Embedded Systems Experiments

Experiment 1:

Measure Analog Signal from DHT 11 Sensor in Arduino UNO

Objective:

To measure and display the analog signal from a DHT 11 sensor using Arduino UNO.

Components Required:

1. Arduino UNO
2. DHT 11 Sensor
3. Jumper Wires
4. Breadboard

Theory:

DHT 11 is a sensor used to measure temperature and humidity. The sensor provides a calibrated digital signal output. The Arduino reads the signal and converts it to a readable format.

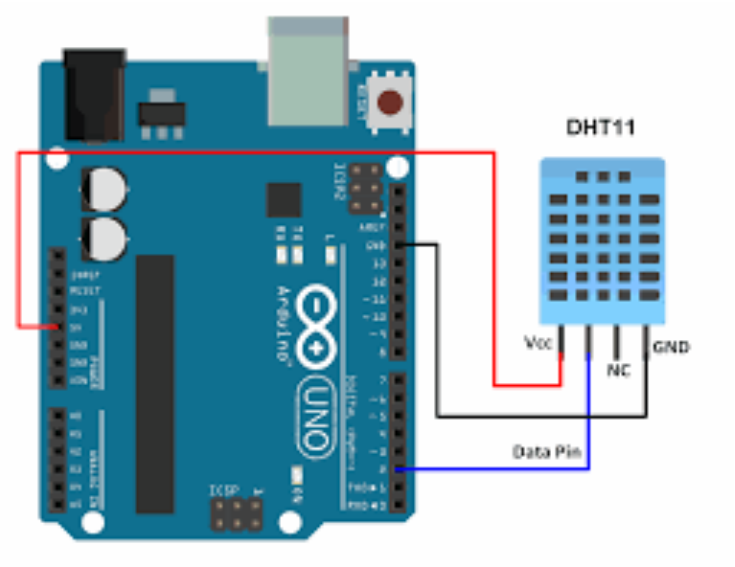
Procedure:

1. Connect the DHT 11 sensor to the Arduino UNO as per the PIN diagram.
2. Write and upload the code to read the sensor data.
3. Display the sensor data on the Serial Monitor.

Connections:

DHT11 - Arduino
VCC - 5V
GND - GND
Data - Digital Pin 2

Circuit Diagram:



Code:

```
#include "DHT11.h"
#define DHT11_PIN 2 // Pin where the data pin of the DHT11 is connected
DHT11 dht(DHT11_PIN); // Create an instance of DHT11

void setup() {
 Serial.begin(9600); // Start the serial communication at 9600 baud rate
}

void loop() {
 int temperatureC = dht.readTemperature(); // Read temperature in Celsius
 int humidity = dht.readHumidity();      // Read humidity

 // Convert temperature from Celsius to Fahrenheit
 float temperatureF = temperatureC * 9.0 / 5.0 + 32.0;

 // Check if there is an error in reading temperature or humidity
 if (temperatureC == DHT11::ERROR_TIMEOUT || temperatureC == DHT11::ERROR_CHECKSUM) {
  Serial.print("Error reading temperature: ");
  Serial.println(DHT11::getErrorString(temperatureC));
 } else {
  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.print(" *C / ");
  Serial.print(temperatureF);
  Serial.println(" *F");
 }

 if (humidity == DHT11::ERROR_TIMEOUT || humidity == DHT11::ERROR_CHECKSUM) {
  Serial.print("Error reading humidity: ");
```

```
  Serial.println(DHT11::getErrorString(humidity));
 } else {
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
 }

 delay(2000); // Wait for 2 seconds before the next reading
}
```

Observation:

Record the temperature and humidity values displayed on the Serial Monitor.

Result:

The temperature and humidity values were successfully measured and displayed on the Serial Monitor.

Embedded Systems & IoT Experiments Manual

Experiment 2:

Generate PWM Output in Arduino UNO Serial Plotter

Objective:

To generate and observe PWM signals on the Serial Plotter using Arduino UNO.

Components Required:

1. Arduino UNO
2. Jumper Wires
3. Breadboard

Theory:

Pulse Width Modulation (PWM) is a technique for getting analog results with digital means. By varying the duty cycle of the PWM signal, you can control the power supplied to electronic components.

Procedure:

1. Connect an LED to a PWM-capable pin on Arduino.
2. Write and upload the code to generate a PWM signal.
3. Observe the PWM signal on the Serial Plotter.

Connections:

LED - Arduino
Anode - PWM Pin 9
Cathode – GND

Code:

```
int pwmPin = 9;  // PWM pin (9, 10, 11, or 3)

void setup() {
 Serial.begin(9600); // Start the serial communication at 9600 baud rate
 pinMode(pwmPin, OUTPUT); // Set the pin as an output
}

void loop() {
 int pwmValue = 128; // 50% duty cycle (0 to 255 range)

 // Generate the PWM signal
 analogWrite(pwmPin, pwmValue);

 // Plot the PWM signal on the Serial Plotter
 if (millis() % 1000 < 500) {
  Serial.println(255);  // High state
 } else {
  Serial.println(0);    // Low state
 }

 delay(10); // Add a small delay to control the update rate on the plotter
}
```

Observation:

Observe the PWM signal waveform on the Serial Plotter.

Result:

The PWM signal was successfully generated and observed on the Serial Plotter.

Embedded Systems & IoT Experiments Manual

Experiment 3:

Drive Single Character Generation on Hyper Terminal in Arduino UNO

Objective:

To drive a single character to be displayed on the Hyper Terminal using Arduino UNO.

Components Required:

1. Arduino UNO
2. USB Cable
3. Computer with Hyper Terminal software

Theory:

The Hyper Terminal is a communication program that connects to devices via a serial port. Arduino UNO can send characters through its serial port to be displayed on the Hyper Terminal.

Procedure:

1. Connect the Arduino UNO to the computer using the USB cable.
2. Open Hyper Terminal on the computer and set up the serial communication settings.
3. Write and upload code to the Arduino to send a character.
4. Observe the character displayed on the Hyper Terminal.

Connections:

Arduino UNO - Computer
TX - RX (via USB)

Code:

```
char characterToSend = 'A';  // The character to send


void setup() {
 Serial.begin(9600);  // Initialize serial communication at 9600 baud
}


void loop() {
 Serial.println(characterToSend);  // Send the character to HyperTerminal
 delay(1000);  // Wait for 1 second before sending the next character
}
```

Observation:

Check the Hyper Terminal for the received character.

Result:

The single character was successfully sent from the Arduino UNO and displayed on the Hyper Terminal.

Embedded Systems & IoT Experiments Manual

Experiment 4:

Drive a Given String on Hyper Terminal in Arduino UNO

Objective:

To send and display a string of characters on the Hyper Terminal using Arduino UNO.

Components Required:

1. Arduino UNO
2. USB Cable
3. Computer with Hyper Terminal software

Theory:

Arduino UNO can send a string of characters via its serial port. The string can be observed on the Hyper Terminal, which is set up to receive data from the Arduino.

