThingSpeak and Raspberry Pi.

## Module 5: Retrieving Data and M2M Communication

# Experiment 1:

# Device Control using Mobile Apps or Web Pages

### Objective

To control IoT devices using mobile apps or web pages.

### Components Required

- ESP32
- Internet Connectivity
- Mobile App/Web Interface

### Theory

Device control via mobile apps or web pages allows users to remotely operate IoT devices. This control mechanism is crucial in smart home applications and industrial automation.

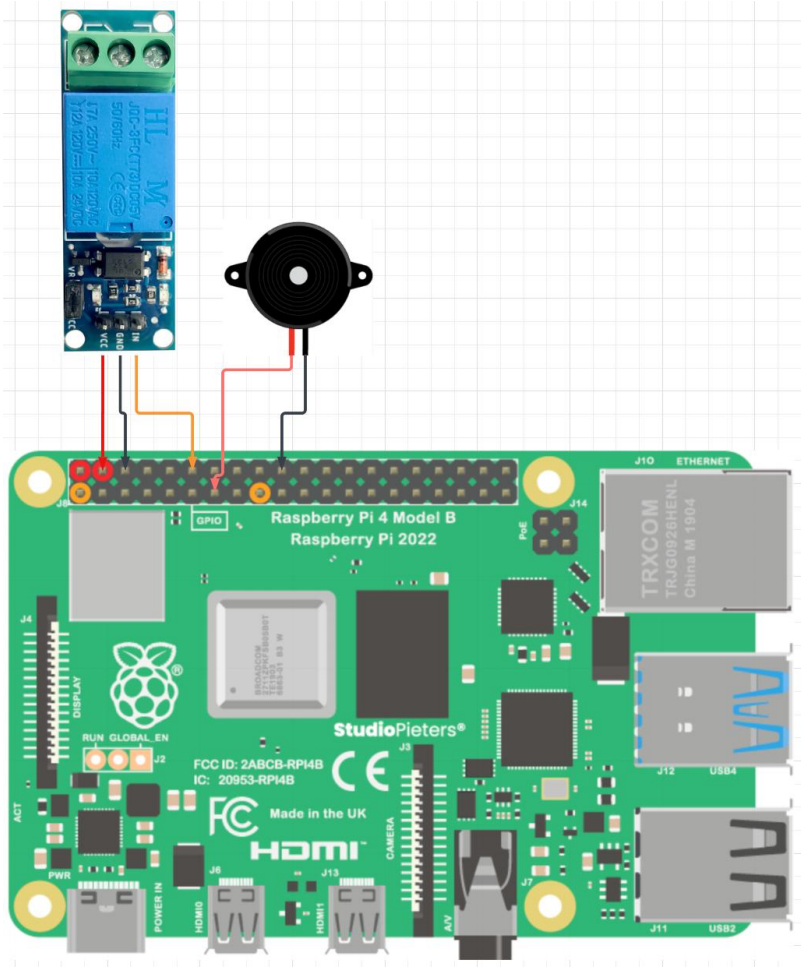
### Procedure

1. Connect the ESP32 to the internet.
   2. Develop a mobile app or web interface for controlling the device.
   3. Use the app or web page to send commands to the ESP32 and observe the response.

### Connections:

| Component | Pin | Function |
|---|---|---|
| Relay | GPIO 18 | Analog Output |
| Buzzer | GPIO 27 | Analog Output |

