Lab Manual

Internet of Things Lab

(MCA-135)

2024-25

Submitted by:

Adarsh Kumar Gaur

524110032

1st sem

Submitted to: Dr. Suresh

lab in-charge : Aman Jahkar Sir

Department of Computer Applications

National Institute of Technology

Kurukshetra (136119)

EXPERIMENT NO-1

Objective:

Familiarization with development board Arduino and perform necessary software installation.

Theory:

Introduction to IoT Device: -

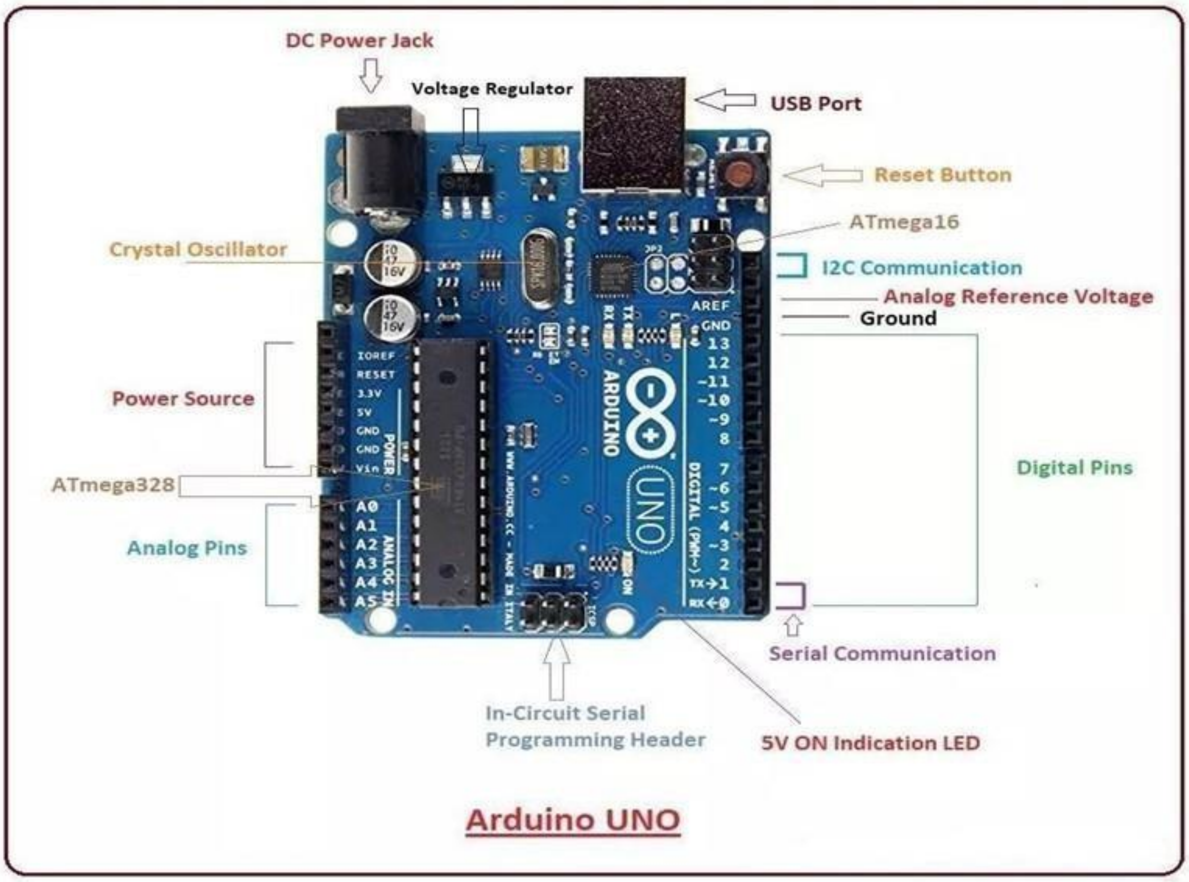
1. Arduino

It is a microcontroller board developed by Arduino.cc and is based on Atmega328 Microcontroller. The first Arduino project was started in Interaction Design Institute Ivrea in 2003 by David Cuartielles and Massimo Banzi with the intention of providing a cheap and flexible way for students and professionals to learn embedded programming.

Arduino UNO is a very valuable addition in electronics that consists of a USB interface, 14 digital I/O pins(of which 6 Pins are used for PWM), 6 analog pins and an Atmega328 microcontroller. It also supports 3 communication protocols named Serial, I2C and SPI protocol.

● Few main features of Arduino UNO are shown in the below figure:

Arduino UNO Features and Technical Specs		
No.	**Parameter Name**	**Parameter Value**
1	Microcontroller	Atmega328
2	Crystal Oscillator	16MHz
3	Operating Voltage	5V
4	Input Voltage	5-12V
5	Digital I/O Pins	14 (D0 to D13)
6	Analog I/O Pins	6 (A0 to A5)
7	PWM Pins	6 (Pin # 3, 5, 6, 9, 10, 11)
8	Power Pins	5V, 3.3V, Vin, GND
9	Communication	UART(1), SPI(1), I2C(1)
10	Flash Memory	32 KB (0.5KB used by bootloader)
11	SRAM	2 KB
12	EEPROM	1 KB
13	ICSP Header	Yes
14	Power sources	DC Power Jack & USB Port



Arduino UNO Pin Description

There are several I/O digital and analog pins placed on the board which operate at 5V. These pins come with standard operating ratings ranging between 20mA to 40mA. Internal pull-up resistors are used in the board that limits the current exceeding the given operating conditions. However, too much increase in current makes these resistors useless and damages the device.

- **LED**: Arduino Uno comes with a built-in LED which is connected through pin 13. Providing HIGH value to the pin will turn it ON and LOW will turn it OFF.
- **Vin**: It is the input voltage provided to the Arduino Board. It is different than 5 V supplied through a USB port. This pin is used to supply voltage. If a voltage is provided through a power jack, it can be accessed through this pin.
- **5V**: This board comes with the ability to provide voltage regulation. The 5V pin is used to provide output regulated voltage. The board is powered up using three ways: USB, Vin pin of the board, or DC power jack. USB supports voltage around 5V while Vin and Power Jack support a voltage range between 7V to 20V. It is recommended to operate the board on 5V. It is important to note that if a voltage is supplied through 5V or 3.3V pins, it results in bypassing the voltage regulator that can damage the board if the voltage surpasses its limit.
- **GND**: These are ground pins. More than one ground pin is provided on the board which can be used as per requirement.

- **Reset**: This pin is incorporated on the board which resets the program running on the board. Instead of a physical reset on the board, the IDE comes with a feature of resetting the board through programming.
- **IOREF**: This pin is very useful for providing voltage reference to the board. A shield is used to read the voltage across this pin which then selects the proper power source.
- **PWM**: PWM is provided by pins 3, 5, 6, 9, 10, and 11. These pins are configured to provide 8-bit output PWM.
- **SPI**: It is known as Serial Peripheral Interface. Four pins provide SPI communication with the help of the SPI library:
    - 10 (SS)
    - 11 (MOSI)
    - 12 (MISO)
    - 13 (SCK)
- **AREF**: It is called Analog Reference. This pin is used for providing a reference voltage to the analog inputs.
- **TWI**: It is called Two-wire Interface. TWI communication is accessed through the Wire Library. Pins A4 and A5 are used for this purpose.
- **Serial Communication**: Serial communication is carried out through two pins called Pin 0 (Rx) and Pin 1 (Tx).
    - Rx pin is used to receive data while Tx pin is used to transmit data.
- **External Interrupts**: Pins 2 and 3 are used for providing external interrupts. An interrupt is called by providing LOW or changing value.