Procedure:

1. Connect the Arduino UNO to the computer using the USB cable.
2. Open Hyper Terminal on the computer and set up the serial communication settings.
3. Write and upload code to the Arduino to send a string of characters.
4. Observe the string displayed on the Hyper Terminal.

Connections:

Arduino UNO - Computer

TX - RX (via USB)

Code:

```
String stringToSend = "Hello, SVEC!";  // The string to send

void setup() {
  Serial.begin(9600);  // Initialize serial communication at 9600 baud
}

void loop() {
  Serial.println(stringToSend);  // Send the string to HyperTerminal
  delay(1000);  // Wait for 1 second before sending the string again
}
```


Observation:

Check the Hyper Terminal for the received string of characters.

Result:

The string was successfully sent from the Arduino UNO and displayed on the Hyper Terminal.

Embedded Systems & IoT Experiments Manual

Experiment 5:

Full Duplex Link Establishment using Hyper Terminal in Arduino UNO

Objective:

To establish a full duplex communication link between Arduino UNO and a computer using Hyper Terminal.

Components Required:

1. Arduino UNO
2. USB Cable
3. Computer with Hyper Terminal software

Theory:

Full duplex communication allows data to be sent and received simultaneously. The Arduino UNO can communicate with the Hyper Terminal in both directions.

Procedure:

1. Connect the Arduino UNO to the computer using the USB cable.
2. Open Hyper Terminal on the computer and set up the serial communication settings.
3. Write and upload code to the Arduino to send and receive data.
4. Test full duplex communication by sending data from both ends.

Connections:

Arduino UNO - Computer
TX - RX (via USB)

Code:


```
void setup() {
  Serial.begin(9600);  // Initialize serial communication at 9600 baud
  Serial.println("Full Duplex Communication Started."); // Initial message
}

void loop() {
  // Check if data is available to read from HyperTerminal
  if (Serial.available() > 0) {
    // Read the incoming data byte
    char incomingByte = Serial.read();

    // Echo the received byte back to HyperTerminal
    Serial.print("Received: ");
    Serial.println(incomingByte);

    // Optionally, you can add logic here to process the incoming data
```

```
  // For example, respond with a predefined message:
  if (incomingByte == 'A') {
    Serial.println("You sent 'A'");
  } else if (incomingByte == 'B') {
    Serial.println("You sent 'B'");
  } else {
    Serial.println("Unknown character received.");
  }
 }

 // Send a periodic message to HyperTerminal
 Serial.println("Sending periodic message...");
 delay(2000);  // Delay of 2 seconds before sending the next message
}
```

Observation:

Verify the simultaneous sending and receiving of data on the Hyper Terminal.

Result:

Full duplex communication was successfully established between the Arduino UNO and Hyper Terminal.

Embedded Systems & IoT Experiments Manual

Experiment 6:

Drive a Given Value on an 8-bit DAC consisting of SPI in Arduino UNO

Objective:

To drive an 8-bit digital value to a DAC using SPI communication in Arduino UNO.

Components Required:

1. Arduino UNO
2. 8-bit DAC module
3. Jumper Wires
4. Breadboard

Theory:

A Digital-to-Analog Converter (DAC) converts digital data into an analog signal. Arduino UNO can send digital data via the SPI protocol to drive the DAC.

Procedure:

1. Connect the 8-bit DAC to the Arduino UNO as per the PIN diagram.
2. Write and upload code to send an 8-bit value to the DAC using SPI.
3. Observe the analog output on the DAC.

Connections:

DAC - Arduino
VCC - 5V
GND - GND
SCK - Digital Pin 13
MOSI - Digital Pin 11
SS - Digital Pin 10

Observation:

Measure the analog output corresponding to the digital value sent.

Result:

The digital value was successfully converted to an analog signal by the DAC.

Embedded Systems & IoT Experiments Manual

Experiment 7:

Drive SG90 Motor using Analog GPIOs in Arduino UNO

Objective:

To control the position of an SG90 servo motor using the PWM output from Arduino UNO.

Components Required:

1. Arduino UNO
2. SG90 Servo Motor
3. Jumper Wires
4. Breadboard

Theory:

The SG90 is a small servo motor that can be controlled using PWM signals. The Arduino UNO can generate the PWM signal to control the motor's position.

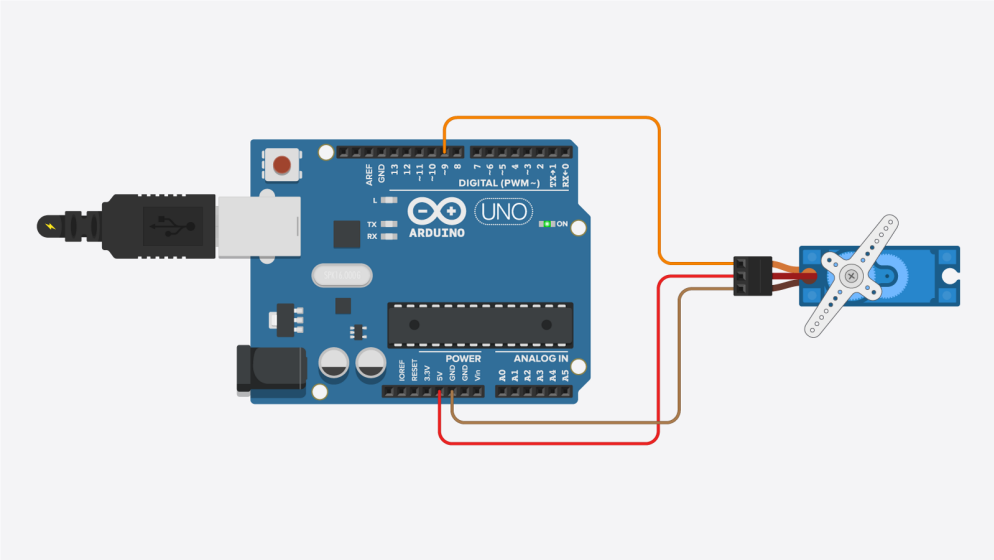
Procedure:

1. Connect the SG90 servo motor to the Arduino UNO as per the PIN diagram.
2. Write and upload code to generate PWM signals to control the servo.
3. Observe the movement of the servo motor.

Connections:

SG90 - Arduino
VCC - 5V
GND - GND
Signal - PWM Pin 9

Circuit Diagram:



Code:

```
#include <Servo.h>

Servo myServo;  // Create a servo object

int servoPin = 9;  // PWM pin to control the servo

void setup() {
 myServo.attach(servoPin);  // Attach the servo on pin 9 to the servo object
}

void loop() {
 // Sweep from 0 to 180 degrees
 for (int angle = 0; angle <= 180; angle++) {
  myServo.write(angle);  // Set the servo to the current angle
  delay(15);  // Delay for smooth motion
 }

 delay(1000);  // Pause for 1 second

 // Sweep from 180 to 0 degrees
 for (int angle = 180; angle >= 0; angle--) {
  myServo.write(angle);  // Set the servo to the current angle
  delay(15);  // Delay for smooth motion
 }

 delay(1000);  // Pause for 1 second
}
```


Observation:

Observe the position of the servo motor as it changes according to the PWM signal.

