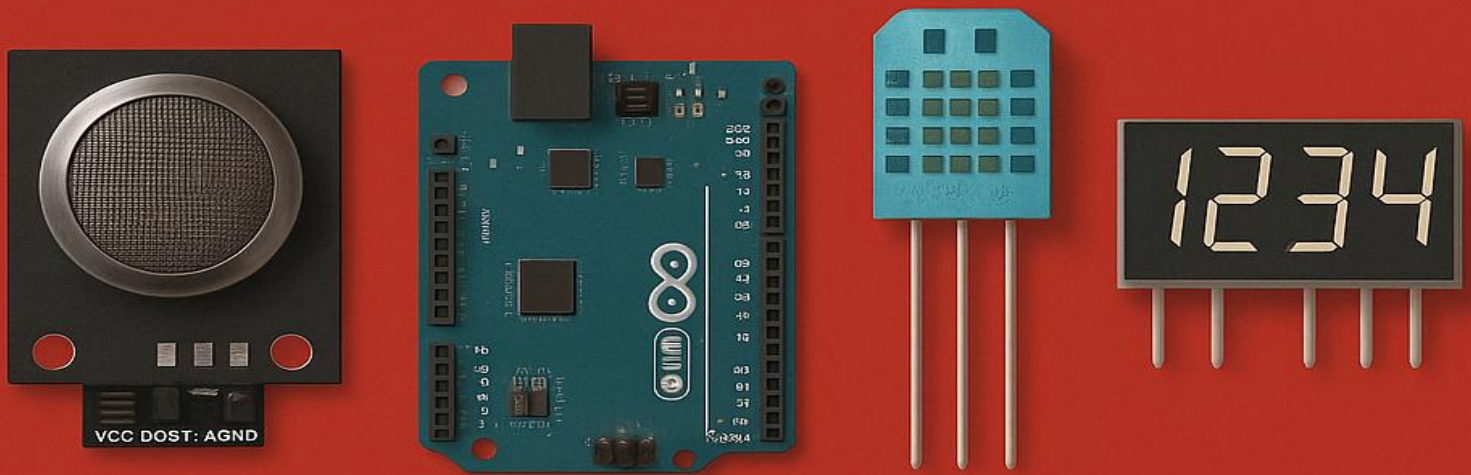


TOP 5 INTERMEDIATE ARDUINO PROJECTS



BY HOBITRONICS

TOP 5 INTERMEDIATE ARDUINO PROJECTS

S. NO	CONTENTS	PAGE NO
1.	16x2 LCD Display Interfacing with Arduino	1
	<ul style="list-style-type: none">• <i>Introduction</i>• <i>Components Required</i>• <i>Wiring Instructions</i>• <i>Arduino Code</i>• <i>Circuit Diagram</i>• <i>Simulation</i>• <i>Output</i>• <i>What You Learn</i>• <i>Real-time Application</i>	
2.	Temperature and Humidity Live Display	5
	<ul style="list-style-type: none">• <i>Introduction</i>• <i>Components Required</i>• <i>Wiring Instructions</i>• <i>Arduino Code</i>• <i>Circuit Diagram</i>• <i>Simulation</i>• <i>Output</i>• <i>What You Learn</i>• <i>Real-time Application</i>	
3.	Soil Moisture Sensor with Water Pump Control	10
	<ul style="list-style-type: none">• <i>Introduction</i>• <i>Components Required</i>• <i>Wiring Instructions</i>• <i>Arduino Code</i>• <i>Circuit Diagram</i>• <i>Simulation</i>• <i>Output</i>• <i>What You Learn</i>• <i>Real-time Application</i>	

4. Gas Sensor with LCD Display 17

- *Introduction*
- *Components Required*
- *Wiring Instructions*
- *Arduino Code*
- *Circuit Diagram*
- *Simulation*
- *Output*
- *What You Learn*
- *Real-time Application*

5. RFID Delivery Tracking System 22

- *Introduction*
- *Components Required*
- *Wiring Instructions*
- *Arduino Code*
- *Circuit Diagram*
- *Simulation*
- *Output*
- *What You Learn*
- *Real-time Application*

HOBITRONICS

NOTE

This guide is perfect for learners who have mastered basic Arduino programming and are ready to apply their skills to real-world, industry-relevant systems.

If you have any queries or suggestions, feel free to reach us at hobitronicsin@gmail.com

Let's begin the fun!

HOBITRONICS

Regards!

Team Hobitronics.

Project 1: 16x2 LCD Display Interfacing with Arduino

Introduction:

In this project, you'll learn how to interface a 16x2 LCD Display with an Arduino. This simple yet powerful project serves as the foundation for future, more complex systems that involve display technologies.

By the end of this project, you'll be able to display custom messages on an LCD screen and gain hands-on experience with **basic interfacing**, **wiring**, and **coding**.

Components Required:

- Arduino UNO
- 16x2 LCD Display (**Non-I2C, 16 pins**)
- 10k Ω Potentiometer (**for contrast adjustment**)
- Breadboard
- Jumper Wires
- USB Cable for Arduino

Wiring Instructions:

Potentiometer (for LCD contrast):

- **Left Pin** → 5V on Arduino
- **Right Pin** → GND on Arduino
- **Middle Pin (wiper)** → Pin 3 (V0) on LCD

16x2 LCD Display (16 Pins):

- **Pin 1 (VSS)** → GND
- **Pin 2 (VDD)** → 5V
- **Pin 3 (VO)** → Middle pin of potentiometer (contrast control)
- **Pin 4 (RS)** → Pin 12 (Arduino)
- **Pin 5 (RW)** → GND
- **Pin 6 (E)** → Pin 11 (Arduino)
- **Pin 7 (D0)** → Not connected
- **Pin 8 (D1)** → Not connected
- **Pin 9 (D2)** → Not connected
- **Pin 10 (D3)** → Not connected
- **Pin 11 (D4)** → Pin 5 (Arduino)
- **Pin 12 (D5)** → Pin 4 (Arduino)
- **Pin 13 (D6)** → Pin 3 (Arduino)
- **Pin 14 (D7)** → Pin 2 (Arduino)
- **Pin 15 (A)** → 5V
- **Pin 16 (K)** → GND

Arduino Code:

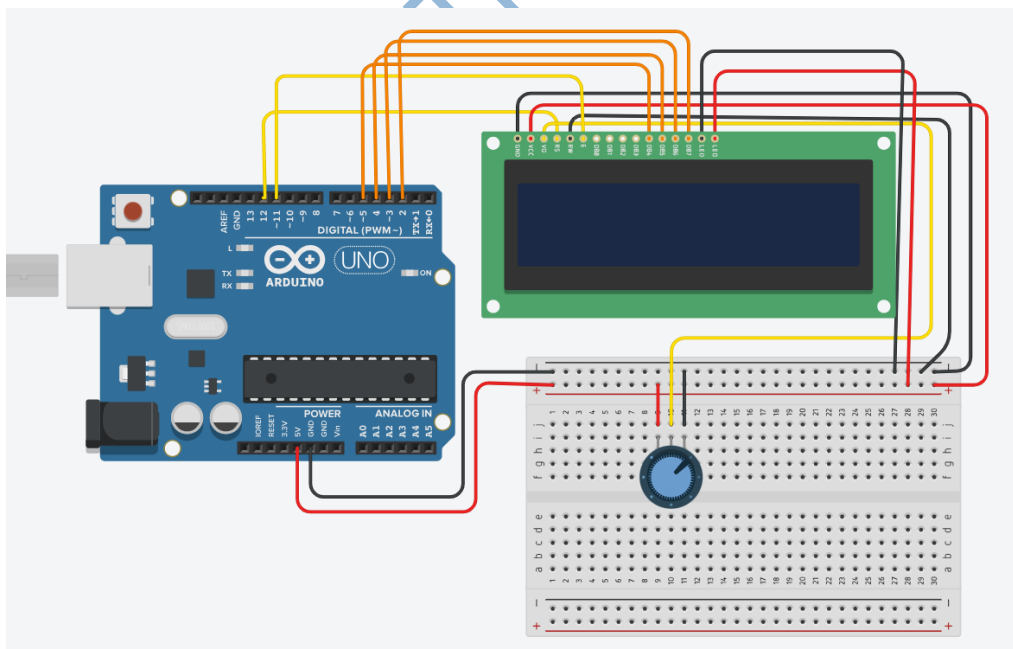
```
#include <LiquidCrystal.h>                                // Include the LCD library

// Pin setup for the LCD: RS, E, D4, D5, D6, D7
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

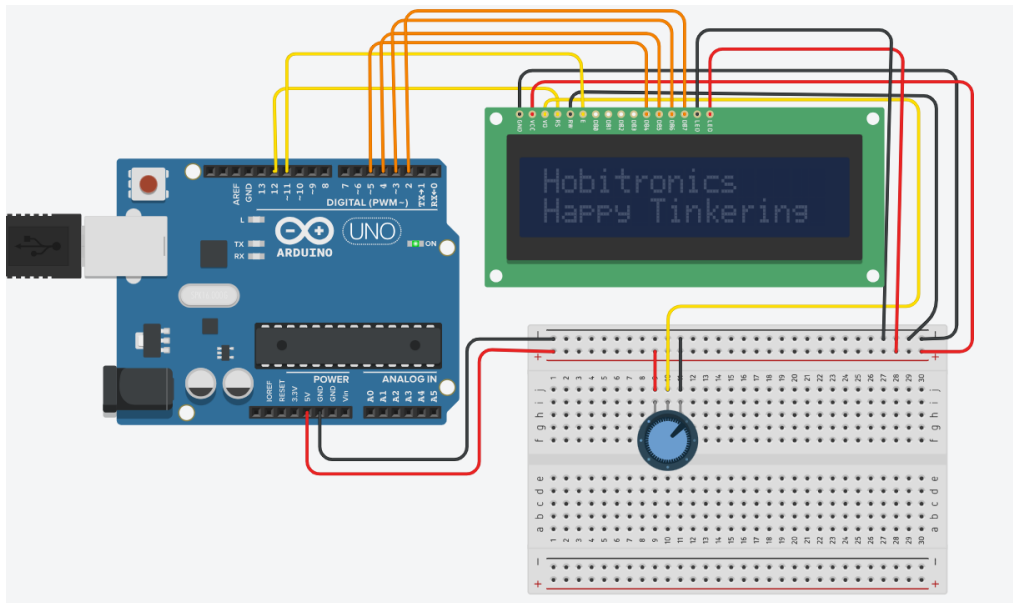
void setup() {
    lcd.begin(16, 2);                                     // Initialize the 16x2 LCD
    lcd.setCursor(0, 0);                                  // Set cursor to the first row
    lcd.print("Hobitronics");                             // Display message on Row 1
    lcd.setCursor(0, 1);                                  // Set cursor to the second row
    lcd.print("Happy Tinkering");                         // Display message on Row 2
}

void loop() {
    //Since we only display the messages, no need for looping, unless clear the lcd
    using lcd.clear()
}
```