Communication and Programming
Arduino Uno comes with the ability to interface with other Arduino boards, microcontrollers, and computers. The Atmega328 placed on the board provides serial communication using pins like Rx and Tx. The Atmega16U2 incorporated on the board provides a pathway for serial communication using USB COM drivers. A serial monitor is provided on the IDE software, which is used to send or receive text data from the board. If the LEDs placed on the Rx and Tx pins flash, they indicate the transmission of data.
Arduino Uno is programmed using Arduino Software, a cross-platform application called IDE written in Java. The AVR microcontroller Atmega328 laid out on the base comes with a built-in bootloader that sets you free from using a separate burner to upload the program on the board.

2. Motor
The motor is used to control the flow of water from a water source (such as a reservoir or irrigation system) to the crops. It is typically a DC motor that can be turned on or off to start or stop the water flow. The motor is connected to a relay, which acts as a switch controlled by the Node MCU. When the Node MCU determines that irrigation is required based on the soil moisture sensor readings, it activates the relay, which in turn powers the motor to start the water flow. Similarly, when irrigation is no longer needed, the Node MCU deactivates the relay, stopping the motor and halting the water flow.

3. Relay

A relay is an electrically operated switch in the context of electronics and electrical engineering. It is made up of a coil and one or more contact sets. When an electrical current runs through the coil, a magnetic field is created that activates the contacts, allowing them to open or close the circuits. Relays are frequently employed in automotive applications, industrial automation, and control systems to control high-power or high-voltage devices with a lower-power signal.

Figure 2. Motor

Figure 3: Relay

4. Push-Button

A push button is a type of switch that is activated by pressing it. It is commonly used in electronic devices, control panels, and other applications where a user needs to initiate an action or give a command. Push buttons come in different sizes, shapes, and colors, and can be designed to be momentary (meaning they only stay in the on position as long as they are being pressed) or latching (meaning they stay in the on position after being pressed until they are pressed again to turn off). They can also be illuminated or non-illuminated. They can be used to control the relay, which can then control the 5V DC motor as well as other electronic devices.

Figure 4. Push Button

5. Node MCU (ESP8266-based Microcontroller)

Node MCU is an open-source firmware and development kit that enables easy prototyping of IoT (Internet of Things) projects. It is based on the ESP8266 Wi-Fi module, which provides both processing power and Wi-Fi connectivity. Node MCU is programmed using the Arduino IDE or Lua scripting language, making it accessible to a wide range of developers.

The NodeMCU is a versatile development board widely utilized in the realm of Internet of Things (IoT) projects. It is built upon the ESP8266 Wi-Fi module, offering a compact yet powerful platform for prototyping and deploying IoT applications. The board integrates a microcontroller unit (MCU) compatible with the Arduino Integrated Development Environment (IDE), simplifying the development process for both beginners and experienced developers. One of its key features is the inclusion of builtin Wi-Fi connectivity, allowing seamless communication with other devices and the internet. Additionally, the NodeMCU provides a range of General Purpose Input/Output (GPIO) pins, enabling interfacing with various sensors, actuators, and peripheral devices. Its USB connectivity facilitates convenient power supply and programming, streamlining the development workflow. With its affordability, ease of use, and robust community support, the NodeMCU has become a preferred choice for IoT enthusiasts and professionals alike, enabling the rapid prototyping and deployment of innovative IoT solutions across diverse domains.

6. DHT11 Sensor

The DHT11 is a simple, digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding temperature and humidity and gives out a digital signal on the data pin. It is very simple to use but requires careful timing to grab data. The only real downside of this sensor is that we can only get new data from it every 2 seconds, so when using the library, sensor readings can have up to a 2-second delay. The sensor is designed to receive commands through a microcontroller, either Arduino or Node MCU, and give back serial data output. We can program it to provide data in degrees Celsius or degrees Fahrenheit.

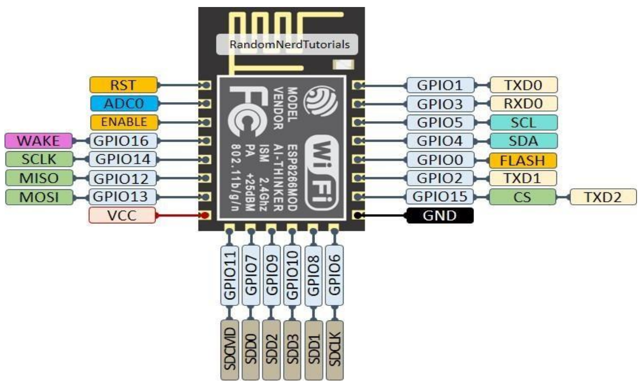


Figure 5. NodeMCU Pin Diagram

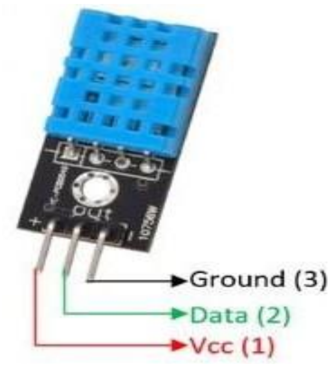


Figure 6. DHT11 Sensor

7. IR Sensors (Infrared Sensors)

IR sensors detect infrared radiation emitted or reflected by objects, enabling proximity sensing, motion detection, and object tracking in various applications such as automated systems, security devices, and smart home appliances. They offer fast response times, cost-effectiveness, and non-contact sensing capabilities but may be susceptible to environmental factors and have limited detection ranges.

8. Buzzer

Provides audible feedback to indicate entry and exit events. It is connected to a digital pin of the Arduino board.

Figure 7. IR Sensor

Figure 8. Buzzer

9. LEDs (Light Emitting Diodes)
Controlled by the Arduino to adjust the lighting based on detected entry and exit. These are connected to a digital pin of the Arduino board.

10. Soil Moisture Sensor
The soil moisture sensor typically consists of two electrodes that measure the electrical conductivity of the soil. As the soil moisture content changes, the conductivity between the electrodes varies, allowing the sensor to detect moisture levels. The sensor outputs an analog voltage proportional to the soil moisture, which is read by the Node MCU's analog-to-digital converter (ADC).

