SL no	List of Experiment	
1.	7 Segment display (0-9) & (A-F)	
2.	Right scrolling LCD display	
3.	Analog Voltage Reader on LCD display	
4.	Temperature & Humidity Sensor	
5.	Gas Sensor with Display	
6.	PIR sensor	
7.	Ultrasonic sensor	

Experiment-1a

7 Segment Display (0-9) numbers

A 7-segment display consists of 7 LEDs arranged in the shape of the number "8." Each segment is labelled as a, b, c, d, e, f, and g. By turning specific segments ON or OFF, you can form the digits 0 to 9. . Here's a step-by-step explanation of its working:

Components of an7 Segment Display (0-9) numbers

1. Arduino board
2. 7-segment display (common cathode)
3. 7 current-limiting resistors (220Ω)
4. Connecting wires
5. Breadboard

Working Principle:

1. Structure of the 7-Segment Display

A 7-segment display consists of:

- 7 LEDs: Arranged in the shape of the number "8," each LED is called a segment and is labelled a, b, c, d, e, f, and g.
- Common Connection:
  - Common Cathode: The negative terminals (cathodes) of all LEDs are connected and grounded (GND).
  - Common Anode: The positive terminals (anodes) of all LEDs are connected together and supplied with VCC.

Each segment lights up when current flows through it.

2. Electrical Operation

- Current Flow:
  - In a common cathode display, connecting a segment pin to HIGH allows current to flow from VCC through the segment to GND, lighting it up.
  - In a common anode display, connecting a segment pin to LOW allows current to flow from VCC through the anode and the segment to the pin, lighting it up.

- Controlling Segments: By applying HIGH/LOW signals to specific pins, the required segments are turned ON or OFF, forming a desired pattern (digit).

3. Logic Behind Displaying Numbers

Each digit (0 to 9) corresponds to a specific combination of segments that need to be turned ON:

- Example: To display **3**, segments a, b, c, d, g need to be ON, while e and f remain OFF.

The working principle of a 7-segment display involves selectively lighting up individual LEDs (segments) to form numbers or characters. Here's a detailed explanation

4. Arduino Control

The Arduino controls the 7-segment display by:

1. Setting the pins connected to the display as output.
2. Using the binary pattern for each digit to decide which pins should be set to HIGH or LOW.
3. Sending signals to the pins to turn the appropriate segments ON or OFF.

6. Power Management

- Resistors: Current-limiting resistors are used to protect the LEDs from excess current.
- Power Supply: The Arduino provides the required power to the display, typically 5V.

Applications:

1. Clocks and Timers
2. Counters
3. Measuring Instruments
4. Electronic Gadgets
5. Home Appliances
6. Automobiles
7. Consumer Electronics
8. Medical Equipment
9. Information Displays Boards.

Key Points to Remember:

• Types and Configuration:

- Common Cathode (CC): Segments light up with HIGH signals.
- Common Anode (CA): Segments light up with LOW signals.

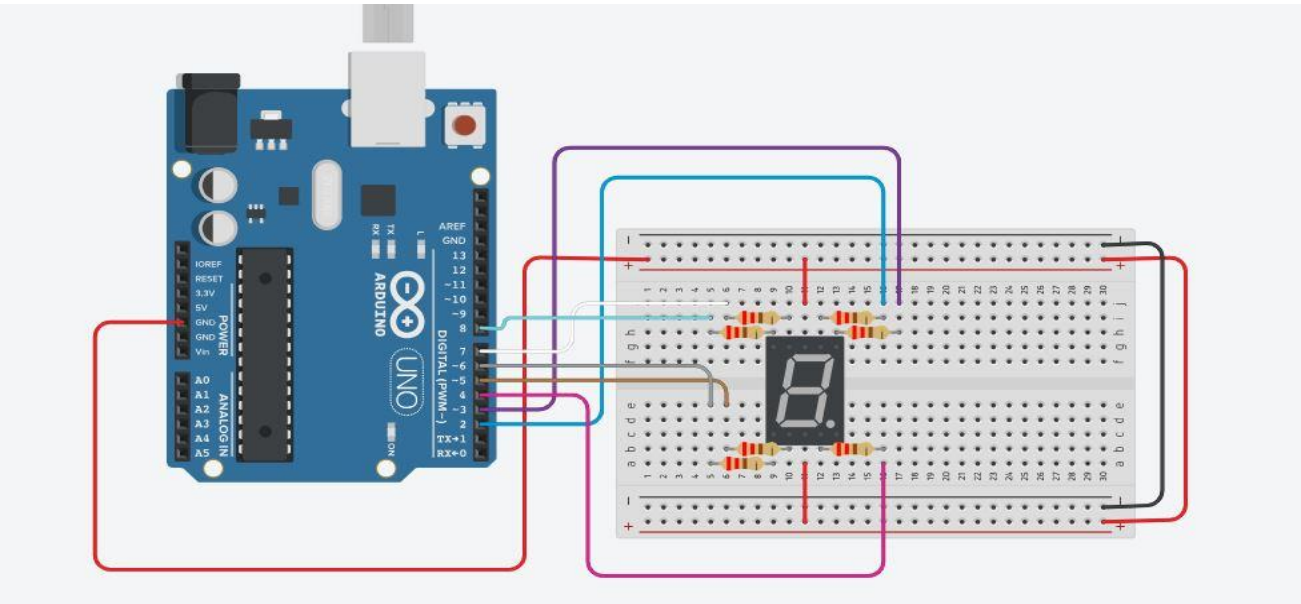
• Power and Protection:

- Operates at 5V; use current-limiting resistors (220Ω–330Ω) to prevent damage.

• Control and Logic:

- Each digit has a unique binary pattern to light specific segments; use Arduino or driver ICs for control.

