



**ADHIYAMAAN COLLEGE OF ENGINEERING**

**(Autonomous), Hosur**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**(Accredited by NBA)**



# **722CSP07 – INTERNET OF THINGS LABORATORY (REGULATION - 2022)**

**STAFF INCHARGE**

**HOD**



## **ADHIYAMAAN COLLEGE OF ENGINEERING (Autonomous), Hosur**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
(Accredited by NBA)**



### **Vision of the Institute**

To foster ACE as a centre for nurturing and developing world class Engineers and Managers who convert global challenges into opportunities through value-based quality education.

### **Mission of the Institute**

**M 1 :** To impart value-based quality education through effective teaching-learning processes

**M 2 :** To nurture creativity, excellence and critical thinking by applying global competency factors to contribute and excel in the rapidly growing technological world.

**M 3 :** To continuously develop and improve holistic and innovative personality for global Mobility.

**M 4 :** To make ACE a centre for excellence.

### **Vision of the Department**

To empower young minds to become resilient professionals, instilled with ethical principles and equipped with cutting-edge technologies to meet the evolving demands of the world

### **Mission of the Department**

**M 1 :** To empower individuals with a comprehensive understanding of computer engineering principles and its applications through effective teaching and learning practices.

**M 2 :** To cultivate excellence and critical thinking, while leveraging global competency, thus enabling significant contributions to societal challenges in the fast-paced technological landscape.

**M 3 :** To facilitate the students to work with modern tools and technologies to foster innovation, a zest for higher studies and to build leadership qualities by inculcating the spirit of ethical values.

**PEO1 :** The graduates will have sound knowledge in Mathematics, Science and Engineering concepts

### **Program Educational Objectives (PEOs)**

necessary to formulate, analyse, design and solve Engineering problems and to prepare them for higher learning, research and industry.

**PEO2 :** The graduates will possess innovative skills to assess and apply the rapid changes in technology and to engage in research leading to novel solutions for human, social and global competency.

**PEO3 :** The graduates will acquire knowledge and grab opportunities to work as teams in a multidisciplinary environment, communicate ideas effectively with diverse audiences demonstrate leadership qualities with ethical values and engage in lifelong learning.



**ADHIYAMAAN COLLEGE OF ENGINEERING  
(Autonomous), Hosur**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
(Accredited by NBA)**



722CSP07

INTERNET OF THINGS LABORATORY

### **COURSE OBJECTIVES:**

- To implement the concepts of IoT.
- To interface different platforms like Arduino and Raspberry pi
- To design and implement the related applications.
- To learn how to analysis the data in IoT.

### **LIST OF EXPERIMENTS:**

#### **1. LED Blinking using ESP32 [2 hours]**

Concepts [1 hour]: Introduction to ESP32, GPIO operations, digital input/output, Arduino IDE configuration.

Hands-On [1 hour]:

- Connect an LED to GPIO pin of ESP32.
- Write a program to blink the LED at regular intervals using Arduino IDE.

#### **2. Working with GPIO using ESP32 [2 hours]**

Concepts [1 hour]: Understanding digital input/output, push button interfacing, and event-driven control.

Hands-On [1 hour]:

- Connect a push button to ESP32.
- Write a program to control an LED using the button input.

#### **3. LED Blinking using Raspberry Pi [2 hours]**

Concepts [1 hour]: Introduction to Raspberry Pi GPIO, pin mapping, and Python GPIO library.

Hands-On [1 hour]:

- Connect an LED to Raspberry Pi GPIO.
- Write a Python program using RPi.GPIO to blink the LED.

#### **4. Interfacing DHT11 and Soil Moisture Sensors with Raspberry Pi [3 hours]**

Concepts [1.5 hours]: Digital vs Analog sensors, DHT11 temperature & humidity sensing, soil moisture measurement using ADC (MCP3008).

Hands-On [1.5 hours]:

- Interface DHT11 and soil moisture sensors with Raspberry Pi.
- Display the temperature, humidity, and moisture readings on the console.

#### **5. Interfacing Ultrasonic Sensor (HC-SR04) with Raspberry Pi for Distance Measurement [2 hours]**

Concepts [1 hour]: Working principle of ultrasonic sensors, time-of-flight measurement, GPIO input/output operations.

Hands-On [1 hour]:

- Interface HC-SR04 sensor with Raspberry Pi.
- Measure and display object distance in centimeters using Python.

#### **6. Uploading DHT11 Sensor Data from Raspberry Pi to ThingSpeak Cloud [3 hours]**

Concepts [1.5 hours]: IoT data communication, HTTP protocol, cloud platforms (ThingSpeak), API keys, data visualization.

Hands-On [1.5 hours]:

- Connect DHT11 to Raspberry Pi.
- Write a Python script to upload temperature and humidity data to ThingSpeak using API key.

#### **7. Monitoring SpO<sub>2</sub> and Heart Rate using Arduino and MAX30100 Sensor [3 hours]**

Concepts [1.5 hours]: Principles of pulse oximetry, I<sup>2</sup>C communication, MAX30100 sensor operation.

Hands-On [1.5 hours]:

- Interface MAX30100 sensor with Arduino UNO.
- Display SpO<sub>2</sub> and heart rate on Serial Monitor.

#### **8. Motion Tracking using MPU6050 Sensor and Cloud Logging [3 hours]**

Concepts [1.5 hours]: MEMS sensors, accelerometer and gyroscope data, I<sup>2</sup>C protocol, ThingSpeak integration.

Hands-On [1.5 hours]:

- Interface MPU6050 with Arduino UNO.
- Upload accelerometer and gyroscope data to ThingSpeak cloud for visualization.

### **9. Barometric Pressure and Altitude Measurement using BMP Sensor with Arduino [2 hours]**

Concepts [1 hour]: Working of BMP180/BMP280, atmospheric pressure and altitude calculation, I<sup>2</sup>C interfacing.

Hands-On [1 hour]:

- Interface BMP sensor with Arduino UNO.
- Measure and display pressure, temperature, and altitude readings using Serial Monitor.

### **10. Weather Station using Raspberry Pi with DHT11, BMP Sensor, and ThingSpeak Integration [4 hours]**

Concepts [2 hours]: IoT-based weather monitoring, multi-sensor data integration, cloud dashboard setup.

Hands-On [2 hours]:

- Connect DHT11 and BMP sensors to Raspberry Pi.
- Write a Python program to read temperature, humidity, and pressure values.
- Upload the collected data to ThingSpeak and visualize live environmental trends.

### **11. Smart Home Automation using ESP32 and Blynk App [4 hours]**

**Concepts [2 hours]:**

Home automation architecture, IoT mobile control interfaces, Blynk IoT platform setup.

**Hands-On [2 hours]:**

- Connect ESP32 to the Blynk cloud.
- Control multiple home appliances (LED, fan, etc.) using a smartphone app.

### **12. Mini Project: IoT-based Real-Time Monitoring System [6 hours]**

Concepts [3 hours]: End-to-end IoT system design, edge data processing, and cloud reporting.

Hands-On [3 hours]:

- Choose any IoT use case (e.g., smart agriculture, air quality, or health monitoring).

## **COURSE OUTCOMES:**

On successful completion the students will be able to

CO1: Demonstrate basic programming and control using Arduino and Raspberry Pi.

CO2: Operate analog and digital sensors using microcontroller and microprocessor platforms.

CO3: Implement real-time monitoring using sensor data for health and environment applications.

CO4: Upload and analyze sensor data using cloud platforms like Thing Speak for IoT applications.

CO5: Design and develop an integrated IoT-based weather monitoring system with cloud connectivity.



