

Experiment No: 1

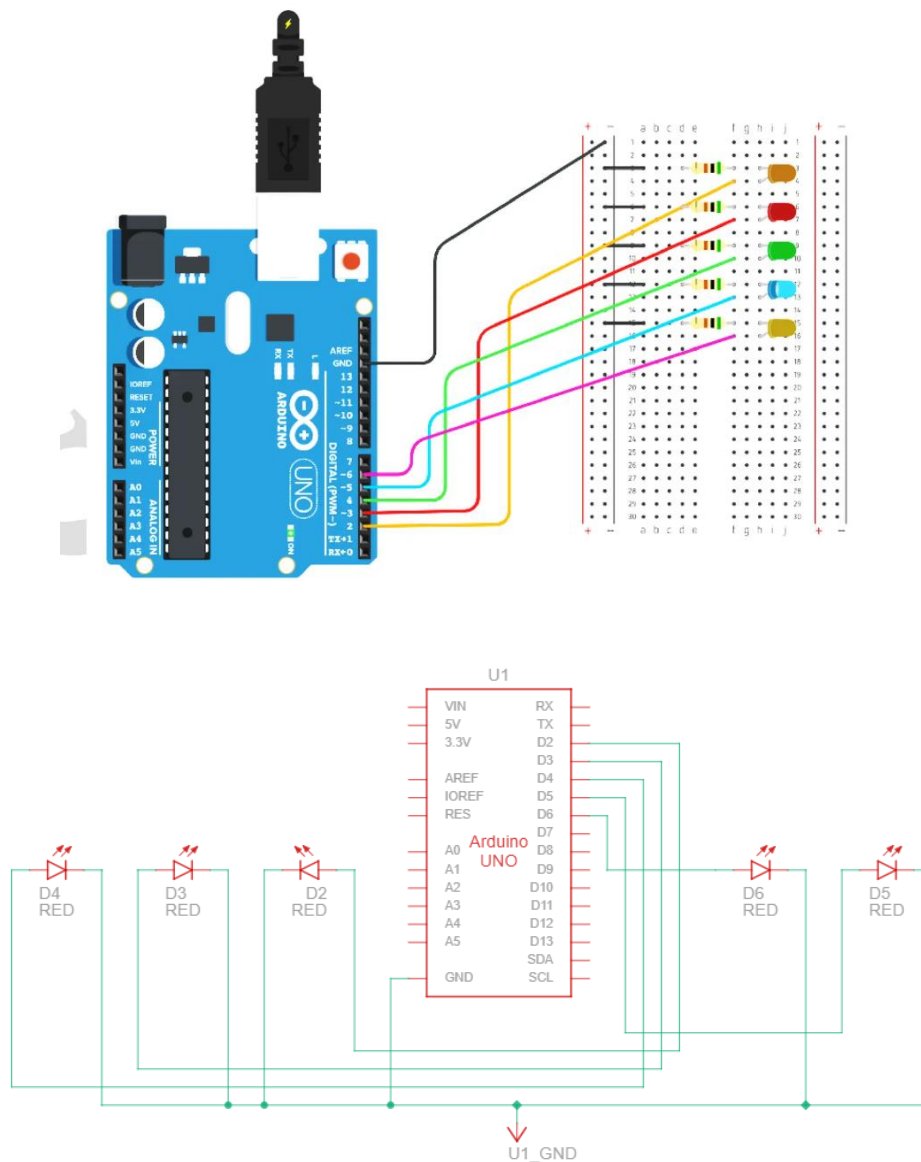
Develop a program to blink 5 LEDs back and forth.

Aim: To develop an Arduino program that blinks **five LEDs back and forth** (left to right and right to left) in a continuous chasing pattern.

Components Required:

Particulars	Quantity
Arduino Uno	1
LED	5
Resistor 330 ohm	5
Jumper Wires	As required

Circuit Diagram:



Theory: LED (Light Emitting Diode) is a diode which when connected in forward bias generates light. The light generated depends on the colour for which the LED is designed.

A LED contains two leads positive terminal (longer one) and negative terminal (shorter one). LED will glow when positive pin of LED is connected to +ve terminal of the battery and negative pin of LED is connected to -ve terminal of the battery. The operating voltage of an LED is between 1.8 and 3.3 volts.

Procedure

1. Connect 5 LEDs to Arduino digital pins 2–6.
2. Connect each LED in series with a 220Ω resistor to ground.
3. Open Arduino IDE and write the program for back-and-forth blinking.
4. Upload the program to Arduino.
5. Observe LEDs blinking sequentially from left to right, then right to left.

//Program

```
int leds[] = {2, 3, 4, 5, 6}; // LED pins
int numLeds = 5;
int delayTime = 200;          // Speed of blinking (ms)

void setup() {
    // Set all LED pins as output
    for (int i = 0; i < numLeds; i++) {
        pinMode(leds[i], OUTPUT);
    }
}

void loop() {
    // Forward direction (left to right)
    for (int i = 0; i < numLeds; i++) {
        digitalWrite(leds[i], HIGH);
        delay(delayTime);
        digitalWrite(leds[i], LOW);
    }

    // Reverse direction (right to left)
    for (int i = numLeds - 2; i > 0; i--) {
        digitalWrite(leds[i], HIGH);
        delay(delayTime);
        digitalWrite(leds[i], LOW);
    }
}
```

Result: A program was successfully developed to blink **five LEDs back and forth** using Arduino

Experiment No: 2

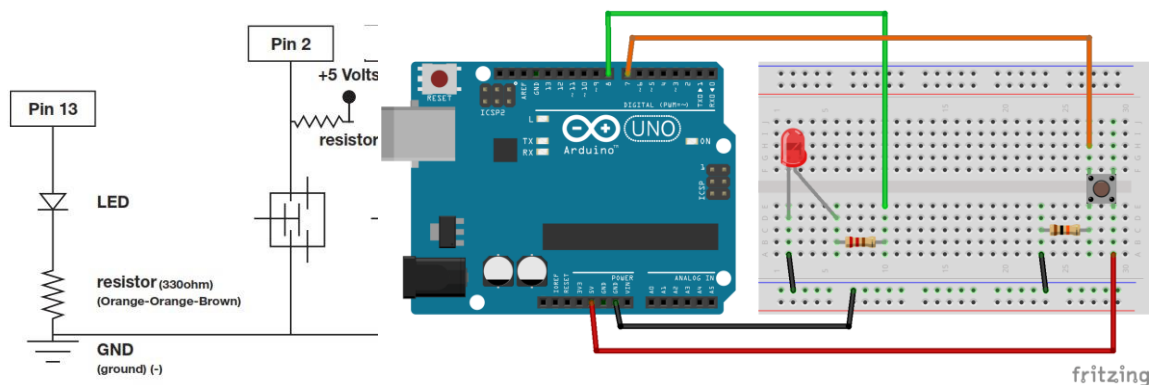
Develop a program to interface a relay with Arduino board.

Aim: To develop an Arduino program to interface a relay with Arduino board.

Components Required:

Particulars	Quantity
Arduino Uno	1
5V Relay Module	1
DC Motor 9/12V	2
Power Supply 9/12V	1
Push Button	1
Jumper Wires	As required

Circuit Diagram:



Theory: A **relay** almost works like a switch, which can be electrically controlled. It can be turned on or off, letting the current go through or not, and can be controlled with low voltages, like the 5V provided by the Arduino pins. This relay module has two channels (those blue cubes). There are other models with one, four and eight channels. This module should be powered with 5V, which is appropriate to use with an Arduino. There are other relay modules that are powered using 3.3V, which is ideal for ESP32, ESP8266, and other microcontrollers.