Connection:



Code:

```
// Define the segment pins for the 7-segment display
int a = 2;  // Segment a
int b = 3;  // Segment b
int c = 4;  // Segment c
int d = 5;  // Segment d
int e = 6;  // Segment e
int f = 7;  // Segment f
int g = 8;  // Segment g

// Function to display a number (0-9) on a 7-segment display
void displayDigit(int num)
{
  switch (num)
  {
    case 0: // Display 0
      digitalWrite(a, HIGH);
      digitalWrite(b, HIGH);
      digitalWrite(c, HIGH);
      digitalWrite(d, HIGH);
      digitalWrite(e, HIGH);
      digitalWrite(f, HIGH);
      digitalWrite(g, LOW);
```

```
  break;
 case 1: // Display 1
  digitalWrite(a, LOW);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, LOW);
  break;
 case 2: // Display 2
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, LOW);
  digitalWrite(d, HIGH);
  digitalWrite(e, HIGH);
  digitalWrite(f, LOW);
  digitalWrite(g, HIGH);
  break;
 case 3: // Display 3
  digitalWrite(a, HIGH);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, LOW);
  digitalWrite(f, LOW);
  digitalWrite(g, HIGH);
  break;
 case 4: // Display 4
  digitalWrite(a, LOW);
  digitalWrite(b, HIGH);
  digitalWrite(c, HIGH);
  digitalWrite(d, LOW);
  digitalWrite(e, LOW);
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
  break;
 case 5: // Display 5
  digitalWrite(a, HIGH);
  digitalWrite(b, LOW);
  digitalWrite(c, HIGH);
  digitalWrite(d, HIGH);
  digitalWrite(e, LOW);
  digitalWrite(f, HIGH);
  digitalWrite(g, HIGH);
  break;
 case 6: // Display 6
  digitalWrite(a, HIGH);
```

```
    digitalWrite(b, LOW);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
   case 7: // Display 7
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, LOW);
    digitalWrite(e, LOW);
    digitalWrite(f, LOW);
    digitalWrite(g, LOW);
    break;
   case 8: // Display 8
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
   case 9: // Display 9
    digitalWrite(a, HIGH);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, LOW);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
  }
}

void setup()
{
  // Set the segment pins as output
  pinMode(a, OUTPUT);
  pinMode(b, OUTPUT);
  pinMode(c, OUTPUT);
  pinMode(d, OUTPUT);
  pinMode(e, OUTPUT);
  pinMode(f, OUTPUT);
  pinMode(g, OUTPUT);
}
```

```
void loop() {
 // Loop through digits 0 to 9 and display each for 1 second
 for (int i = 0; i <= 9; i++)
 {
  displayDigit(i);
  delay(1000); // Wait 1 second before displaying the next number
 }
}
```

Experiment-1b

7 Segment Display (A-F) Alphabets:

To display alphabets **a-f** on a 7-segment display, you need to light up specific segments for each letter. Since the display is limited to 7 segments, only some alphabets can be represented, and their appearance may be stylized.

Components of an7 Segment Display (0-9) numbers

1. Arduino board
2. 7-segment display (common cathode)
3. 7 current-limiting resistors (220Ω)
4. Connecting wires
5. Breadboard

Working Principle:

1. Structure of the 7-Segment Display

A 7-segment display consists of:

- 7 LEDs: Arranged in the shape of the number "8," each LED is called a segment and is labelled a, b, c, d, e, f, and g.
- Common Connection:
  - Common Cathode: The negative terminals (cathodes) of all LEDs are connected and grounded (GND).
  - Common Anode: The positive terminals (anodes) of all LEDs are connected together and supplied with VCC.

Each segment lights up when current flows through it.

2. Electrical Operation

- Current Flow:
  - In a common cathode display, connecting a segment pin to HIGH allows current to flow from VCC through the segment to GND, lighting it up.
  - In a common anode display, connecting a segment pin to LOW allows current to flow from VCC through the anode and the segment to the pin, lighting it up.
- Controlling Segments: By applying HIGH/LOW signals to specific pins, the required segments are turned ON or OFF, forming a desired pattern (digit).

3. Logic Behind Displaying Numbers

Each digit (0 to 9) corresponds to a specific combination of segments that need to be turned ON:

- Example: To display **3**, segments a, b, c, d, g need to be ON, while e and f remain OFF.

The working principle of a 7-segment display involves selectively lighting up individual LEDs (segments) to form numbers or characters. Here's a detailed explanation.

4. Arduino Control

The Arduino controls the 7-segment display by:

4. Setting the pins connected to the display as output.
5. Using the binary pattern for each digit to decide which pins should be set to HIGH or LOW.
6. Sending signals to the pins to turn the appropriate segments ON or OFF.

6. Power Management

- Resistors: Current-limiting resistors are used to protect the LEDs from excess current.
- Power Supply: The Arduino provides the required power to the display, typically 5V.

Applications:

- Clocks and Timers
- Counters
- Measuring Instruments
- Electronic Gadgets
- Home Appliances
- Automobiles
- Consumer Electronics
- Medical Equipment
- Information Displays Boards.

Key Points to Remember:

• Types and Configuration:

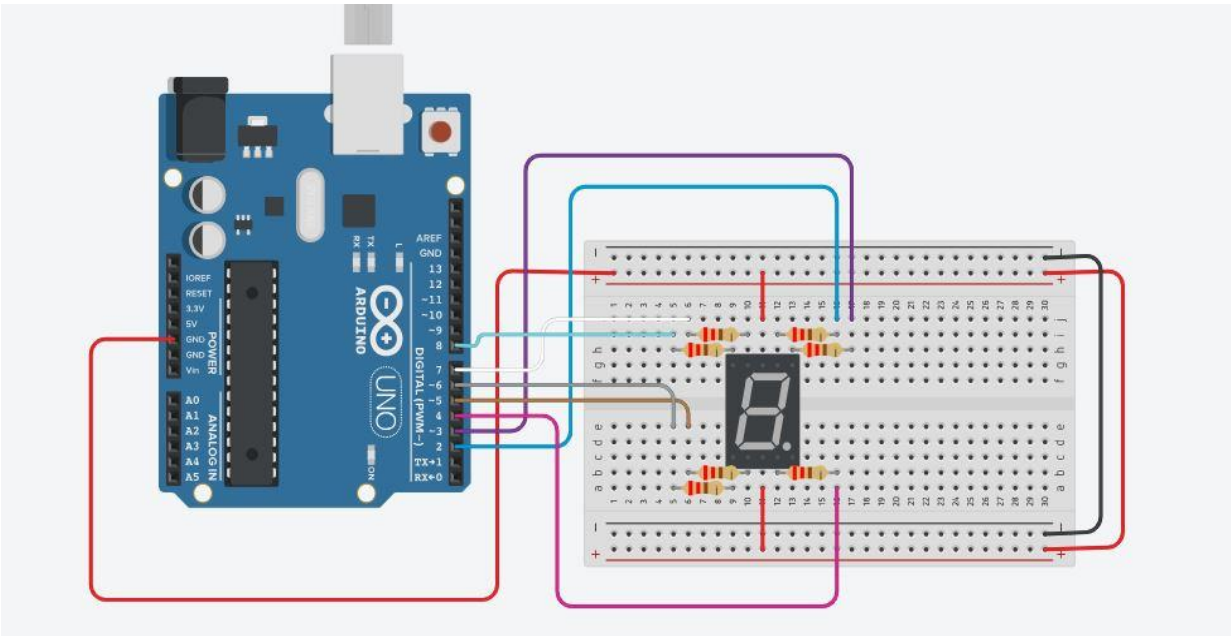
- Common Cathode (CC): Segments light up with HIGH signals.
- Common Anode (CA): Segments light up with LOW signals.