Result:

The SG90 servo motor was successfully controlled using the Arduino UNO.

Embedded Systems & IoT Experiments Manual

Experiment 8:

Drive Accelerometer and Display the Readings on Hyper Terminal

Objective:

To interface an accelerometer with Arduino UNO and display the readings on Hyper Terminal.

Components Required:

1. Arduino UNO
2. Accelerometer Module
3. Jumper Wires
4. Breadboard

Theory:

An accelerometer measures acceleration along one or more axes. The Arduino UNO can read these measurements and send them to Hyper Terminal for display.

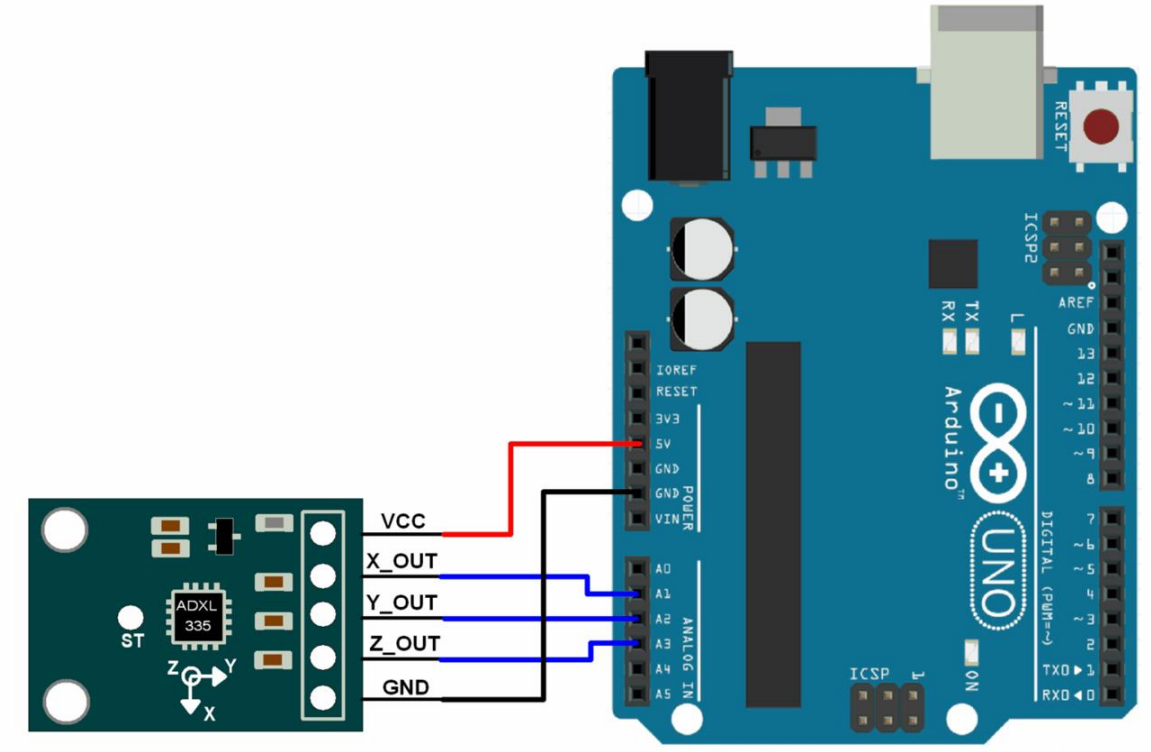
Procedure:

1. Connect the accelerometer to the Arduino UNO as per the PIN diagram.
2. Write and upload code to read the accelerometer data.
3. Send the accelerometer data to the Hyper Terminal and observe the readings.

Connections:

Accelerometer - Arduino
VCC - 3.3V
GND - GND
X-Axis - Analog Pin A0
Y-Axis - Analog Pin A1
Z-Axis - Analog Pin A2

Circuit Diagram:



Code:

```
const int xPin = A0;  // X-axis output to analog pin A0
const int yPin = A1;  // Y-axis output to analog pin A1
const int zPin = A2;  // Z-axis output to analog pin A2

void setup() {
 Serial.begin(9600);  // Initialize serial communication at 9600 baud
 Serial.println("ADXL335 Accelerometer Readings:");
}

void loop() {
 // Read the analog values from the accelerometer
 int xValue = analogRead(xPin);
 int yValue = analogRead(yPin);
 int zValue = analogRead(zPin);

 // Convert the analog readings to voltage (assuming 5V reference)
 float xVoltage = (xValue / 1023.0) * 5.0;
 float yVoltage = (yValue / 1023.0) * 5.0;
 float zVoltage = (zValue / 1023.0) * 5.0;

 // Display the readings on the serial monitor/HyperTerminal
```

```
  Serial.print("X Voltage: ");
  Serial.print(xVoltage, 2);
  Serial.print(" V, Y Voltage: ");
  Serial.print(yVoltage, 2);
  Serial.print(" V, Z Voltage: ");
  Serial.println(zVoltage, 2);

  // Small delay before the next reading
  delay(500);
}
```

Observation:

Observe the accelerometer readings displayed on the Hyper Terminal.

Result:

The accelerometer data was successfully read and displayed on the Hyper Terminal.

Internet of Things Experiments

Experiment 1:

Getting started with Raspberry Pi, Install Raspbian on your SD card

Objective:

To install the Raspbian OS on an SD card and set up Raspberry Pi for the first time.

Components Required:

1. Raspberry Pi
2. SD Card (8GB or larger)
3. Computer with SD Card Reader
4. Raspbian OS image

Theory:

Raspberry Pi is a small, affordable computer used for various IoT projects. Installing the Raspbian OS on an SD card is the first step to setting up the Raspberry Pi.

Procedure:

1. Download the Raspbian OS image from the official Raspberry Pi website.
2. Use an image writing tool to write the Raspbian OS image to the SD card.
3. Insert the SD card into the Raspberry Pi and power it on.
4. Follow the on-screen instructions to complete the setup.

Connections:

No specific PIN diagram is required for this setup as it involves setting up the Raspberry Pi with a pre-configured OS.

Observation:

Verify that the Raspberry Pi boots up and the Raspbian OS is running successfully.

Result:

The Raspbian OS was successfully installed on the SD card, and the Raspberry Pi booted up correctly.

Embedded Systems & IoT Experiments Manual

Experiment 2 & 3:

Python-based IDE for the Raspberry Pi and how to trace and debug Python code

Objective:

To install a Python-based IDE on Raspberry Pi and learn how to trace and debug Python code.

Components Required:

1. Raspberry Pi
2. Raspbian OS installed
3. Internet connection

Theory:

A Python-based IDE such as Thonny or VS Code can be installed on Raspberry Pi for Python development. These IDEs provide tools for writing, tracing, and debugging Python code.