Circuit Diagram:



Simulation:



Output:

You should be able to display:

- Row 1: "*Hobitronics*"
- Row 2: "Happy Tinkering"

What You Learn:

By completing this project, you'll gain the following skills:

- **Basic LCD Interfacing:** Learn how to connect and control a 16x2 LCD with Arduino.
- **Contrast Control:** Use a potentiometer to adjust the contrast on the LCD.
- **Arduino Code Structure:** Understand the setup and loop functions in Arduino programming.
- **Cursor Control:** Learn how to move the cursor to different rows and columns on the LCD.
- **Message Display:** Learn how to display custom text on an LCD screen.

Real-World Applications:

1. **Digital Meters:** LCD displays can be used to show readings from various sensors, like temperature, humidity, or gas concentration, making them ideal for use in digital meters.
2. **Home Automation Panels:** Integrate LCDs into smart home systems to display real-time status updates such as "Fan ON", "Door Locked", or system notifications.
3. **Smart Farming:** In agriculture, LCD screens can display crucial environmental conditions like soil moisture levels, air quality, and temperature, helping farmers monitor crops efficiently.
4. **Industrial Machine Monitoring:** LCD displays are often used in factory automation systems to show live status, such as "Motor Overheating" or "System Normal", for efficient maintenance and operation.
5. **DIY Gadgets:** LCDs are perfect for displaying information in personal DIY projects, such as clocks, timers, or even counters, providing live feedback.

HOBITRONICS

Project 2: Temperature and Humidity Live Display

Introduction:

In this project, you'll learn how to interface the **DHT22 sensor** with an Arduino and display real-time temperature and humidity readings on a **16x2 LCD Display**.

This project forms the foundation for **smart home systems**, **environmental monitoring**, and **agriculture automation**, and provides essential skills for working with sensors in various applications.

Components Required:

- Arduino UNO
- DHT22 Temperature & Humidity Sensor
- 16x2 LCD Display
- 10kΩ Potentiometer (**for LCD contrast**)
- Breadboard
- Jumper Wires
- USB Cable for Arduino

Wiring Instructions:

DHT22 Sensor:

- **Pin 1 (VCC)** → 5V (Arduino)
- **Pin 2 (Data Out)** → Pin 7 (Arduino Digital Pin)
- **Pin 3** → No connection
- **Pin 4 (GND)** → GND (Arduino)

Note: Some DHT22 sensors have only 3 pins. In that case, the middle pin will be the digital output.

16x2 LCD Display (same setup as Project 1):

- **Pin 1 (VSS)** → GND
- **Pin 2 (VDD)** → 5V
- **Pin 3 (VO)** → Middle pin of potentiometer
- **Pin 4 (RS)** → Pin 12 (Arduino)
- **Pin 5 (RW)** → GND
- **Pin 6 (E)** → Pin 11 (Arduino)
- **Pins 7-10** → Not connected
- **Pin 11 (D4)** → Pin 5 (Arduino)
- **Pin 12 (D5)** → Pin 4 (Arduino)
- **Pin 13 (D6)** → Pin 3 (Arduino)
- **Pin 14 (D7)** → Pin 2 (Arduino)

- Pin 15 (A, LCD +) → 5V
- Pin 16 (K, LCD -) → GND

Why DHT22 output is connected to D7 and not A0 of Arduino Uno?

Even if the DHT22 Sensor measures analog data from our surroundings, both DHT22 and DHT11 converts analog information to digital format and hence we can directly connect it to our digital pins.

Arduino Code:

```
#include <DHT.h> // Include DHT sensor library
#include <LiquidCrystal.h> // Include LCD library

// DHT sensor setup
#define DHTPIN 7 // Pin connected to DHT22 data pin
#define DHTTYPE DHT22 // Define DHT type as DHT22

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Initialize LCD

void setup() {
    lcd.begin(16, 2); // Start the LCD with 16 columns and 2 rows
    dht.begin(); // Initialize DHT sensor
}

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

    // Check if any readings failed and exit if so
    if (isnan(humidity) || isnan(temperature)) {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Failed to read!");
        return;
    }

    // Display temperature on Row 1
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Temp: ");
    lcd.print(temperature);
    lcd.print(" C");

    // Display humidity on Row 2
```

```

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

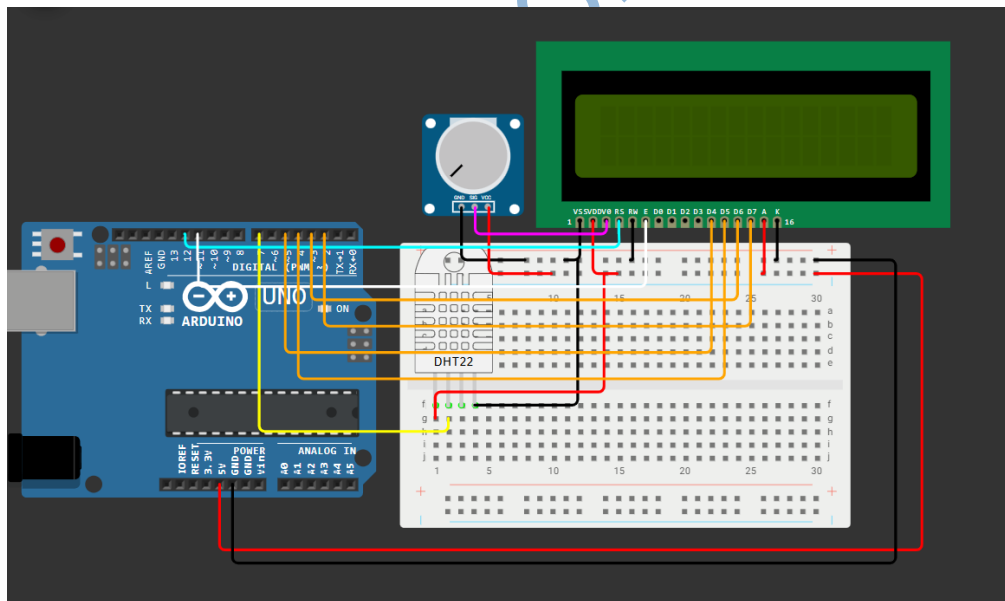
    delay(2000); // Wait for 2 seconds before updating
}

```

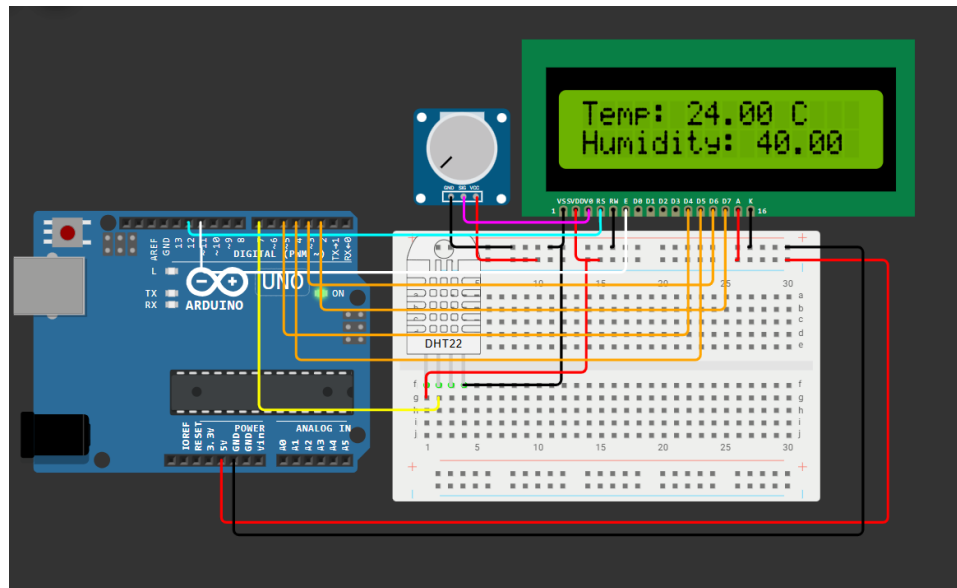
Code Explanation:

- Include the **DHT** sensor library (DHT.h) and the **LiquidCrystal** library for LCD functionality.
- Initialize the **DHT22 sensor** on **Pin 7**.
- Initialize the **16x2 LCD display**.
- Continuously read the **temperature** and **humidity** from the **DHT22** sensor.
- Display the temperature on **Row 1** and humidity on **Row 2** of the LCD.
- Update the values every **2 seconds** to ensure real-time feedback.

Circuit Diagram:



Simulation:



Output:

The LCD will continuously display the following updates:

- **Row 1:** Temperature, e.g., "Temp: 24.00°C"
- **Row 2:** Humidity, e.g., "Humidity: 40%"

These values will update every 2 seconds, providing users with real-time environmental monitoring.

What You Learn:

- **Sensor Interfacing:** Understanding how to interface the **DHT22 sensor** with the **Arduino** to capture environmental data.
- **LCD Display Usage:** Displaying live data on a **16x2 LCD Display**.
- **Real-Time Data:** Working with real-time data and updating the display at regular intervals.
- **Handling Sensors:** Reading and interpreting temperature and humidity data from sensors.

Real-World Applications:

1. **Smart Home Automation:** Monitor room conditions (temperature & humidity) in real time to control AC, fans, and humidifiers automatically.
2. **Agricultural Automation:** Continuously monitor environmental conditions like temperature and humidity in greenhouses, optimizing plant growth and harvest.
3. **Data Logging Systems:** Record temperature and humidity data for research purposes, weather stations, or long-term environmental studies.
4. **HVAC Systems:** Integrate temperature and humidity readings into HVAC systems for automatic adjustments, improving energy efficiency.
5. **Industrial Plant Monitoring:** Ensure that temperature and humidity levels in production lines or storage areas are maintained within safe and optimal ranges.
6. **Educational Science Projects:** Introduce students to environmental science concepts using real-time temperature and humidity data collection.

HOBITRONICS

Project 3: Soil Moisture Sensor with Water Pump Control

Introduction:

In this project, you will build an **automated irrigation system** using an **Arduino**, **soil moisture sensor**, and **water pump**. The Arduino will monitor the soil's moisture level and activate the pump when the soil becomes too dry. This is an essential step towards building **smart agriculture systems** and improving water conservation.

Components Required:

- Arduino UNO
- Soil Moisture Sensor (**Analog type**)
- Breadboard
- Jumper Wires
- USB Cable for Arduino
- NPN Transistor (**to safely control the pump**)
- Snap Connector (to connect 9V battery)
- 9V DC Battery
- 9V DC pump
- 1k ohm resistor
- 16X2 LCD display
- Optocoupler

Wiring Instructions:

Since the Arduino output is limited to 5V and low current, but our pump requires 9V and higher current to operate, we can use a relay or an NPN transistor to switch the 9V supply to the pump under Arduino control

Soil Moisture Sensor:

- **VCC** → 5V (Arduino)
- **GND** → GND (Arduino)
- **A0 (Analog Output)** → A0 (Arduino Analog Pin)

NPN Transistor:

- **Pump+** → 9V (External power supply)
- **Pump -** → Collector of NPN Transistor (or the Relay for motor control)
- **Emitter of NPN Transistor** → GND
- **Base of NPN Transistor** → Pin 9 (Arduino) through a 1kΩ resistor (for control)

Optocoupler Input Side (LED side):

- **Anode (+)** → Arduino Pin 6 through a **220Ω(Current limiting) resistor**

- **Cathode (-) → GND**

This allows ~20mA current to safely drive the internal LED of the optocoupler.

Optocoupler Output Side (Transistor side):

- **Collector → 5V via a 10kΩ pull-up resistor**
(This keeps the collector high when the optocoupler is off.)
- Also connect the **collector** to the **base of your external NPN transistor** through a **1kΩ resistor**
- **Emitter → GND**

16x2 LCD Display (same setup as Project 1):

- **Pin 1 (VSS) → GND**
- **Pin 2 (VDD) → 5V**
- **Pin 3 (VO) → Middle pin of potentiometer**
- **Pin 4 (RS) → Pin 12 (Arduino)**
- **Pin 5 (RW) → GND**
- **Pin 6 (E) → Pin 11 (Arduino)**
- **Pins 7-10 → Not connected**
- **Pin 11 (D4) → Pin 5 (Arduino)**
- **Pin 12 (D5) → Pin 4 (Arduino)**
- **Pin 13 (D6) → Pin 3 (Arduino)**
- **Pin 14 (D7) → Pin 2 (Arduino)**
- **Pin 15 (A, LCD +) → 5V**
- **Pin 16 (K, LCD -) → GND**

What is the function of NPN transistor here?

- Arduino **cannot supply** enough current directly to run a motor/pump. So the transistor acts like a **switch**, controlled by Arduino.
- Arduino sends a **tiny current** into the transistor **base** (through 1kΩ resistor). This small current **allows a much bigger current** to flow from **collector to emitter** (pump current).
- The **pump needs 9V**, but Arduino can only output 5V signals. The NPN transistor allows Arduino (5V logic) to **control 9V power** safely.
- If you connect the pump directly to Arduino, the high current demand could **burn** the Arduino output pin. Using a transistor isolates the Arduino from the high-power pump side.
- Transistors can **turn ON and OFF very fast** (much faster than mechanical relays).

Why optocoupler?

An **optocoupler** is used here primarily for **electrical isolation** and **safety**, ensuring that the low-voltage components (like your Arduino) and the high-power components (like the pump and motor) don't interfere with each other. It isolates the microcontroller (Arduino) from the pump circuit, protecting the Arduino from any electrical noise, voltage spikes, or surges that might come from the pump or external power supply.

Optocoupler Input (LED side):

- The **220Ω resistor** limits current to the LED of the optocoupler.
- The Arduino drives the LED safely, turning the optocoupler **on** or **off**.

Optocoupler Output (Transistor side):

- The **10kΩ pull-up resistor** ensures the transistor stays **off** when the LED is off (collector pulled high).
- The **1kΩ resistor** limits the current to the base of the NPN transistor, protecting it.

No Need for Isolation of LCD or 9V Supply:

- The **LCD** and **9V supply** do not require isolation as they are part of the low-voltage system.
- The optocoupler only isolates the high-power components (pump and motor) from the Arduino.

Arduino Code:

```
#include <LiquidCrystal.h> // Include the LCD library

// Pin setup
const int motorPin = 9; // Pin connected to the base of the transistor
                        // controlling the motor
const int moisturePin = A0; // Pin connected to the soil moisture sensor

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Pin connections for the LCD

void setup() {
    lcd.begin(16, 2); // Initialize the LCD with 16 columns and 2 rows
    pinMode(motorPin, OUTPUT); // Set motor pin as output
    pinMode(moisturePin, INPUT); // Set moisture sensor pin as input
}
```



```

void loop() {
    int moistureLevel = analogRead(moisturePin); // Read the soil moisture level (0-1023)
    float moisturePercentage = map(moistureLevel, 1023, 0, 0, 100);
    // Map it to percentage

    lcd.setCursor(0, 0);
    lcd.print("Moisture: ");
    lcd.print(moisturePercentage);
    lcd.print("%");

    if (moisturePercentage < 30) {           // If soil moisture is less than 30%
        digitalWrite(motorPin, HIGH);      // Turn motor ON (pump activated)
        lcd.setCursor(0, 1);
        lcd.print("Pump: ON ");
    }
    else {
        digitalWrite(motorPin, LOW);       // Turn motor OFF (pump deactivated)
        lcd.setCursor(0, 1);
        lcd.print("Pump: OFF");
    }

    delay(1000);                          // Wait for 1 second before updating
}

```

Code Explanation:

1. Reading Sensor Data:

Continuously read the analog value from the soil moisture sensor (`analogRead(moisturePin)`) to determine how dry or wet the soil is.

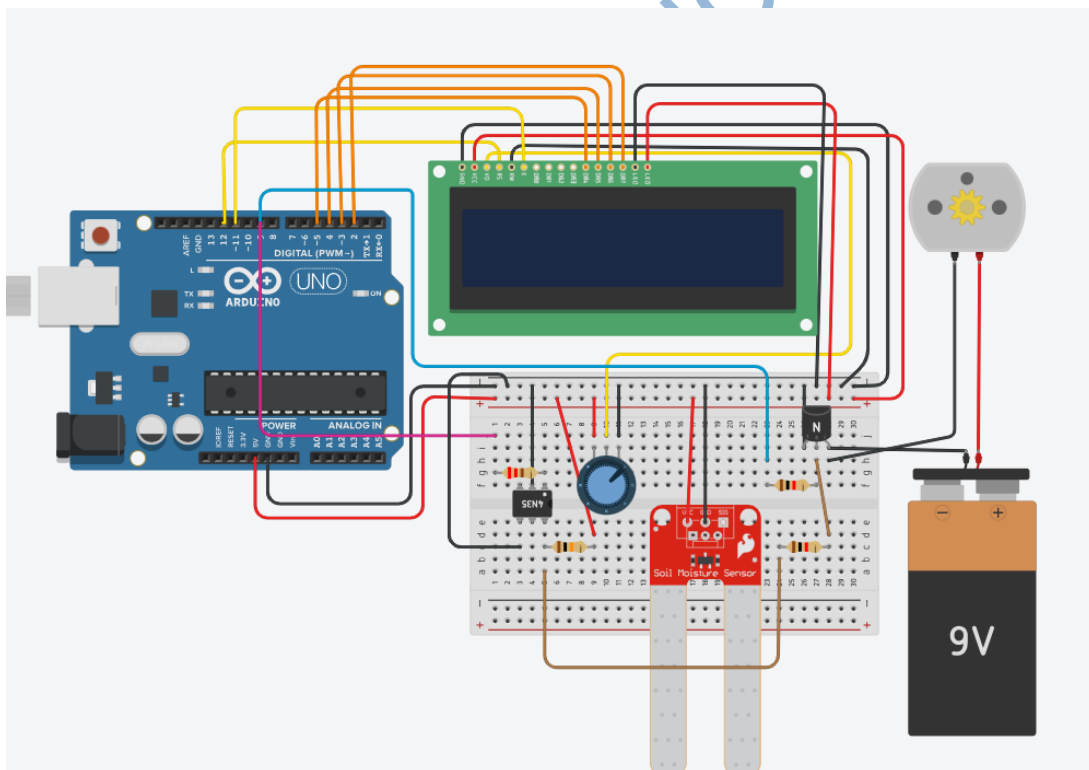
- A **higher value (near 1023)** means **dry soil**
- A **lower value (near 0)** means **wet soil**

2. Percentage Conversion (Inverted Mapping):

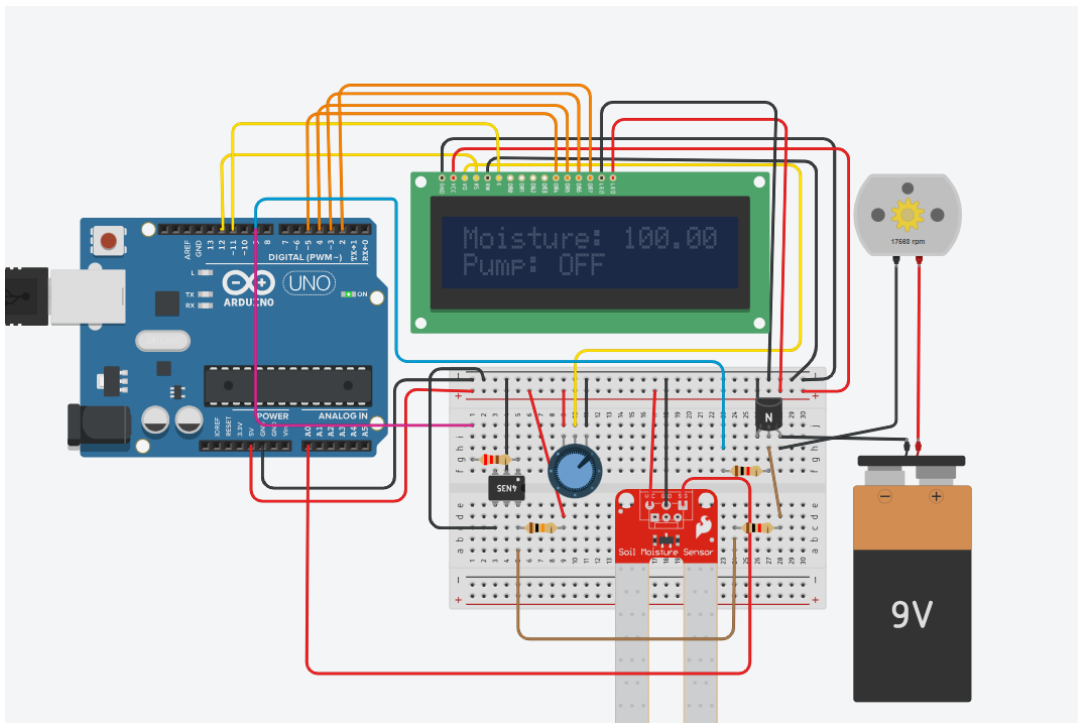
Use the `map()` function to convert the raw analog value into a moisture percentage.

- The mapping is inverted (`map(value, 1023, 0, 0, 100)`) so **wet soil shows higher %** and dry soil shows lower %
3. **Threshold-Based Pump Control:**
- Check if the moisture percentage is below a threshold (e.g., 30%).
- If **moisture < 30%** → soil is dry → **turn ON the pump**
 - If **moisture ≥ 30%** → soil is moist enough → **turn OFF the pump**
4. **Water Pump Automation:**
- The pump is automatically controlled using a transistor or optocoupler circuit.
- No need for manual checking or switching — the system handles everything based on real-time soil moisture!
5. **LCD Display Output:**
- Real-time values are displayed on a 16x2 LCD:
- **Top line:** Shows current soil moisture percentage
 - **Bottom line:** Indicates whether the pump is **ON** or **OFF**

Circuit Diagram:

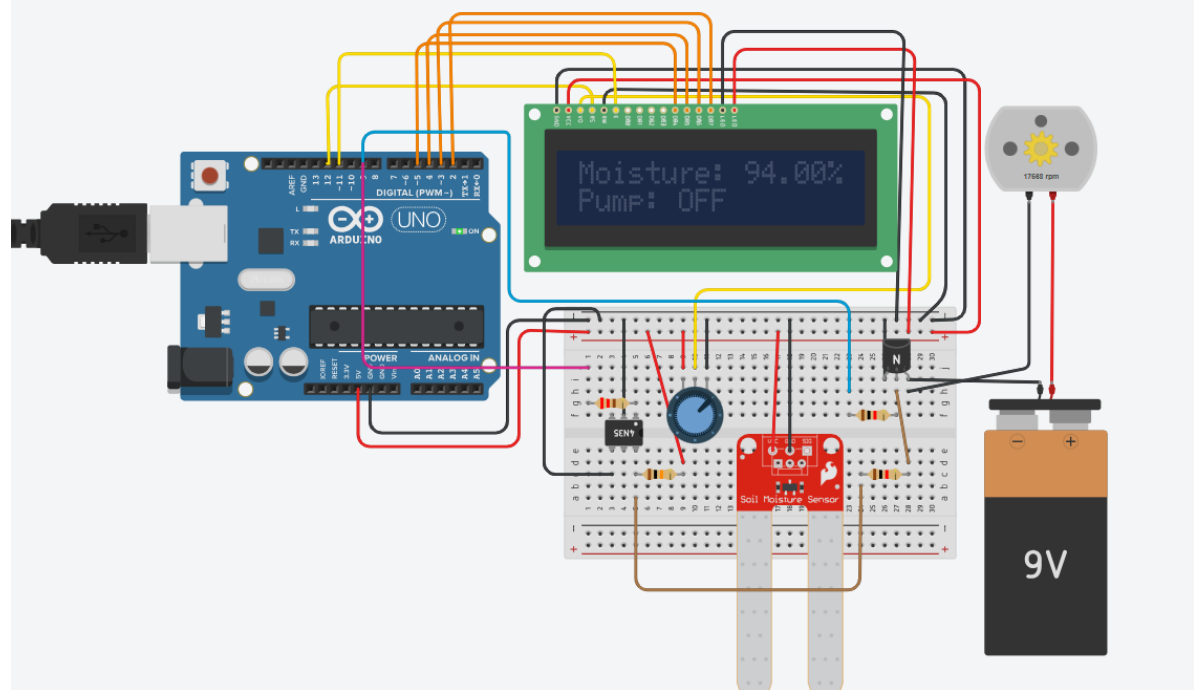
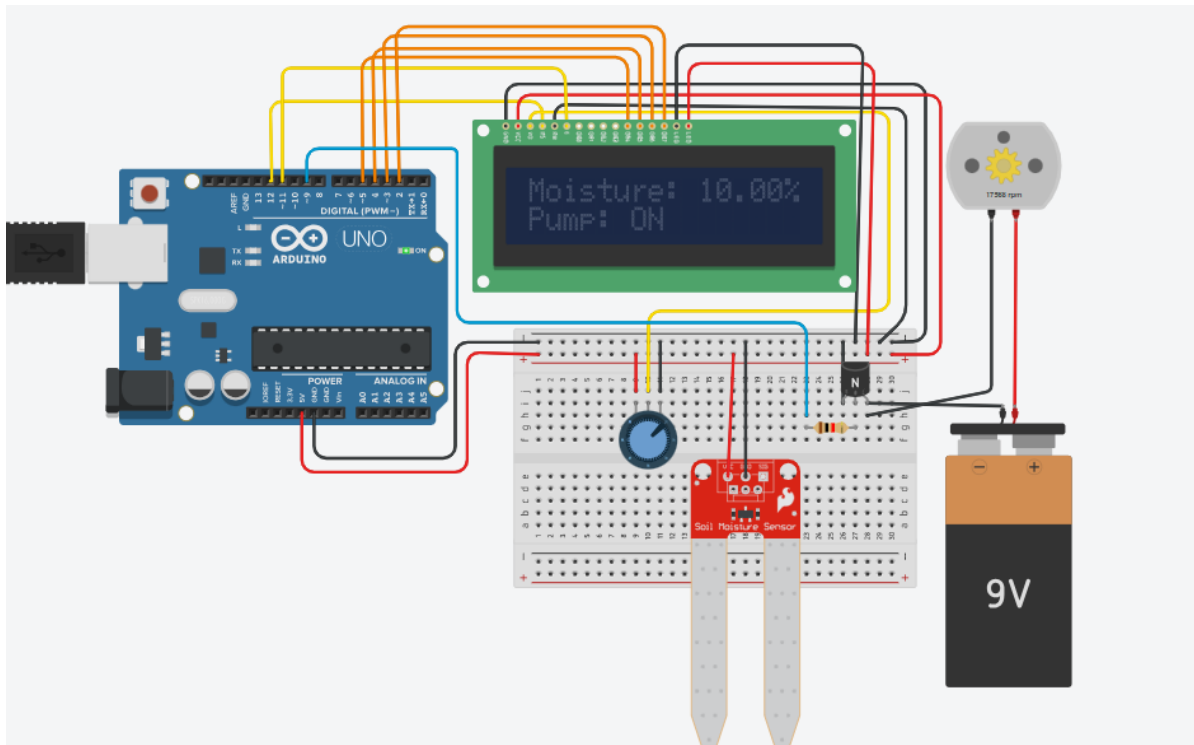


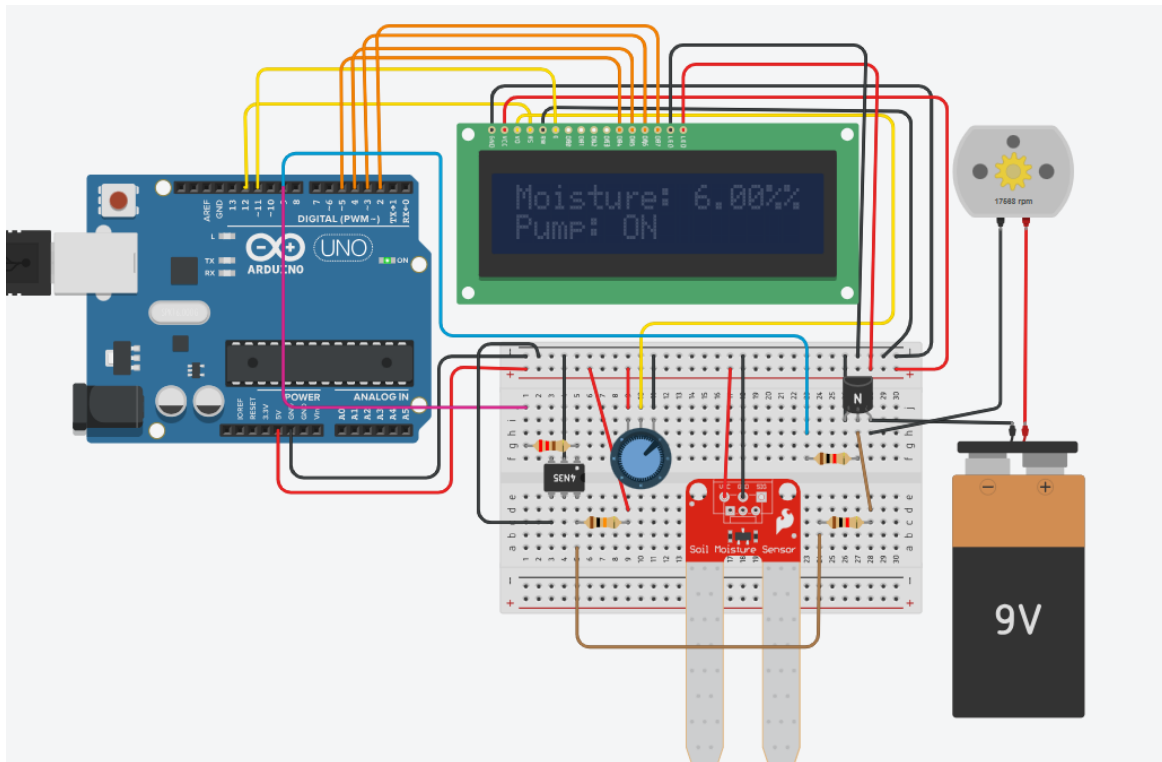
Simulation:



We will remove sensor (by disconnecting Arduino pin (A0) and Signal pin of Soil moisture sensor).

Then we can see Arduino generating random values of moisture causing the pump to ON and OFF.





Output:

- **Dry soil:** The pump automatically turns on to water the plant.
- **Moist soil:** The pump automatically turns off, conserving both water and energy. This setup functions as a **mini prototype** of a **smart irrigation system** for greenhouses.

3. **Urban Farming Projects:** Ideal for applications like **hydroponics**, **terrace gardens**, and **vertical farming** where water usage needs to be monitored and controlled.
4. **Remote Agricultural Fields:** With additional components like **GSM** or **IoT**, farmers can monitor and control irrigation remotely, optimizing water usage in vast fields.
5. **Home Plant Care Systems:** Forget to water your plants? This system takes care of watering them for you, ensuring they receive proper care even when you're not around.
6. **Agricultural Research:** This project serves as a foundational model for developing advanced **smart agriculture** and **sustainable farming techniques**.

HOBITRONICS

Project 4: Gas Sensor with LCD Display

Introduction:

In this project, you will build a **Gas Concentration Monitoring System** using an **MQ Gas Sensor** and a **16x2 LCD Display** (16 pins, non-I2C type).

This system will continuously measure and display the concentration levels of various gases like **Carbon Monoxide (CO)** and **Methane (CH₄)** in real-time. This is an essential system for **industrial safety**, **mining operations**, **smart homes**, and **environmental monitoring**.

Components Required:

- Arduino UNO
- MQ Gas Sensor (e.g., MQ-2, MQ-135)
- 16x2 LCD Display (16 pins, non-I2C type)
- 10kΩ Potentiometer (for LCD contrast adjustment)
- Breadboard and Jumper Wires
- USB Cable for Arduino
- Optional: External Power Supply (for more stable operation)

Note:

- **MQ-2** can detect **LPG** (liquefied petroleum gas), **Methane**, **Butane**, **Smoke**, **Hydrogen**, **Propane**.
- **MQ-135** can detect **Air Quality** overall (pollution), **Ammonia (NH₃)**, **Sulfur (SO₂)**, **Benzene**, **Carbon dioxide (CO₂)** (minor sensitivity), **Alcohol vapors** (small).

Wiring Instructions:

MQ Gas Sensor:

- **VCC** → 5V (Arduino)
- **GND** → GND (Arduino)
- **A0** → A0 (Arduino Analog Pin)

Potentiometer (for LCD contrast):

- **Left Pin** → 5V (Arduino)
- **Right Pin** → GND (Arduino)
- **Middle Pin** (wiper) → Pin 3 (V0) of the LCD

16x2 LCD Display (16 Pins):

- **Pin 1 (VSS)** → GND
- **Pin 2 (VDD)** → 5V
- **Pin 3 (V0)** → Potentiometer wiper (middle pin)
- **Pin 4 (RS)** → Pin 12 (Arduino)
- **Pin 5 (RW)** → GND
- **Pin 6 (E)** → Pin 11 (Arduino)
- **Pins 7–10 (D0–D3)** → Not connected
- **Pin 11 (D4)** → Pin 5 (Arduino)
- **Pin 12 (D5)** → Pin 4 (Arduino)
- **Pin 13 (D6)** → Pin 3 (Arduino)
- **Pin 14 (D7)** → Pin 2 (Arduino)
- **Pin 15 (A)** → 5V (for LCD backlight)
- **Pin 16 (K)** → GND (for LCD backlight)

Arduino Code:

```
#include <LiquidCrystal.h>           // Include the LCD library

// Pin setup for the LCD: RS, E, D4, D5, D6, D7
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// MQ Gas Sensor Pin
int gasSensorPin = A0;               // The sensor is connected to analog pin A0

// Variables for gas concentration
int sensorValue = 0;
float gasConcentration = 0.0;

void setup() {
    lcd.begin(16, 2);                 // Initialize the 16x2 LCD
    lcd.setCursor(0, 0);              // Set cursor to the first row
    lcd.print("Gas Monitoring");       // Display message on Row 1
    lcd.setCursor(0, 1);              // Set cursor to the second row
    lcd.print("Hobitronics System");  // Display message on Row 2
    // Wait for a moment to display the welcome message
    delay(2000);                      // Wait for 2 seconds
}
```



```

void loop() {
    // Read the analog value from the gas sensor
    sensorValue = analogRead(gasSensorPin);

    // Map the sensor value to a percentage of gas concentration (0-100%)
    gasConcentration = (float)map(sensorValue, 0, 1023, 0, 100);

    // Clear the LCD display for new data
    lcd.clear();

    // Display the gas concentration on the LCD
    lcd.setCursor(0, 0);           // Set cursor to the first row
    lcd.print("CO Gas: ");
    lcd.print(gasConcentration);
    lcd.print("%");
    // Wait for 1 second before updating the display
    delay(1000);                   // Update every 1 second
}

```

Remember:

Burn-in time is a **pre-conditioning** period during which a gas sensor is operated continuously (often at a low concentration of the target gas) before it can provide accurate and stable readings. This period is crucial for ensuring that the sensor reaches its **optimal performance** and gives reliable results over time.

The **24-hour burn-in period** for gas sensors is necessary to stabilize the sensor's sensitive elements, improve sensitivity, clear contaminants, and allow for proper temperature-dependent calibration.

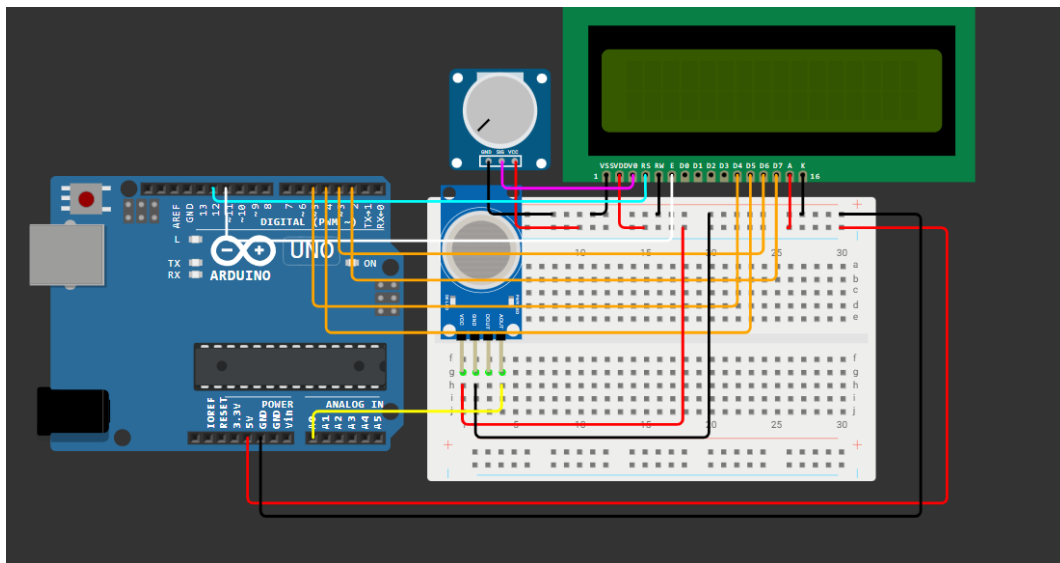
It prevents **false readings** during the initial operation phase and ensures that the sensor can deliver **accurate, reliable data** over its lifetime.

This burn-in period is a standard practice for getting **optimal performance** from most gas sensors, especially those used in environments where precise gas detection is critical.

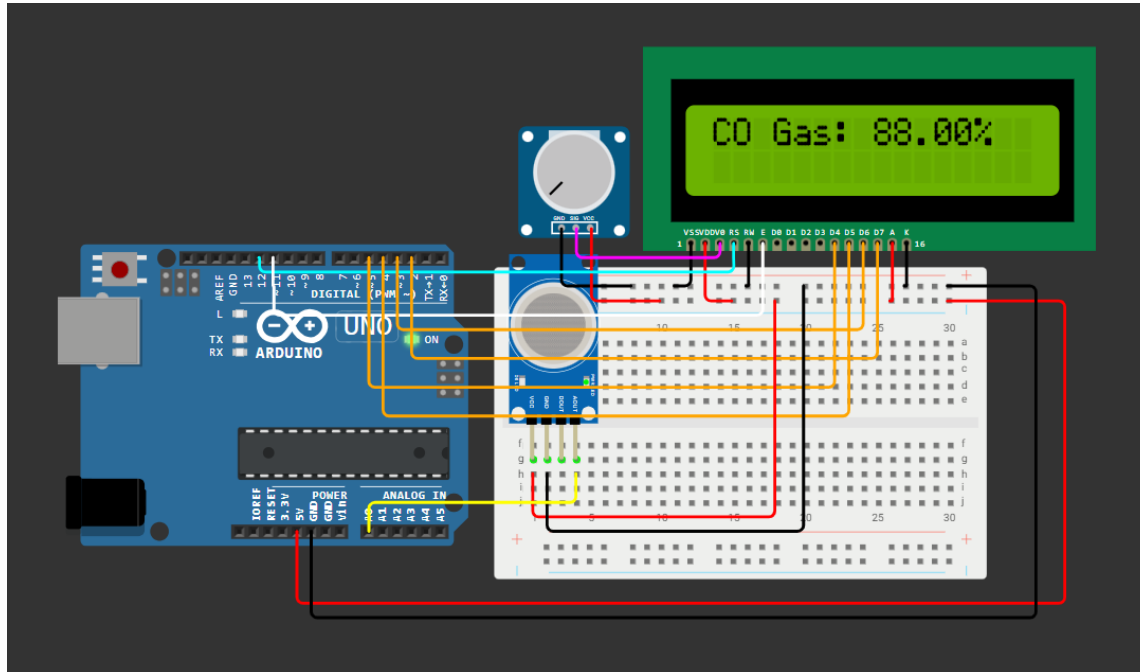
Code Explanation:

- **Analog Sensor Reading:** The code continuously reads the analog value from the MQ gas sensor connected to **A0**.
- **Mapping Sensor Value:** The raw sensor value is mapped to a percentage concentration (e.g., 0–100%) for easier interpretation.
- **LCD Display:** The mapped gas concentration is displayed on the 16x2 LCD in real-time.
- **Continuous Monitoring:** The display updates every second, ensuring live monitoring of the gas concentration.

Circuit Diagram:



Simulation:



Output:

- The **LCD display** will show the live concentration of the detected gas (e.g., "CO Gas: 88%").
- Adjust the **sensitivity** of the gas sensor using the potentiometer for optimal performance.
- The system provides instant **visual feedback** of gas concentration, helping to identify hazardous gas levels early.

What You Learn:

- **Gas Detection Basics:** Learn how to detect gases like Carbon Monoxide, Methane, and Air Quality using MQ sensors.
- **Sensor Data Processing:** Understand how to read and map analog sensor values for real-world interpretation.
- **Analog-to-Digital Conversion:** Learn how Arduino converts sensor voltages into readable numeric values.
- **Environmental Safety Systems:** Develop basic skills needed for designing air quality and gas safety monitors.

Real-World Applications:

1. **Industrial Gas Leak Detection:** Early detection of dangerous gases like **Carbon Monoxide (CO)**, **Methane (CH₄)**, and **LPG leaks** in factories, chemical plants, and manufacturing units.
2. **Environmental Air Quality Monitoring:** Track **pollution levels** in urban areas or residential neighborhoods. This data can be used by governments and researchers for better air quality management.
3. **Mining Industry Safety:** Continuous monitoring of **underground mining shafts** for harmful gas buildups, preventing explosions and ensuring worker safety.
4. **Home Safety Systems:** **Smart gas leak detection** for homes, offering early warnings for dangerous gas leaks from appliances like stoves, heaters, and faulty gas lines.
5. **Smart Greenhouses:** **Monitor CO₂ and air quality** in greenhouses to optimize crop yield and improve plant health in controlled agricultural environments.
6. **IoT-based Remote Monitoring:** **Integrate with IoT systems** to send real-time data to cloud dashboards, mobile alerts, and smart control systems, providing remote gas monitoring capabilities.

Project 5: RFID Delivery Tracking System

Introduction:

In this project, you will build an **RFID-based tracking system** — similar to how major logistics companies like **Amazon**, **Flipkart**, and **FedEx** track their parcels.

By using **RFID tags** to uniquely identify packages and an **Arduino** to read and verify these tags, this system demonstrates how RFID can **streamline delivery processes**, reduce errors, and improve time efficiency in **supply chain management**.

Components Required:

- Arduino UNO
- RFID Reader (**RC522 Module**)
- RFID Tags/Cards
- Breadboard
- Jumper Wires
- Optional: 16x2 LCD Display (**for displaying package information**)
- Optional: Buzzer (**for "scan success" beep**)
- Optional: LED Indicators (**Green = Success, Red = Error**)

Wiring Instructions:

RC522 RFID Reader:

- **SDA** → **Pin 10** (Arduino)
- **SCK** → **Pin 13**
- **MOSI** → **Pin 11**
- **MISO** → **Pin 12**
- **IRQ** → Not connected
- **GND** → **GND**
- **RST** → **Pin 9**
- **3.3V** → **3.3V** (Important: **Do NOT** connect to 5V)

Optional 16x2 LCD Display (16 Pins):

- **Pin 1 (VSS)** → **GND**
- **Pin 2 (VDD)** → **5V**
- **Pin 3 (V0)** → Potentiometer wiper (middle pin)
- **Pin 4 (RS)** → **Pin 12** (Arduino)
- **Pin 5 (RW)** → **GND**
- **Pin 6 (E)** → **Pin 11** (Arduino)
- **Pins 7–10 (D0–D3)** → Not connected
- **Pin 11 (D4)** → **Pin 5** (Arduino)
- **Pin 12 (D5)** → **Pin 4** (Arduino)

- **Pin 13 (D6)** → **Pin 3** (Arduino)
- **Pin 14 (D7)** → **Pin 2** (Arduino)
- **Pin 15 (A)** → **5V** (for LCD backlight)
- **Pin 16 (K)** → **GND** (for LCD backlight)

Use the standard **16x2 LCD 4-bit wiring** to display "**Package ID: xxxxx**" after each scan.

Optional Buzzer/LEDs:

- **Buzzer or LEDs** can be connected to any available digital pins on the Arduino to provide **scan feedback** (Green LED or Buzzer for success, Red LED for error).

What is RFID?

RFID (Radio Frequency Identification) is a technology that uses radio waves to wirelessly transfer data between a tag (RFID tag) and a reader (RFID reader). Here's a concise overview:

1. How It Works:

- RFID tags consist of an integrated circuit (IC) and an antenna, enabling them to receive and transmit data via radio waves. When the RFID reader sends a signal, the tag responds with its unique identifier or other stored information.

2. Types of RFID:

- RFID operates across various frequency ranges: Low Frequency (LF, 125-134 kHz), High Frequency (HF, 13.56 MHz), and Ultra High Frequency (UHF, 860-960 MHz). **High Frequency (HF)** RFID is the most common for applications like contactless payment and access control.

3. Low-Frequency vs. High-Frequency:

- **Low Frequency (LF)** operates at 125-134 kHz with a **shorter range** (up to 10 cm) and slower data transfer rates, making it suitable for applications requiring short-range, lower-speed communications.
- **Keep in Mind: Lower frequencies means higher wavelength!** The range mentioned here is not because of wavelength but the technologies limitations, such as its **lower data transfer speed** as well as the **Limited energy** present in the signal.
- **High Frequency (HF)** operates at 13.56 MHz and provides better read ranges (up to 1 meter) and faster data transfer rates, commonly used in applications such as libraries, retail, and transportation.

4. Applications:

- RFID is used extensively for inventory management, supply chain tracking, asset management, access control (e.g., keycards, contactless payments), and animal identification.

5. **Passive vs. Active Tags:**

- **Passive tags** are powered by the reader's signal and do not have their own power source. These are inexpensive and widely used.
- **Active tags** have an internal power source, allowing them to communicate over longer distances, making them suitable for applications that require greater range.

6. **Benefits:**

- RFID provides **contactless operation**, **real-time tracking**, **enhanced security**, and **improved efficiency**, making it invaluable in industries like logistics, healthcare, retail, and security.

In summary, RFID is a high-frequency wireless technology used for seamless identification and data transmission, significantly improving efficiency across a range of industries, from access control to inventory management.

Arduino Code:

```
#include <SPI.h> // Include SPI library (needed for RC522)
#include <MFRC522.h> // Include MFRC522 library for RFID
#include <LiquidCrystal.h> // Include LCD library

#define SIMULATION_MODE // Uncomment this line to simulate, comment it
for hardware usagerea

// LCD pin connections: (rs, en, d4, d5, d6, d7)
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// RFID Pins
#define RST_PIN 9 // Reset pin for RFID
#define SS_PIN 10 // Slave select pin for RFID
MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance

unsigned long previousMillis = 0;
int state = 0;
```

```

void setup() {
    Serial.begin(9600);
    lcd.begin(16, 2);           // Initialize 16x2 LCD
    lcd.print("System Ready");
    delay(2000);               // Wait for 2 seconds

    // Initialize the RFID reader if in real mode
    #ifdef SIMULATION_MODE
    // Simulating mode, no need to initialize the RFID reader
    #else
    SPI.begin();               // Start the SPI bus
    mfrc522.PCD_Init();        // Initialize the MFRC522 RFID reader
    Serial.println("RFID Reader Initialized.");
    #endif
}

void loop() {
    unsigned long currentMillis = millis();

    // Simulating package detection
    #ifdef SIMULATION_MODE
    if (state == 0 && currentMillis - previousMillis >= 1000) {
        Serial.println("No Package Detected...");
        lcd.clear();
        lcd.setCursor(0, 0);    // Sets cursor at row 1 starting
        lcd.print("No Package");
        previousMillis = currentMillis;
        state = 1;
    }
    #endif
}

```



```

else if (state == 1 && currentMillis - previousMillis >= 4000) {
    Serial.println("Package Detected: ID 123ABC");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Package Found!");
    lcd.setCursor(0, 1);
    lcd.print("ID: 123ABC");
    previousMillis = currentMillis;
    state = 2;
}

else if (state == 2 && currentMillis - previousMillis >= 2000) {
    Serial.println("Package Left...");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Package Left...");
    previousMillis = currentMillis;
    state = 0;
}

#else
// Real RFID reading mode begins
if (state == 0) {
    if (mfr522.PICC_IsNewCardPresent()) {
        // If a new card is detected
        if (mfr522.PICC_ReadCardSerial()) {
            // If the card is read successfully
            Serial.println("Package Detected: ID ");
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("Package Found!");
            lcd.setCursor(0, 1);

```

```

        // Print RFID ID
        String rfidID = "";
        for (byte i = 0; i < mfrc522.uid.size; i++) {
            rfidID += String(mfrc522.uid.uidByte[i], HEX);
            if (i < mfrc522.uid.size - 1) rfidID += ":";
        }
        // Convert to uppercase
        rfidID.toUpperCase();

        lcd.print("ID: ");
        lcd.print(rfidID);

        // Print the RFID ID to Serial Monitor
        Serial.print("Package Detected: ID ");
        Serial.println(rfidID);

        mfrc522.PICC_HaltA(); // Halt the RFID card
        mfrc522.PCD_StopCrypto1(); // Stop encryption for the
card to avoid rereading
        previousMillis = currentMillis;
        state = 1;
    }
}

}

else if (state == 1 && currentMillis - previousMillis >= 5000) {
    Serial.println("Package Left...");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Package Left...");
    previousMillis = currentMillis;
    state = 0;
}

#endif
}

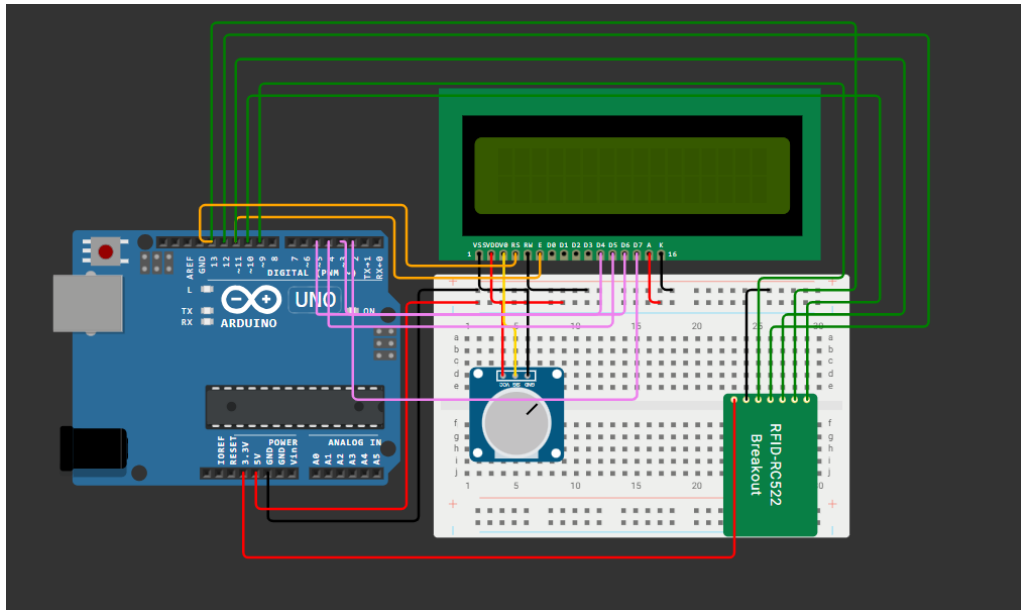
```

Code Explanation:

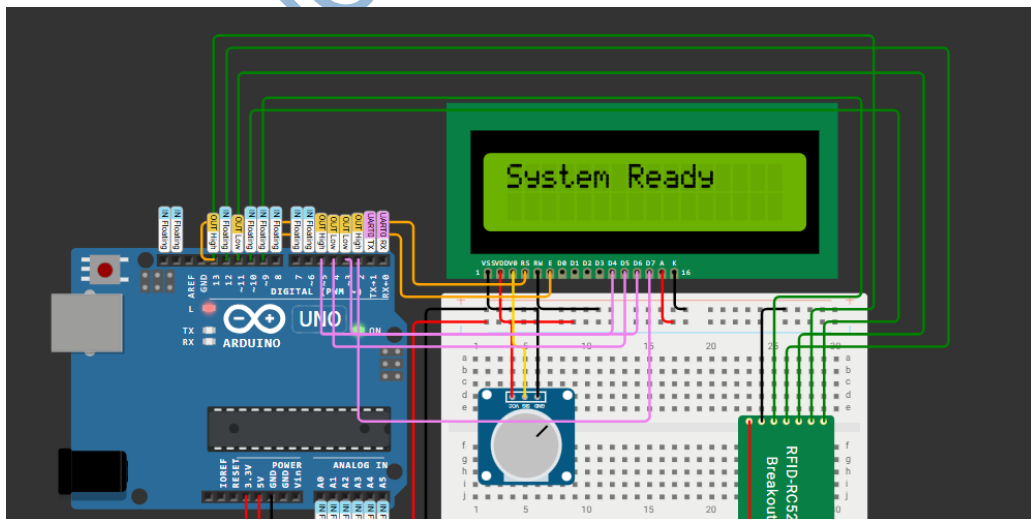
- The RFID module is initialized by setting up SPI communication (SPI.begin()) and configuring the MFRC522 RFID reader (mfrc522.PCD_Init()), if in real RFID mode. If in **simulation mode**, the RFID module is **skipped**, and the program uses predefined logic to simulate package detection and handling.
- In **simulation mode**, no physical RFID tags are scanned. Instead, the program simulates package detection and removal based on time intervals.
- The system uses millis() to track elapsed time and trigger state changes. After waiting for specific time periods (1 second for "No Package," 4 seconds for "Package Found," and 2 seconds for "Package Left"), the system updates the LCD and Serial Monitor with corresponding messages (e.g., "No Package," "Package Found! ID: 123ABC," and "Package Left...").
- This helps simulate how the system would behave in a real scenario with actual RFID scans.
- In **real RFID mode**, the system waits for a real RFID tag to be scanned by the RFID reader.
- **Read the UID (Unique ID) of the RFID tag:** When an RFID card is detected, the system reads the UID of the tag using mfrc522.PICC_ReadCardSerial().
- **Compare the UID with a pre-programmed database (stored in code):** The UID of the scanned RFID tag is compared to a pre-registered UID (e.g., 123ABC).
- **If the UID matches a registered package:**
 1. A success message is displayed on the LCD (Package Found! ID: [UID]).
 2. Optional hardware actions (e.g., buzzer sound or green LED blink) can be triggered to indicate success.
- **If the UID does not match:**
 1. An error message is displayed (Package Not Found).
 2. Optional hardware actions (e.g., red LED blink or error beep) can be triggered to indicate failure.

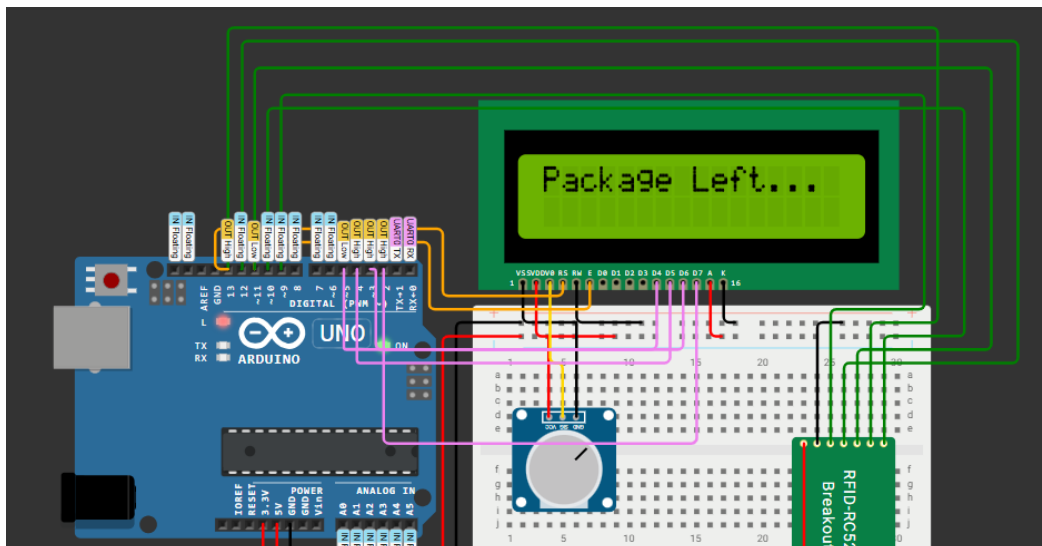
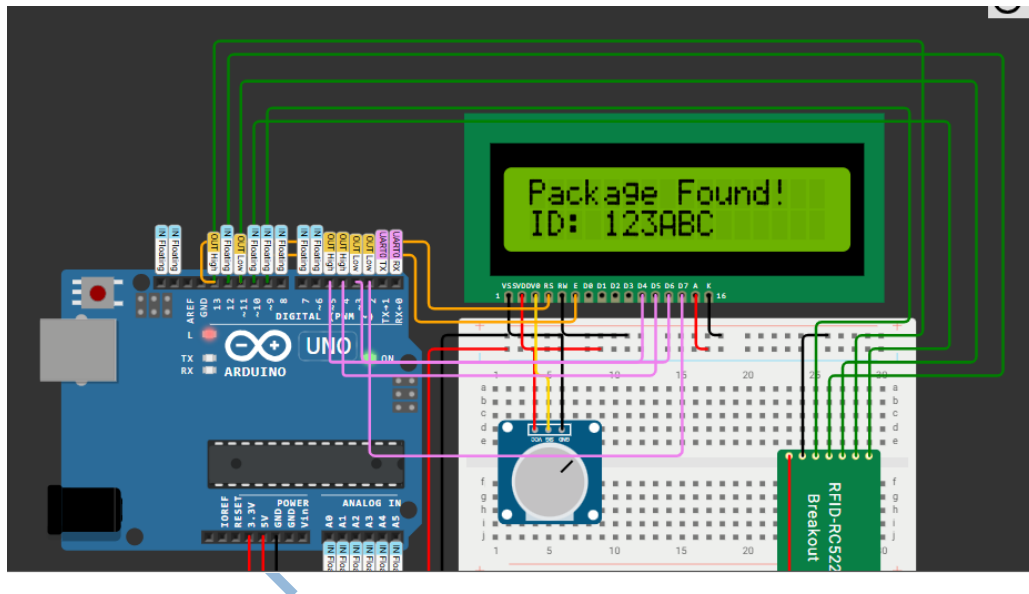
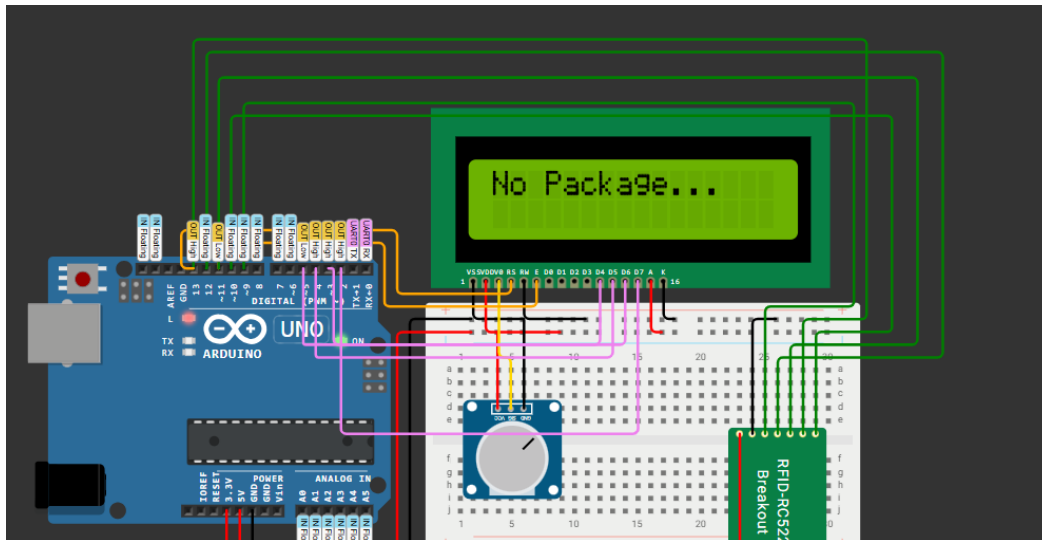
- The UID of the detected RFID tag is logged to the Serial Monitor for tracking and monitoring purposes. This helps in debugging and verifying which tags are being scanned.

Circuit Diagram:



Simulation:





This project has been fully simulated using WowKi and/or Tinkercad platforms. Due to the absence of certain physical components (like RFID tags), simulated inputs were used where needed. The wiring, coding, and logic are verified and correct. In a real-world setup, once actual hardware is connected, the system will function exactly as intended.

Output:

- **Scan an RFID tag** attached to a package.
- The **Arduino** reads and verifies the package's identity.
- The **LCD Display** (or **Serial Monitor**) shows the success or failure of the scan.
- The result simulates a **real-world RFID-based delivery tracking system**.

What You Learn:

- **Understanding RFID Technology:** Learn how RFID readers and tags work for real-world tracking applications like logistics and supply chain management.
- **Arduino-RFID Interfacing:** Master connecting and communicating with the RC522 RFID reader via Arduino using SPI protocol.
- **Reading and Verifying UIDs:** Learn to detect RFID tags, read their unique IDs, and verify them against a stored list.

Real-World Applications:

1. **Warehouse Package Management:** Streamline parcel tracking with RFID in large warehouses for faster, error-free management.
2. **Supply Chain Optimization:** Automate inventory updates as items move in and out of the warehouse, improving the overall efficiency of the supply chain.
3. **Logistics and Courier Companies:** **RFID technology** is already being used by leading companies like **Amazon**, **Flipkart**, and **FedEx** to provide real-time tracking and improve the sorting process.
4. **Library Book Management:** The same concept can be applied to **library book management**, where each book has an RFID tag for automatic check-in and check-out processes.
5. **Employee Attendance Systems:** Modify this system to log employee attendance in offices and factories using RFID badges.
6. **Vehicle and Asset Tracking:** Use RFID for **tracking vehicles, containers**, or even **hospital equipment**, providing real-time monitoring for better logistics management.

THANK YOU!



We hope you enjoyed building these exciting Arduino projects.
You've just taken a big step toward mastering real-world electronics.

Explore our Beginner friendly series...


[Top 5 Arduino Projects for Beginners](#)

 Share Your Projects

 Tag us on Instagram [@hobitronics](#)

And show the world your creations!

We'll repost the best projects on our page

 Follow us on Youtube [@Hobitronics](#)

 Also follow our Whatsapp Channel [@Hobitronics](#)

 Resources & Community

 Visit our blog: [hobitronics.blog](#)

For feedback, questions, or collaboration: hobitronicsin@gmail.com

"We're engineers, dreamers, and learners just like you. Hobitronics was born from our passion to make electronics accessible, inspiring, and fun."

— Team Hobitronics 