• Power and Protection:

- Operates at 5V; use current-limiting resistors (220Ω–330Ω) to prevent damage.

- Control and Logic:
    - Each digit has a unique binary pattern to light specific segments; use Arduino or driver ICs for control.

Connection:



Code:

```
// Define the segment pins for the 7-segment display
int a = 2;  // Segment a
int b = 3;  // Segment b
int c = 4;  // Segment c
int d = 5;  // Segment d
int e = 6;  // Segment e
int f = 7;  // Segment f
int g = 8;  // Segment g
// Function to display a letter (A to F) on a 7-segment display
void displayLetter(char letter) {
  switch (letter) {
    case 'A': // Display A
      digitalWrite(a, HIGH);
      digitalWrite(b, HIGH);
      digitalWrite(c, HIGH);
      digitalWrite(d, LOW);
      digitalWrite(e, HIGH);
      digitalWrite(f, HIGH);
      digitalWrite(g, HIGH);
      break;
    case 'b': // Display b
      digitalWrite(a, LOW);
      digitalWrite(b, LOW);
      digitalWrite(c, HIGH);
      digitalWrite(d, HIGH);
```

```
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
   case 'C': // Display C
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, LOW);
    break;
   case 'd': // Display d
    digitalWrite(a, LOW);
    digitalWrite(b, HIGH);
    digitalWrite(c, HIGH);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, LOW);
    digitalWrite(g, HIGH);
    break;
   case 'E': // Display E
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, HIGH);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
   case 'F': // Display F
    digitalWrite(a, HIGH);
    digitalWrite(b, LOW);
    digitalWrite(c, LOW);
    digitalWrite(d, LOW);
    digitalWrite(e, HIGH);
    digitalWrite(f, HIGH);
    digitalWrite(g, HIGH);
    break;
  }
}

void setup() {
 // Set the segment pins as output
 pinMode(a, OUTPUT);
 pinMode(b, OUTPUT);
 pinMode(c, OUTPUT);
 pinMode(d, OUTPUT);
 pinMode(e, OUTPUT);
 pinMode(f, OUTPUT);
```

```
  pinMode(g, OUTPUT);
}

void loop() {
  // Display letters A, b, C, d, E, F with a 1-second delay between each
  displayLetter('A');
  delay(1000);

  displayLetter('b');
  delay(1000);

  displayLetter('C');
  delay(1000);

  displayLetter('d');
  delay(1000);

  displayLetter('E');
  delay(1000);

  displayLetter('F');
  delay(1000);
}
```

Experiment-2

Liquid Crystal Display LCD (Right Scrolling LCD Display):

To create a right-scrolling display on an **LCD (16x2)** using Arduino, you can use the `Liquid Crystal` library, which provides functions to manipulate text and implement scrolling effects.

Components of a Liquid Crystal Display LCD (Right Scrolling LCD Display):

1. 16x2 LCD Module
2. Arduino Board
3. 10kΩ Potentiometer
4. 220Ω Resistors 1
5. Connecting Wires
6. Breadboard
7. Breadboard
8. Power Source

Working Principle:

• Structure of an LCD:

- The 16x2 LCD consists of 16 columns and 2 rows, making it capable of displaying 32 characters at a time.
- Each character is formed using a 5x8 pixel matrix.
- The LCD uses a controller (e.g., HD44780) to interpret commands and manage the display.

• LCD Control with Microcontroller:

- The LCD is interfaced with a microcontroller (e.g., Arduino) through a set of control pins (`RS`, `RW`, `E`) and data pins (`D0` to `D7` in 8-bit mode or `D4` to `D7` in 4-bit mode).
- Commands are sent to the LCD to control its functionality, including text scrolling.

• Scrolling Mechanism:

- The scrolling is achieved by sending specific commands to the LCD.
- The "Shift Display Right" command (`0x1C`) is used to move all the text on the display one position to the right.
- The microcontroller periodically sends this command in a loop to create a smooth scrolling effect.

• Right Scrolling Steps:

- Step 1: Initialize the LCD in 4-bit or 8-bit mode using the `lcd.begin()` or `lcd.init()` function in Arduino.
- Step 2: Display the desired text using the `lcd.print()` function.
- Step 3: Introduce a loop that repeatedly sends the `Shift Display Right` command.
- Step 4: Introduce a delay between iterations to control the speed of scrolling.

• Key LCD Commands:

- `0x01`: Clear display.
- `0x02`: Return cursor to the home position.
- `0x0C`: Display ON, cursor OFF.
- `0x1C`: Shift display right.

Applications:

1. Digital Notice Boards
2. Industrial Automation Systems
3. Retail and Advertising
4. Transportation Systems
5. Security Systems
6. Automotive Displays
7. Event Displays
8. Energy Metering Systems
9. Health Monitoring Devices
10. Consumer Electronics

Key Points to Remember:

1.Basics of LCD:

- LCDs (e.g., 16x2) are commonly used for displaying alphanumeric characters.
- A controller (e.g., HD44780) handles the command and data inputs for the display.

2.Right Scrolling Command:

- The **"Shift Display Right"** command (0x1C) moves all characters one position to the right.
- It is essential for dynamic message displays.

3.Code Execution:

- Scrolling is achieved through a loop that sends the right-shift command at regular intervals.
- Use **delays** (delay()) to control scrolling speed.