**ADHIYAMAAN COLLEGE OF ENGINEERING  
(Autonomous), Hosur**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
(Accredited by NBA)**



CO's / PO's	PO 1	PO2	PO 3	PO 4	PO 5	PO 6	PO7	PO8	PO9	PO1 0	PO1 1	PO1 2	PS0 1	PS0 2	PSO 3
<b>CO1</b>	2	3	2	2	-	2	-	-	-	-	-	-	2	2	3
<b>CO2</b>	2	3	2	2	-	2	-	-	-	-	-	-	1	2	2
<b>CO3</b>	2	3	2	2	-	1	-	-	-	-	-	-	3	3	3
<b>CO4</b>	2	3	2	2	-	1	-	-	-	-	-	-	2	2	2
<b>CO5</b>	2	3	2	2	-	1	-	-	-	-	-	-	3	2	3
<b>AVG</b>	2.1	3	2.1	2	-	1.5	-	-	-	-	-	-	2	2	3

**3- HIGH, 2 MODERATE, 1-LOW, '- 'NO CORRELATION**

## INDEX

<b>S. NO.</b>	<b>NAME OF THE EXPERIMENT</b>	<b>PAGE NO</b>
1	LED Blinking using ESP32	9
2	Working with GPIO	13
3	LED Blinking using Raspberry Pi	17
4	Interfacing DHT11 and Soil Moisture Sensors with Raspberry Pi	21
5	Interfacing Ultrasonic Sensor (HC-SR04) with Raspberry Pi for Distance Measurement	26
6	Uploading DHT11 Sensor Data from Raspberry Pi to Thing Speak Cloud	30
7	Monitoring SPO2 and Heart Rate using Arduino and MAX30100 Sensor	35
8	Motion Tracking using MPU6050 Sensor and Cloud Logging	40
9	Barometric Pressure and Altitude Measurement using BMP Sensor with Arduino	46
10	Weather Station using Raspberry Pi with DHT11, BMP Sensor, and Thing Speak Integration	51

E.NO: 01	LED Blinking using ESP32
DATE:	

**AIM:**

To write and upload a program on an ESP32 to blink an LED at a fixed interval, verifying basic GPIO digital output control.

**PROCEDURE:**

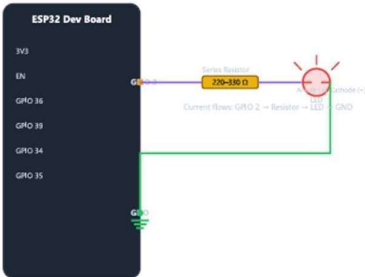
- Connect the ESP32 board to your computer using a USB cable.
- Install and configure **ESP32 board support** in Arduino IDE.
- Select the correct **Board** (ESP32 Dev Module) and **COM Port** in Arduino IDE.
- Connect an LED with a 220–330  $\Omega$  resistor to **GPIO 2** and **GND** (or use the onboard LED).
- Open Arduino IDE and write the LED blink program using `digitalWrite()` and `delay()`.
- Compile the code and upload it to the ESP32 board.
- Observe the LED blinking at regular intervals (ON for 500 ms, OFF for 500 ms).
- Verify the output in the **Serial Monitor** and record the observations.

# CIRCUIT DIAGRAM:

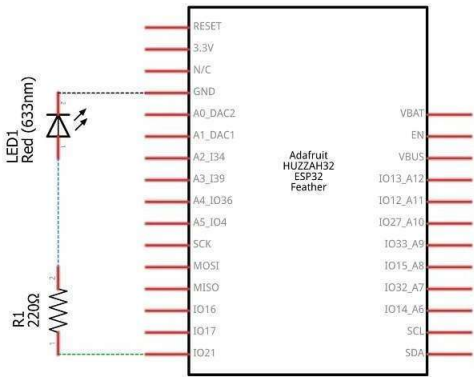
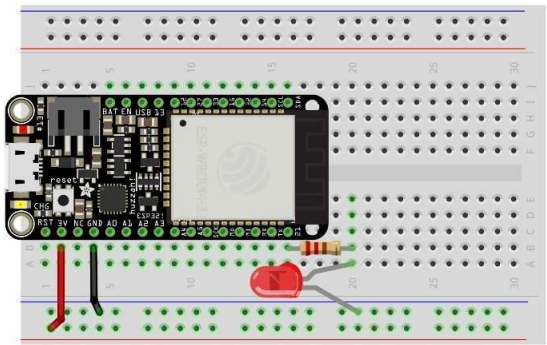
## ESP32 LED Blinking – Circuit Diagram

This diagram shows how to connect an external LED to an ESP32 using **GPIO 2** with a current-limiting resistor (220–330  $\Omega$ ). Many ESP32 boards also have a built-in LED on GPIO 2; in that case the wiring is not required.

### 1) Schematic (Conceptual)



### 2) Minimal Wiring Map



## **PROGRAM (Arduino – ESP32):**

```
const int LED_PIN = 2; // Built-in LED on many ESP32 boards  
(often GPIO 2)
```

```
const unsigned long ON_MS = 500; // LED ON time in
```

```
milliseconds const unsigned long OFF_MS = 500; // LED OFF  
time in milliseconds
```

```
void setup() {
```

```
  pinMode(LED_PIN,
```

```
  OUTPUT);
```

```
  digitalWrite(LED_PIN, LOW); // start OFF
```

```
  Serial.begin(115200);
```

```
  delay(50);
```

```
  Serial.println("ESP32 LED Blink: starting...");
```

```
}
```

```
void loop() {
```

```
  // Turn LED ON
```

```
  digitalWrite(LED_PIN, HIGH);
```

```
  Serial.println("LED: ON");
```

```
  delay(ON_MS);
```

```
  // Turn LED OFF
```

```
  digitalWrite(LED_PIN, LOW);
```

```
  Serial.println("LED: OFF");
```

```
  delay(OFF_MS);
```

```
}
```

## **EXPECTED OUTPUT:**

- The LED **turns ON for 500 ms** and **OFF for 500 ms**, repeating (i.e., ~1 Hz blink).
- (Optional) Serial Monitor at **115200 baud** prints:
- ESP32 LED Blink: starting...
- LED: ON
- LED: OFF
- LED: ON
- LED: OFF

## **RESULT:**

The ESP32 successfully drives a GPIO pin to toggle an LED at a fixed interval, demonstrating correct setup of the Arduino toolchain for ESP32, GPIO configuration as output, and timing using `delay()`.

E.NO: 02	Working with GPIO
DATE:	

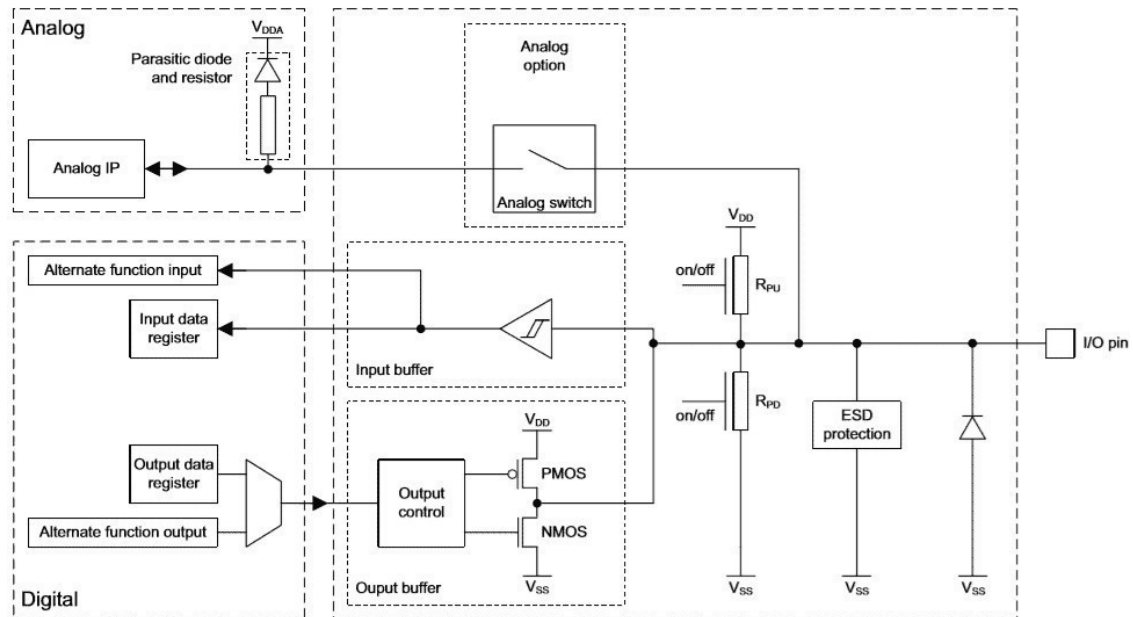
**AIM:**

To write and upload a program on an ESP32 to blink an LED at a fixed interval, verifying basic GPIO digital output control.