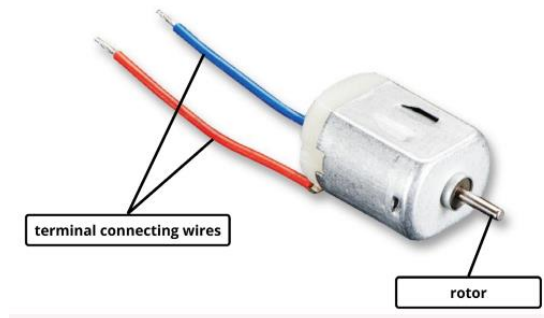
There are two different types of relays that are usually available, based on the number of channels present:-

1. Single channel: This type of relay module can only be connected to one schematic at a time.
2. Double channel: This type of relay is just like a collection of two single channel relays put together, and can be connected to two schematics at a time.

A relay has 3 pins like a lot of other modules, namely:- 1.VCC 2.Ground 3.Signal

On the other side, it has a connector with 3 sockets, common (COM), normally closed (NC), and normally open (NO).

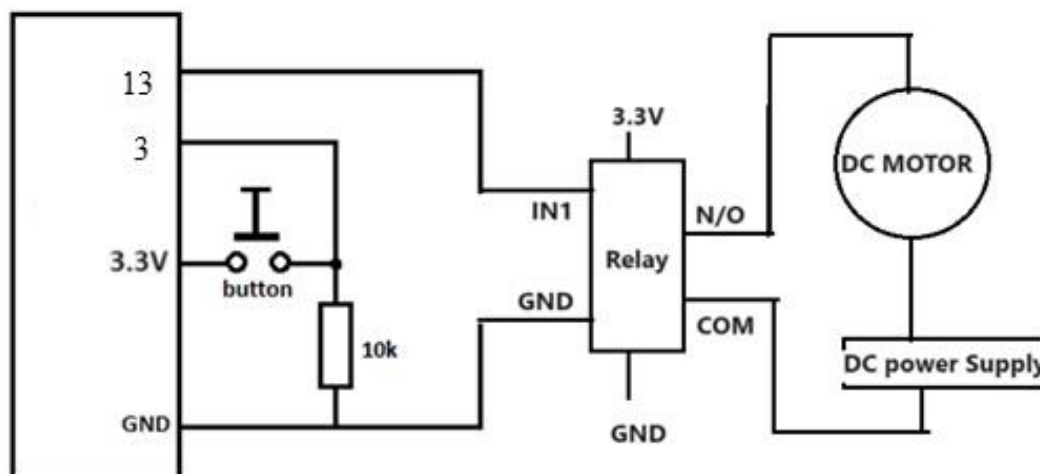
A **DC motor** is a machine that converts chemical or electrical energy into mechanical energy by making the rotors rotate either clockwise or anti-clockwise. The direction of rotation usually depends on the polarity they are connected in.



A DC motor majorly has 3 components on the outside:

1. A rotor which rotates when power is supplied to the motor.
2. Connecting wires.
3. A motor body enclosing the functional parts of the motor.

Circuit Diagram:



Procedure

1. Connect the relay module to Arduino (VCC, GND, and IN pin).
2. Connect the load (motor) through the relay's NO and COM terminals.
3. Write an Arduino program to turn relay ON and OFF with push button.
4. Upload the code to Arduino.
5. Observe the relay clicking and load switching ON/OFF.

//Program

```
const int button1Pin = 2; // pushbutton 1 pin
const int button2Pin = 3; // pushbutton 2 pin
const int RelayPin = 13; // Relay pin

int button1State, button2State; // variables to hold the pushbutton states

void setup()
{
    // Set up the pushbutton pins to be an input:
    pinMode(button1Pin, INPUT);
    pinMode(button2Pin, INPUT);

    // Set up the LED pin to be an output:
    pinMode(RelayPin, OUTPUT);
}

void loop()
{
    button1State = digitalRead(button1Pin);
    button2State = digitalRead(button2Pin);

    // if button1 or button 2 are pressed (but not both)
    if (((button1State == LOW) && (button2State == HIGH)) || ((button1State == HIGH) && (button2State == LOW)))
    {
        digitalWrite(RelayPin, HIGH); // turn the Relay pin on
    }
    else
    {
        digitalWrite(RelayPin, LOW); // turn the Relay pin off
    }
}
```

Result: A relay was successfully interfaced with Arduino. The relay switched ON/OFF based on the push button connected to arduino, enabling control of external loads.

Experiment No: 3

Develop a program to deploy an intrusion detection system using Ultrasonic and sound sensors.

Aim: To develop an Arduino-based program to detect intrusion using an ultrasonic sensor (distance measurement) and a sound sensor (noise detection), and trigger an alert using buzzer/LED.

Components Required:

Particulars	Quantity
Arduino Uno	1
Ultrasonic Sensor	1
LED	1
Buzzer	1
Jumper Wires	As required

Theory: Ultrasonic Sensor: An ultrasonic Sensor is a device used to measure the distance between the sensor and an object without physical contact. This device works based on time-to-distance conversion.

Working Principle of Ultrasonic Sensor:

Ultrasonic sensors measure distance by sending and receiving the ultrasonic wave. The ultrasonic sensor has a sender to emit the ultrasonic waves and a receiver to receive the ultrasonic waves. The transmitted ultrasonic wave travels through the air and is reflected by hitting the Object. Arduino calculates the time taken by the ultrasonic pulse wave to reach the receiver from the sender.

We know that the speed of sound in air is nearly 344 m/s,

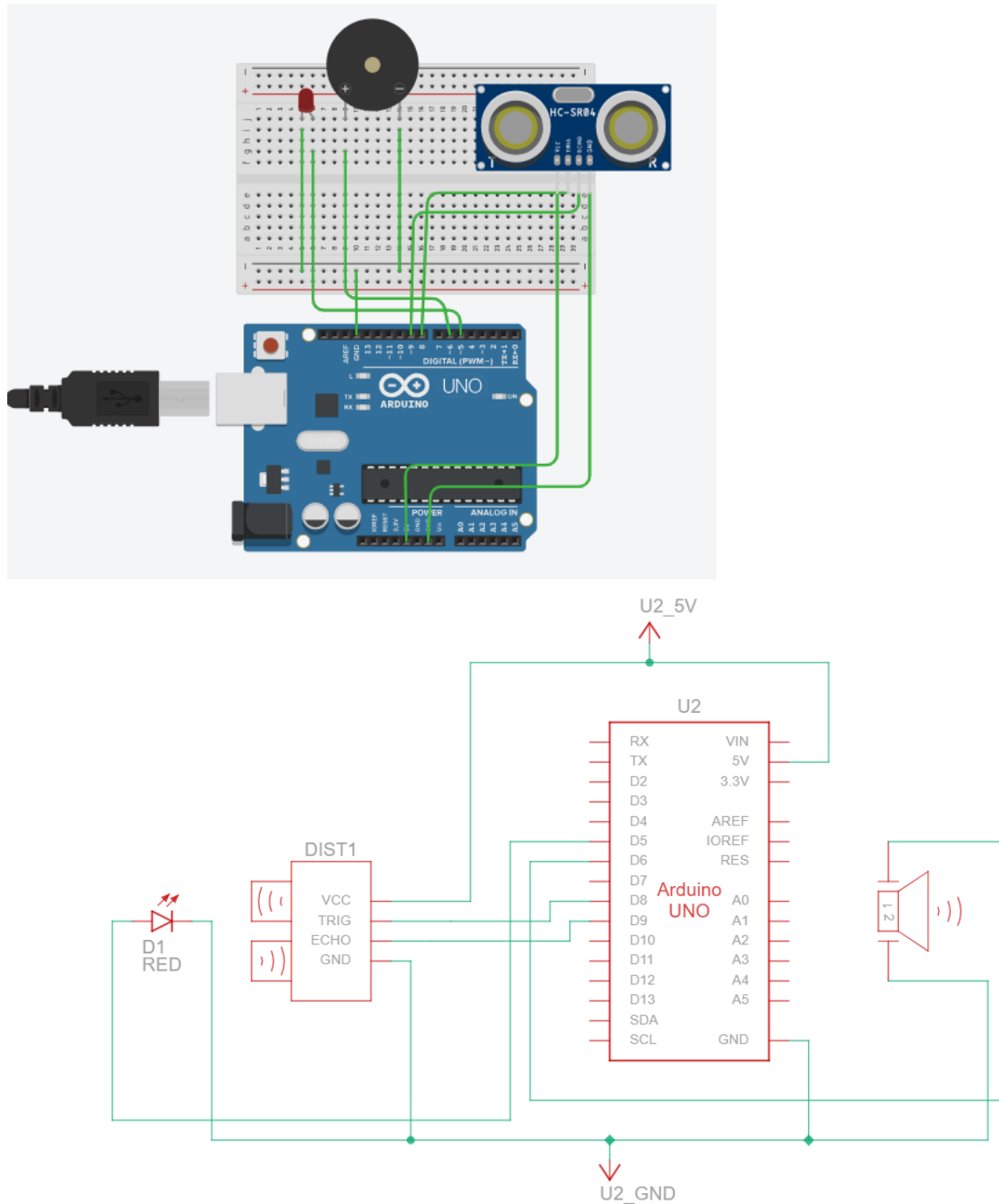
So, the known parameters are time and speed (constant). Using these parameters, we can calculate the distance traveled by the sound wave.