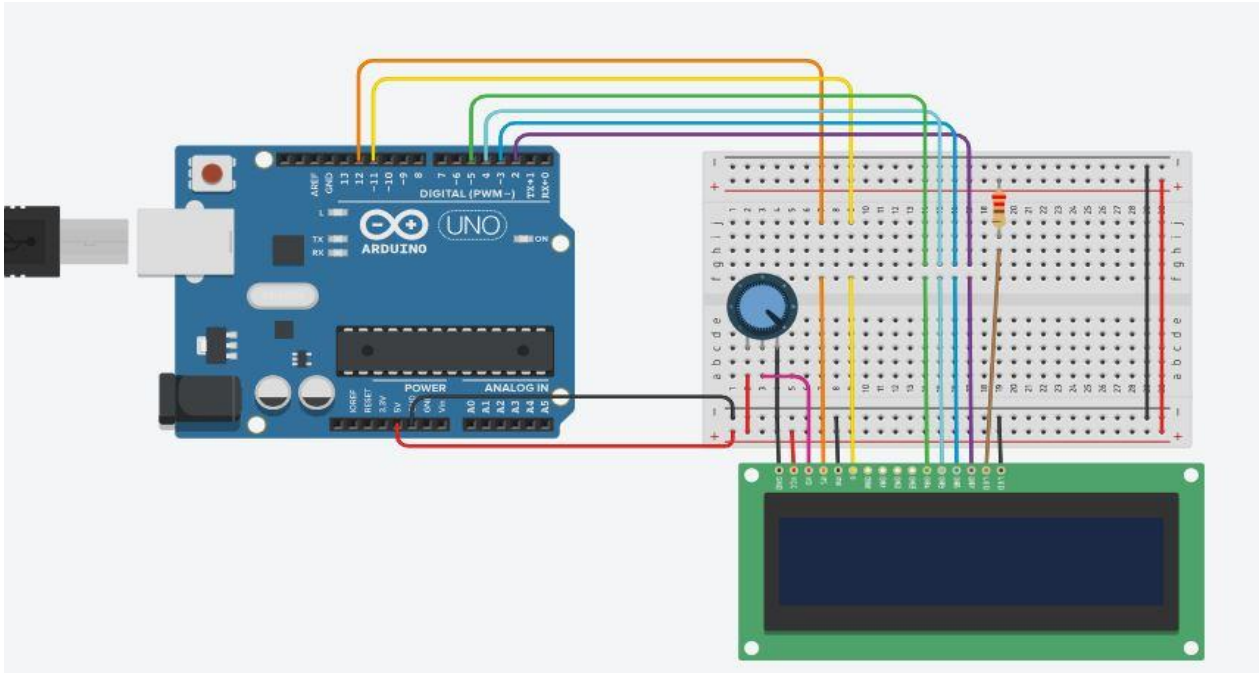
4.Debugging:

- Check wiring and initialization steps if the scrolling doesn't work.
- Verify correct use of the LCD library (e.g., LiquidCrystal for Arduino).

5.Common Commands:

- 0x01 – Clear display.
- 0x02 – Return cursor to home.
- 0x0C – Turn on display, turn off cursor.
- 0x1C – Shift display right.

Connection:



Code:

```
#include <LiquidCrystal.h>
// Initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // RS, E, D4, D5, D6, D7

// Define the sentences for both rows
String firstRowMessage = " The quick brown fox jumps over the lazy dog ";  // First row sentence
String secondRowMessage = " Scroll second line after first line finishes ";  // Second row sentence

void setup() {
  // Set up the number of columns and rows on the LCD
  lcd.begin(16, 2);
  lcd.clear(); // Clear the screen
}

void loop() {
  // Step 1: Scroll the first row from left to right
  for (int position = 0; position < firstRowMessage.length(); position++) {
    // Clear the screen
    lcd.clear();

    // Scroll the first row message
    lcd.setCursor(0, 0); // Set cursor to the first row
    lcd.print(firstRowMessage.substring(position, position + 16)); // Show 16 chars

    // Keep second row blank while first row scrolls
    lcd.setCursor(0, 1);
```

```
    lcd.print(" "); // Optionally keep this blank or display something static

    // Wait before scrolling the next character
    delay(300);  // Adjust the delay to control the scroll speed
   }

   // Step 2: Scroll the second row from left to right after the first row finishes
   for (int position = 0; position < secondRowMessage.length(); position++) {
    // Clear the screen
    lcd.clear();

    // Keep first row static or blank after it finishes
    lcd.setCursor(0, 0);
    lcd.print(" "); // Keep blank after first row scroll

    // Scroll the second row message
    lcd.setCursor(0, 1); // Set cursor to the second row
    lcd.print(secondRowMessage.substring(position, position + 16)); // Show 16 chars

    // Wait before scrolling the next character
    delay(300);  // Adjust the delay to control the scroll speed
   }

   // Optional: Delay before restarting the loop
   delay(2000);  // Wait for 2 seconds before repeating the scroll
  }
```

Experiment-3

Analog voltage reader on LCD Display:

To create an analog voltage reader on an LCD display using a microcontroller like Arduino, you can use an analog-to-digital converter (ADC) and display the voltage readings on a 16x2 LCD**.** Here's how you can achieve this:

Components of an Analog voltage reader on LCD Display:

- Arduino (e.g., UNO, Nano, etc.)
- 16x2 LCD display
- Potentiometer (10kΩ for LCD contrast adjustment)
- Breadboard
- Wires

Working Principle:

- The Arduino reads the analog voltage from a pin using the `analogRead()` function.
- The ADC in Arduino converts the voltage (0-5V for most boards) to a digital value (0-1023 for a 10-bit ADC).
- The actual voltage is calculated using the formula:

$$Voltage = (\frac{ADC\ Value}{1023}) \times V_{REF}$$

Here, V_{REF} is the reference voltage (usually 5V or 3.3V).

Applications:

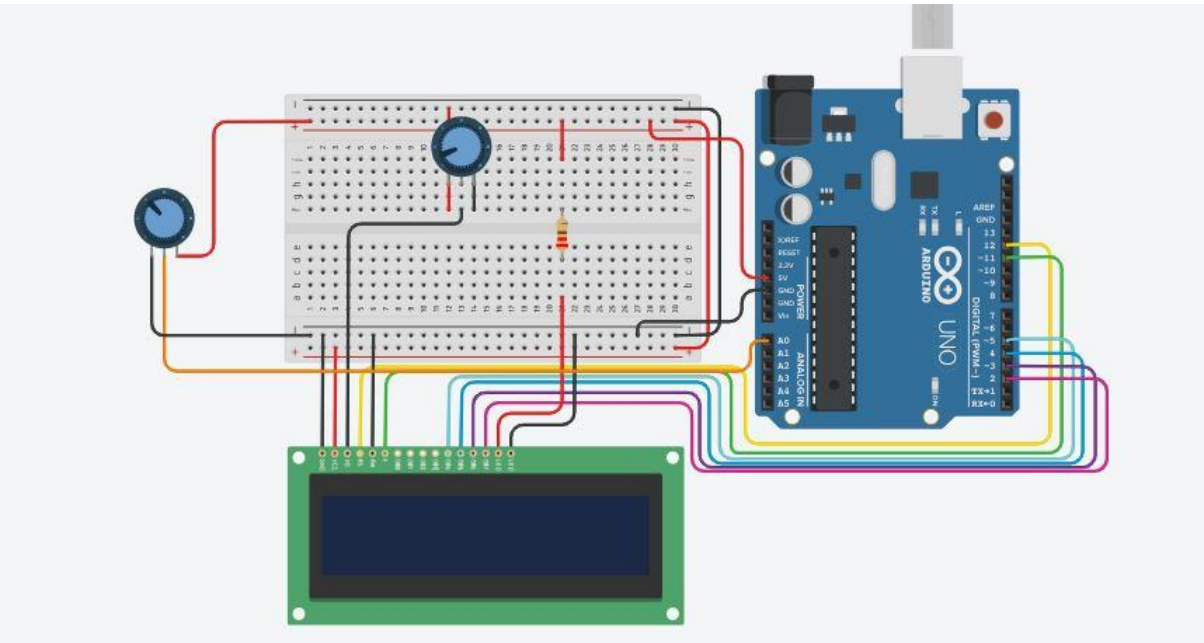
1. Battery Monitoring: Display voltage levels of batteries.
2. Sensor Interfacing: Measure and display sensor output voltages.
3. Educational Projects: Demonstrate analog voltage measurement.
4. DIY Multimeter: Create a basic voltmeter for testing circuits.

