ERODE SENGUNTHAR

ENGINEERING COLLEGE

(APPROVED BY AICTE, NEW DELHI & PERMANENTLY AFFILIATED TO ANNA UNIVERSITY, CHENNAI.
ACCREDITED BY NBA, NEW DELHI, NAAC WITH GRADE “A” & IE(I),KOLKATA)

PERUNDURAI, ERODE – 638 057

An Autonomous Institution

BONAFIDE CERTIFICATE

Register No: ....

Certified that this is the Bonafide Record of Work Done By

Name of the Student:..

Branch :..

Lab Code/Name :..

Year/Semester :..

Faculty Incharge

Head of the Department

Submitted for the End Semester Practical Examination held on ....

Internal Examiner

External Examiner

Ex.No :1 **STUDY OF RASPBERRY PI**
Date:

Aim:

To understand the basic functionalities and setup process of Raspberry Pi, focusing on the Raspberry Pi 4 model.

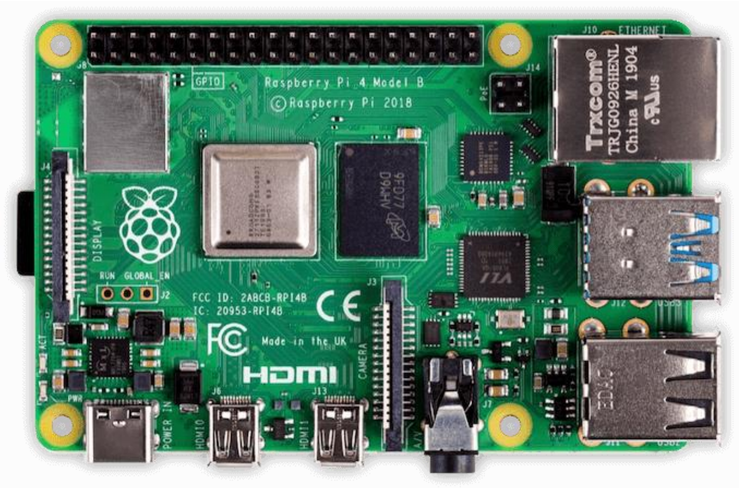
Components Needed:

Raspberry Pi, power supply, microSD card, monitor, keyboard, mouse, breadboard, LEDs, resistors, jumper wires, sensors (optional), and networking tools

Introduction

Raspberry Pi is a small, affordable, and powerful single-board computer designed to promote computer science education and innovation. Originally developed by the Raspberry Pi Foundation, it has evolved into a popular platform for projects ranging from DIY electronics to advanced IoT applications. Raspberry Pi supports various operating systems, making it versatile for educational and professional use.

The Raspberry Pi 4, the latest in the series, offers significant improvements in performance, making it suitable for tasks such as programming, media streaming, and AI/ML applications.



Flavors

Raspberry Pi supports multiple operating systems tailored for different purposes. Some popular options include:

Raspberry Pi OS (formerly Raspbian): A Debian-based Linux OS optimized for the Raspberry Pi hardware.

Ubuntu: A full-featured Linux distribution widely used in development and server applications.

Kali Linux: A security-focused OS designed for penetration testing and ethical hacking.

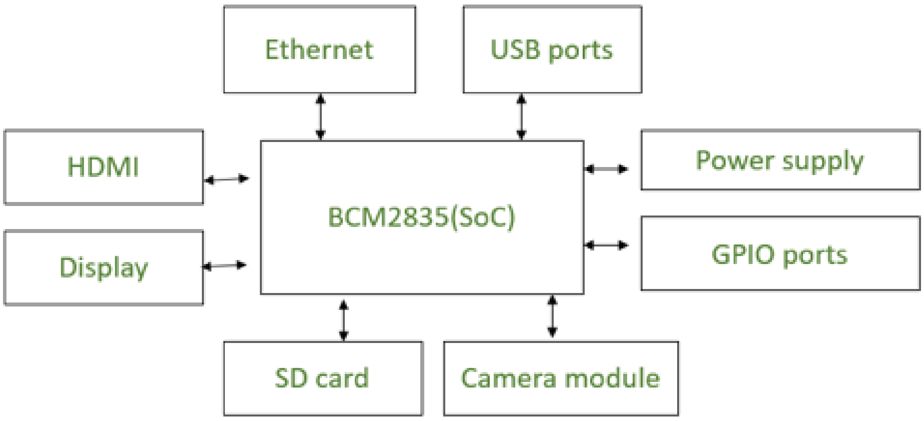
Retropie: A platform for retro gaming, allowing users to emulate classic game consoles.