## Circuit Diagram




## Code



```
import RPi.GPIO as GPIO

import time

import requests


# Setup GPIO mode

GPIO.setmode(GPIO.BCM)
```

```
# Relay and Buzzer Pins
RELAY_PIN = 18
BUZZER_PIN = 27

# Setup GPIO pins
GPIO.setup(RELAY_PIN, GPIO.OUT)
GPIO.setup(BUZZER_PIN, GPIO.OUT)

# ThingSpeak API details
THINGSPEAK_CHANNEL_ID = "Your_Channel_ID"
THINGSPEAK_API_KEY = "Your_Read_API_Key"
THINGSPEAK_READ_URL = f"https://api.thingspeak.com/channels/{THINGSPEAK_CHANNEL_ID}/fields/5.json?api_key={THINGSPEAK_API_KEY}&results=1"

# Function to fetch the latest value of the switch (field 5 and field 6) from ThingSpeak
def get_thingspeak_data():
    try:
        response = requests.get(THINGSPEAK_READ_URL)
        data = response.json()
        relay_value = int(data['feeds'][0]['field5'])  # Field 5 is for relay
        buzzer_value = int(data['feeds'][0]['field6'])  # Field 6 is for buzzer
        return relay_value, buzzer_value
    except Exception as e:
        print("Error fetching data from ThingSpeak:", e)
        return None, None
```

```
# Main loop
try:
    while True:
        # Get the values for relay and buzzer from ThingSpeak
        relay_value, buzzer_value = get_thingspeak_data()

        if relay_value is not None and buzzer_value is not None:
            print(f"Relay: {relay_value}, Buzzer: {buzzer_value}")

            # Control relay based on the value in Field 5
            if relay_value == 1:
                GPIO.output(RELAY_PIN, GPIO.HIGH)  # Turn on relay
            else:
                GPIO.output(RELAY_PIN, GPIO.LOW)  # Turn off relay

            # Control buzzer based on the value in Field 6
            if buzzer_value == 1:
                GPIO.output(BUZZER_PIN, GPIO.HIGH)  # Turn on buzzer
            else:
                GPIO.output(BUZZER_PIN, GPIO.LOW)  # Turn off buzzer

        # Wait for some time before checking again
        time.sleep(10)  # Fetch every 10 seconds

except KeyboardInterrupt:
    print("Process interrupted by user")
    GPIO.cleanup()
```

### Observation

Successfully controlled the IoT devices using the mobile app/web page.

### Result

Demonstrated remote control of IoT devices using a mobile app/web interface.

# Embedded Systems & IoT Experiments Manual

## Experiment 2:

## Machine to Machine Communication using ThingSpeak

### Objective

To demonstrate machine-to-machine (M2M) communication using ESP32 and ThingSpeak.

### Components Required

- ESP32
- Internet Connectivity
- ThingSpeak Account

### Theory

Machine-to-Machine (M2M) communication enables direct communication between devices over the internet. This allows devices to share data and control each other without human intervention, which is essential in automation systems.

### Procedure

1. Connect the ESP32 to the internet and configure it to communicate with ThingSpeak.
2. Set up another device to listen for data from the ESP32 via ThingSpeak.
3. Demonstrate the M2M communication by having the devices interact based on the data exchanged.

- **Sender Code**



```
#include <WiFi.h>
#include <HTTPClient.h>

// Replace with your network credentials
const char* ssid = "Your_SSID";
const char* password = "Your_PASSWORD";

// ThingSpeak API details
String apiKey = "Your_Write_API_Key";  // Write API Key
const char* server = "http://api.thingspeak.com";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  // Connect to Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
```

```
  Serial.println("Connecting to WiFi...");
 }

 Serial.println("Connected to WiFi");
}

void loop() {
 if (WiFi.status() == WL_CONNECTED) {
  HTTPClient http;

  // Simulate sensor data (or replace with actual sensor readings)
  float sensorValue = random(0, 100);  // Simulated sensor value

  // Prepare the URL with the data to send
  String url = String(server) + "/update?api_key=" + apiKey + "&field1=" +
String(sensorValue);

  // Send the HTTP GET request to ThingSpeak
  http.begin(url);
  int httpResponseCode = http.GET();

  if (httpResponseCode > 0) {
   Serial.print("Data sent to ThingSpeak, response code: ");
   Serial.println(httpResponseCode);
  } else {
   Serial.print("Error sending data: ");
   Serial.println(httpResponseCode);
  }

  http.end();  // Close the connection
 }

 // Send data every 20 seconds
 delay(20000);
}
```