Formula: Distance = Speed * Time

Distance is calculated using the formula:

$$\text{Distance (cm)} = \frac{\text{Time} \times 0.034}{2}$$

Circuit Diagram:



Procedure

1. Connect ultrasonic, sound sensor, and buzzer to Arduino.
2. Write a program to measure distance and read sound sensor output.
3. Set intrusion threshold (distance < 50 cm or sound detected).
4. Upload code to Arduino.
5. Observe buzzer activating when intruder is present or a loud sound occurs.

```
//Program

/*
  Intrusion Detection System
  Using Ultrasonic Sensor (HC-SR04)
  Alarm stays ON until intruder leaves
*/

#define TRIG 9
#define ECHO 10
#define BUZZER 5
#define LED 6

long duration;
int distance;
const int distanceThreshold = 50; // cm

void setup() {
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);
  pinMode(BUZZER, OUTPUT);
  pinMode(LED, OUTPUT);

  Serial.begin(9600);
}

void loop() {
  // --- Measure Distance ---
  digitalWrite(TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  duration = pulseIn(ECHO, HIGH);
  distance = duration * 0.034 / 2; // cm

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // --- Intrusion Logic ---
  if (distance > 0 && distance <= distanceThreshold) {
    Serial.println("? ALERT! Intruder Detected!");
    digitalWrite(BUZZER, HIGH);
    digitalWrite(LED, HIGH);
  } else {
    digitalWrite(BUZZER, LOW);
    digitalWrite(LED, LOW);
  }

  delay(200);
}
```

Result: An intrusion detection system was successfully designed using an ultrasonic sensor and sound sensor.

Experiment No: 4

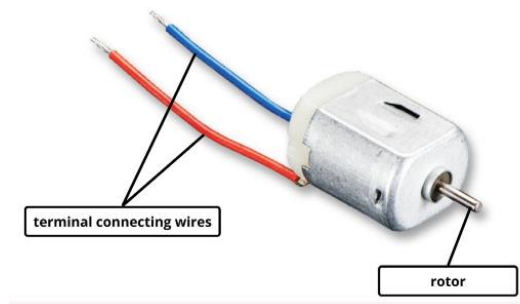
Develop a program to control a DC motor with Arduino board.

Aim: To develop an Arduino program to control the direction of a DC motor using an Arduino board and an L293D motor driver.

Components Required:

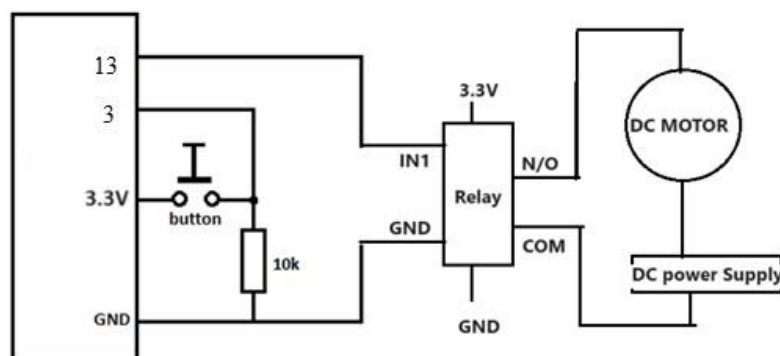
Particulars	Quantity
Arduino Uno	1
DC Motor	1
Motor Driver Module	1
Jumper Wires	As needed

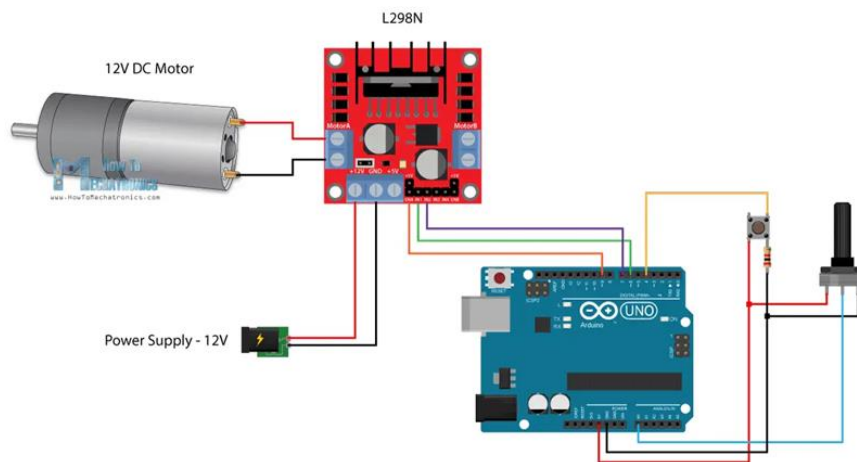
Theory: A **DC motor** is a machine that converts chemical or electrical energy into mechanical energy by making the rotors rotate either clockwise or anti-clockwise. The direction of rotation usually depends on the polarity they are connected in.



A DC motor majorly has 3 components on the outside:

1. A rotor which rotates when power is supplied to the motor.
2. Connecting wires.
3. A motor body enclosing the functional parts of the motor.

Circuit Diagram:

Circuit Diagram:**. Procedure**

1. Connect L293D to Arduino on a breadboard.
2. Connect motor terminals to OUT1 and OUT2 of L293D.
3. Connect IN1 and IN2 pins of L293D to Arduino pins 6 and 7.
4. Connect external motor power to Vcc2.
5. Make all grounds common.
6. Upload the program to Arduino and observe motor rotation.

//Program

```
/* Arduino DC Motor Control - PWM | H-Bridge | L298N - */

#define enA 9
#define in1 6
#define in2 7
#define button 4

int rotDirection = 0;
int pressed = false;

void setup() {
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(button, INPUT);
  // Set initial rotation direction
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
}

void loop() {
  int potValue = analogRead(A0); // Read potentiometer value
  int pwmOutput = map(potValue, 0, 1023, 0, 255); // Map the potentiometer value from 0 to 255
  analogWrite(enA, pwmOutput); // Send PWM signal to L298N Enable pin
```

```
// Read button - Debounce
if (digitalRead(button) == true) {
    pressed = !pressed;
}
while (digitalRead(button) == true);
delay(20);

// If button is pressed - change rotation direction
if (pressed == true & rotDirection == 0) {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    rotDirection = 1;
    delay(20);
}
// If button is pressed - change rotation direction
if (pressed == false & rotDirection == 1) {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    rotDirection = 0;
    delay(20);
}
}
```

Result: A program was successfully developed to control a **DC motor's direction** using Arduino and L293D driver.