Figure 9. LED

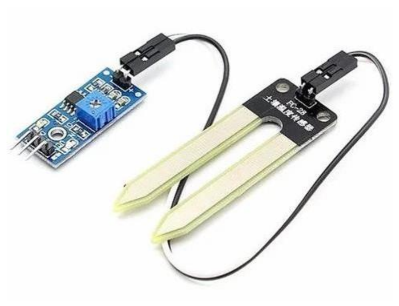


Figure 10. Soil Moisture Sensor

11. Power Supply
The system requires a stable power supply to operate reliably. The Node MCU, soil moisture sensor, relay, and motor all require appropriate voltage levels and current capacities. A power source such as a battery, solar panel, or mains power supply can be used, depending on the specific application and environmental conditions.

Software Installation for Arduino

Download Software: Go to Arduino Software and follow these steps:

Step 1:
Click on **I Agree**.

Step 2:
Select **Only for me** and click on **Next**.

Step 3:
Provide the destination folder and click on **Install**.

Step 4:
(This step appears to be missing; please include any necessary instructions here.)

Step 5:
Click on **Finish**.

The installation process is now complete, and you are able to use the Arduino IDE setup.

Software Installation for Raspberry Pi

1. **Download Raspberry Pi Imager**:
   Visit the Raspberry Pi website to download the Imager tool.
2. **Prepare SD Card**:
   - Insert the SD card into your computer.
   - Use the Imager to install Raspberry Pi OS on the SD card. This tool will format the SD card and copy the necessary files.
3. **Set Up Raspberry Pi**:
   - Insert the SD card into the Raspberry Pi.
   - Connect the peripherals (monitor, keyboard, mouse).
   - Power on the Raspberry Pi. The device should boot into the Raspberry Pi OS.

Results

1. Successful installation of Arduino IDE.
2. Successful installation of Raspberry Pi OS and completion of the initial setup.

Observations

1. Both development boards were set up successfully and are ready for further experimentation.
2. The initial setup confirms that the Raspberry Pi is ready for use.

Precautions

- **Power Supply**: Ensure a stable power supply to all components to prevent system failures.
- **Calibration**: Proper calibration is essential to ensure that the system accurately responds to soil moisture levels.
- **Maintenance**: Regular maintenance is required to keep the system functioning optimally, including sensor calibration and checking for any hardware issues.
- **Drivers**: Ensure the correct drivers are installed for the Arduino board if the IDE does not recognize it.
- **Power for Raspberry Pi**: Use a reliable power supply for the Raspberry Pi to prevent power-related issues.
- **Physical Care**: Handle the boards carefully to avoid physical damage.

EXPERIMENT NO-2

Objective:

To interface an LED with Arduino UNO and write a program to turn ON the LED for 1 second after every 2 seconds.

Apparatus Required:

Arduino UNO, Resistance (330 ohm), LED, Breadboard, Connecting wires, Power Supply

Theory:

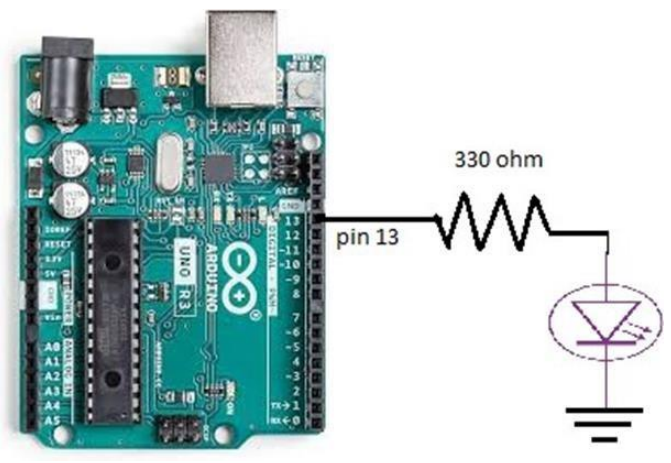
An LED is a simple diode that emits light in forward bias. When a voltage is applied to a PN Junction Diode, electrons and holes recombine in the PN Junction, releasing energy in the form of light (photons). An LED's electrical behavior is similar to that of a PN Junction Diode. When free electrons in the conduction band recombine with holes in the valence band in forward bias, energy is released as light.

The Arduino UNO is built around the ATmega328P microcontroller and is widely regarded as user-friendly compared to other boards, such as the Arduino Mega. It features 6 analog pin inputs, 14 digital pins, a USB connector, a power jack, and an ICSP (In-Circuit Serial Programming) header. Due to its popularity and standardized form factor, the Arduino UNO is one of the most commonly used boards among available Arduino options, especially recommended for beginners due to its ease of use.

Program:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT); // initialize digital pin LED_BUILTIN as an
output.
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on
  delay(1000); // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off
  delay(2000); // wait for two seconds
}
```

Connection Diagram:



Procedure:

1. Open the Arduino IDE.
2. Write the program.
3. Select the board as Arduino UNO.
4. Click on the verify icon.
5. Upload the program to the Arduino.
6. Make connections as per the circuit diagram.

Results:

This setup allows you to control the LED, turning it on for 1 second and off for 2 seconds in a continuous loop.

Observations:

1. Users can turn on the LED for 1 second and turn it off for 2 seconds.
2. This project provides a practical example of IoT application development.

Precautions:

- **Power Supply**: Ensure a stable power supply to all components to prevent system failures.
- **Maintenance**: Regular maintenance is required to keep the system functioning optimally, including sensor calibration and checking for any hardware issues.

EXPERIMENT NO-3

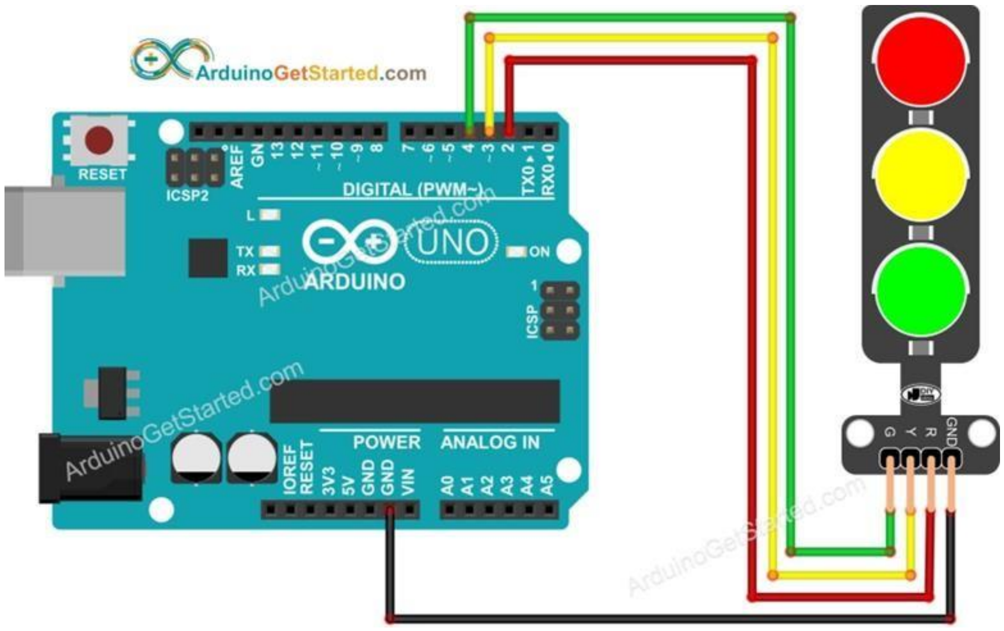
Objective:
Simulate a traffic light with red, yellow, and green using RGB LEDs, changing states after specific intervals.

Apparatus Required:
Arduino UNO, Resistance (330 ohm), RGB LEDs, Breadboard, Connecting wires, Power Supply

Theory:
An RGB LED combines three LEDs—red, green, and blue—into a single package. By varying the intensity of each color, an RGB LED can produce a wide spectrum of colors, allowing for rich and vibrant lighting effects. Each color can be controlled independently using Pulse Width Modulation (PWM), enabling the creation of custom colors by mixing different levels of red, green, and blue. RGB LEDs are commonly used in projects that require dynamic and customizable lighting.

Program:
```
#define PIN_RED 2 // The Arduino pin connected to R pin of traffic light module
#define PIN_YELLOW 3 // The Arduino pin connected to Y pin of traffic light module
#define PIN_GREEN 4 // The Arduino pin connected to G pin of traffic light module
#define RED_TIME 4000 // RED time in milliseconds
#define YELLOW_TIME 4000 // YELLOW time in milliseconds
#define GREEN_TIME 4000 // GREEN time in milliseconds

void setup() {
  pinMode(PIN_RED, OUTPUT);
  pinMode(PIN_YELLOW, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);
}

void loop() {
  // Red light on
  digitalWrite(PIN_RED, HIGH); // turn on
  digitalWrite(PIN_YELLOW, LOW); // turn off
  digitalWrite(PIN_GREEN, LOW); // turn off
  delay(RED_TIME); // keep red light on during a period of time

  // Yellow light on
  digitalWrite(PIN_RED, LOW); // turn off
  digitalWrite(PIN_YELLOW, HIGH); // turn on
  digitalWrite(PIN_GREEN, LOW); // turn off
  delay(YELLOW_TIME); // keep yellow light on during a period of time

  // Green light on
  digitalWrite(PIN_RED, LOW); // turn off
  digitalWrite(PIN_YELLOW, LOW); // turn off
  digitalWrite(PIN_GREEN, HIGH); // turn on
  delay(GREEN_TIME); // keep green light on during a period of time
}
```

Connection Diagram:



Procedure:

1. Open the Arduino IDE.
2. Write the program.
3. Select the board as Arduino UNO.
4. Click on the verify icon.
5. Upload the program to the Arduino.
6. Make connections as per the circuit diagram.

Results:

This setup allows you to control the RGB LED and change its color every 4 seconds.

Observations:

1. Users can turn on the RGB LED for 4 seconds and turn it off for 4 seconds.
2. This project provides a practical example of IoT traffic lights application development.

Precautions:

- **Power Supply**: Ensure a stable power supply to all components to prevent system failures.
- **Maintenance**: Regular maintenance is required to keep the system functioning optimally, including sensor calibration and checking for any hardware issues.

EXPERIMENT NO-4

Objective:

To interface the Push Button/Digital Sensor (IR/LDR) with a development board (Arduino/Raspberry Pi) and write a program to turn ON an LED when the push button is pressed or when the sensor detects an object.

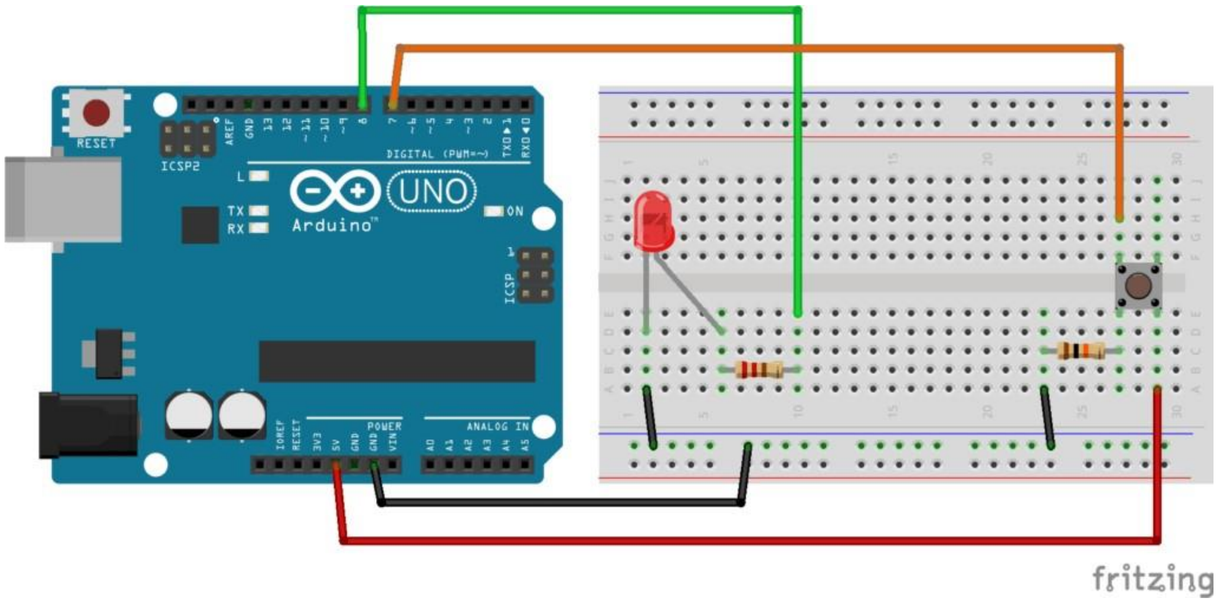
Apparatus Required:

Arduino UNO, Push Button, LED, Resistance (330 ohm), Breadboard, Connecting wires, Power Supply

Theory:

A push button is a simple switch that, when pressed, allows current to flow through the circuit, activating connected components, such as LEDs. Similarly, digital sensors like IR (Infrared) and LDR (Light Dependent Resistor) can detect changes in light or the presence of an object. The Arduino can be programmed to read the state of the push button or the sensor and turn the LED ON when the button is pressed or when an object is detected.

Circuit Diagram:



Connections:

1. **Push Button**: Connect one terminal of the push button to a digital pin (e.g., Pin 2) on the Arduino and the other terminal to GND. Use a pull-up resistor (10k ohm) to connect the digital pin to 5V.
2. **LED**: Connect the anode of the LED to another digital pin (e.g., Pin 13) and the cathode to GND through a 330-ohm resistor.

Program:

```
#define BUTTON_PIN 2 // Pin for push button
#define LED_PIN 13   // Pin for LED

void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP); // Set button pin as input with internal pull-
up resistor
```

```
  pinMode(LED_PIN, OUTPUT);           // Set LED pin as output
}

void loop() {
  int buttonState = digitalRead(BUTTON_PIN); // Read the state of the push button

  if (buttonState == LOW) { // Button is pressed
    digitalWrite(LED_PIN, HIGH); // Turn ON the LED
  } else {
    digitalWrite(LED_PIN, LOW); // Turn OFF the LED
  }
}
```

Procedure:

1. Open the Arduino IDE.
2. Write the program as specified above.
3. Select the board as Arduino UNO.
4. Click on the verify icon to check for errors.
5. Upload the program to the Arduino.
6. Make connections as per the circuit diagram.

Results:

The LED will turn ON when the push button is pressed and turn OFF when it is released. If using an IR or LDR sensor, the LED will turn ON when an object is detected (for IR) or when light intensity changes (for LDR).

Observations:

1. The LED responds to the push button or sensor detection as intended.
2. This experiment demonstrates the basic interfacing of digital sensors with a microcontroller.

Precautions:

- **Power Supply**: Ensure a stable power supply to all components to prevent system failures.
- **Connections**: Double-check connections to avoid short circuits or incorrect wiring.
- **Button Debouncing**: If the button response is erratic, consider implementing debouncing in the code to filter out noise.

EXPERIMENT NO-5

Objective:

To interface the DHT11 sensor with a development board (Arduino/Raspberry Pi) and write a program to print temperature and humidity readings.

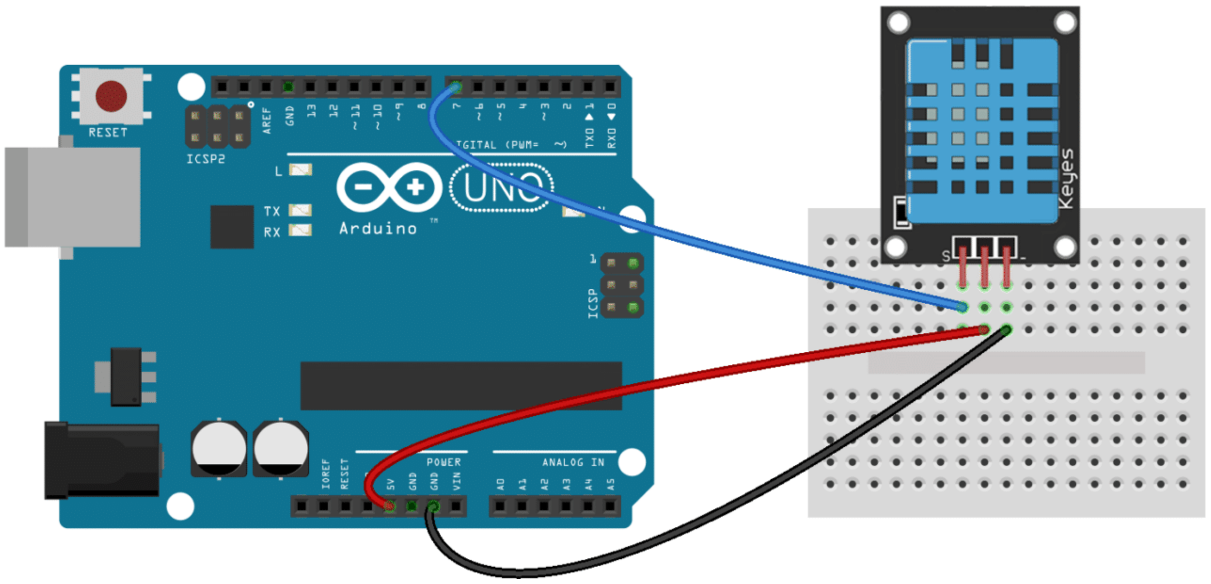
Apparatus Required:

Arduino UNO, DHT11 Sensor, Breadboard, Connecting wires, Power Supply, 10k ohm Pull-up Resistor (optional)

Theory:

The DHT11 is a digital temperature and humidity sensor that provides accurate and reliable readings. It communicates with microcontrollers using a single-wire interface. The sensor uses a capacitive humidity sensor and a thermistor to measure the surrounding temperature and humidity, providing outputs in a digital format. The Arduino can be programmed to read the data from the DHT11 sensor and display the temperature and humidity values.

Circuit Diagram:



Connections:

1. **DHT11 Sensor**: Connect the VCC pin to the 5V pin on the Arduino, the GND pin to GND, and the DATA pin to a digital pin on the Arduino (e.g., Pin 7). If necessary, connect a 10k ohm pull-up resistor between the VCC and DATA pins.

Program:

To use the DHT11 sensor, you need to include the DHT library. Make sure to install the DHT sensor library from the Library Manager in the Arduino IDE before uploading the code.

```
#include <DHT11.h>

// Create an instance of the DHT11 class.
DHT11 dht11(2);

void setup() {
    // Initialize serial communication to allow debugging and data readout.
    // Using a baud rate of 9600 bps.
    Serial.begin(9600);

}

void loop() {
    int temperature = 0;
    int humidity = 0;

    // Attempt to read the temperature and humidity values from the DHT11 sensor.
    int result = dht11.readTemperatureHumidity(temperature, humidity);

    // Check the results of the readings.
    // If the reading is successful, print the temperature and humidity values.
    // If there are errors, print the appropriate error messages.
    if (result == 0) {
        Serial.print("Temperature: ");
        Serial.print(temperature);
        Serial.print(" °C\tHumidity: ");
        Serial.print(humidity);
        Serial.println(" %");
    } else {
        // Print error message based on the error code.
        Serial.println(DHT11::getErrorString(result));
    }
}
```

Procedure:

1. Open the Arduino IDE.
2. Write the program as specified above.
3. Select the board as Arduino UNO.
4. Click on the verify icon to check for errors.
5. Upload the program to the Arduino.
6. Make connections as per the circuit diagram.
7. Open the Serial Monitor to view the temperature and humidity readings.

Results:

The program will print the temperature and humidity readings from the DHT11 sensor to the Serial Monitor.

Observations:

1. The DHT11 sensor provides temperature and humidity readings as expected.
2. This experiment demonstrates the basic interfacing of a temperature and humidity sensor with a microcontroller.

Precautions:

- **Power Supply**: Ensure a stable power supply to all components to prevent system failures.
- **Connections**: Double-check connections to avoid short circuits or incorrect wiring.
- **Sensor Calibration**: Verify the accuracy of the DHT11 readings by comparing them to a reliable reference.

EXPERIMENT NO-6

Objective:
To interface a gear motor with a development board (Arduino/Raspberry Pi) and write a program to control the motor in a clockwise and anticlockwise direction using two buttons.

Apparatus Required:
Arduino UNO, Gear Motor, H-Bridge Motor Driver (e.g., L298N), Two Push Buttons, Resistance (10k ohm), Breadboard, Connecting wires, Power Supply

Theory:
A gear motor is a motor that uses gears to increase torque and reduce speed. To control the direction of the motor, an H-Bridge motor driver is used. The H-Bridge allows the voltage to be applied across the motor in both directions, enabling clockwise and anticlockwise rotation. The Arduino can be programmed to read the state of a push button and control the motor's direction accordingly.

Circuit Diagram:
(Insert circuit diagram here)

Connections:
1. **H-Bridge Motor Driver:**
   - Connect the motor terminals to the output pins of the H-Bridge (e.g., OUT1 and OUT2).
   - Connect the IN1 pin to a digital pin on the Arduino (e.g., Pin 9) and the IN2 pin to another digital pin (e.g., Pin 8).
   - Connect the ENA pin to a PWM-capable digital pin (e.g., Pin 10) on the Arduino to control the motor speed.
   - Connect the GND pin of the motor driver to the GND of the Arduino.
2. **Push Buttons:**
   - Connect one terminal of the first push button to another digital pin on the Arduino (e.g., Pin 2) and the other terminal to GND (button1).
   - Connect one terminal of the second push button to another digital pin on the Arduino (e.g., Pin 6) and the other terminal to GND (button2).
   - Use pull-up resistors (10k ohm) to connect both button pins to 5V.

Program:
```
const int button1 = 2; // Pin for clockwise button
const int button2 = 6; // Pin for anticlockwise button

int IN1 = 9; // Pin to control motor direction 1
int IN2 = 8; // Pin to control motor direction 2
int ENA = 10; // Pin for motor speed control

int aa, bb;

void setup() {
  pinMode(button1, INPUT); // Set button1 as input
  pinMode(button2, INPUT); // Set button2 as input

  pinMode(IN1, OUTPUT); // Set IN1 pin as output
  pinMode(IN2, OUTPUT); // Set IN2 pin as output
  pinMode(ENA, OUTPUT);  // Set ENA pin as output
  digitalWrite(IN1, LOW);  // Initialize IN1 to LOW
  digitalWrite(IN2, LOW);  // Initialize IN2 to LOW
  digitalWrite(ENA, 0);  // Initialize ENA to 0
}

void loop() {
  analogWrite(ENA, 255); // Set maximum speed for the motor

  aa = digitalRead(button1); // Read the state of button1
  bb = digitalRead(button2); // Read the state of button2

  if (aa == HIGH && bb == LOW) { // Button1 is pressed
    digitalWrite(IN1, HIGH); // Rotate motor clockwise
    digitalWrite(IN2, LOW);
  }
  else if (aa == LOW && bb == HIGH) { // Button2 is pressed
    digitalWrite(IN1, LOW); // Rotate motor anticlockwise
    digitalWrite(IN2, HIGH);
  }
  else {
    digitalWrite(IN1, LOW); // Stop the motor if no button is pressed
    digitalWrite(IN2, LOW);
  }
}
```

Procedure:

1. Open the Arduino IDE.
2. Write the program as specified above.
3. Select the board as Arduino UNO.
4. Click on the verify icon to check for errors.
5. Upload the program to the Arduino.
6. Make connections as per the circuit diagram.

Results:

The gear motor will rotate in a clockwise direction when the first button is pressed and in an anticlockwise direction when the second button is pressed. If no buttons are pressed, the motor will stop.

Observations:

1. The motor responds correctly to each button press, rotating in the intended direction.
2. This experiment demonstrates the ability to control a motor's direction using push buttons and an H-Bridge driver.

Precautions:

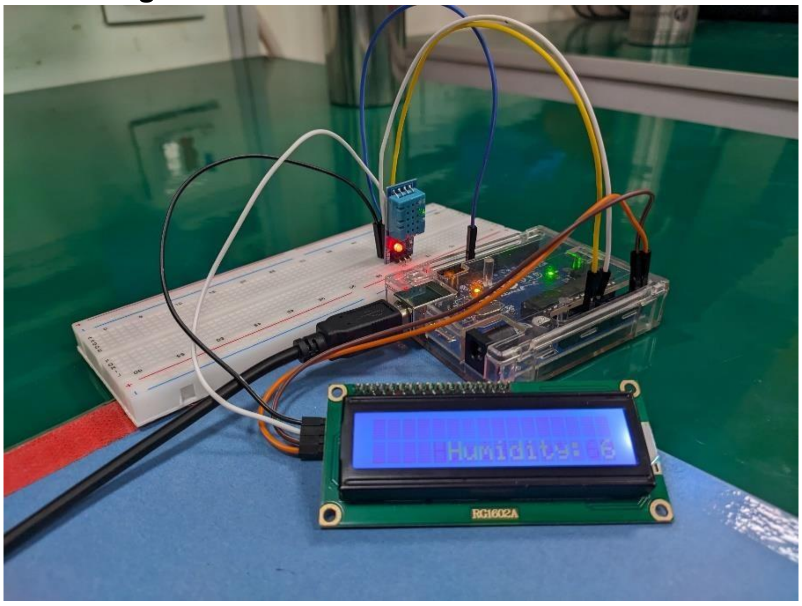
- **Power Supply**: Ensure a stable power supply to all components, especially for the motor.
- **Connections**: Double-check connections to avoid short circuits or incorrect wiring.
- **Motor Load**: Ensure the motor is not overloaded to prevent damage to the driver or the motor itself.

EXPERIMENT NO-7

Objective:
To interface an OLED display with a development board (Arduino/Raspberry Pi) and write a program to print temperature and humidity readings on it.

Apparatus Required:
Arduino UNO, OLED Display (e.g., 0.96" I2C OLED), DHT11 Temperature and Humidity Sensor, Breadboard, Connecting wires, Power Supply

Theory:
An OLED (Organic Light Emitting Diode) display is a type of display that uses organic compounds to emit light when an electric current is applied. OLEDs are known for their high contrast, wide viewing angles, and low power consumption. In this experiment, we will use a DHT11 sensor to read temperature and humidity data and display the readings on the OLED screen.

Circuit Diagram:


Connections:
1. **DHT11 Sensor**:
    - Connect the VCC pin of the DHT11 to the 5V pin on the Arduino.
    - Connect the GND pin to the GND pin on the Arduino.
    - Connect the Data pin to a digital pin on the Arduino (e.g., Pin 2).
2. **OLED Display**:
    - Connect the VCC pin of the OLED display to the 5V pin on the Arduino.
    - Connect the GND pin to the GND pin on the Arduino.
    - Connect the SDA pin to the A4 pin on the Arduino (I2C Data).
    - Connect the SCL pin to the A5 pin on the Arduino (I2C Clock).

Program:
```
#include <LiquidCrystal_PCF8574.h>
#include <DHT.h>

#define DHTPIN 2          // Pin where the DHT11 data pin is connected
#define DHTTYPE DHT11     // DHT 11
```

```
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_PCF8574 lcd(0x27);

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

  // Initialize the OLED display
  lcd.begin(16, 2); // Initialize the display dimensions (modify based on your
OLED)
}

void loop() {
  delay(2000); // Wait a few seconds between readings

  // Read temperature and humidity
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Clear the display
  lcd.clear();

  // Display the temperature and humidity
  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(t);
  lcd.print(" C");

  lcd.setCursor(0, 1);
  lcd.print("Humidity: ");
  lcd.print(h);
  lcd.print(" %");

  // Delay before next reading
  delay(2000);
}
```

Procedure:

1. Open the Arduino IDE.
2. Install the necessary libraries:
   - o DHT sensor library
   - o LiquidCrystal_PCF8574
3. Write the program as specified above.
4. Select the board as Arduino UNO in the IDE.
5. Click the verify icon to check for errors.
6. Upload the program to the Arduino.
7. Make connections as per the circuit diagram.

Results:
The OLED display will show the current temperature and humidity readings from the DHT11 sensor.

Observations:

1. The readings are updated every 2 seconds on the OLED display.
2. This experiment demonstrates the integration of a temperature and humidity sensor with an OLED display.

Precautions:

- **Power Supply**: Ensure a stable power supply to all components.
- **Connections**: Double-check connections to avoid incorrect wiring.
- **Library Installation**: Ensure all necessary libraries are installed in the Arduino IDE for the program to compile successfully.

EXPERIMENT NO.8

Objective
To interface a Bluetooth module with a development board (Arduino/Raspberry Pi) to send real-time sensor data (e.g., temperature, humidity, or distance) to a smartphone via Bluetooth.

Apparatus Required

1. Development board: Arduino Uno/Nano or Raspberry Pi
2. Bluetooth module: HC-05/HC-06 for Arduino or built-in Bluetooth for Raspberry Pi
3. Sensor: DHT11 (temperature and humidity sensor) or ultrasonic sensor
4. Connecting wires
5. Breadboard
6. Power supply (if external)
7. Smartphone with a Bluetooth terminal app (e.g., "Serial Bluetooth Terminal" for Android or iOS)

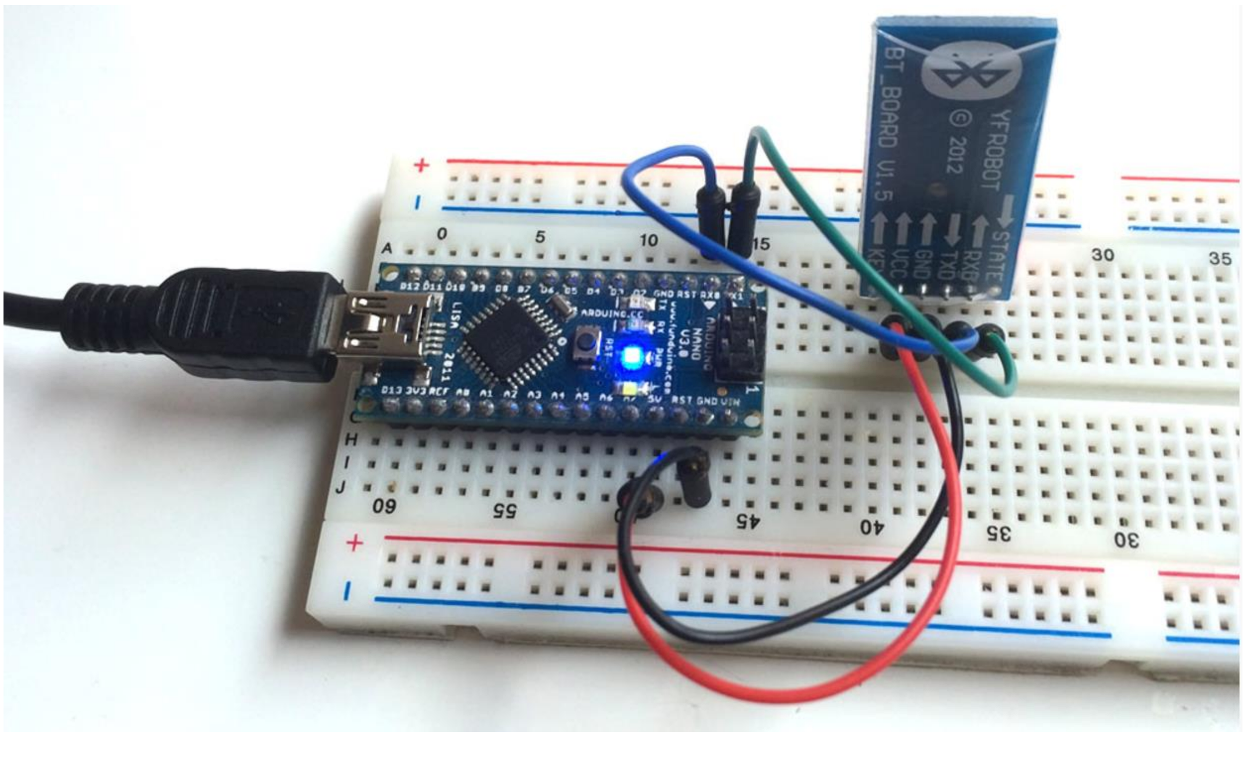
Theory
Bluetooth is a wireless communication protocol enabling data exchange over short distances. A Bluetooth module like HC-05/HC-06 can interface with a development board to establish a serial communication link. Sensor data collected by the board can be transmitted to a smartphone, where it can be displayed using a Bluetooth terminal app.
The process involves:

- Configuring the Bluetooth module for communication.
- Capturing sensor data using the appropriate library for the sensor.
- Sending data over a serial interface via Bluetooth.

Circuit Diagram
For Arduino (with HC-05/HC-06)

- Connect VCC of HC-05 to 5V of Arduino.
- GND of HC-05 to GND of Arduino.
- TXD of HC-05 to RXD (Pin 0) of Arduino (through a voltage divider, if required).
- RXD of HC-05 to TXD (Pin 1) of Arduino.
- Sensor connections to appropriate pins (refer to sensor datasheet).

Connections:



Component:	**Arduino Pin**
HC-05 VCC	5V
HC-05 GND	GND
HC-05 TXD	RX (Pin 0, Arduino) / GPIO RX
HC-05 RXD	TX (Pin 1, Arduino) / GPIO TX
DHT11 VCC	5V
DHT11 GND	GND
DHT11 Data Pin	Digital Pin (e.g., D2)

Program
Arduino Code:
cpp
Copy code

```
#include <SoftwareSerial.h>
#include <DHT.h>

#define DHTPIN 2       // Pin connected to the sensor
#define DHTTYPE DHT11   // DHT11 sensor type

DHT dht(DHTPIN, DHTTYPE);
SoftwareSerial BT(10, 11); // RX, TX for HC-05

void setup() {
  Serial.begin(9600);   // Debugging
  BT.begin(9600);       // Bluetooth communication
  dht.begin();
  Serial.println("System Ready. Waiting for data...");
}

void loop() {
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();

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

  String data = "Temp: " + String(temp) + "C, Humidity: " + String(hum) + "%";
  Serial.println(data);  // Display on Serial Monitor
  BT.println(data);      // Send data over Bluetooth
  delay(2000);           // Send data every 2 seconds
}
```

Procedure

1. Connect the Bluetooth module and sensor to the development board as per the circuit diagram.
2. Load the program onto the development board.
3. Pair the Bluetooth module with the smartphone.
4. Open the Bluetooth terminal app on the smartphone and connect to the Bluetooth module.
5. Observe the data sent from the development board to the smartphone in real time.

Result
Sensor data (e.g., temperature and humidity) was successfully transmitted from the development board to the smartphone via Bluetooth.

Observation

- The data received on the smartphone matches the real-time sensor readings.
- Bluetooth communication establishes a reliable connection within a specific range.

Precautions

1. Ensure the Bluetooth module is correctly paired with the smartphone.
2. Use appropriate voltage levels for the Bluetooth module to avoid damage.
3. Verify sensor connections and library compatibility.
4. Maintain a stable power supply for the development board.
5. Avoid physical obstructions that can weaken Bluetooth signals.

Experiment No.9

Objective:
To interface a Bluetooth module with a microcontroller (Arduino or Raspberry Pi) and write a program to control an LED by sending "1" or "0" commands from a smartphone via Bluetooth communication.

Apparatus Required:

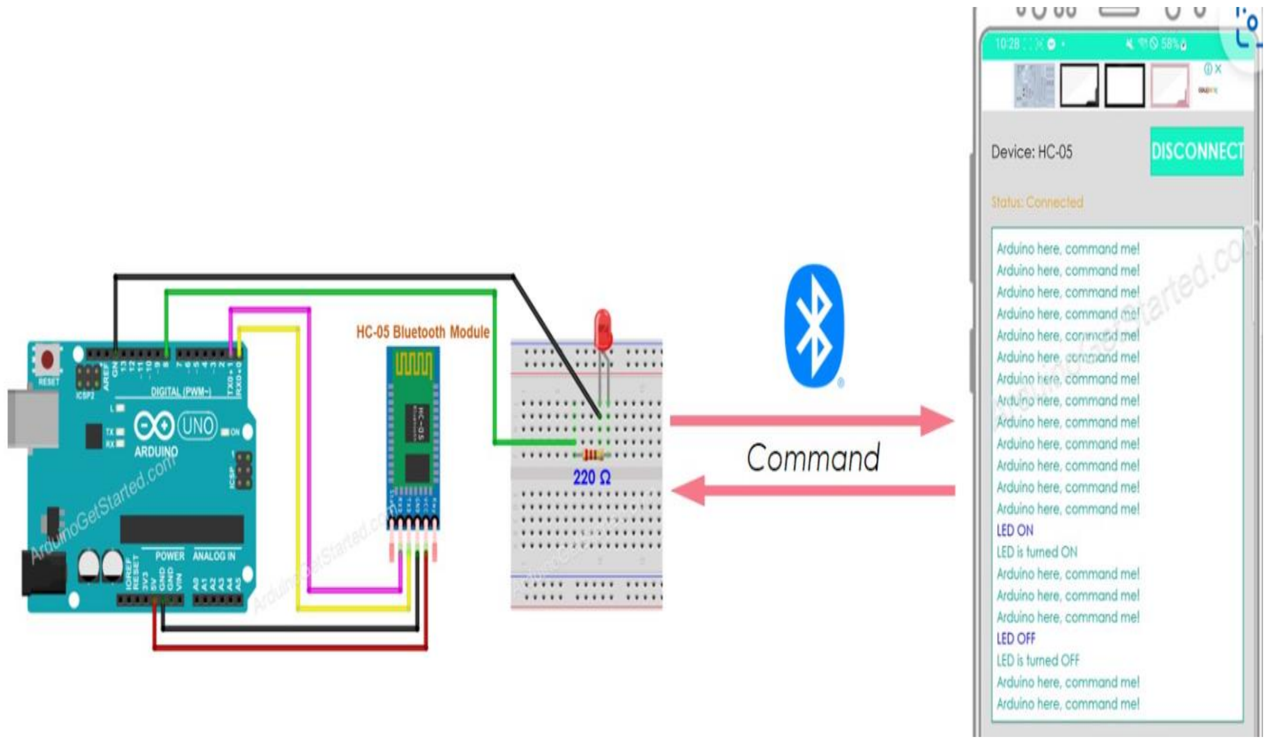
1. **Development Board:** Arduino Uno or Raspberry Pi.
2. **Bluetooth Module:** HC-05 or HC-06 (for Arduino) or a built-in Bluetooth dongle for Raspberry Pi.
3. **LED:** Standard LED (any color).
4. **Resistor:** 220Ω resistor for the LED.
5. **Smartphone:** Android/iOS device with a Bluetooth terminal app.
6. **Jumper wires:** For connections.
7. **Breadboard:** For prototyping.
8. **Power Supply:** 5V for Arduino or appropriate source for Raspberry Pi.

Theory:
Bluetooth modules like the HC-05 provide serial communication over Bluetooth. The microcontroller reads the data sent by the smartphone, interprets it, and performs actions accordingly. In this project:

- Sending "1" turns the LED ON.
- Sending "0" turns the LED OFF.

The system demonstrates simple wireless control over devices using a smartphone.

Circuit and Connections:



```
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
LED ON
LED is turned ON
Arduino here, command me!
Arduino here, command me!
Arduino here, command me!
LED OFF
LED is turned OFF
Arduino here, command me!
Arduino here, command me!
```

For Arduino:

1. **HC-05 Bluetooth Module:**
   - **VCC** → 5V (Arduino)
   - **GND** → GND (Arduino)
   - **TXD** → RX (Arduino pin 0)
   - **RXD** → TX (Arduino pin 1)
2. **LED Circuit:**
   - Connect the LED's longer leg (anode) to a 220Ω resistor, and then to **Digital Pin 13** of the Arduino.
   - Connect the shorter leg (cathode) to **GND**.

Program:
Arduino Code (Using HC-05):

cpp
Copy code

```
char receivedChar;  // Variable to store received character
const int ledPin = 13;  // LED connected to digital pin 13

void setup() {
  Serial.begin(9600);       // Start serial communication
  pinMode(ledPin, OUTPUT);  // Set LED pin as output
  digitalWrite(ledPin, LOW); // Ensure LED is initially OFF
}

void loop() {
  if (Serial.available()) {         // Check if data is available to read
    receivedChar = Serial.read();    // Read the incoming character
    if (receivedChar == '1') {       // Turn LED ON
      digitalWrite(ledPin, HIGH);
      Serial.println("LED ON");
    } else if (receivedChar == '0') { // Turn LED OFF
      digitalWrite(ledPin, LOW);
      Serial.println("LED OFF");
    }
  }
}
```

Procedure:

1. **Assemble the Circuit:** Connect the Bluetooth module, LED, and resistors as per the circuit diagram.
2. **Upload the Code:** Upload the Arduino sketch to the Arduino board or run the Python script on the Raspberry Pi.
3. **Pair Bluetooth Module:** Pair the Bluetooth module with the smartphone.
4. **Install Bluetooth Terminal App:** Use a terminal app (like "Serial Bluetooth Terminal") on the smartphone.
5. **Send Commands:** Open the Bluetooth terminal app and send:
   - "1" to turn the LED ON.
   - "0" to turn the LED OFF.

Result:
When the smartphone sends "1," the LED turns ON. When it sends "0," the LED turns OFF. Feedback messages like "LED ON" or "LED OFF" are displayed on the smartphone app.

Observation:

1. Sending "1" from the smartphone instantly turns the LED ON.
2. Sending "0" from the smartphone instantly turns the LED OFF.
3. Any delay may indicate issues with Bluetooth communication.

Precautions:

1. Ensure proper wiring and secure connections.
2. Use a resistor with the LED to prevent damage.
3. Confirm the pairing between the Bluetooth module and the smartphone.
4. Avoid prolonged operation without heat dissipation for components.
5. Double-check the Bluetooth communication settings (baud rate, pairing code).