Windows IoT Core: A simplified version of Windows for IoT applications.

Each flavor is customizable and suited to specific applications, making Raspberry Pi a flexible tool for learners and professionals.

Architecture

Raspberry Pi's architecture is based on the ARM (Advanced RISC Machine) processor family, known for energy efficiency and compact design. Key features of the Raspberry Pi 4 include:



- Processor: Quad-core Cortex-A72 64-bit ARMv8 processor, running at 1.5 GHz.
- RAM: Available in 2GB, 4GB, or 8GB LPDDR4 options.
- Storage: MicroSD card slot for OS and data storage.
- Ports:
  2 × USB 3.0 and 2 × USB 2.0 ports.
  Dual micro-HDMI ports supporting 4K resolution.
  Ethernet port for high-speed network connectivity.
- Wireless Connectivity: Built-in Wi-Fi (802.11ac) and Bluetooth 5.0.
- GPIO Pins: 40 General-Purpose Input/Output pins for hardware interfacing.

This architecture provides a balance between affordability and performance, suitable for educational and real-world applications.

Experiment: Using Raspberry Pi 4

The experiment involves setting up the Raspberry Pi 4, installing Raspberry Pi OS, and running basic Python scripts.

Materials Required:

- Raspberry Pi 4 (any RAM variant
- Power supply (USB-C adapter)
- MicroSD card (minimum 16GB)
- HDMI monitor or TV
- Keyboard and mouse
- Internet connection

Procedure:

1. Download and flash Raspberry Pi OS onto the microSD card using tools like Balena Etcher.
2. Insert the microSD card into the Raspberry Pi 4 and connect peripherals (monitor, keyboard, mouse).
3. Power on the Raspberry Pi and complete the initial OS setup.

4. Open the terminal and update the system:
   sudo apt update &&sudo apt upgrade -y
5. Run a basic Python script to test functionality:
   print("Hello, Raspberry Pi!")
6. Explore GPIO pins by connecting an LED and writing a Python program to control it.

Result:

Ex.No:2 **BLINKING LED USING RASPBERRY PI**

Date:

Aim:

To create a smooth fading effect for an LED by continuously adjusting its brightness. The brightness is increased and decreased in a loop to create the fading effect.

Apparatus Required:

S.No	Apparatus	Quantity
1	Raspberry Pi	1
2	Power Supply	1
3	microSD Card	1
4	LED	1
5	330 Ω Resistor	1
6	Breadboard	1
7	Jumper Wires	10
8	HDMI Monitor	1
9	Keyboard	1
10	Mouse	1
11	HDMI Cable	1

Procedure:

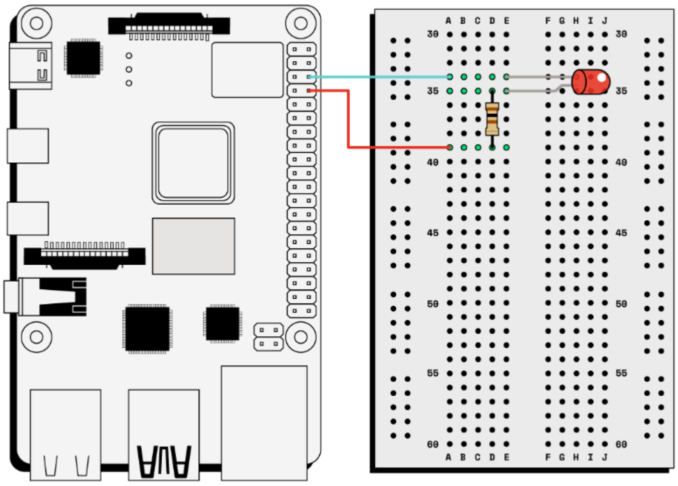
Step 1: Initialize: Set initial brightness to 0 and set fade amount to 5.
Step 2: Set Up LED Pin: Configure the LED pin as an output.
Step 3: Main Loop: Write current brightness to the LED, adjust brightness by adding fade amount, reverse fade direction if brightness reaches 0 or 255, and delay briefly to show the fade effect.