- Receiver Code

```
#include <WiFi.h>
#include <HTTPClient.h>

// Replace with your network credentials
const char* ssid = "Your_SSID";
const char* password = "Your_PASSWORD";
```

```
// ThingSpeak API details
String apiKey = "Your_Read_API_Key";  // Read API Key
String channelID = "Your_Channel_ID";  // Channel ID
const char* server = "http://api.thingspeak.com";

// Pin where you want to control an actuator (e.g., LED, relay, or buzzer)
const int actuatorPin = 2;

void setup() {
 Serial.begin(115200);
 WiFi.begin(ssid, password);

 // Setup the pin as output
 pinMode(actuatorPin, OUTPUT);

 // Connect to Wi-Fi
 while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
 }

 Serial.println("Connected to WiFi");
}

void loop() {
 if (WiFi.status() == WL_CONNECTED) {
  HTTPClient http;

  // Prepare the URL to read data from ThingSpeak
  String url = String(server) + "/channels/" + channelID +
"/fields/1.json?api_key=" + apiKey + "&results=1";

  // Send the HTTP GET request to ThingSpeak
  http.begin(url);
  int httpResponseCode = http.GET();

  if (httpResponseCode > 0) {
   String payload = http.getString();
   Serial.println("Received data from ThingSpeak:");
   Serial.println(payload);

   // Extract the field1 value from the JSON response
   int valueIndex = payload.indexOf("\"field1\":\"");
   if (valueIndex != -1) {
    String valueStr = payload.substring(valueIndex + 9, payload.indexOf("\"",
valueIndex + 9));
    float receivedValue = valueStr.toFloat();

    // Actuate based on the received value
    if (receivedValue > 50) {  // Example condition
```

```
        digitalWrite(actuatorPin, HIGH);  // Turn on actuator
        Serial.println("Actuator ON");
       } else {
        digitalWrite(actuatorPin, LOW);  // Turn off actuator
        Serial.println("Actuator OFF");
       }
     }
    } else {
     Serial.print("Error receiving data: ");
     Serial.println(httpResponseCode);
    }

    http.end();  // Close the connection
  }

  // Read data every 20 seconds
  delay(20000);
}
```

### Observation

Successfully demonstrated M2M communication using ThingSpeak and ESP32.

### Result

Implemented M2M communication, allowing devices to interact with each other over ThingSpeak.

## Module 6: Control & Supervisory Level of Automation

### Objective

To understand the control and supervisory aspects of automation using PLC, real-time control systems, and SCADA.

### Components Required

- Programmable Logic Controller (PLC)
- Computer with SCADA software

### Theory

Programmable Logic Controller (PLC)
PLCs are industrial digital computers designed for the control of manufacturing processes. They are robust, capable of handling harsh industrial environments, and are programmable to handle various types of tasks.
Real-Time Control Systems
These systems are designed to respond to inputs or events within a strict time frame, crucial in processes where timing is critical. Real-time control systems are essential for ensuring smooth and timely operations in industrial automation.
Supervisory Control & Data Acquisition (SCADA)
SCADA systems are used for controlling and monitoring industrial processes at a supervisory level. They provide real-time data acquisition, process control, and automation in industries like energy, water management, and manufacturing.

### Procedure

1. Set up the PLC with the necessary input and output modules.
2. Install and configure SCADA software on the computer.
3. Program basic logic gates and Boolean expressions using ladder diagrams.



### PIN Diagram

| Device | Pin | Function |
|---|---|---|
| PLC | I1 | Input for sensor data |
| PLC | O1 | Output to actuator |

### Observation

Successfully create ladder diagrams and simulate PLC operations.
Monitor and control the process via SCADA.

### Result

Implemented basic control and supervisory systems using PLC and SCADA.