Key Points to Remember:

Hardware Setup

1. **Analog Input:**
   - Use an analog pin (e.g., A0) to read voltage values.
   - Ensure the input voltage is within the ADC range (usually 0-5V for most Arduinos).
2. **LCD Connections:**
   - Connect the 16x2 LCD in 4-bit or 8-bit mode.
   - Use a potentiometer to adjust the LCD contrast for better readability.
3. **Voltage Scaling:**
   - If the input voltage exceeds the ADC's maximum range, use a **voltage divider** to scale it down.
   - Verify scaling calculations to avoid incorrect readings.

Connection:



Code:

```
#include <LiquidCrystal.h>
// Initialize the LCD (RS = 12, EN = 11, D4 = 5, D5 = 4, D6 = 3, D7 = 2)
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int potPin = A0; // Potentiometer connected to A0
const float supplyVoltage = 5.0; // Arduino supply voltage (5V)
const int analogMax = 1023; // Maximum value from analogRead
const float Rref = 10000.0; // Reference resistor (10K Ohms)

float voltage; // Measured voltage across potentiometer
float resistance; // Calculated resistance of the potentiometer
float current;
float power;

void setup() {
  lcd.begin(16, 2); // Initialize a 16x2 LCD
  lcd.print("Initializing...");
  delay(1000);
  lcd.clear();
}

void loop() {
  // Read the voltage from the potentiometer
  int sensorValue = analogRead(potPin);
  voltage = (sensorValue * supplyVoltage) / analogMax; // Convert analog reading to voltage

  // Calculate potentiometer resistance using voltage divider formula
  // Rpot = (Vout * Rref) / (Vin - Vout)
  resistance = (voltage * Rref) / (supplyVoltage - voltage);
```

```
  // Calculate current using Ohm's law: I = V / R
  current = voltage / resistance;

  // Calculate power: P = V * I
  power = voltage * current;

  // Clear the LCD before updating
  lcd.clear();

  // Display voltage and resistance on the first row
  lcd.setCursor(0, 0);
  lcd.print("V: ");
  lcd.print(voltage);
  lcd.print("V");

  lcd.setCursor(9, 0);
  lcd.print("R: ");
  lcd.print(resistance);
  lcd.print(" Ohm");

  // Display current and power on the second row
  lcd.setCursor(0, 1);
  lcd.print("I: ");
  lcd.print(current * 1000); // Convert to mA for readability
  lcd.print("mA");

  lcd.setCursor(10, 1);
  lcd.print("P: ");
  lcd.print(power * 1000); // Convert to mW
  lcd.print("mW");

  // Wait 500 milliseconds before the next update
  delay(500);
}
```

Experiment-4

Temperature & Humidity Sensor:

A Temperature & Humidity Sensor is a device used to measure environmental conditions. Popular sensors like the DHT11 and DHT22 provide both temperature and humidity readings in a single module. These sensors are widely used in embedded systems, IoT applications, and weather monitoring stations.

Components of a Temperature & Humidity Sensor:

- Arduino (e.g., UNO, Nano, etc.)
- 16x2 LCD display
- Potentiometer (10kΩ for LCD contrast adjustment)
- Temperature sensor
- Breadboard
- Wires
- Resistor 10 kΩ

Working Principle:

1. **Humidity Sensing:**
    - Uses a **capacitive humidity sensor** to measure the relative humidity (RH).
    - The sensor's capacitance changes with moisture levels, and this change is converted into a digital signal.
2. **Temperature Sensing:**
    - Uses a **thermistor** or semiconductor temperature sensor to measure temperature.
    - The resistance changes with temperature, which is translated into a digital signal.
3. **Communication Protocol:**
    - The sensor uses a **single-wire serial interface** for data transmission, simplifying communication with microcontrollers

Applications:

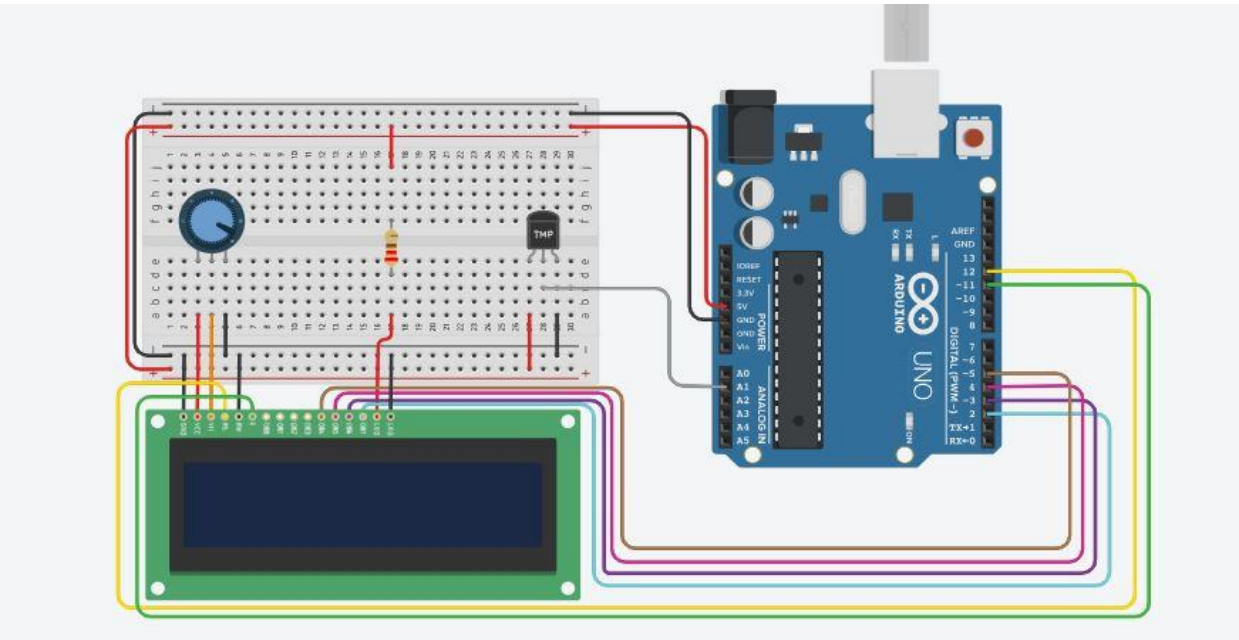
1.Weather Monitoring Station
2.Home Automation
3.Greenhouse Monitoring
4.Industrial Monitoring
5.Healthcare

Key Points to Remember:

1. Sensor Selection
2. Power Supply
3. Connections
4. Read Interval
5. Communication
6. Calibration
7. Troubleshooting

- Check wiring and connections if the sensor fails to read.
- Use a logic analyser or oscilloscope to verify data signals if debugging communication.

Connection:



Code:

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

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

// Define DHT sensor pin and type
#define DHTPIN A1
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  lcd.begin(16, 2); // Set up the number of columns and rows
  lcd.print("Temp: "); // Print label for temperature
  lcd.setCursor(0, 1);
  lcd.print("Hum: "); // Print label for humidity

  dht.begin(); // Initialize the DHT sensor
  Serial.begin(9600); // Start serial communication for debugging
}

void loop() {
```

```
 // Read temperature and humidity values
 float temperatureC = dht.readTemperature(); // Read temperature in Celsius
 float humidity = dht.readHumidity(); // Read humidity percentage

 // Check if readings are valid
 if (isnan(temperatureC) || isnan(humidity)) {
  lcd.setCursor(6, 0);
  lcd.print("Error"); // Display error if reading fails
  lcd.setCursor(5, 1);
  lcd.print("Sensor");
  Serial.println("Failed to read from DHT sensor!");
 } else {
  // Display temperature on the first row
  lcd.setCursor(6, 0);
  lcd.print(temperatureC, 1); // Print temperature with 1 decimal place
  lcd.print(" C "); // Celsius symbol and clear old values

  // Display humidity on the second row
  lcd.setCursor(5, 1);
  lcd.print(humidity, 1); // Print humidity with 1 decimal place
  lcd.print(" % "); // Percent symbol and clear old values

  // Debugging outputs to Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.println(" C");
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
 }

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

Experiment-5

Gas Sensor with LCD Display:

A gas sensor can detect and measure the concentration of gases in the environment. Combining it with a 16x2 LCD display allows you to monitor gas levels in real-time. Common gas sensors like the MQ series (e.g., MQ-2, MQ-5) are widely used for this purpose.

Components of an Ultrasonic Sensor:

- Gas Sensor (e.g., MQ-2, MQ-5, etc.)
- Arduino Board (e.g., Uno, Nano)
- 16x2 LCD Display
- Potentiometer (10kΩ for LCD contrast adjustment)
- Resistors (as needed for additional circuitry)
- Jumper Wires and Breadboard

Working Principle:

1. Gas Detection:
    - Gas sensors (MQ series) have a sensing element whose resistance changes when exposed to specific gases.
    - The sensor outputs an analog voltage proportional to the gas concentration.
2. Analog-to-Digital Conversion:
    - The Arduino reads the analog voltage from the sensor using its ADC (Analog-to-Digital Converter).
    - The gas concentration is calculated based on the sensor's characteristics.
3. Display:
    - The calculated gas concentration is displayed on the 16x2 LCD.

Applications:

1. Air Quality Monitoring
2. Safety Systems
3. IoT Integration
4. Environmental Projects

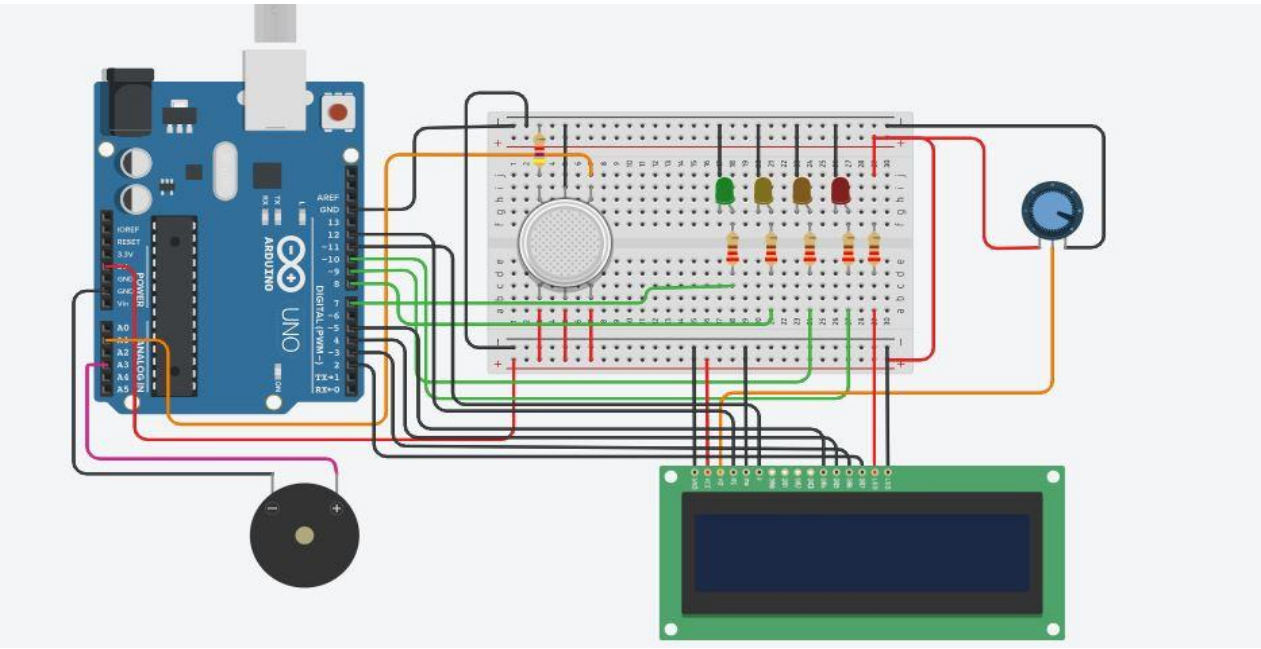
Key Points to Remember:

- Sensor **Warm-up:**
    - Gas sensors require a warm-up time (2–5 minutes) for accurate operation.
- **Voltage Range:**

- Ensure the analog output is within the Arduino's ADC range (0–5V).

- **Display Update:**
  - Introduce delays to ensure the LCD updates without flickering.
- **Environmental Factors:**
  - Place the sensor in a location free of direct sunlight or extreme conditions for consistent readings.
- **Sensor Lifetime:**
  - Gas sensors degrade over time; recalibrate periodically for accuracy.

Connection:



Code:

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

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

int const GAS_PIN = A1;  // Gas sensor pin
int GREEN_LED = 7;      // Safe indicator
int YELLOW_LED = 8;     // Warning indicator (changed to pin 8)
int RED_LED1 = 9;       // Danger indicator (changed to pin 9)
int RED_LED2 = 10;      // Hazard indicator (changed to pin 10)
```

```
int BUZZER = A3;       // Buzzer for alert

void setup() {
  pinMode(GREEN_LED, OUTPUT);
  pinMode(YELLOW_LED, OUTPUT);
  pinMode(RED_LED1, OUTPUT);
  pinMode(RED_LED2, OUTPUT);
  pinMode(BUZZER, OUTPUT);

  // Initialize LCD
  lcd.begin(16, 2);  // Set up the LCD's number of columns and rows
  lcd.print("Gas Detector");  // Initial message

  Serial.begin(9600);
  delay(2000);  // Hold message for 2 seconds
  lcd.clear();  // Clear LCD screen
}

void loop() {
  int value = map(analogRead(GAS_PIN), 300, 750, 0, 100);

  // Control LEDs and Buzzer based on gas value
  digitalWrite(GREEN_LED, HIGH);  // Green LED always ON

  digitalWrite(YELLOW_LED, value >= 30 ? HIGH : LOW);
  digitalWrite(RED_LED1, value >= 50 ? HIGH : LOW);
  digitalWrite(RED_LED2, value >= 80 ? HIGH : LOW);

  if (value >= 80) {
    tone(BUZZER, 440, 200);  // Activate buzzer if value >= 80
  }

  // Display alert message on LCD
  lcd.setCursor(0, 0);  // Set cursor to first row
  lcd.print("Gas Level: ");
  lcd.print(value);  // Display gas value

  lcd.setCursor(0, 1);  // Set cursor to second row
  if (value < 30) {
    lcd.print("Status: Safe    ");  // Safe level
  } else if (value < 50) {
    lcd.print("Status: Warning  ");  // Warning level
  } else if (value < 80) {
    lcd.print("Status: Danger   ");  // Danger level
  } else {
    lcd.print("Status: Hazard   ");  // Hazardous level
  }
```

```
  delay(250);  // Small delay between readings
}
```

Experiment-6

PIR Sensor with LCD Display:

A PIR (Passive Infrared) sensor detects motion by sensing infrared radiation changes in its surroundings. Here's a detailed explanation:

How a PIR Sensor Works:

Key Components:

1. Pyroelectric Sensor: Detects infrared radiation (heat) from objects like humans or animals.
2. Fresnel Lens: Focuses infrared radiation onto the pyroelectric sensor.
3. Control Circuit: Processes the signal and generates an output when motion is detected.

Working Principle:

- The PIR sensor detects changes in infrared levels within its detection area.
- When a warm object (like a person or animal) moves within its range, it causes a rapid change in the infrared energy detected by the sensor.
- This change triggers the sensor to output a HIGH signal.

PIR Sensor Details:

Pin Configuration:

1. VCC: Power supply (usually 5V or 3.3V).
2. OUT: Output pin to indicate motion (HIGH = motion detected, LOW = no motion).
3. GND: Ground.

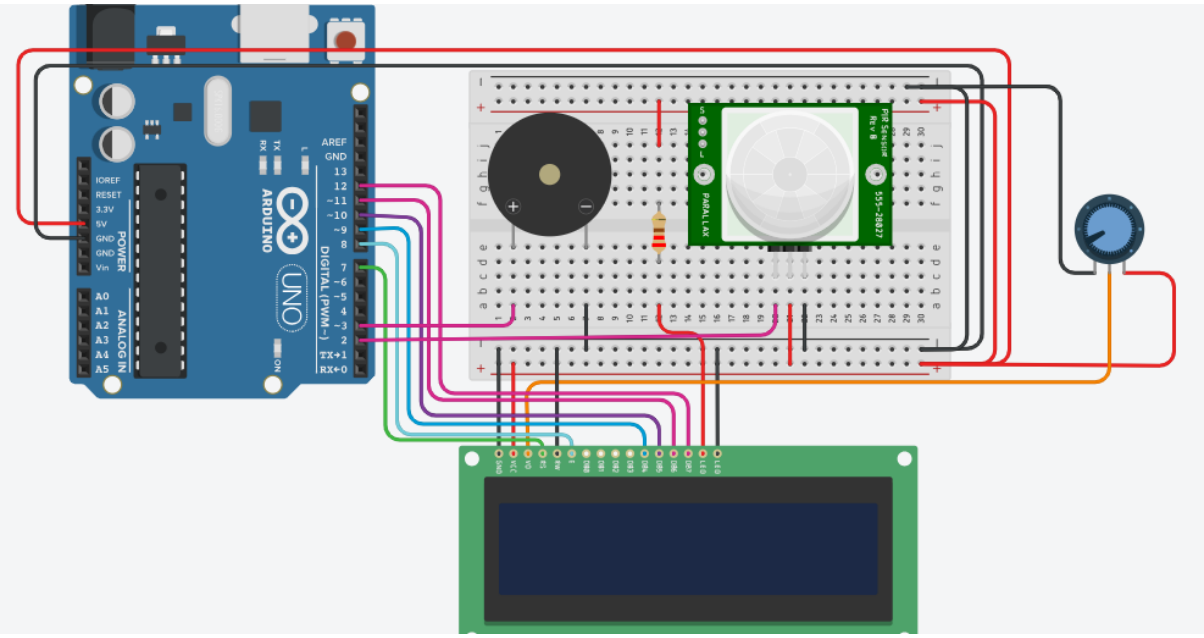
Adjustable Parameters:

- Sensitivity (Range): Adjusts how far the sensor can detect motion.
- Time Delay: Controls how long the output remains HIGH after motion is detected.

Specifications:

- Detection Range: Typically 3-7 meters.
- Operating Voltage: 3.3V to 5V.
- Viewing Angle: ~120°.

Connection:



Code:

```
#include <LiquidCrystal.h>
// Initialize the LCD (RS, E, D4, D5, D6, D7)
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
// Define the PIR sensor and buzzer pins
const int pirPin = 2;
const int buzzerPin = 3;
void setup() {
  // Set up the LCD's number of columns and rows
  lcd.begin(16, 2);
  // Print a welcome message
  lcd.print("PIR Sensor Init");
  // Set PIR sensor as input and buzzer as output
  pinMode(pirPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
  // Allow the PIR sensor to calibrate
  delay(5000);  // 5 seconds calibration time
  lcd.clear();
  lcd.print("Ready to Detect");
}
void loop() {
  // Read the PIR sensor output
  int motionDetected = digitalRead(pirPin);
  if (motionDetected == HIGH) {
    // If motion is detected
    lcd.clear();
    lcd.print("Motion Detected!");
    digitalWrite(buzzerPin, HIGH);  // Turn on the buzzer
    delay(1000);                    // Delay for 1 second
    digitalWrite(buzzerPin, LOW);   // Turn off the buzzer
```

```
  } else {
    // If no motion is detected
    lcd.clear();
    lcd.print("No Motion");
    delay(500);  // Delay for half a second
  }
}
```

Experiment-7

Ultrasonic Sensor with LCD Display:

An **ultrasonic sensor** works by emitting sound waves at a frequency higher than humans can hear (ultrasound) and then detecting the reflected waves (echo) to measure the distance to an object. Here's a step-by-step explanation of its working:

Components of an Ultrasonic Sensor:

1. **Transmitter:** Emits ultrasonic sound waves.
2. **Receiver:** Captures the reflected sound waves.
3. **Control Circuitry:** Processes the signals and calculates the distance.

Working Principle:

1. **Emission of Sound Waves:**
    - The transmitter sends out a short ultrasonic pulse (usually at 40 kHz).
    - This sound wave travels through the air.
2. **Reflection of Waves (Echo):**
    - If the sound wave encounters an object, it reflects back to the sensor.
    - The reflected wave (echo) is captured by the receiver.
3. **Time of Flight Measurement:**
    - The sensor measures the time it takes for the sound wave to travel to the object and back.
4. **Distance Calculation:**
    - The distance is calculated using the formula:

$$Distance = \frac{Speed\ of\ time \times Time\ Taken}{2}$$

        - The division by 2 accounts for the round trip of the sound wave.

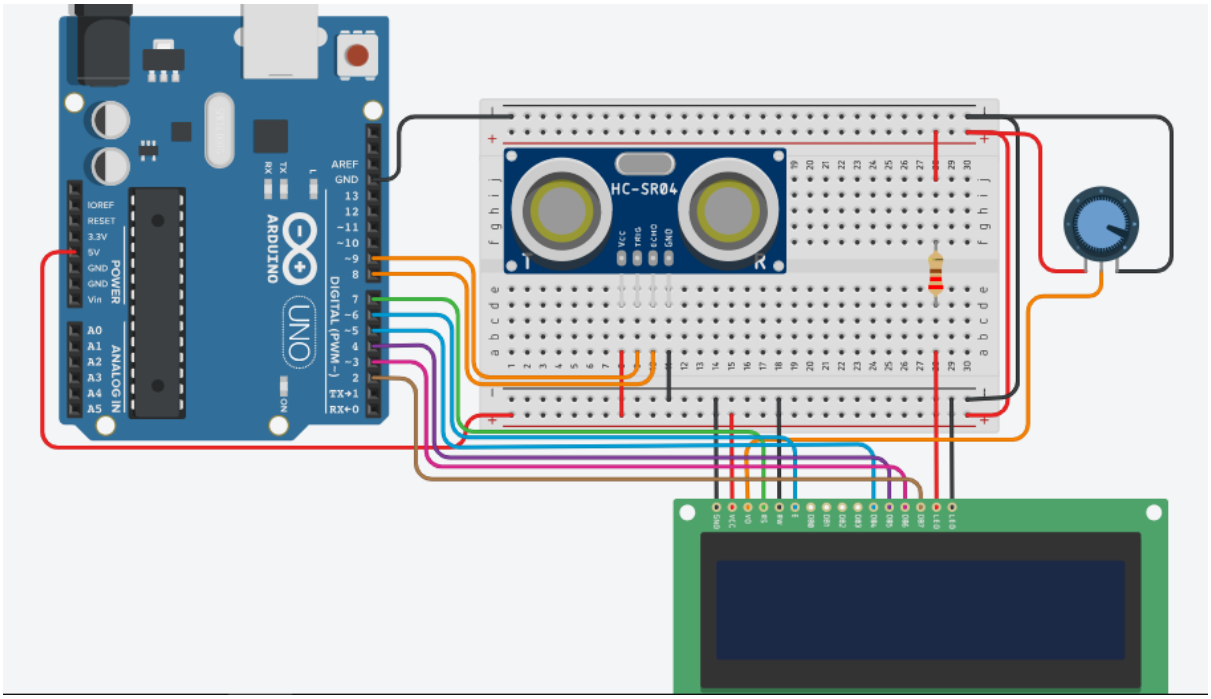
Applications:

- Object detection.
- Distance measurement.
- Obstacle avoidance in robotics.
- Water level monitoring.
- Parking sensors in vehicles.

Key Points to Remember:

- The speed of sound in air is approximately **343 m/s** at 20°C. Temperature changes can affect measurements.
- The sensor works best within its specified range (e.g., 2 cm to 400 cm for most models like HC-SR04).

Connection:



Code:

```
#include <LiquidCrystal.h>

// Initialize the LCD (RS, E, D4, D5, D6, D7)

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

// Define pins for the ultrasonic sensor

const int trigPin = 9;

const int echoPin = 8;

void setup() {

 // Set up the LCD's number of columns and rows

 lcd.begin(16, 2);

 lcd.print("Ultrasonic Init");
```

```
  // Set trigPin as output and echoPin as input
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  // Wait for the setup to complete
  delay(2000);
  lcd.clear();
  lcd.print("Measuring...");
}
void loop() {
  // Send a 10-microsecond pulse to the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Read the echoPin, and calculate the distance in cm
  long duration = pulseIn(echoPin, HIGH);
  int distance = duration * 0.034 / 2;  // Convert to cm

  // Display the distance on the LCD
  lcd.clear();
  lcd.print("Distance: ");
  lcd.print(distance);
  lcd.print(" cm");

  // Delay for a short period
  delay(500);  // 0.5-second delay
}
```