Program:

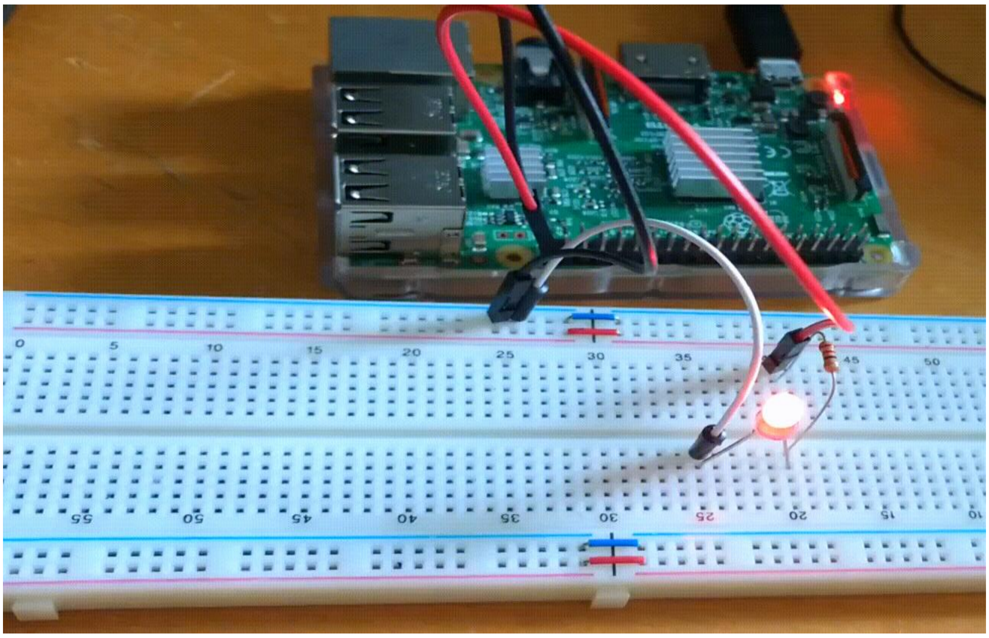
```
importRPi.GPIO as GPIO
import time

LED_PIN = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN,GPIO.OUT)
try:
    while True:
GPIO.output(LED_PIN,GPIO.HIGH)
Time.sleep(1)
GPIO.output(LED_PIN,GPIO.LOW)
Time.sleep(1)
```

Circuit Diagram:



Output:



Result:

Ex.No:3 **REAL TIME MOTION DETECTION USING SENSOR**
Date:

Aim:

To develop a real-time motion detection system using a motion sensor (PIR sensor) and Raspberry Pi, demonstrating basic hardware interfacing and sensor-based event detection.

Apparatus Required:

S.No	Apparatus	Quantity
1	Arduino Uno	1
2	Arduino IDE Tool (Software)	1
3	Ultrasonic Sensor (HC-SR04)	1
4	Breadboard	1
5	Jumper Wires	10

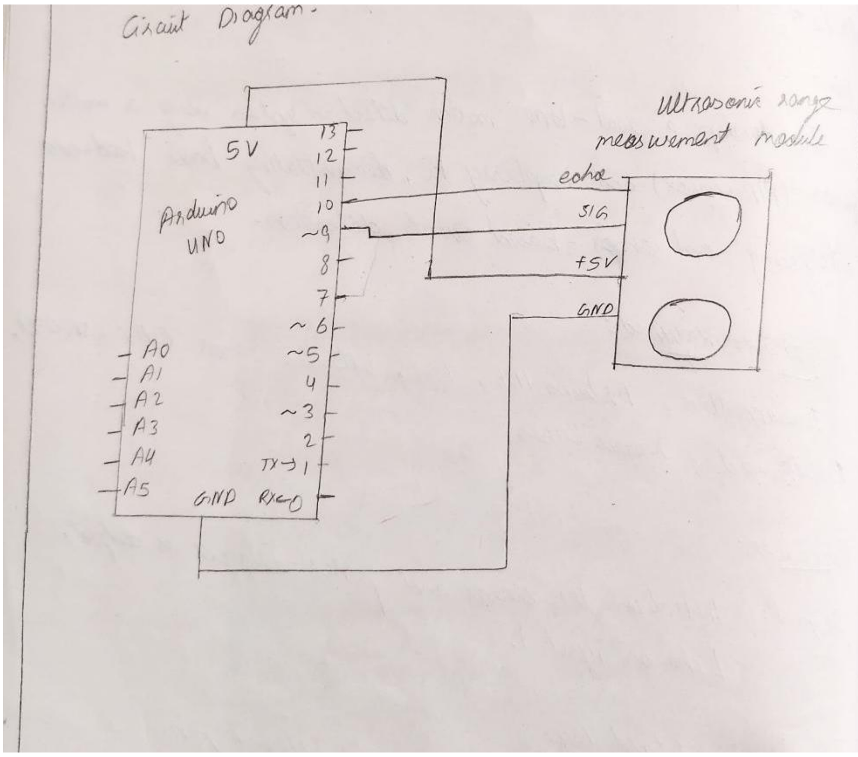
Procedure:

Step 1: Initialize the sensor and pins: Set Trigger as output and Echo as input.
Step 2: Start loop: Send a short pulse on the Trigger pin, measure the time for the Echo pin to receive the reflected signal, calculate distance using the formula Distance(cm) = Time(μs) × 0.0343 / 2, and display the distance on the Serial Monitor or LCD.
Step 3: Repeat the loop continuously.

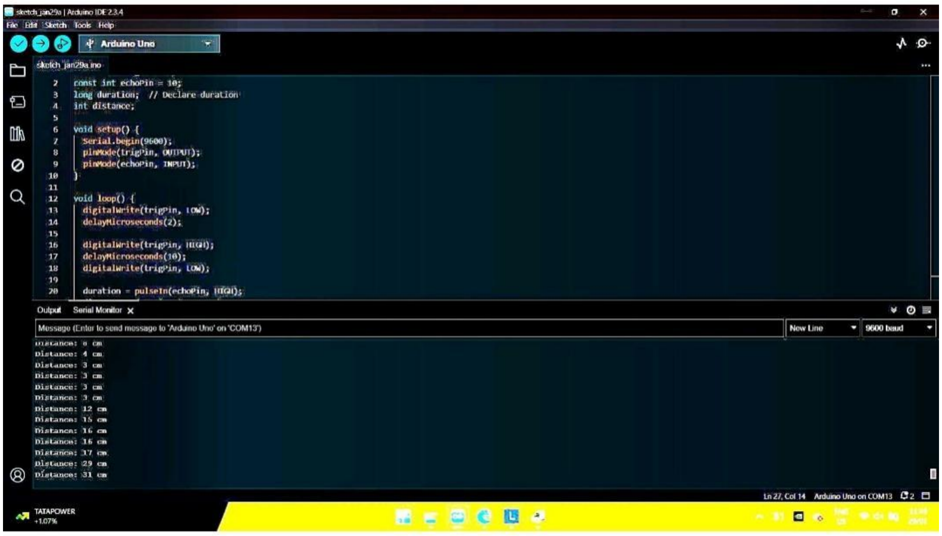
Program:

```
constinttrigPin = 9;
constintechoPin=10;
int distance;
void setup() {
Serial.begin(9600);
pinMode(trigPin,OUTPUT);
pinMode(echoPin,
INPUT);}
void loop() {
digitalWrite(trigPin,
LOW);
delayMicroseconds(2);
digitalWrite(trigPin,HIGH
);delayMicroseconds(10);
digitalWrite(trigPin,
LOW);
duration=pulseIn(echoPin,HIGH)
; distance = duration * 0.034 / 2;
Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");
```

Circuit Diagram:



Output:



Result:

Ex.No:4
Date:

MEASURING TEMPERATURE AND HUMIDITY USING SENSOR

Aim:

To design and implement a temperature monitoring system using an Arduino board and a temperature sensor to continuously measure and display temperature in real-time.

Apparatus Required:

S.No	Apparatus	Quantity
1	Arduino UNO Board	1
2	Temperature Sensor (e.g., LM35/DHT11)	1
3	Jumper Wires	10
4	Breadboard	1
5	Resistor	1
6	USB Cable	1
7	Arduino IDE Tool (Software)	1

Procedure:

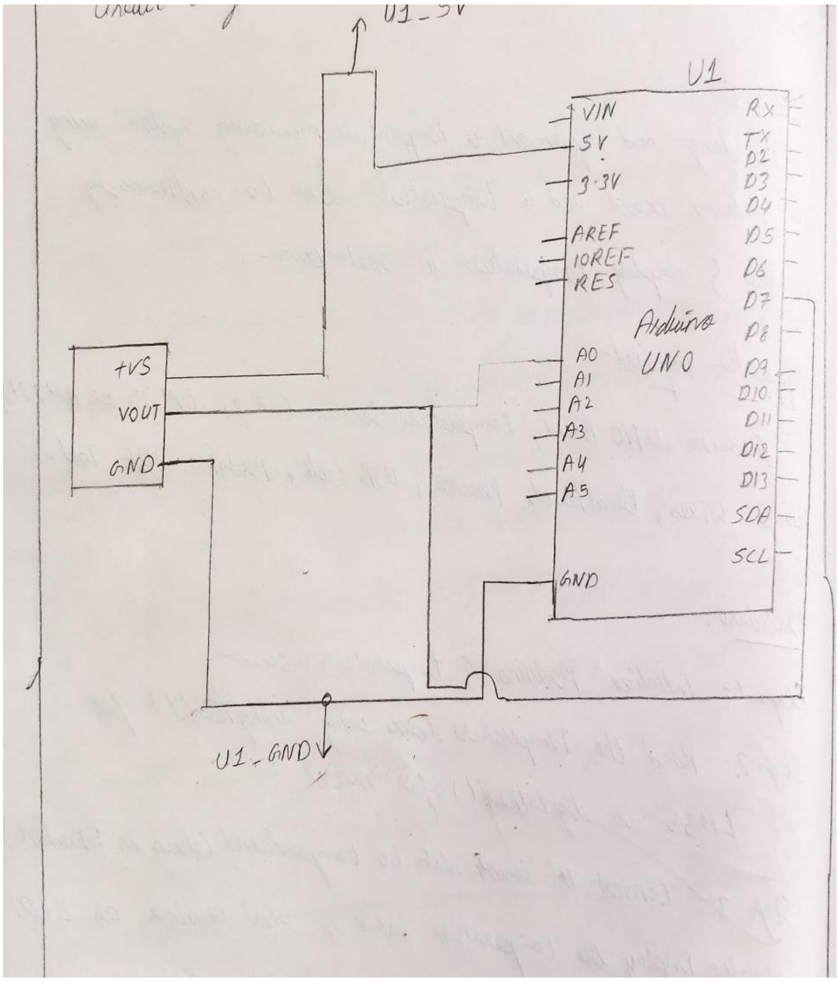
Step1:Initialize Arduino and temperature sensor.
Step2:Read the temperature data using analog Read() for LM35ordigital Read() for DH11
Step 3: Convert the sensor data to temperature (Celsius or Fahrenheit).
Step4:Display the temperature value on Serial Monitoror LCD.
Step 5: Repeat the process with a delay.

Program:

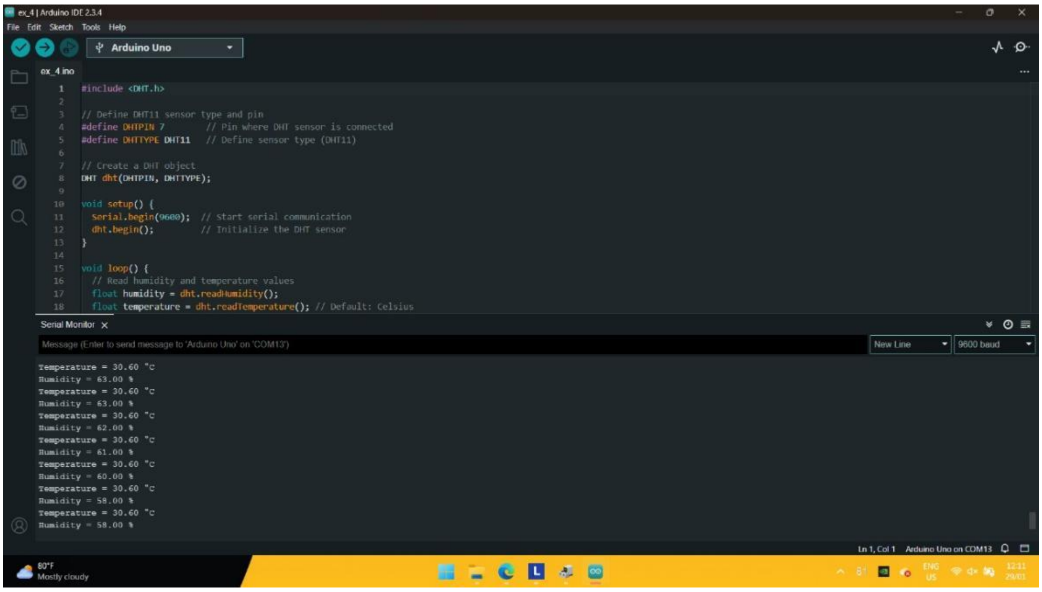
```
#include <DHT.h>  // Include the DHT sensor library from Adafruit
#define DHTPIN 7        // Define the pin where the DHT11 sensor is connected
#define DHTTYPE DHT11    // Define the sensor type as DHT11
DHT dht(DHTPIN, DHTTYPE);  // Create an instance of the DHT class
void setup() {
  Serial.begin(9600);  // Start serial communication at 9600 baud rate
  dht.begin();        // Initialize the DHT sensor
}
void loop() {
```

```
float h = dht.readHumidity();    // Read humidity
float t = dht.readTemperature(); // Read temperature in Celsius
// Check if any reading failed and exit early
if (isnan(h) || isnan(t)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
}
// Print temperature and humidity to the Serial Monitor
Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" °C");
```