Experiment No: 5

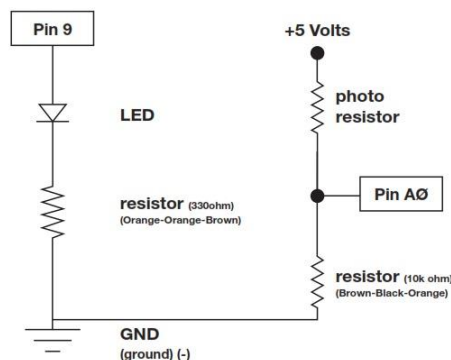
Develop a program to deploy smart street light system using LDR sensor.

Aim: To develop an Arduino program that automatically turns street lights ON at night and **OFF during daylight** using an LDR (light-dependent resistor) as the light sensor.

Components Required:

Particulars	Quantity
Arduino Uno	1
LDR/Photo Sensor.	1
LED	1
10K-ohm resistor	1
Jumper Wires	7

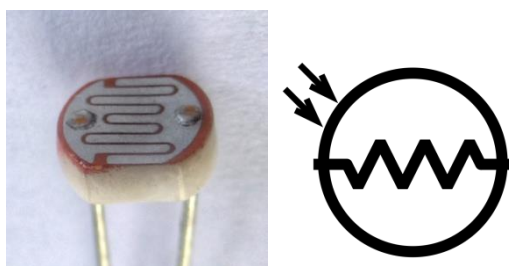
Circuit Diagram:



Theory: An LDR changes resistance with light intensity: resistance decreases in bright light and increases in darkness. By forming a voltage divider (LDR + fixed resistor) and reading the divider voltage on an analog input, the Arduino can determine ambient light level and switch a lamp using a relay or transistor when the light level falls below a threshold.

Typical steps:

- Read analog value from voltage divider (0–1023).
- Compare reading with a threshold (calibrated for your location).
- Drive relay/MOSFET to turn lamp ON/OFF accordingly.



//Program

```
// Define the pin where the LDR is connected
const int ldrPin = A0;
// Define the pin where the LED is connected
const int ledPin = 9;

void setup() {
  // Initialize serial communication
  Serial.begin(9600);
  // Set the LED pin as an output
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // Read the analog value from the LDR
  int ldrValue = analogRead(ldrPin);

  // Print the analog value to the Serial Monitor
  Serial.print("LDR Value: ");
  Serial.println(ldrValue);

  // Check if it's dark (adjust threshold as needed)
  if (ldrValue > 200) {
    // If it's dark, turn on the LED
    digitalWrite(ledPin, HIGH);
  } else {
    // If it's not dark, turn off the LED
    digitalWrite(ledPin, LOW);
  }

  // Wait for a short delay before taking the next reading
  delay(1000);
}
```

Result: Successfully Interfaced Arduino with LDR/Photo resistor sensor and displayed sensor value on serial monitor.

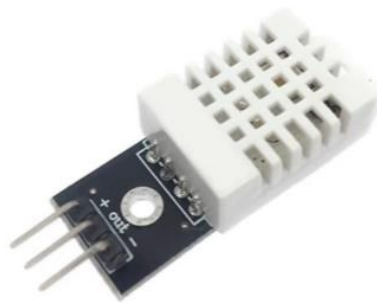
Experiment No: 6

Develop a program to classify dry and wet waste with the Moisture sensor (DHT22).

Aim: To develop an Arduino program that classifies dry waste and wet waste using the DHT22 humidity sensor, based on the moisture level detected.

Components Required:

Particulars	Quantity
Arduino Uno	1
DHT22 Sensor Module	2
Jumper Wires	As required

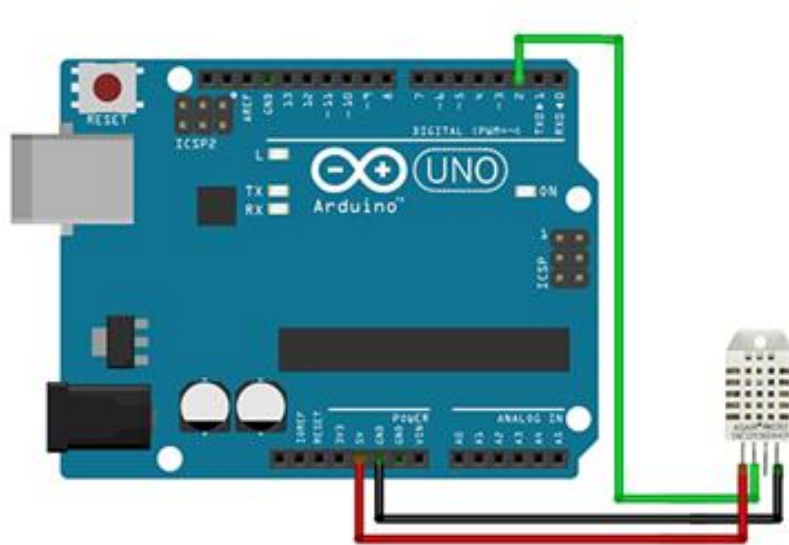


Theory: The **DHT22** is a widely used, low-cost, and highly accurate **temperature and humidity sensor** available in the market. It uses a **capacitive humidity sensor** and a **thermistor** to measure moisture and temperature levels and provides the readings as a **digital output**, making it very easy to interface with microcontrollers like Arduino, ESP8266, ESP32, and Raspberry Pi. Its **high accuracy, wider measurement range, and stability** make the DHT22 a preferred choice for professional and hobby IoT projects.

The DHT22 measures **relative humidity**, which represents the amount of water vapor present in the air compared to the maximum amount of vapor the air can hold at the same temperature. When this limit is reached, the air becomes saturated, causing water vapor to condense and form dew. Because of its precision in measuring relative humidity and temperature, the DHT22 is suitable for applications like weather monitoring, greenhouses, and industrial control.

Some popular alternative sensors to the DHT22 include **DHT11**, **AM2302** (which is actually a DHT22 in a different casing), and more advanced sensors such as the **SHT15** or **SHT31**, which offer even better accuracy and faster response times.

Circuit Diagram:



Procedure

1. Connect the DHT22 sensor to Arduino (with pull-up on DATA pin).
2. Connect LEDs to show classification status.
3. Upload the program given below.
4. Place different types of waste near the DHT22 and observe humidity readings.
5. System classifies waste as **Dry** or **Wet** based on humidity threshold.

```
//Program

#include <Adafruit_Sensor.h>
#include <DHT.h>

#define DHTPIN 2          // DHT22 data pin connected to digital pin 2
#define DHTTYPE DHT22     // Sensor type

DHT dht(DHTPIN, DHTTYPE); // Create DHT object

void setup() {
  Serial.begin(9600);
  Serial.println("Dry/Wet Waste Classification System (Humidity Only)");
  dht.begin();
  delay(2000);
}

void loop() {
  // Read humidity
  float humidity = dht.readHumidity();

  // Check if the reading is valid
  if (isnan(humidity)) {
    Serial.println("? Failed to read from DHT22 sensor!");
    delay(2000);
    return;
  }

  // Display humidity
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");

  // Classification logic
  if (humidity > 65.0) {
    Serial.println("?? Classification: WET WASTE");
  } else {
    Serial.println("?? Classification: DRY WASTE");
  }

  Serial.println("-----");
  delay(3000); // Wait 3 seconds before next reading
}
```

Result: A program was successfully developed to classify **dry and wet waste** using the **DHT22 humidity sensor**.

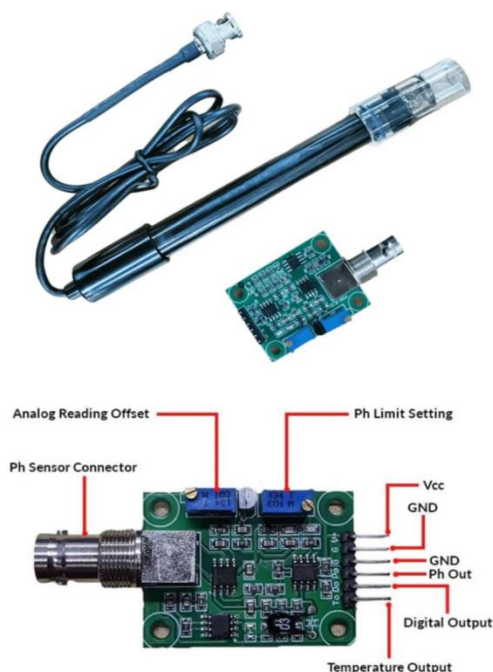
Experiment No: 7

Develop a program to read the pH value of a various substances like milk, lime and water.

Aim: To develop an Arduino program to measure the pH level of different substances such as milk, lime juice, and water using an analog pH sensor.

Components Required:

Particulars	Quantity
Arduino Uno	1
Bluetooth Module	1
LED	1
Jumper Wires	--



Theory: A **pH sensor** is an economical and commonly used device for measuring the acidity or alkalinity of different liquids. It works using a **glass electrode** that generates a small voltage depending on the hydrogen ion concentration present in the solution. This voltage is then converted into a **pH value**, which ranges from **0 to 14**, where 7 is neutral, values below 7 indicate acidity, and values above 7 indicate alkalinity.

Because of its simplicity, wide availability, and ability to interface easily with microcontrollers through an **analog output**, the pH sensor is popular in laboratories, agriculture, food processing, and IoT-based monitoring systems.

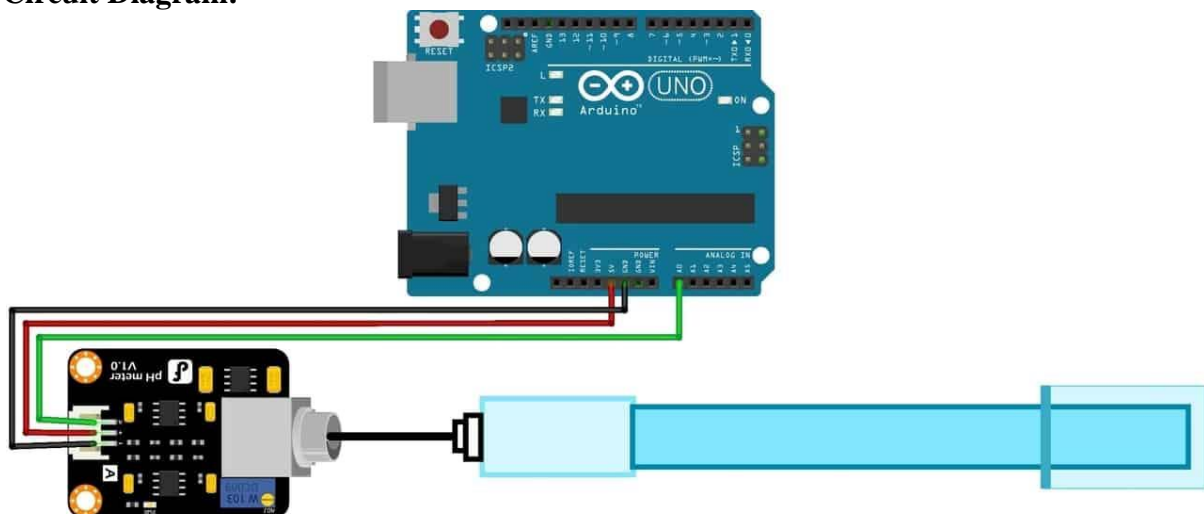
This sensor can be used to detect the **pH of lime juice, water, and milk**—each of which has its own characteristic pH value.

- **Lime juice** is highly acidic and typically shows a **pH of around 2–3**.
- **Pure water** is neutral and shows a **pH close to 7**, although impurities can shift it slightly.
- **Milk** is mildly acidic, usually in the **pH range of 6.4–6.8**, depending on freshness and storage conditions.

Measuring pH is important because the pH level affects taste, safety, chemical reactions, and biological activity. In food quality control, for example, a change in the pH of milk can indicate spoilage. Similarly, monitoring pH in water is essential for drinking safety and agricultural applications.

Some alternatives to the commonly used analog pH sensor kit include more advanced probes such as the **Atlas Scientific pH sensor**, **Gravity Analog pH Sensor V2**, and **SEN0169 pH probe**, which offer higher accuracy, better stability, and longer lifespan.

Circuit Diagram:



Procedure

1. Power the pH sensor board and connect its analog output to A0.
2. Rinse pH probe with distilled water and dry gently.
3. Calibrate:
 - Place probe in pH 7 solution → note voltage
 - Place in pH 4 solution → note voltage
4. Update the formula constants (if needed).
5. Dip probe in **water, milk, and lime juice** one by one.
6. Observe readings on the Serial Monitor.

//Program

```
// pH Measurement Program - Milk, Lime, Water
const int pHSensorPin = A0;
float voltage, pHValue;

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

void loop() {
  int sensorValue = analogRead(pHSensorPin);

  // Convert ADC value (0-1023) to voltage (0-5V)
  voltage = sensorValue * (5.0 / 1023.0);

  // pH conversion formula (adjust after calibration)
  pHValue = 7 + ((2.5 - voltage) / 0.18);

  Serial.print("Voltage: ");
  Serial.print(voltage);
  Serial.print(" V | pH Value: ");
  Serial.println(pHValue);

  delay(1000);
}
```

Result: The Arduino successfully measured the **pH values** of water, milk, and lime.

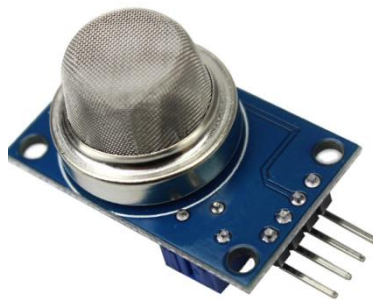
Experiment No: 8

Develop a program to detect the gas leakage in the surrounding environment.

Aim: To develop an Arduino-based system to detect gas leakage in the surrounding environment using an MQ-series gas sensor and trigger an alarm using a buzzer and LED.

Components Required:

Particulars	Quantity
Arduino Uno	1
MQ2 Gas Sensor	2
LED	1
Buzzer	1
Jumper Wires	As required



Theory: The **MQ-series gas sensors** are inexpensive, easy-to-use gas detection modules commonly used for sensing various gases in the environment. Each sensor in the MQ family is designed to detect a specific type of gas or a group of gases, making them suitable for safety monitoring, industrial automation, and home-based IoT projects. These sensors operate using a **tin-dioxide (SnO_2) sensitive layer**, whose resistance changes when it comes in contact with certain gases. This change in resistance is converted into an **analog output**, which can be read by microcontrollers like Arduino or ESP boards.

MQ sensors are popular among students and hobbyists because of their **low cost, simple interface, wide detection range, and fast response time**. They are commonly used in gas leakage detection systems, air quality monitors, and smart home safety devices.

Different MQ sensors detect different types of gases. For example:

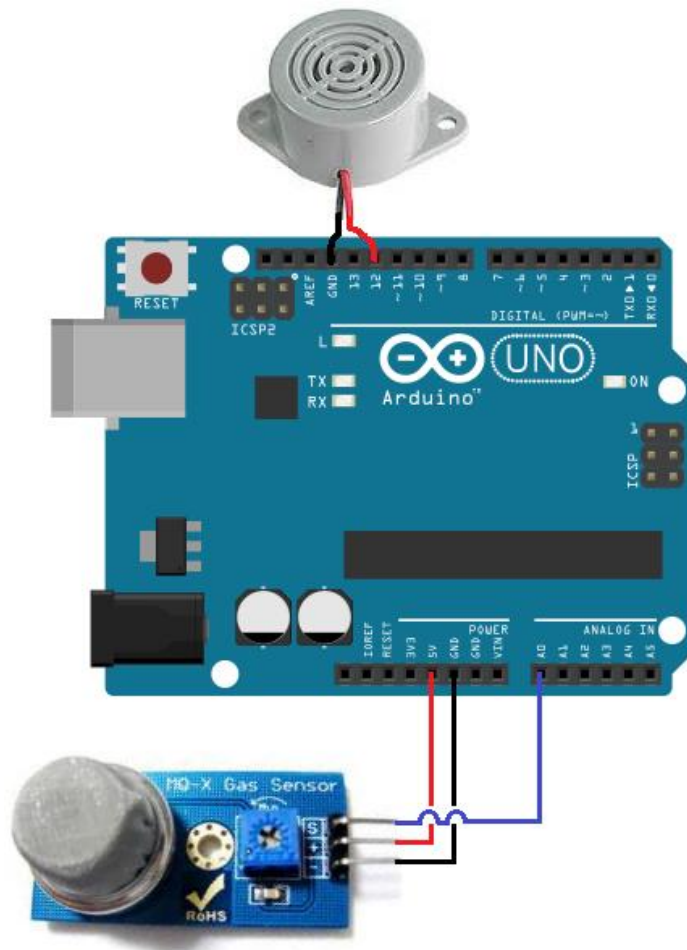
- **MQ-2** detects LPG, propane, hydrogen, and smoke
- **MQ-3** detects alcohol vapor
- **MQ-4** detects methane (CNG gas)

- **MQ-5** detects LPG and natural gas
- **MQ-7** detects carbon monoxide (CO)
- **MQ-135** detects ammonia, benzene, and general air pollutants

These sensors help determine whether the concentration of gases in the air is within safe limits. For instance, the **MQ-2** can be used in kitchens or industries to detect LPG leakage, while the **MQ-135** is useful for monitoring indoor air quality.

Some alternatives to MQ sensors include more advanced modules like **NDIR CO₂ sensors**, **electrochemical gas sensors**, and **semiconductor-based air quality sensors**, which offer higher accuracy and better long-term stability.

Circuit Diagram:



Procedure

1. Connect the MQ gas sensor to the Arduino (VCC, GND, A0).
2. Connect the buzzer to pin D8 and the LED to pin D9 with a resistor.
3. Upload the program to Arduino using Arduino IDE.
4. Open Serial Monitor to observe gas values.
5. Bring a small amount of gas/smoke near the sensor to test detection.
6. Observe LED glow and buzzer sound when leakage is detected.

//Program

```
int gas = A0;          // MQ Sensor Analog pin
int buzzer = 8;        // Buzzer pin
int led = 9;           // LED pin
int threshold = 300;   // Gas level threshold (adjust based on testing)

void setup() {
  Serial.begin(9600);
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  pinMode(gas, INPUT);
}

void loop() {
  int gasValue = analogRead(gas);
  Serial.print("Gas Level: ");
  Serial.println(gasValue);

  if (gasValue > threshold) {
    digitalWrite(buzzer, HIGH);
    digitalWrite(led, HIGH);
    Serial.println("? Gas Leakage Detected!");
  }
  else {
    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
  }

  delay(500);
}
```

Result: A gas leakage detection system was successfully developed using Arduino and an MQ gas sensor.

Experiment No: 9

Develop a program to demonstrate weather station readings using Arduino.

Aim: To develop a program to demonstrate weather station readings using Arduino.

Components Required:

Particulars	Quantity
NodeMCU	1
DHT11 Sensor Module	2
Jumper Wires	As required

Theory: The DHT11 sensor is a very economical, simple temperature and humidity sensor available in the market, It uses a capacitive humidity sensor and a thermistor to detect the values and show them in digital output. Its compact size and sampling rate made this sensor popular among hobbyists. Some of the sensors which can be used as an alternative to DHT11 sensor are DHT22, AM2302, SHT71.

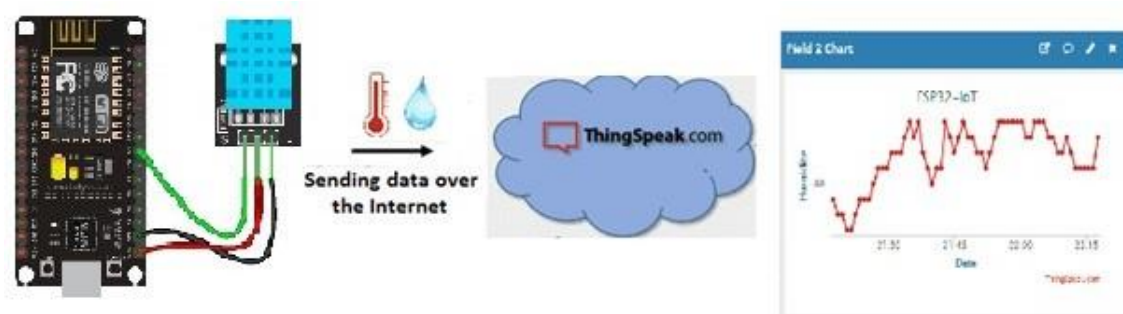
The DHT11 also measures relative humidity. The relative humidity is the amount of water vapour in air vs. the saturation point of water vapour in the air. At the saturation point, water vapour starts to condense and accumulate on surfaces forming dew.

Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. For example, the LiquidCrystal library makes it easy to talk to character LCD displays. There are thousands of libraries available for download directly through the Arduino IDE, and you can find all of them listed at the Arduino Library Reference.

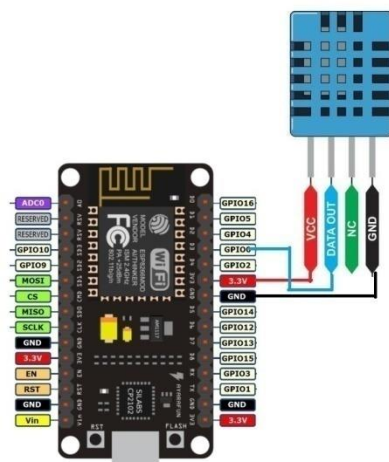
Steps to Add DHT11 Sensor Library

- 1) Open your Arduino IDE and go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open.
- 2) Search DHT then find the DHT Sensor library by Adafruit
- 3) Click install button to install library
- 4) If ask click on install all button to install library dependencies

ThingSpeak is an open-source Internet of Things (IoT) application and API that allows users to collect and store sensor data in the cloud and perform analytics on that data. It allows users to create “channels” to collect data from multiple sensors, and also has built-in support for visualizing and analyzing the data. **ThingSpeak** can be used for a variety of applications, such as monitoring environmental conditions, tracking the location of assets, and controlling devices remotely. It is available for free and also has paid subscription plans for additional features and support. The device that sends the data must be configured with the correct channel information, such as the channel ID and write API key.



Circuit Diagram:



Steps to Connect

Step 1: ThingSpeak Setup for Temperature and Humidity Monitoring

- For creating your channel on Thingspeak, you first need to Sign up on Thingspeak.
- In case if you already have an account on Thingspeak, just sign in using your id and password.
- For creating your account go to www.thingspeak.com.
- After this, verify your E-mail id and click on continue.

Step 2: Create a Channel for Your Data

- Once you Sign in after your account verification, create a new channel by clicking “New Channel” button.
- After clicking on “New Channel”, enter the Name and Description of the data you want to upload on this channel.
- Enter the name of your data ‘Temperature’ in Field1 and ‘Humidity’ in Field2.
- Click on the save channel button to save your details.

Step 3: API Key

- To send data to Thingspeak, we need a unique API key, which we will use later in our code to upload our sensor data to Thingspeak Website.
- Click on “API Keys” button to get your unique API key for uploading your sensor data.
- Now copy your “Write API Key”. We will use this API key in our code.

//Program

```
#include <DHT.h>    // Including library for dht

#include <ESP8266WiFi.h>

String apiKey = "ZEMV49YW59MIN8K9";    // Enter your Write API key from ThingSpeak

const char *ssid = "Redmi";    // replace with your wifi ssid and wpa2 key
const char *pass = "123456789";
const char* server = "api.thingspeak.com";

#define DHTPIN 0    //pin where the dht11 is connected

DHT dht(DHTPIN, DHT11);

WiFiClient client;

void setup()
{
    Serial.begin(115200);
    delay(10);
    dht.begin();

    Serial.println("Connecting to ");
    Serial.println(ssid);

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

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

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
}

void loop()
{
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (isnan(h) || isnan(t))
    {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    if (client.connect(server,80))    // "184.106.153.149" or
api.thingspeak.com
    {
        String postStr = apiKey;
        postStr += "&field1=";
        postStr += String(t);
        postStr += "&field2=";
        postStr += String(h);
        postStr += "\r\n\r\n";

        client.print("POST /update HTTP/1.1\n");
        client.print("Host: api.thingspeak.com\n");
        client.print("Connection: close\n");

        client.print("X-THINGSPEAKAPIKEY: "+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);

        Serial.print("Temperature: ");
        Serial.print(t);
        Serial.print(" degrees Celcius, Humidity: ");
        Serial.print(h);
        Serial.println("%. Send to Thingspeak.");
    }
    client.stop();

    Serial.println("Waiting...");

    // thingspeak needs minimum 15 sec delay between updates
    delay(1000);
}
```

Result : Successfully uploaded temperature and humidity data to thingspeakcloud.

Experiment No: 10

Develop a program to setup a UART protocol and pass a string through the protocol.

Aim: To develop a program that sets up UART (Universal Asynchronous Receiver/Transmitter) communication on Arduino and transmits a string through the serial protocol.

Component Required:

Sl No	Components	Quantity
1	Arduino Uno	2
2	USB cable	1
3	Jumper wires	--
4	Breadboard	1

Theory

What is UART?

UART (Universal Asynchronous Receiver/Transmitter) is a **serial communication protocol** that sends data **bit-by-bit** over a single wire without a clock signal.

Key Features

- Asynchronous (no separate clock)
- Full-duplex (simultaneous TX & RX)
- Uses:
 - **TX** (Transmit pin)
 - **RX** (Receive pin)
 - **GND**

Procedure

1. Upload the **transmitter code** to Arduino-1.
2. Upload the **receiver code** to Arduino-2.
3. After uploading, **disconnect USB from the transmitter** (to avoid serial conflict).
4. Connect Arduino 1 and 2 using the UART wiring.
5. Open Serial Monitor for **Arduino Receiver only**.
6. Observe the received string being printed.

```
//Program

/*
  UART String Transmission Example
  Board: Arduino Uno
  TX -> Pin 1
  RX -> Pin 0
*/

void setup() {
  // Initialize UART at 9600 baud
  Serial.begin(9600);

  // Wait for Serial Monitor to open (optional, for debugging)
  while (!Serial) {
    ; // Wait
  }

  // Send an initial string
  Serial.println("UART Communication Started!");
}

void loop() {
  // Transmit a string every 2 seconds
  Serial.println("Hello, UART! This is Arduino Uno transmitting.");
  delay(2000);
}

/*
  UART Receiver
  Arduino Uno #2
  TX -> Pin 1
  RX -> Pin 0
*/

void setup() {
  Serial.begin(9600); // UART at 9600 baud
}

void loop() {
  if (Serial.available() > 0) { // Check if data is available
    String received = Serial.readStringUntil('\n'); // Read until newline
    Serial.print("Received: ");
    Serial.println(received); // Display on Serial Monitor
  }
}
```

Result: UART protocol communication was successfully established between two Arduino boards.

Experiment No: 11

Develop a water level depth detection system using Ultrasonic sensor..

Aim: To develop an Arduino-based system that measures the water depth (and level) inside a tank using an ultrasonic sensor and takes actions (display, buzzer/relay) based on measured depth.

Component Required:

Sl No	Components	Quantity
1	Arduino Uno	1
2	Ultrasonic sensor	1
3	LED	2
4	Jumper wires	--
6	Breadboard	1

Theory: **Ultrasonic Sensor:** An ultrasonic Sensor is a device used to measure the distance between the sensor and an object without physical contact. This device works based on time-to-distance conversion.

Working Principle of Ultrasonic Sensor:

Ultrasonic sensors measure distance by sending and receiving the ultrasonic wave. The ultrasonic sensor has a sender to emit the ultrasonic waves and a receiver to receive the ultrasonic waves. The transmitted ultrasonic wave travels through the air and is reflected by hitting the Object. Arduino calculates the time taken by the ultrasonic pulse wave to reach the receiver from the sender.

We know that the speed of sound in air is nearly 344 m/s,

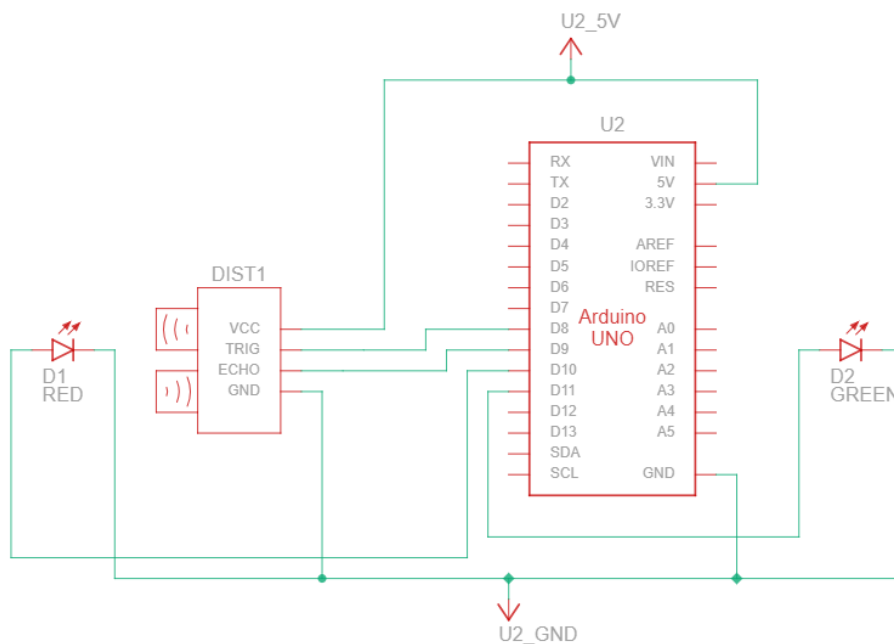
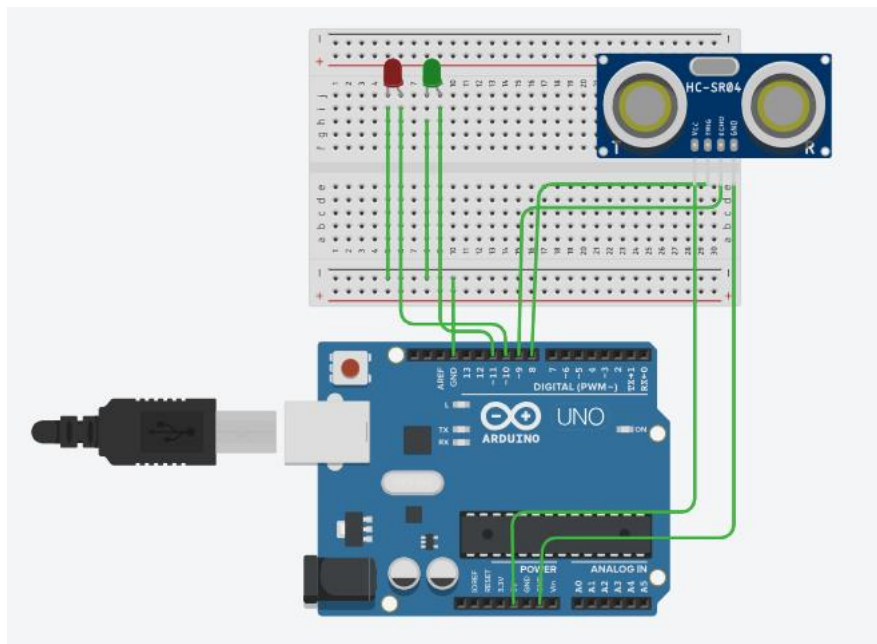
So, the known parameters are time and speed (constant). Using these parameters, we can calculate the distance traveled by the sound wave.

Formula: Distance = Speed * Time

Distance is calculated using the formula:

$$\text{Distance (cm)} = \frac{\text{Time} \times 0.034}{2}$$

Circuit Diagram:



Procedure

1. Mount the HC-SR04 on tank lid facing down, centered and stable. Measure tankHeight (sensor to bottom).
2. Wire components per above.
3. Upload the sketch below to Arduino.
4. Open Serial Monitor (9600 baud) to see distance and depth.
5. Test by changing water level and observe readings; calibrate thresholds

//Program

```
// Define pins for ultrasonic sensor
#define TRIG_PIN 8
#define ECHO_PIN 9

// Define pin for relay module to control motor
#define RELAY_PIN 7

// Define pins for LEDs
#define RED_LED_PIN 10
#define GREEN_LED_PIN 11

// Declare global variables for distance measurement
long duration; // To store the duration of the ultrasonic pulse
int distance; // To store the calculated distance

void setup() {
    // Set up pins
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
    pinMode(RELAY_PIN, OUTPUT);

    pinMode(RED_LED_PIN, OUTPUT);
    pinMode(GREEN_LED_PIN, OUTPUT);

    // Initially, turn off motor and LEDs
    digitalWrite(RELAY_PIN, LOW);
    digitalWrite(RED_LED_PIN, LOW);
    digitalWrite(GREEN_LED_PIN, LOW);

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

void loop() {
    // Clear the TRIG_PIN
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);

    // Trigger the ultrasonic sensor to send a pulse
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    // Measure the duration of the pulse
    duration = pulseIn(ECHO_PIN, HIGH);

    // Calculate distance in cm
    distance = duration * 0.034 / 2;

    // Print distance for debugging purposes
    Serial.print("Water Level: ");
    Serial.print(distance);
    Serial.println(" cm");
}
```

```
// Control motor & LEDs with only LOW and HIGH thresholds
if (distance < 10) { // High water level (full tank)
  digitalWrite(RELAY_PIN, LOW); // Turn on the motor
  digitalWrite(RED_LED_PIN, LOW); // Red LED OFF
  digitalWrite(GREEN_LED_PIN, HIGH);
  Serial.println("Motor: OFF (Water Full)");
}
else if (distance > 30) { // Low water level (empty tank)
  digitalWrite(RELAY_PIN, HIGH); // Turn on the motor
  digitalWrite(RED_LED_PIN, HIGH);
  digitalWrite(GREEN_LED_PIN, LOW); // Green LED OFF
  Serial.println("Motor: ON (Water Low)");
}

// Small delay before next measurement
delay(1000);
}
```

Result: The water level depth detection system using an ultrasonic sensor was successfully developed and tested.

Experiment No: 12

Develop a program to simulate interfacing with the keypad module to record the keystrokes.

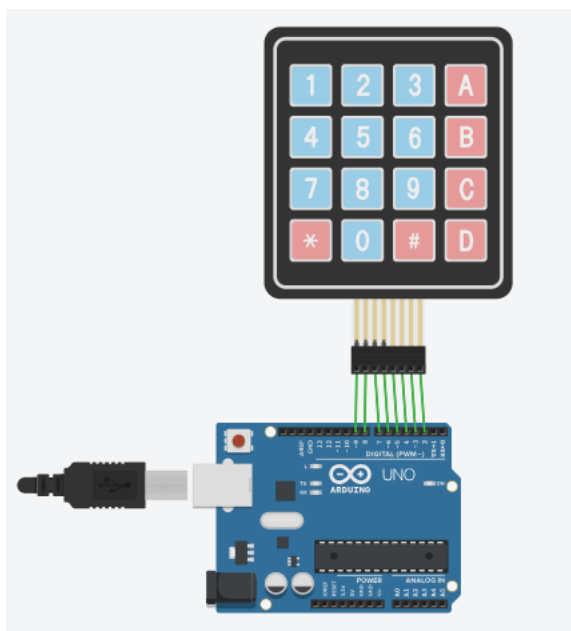
Aim: To develop and execute an Arduino program to interface a 4×4 keypad module and record the keystrokes entered by the user.

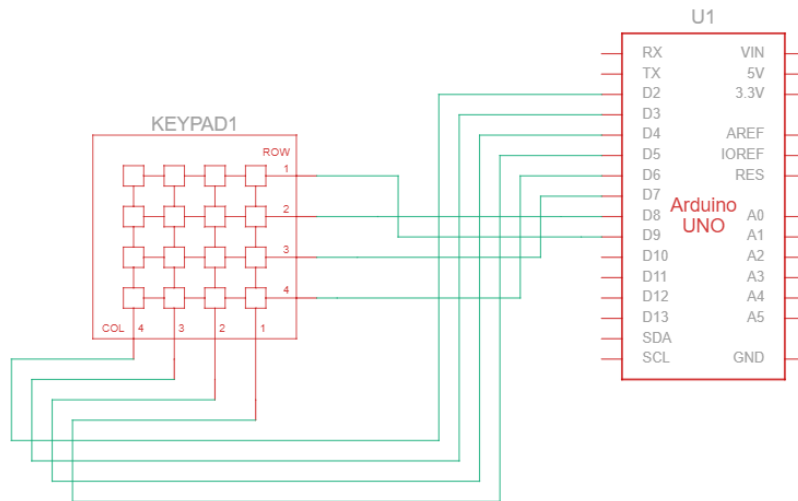
Component Required:

Sl No	Components	Quantity
1	Arduino Uno	1
2	Ultrasonic sensor	1
3	LED	2
4	Jumper wires	--
6	Breadboard	1

Theory: A **matrix keypad** is arranged in rows and columns. When a key is pressed, it connects a specific row and column. Arduino scans these rows and columns to detect the exact key pressed.

We use the **Keypad.h** library to simplify keypad scanning.

Circuit Diagram:



PROCEDURE

1. Connect the keypad module to Arduino as per pin mapping.
2. Open Arduino IDE → Create a new sketch.
3. Include the Keypad library.
4. Define rows, columns, and keymap.
5. Initialize the keypad object.
6. Write code to detect which key is pressed.
7. Display the key on the Serial Monitor.
8. Upload the program into Arduino.
9. Observe keystrokes displayed in the Serial Monitor

//Program

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;

char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

String buffer = "";
const unsigned int MAX_BUFFER = 200;

void setup() {
  Serial.begin(9600);
  Serial.println(F("=== Keypad Keystroke Recorder ==="));
  Serial.println(F("Press keys on keypad..."));
  Serial.println(F("Special keys:"));
  Serial.println(F("  # ? Show recorded keys"));
  Serial.println(F("  * ? Clear recorded keys"));
}

void loop() {
  char key = keypad.getKey();
  if (key) {
    unsigned long ts = millis();

    Serial.print("Key pressed: ");
    Serial.print(key);
    Serial.print(" | Time (ms): ");
    Serial.println(ts);

    if (buffer.length() < MAX_BUFFER) {
      buffer += key;
    }

    if (key == '#') {
      Serial.print("Recorded sequence: ");
      Serial.println(buffer);
    }
    else if (key == '*') {
      buffer = "";
      Serial.println("Buffer cleared!");
    }
  }
}
```

Result: The keypad interfacing program was successfully developed and executed.