Procedure:

1. Install the Python IDE (e.g., Thonny) via the terminal using 'sudo apt-get install thonny'.
2. Launch the IDE and open a Python script.
3. Use breakpoints and step-through debugging features to trace and debug the code.

Connections:

No specific PIN diagram is required as this is a software-based experiment.

Code:

```
import logging

# Set up basic logging configuration
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

def factorial(n):
    logging.debug(f'Starting factorial calculation for n={n}')
    if n == 1:
        logging.debug(f'Base case reached with n={n}')
        return 1
    else:
        result = n * factorial(n - 1)
        logging.debug(f'Computed factorial({n}) = {result}')
        return result

if __name__ == "__main__":
    number = 5  # You can change this number to test other values
    logging.info(f'Calculating factorial for {number}')
    result = factorial(number)
```

```
    logging.info(f'Factorial of {number} is {result}')
    print(f'Factorial of {number} is {result}')
```

Observation:

Observe the code execution flow using breakpoints and the debugger.

Result:

The IDE was successfully installed, and Python code was traced and debugged.

Embedded Systems & IoT Experiments Manual

Experiment 4a:

Calculate the Distance using Ultrasonic Sensor and Raspberry Pi

Objective:

To calculate the distance to an object using an Ultrasonic Sensor connected to a Raspberry Pi.

Components Required:

1. Raspberry Pi
2. Ultrasonic Sensor (HC-SR04)
3. Jumper Wires
4. Breadboard

Theory:

The Ultrasonic Sensor measures the distance to an object by emitting ultrasonic waves and measuring the time taken for the echo to return. The Raspberry Pi processes this data to calculate the distance.

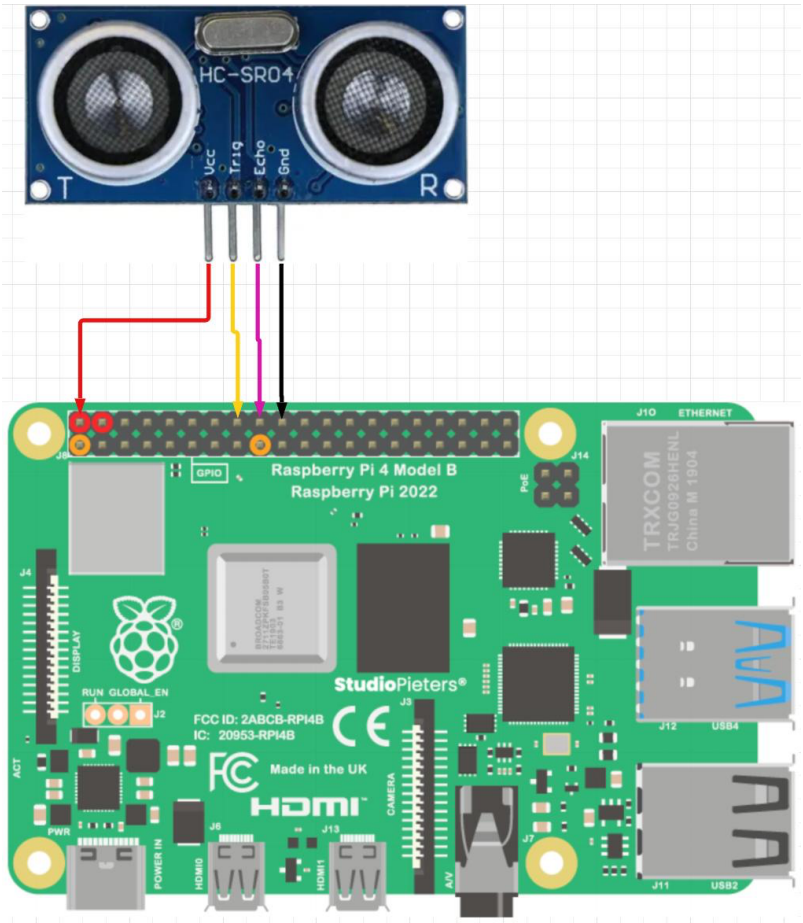
Procedure:

1. Connect the Ultrasonic Sensor to the Raspberry Pi as per the PIN diagram.
2. Write and execute the Python code to read data from the Ultrasonic Sensor.
3. Observe the distance calculation on the Raspberry Pi terminal.

Connections:

HC-SR04 - Raspberry Pi
VCC - 5V
GND - GND
Trig - GPIO 23
Echo - GPIO 24

Circuit Diagram:



Code:

```
import RPi.GPIO as GPIO
import time

# Set up the GPIO pins
GPIO.setmode(GPIO.BCM)
TRIG = 23
ECHO = 24

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

def measure_distance():
    # Trigger the sensor
    GPIO.output(TRIG, True)
    time.sleep(0.00001)  # Trigger pulse of 10µs
```

```
    GPIO.output(TRIG, False)

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

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

    # Calculate the distance
    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150  # Speed of sound is 34300 cm/s (17150 cm/s one-way)
    distance = round(distance, 2)
    return distance

try:
    while True:
        dist = measure_distance()
        print(f"Distance: {dist} cm")
        time.sleep(1)  # Delay of 1 second between measurements
except KeyboardInterrupt:
    print("Measurement stopped by user")
    GPIO.cleanup()  # Reset GPIO settings
```

Observation:

Observe the distance displayed in the terminal output.

Result:

The distance to the object was successfully calculated and displayed on the Raspberry Pi terminal.

Embedded Systems & IoT Experiments Manual

Experiment 4b:

Basic LED Blinking Functionality Using Raspberry Pi

Objective:

To control an LED connected to Raspberry Pi and make it blink on and off at regular intervals.

Components Required:

1. Raspberry Pi
2. LED
3. Resistor (220Ω)
4. Jumper Wires
5. Breadboard

Theory:

Blinking an LED is a basic experiment to understand GPIO pin control in Raspberry Pi. The GPIO pins are programmed to output a HIGH or LOW signal to turn the LED on or off.