Circuit diagram:



Output:



Result:

Ex.No:5
Date:

REAL TIME SMOKE DETECTION WITH ALERT SYSTEM

Aim:

To design and implement a system using an MQ sensor and Arduino to detect the presence and concentration of gases in the environment and display the sensor data in real-time.

Apparatus Required:

S.No	Apparatus	Quantity
1	Arduino UNO Board	1
2	MQ Sensor (e.g., MQ-2/MQ-135)	1
3	Jumper Wires	10
4	Breadboard	1
5	Resistor	1
6	USB Cable	1
7	Arduino IDE Tool (Software)	1
8	LCD Display (e.g., 16x2)	1
9	Power Supply	1
10	LED	1
11	Buzzer	1

Procedure:

Step 1: Initialize Arduino and MQ Sensor: Connect the MQ sensor to the Arduino and set up the Arduino with the required power supply.
Step 2: Sensor Calibration: Initialize sensor parameters and run the MQ sensor in clean air for a period to establish baseline readings.
Step 3: Read Analog Data from the MQ Sensor: Use the analogRead() function to read the gas concentration from the MQ sensor and store the sensor reading in a variable.
Step 4: Process the Data: Convert the raw analog data to a human-readable form (e.g., gas concentration in ppm) and optionally compare sensor values against predefined threshold levels to detect dangerous gas concentrations.
Step 5: Output the Sensor Data: Print the sensor readings to the Serial Monitor of the Arduino IDE and optionally display the gas concentration on an LCD or trigger an alert via LED or buzzer if the concentration exceeds a threshold.
Step 6: Repeat the Reading Process: Continuously monitor gas concentration by running the data reading and processing in an infinite loop, and add a small delay between readings to avoid flooding the data.

Program:

```
#define sensorPin A0  // Define the analog pin where MQ135 is connected
#define buzzerPin 8   // Define the digital pin where the buzzer is connected

int sensorValue = 0;  // Variable to store sensor reading

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

void loop() {
  sensorValue = analogRead(sensorPin);  // Read the analog value from the sensor
  Serial.print("Gas Sensor Value: ");   // Print the label
  Serial.println(sensorValue);         // Print the sensor value

  // Alert when smoke/gas is detected
  if (sensorValue > 150) {  // Example threshold, adjust as necessary
    Serial.println("Alert: Smoke/Gas detected!");

    // Activate buzzer with 1-second interval
    digitalWrite(buzzerPin, HIGH);  // Turn on the buzzer
    delay(500);                     // Buzzer on for 0.5 seconds
    digitalWrite(buzzerPin, LOW);   // Turn off the buzzer
    delay(500);                     // Buzzer off for 0.5 seconds
  } else {
    digitalWrite(buzzerPin, LOW);   // Ensure buzzer is off when no gas is detected
  }

  delay(1000);  // Delay 1 second before taking the next reading
}
```

Circuit Diagram:

Output:

Result:

Ex.No:6
Date:

IOT BASED INTELLIGENCT TRAFFIC MANAGEMENT SYSTEM

Aim:

To design and implement an IoT-based intelligent traffic management system using Arduino, enabling efficient traffic flow by monitoring and controlling traffic signals based on real-time data.

Apparatus Required:

S.No	Apparatus	Quantity
1	Arduino Uno or Mega	1
2	IR Sensors or Ultrasonic Sensors	2 (1 of each or 2 of same)
3	LEDs (Red, Yellow, Green)	3 (1 of each color)
4	Breadboard	1
5	Jumper Wires	10
6	Resistors (330 Ω)	3
7	Wi-Fi Module (ESP8266 or ESP32)	1
8	Power Supply or USB Cable	1

Procedure:

Step1:Connect the IR/ultrasonic sensors to Arduino for detecting vehicle density.
Step2:InConnect LEDs to Arduino as traffic signals using resistors.
Step3:Attach the Wi-Fi module to enable IoT connectivity.
Step4:Write and upload code using Arduino IDE for traffic control based on sensor data.
Step5:Link the setup to an IoT platform for real-time data monitoring.
Step6:Test and verify lane prioritization and data transmission.