**PROCEDURE:**

1. Connect the ESP32 board to your computer using a USB cable.
2. Install and configure ESP32 board support in Arduino IDE.
3. Select the correct Board (ESP32 Dev Module) and COM Port in Arduino IDE.
4. Connect an LED with a 220–330  $\Omega$  resistor to GPIO 2 and GND (or use the onboard LED).
5. Open Arduino IDE and write the LED blink program using `digitalWrite()` and `delay()`.
6. Compile the code and upload it to the ESP32 board.
7. Observe the LED blinking at regular intervals (ON for 500 ms, OFF for 500 ms).
8. Verify the output in the Serial Monitor and record the observations.

## CIRCUIT DIAGRAM:



## PROGRAM (Arduino – ESP32):

```
// Experiment 2: Button with LED using ESP32
```

```
// LED → GPIO23, Button → GPIO22 (with 10k pull-down)
```

```
const int buttonPin = 22; // Push button
```

```
input pin const int ledPin    = 23; // LED
```

```
output pin
```

```
int buttonState = 0; // variable for reading button status
```

```
void setup() {
```

```
    Serial.begin(115200); // Start serial
```

```
    communication pinMode(ledPin, OUTPUT); // Set
```

## LED pin as OUTPUT

```
pinMode(buttonPin, INPUT); // Set button pin as INPUT
(external pull- down)

digitalWrite(ledPin, LOW); // Ensure LED is OFF initially

Serial.println("Experiment 2: Button + LED");
}
void loop() {
    // Read the state of the push
    button buttonState =
    digitalRead(buttonPin);

    if (buttonState == HIGH) {
        digitalWrite(ledPin, HIGH); // Turn
        LED ON Serial.println("Button Pressed
        → LED ON");
    } else {
        digitalWrite(ledPin, LOW); // Turn
        LED OFF Serial.println("Button
        Released → LED OFF");
    }

    delay(200); // Small delay for stability
}
```

## **EXPECTED OUTPUT:**

Experiment 2: Button + LED

Button Released → LED OFF

Button Released → LED OFF

Button Pressed → LED ON

Button Pressed → LED ON

Button Released → LED OFF

Button Released → LED OFF

## **RESULT:**

The LED glowed when the push button was pressed and turned off when released, verifying GPIO input-output operation in ESP32.

E.NO: 03	LED Blinking using Raspberry Pi
DATE:	

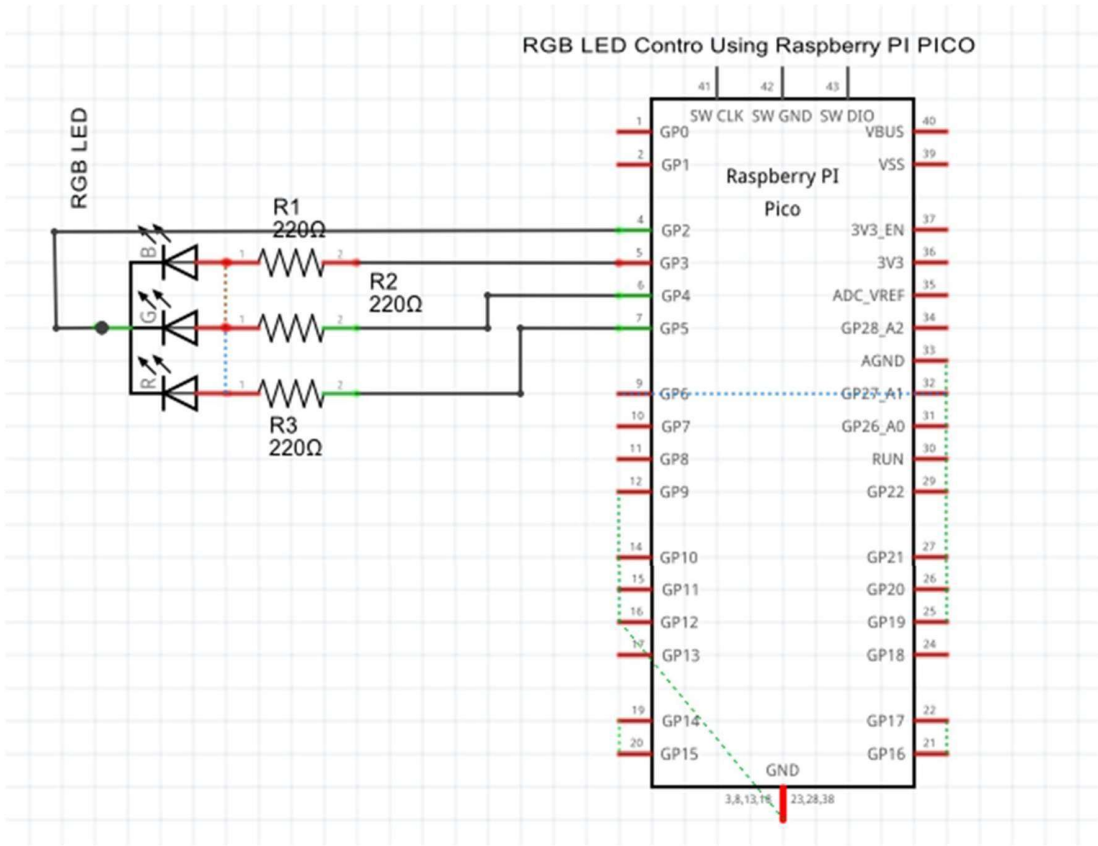
**AIM:**

To write and execute a Python program on Raspberry Pi to blink an LED at a fixed interval, verifying GPIO output control.

**PROCEDURE:**

1. Connect the **long leg (anode)** of the LED to **GPIO17 (Pin 11)** of the Raspberry Pi through a **220  $\Omega$  resistor**.
2. Connect the **short leg (cathode)** of the LED to the **GND pin (Pin 6)**.
3. Power up the Raspberry Pi and open the **Thonny IDE** or terminal.
4. Write a Python program using the **RPI.GPIO library** to blink the LED.
5. Save the program and run it. The LED should blink at a fixed interval.

CIRCUIT DIAGRAM:



## **PROGRAM (Arduino – ESP32):**

```
# Experiment 3: LED Blinking using Raspberry

Pi import RPi.GPIO as GPIO
import time

LED_PIN = 17 # GPIO pin 17 (Pin 11 on header)

GPIO.setmode(GPIO.BCM)# Use BCM pin
numbering GPIO.setup(LED_PIN, GPIO.OUT)

print("LED Blink Program Started...")
try:
    while True:
        GPIO.output(LED_PIN, GPIO.HIGH) # LED ON
        print("LED ON")
        time.sleep(1) # 1 second
        delay
        GPIO.output(LED_PIN, GPIO.LOW) # LED OFF
        print("LED OFF")
        time.sleep(1)
except KeyboardInterrupt:
    print("Program stopped by
    User")
finally:
    GPIO.cleanup()
```

### **EXPECTED OUTPUT:**

LED Blink Program Started... LED ON

LED OFF LED ON

LED OFF

...

### **RESULT:**

The LED blinked at a fixed interval using Raspberry Pi GPIO, verifying successful GPIO output control.p

E.NO: 04	Interfacing DHT11 and Soil Moisture Sensors with Raspberry Pi
DATE:	

**AIM:**

To interface a DHT11 temperature & humidity sensor and a Soil Moisture sensor with Raspberry Pi, and to display the temperature, humidity, and soil moisture level readings using Python.

**PROCEDURE:**

1. Connect the DHT11 sensor:

- VCC → 3.3V (Pin 1)
- DATA → GPIO4 (Pin 7)
- GND → GND (Pin 6)

2. Connect the Soil Moisture Sensor via MCP3008 ADC:

- Soil Sensor Analog Output → CH0 of MCP3008
- MCP3008 VDD, VREF → 3.3V
- MCP3008 AGND, DGND → GND
- MCP3008 CLK → GPIO11 (Pin 23)
- MCP3008 DOUT → GPIO9 (Pin 21)
- MCP3008 DIN → GPIO10 (Pin 19)
- MCP3008 CS → GPIO8 (Pin 24)

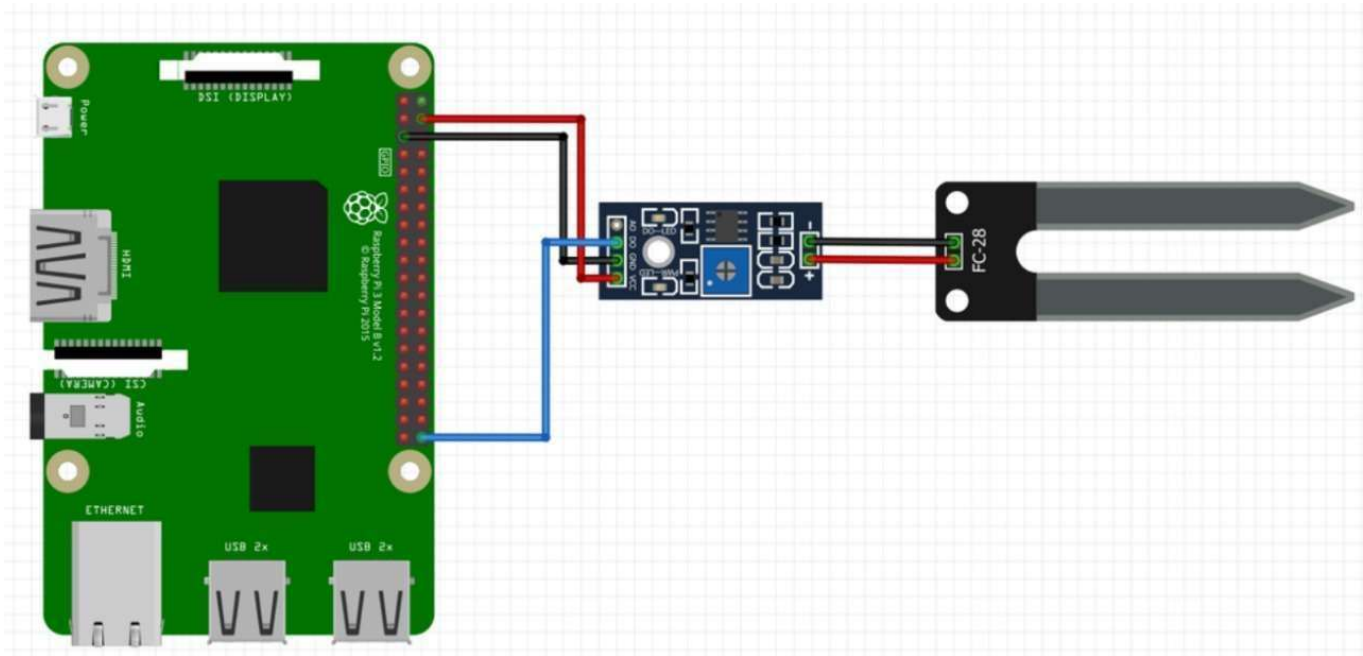
3. Open the Thonny IDE or terminal on Raspberry Pi.

4. Install necessary libraries: `sudo pip3 install Adafruit_DHT spidev`.

5. Write and execute the Python program to read values from both sensors.

6. Observe the readings displayed on the terminal.

CIRCUIT DIAGRAM:



## PROGRAM (Arduino – ESP32):

# Experiment 4: Interfacing DHT11 and Soil Moisture Sensor with Raspberry Pi

```
import Adafruit_DHT
import spidev
import time
```

```
# Initialize SPI for MCP3008
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 1350000
```

```
# DHT11 setup
DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 4 # GPIO4 (Pin 7)
```

```
# Function to read MCP3008 channel
def read_channel(channel):
    adc = spi.xfer2([1, (8 + channel) << 4, 0])
    data = ((adc[1] & 3) << 8) + adc[2]
    return data
```

```
print("DHT11 and Soil Moisture Sensor Reading Started...")
```

```
try:
    while True:
        # Read DHT11 sensor data
        humidity, temperature = Adafruit_DHT.read(DHT_SENSOR,
DHT_PIN)

        # Read soil moisture value
        soil_value = read_channel(0)
        moisture_percent = (1023 - soil_value) / 10.23 # Convert to
percentage
```

```
        if humidity is not None and temperature is not None:
            print(f"Temperature: {temperature:.1f}°C Humidity:
{humidity:.1f}% Soil Moisture: {moisture_percent:.1f}%")
        else:
            print("Failed to retrieve data from DHT11 sensor")

        time.sleep(2)

except KeyboardInterrupt:
    print("Program stopped by User")
finally:
    spi.close()
```

## **EXPECTED OUTPUT:**

DHT11 and Soil Moisture Sensor

Reading Started...

Temperature: 29.3°C Humidity:

65.0% Soil Moisture: 45.2%

Temperature: 29.4°C Humidity:

64.8% Soil Moisture: 46.0%

## **RESULT:**

The DHT11 sensor successfully measured temperature and humidity, and the soil moisture sensor accurately reported the soil's moisture level using the MCP3008 ADC with Raspberry Pi.

E.NO: 05

## Interfacing Ultrasonic Sensor (HC-SR04) with Raspberry Pi for Distance Measurement

DATE:

### AIM:

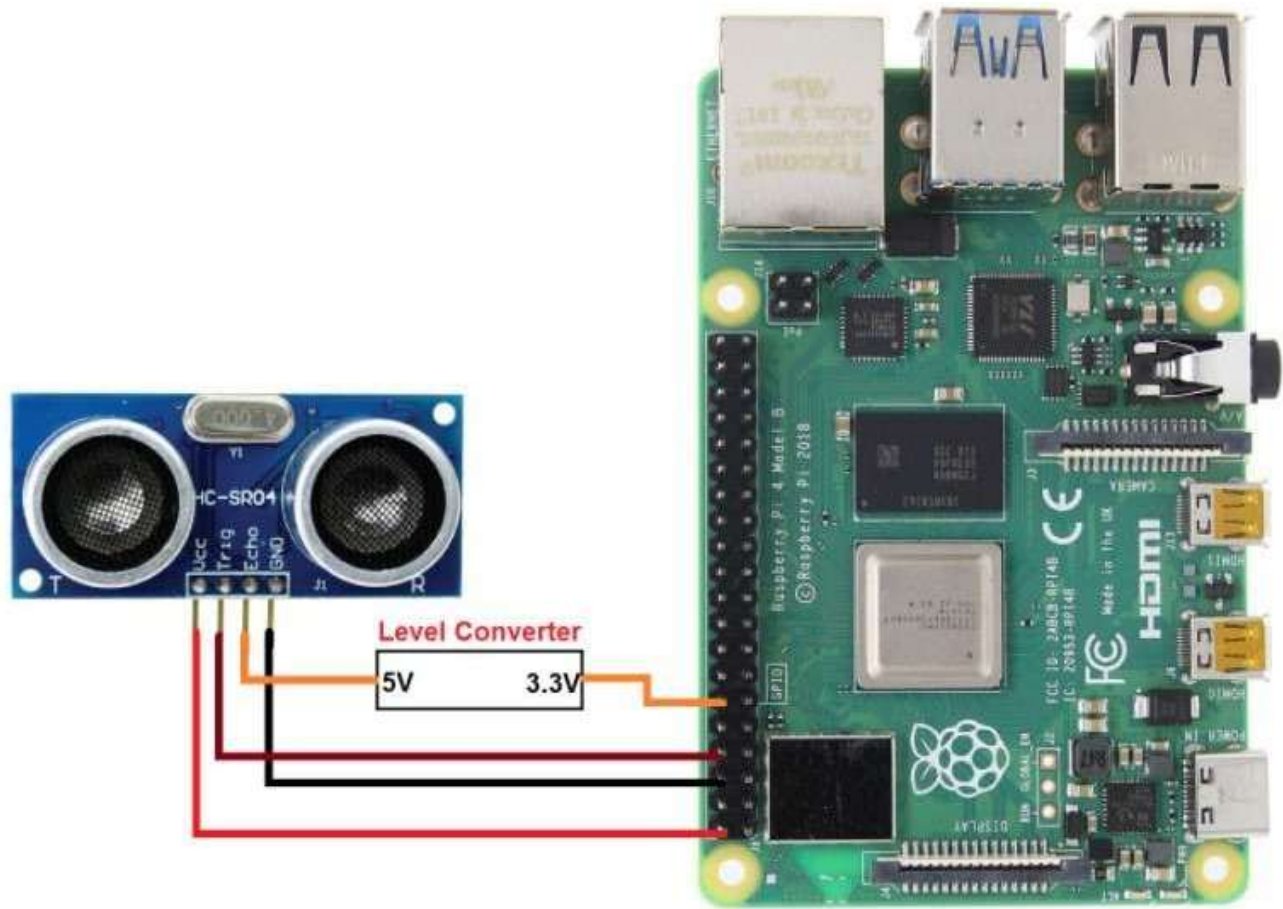
To interface an Ultrasonic Sensor (HC-SR04) with Raspberry Pi and measure the distance of an object using Python programming.

### PROCEDURE:

➤ **Connect the HC-SR04 sensor to Raspberry Pi as follows:**

- VCC → 5V (Pin 2)
- GND → GND (Pin 6)
- TRIG → GPIO23 (Pin 16)
- ECHO → GPIO24 (Pin 18) via voltage divider (to reduce 5V signal to 3.3V)
- The **voltage divider** circuit is formed using two resistors (1 k $\Omega$  and 2 k $\Omega$ ) to step down the ECHO pin output to a safe 3.3V for Raspberry Pi.
- Open the **Thonny IDE** or terminal on the Raspberry Pi.
- Write and execute the Python code for distance measurement.
- Observe the measured distance in the terminal output.

## CIRCUIT DIAGRAM:



## PROGRAM (Arduino – ESP32):

```
# Experiment 5: Interfacing Ultrasonic Sensor HC-SR04 with Raspberry Pi
import RPi.GPIO as GPIO
import time
```

```
# Pin configuration
TRIG = 23 # GPIO23 (Pin 16)
ECHO = 24 # GPIO24 (Pin 18)
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
```

```
print("Ultrasonic Distance Measurement Started...")
```

```
try:
```

```
    while True:
```

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

```
        # Wait for echo start
        while GPIO.input(ECHO) == 0:
            pulse_start = time.time()
```

```
        # Wait for echo end
        while GPIO.input(ECHO) == 1:
            pulse_end = time.time()
```

```
        # Calculate distance
        pulse_duration = pulse_end - pulse_start
        distance = pulse_duration * 17150 # Speed of sound (34300 cm/s ÷ 2)
        distance = round(distance, 2)
        print(f'Distance: {distance} cm')
        time.sleep(1)
```

```
except KeyboardInterrupt:
```

```
    print("Measurement stopped by User")
    GPIO.cleanup()
```

## **EXPECTED OUTPUT:**

Ultrasonic Distance Measurement

Started...

Distance: 15.63 cm

Distance: 16.02 cm

Distance: 15.78 cm

## **RESULT:**

The Ultrasonic Sensor (HC-SR04) successfully measured the distance of objects from the sensor using Raspberry Pi GPIO pins, demonstrating accurate distance sensing and Python-based control.

E.NO: 06

DATE:

## Uploading DHT11 Sensor Data from Raspberry Pi to ThingSpeak Cloud

### AIM:

To interface a DHT11 sensor with Raspberry Pi and upload temperature and humidity data to the ThingSpeak cloud using Python programming.

### PROCEDURE:

#### 1. ThingSpeak Cloud Setup:

- Go to <https://thingspeak.com>.
- Create a free account and sign in.
- Create a New Channel named “DHT11 Sensor Data”.
- Add two fields:
  - Field 1 → Temperature (°C)
  - Field 2 → Humidity (%)
- Save the channel and note the Write API Key from the API Keys tab.

#### 2. Hardware Connections:

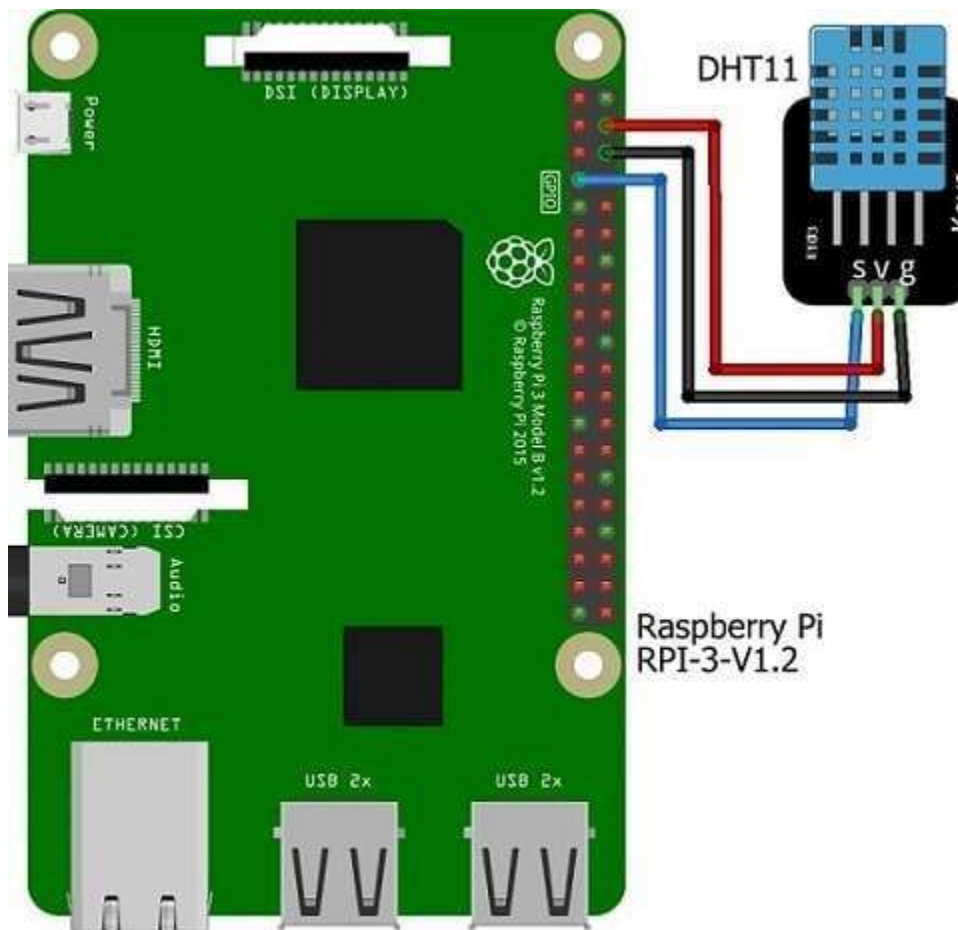
- Connect the DHT11 sensor to Raspberry Pi as follows:
  - VCC → 3.3V (Pin 1)
  - DATA → GPIO4 (Pin 7)
  - GND → GND (Pin 6)

#### 3. Software Setup:

- Open the Thonny IDE or terminal on Raspberry Pi.
- Install required libraries:
- `sudo pip3 install Adafruit_DHT requests`

4. Write the Python program to read data from the DHT11 sensor and upload it to ThingSpeak using HTTP requests.
5. Run the program and observe live updates on the ThingSpeak Cloud dashboard under your created channel.

### CIRCUIT DIAGRAM:



## **PROGRAM (Arduino – ESP32):**

# Experiment 6: Uploading DHT11 Sensor Data from Raspberry Pi to ThingSpeak Cloud

```
import Adafruit_DHT
```

```
import requests
```

```
import time
```

```
# ThingSpeak API configuration
```

```
API_KEY = "YOUR_WRITE_API_KEY" # Replace with your ThingSpeak Write API Key
```

```
BASE_URL = "https://api.thingspeak.com/update?api_key=" + API_KEY
```

```
# DHT11 setup
```

```
DHT_SENSOR = Adafruit_DHT.DHT11
```

```
DHT_PIN = 4 # GPIO4 (Pin 7)
```

```
print("Uploading DHT11 Data to ThingSpeak Cloud...")
```

```
try:
```

```
    while True:
```

```
        # Read temperature and humidity
```

```
        humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN)
```

if humidity is not None and temperature is not None:

    # Prepare the URL with field data

    url = BASE\_URL + "&field1={0:.2f}&field2={1:.2f}".format(temperature,  
humidity)

    # Send data to ThingSpeak

    response = requests.get(url)

    print(f'Data Uploaded → Temp: {temperature:.1f}°C, Humidity:  
{humidity:.1f}% | Response: {response.status\_code}')

else:

    print("Sensor failure. Check connection.")

time.sleep(15) # ThingSpeak allows updates every 15 seconds

except KeyboardInterrupt:

print("Data upload stopped by user.")

### **EXPECTED OUTPUT:**

Uploading DHT11 Data to ThingSpeak

Cloud...

Data Uploaded → Temp: 29.5°C,

Humidity: 62.0% | Response: 200

Data Uploaded → Temp: 29.6°C,

Humidity: 63.1% | Response: 200

### **RESULT:**

The DHT11 sensor data (temperature and humidity) was successfully uploaded to the ThingSpeak Cloud using Raspberry Pi, demonstrating IoT data transmission and visualization on a real-time cloud platform.

E.NO: 07	<b>Monitoring SPO2 and Heart Rate using Arduino and MAX30100 Sensor</b>
DATE:	

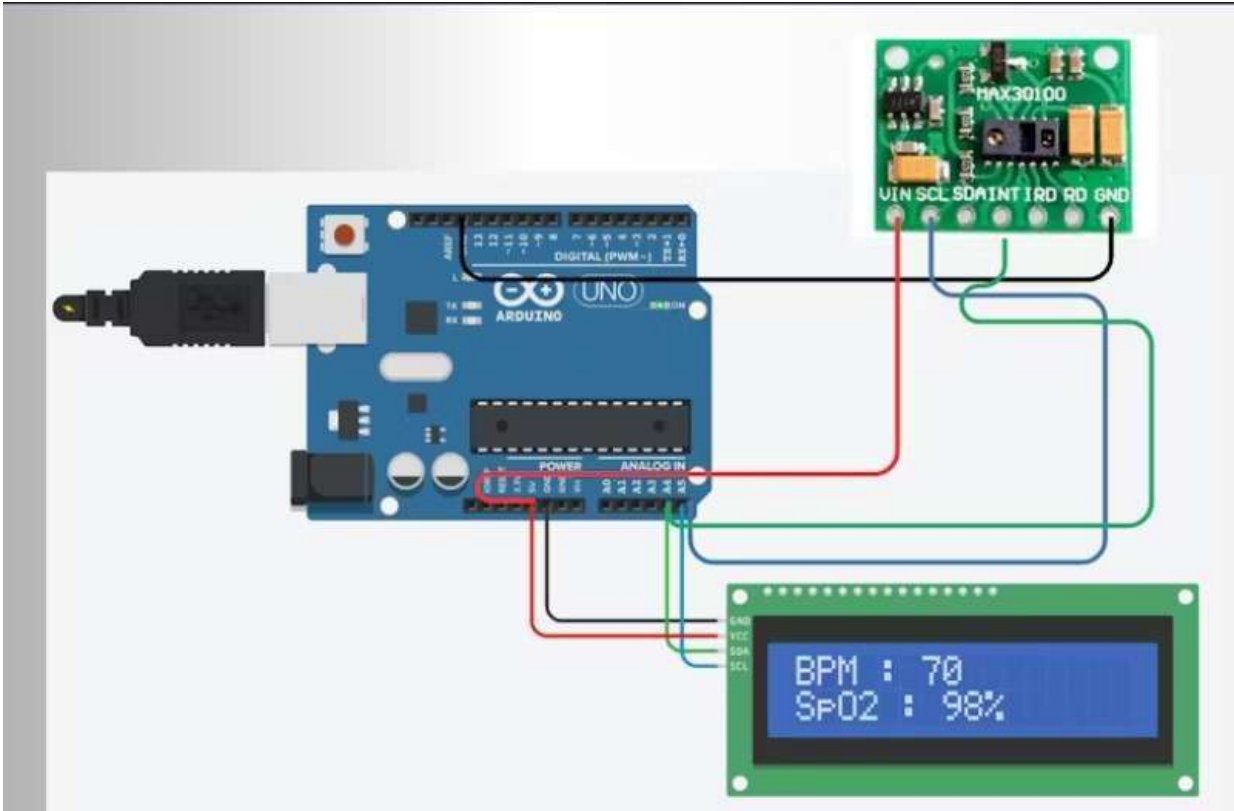
**AIM:**

To interface a MAX30100 pulse oximeter sensor with Arduino to measure and display SpO<sub>2</sub> (oxygen saturation) and heart rate (pulse rate) values.

**PROCEDURE:**

- Connect the MAX30100 sensor to the Arduino UNO as per the above table.
  - Open the Arduino IDE and install the required libraries:
  - Go to Sketch → Include Library → Manage Libraries...
  - Search and install:
    - “MAX30100 Pulse Oximeter” by OXullo Intersecans
    - “Wire” library (if not already installed)
  - Open a new sketch and write/upload the given program.
  - Open the Serial Monitor at 115200 baud rate.
- Place your finger gently on the MAX30100 sensor and observe the readings.

**CIRCUIT DIAGRAM:**



## PROGRAM (Arduino – ESP32):

```
// Experiment 7: Monitoring SpO2 and Heart Rate using Arduino and  
MAX30100
```

```
#include <Wire.h>
```

```
#include "MAX30100_PulseOximeter.h"
```

```
#define REPORTING_PERIOD_MS 1000 // Report every 1 second
```

```
PulseOximeter pox;
```

```
uint32_t tsLastReport = 0;
```

```
// Callback when a beat is detected
```

```
void onBeatDetected() {
```

```
  Serial.println("Beat Detected!");
```

```
}
```

```
void setup() {
```

```
  Serial.begin(115200);
```

```
  Serial.println("Initializing MAX30100 Sensor...");
```

```
  if (!pox.begin()) {
```

```
    Serial.println("FAILED to initialize sensor. Check connections!");
```

```
    for(;;);
```

```
  } else {
```

```
    Serial.println("Sensor initialized successfully!");
```

```
  }
```

```
  pox.setOnBeatDetectedCallback(onBeatDetected);
```

```
}
```

```
void loop() {
```

```
  pox.update();
```

```
  // Print results every second
```

```
  if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
```

```
    Serial.print("Heart Rate (BPM): ");
```

```
Serial.print(pox.getHeartRate());  
Serial.print(" | SpO2 (%): ");  
Serial.println(pox.getSpO2());  
  
    tsLastReport = millis();  
}  
}
```

## **EXPECTED OUTPUT:**

Initializing MAX30100 Sensor...

Sensor initialized successfully!

Beat Detected!

Heart Rate (BPM): 76.0 | SpO2 (%): 98.0

Heart Rate (BPM): 78.5 | SpO2 (%): 97.5

Beat Detected!

Heart Rate (BPM): 77.8 | SpO2 (%): 98.1

## **RESULT:**

The MAX30100 sensor successfully measured and displayed the SpO<sub>2</sub> and heart rate values using Arduino UNO, verifying the accurate working of the pulse oximetry system.

E.NO: 08	<b>Motion Tracking using MPU6050 Sensor and Cloud Logging</b>
DATE:	

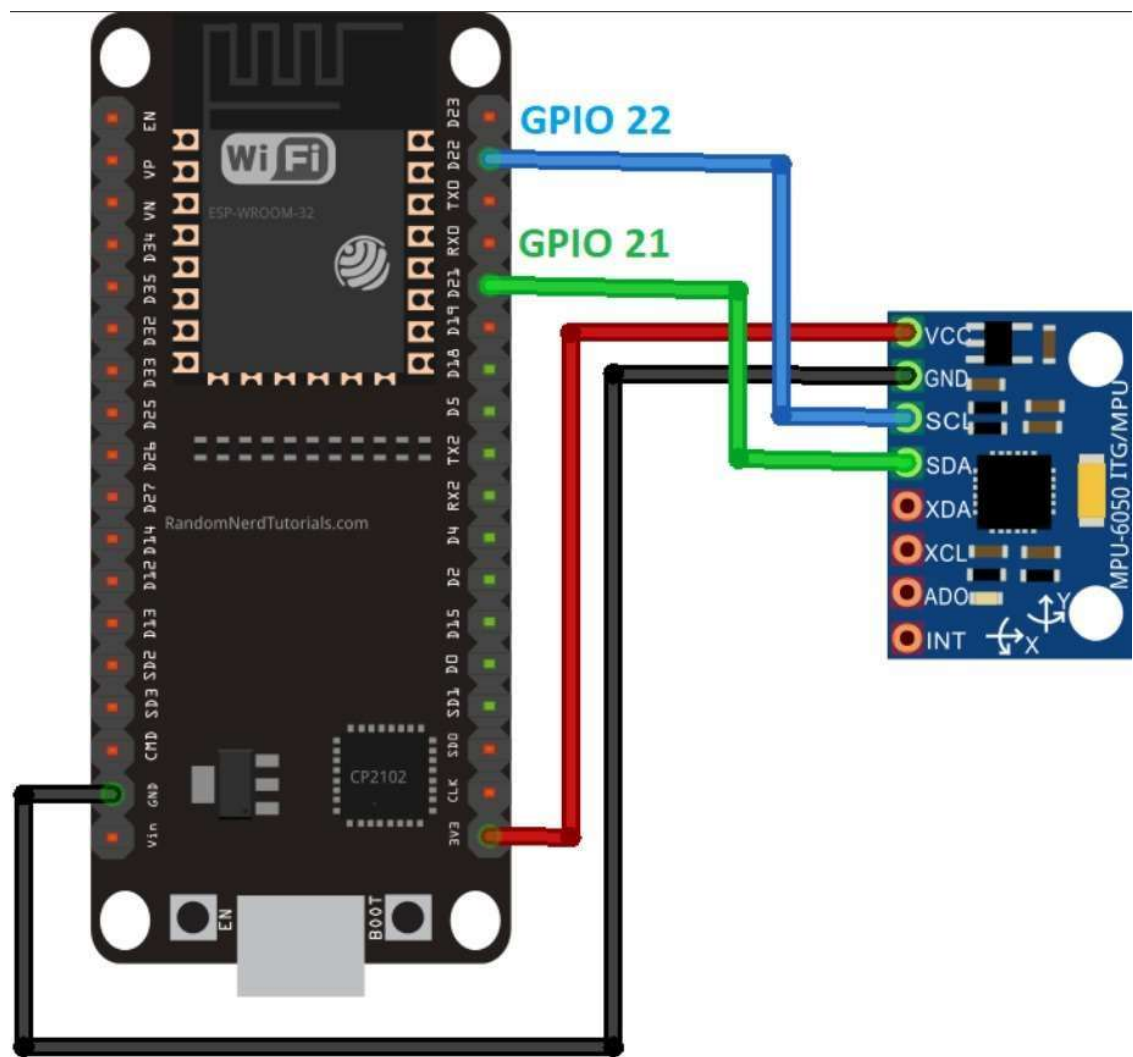
**AIM:**

To interface the MPU6050 sensor with Arduino for detecting motion (acceleration and rotation) and log the sensor data to a cloud platform for real-time monitoring.

**PROCEDURE:**

1. Connect the MPU6050 to the Arduino UNO as per the circuit diagram.
2. Install the necessary libraries in Arduino IDE:
  - Go to Sketch → Include Library → Manage Libraries...
  - Install the following:
    - “MPU6050” by Electronic Cats / Jeff Rowberg
    - “Wire” for I<sup>2</sup>C communication
3. Create a ThingSpeak account at <https://thingspeak.com>.
4. Create a new channel with fields for Accelerometer (X, Y, Z) and Gyroscope (X, Y, Z) data.
5. Note your Write API Key from ThingSpeak’s API Keys tab.
6. Connect ESP8266 to Arduino for Wi-Fi communication.
7. Write and upload the program to read data from MPU6050 and send it to ThingSpeak.
8. Open the Serial Monitor to verify data transmission, and check ThingSpeak dashboard for live updates.

**CIRCUIT DIAGRAM:**



## **PROGRAM (Arduino – ESP32):**

// Experiment 8: Motion Tracking using MPU6050 Sensor and Cloud Logging

```
#include <Wire.h>
```

```
#include <MPU6050.h>
```

```
#include <ESP8266WiFi.h>
```

```
const char* ssid = "YOUR_WIFI_SSID";
```

```
const char* password = "YOUR_WIFI_PASSWORD";
```

```
const char* apiKey = "YOUR_THINGSPEAK_WRITE_API_KEY";
```

```
const char* server = "api.thingspeak.com";
```

```
WiFiClient client;
```

```
MPU6050 mpu;
```

```
void setup() {
```

```
    Serial.begin(115200);
```

```
    Wire.begin();
```

```
    mpu.initialize();
```

```
    Serial.println("Connecting to Wi-Fi...");
```

```
    WiFi.begin(ssid, password);
```

```
    while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(500);
    Serial.print(".");
}
Serial.println("\nWi-Fi connected.");
}

void loop() {
    int16_t ax, ay, az;
    int16_t gx, gy, gz;

    // Read MPU6050 data
    mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    // Print values to Serial Monitor
    Serial.print("Accel (X,Y,Z): ");
    Serial.print(ax); Serial.print(", ");
    Serial.print(ay); Serial.print(", ");
    Serial.println(az);

    Serial.print("Gyro (X,Y,Z): ");
    Serial.print(gx); Serial.print(", ");
    Serial.print(gy); Serial.print(", ");
    Serial.println(gz);
}
```

```
// Send data to ThingSpeak
if (client.connect(server, 80)) {
    String postStr = "api_key=" + String(apiKey) +
        "&field1=" + String(ax) +
        "&field2=" + String(ay) +
        "&field3=" + String(az) +
        "&field4=" + String(gx) +
        "&field5=" + String(gy) +
        "&field6=" + String(gz);

    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: " + String(apiKey) + "\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(postStr.length());
    client.print("\n\n");
    client.print(postStr);
}
client.stop();

delay(20000); // ThingSpeak update interval (20 seconds)
}
```

## **EXPECTED OUTPUT:**

Initializing MAX30100 Sensor...

Sensor initialized successfully!

Beat Detected!

Heart Rate (BPM): 76.0 | SpO2 (%): 98.0

Heart Rate (BPM): 78.5 | SpO2 (%): 97.5

Beat Detected!

Heart Rate (BPM): 77.8 | SpO2 (%): 98.1

## **RESULT:**

The MPU6050 sensor successfully measured acceleration and angular velocity, and the readings were uploaded to the ThingSpeak Cloud for real-time motion tracking and data visualization.

E.NO: 09	Barometric Pressure and Altitude Measurement using BMP Sensor with Arduino
DATE:	

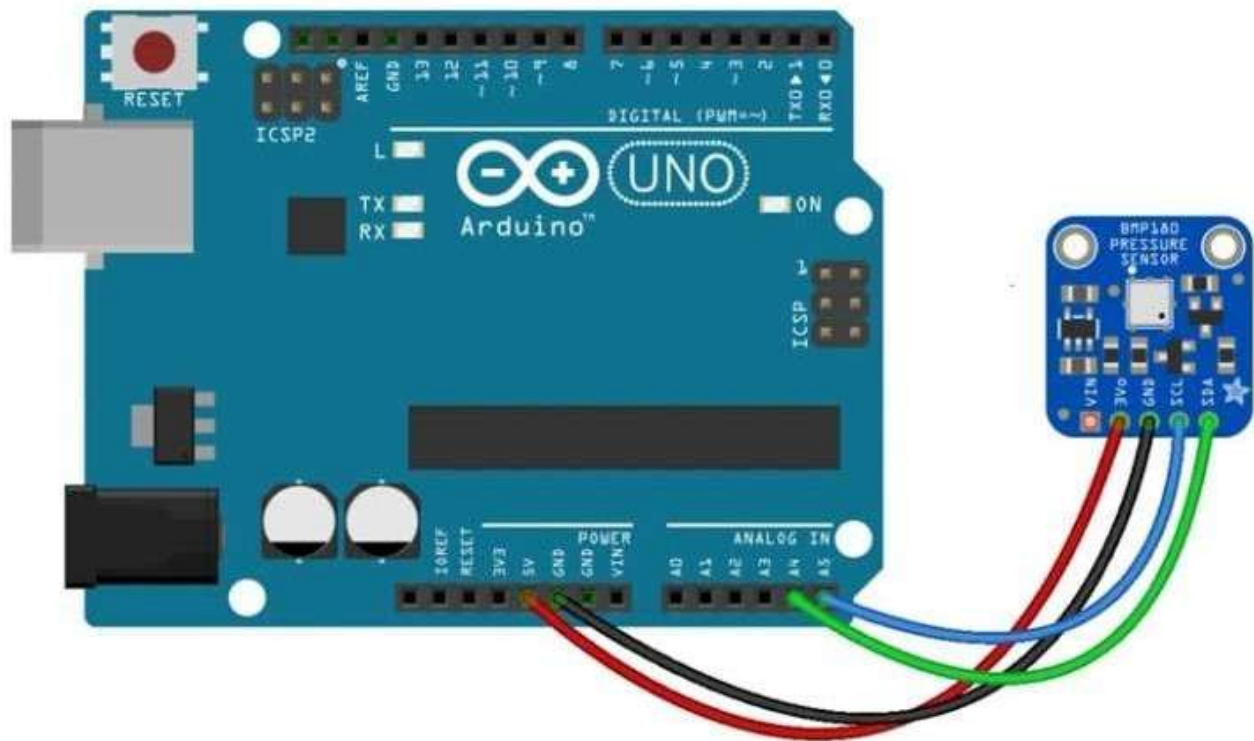
**AIM:**

To interface the BMP180/BMP280 barometric pressure sensor with Arduino to measure atmospheric pressure and altitude using I<sup>2</sup>C communication.

**PROCEDURE:**

- Connect the BMP180/BMP280 sensor to the Arduino UNO as per the table above.
- Open Arduino IDE on your system.
- Go to Sketch → Include Library → Manage Libraries...
- Install the following libraries:
  - “Adafruit BMP085 Unified” or “Adafruit BMP280”
  - “Adafruit Unified Sensor”
- Open a new sketch and write the given program.
- Upload the program to the Arduino UNO board.
- Open the Serial Monitor (baud rate: 9600) to view the real-time pressure, temperature, and altitude readings.

## CIRCUIT DIAGRAM:



## **PROGRAM (Arduino – ESP32):**

// Experiment 9: Barometric Pressure and Altitude Measurement using BMP Sensor with Arduino

```
#include <Wire.h>
```

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

```
#include <Adafruit_BMP085_U.h> // Use Adafruit_BMP280.h for BMP280
```

```
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
```

```
void setup() {
```

```
    Serial.begin(9600);
```

```
    Serial.println("Initializing BMP180 Sensor...");
```

```
    if (!bmp.begin()) {
```

```
        Serial.print("Could not find a valid BMP sensor, check connections!");
```

```
        while (1);
```

```
    }
```

```
}
```

```
void loop() {
```

```
    sensors_event_t event;
```

```
    bmp.getEvent(&event);
```

```
if (event.pressure) {  
    // Display pressure in hPa  
    Serial.print("Pressure: ");  
    Serial.print(event.pressure);  
    Serial.println(" hPa");  
    // Display temperature  
    float temperature;  
    bmp.getTemperature(&temperature);  
    Serial.print("Temperature: ");  
    Serial.print(temperature);  
    Serial.println(" °C");  
    // Calculate altitude  
    float seaLevelPressure = 1013.25; // hPa  
    float altitude = bmp.pressureToAltitude(seaLevelPressure, event.pressure);  
    Serial.print("Altitude: ");  
    Serial.print(altitude);  
    Serial.println(" meters");  
} else {  
    Serial.println("Sensor error!");  
}  
Serial.println("-----");  
delay(2000);  
}
```

## **EXPECTED OUTPUT:**

Initializing BMP180 Sensor...

Pressure: 1008.65 hPa

Temperature: 28.4 °C

Altitude: 39.26 meters

-----

Pressure: 1008.71 hPa

Temperature: 28.5 °C

Altitude: 39.10 meters

...

## **RESULT:**

The BMP180/BMP280 sensor successfully measured the barometric pressure and calculated the corresponding altitude and temperature, verifying proper I<sup>2</sup>C communication and environmental sensing using Arduino.

E.NO: 10	Weather Station using Raspberry Pi with DHT11, BMP Sensor, and Thing Speak Integration
DATE:	

**AIM:**

Weather Station using Raspberry Pi with DHT11, BMP Sensor, and ThingSpeak Integration

**PROCEDURE:**

- Connect the DHT11 and BMP180/BMP280 sensors to the Raspberry Pi as per the above table.
- Power on the Raspberry Pi and open the Thonny IDE or terminal.
- Install the required Python libraries:
- `sudo pip3 install Adafruit_DHT adafruit-circuitpython-bmp280 requests`
- Create a ThingSpeak account at <https://thingspeak.com>.
- Create a new channel with three fields:
- Field 1 → Temperature (°C)
- Field 2 → Humidity (%)
- Field 3 → Pressure (hPa)
- Copy your Write API Key from the API Keys section.
- Write the Python program to read sensor values and upload them to ThingSpeak every 15 seconds.
- Run the program and observe the live graphs on the ThingSpeak dashboard.

## **PROGRAM (Arduino – ESP32):**

# Experiment 10: Weather Station using Raspberry Pi with DHT11, BMP Sensor, and ThingSpeak Integration

```
import Adafruit_DHT
```

```
import board
```

```
import adafruit_bmp280
```

```
import busio
```

```
import requests
```

```
import time
```

```
# ThingSpeak API Configuration
```

```
API_KEY = "YOUR_WRITE_API_KEY" # Replace with your ThingSpeak  
Write API Key
```

```
BASE_URL = "https://api.thingspeak.com/update?api_key=" + API_KEY
```

```
# DHT11 Configuration
```

```
DHT_SENSOR = Adafruit_DHT.DHT11
```

```
DHT_PIN = 4 # GPIO4 (Pin 7)
```

```
# BMP180/BMP280 Configuration
```

```
i2c = busio.I2C(board.SCL, board.SDA)
```

```
bmp = adafruit_bmp280.Adafruit_BMP280_I2C(i2c)
```

```
bmp.sea_level_pressure = 1013.25
```

```
print("Weather Station Data Upload to ThingSpeak Started...")

try:
    while True:

        # Read DHT11 data
        humidity, temperature_dht = Adafruit_DHT.read(DHT_SENSOR,
DHT_PIN)

        # Read BMP data
        pressure = bmp.pressure
        temperature_bmp = bmp.temperature

        # Calculate average temperature from both sensors
        temperature = (temperature_dht + temperature_bmp) / 2

        if humidity is not None and pressure is not None:
            # Send data to ThingSpeak
            url = BASE_URL +
"&field1={0:.2f}&field2={1:.2f}&field3={2:.2f}".format(
                temperature, humidity, pressure)
            response = requests.get(url)

            print(f'Data Uploaded → Temp: {temperature:.2f}°C | Humidity:
{humidity:.2f}% | Pressure: {pressure:.2f} hPa | Response:
{response.status_code}')
        else:
```

```
        print("Sensor reading failed. Check connections.")
        time.sleep(15) # ThingSpeak update interval (minimum 15 sec)
except KeyboardInterrupt:
    print("Program stopped by user.")
```

## **EXPECTED OUTPUT:**

Weather Station Data Upload to

ThingSpeak Started...

Data Uploaded → Temp: 28.75°C |

Humidity: 65.00% | Pressure:

1008.42 hPa | Response: 200

Data Uploaded → Temp: 28.80°C |

Humidity: 64.80% | Pressure:

1008.36 hPa | Response: 200

...

## **RESULT:**

The Raspberry Pi successfully collected environmental data from DHT11 and BMP sensors and uploaded it to the ThingSpeak Cloud, effectively functioning as a real-time IoT-based weather monitoring station.