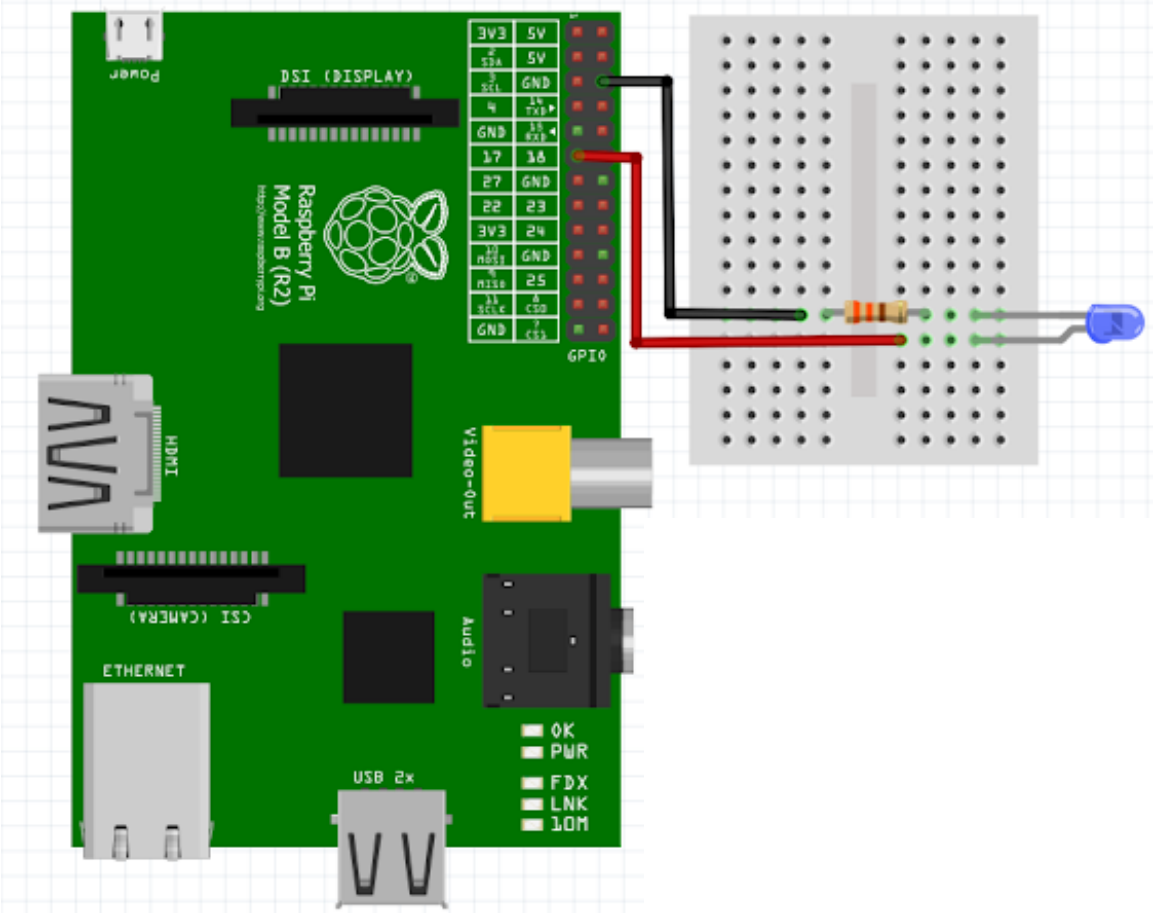
Procedure:

1. Connect the LED and resistor to the Raspberry Pi as per the PIN diagram.
2. Write a Python script to toggle the GPIO pin connected to the LED.
3. Run the script and observe the LED blinking.

Connections:

LED - Raspberry Pi
Anode - GPIO 17 (with 220Ω resistor)
Cathode – GND

Circuit Diagram:



Code:

```
import RPi.GPIO as GPIO
import time

# Set up the GPIO pin
GPIO.setmode(GPIO.BCM)
LED_PIN = 17
GPIO.setup(LED_PIN, GPIO.OUT)

try:
   while True:
      GPIO.output(LED_PIN, GPIO.HIGH)  # Turn on LED
      time.sleep(1)  # Wait for 1 second
      GPIO.output(LED_PIN, GPIO.LOW)   # Turn off LED
      time.sleep(1)  # Wait for 1 second
except KeyboardInterrupt:
```

```
print("Blinking stopped by user")
GPIO.cleanup()  # Reset GPIO settings
```

Observation:

Observe the LED blinking on and off at the specified intervals.

Result:

The LED successfully blinked at regular intervals as per the script.

Embedded Systems & IoT Experiments Manual

Experiment 5:

Raspberry Pi interacting with online services through public APIs

Objective:

To use public APIs to interact with online services using a Raspberry Pi.

Components Required:

1. Raspberry Pi
2. Internet connection
3. Python with requests library

Theory:

Public APIs allow you to retrieve data from online services such as weather, location, etc. Using Python's requests library, Raspberry Pi can send HTTP requests to these APIs and process the responses.

Procedure:

1. Choose a public API (e.g., OpenWeather API, Zomato API).
2. Write a Python script to send a GET request to the API.
3. Process the JSON response to extract and display the relevant information.

Connections:

No specific PIN diagram is required as this is a software-based experiment.

Code:

```
import requests

url = "https://www.weatherunion.com/gw/weather/external/v0/get_locality_weather_data"

querystring = {"locality_id":"ZWL008599"}

headers = {"X-Zomato-Api-Key": "19d4219e1b6ad2c7b82708c802622a8e"}

response = requests.get(url, headers=headers, params=querystring)

#print(response.json())
data = response.json()

if data.get("status") == '200':
    # Extract the weather data
    weather_data = data.get("locality_weather_data", {})

    # Assign variables for each piece of data
    temperature = weather_data.get("temperature")
    humidity = weather_data.get("humidity")
```

```
    wind_speed = weather_data.get("wind_speed")
    wind_direction = weather_data.get("wind_direction")
    rain_intensity = weather_data.get("rain_intensity")
    rain_accumulation = weather_data.get("rain_accumulation")

    # Print the data in a nicely formatted way
    print("Weather Data for the Given Location: Miyapur")
    print(f"Temperature: {temperature} C")
    print(f"Humidity: {humidity}%")
    print(f"Wind Speed: {wind_speed} m/s")
    print(f"Wind Direction: {wind_direction}")
    print(f"Rain Intensity: {rain_intensity} mm/h")
    print(f"Rain Accumulation: {rain_accumulation} mm")
else:
    print("Failed to retrieve weather data.")
```

Observation:

Check the terminal or script output for the data retrieved from the API.

Result:

The Raspberry Pi successfully retrieved and processed data from the public API.

Embedded Systems & IoT Experiments Manual

Experiment 6:

Study and Install IDE of Arduino and Different Types of Arduino

Objective:

To install the Arduino IDE and study the various types of Arduino boards available.

Components Required:

1. Computer with Internet connection
2. Arduino UNO (for testing)
3. USB Cable

Theory:

The Arduino IDE is an open-source software that makes it easy to write code and upload it to the Arduino board. There are different types of Arduino boards such as Arduino UNO, Nano, Mega, each serving different purposes.

Procedure:

1. Download and install the Arduino IDE from the official Arduino website.
2. Explore the various board options available in the 'Tools' menu.
3. Connect an Arduino UNO to the computer and upload a basic 'Blink' sketch.

Connections:

No specific PIN diagram is required as this experiment focuses on software and understanding board types.

Observation:

Observe the LED blinking on the Arduino UNO as per the uploaded sketch.

Result:

The Arduino IDE was successfully installed, and a basic sketch was uploaded to the Arduino UNO.

Embedded Systems & IoT Experiments Manual

Experiment 7:

Study and Implement Wireless Communication using HC-12 Modules with Arduino UNO

Objective:

To study and implement wireless communication between two Arduino UNO boards using HC-12 modules.

Components Required:

1. 2 x Arduino UNO
2. 2 x HC-12 Wireless Communication Modules
3. 2 x USB Cables
4. Jumper Wires
5. Breadboards

Theory:

HC-12 is a wireless communication module that operates in the 433 MHz band. It is commonly used for short-distance wireless communication. In this experiment, two Arduino UNO boards will communicate wirelessly using HC-12 modules.

Procedure:

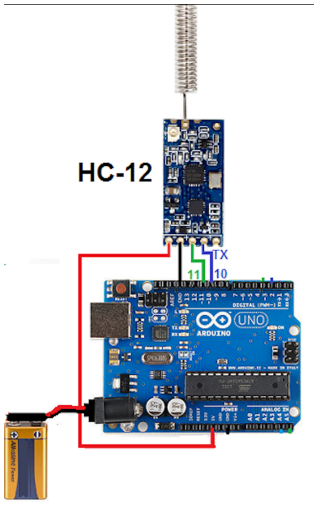
1. Connect each HC-12 module to an Arduino UNO as per the PIN diagram.
2. Write and upload the code to the first Arduino UNO for transmitting data.
3. Write and upload the code to the second Arduino UNO for receiving data.
4. Power both Arduino boards and observe the data transmission and reception.

Connections:

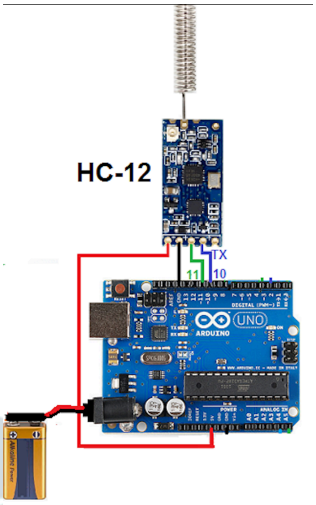
Connections for Transmitter:
HC-12 - Arduino UNO
VCC - 5V
GND - GND
TX - Digital Pin 2
RX - Digital Pin 3

Connections for Receiver:
HC-12 - Arduino UNO
VCC - 5V
GND - GND
TX - Digital Pin 2
RX - Digital Pin 3

Circuit Diagram:



Transmitter



Receiver

Code:

- **Transmitter**

```
#include <SoftwareSerial.h>

// Define the HC-12 serial pins
SoftwareSerial HC12(10, 11); // RX, TX

void setup() {
  Serial.begin(9600);      // Serial monitor communication
  HC12.begin(9600);        // HC-12 communication

  Serial.println("HC-12 Transmitter Ready");
  Serial.println("Type your message and press Enter to send:");
}

void loop() {
  if (Serial.available()) {
    String dataToSend = Serial.readStringUntil('\n'); // Read input from Serial Monitor until newline

    HC12.println(dataToSend); // Send the data to the receiver

    Serial.println("Sent: " + dataToSend); // Print to Serial monitor
  }
}
```

➢ **Receiver**

```
#include <SoftwareSerial.h>

// Define the HC-12 serial pins
SoftwareSerial HC12(10, 11); // RX, TX

void setup() {
  Serial.begin(9600);      // Serial monitor communication
  HC12.begin(9600);        // HC-12 communication

  Serial.println("HC-12 Receiver Ready");
}

void loop() {
  if (HC12.available()) {
    String receivedData = HC12.readString(); // Read the incoming data

    Serial.println("Received: " + receivedData); // Print the received data to Serial monitor
  }
}
```

Observation:

Check the Serial Monitor on the receiving Arduino to observe the received data.

Result:

Wireless communication was successfully established between the two Arduino UNO boards using HC-12 modules.

Embedded Systems & IoT Experiments Manual

Experiment 8:

Calculate the Distance using Ultrasonic Sensor using Arduino UNO

Objective:

To calculate the distance to an object using an Ultrasonic Sensor connected to an Arduino UNO.

Components Required:

1. Arduino UNO
2. Ultrasonic Sensor (HC-SR04)
3. Jumper Wires
4. Breadboard

Theory:

The Ultrasonic Sensor measures the distance to an object by emitting ultrasonic waves and measuring the time taken for the echo to return. The Arduino UNO processes this data to calculate the distance.

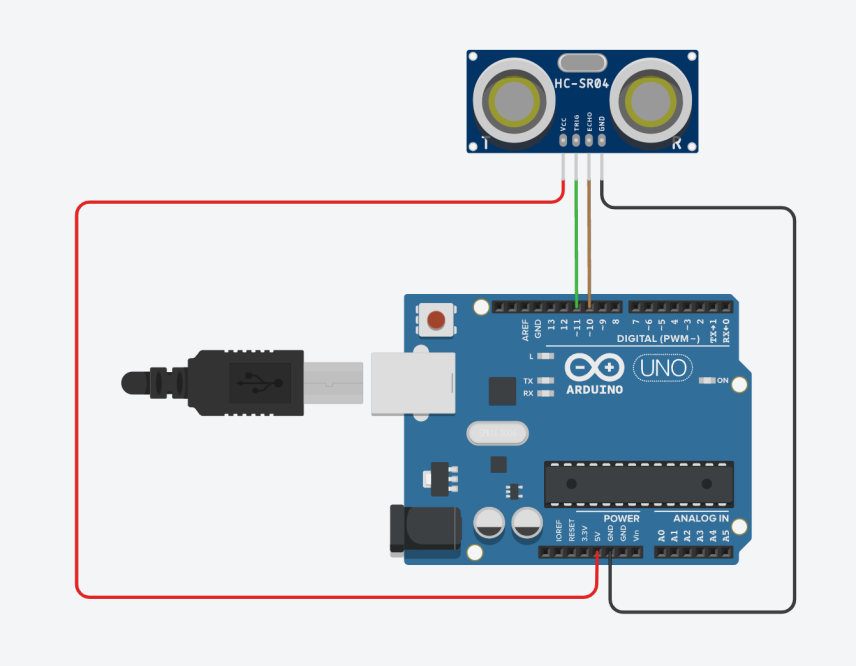
Procedure:

1. Connect the Ultrasonic Sensor to the Arduino UNO as per the PIN diagram.
2. Write and upload the code to read data from the Ultrasonic Sensor.
3. Observe the distance calculation on the Serial Monitor.

Connections:

HC-SR04 - Arduino UNO
VCC - 5V
GND - GND
Trig - Digital Pin 11
Echo - Digital Pin 10

Circuit Diagram:



Code:

```
#define trigPin 11
#define echoPin 10

void setup() {
 Serial.begin(9600);     // Start the serial communication
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
}

void loop() {
 long duration;
 float distance;

 // Clear the trigPin by setting it LOW
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);

 // Trigger the sensor by setting the trigPin HIGH for 10 microseconds
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);

 // Read the echoPin, returns the sound wave travel time in microseconds
 duration = pulseIn(echoPin, HIGH);
```

```
  // Calculate the distance (in centimeters)
  distance = duration * 0.034 / 2;

  // Print the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(1000); // Wait 1 second before the next measurement
}
```

Observation:

Observe the distance displayed in the Serial Monitor.

Result:

The distance to the object was successfully calculated and displayed on the Serial Monitor.

Embedded Systems & IoT Experiments Manual

Experiment 9:

Basic LED Blinking Functionality using Arduino UNO

Objective:

To control an LED connected to Arduino UNO and make it blink on and off at regular intervals.

Components Required:

1. Arduino UNO
2. LED
3. Resistor (220Ω)
4. Jumper Wires
5. Breadboard

Theory:

Blinking an LED is a basic experiment to understand GPIO pin control in Arduino UNO. The GPIO pins are programmed to output a HIGH or LOW signal to turn the LED on or off.

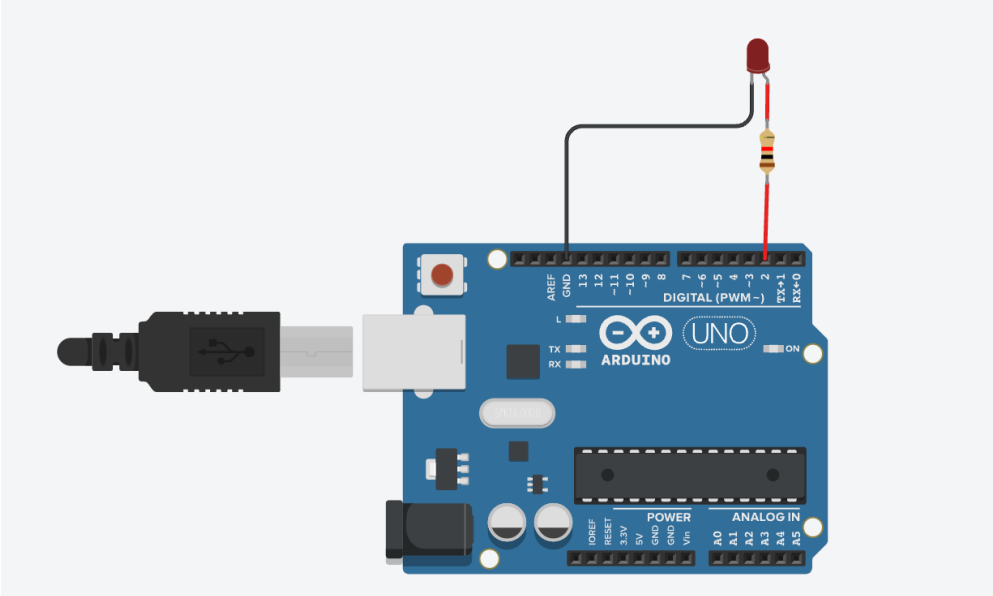
Procedure:

1. Connect the LED and resistor to the Arduino UNO as per the PIN diagram.
2. Write and upload a sketch to toggle the GPIO pin connected to the LED.
3. Run the sketch and observe the LED blinking.

Connections:

LED - Arduino UNO
Anode - Digital Pin 2 (with 220Ω resistor)
Cathode – GND

Circuit Diagram:



Code:

```
const int ledPin = 2; // Define the pin connected to the LED

void setup() {
 pinMode(ledPin, OUTPUT);   // Set the LED pin as an output
 Serial.begin(9600);      // Start serial communication at 9600 baud rate
 Serial.println("Type 'on' to turn the LED on, 'off' to turn it off, or 'blink' to blink.");
}

void loop() {
 if (Serial.available()) {          // Check if data is available on the Serial Monitor
  String command = Serial.readStringUntil('\n'); // Read the input as a string until a newline

  command.trim();                  // Remove any extra white spaces

  if (command.equalsIgnoreCase("on")) {
   digitalWrite(ledPin, HIGH);      // Turn the LED on
   Serial.println("LED is ON");
  }
  else if (command.equalsIgnoreCase("off")) {
   digitalWrite(ledPin, LOW);       // Turn the LED off
   Serial.println("LED is OFF");
  }
  else if (command.equalsIgnoreCase("blink")) {
   Serial.println("LED is blinking...");
   for (int i = 0; i < 5; i++) {    // Blink the LED 5 times
    digitalWrite(ledPin, HIGH);
    delay(500);
    digitalWrite(ledPin, LOW);
    delay(500);
   }
```

```
    Serial.println("LED stopped blinking.");
   }
   else {
    Serial.println("Invalid command. Type 'on', 'off', or 'blink'.");
   }
  }
}
```

Observation:

Observe the LED blinking on and off at the specified intervals.

Result:

The LED successfully blinked at regular intervals as per the uploaded sketch.

Embedded Systems & IoT Experiments Manual

Experiment 10:

Calculate the Moisture Content in the Soil using Arduino UNO

Objective:

To measure the moisture content in soil using a soil moisture sensor connected to Arduino UNO.

Components Required:

1. Arduino UNO
2. Soil Moisture Sensor
3. Jumper Wires
4. Breadboard

Theory:

The Soil Moisture Sensor measures the volumetric water content in the soil. The sensor outputs an analog voltage signal that can be read by the Arduino UNO to determine the moisture level.

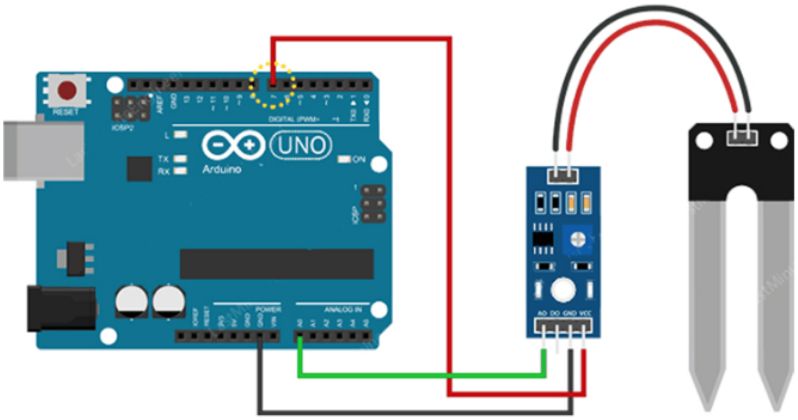
Procedure:

1. Connect the Soil Moisture Sensor to the Arduino UNO as per the PIN diagram.
2. Write and upload the code to read the sensor data.
3. Observe the moisture level readings on the Serial Monitor.

Connections:

Soil Moisture Sensor - Arduino UNO
VCC - 5V
GND - GND
AOUT – 7

Circuit Diagram:



Code:

```
const int soilMoisturePin = 7;  // Pin where the soil moisture sensor is connected
int soilMoistureValue = 0;     // Variable to store the sensor value

void setup() {
  Serial.begin(9600);         // Initialize serial communication at 9600 baud rate
}

void loop() {
  soilMoistureValue = digitalRead(soilMoisturePin);  // Read the value from the sensor
  Serial.print("Soil Moisture Value: ");
  Serial.println(soilMoistureValue);              // Print the sensor value to the Serial Monitor

  delay(4000);                              // Wait for 4 seconds before the next reading
}
```

Observation:

Record the moisture level readings displayed on the Serial Monitor.

Result:

The soil moisture content was successfully measured and displayed on the Serial Monitor.

Embedded Systems & IoT Experiments Manual

Experiment 11:

Calculate the Distance using Ultrasonic Sensor using ESP32

Objective:

To calculate the distance to an object using an Ultrasonic Sensor connected to an ESP32.

Components Required:

1. ESP32
2. Ultrasonic Sensor (HC-SR04)
3. Jumper Wires
4. Breadboard

Theory:

The Ultrasonic Sensor measures the distance to an object by emitting ultrasonic waves and measuring the time taken for the echo to return. The ESP32 processes this data to calculate the distance.

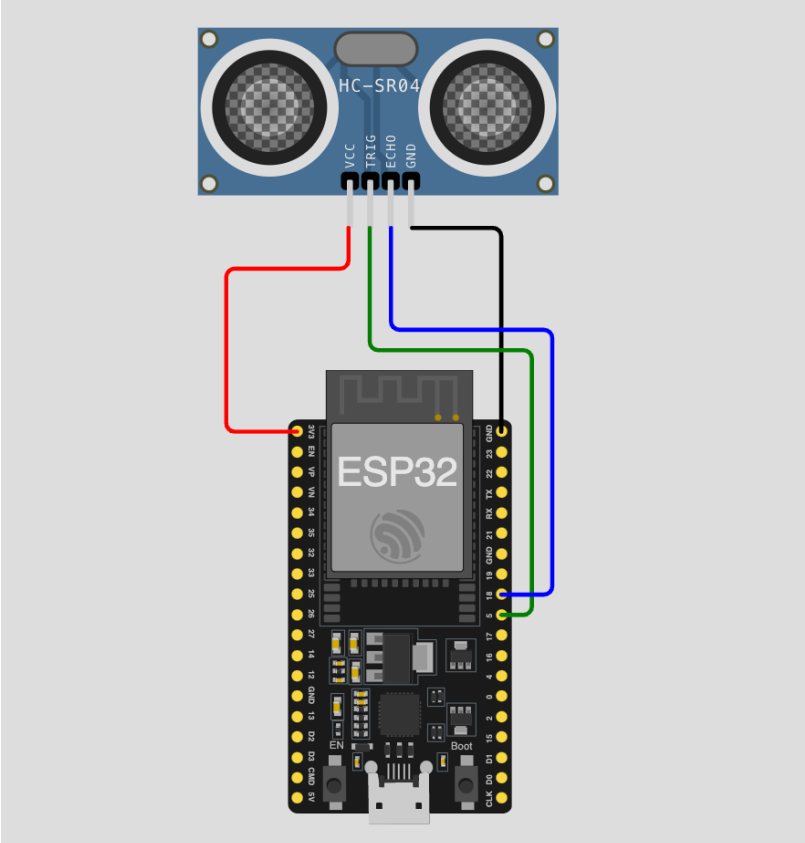
Procedure:

1. Connect the Ultrasonic Sensor to the ESP32 as per the PIN diagram.
2. Write and upload the code to read data from the Ultrasonic Sensor.
3. Observe the distance calculation on the Serial Monitor.

Connections:

HC-SR04 - ESP32
VCC – 3.3V
GND - GND
Trig - GPIO 5
Echo - GPIO 18

Circuit Diagram:



Code:

```
const int trigPin = 5;  // GPIO connected to Trig pin of the ultrasonic sensor
const int echoPin = 18; // GPIO connected to Echo pin of the ultrasonic sensor

void setup() {
  Serial.begin(115200);     // Start serial communication at 115200 baud rate
  pinMode(trigPin, OUTPUT);   // Set the trigPin as an output
  pinMode(echoPin, INPUT);    // Set the echoPin as an input
}

void loop() {
  long duration;
  float distance;

  // Clear the trigPin by setting it LOW
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  // Trigger the sensor by setting the trigPin HIGH for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
```

```
  // Read the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance (in centimeters)
  distance = duration * 0.034 / 2;

  // Print the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(1000); // Wait 1 second before the next measurement
}
```

Observation:

Observe the distance displayed in the Serial Monitor.

Result:

The distance to the object was successfully calculated and displayed on the Serial Monitor.

Embedded Systems & IoT Experiments Manual

Experiment 12:

Basic LED Blinking Functionality using ESP32

Objective:

To control an LED connected to ESP32 and make it blink on and off at regular intervals.

Components Required:

1. ESP32
2. LED
3. Resistor (220Ω)
4. Jumper Wires
5. Breadboard

Theory:

Blinking an LED is a basic experiment to understand GPIO pin control in ESP32. The GPIO pins are programmed to output a HIGH or LOW signal to turn the LED on or off.

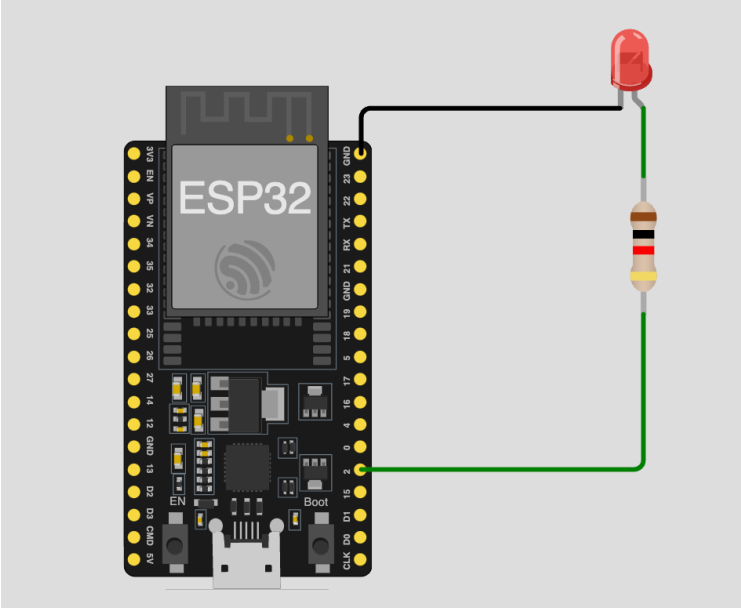
Procedure:

1. Connect the LED and resistor to the ESP32 as per the PIN diagram.
2. Write and upload the code to toggle the GPIO pin connected to the LED.
3. Run the code and observe the LED blinking.

Connections:

LED - ESP32
Anode - GPIO 2 (with 220Ω resistor)
Cathode – GND

Circuit Diagram:



Code:

```
const int ledPin = 2; // Define the pin connected to the LED

void setup() {
  pinMode(ledPin, OUTPUT);   // Set the LED pin as an output
}

void loop() {
  digitalWrite(ledPin, HIGH);  // Turn the LED on
  delay(1000);                 // Wait for 1 second
  digitalWrite(ledPin, LOW);   // Turn the LED off
  delay(1000);                 // Wait for 1 second
}
```


Observation:

Observe the LED blinking on and off at the specified intervals.

Result:

The LED successfully blinked at regular intervals as per the uploaded code.