Program:

```
const int red = 7, yellow = 6, green = 5;
const int trig = 9, echo = 8;

void setup() {
  for (int pin : {red, yellow, green}) pinMode(pin, OUTPUT);
  pinMode(trig, OUTPUT); pinMode(echo, INPUT);
  Serial.begin(9600);
}

void loop() {
  long dist = getDistance();
```

```
  if (dist < 20) {
    Serial.println("Vehicle detected! Green ON");
    showLights(HIGH, LOW, LOW);
    delay(5000);
    showLights(LOW, HIGH, LOW);
    delay(2000);
  }
  showLights(LOW, LOW, HIGH);
  Serial.println("No vehicle or waiting. Red ON");
  delay(1000);
}

long getDistance() {
  digitalWrite(trig, LOW); delayMicroseconds(2);
  digitalWrite(trig, HIGH); delayMicroseconds(10);
  digitalWrite(trig, LOW);
  return pulseIn(echo, HIGH) * 0.034 / 2;
}

void showLights(bool g, bool y, bool r) {
  digitalWrite(green, g);
  digitalWrite(yellow, y);
  digitalWrite(red, r);
}
```

Circuit Diagram:

Output:

Result:

Ex.No:7 **IOT BASED SMART IRRIGATION SYSTEM**
Date:

Aim:

To design and implement an IoT-based smart irrigation system using Arduino to monitor soil moisture and automate water supply for efficient water management.

Apparatus Required:

S.No	Apparatus	Quantity
1	Arduino Uno or Mega	1
2	Soil Moisture Sensor	1
3	Water Pump with Relay Module	1
4	Water Pipe	1
5	Jumper Wires	10
6	Breadboard	3
7	Wi-Fi Module (ESP8266 or ESP32)	1
8	Power Supply or USB Cable	1
9	9V Battery (for Pump)	1

Procedure:

Step1:Connect the soil moisture sensor to Arduino to measure soil moisture levels.

Step2:Interface the water pump with the Arduino using a relay module for automated water supply.

Step3:Attach the Wi-Fi module for IoT connectivity to monitor data remotely.
Step4:Write and upload code using Arduino IDE to automate watering based on sensor data.
Step5:Link the setup to an IoT platform for real-time monitoring and control.
Step6:Test the system by varying soil moisture levels and observing pump activation.

Program:

```
int soilMoistureValue = 0;
int percentage=0;
void setup() {
  pinMode(3,OUTPUT);
  Serial.begin(9600);
}
void loop() {
soilMoistureValue = analogRead(A0);
Serial.println(percentage);
```

```
percentage = map(soilMoistureValue, 490, 1023, 100, 0);
if(percentage < 10)
{
  Serial.println(" pump on");
  digitalWrite(3,LOW);
}
if(percentage >80)
{
  Serial.println("pump off");
  digitalWrite(3,HIGH);
}
}
```

Circuit Diagram:

Output:

Result:

Ex.No:8 **IOT BASED SMART WASTE MANAGEMENT SYSTEM FOR SMART CITY**
Date:

Aim:

To develop an IoT-based smart waste management system using Arduino to monitor waste levels in bins and optimize waste collection in smart cities.

Apparatus Required:

S.No	Apparatus	Quantity
1	Arduino Uno or Mega	1
2	Ultrasonic Sensor	1
3	Wi-Fi Module (ESP8266 or ESP32)	1
4	Buzzer or LED Indicator	1(Choose one)
5	Jumper Wires	10
6	Breadboard	1
7	Power Supply or USB Cable	1
8	9V Battery (Optional for Independent Modules)	1

Procedure:

Step1:Connect ultrasonic sensors to Arduino to measure the waste levels in bins.
Step2:Attach a Wi-Fi module for transmitting data to an IoT platform.
Step3:Add a buzzer or LED indicator to signal when the bin is full.
Step4:Write and upload code using Arduino IDE for monitoring and transmitting waste levels.
Step5:Link the setup to an IoT platform to display bin status in real-time.
Step6:Test the system by filling bins to different levels and observing data transmission and alerts.

Program:

```
#include   //INCLUDES SERVO LIBRARY

Servo servo;
int trigPin = 5;
int echoPin = 6;
int servoPin = 9;
long duration, distance, average;
long averDist[3];

void setup() {
  Serial.begin(9600);
  servo.attach(servoPin);
```

```
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  servo.write(0); //CLOSES CAP ON STARTING
  delay(100);
  servo.detach();
}

void measure() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(15);
  digitalWrite(trigPin, LOW);
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);
  distance = (duration / 2) / 29.1; //CALCULATES DISTANCE
}

void loop() {
  Serial.println(distance);
  for (int i = 0; i <= 2; i++) { //CALCULATES AVERAGE DISTANCE
    measure();
    averDist[i] = distance;
    delay(10);
  }
  distance = (averDist[0] + averDist[1] + averDist[2]) / 3;

  if ( distance <= 30 ) {
    servo.attach(servoPin);
    delay(1);
    servo.write(180);
    delay(3500);
  }
  else
    servo.write(0);
    delay(1500);
    servo.detach();
}
```

Circuit Diagram:

Output:

Result:

Ex.No:9
Date: **IOT BASED SMART WATER MANAGEMENT SYSTEM**

Aim:

To design and implement an IoT-based smart water management system using Arduino to monitor water levels, detect leakages, and optimize water usage.

Apparatus Required:

S.No	Apparatus	Quantity
1	Arduino Uno or Mega	1
2	Ultrasonic Sensor	1
3	Water Flow Sensor	1
4	Wi-Fi Module (ESP8266 or ESP32)	1
5	Buzzer or LED Indicator	1(Choose one)
6	Relay Module	1
7	Jumper Wires	10
8	Breadboard	1
9	Power Supply or USB Cable	1

Procedure:

Step1:Connect the ultrasonic sensor to monitor water tank levels.
Step2:Attach a water flow sensor to measure water usage and detect leakages.
Step3:Interface a Wi-Fi module for IoT connectivity to transmit data.

Step4:Write and upload Arduino IDE code for real-time water monitoring and alerts.

Step5:Link the system to an IoT platform for data visualization and control.
Step6:Test the system for water level changes and flow rates to ensure proper functioning.

Program:

```
#define sensorPin A0
int leds[] = {2, 3, 4};

void setup() {
  Serial.begin(9600);
  for (int i = 0; i < 3; i++) pinMode(leds[i], OUTPUT);
}

void loop() {
```

```
  int val = analogRead(sensorPin);
  Serial.println(val);

  // Reset all LEDs
  for (int i = 0; i < 3; i++) digitalWrite(leds[i], LOW);

  // Indicate water level
  if (val < 280) digitalWrite(leds[0], HIGH);
  else if (val < 564) digitalWrite(leds[1], HIGH);
  else if (val < 640) digitalWrite(leds[2], HIGH);

  delay(100);
}
```

Circuit Diagram:

Output:

Result:

CONTENT BEYOND SYLLABUS

Ex.No 10 **ARDUINO MEGA AND SERVO MOTORS**
Date:

Aim:

To design and implement a system using Arduino Mega to control a servo motor, Allowing precise positioning based on input values.

Apparatus Required:

S.No	Apparatus	Quantity
1	Arduino Mega	1
2	Servo Motor	1
3	Jumper Wires	10
4	USB Cable	1
5	Arduino IDE Tool (Software)	1

Procedure:

Step 1: Initialize Hardware: Initialize Arduino Mega, connect and initialize the servo motor using the Servo library, and connect the potentiometer to an analog input pin on the Arduino.
Step 2: Setup: In the setup() function, set up serial communication for debugging (if needed) and attach the servo motor to a specific PWM pin using servo.attach(pinNumber).
Step 3: Read Input: In the loop() function, read the potentiometer value using analogRead(pinNumber), where the value ranges from 0 to 1023.
Step 4: Process Input: Map the potentiometer value to the servo motor's angle range (0° to 180°) using the map() function with ServoAngle = map(potValue, 0, 1023, 0, 180) and ensure the mapped angle is within the servo's acceptable range.
Step 5: Control Servo: Set the servo position to the mapped angle using servo.write(angle), which adjusts the servo's angle based on the potentiometer input.

Program:

```
#include <Servo.h>
Servo myServo;
void setup() {
myServo.attach(9);
}
void loop() {
  myServo.write(0);
  delay(1000);
```

```
  myServo.write(90);
  delay(1000);
  myServo.write(180);
  delay(1000);
}
```

Circuit Diagram:

Output:

Result: