ZAS ROBOTICS
BUILD THE FUTURE

ZAS Robotics –Sensors

Foundation Projects

Mr. Fakruddin Mohammed & Dr. Roohi Banu
DIRECTORS OF ZAS ROBOTICS

Table of Contents

Chapter 1: Controlling an LED with Arduino UNO

Introduction

In this lesson, students will learn how to control an LED using an Arduino UNO microcontroller. They will understand the basics of programming the Arduino, wiring components, and the fundamental concepts of digital output.

Background

The Arduino UNO is a popular microcontroller platform that allows users to create interactive electronic projects. In this project, we will use the Arduino to turn an LED on and off at regular intervals. This exercise will introduce students to basic programming concepts, circuit design, and the functionality of microcontrollers.

Learning Objectives

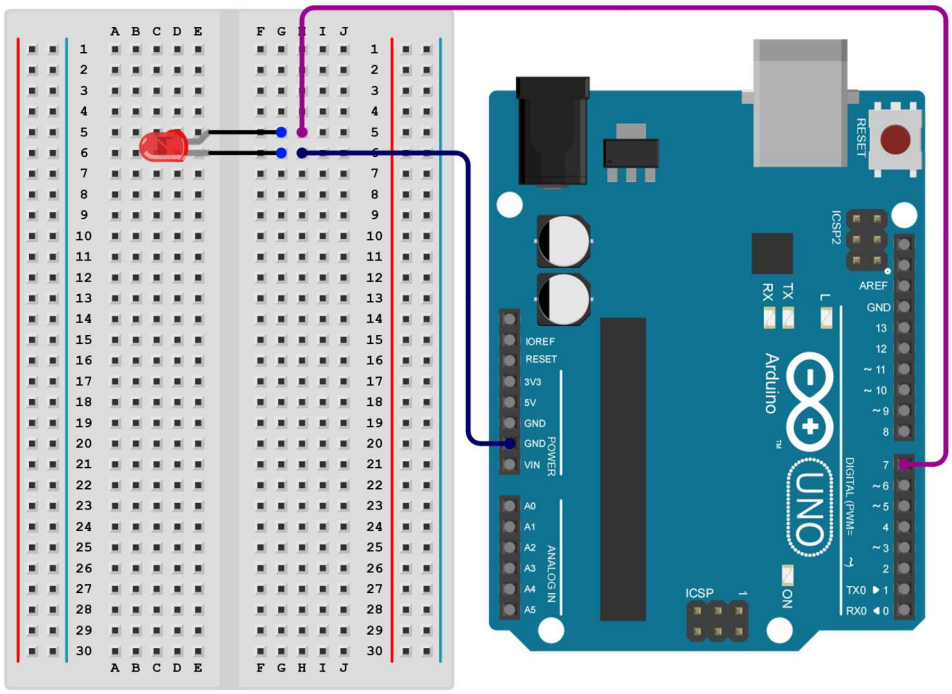
By the end of this lesson, students will be able to:

1. Understand the basic components of an Arduino circuit.
2. Write and upload a simple Arduino sketch to control an LED.
3. Wire components correctly on a breadboard.
4. Explain the concepts of digital output and delay in programming.

Functionality

The Arduino sketch provided will control an LED connected to pin D7. The LED will turn on for one second and then turn off for one second, repeating this cycle indefinitely. This demonstrates how to use digital signals to control electronic components.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components:**
    - Arduino UNO
    - LED (red)
    - Breadboard
    - Jumper wires
2. **Connect the LED:**
    - Insert the LED into the breadboard.
    - Connect the anode (longer leg) of the LED to pin D7 on the Arduino using a jumper wire.
    - Connect the other end of the resistor to the GND pin on the Arduino.
3. **Power the Arduino:**
    - Connect the Arduino to a power source (USB or battery).

4. **Upload the Code:**
    - Open the Arduino IDE on your computer.
    - Copy and paste the following code into the IDE:

```
/*
 * Arduino Sketch for controlling an LED connected to Pin 7.
 * The LED will turn on for 1 second, then off for 1 second, repeatedly.
 */

const int ledPin = 7; // Pin connected to the anode of the LED

void setup() {
  pinMode(ledPin, OUTPUT); // Set the LED pin as an output
}

void loop() {
  digitalWrite(ledPin, HIGH); // Turn the LED on
  delay(1000);                // Wait for 1 second
  digitalWrite(ledPin, LOW);  // Turn the LED off
  delay(1000);                // Wait for 1 second
}
```

5. **Upload the Code:**
    - Select the correct board and port in the Arduino IDE.
    - Click on the upload button to transfer the code to the Arduino.
6. **Observe the LED:**
    - Once the code is uploaded, observe the LED blinking on and off.

Using ZAS Creative Board

Image – 1

Image – 2

Conclusions

In this lesson, students learned how to control an LED using an Arduino UNO. They gained hands-on experience with wiring components, writing code, and understanding the basic principles of digital output. This foundational knowledge will serve as a stepping stone for more complex projects in the future. Encourage students to experiment with different timing intervals or add more LEDs to expand their learning.

Chapter 2: Traffic Light Simulation with Arduino

Introduction

In this lesson, students will learn how to create a traffic light simulation using an Arduino UNO and three LEDs (red, yellow, and green). This project will help students understand basic programming concepts, circuit design, and the functionality of microcontrollers.

Background

Traffic lights are essential for managing vehicle and pedestrian traffic at intersections. This project simulates a traffic light sequence using LEDs, allowing students to visualize how traffic lights operate. The Arduino UNO will control the timing and sequence of the LEDs, providing hands-on experience with programming and electronics.

Learning Objectives

By the end of this lesson, students will be able to:

1. Understand the basic components of a circuit.
2. Write and upload code to an Arduino microcontroller.
3. Explain the sequence of a traffic light system.
4. Construct a simple circuit using LEDs and an Arduino.
5. Troubleshoot common issues in circuit design and programming.

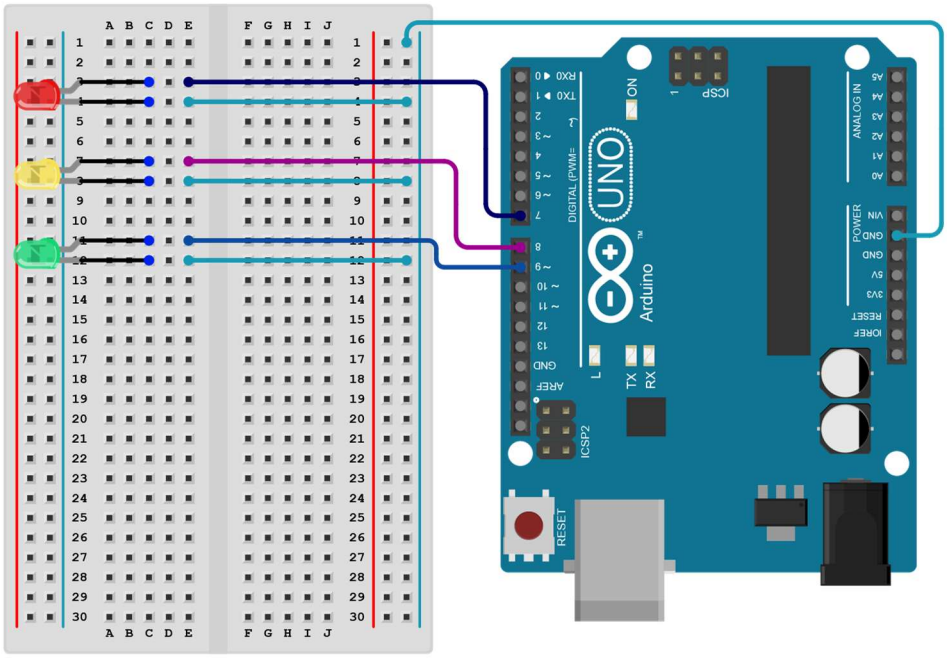
Functionality

The traffic light simulation will follow this sequence:

- Red LED ON for 5 seconds.
- Red and Yellow LEDs ON for 2 seconds.
- Green LED ON for 5 seconds.
- Yellow LED ON for 2 seconds. This cycle will repeat indefinitely, simulating a real traffic light.

Circuit Diagram

(Replace with actual diagram)



Step-by-Step Wiring Instructions

1. **Components Needed:**
    - 1 x Arduino UNO
    - 1 x Red LED
    - 1 x Yellow LED
    - 1 x Green LED
    - Breadboard and jumper wires
2. **Wiring Instructions:**
    - Connect the cathode (short leg) of each LED to the GND pin on the Arduino.
    - Connect the anode (long leg) of the Red LED to pin D9.
    - Connect the anode of the Yellow LED to pin D7.
    - Connect the anode of the Green LED to pin D10.
3. **Power the Arduino:**
    - Connect the Arduino to a power source via USB or an external power supply.

Sample Code

```
//Traffic Light Sequence Code
// Pin definitions
const int redPin = 9;    // Pin connected to the red LED
const int yellowPin = 7; // Pin connected to the yellow LED
const int greenPin = 10; // Pin connected to the green LED

void setup() {
  // Set the LED pins as outputs
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}

void loop() {
  // Red light on
  digitalWrite(redPin, HIGH);   // Turn on red light
  digitalWrite(yellowPin, LOW);  // Turn off yellow light
  digitalWrite(greenPin, LOW);   // Turn off green light
  delay(5000);                   // Keep red light on for 5
seconds

  // Green light on
  digitalWrite(redPin, LOW);     // Turn off red light
  digitalWrite(yellowPin, LOW);  // Turn off yellow light
  digitalWrite(greenPin, HIGH);  // Turn on green light
  delay(5000);                   // Keep green light on for 5
seconds

  // Yellow light on
  digitalWrite(redPin, LOW);     // Turn off red light
  digitalWrite(yellowPin, HIGH); // Turn on yellow light
  digitalWrite(greenPin, LOW);   // Turn off green light
  delay(2000);                   // Keep yellow light on for 2
seconds
}
```

Explanation of the Code

1. **Pin Definitions**: The code defines the pins for the red, yellow, and green LEDs.
2. **Setup**: In the setup() function, the LED pins are configured as outputs.
3. **Loop**: In the loop() function, the code controls the traffic light sequence:
    - The red light is turned on for 5 seconds.

- The green light is turned on for 5 seconds.
- The yellow light is turned on for 2 seconds.
- The sequence then repeats indefinitely.

Additional Notes

- Ensure that the LEDs are connected correctly with appropriate resistors to limit current and prevent damage.
- You can adjust the delay() times to change how long each light stays on based on your requirements.

Conclusions

This lesson provides students with a practical understanding of how to use an Arduino to control LEDs in a sequence that mimics a traffic light. By completing this project, students will gain valuable skills in programming, circuit design, and problem-solving. Encourage students to modify the timing or add additional features to enhance their learning experience.

Chapter 3: Using a Buzzer in Circuits

Introduction

In this lesson, students will learn about buzzers, their functionality, and how to integrate them into simple electronic circuits. Buzzers are commonly used in various applications, such as alarms, notifications, and sound effects in electronic devices.

Background

A buzzer is an electromechanical device that produces sound when an electrical signal is applied. There are two main types of buzzers: active and passive. Active buzzers generate sound when power is applied, while passive buzzers require an oscillating signal to produce sound. Understanding how to use buzzers can enhance students' knowledge of sound generation in electronics.

Learning Objectives

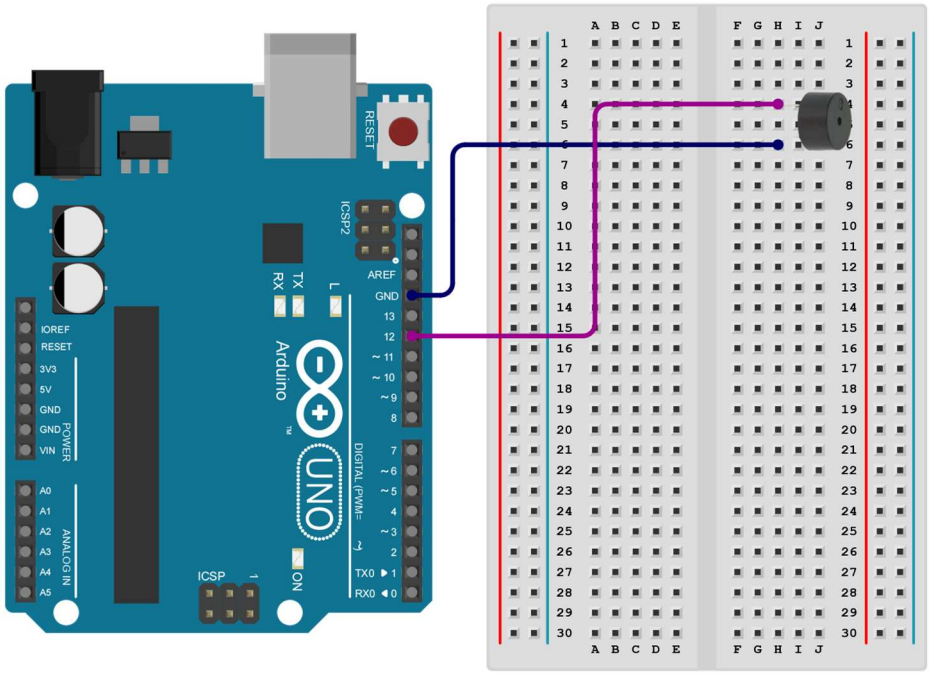
By the end of this lesson, students will be able to:

1. Identify the components of a buzzer circuit.
2. Understand the difference between active and passive buzzers.
3. Construct a simple circuit using a buzzer.
4. Write a basic program to control the buzzer using a microcontroller.

Functionality

The buzzer will be used to produce sound in response to a signal from a microcontroller. When the microcontroller sends a HIGH signal to the buzzer, it will emit a sound. This can be used in various applications, such as alerting users or indicating the completion of a task.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - Active Buzzer
    - Breadboard and jumper wires
    - Power supply (USB or battery)
2. **Connect the Buzzer**:
    - Connect the positive pin of the buzzer to a digital output pin on the microcontroller (e.g., Pin 12).
    - Connect the negative pin of the buzzer to the ground (GND) of the microcontroller.
3. **Power the Microcontroller**:
    - Connect the microcontroller to a power source (USB or battery).
4. **Upload Code**:
    - Write and upload the code to the microcontroller to control the buzzer.

Example Code (for Arduino)

```
#define BUZZER_PIN 12

void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
}

void loop() {
  digitalWrite(BUZZER_PIN, HIGH); // Turn on the buzzer
  delay(1000);                    // Wait for 1 second
  digitalWrite(BUZZER_PIN, LOW);  // Turn off the buzzer
  delay(1000);                    // Wait for 1 second
}
```

Conclusions

In this lesson, students learned how to use a buzzer in a circuit, the differences between active and passive buzzers, and how to control a buzzer using a microcontroller. This foundational knowledge can be applied to more complex projects involving sound generation and alerts in electronic devices. Encourage students to experiment with different sounds and patterns by modifying the code.

Chapter 4: Using an RGB LED in Circuits

Introduction

In this lesson, students will learn about RGB LEDs, their functionality, and how to integrate them into simple electronic circuits. RGB LEDs can emit a wide range of colors by combining red, green, and blue light, making them a versatile component for various applications, including decorative lighting and indicators.

Background

An RGB LED consists of three individual LEDs (red, green, and blue) housed in a single package. By varying the intensity of each color, students can create different colors and effects. Understanding how to control RGB LEDs will enhance students' skills in electronics and programming.

Learning Objectives

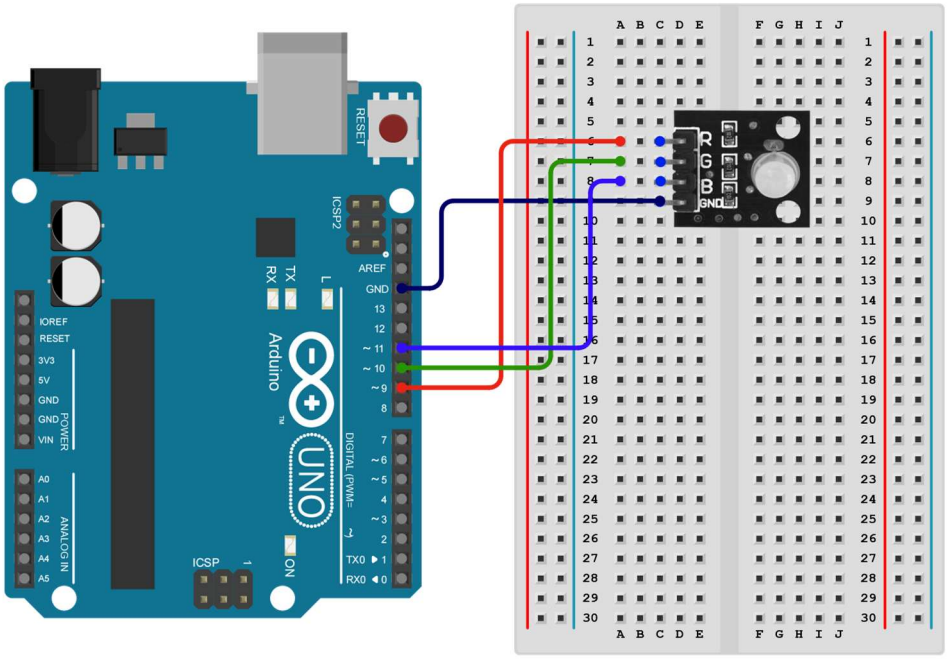
By the end of this lesson, students will be able to:

1. Identify the components of an RGB LED circuit.
2. Understand how to control the color output of an RGB LED.
3. Construct a simple circuit using an RGB LED.
4. Write a basic program to control the RGB LED using a microcontroller.

Functionality

The RGB LED will be used to display different colors based on the signals sent from a microcontroller. By adjusting the brightness of each color channel (red, green, and blue), students can create various colors and transitions.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - Common Cathode RGB LED
    - Breadboard and jumper wires
    - Power supply (USB or battery)
2. **Identify RGB LED Pins**:
    - Determine the pin configuration of the RGB LED (usually, the longest pin is the common cathode, and the other three pins are for red, green, and blue).
3. **Connect the RGB LED**:
    - Connect the common cathode pin of the RGB LED to the ground (GND) of the microcontroller.
    - Connect the red pin of the RGB LED to a PWM-capable digital output pin on the microcontroller (e.g., Pin 9) through a resistor.
    - Connect the green pin of the RGB LED to another PWM-capable digital output pin (e.g., Pin 10) through a resistor.

- Connect the blue pin of the RGB LED to another PWM-capable digital output pin (e.g., Pin 11) through a resistor.

4. **Power the Microcontroller**:
   - Connect the microcontroller to a power source (USB or battery).
5. **Upload Code**:
   - Write and upload the code to the microcontroller to control the RGB LED.

Example Code (for Arduino)

```
#define RED_PIN 9
#define GREEN_PIN 10
#define BLUE_PIN 11

void setup() {
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}

void loop() {
  // Red
  setColor(255, 0, 0);
  delay(1000);
  // Green
  setColor(0, 255, 0);
  delay(1000);
  // Blue
  setColor(0, 0, 255);
  delay(1000);
  // Yellow
  setColor(255, 255, 0);
  delay(1000);
  // Cyan
  setColor(0, 255, 255);
  delay(1000);
  // Magenta
  setColor(255, 0, 255);
  delay(1000);
  // Off
  setColor(0, 0, 0);
  delay(1000);
```

```
}

void setColor(int red, int green, int blue) {
  analogWrite(RED_PIN, red);
  analogWrite(GREEN_PIN, green);
  analogWrite(BLUE_PIN, blue);
}
```

Conclusions

In this lesson, students learned how to use an RGB LED in a circuit, how to control the colour output using a microcontroller, and how to create various colours by adjusting the intensity of each colour channel. This foundational knowledge can be applied to more complex projects involving lighting effects and visual indicators. Encourage students to experiment with different colour combinations and patterns by modifying the code.

Chapter 5: Using a Touch Sensor with LED and Buzzer

Introduction

In this lesson, students will learn how to use a touch sensor in conjunction with an LED and a buzzer. This project will demonstrate how touch sensors can be used to create interactive electronic devices that respond to user input, enhancing students' understanding of sensors and output devices.

Background

A touch sensor is a device that detects physical touch or proximity. When a user touches the sensor, it sends a signal to a microcontroller, which can then trigger various outputs, such as lighting an LED or sounding a buzzer. This lesson will help students understand the principles of touch sensing and how to integrate multiple components in a circuit.

Learning Objectives

By the end of this lesson, students will be able to:

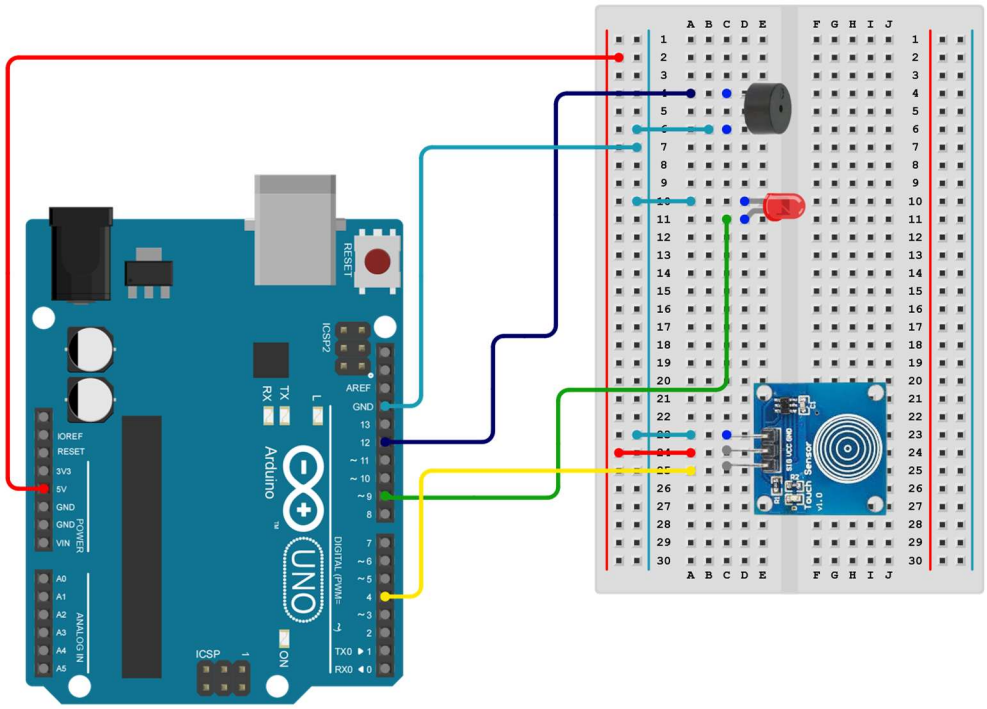
1. Identify the components of a touch sensor circuit.

2. Understand how a touch sensor works and its applications.
3. Construct a simple circuit using a touch sensor, LED, and buzzer.
4. Write a basic program to control the LED and buzzer based on touch input.

Functionality

The touch sensor will detect when a user touches it. Upon detection, the microcontroller will turn on an LED and activate a buzzer, providing both visual and auditory feedback. This interaction can be used in various applications, such as alarms, notifications, or simple user interfaces.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - Touch Sensor
    - LED
    - Active Buzzer

- Breadboard and jumper wires
- Power supply (USB or battery)

2. **Connect the Touch Sensor**:
   - Connect the VCC pin of the touch sensor to the 5V pin on the microcontroller.
   - Connect the GND pin of the touch sensor to the ground (GND) of the microcontroller.
   - Connect the OUT pin of the touch sensor to a digital input pin on the microcontroller (e.g., Pin 4).
3. **Connect the LED**:
   - Connect the anode (longer leg) of the LED to a digital output pin on the microcontroller (e.g., Pin 9) through a resistor.
   - Connect the cathode (shorter leg) of the LED to the ground (GND) of the microcontroller.
4. **Connect the Buzzer**:
   - Connect the positive pin of the buzzer to another digital output pin on the microcontroller (e.g., Pin 12).
   - Connect the negative pin of the buzzer to the ground (GND) of the microcontroller.
5. **Power the Microcontroller**:
   - Connect the microcontroller to a power source (USB or battery).
6. **Upload Code**:
   - Write and upload the code to the microcontroller to control the LED and buzzer based on touch input.

Example Code (for Arduino)

```
#define TOUCH_PIN 4
#define LED_PIN 9
#define BUZZER_PIN 12

void setup() {
  pinMode(TOUCH_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
```

```
}

void loop() {
  int touchState = digitalRead(TOUCH_PIN);

  if (touchState == HIGH) { // If the touch sensor is activated
    digitalWrite(LED_PIN, HIGH); // Turn on the LED
    digitalWrite(BUZZER_PIN, HIGH); // Turn on the buzzer
    delay(1000); // Keep them on for 1 second
    digitalWrite(LED_PIN, LOW); // Turn off the LED
    digitalWrite(BUZZER_PIN, LOW); // Turn off the buzzer
  }
}
```

Conclusions

In this lesson, students learned how to use a touch sensor in a circuit, how to control an LED and a buzzer based on touch input, and the principles behind touch sensing technology. This foundational knowledge can be applied to more complex projects involving user interfaces and interactive devices. Encourage students to experiment with different responses, such as varying the duration of the LED and buzzer activation or adding more outputs.

Chapter 6: Using a Servo Motor in Circuits

Introduction

In this lesson, students will learn about servo motors, their functionality, and how to integrate them into electronic circuits. Servo motors are widely used in robotics, automation, and control systems due to their precision and ability to maintain a specific position.

Background

A servo motor is a rotary actuator that allows for precise control of angular position, velocity, and acceleration. It consists of a motor, a feedback device (usually a potentiometer), and a control circuit. Servo motors are commonly used in applications such as robotic arms, remote-controlled vehicles, and automated systems. Understanding how to control a servo motor will enhance students' skills in electronics and programming.

Learning Objectives

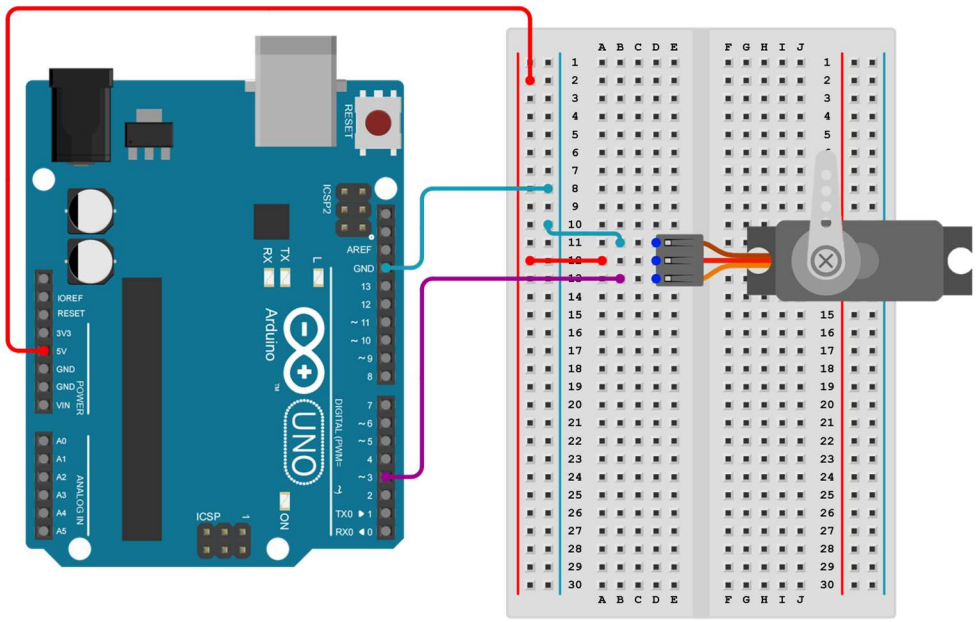
By the end of this lesson, students will be able to:

1. Identify the components of a servo motor circuit.
2. Understand how a servo motor operates and its applications.
3. Construct a simple circuit using a servo motor.
4. Write a basic program to control the position of the servo motor using a microcontroller.

Functionality

The servo motor will be controlled by a microcontroller, which sends a PWM (Pulse Width Modulation) signal to determine the angle of rotation. Students will learn how to set the servo to specific angles, allowing them to create movements in their projects.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - Servo Motor
    - Power supply (USB or battery)
    - Breadboard and jumper wires
2. **Connect the Servo Motor**:
    - Connect the VCC (red wire) of the servo motor to the 5V pin on the microcontroller.
    - Connect the GND (black or brown wire) of the servo motor to the ground (GND) of the microcontroller.
    - Connect the PWM control wire (usually yellow or orange) of the servo motor to a PWM-capable digital output pin on the microcontroller (e.g., Pin 3).
3. **Power the Microcontroller**:
    - Connect the microcontroller to a power source (USB or battery).

4. **Upload Code**:
    - Write and upload the code to the microcontroller to control the servo motor's position.

Example Code (for Arduino)

```
#include <Servo.h>

Servo myServo; // Create a Servo object

void setup() {
  myServo.attach(3); // Attach the servo to pin 9
}

void loop() {
  // Move the servo to 0 degrees
  myServo.write(0);
  delay(1000); // Wait for 1 second

  // Move the servo to 90 degrees
  myServo.write(90);
  delay(1000); // Wait for 1 second

  // Move the servo to 180 degrees
  myServo.write(180);
  delay(1000); // Wait for 1 second
}
```

Conclusions

In this lesson, students learned how to use a servo motor in a circuit, how to control its position using a microcontroller, and the principles behind servo motor operation. This foundational knowledge can be applied to more complex projects involving robotics and automation. Encourage students to experiment with different angles and movements, such as creating a simple robotic arm or a rotating display.

Chapter 7: Using an IR Transmitter/Receiver Sensor with a Servo Motor

Introduction

In this lesson, students will learn how to use an IR (Infrared) transmitter and receiver sensor to control a servo motor. This project will demonstrate how IR communication can be used for remote control applications, allowing students to understand the principles of wireless communication and motor control.

Background

An IR transmitter emits infrared light, while an IR receiver detects this light. When the transmitter sends a signal, the receiver can interpret it and trigger actions in a circuit, such as moving a servo motor. This technology is commonly used in remote controls, obstacle detection, and various automation applications. Understanding how to integrate IR sensors with motors will enhance students' skills in electronics and programming.

Learning Objectives

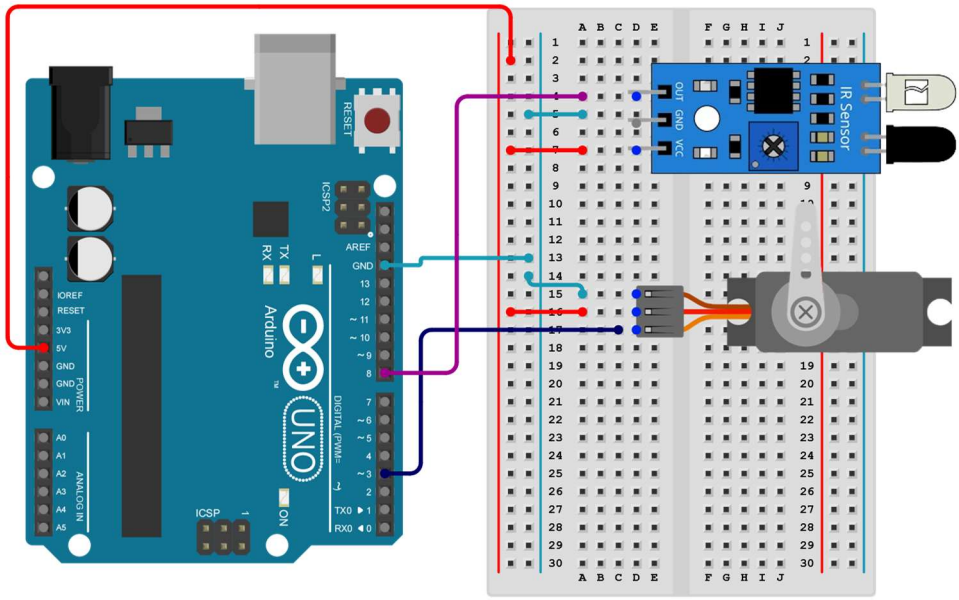
By the end of this lesson, students will be able to:

1. Identify the components of an IR transmitter/receiver circuit.
2. Understand how IR communication works and its applications.
3. Construct a simple circuit using an IR transmitter, IR receiver, and a servo motor.
4. Write a basic program to control the position of the servo motor based on IR signals.

Functionality

The IR transmitter will send a signal when activated, and the IR receiver will detect this signal. Upon receiving the signal, the microcontroller will control the servo motor to move to a specified position. This interaction can be used in various applications, such as remote-controlled devices or automated systems.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - IR Transmitter
    - IR Receiver
    - Servo Motor
    - Breadboard and jumper wires
    - Power supply (USB or battery)
2. **Connect the IR Transmitter**:
    - Connect the anode (longer leg) of the IR transmitter to a digital output pin on the microcontroller (e.g., Pin 3) through a resistor.
    - Connect the cathode (shorter leg) of the IR transmitter to the ground (GND) of the microcontroller.
3. **Connect the IR Receiver**:
    - Connect the VCC pin of the IR receiver to the 5V pin on the microcontroller.

- Connect the GND pin of the IR receiver to the ground (GND) of the microcontroller.
- Connect the OUT pin of the IR receiver to a digital input pin on the microcontroller (e.g., Pin 8).

4. **Connect the Servo Motor**:
   - Connect the VCC (red wire) of the servo motor to the 5V pin on the microcontroller.
   - Connect the GND (black or brown wire) of the servo motor to the ground (GND) of the microcontroller.
   - Connect the PWM control wire (usually yellow or orange) of the servo motor to a PWM-capable digital output pin on the microcontroller (e.g., Pin 9).
5. **Power the Microcontroller**:
   - Connect the microcontroller to a power source (USB or battery).
6. **Upload Code**:
   - Write and upload the code to the microcontroller to control the servo motor based on IR signals.

Example Code (for Arduino)

```
#include <Servo.h>

Servo myServo; // Create a Servo object
const int irReceiverPin = 8; // Pin for IR receiver

void setup() {
  myServo.attach(3); // Attach the servo to pin 9
  pinMode(irReceiverPin, INPUT);

}

void loop() {


  // Check for IR signal
  if (digitalRead(irReceiverPin) == LOW) {
    myServo.write(90); // Move servo to 90 degrees
    delay(1000); // Wait for 1 second
```

```
      myServo.write(0); // Move servo back to 0 degrees
    }
  }
```

Conclusions

In this lesson, students learned how to use an IR transmitter and receiver to control a servo motor, how IR communication works, and the principles behind motor control. This foundational knowledge can be applied to more complex projects involving remote control systems and automation. Encourage students to experiment with different IR signals and servo movements, such as creating a simple remote-controlled robotic arm or a door mechanism.

Chapter 8: Using an LDR (Light) Sensor with LED and Buzzer

Introduction

In this lesson, students will learn how to use an LDR (Light Dependent Resistor) to control an LED and a buzzer. This project will demonstrate how light sensors can be used to create interactive electronic devices that respond to changes in light levels, enhancing students' understanding of sensors and output devices.

Background

An LDR is a type of resistor whose resistance decreases with increasing incident light intensity. It is commonly used in light-sensing applications, such as automatic streetlights, light meters, and alarm systems. By integrating an LDR with an LED and a buzzer, students will learn how to create a simple light-activated system.

Learning Objectives

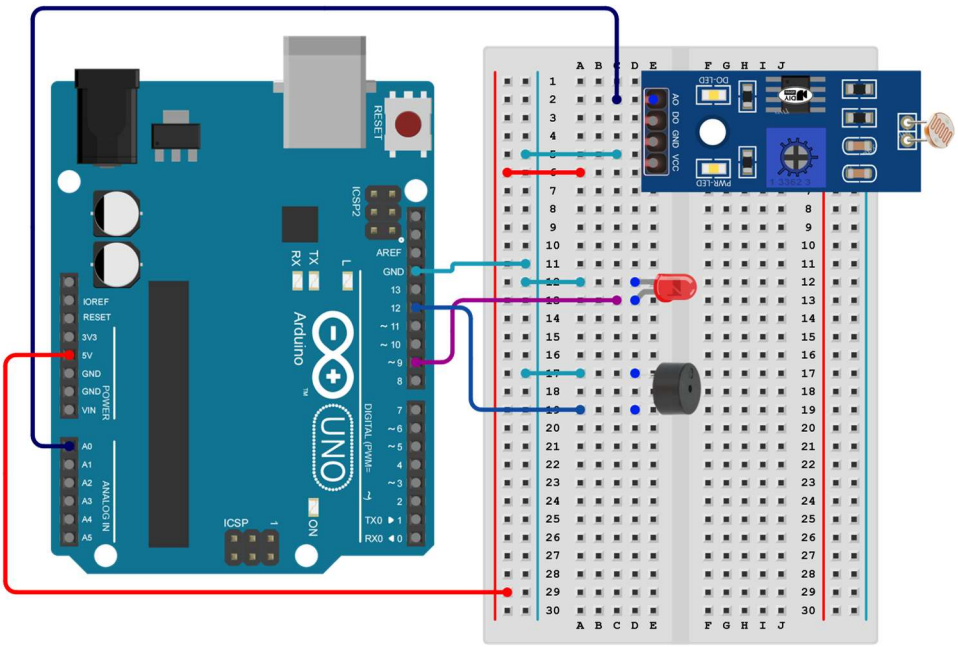
By the end of this lesson, students will be able to:

1. Identify the components of an LDR circuit.
2. Understand how an LDR works and its applications.
3. Construct a simple circuit using an LDR, LED, and buzzer.
4. Write a basic program to control the LED and buzzer based on light levels detected by the LDR.

Functionality

The LDR will detect the ambient light level. When the light level falls below a certain threshold, the microcontroller will turn on an LED and activate a buzzer, providing both visual and auditory feedback. This interaction can be used in various applications, such as alarms for low light conditions or automatic lighting systems.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - LDR (Light Dependent Resistor)
    - LED
    - Active Buzzer
    - Breadboard and jumper wires
    - Power supply (USB or battery)
2. **Connect the LDR**:
    - Connect one terminal of the LDR to the 5V pin on the microcontroller.
    - Connect the other terminal of the LDR to an analog input pin on the microcontroller (e.g., A0).
    - Connect a 10kΩ resistor from the analog input pin (A0) to the ground (GND) of the microcontroller. This creates a voltage divider circuit.
3. **Connect the LED**:

- Connect the anode (longer leg) of the LED to a digital output pin on the microcontroller (e.g., Pin 9) through a resistor (220Ω).
- Connect the cathode (shorter leg) of the LED to the ground (GND) of the microcontroller.

4. **Connect the Buzzer**:
   - Connect the positive pin of the buzzer to another digital output pin on the microcontroller (e.g., Pin 12).
   - Connect the negative pin of the buzzer to the ground (GND) of the microcontroller.
5. **Power the Microcontroller**:
   - Connect the microcontroller to a power source (USB or battery).
6. **Upload Code**:
   - Write and upload the code to the microcontroller to control the LED and buzzer based on the light level detected by the LDR.

Example Code (for Arduino)

```
const int ldrPin = A0; // Pin for LDR
const int ledPin = 9;  // Pin for LED
const int buzzerPin = 12; // Pin for Buzzer
int ldrValue = 0; // Variable to store LDR value

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  Serial.begin(9600); // Start serial communication for debugging
}

void loop() {
  ldrValue = analogRead(ldrPin); // Read the value from the LDR
  Serial.println(ldrValue); // Print the LDR value to the Serial
Monitor

  if (ldrValue > 500) { // Threshold for light level (adjust as
needed)
    digitalWrite(ledPin, HIGH); // Turn on the LED
    digitalWrite(buzzerPin, HIGH); // Turn on the buzzer
  } else {
    digitalWrite(ledPin, LOW); // Turn off the LED
```

```
    digitalWrite(buzzerPin, LOW); // Turn off the buzzer
  }
  delay(100); // Short delay for stability
}
```

Conclusions

In this lesson, students learned how to use an LDR in a circuit, how to control an LED and a buzzer based on light levels, and the principles behind light sensing technology. This foundational knowledge can be applied to more complex projects involving light-activated systems and automation. Encourage students to experiment with different light levels and thresholds, such as creating a light-sensitive alarm or an automatic night light.

Chapter 9: Using a Tilt Sensor with LED and Buzzer

Introduction

In this lesson, students will learn how to use a tilt sensor to control an LED and a buzzer. This project will demonstrate how tilt sensors can be used to create interactive electronic devices that respond to changes in orientation, enhancing students' understanding of sensors and output devices.

Background

A tilt sensor is a device that detects the orientation of an object. It can be used to determine whether an object is in a horizontal or vertical position. Tilt sensors are commonly used in applications such as alarms, gaming devices, and robotics. By integrating a tilt sensor with an LED and a buzzer, students will learn how to create a simple tilt-activated system.

Learning Objectives

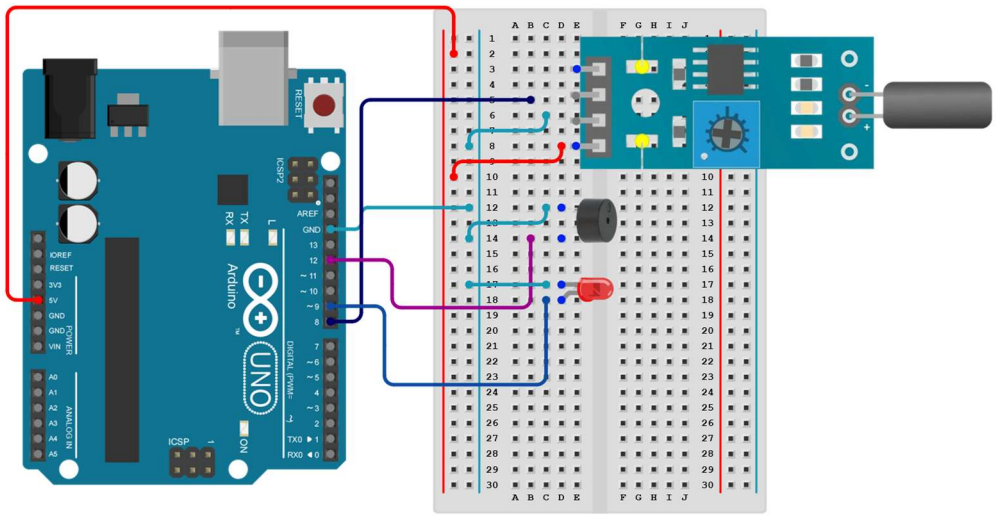
By the end of this lesson, students will be able to:

1. Identify the components of a tilt sensor circuit.
2. Understand how a tilt sensor works and its applications.
3. Construct a simple circuit using a tilt sensor, LED, and buzzer.
4. Write a basic program to control the LED and buzzer based on the tilt sensor's output.

Functionality

The tilt sensor will detect the orientation of the device. When the sensor is tilted beyond a certain angle, the microcontroller will turn on an LED and activate a buzzer, providing both visual and auditory feedback. This interaction can be used in various applications, such as alarms for unauthorized movement or notifications for specific orientations.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - Tilt Sensor (e.g., a simple ball tilt switch)
    - LED
    - Active Buzzer
    - Breadboard and jumper wires
    - Power supply (USB or battery)
2. **Connect the Tilt Sensor**:
    - Connect one terminal of the tilt sensor to a digital input pin on the microcontroller (e.g., Pin 8).
    - Connect the other terminal of the tilt sensor to the ground (GND) of the microcontroller.
    - Optionally, connect a pull-up resistor (10kΩ) from the digital input pin to the 5V pin on the microcontroller to ensure a stable HIGH state when the sensor is not activated.
3. **Connect the LED**:

- Connect the anode (longer leg) of the LED to a digital output pin on the microcontroller (e.g., Pin 9).
- Connect the cathode (shorter leg) of the LED to the ground (GND) of the microcontroller.

4. **Connect the Buzzer**:
   - Connect the positive pin of the buzzer to another digital output pin on the microcontroller (e.g., Pin 12).
   - Connect the negative pin of the buzzer to the ground (GND) of the microcontroller.
5. **Power the Microcontroller**:
   - Connect the microcontroller to a power source (USB or battery).
6. **Upload Code**:
   - Write and upload the code to the microcontroller to control the LED and buzzer based on the tilt sensor's output.

Example Code (for Arduino)

```
const int tiltPin = 8; // Pin for tilt sensor
const int ledPin = 9;  // Pin for LED
const int buzzerPin = 12; // Pin for Buzzer

void setup() {
  pinMode(tiltPin, INPUT);
  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  int tiltState = digitalRead(tiltPin); // Read the state of
the tilt sensor

  if (tiltState == LOW) { // If the tilt sensor is activated
    digitalWrite(ledPin, HIGH); // Turn on the LED
    digitalWrite(buzzerPin, HIGH); // Turn on the buzzer
  } else {
    digitalWrite(ledPin, LOW); // Turn off the LED
    digitalWrite(buzzerPin, LOW); // Turn off the buzzer
  }
  delay(100); // Short delay for stability
```

```
}
```

Conclusions

In this lesson, students learned how to use a tilt sensor in a circuit, how to control an LED and a buzzer based on the tilt sensor's output, and the principles behind tilt sensing technology. This foundational knowledge can be applied to more complex projects involving motion detection and alarms. Encourage students to experiment with different tilt angles and responses, such as creating a simple alarm system for unauthorized movement or a game that responds to tilting.

Chapter 10: Introduction to Arduino and OLED Display

Introduction

In this lesson, students will learn how to use an Arduino UNO microcontroller to control a 0.96" OLED display. They will understand the basics of microcontroller programming, circuit design, and how to display messages on an OLED screen.

Background

Arduino is an open-source electronics platform based on easy-to-use hardware and software. The Arduino UNO is a popular microcontroller board that allows students to create interactive projects. The 0.96" OLED display is a small screen that can show text and graphics, making it ideal for displaying information in projects.

Learning Objectives

By the end of this lesson, students will be able to:

1. Understand the basic components of an Arduino circuit.
2. Write and upload a simple Arduino sketch to display messages on an OLED screen.
3. Connect components on a breadboard and understand the importance of electrical connections.
4. Troubleshoot common issues in circuit design and programming.

Functionality

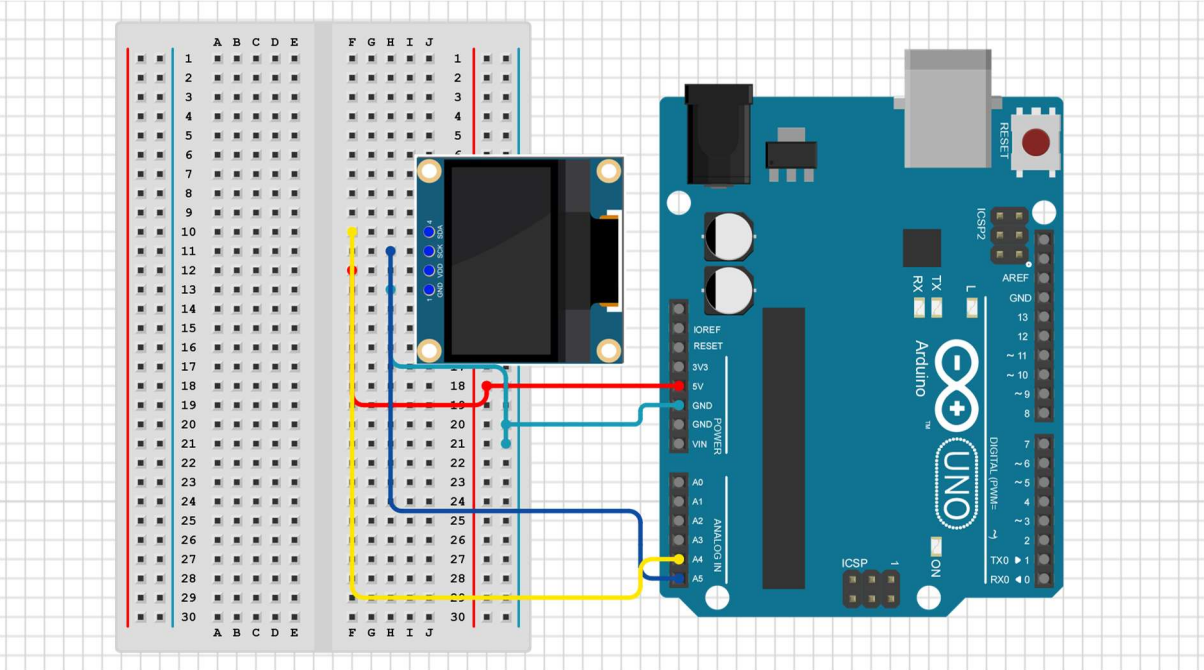
The project will display alternating messages ("Welcome" and "ZAS Robotics") on the OLED screen. The messages will change every two seconds, demonstrating how to control the display using the Arduino.

Materials Needed

- 1 x Arduino UNO
- 1 x 0.96" OLED display
- 1 x Breadboard
- Jumper wires
- USB cable for Arduino
- Computer with Arduino IDE installed

Circuit Diagram

(Replace with actual diagram)



Step-By-Step Wiring Instructions

1. **Connect the OLED Display to the Arduino:**
    - Connect the GND pin of the OLED to the GND pin of the Arduino.
    - Connect the VDD pin of the OLED to the 5V pin of the Arduino.
    - Connect the SCK pin of the OLED to the A5 pin of the Arduino (SCL).
    - Connect the SDA pin of the OLED to the A4 pin of the Arduino (SDA).
2. **Power the Arduino:**
    - Connect the Arduino to the computer using a USB cable.
3. **Open the Arduino IDE:**
    - Launch the Arduino IDE on your computer.
4. **Upload the Code:**
    - Copy the provided code into the Arduino IDE.
    - Ensure the correct board and port are selected in the IDE.
    - Click on the upload button to upload the code to the Arduino.
5. **Observe the Output:**

- Once the code is uploaded, the OLED display should show the messages "Welcome" and "ZAS Robotics" alternating every two seconds.

Sample Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Define the OLED display width and height
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA,
SCL pins)
// The I2C address for most SSD1306 OLEDs is 0x3C or 0x3D
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino
reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

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

  // Initialize OLED display with I2C address 0x3C
  // You might need to change 0x3C to 0x3D depending on your
display.
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Clear the display buffer.
  display.clearDisplay();

  // Set text size (1 is smallest)
  display.setTextSize(1);
  // Set text color (WHITE is default for monochrome OLEDs)
  display.setTextColor(SSD1306_WHITE);
  // Set cursor position (x, y)
  display.setCursor(0, 0);
  // Print text
  display.println("Hello, Arduino!");
```

```
  // Display everything on the screen
  display.display();

  delay(2000); // Pause for 2 seconds

  // Clear display and show some more text
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0, 20);
  display.println("ZAS");
  display.println("Robotics!");
  display.display();
}

void loop() {
  // Nothing to do in the loop for this simple example
}
```

Explanation of the Code

1. **Libraries**: The code includes the necessary libraries for the OLED display (Adafruit_GFX and Adafruit_SSD1306).
2. **Display Initialization**: The display is initialized with the specified width, height, and I2C address (commonly 0x3C for most 0.96-inch OLED displays).
3. **Setup Function**:
    - The display is cleared, and a welcome message is printed.
    - The display is updated to show the message.
4. **Loop Function**:
    - None

Additional Notes

- Make sure to install the Adafruit SSD1306 and GFX libraries in the Arduino IDE before uploading the code.
- If your OLED display has a different I2C address, you may need to change the address in the display.begin() function.
- You can modify the text, shapes, and colors as needed to fit your project requirements.

Conclusions

In this lesson, students learned how to set up a simple circuit using an Arduino and an OLED display. They gained hands-on experience in wiring components, writing code, and troubleshooting. This foundational knowledge will prepare them for more complex projects in the future. Encourage students to experiment with different messages or graphics on the OLED display to enhance their learning experience.

Chapter 11: Using the DHT11 Temperature and Humidity Sensor

Introduction

In this lesson, students will learn how to use the DHT11 temperature and humidity sensor to measure environmental conditions. This project will demonstrate how to interface a sensor with a microcontroller, allowing students to understand the principles of temperature and humidity measurement in electronics.

Background

The DHT11 is a low-cost digital temperature and humidity sensor that provides accurate readings of both temperature (in Celsius) and relative humidity (in percentage). It communicates with microcontrollers using a single-wire digital interface. Understanding how to use the DHT11 sensor will enhance students' skills in data acquisition and environmental monitoring.

Learning Objectives

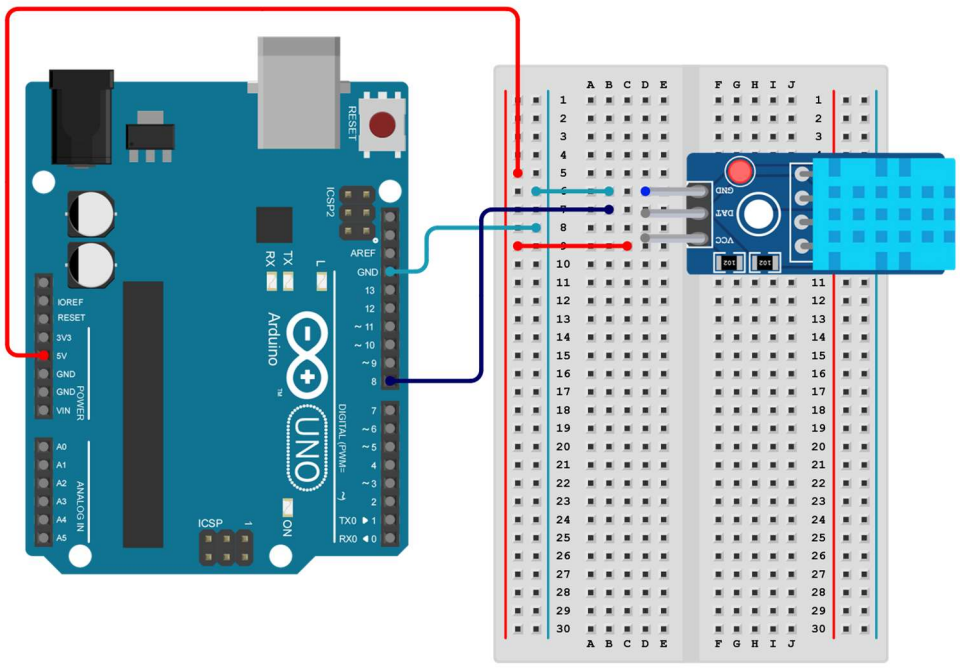
By the end of this lesson, students will be able to:

1. Identify the components of a DHT11 sensor circuit.
2. Understand how the DHT11 sensor works and its applications.
3. Construct a simple circuit using the DHT11 sensor.
4. Write a basic program to read and display temperature and humidity data from the DHT11 sensor using a microcontroller.

Functionality

The DHT11 sensor will measure the ambient temperature and humidity levels. The microcontroller will read the data from the sensor and display it on the Serial Monitor. This interaction can be used in various applications, such as weather stations, climate control systems, and environmental monitoring.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - DHT11 Temperature and Humidity Sensor
    - Breadboard and jumper wires
    - Power supply (USB or battery)
2. **Connect the DHT11 Sensor**:
    - Connect the VCC pin of the DHT11 sensor to the 5V pin on the microcontroller.
    - Connect the GND pin of the DHT11 sensor to the ground (GND) of the microcontroller.
    - Connect the DATA pin of the DHT11 sensor to a digital input pin on the microcontroller (e.g., Pin 8).
3. **Power the Microcontroller**:
    - Connect the microcontroller to a power source (USB or battery).

4. **Install Required Libraries**:
    - Before uploading the code, ensure that the DHT sensor library is installed in the Arduino IDE. You can install it via the Library Manager by searching for "DHT sensor library" by Adafruit.
5. **Upload Code**:
    - Write and upload the code to the microcontroller to read and display temperature and humidity data from the DHT11 sensor.

Example Code (for Arduino)

```
#include <DHT.h>

#define DHTPIN 8     // Pin where the DHT11 is connected
#define DHTTYPE DHT11   // Define the type of DHT sensor

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

void setup() {
  Serial.begin(9600); // Start serial communication
  dht.begin(); // Initialize the DHT sensor
}

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

  // Read temperature as Celsius
  float temperature = dht.readTemperature(true);
  // Read humidity
  float humidity = dht.readHumidity();

  // Check if any reads failed and exit early (to try again).
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Print the results to the Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C, Humidity: ");
```

```
  Serial.print(humidity);
  Serial.println(" %");
}
```

Conclusions

In this lesson, students learned how to use the DHT11 temperature and humidity sensor in a circuit, how to read and display temperature and humidity data using a microcontroller, and the principles behind environmental sensing technology. This foundational knowledge can be applied to more complex projects involving climate monitoring and control systems. Encourage students to experiment with different applications, such as creating a simple weather station or integrating the sensor with other components like displays or alarms.

Chapter 12: Using the DHT11 Temperature Sensor with an OLED Display

Introduction

In this lesson, students will learn how to use the DHT11 temperature and humidity sensor in conjunction with an OLED display to visualize environmental data. This project will demonstrate how to interface sensors and displays with a microcontroller, allowing students to understand the principles of data acquisition and visualization in electronics.

Background

The DHT11 is a low-cost digital sensor that measures temperature and humidity. It communicates with microcontrollers using a single-wire digital interface. An OLED (Organic Light Emitting Diode) display is a type of display technology that provides high contrast and low power consumption, making it ideal for displaying sensor data. By combining these two components, students will learn how to create a simple weather station that displays real-time temperature and humidity readings.

Learning Objectives

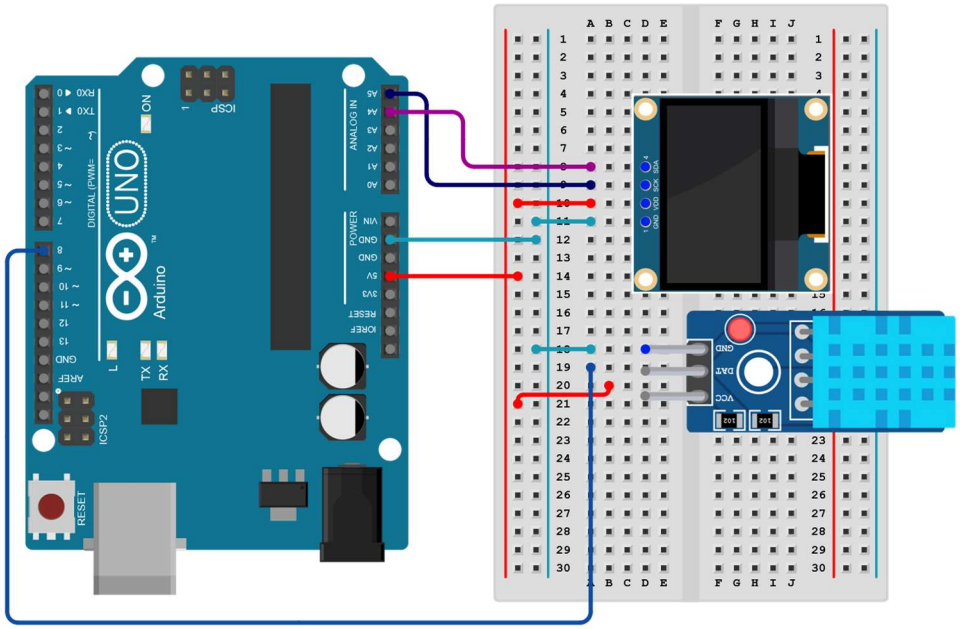
By the end of this lesson, students will be able to:

1. Identify the components of a DHT11 sensor and OLED display circuit.
2. Understand how the DHT11 sensor and OLED display work and their applications.
3. Construct a simple circuit using the DHT11 sensor and OLED display.
4. Write a basic program to read data from the DHT11 sensor and display it on the OLED screen.

Functionality

The DHT11 sensor will measure the ambient temperature and humidity levels. The microcontroller will read the data from the sensor and display it on the OLED screen. This interaction can be used in various applications, such as weather stations, climate control systems, and environmental monitoring.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Gather Components**:
    - Microcontroller (e.g., Arduino)
    - DHT11 Temperature and Humidity Sensor
    - OLED Display (e.g., 0.96" I2C OLED)
    - Breadboard and jumper wires
    - Power supply (USB or battery)
2. **Connect the DHT11 Sensor**:
    - Connect the VCC pin of the DHT11 sensor to the 5V pin on the microcontroller.
    - Connect the GND pin of the DHT11 sensor to the ground (GND) of the microcontroller.
    - Connect the DATA pin of the DHT11 sensor to a digital input pin on the microcontroller (e.g., Pin 8).
3. **Connect the OLED Display**:

- Connect the VCC pin of the OLED display to the 5V pin on the microcontroller.
- Connect the GND pin of the OLED display to the ground (GND) of the microcontroller.
- Connect the SDA pin of the OLED display to the SDA pin on the microcontroller (e.g., A4 on Arduino Uno).
- Connect the SCL pin of the OLED display to the SCL pin on the microcontroller (e.g., A5 on Arduino Uno).

4. **Install Required Libraries**:
   - Before uploading the code, ensure that the following libraries are installed in the Arduino IDE:
     - DHT sensor library (by Adafruit)
     - Adafruit GFX library
     - Adafruit SSD1306 library
5. **Upload Code**:
   - Write and upload the code to the microcontroller to read and display temperature and humidity data from the DHT11 sensor on the OLED display.

Example Code (for Arduino)

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#include <Adafruit_Sensor.h> // Required for DHT library

// DHT sensor definitions
#define DHTPIN 8     // DHT sensor connected to digital pin 8
#define DHTTYPE DHT11 // Type of DHT sensor (DHT11 or DHT22)

// OLED display definitions
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA,
SCL pins)
```

```
// The I2C address for most SSD1306 OLEDs is 0x3C or 0x3D
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino
reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

// Initialize DHT sensor
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println(F("DHT11 & OLED Test"));

  // Initialize OLED display
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Check
your I2C address (0x3C or 0x3D)
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Clear the display buffer.
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0,0);
  display.println("Initializing...");
  display.display();
  delay(1000);

  // Initialize DHT sensor
  dht.begin();
}

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

  // Read temperature and humidity.
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature(true);
  // Read temperature as Fahrenheit (is Fahrenheit is true)
```

```
// float f = dht.readTemperature(true);

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t)) {
  Serial.println(F("Failed to read from DHT sensor!"));
  display.clearDisplay();
  display.setCursor(0,0);
  display.println("Failed to read");
  display.println("DHT sensor!");
  display.display();
  return;
}

// Compute heat index in Celsius (or Fahrenheit)
// float hic = dht.computeHeatIndex(t, h, false);
// float hif = dht.computeHeatIndex(f, h);

Serial.print(F("Humidity:"));
Serial.print(h);
Serial.print(F("%  Temperature: "));
Serial.print(t);
Serial.println(F("°C"));

// Clear display before writing new values
display.clearDisplay();

// Display Humidity
display.setTextSize(2); // Larger text for main values
display.setCursor(0,0);
display.print("Hum:");
display.print(h);
display.println("%");

// Display Temperature
display.setCursor(0, 32); // Move cursor down
display.print("Temp:");
display.print(t);
display.println((char)247); // Degree symbol
display.println("C");

// Update the OLED display
display.display();
```

```
}
```

Conclusions

In this lesson, students learned how to use the DHT11 temperature and humidity sensor in a circuit, how to read and display temperature and humidity data on an OLED display, and the principles behind environmental sensing and data visualization technology. This foundational knowledge can be applied to more complex projects involving climate monitoring and control systems. Encourage students to experiment with different display formats, such as adding graphs or changing the display layout, and to explore additional sensors for more comprehensive environmental monitoring.

Chapter 13: Pollution Sensor (MQ135) with OLED

Introduction

In this lesson, students will learn how to interface a pollution sensor (MQ135) with an OLED display using a microcontroller. The MQ135 sensor is capable of detecting various gases, including ammonia, benzene, and smoke, making it a valuable tool for monitoring air quality. The OLED display will be used to visually present the sensor readings.

Background

- **MQ135 Sensor**: This is an air quality sensor that can detect a range of gases. It operates on the principle of resistive change in the presence of specific gases.
- **OLED Display**: An Organic Light Emitting Diode (OLED) display is a flat panel display technology that uses organic compounds to emit light. It is known for its high contrast and low power consumption.
- **Microcontroller**: A microcontroller (e.g., Arduino) will be used to read the sensor data and control the OLED display.

Learning Objectives

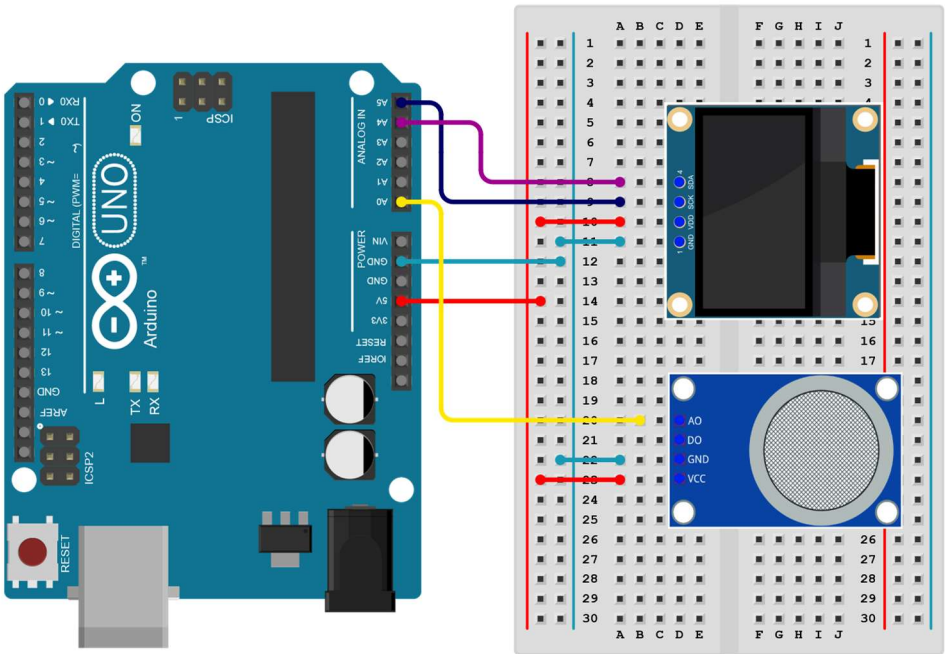
By the end of this lesson, students will be able to:

1. Understand the working principle of the MQ135 sensor.
2. Interface the MQ135 sensor with a microcontroller.
3. Display sensor readings on an OLED display.
4. Write and upload code to the microcontroller to process and display data.

Functionality

- The MQ135 sensor will measure the concentration of various gases in the air.
- The microcontroller will read the analog output from the MQ135 sensor.
- The microcontroller will process the data and send it to the OLED display.
- The OLED display will show real-time air quality readings.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Components Needed**:
    - MQ135 Sensor
    - OLED Display (e.g., 0.96" I2C OLED)
    - Microcontroller (e.g., Arduino Uno)
    - Jumper wires
    - Breadboard (optional)
2. **Wiring Instructions**:
    - Connect the **VCC** pin of the MQ135 to the **5V** pin of the microcontroller.
    - Connect the **GND** pin of the MQ135 to the **GND** pin of the microcontroller.
    - Connect the **AOUT** pin of the MQ135 to an **analog pin** (e.g., A0) on the microcontroller.
    - Connect the **VCC** pin of the OLED display to the **5V** pin of the microcontroller.

- Connect the **GND** pin of the OLED display to the **GND** pin of the microcontroller.
- Connect the **SDA** pin of the OLED display to the **SDA** pin on the microcontroller (A4 for Arduino Uno).
- Connect the **SCL** pin of the OLED display to the **SCL** pin on the microcontroller (A5 for Arduino Uno).

Sample Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

const int mq135Pin = A0; // Pin connected to the MQ135 AOUT
float sensorValue = 0;

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

  // Initialize the OLED display
  display.begin(SSD1306_I2C_ADDRESS, OLED_RESET);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.println("MQ135 Air Quality");
  display.display();
  delay(2000); // Pause for 2 seconds
}

void loop() {
  // Read the analog value from the MQ135 sensor
  sensorValue = analogRead(mq135Pin);

  // Clear the display
  display.clearDisplay();
```

```
  // Display the sensor value
  display.setCursor(0, 0);
  display.print("Sensor Value: ");
  display.println(sensorValue);

  // Optionally, you can convert the sensor value to a more
meaningful unit
  // For example, you can implement a conversion based on
calibration data

  display.display(); // Update the display
  delay(1000); // Update every second
}
```

Explanation of the Code

1. **Libraries**: The code includes the necessary libraries for the OLED display (Adafruit_GFX and Adafruit_SSD1306).
2. **Setup**: In the setup() function, the serial communication is initialized, and the OLED display is set up.
3. **Loop**: In the loop() function, the code reads the analog value from the MQ135 sensor, clears the display, and prints the sensor value on the OLED screen. The display is updated every second.

Conclusions

In this lesson, students will gain hands-on experience with sensors and displays, enhancing their understanding of environmental monitoring. They will learn how to read sensor data, process it, and present it visually, which is a crucial skill in electronics and programming. This project can be expanded further by integrating data logging or wireless communication for remote monitoring.

Additional Notes

- Ensure students understand safety precautions when working with electronic components.
- Encourage students to experiment with different environmental conditions to see how the sensor readings change.

Chapter 14: Ultrasonic Sensor

Introduction

In this lesson, students will learn how to use an ultrasonic sensor (HC-SR04) to measure distance. The ultrasonic sensor emits sound waves and measures the time it takes for the echo to return, allowing it to calculate the distance to an object. This project will help students understand the principles of distance measurement and sensor interfacing.

Background

- **Ultrasonic Sensor (HC-SR04)**: This sensor uses ultrasonic waves to measure distance. It has two main components: a transmitter that emits sound waves and a receiver that listens for the echo. The time taken for the echo to return is used to calculate the distance to the object.
- **Microcontroller**: A microcontroller (e.g., Arduino) will be used to control the ultrasonic sensor and process the data.

Learning Objectives

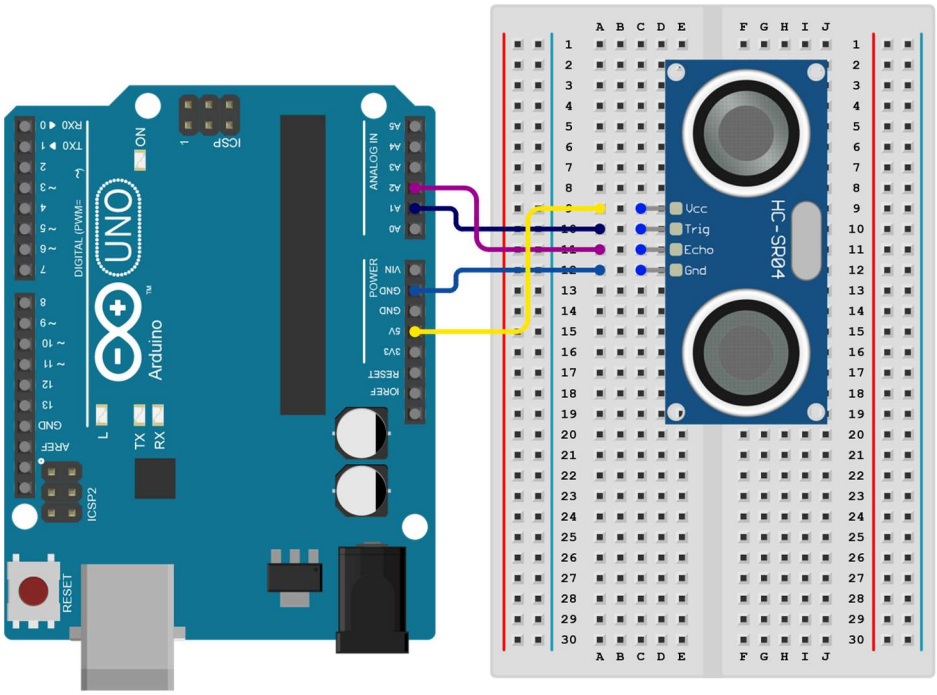
By the end of this lesson, students will be able to:

1. Understand the working principle of the ultrasonic sensor.
2. Interface the ultrasonic sensor with a microcontroller.
3. Write code to measure and display distance readings.
4. Apply the knowledge of sensors in practical applications.

Functionality

- The ultrasonic sensor will emit a sound wave and measure the time it takes for the echo to return.
- The microcontroller will calculate the distance based on the time taken for the echo to return.
- The distance will be displayed on the serial monitor or an OLED display.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Components Needed**:
    - Ultrasonic Sensor (HC-SR04)
    - Microcontroller (e.g., Arduino Uno)
    - Jumper wires
    - Breadboard (optional)
2. **Wiring Instructions**:
    - Connect the **VCC** pin of the HC-SR04 to the **5V** pin of the microcontroller.
    - Connect the **GND** pin of the HC-SR04 to the **GND** pin of the microcontroller.
    - Connect the **TRIG** pin of the HC-SR04 to a digital pin on the microcontroller (e.g., pin A1).
    - Connect the **ECHO** pin of the HC-SR04 to another digital pin on the microcontroller (e.g., pin A2).

Sample Code

```
#define TRIG_PIN A1
#define ECHO_PIN A2

void setup() {
 Serial.begin(9600);
 pinMode(TRIG_PIN, OUTPUT);
 pinMode(ECHO_PIN, INPUT);
}

void loop() {
 // Send a 10 microsecond pulse to trigger
 digitalWrite(TRIG_PIN, LOW);
 delayMicroseconds(2);
 digitalWrite(TRIG_PIN, HIGH);
 delayMicroseconds(10);
 digitalWrite(TRIG_PIN, LOW);

 // Read the echo pin
 long duration = pulseIn(ECHO_PIN, HIGH);

 // Calculate the distance in centimeters
 float distance = duration * 0.034 / 2;

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

 //delay(100);
 delay(1000); // Update every 0.5 seconds
}
```

Explanation of the Code

1. **Pin Definitions**: The code defines the pins for the TRIG and ECHO of the ultrasonic sensor.
2. **Setup**: In the setup() function, serial communication is initialized, and the TRIG pin is set as an output while the ECHO pin is set as an input.

3. **Loop**: In the loop() function, the code sends a pulse to the TRIG pin, measures the duration of the echo received on the ECHO pin, calculates the distance, and prints it to the Serial Monitor.

Conclusions

In this lesson, students will gain practical experience with ultrasonic sensors and learn how to measure distance using sound waves. This project introduces fundamental concepts in electronics, programming, and sensor technology, which can be applied in various applications such as robotics, obstacle detection, and automation. Students are encouraged to experiment with different distances and objects to see how the sensor performs in various conditions.

Chapter 15: Ultrasonic Sensor with OLED and Buzzer

Introduction

In this lesson, students will learn how to use an ultrasonic sensor (HC-SR04) in conjunction with an OLED display and a buzzer. The ultrasonic sensor will measure the distance to an object, the OLED display will show the distance readings, and the buzzer will sound an alert when the object is within a specified distance. This project combines sensor technology with visual and auditory feedback, enhancing the learning experience.

Background

- **Ultrasonic Sensor (HC-SR04)**: This sensor uses ultrasonic waves to measure distance. It emits sound waves and measures the time it takes for the echo to return, allowing it to calculate the distance to an object.
- **OLED Display**: An Organic Light Emitting Diode (OLED) display is used to visually present the distance readings.
- **Buzzer**: A simple piezo buzzer will provide auditory feedback when the measured distance falls below a certain threshold.
- **Microcontroller**: A microcontroller (e.g., Arduino) will be used to control the ultrasonic sensor, OLED display, and buzzer.

Learning Objectives

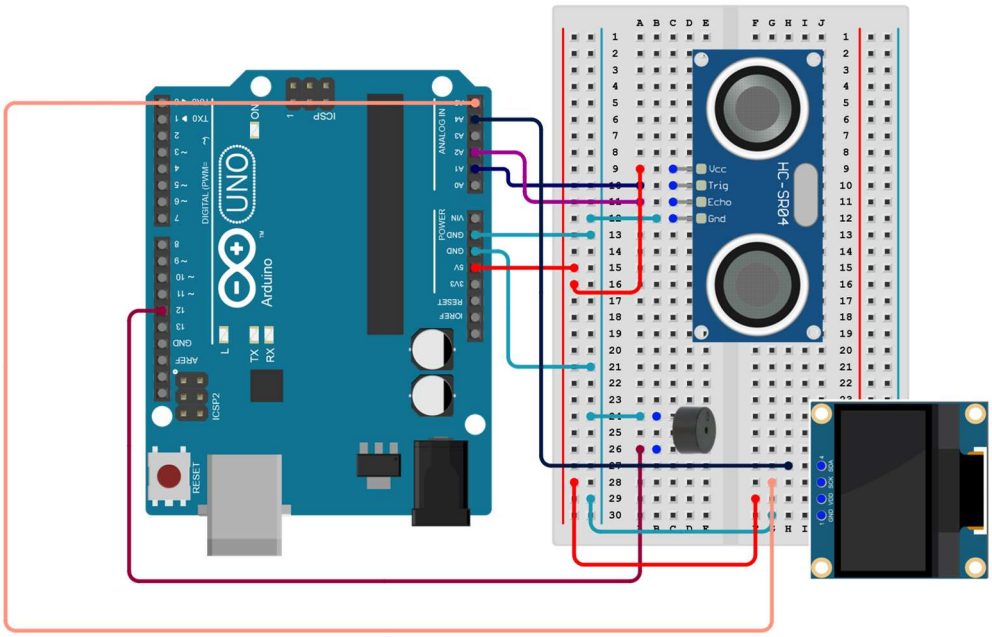
By the end of this lesson, students will be able to:

1. Understand the working principle of the ultrasonic sensor.
2. Interface the ultrasonic sensor, OLED display, and buzzer with a microcontroller.
3. Write code to measure distance, display it, and trigger a buzzer based on distance.
4. Apply knowledge of sensors and displays in practical applications.

Functionality

- The ultrasonic sensor will measure the distance to an object.
- The microcontroller will calculate the distance and display it on the OLED screen.
- If the distance is below a specified threshold, the buzzer will sound an alert.
- The OLED display will update in real-time with the distance readings.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Components Needed**:
    - Ultrasonic Sensor (HC-SR04)
    - OLED Display (e.g., 0.96" I2C OLED)
    - Buzzer (active or passive)
    - Microcontroller (e.g., Arduino Uno)
    - Jumper wires
    - Breadboard (optional)
2. **Wiring Instructions**:
    - Connect the **VCC** pin of the HC-SR04 to the **5V** pin of the microcontroller.
    - Connect the **GND** pin of the HC-SR04 to the **GND** pin of the microcontroller.
    - Connect the **TRIG** pin of the HC-SR04 to a digital pin on the microcontroller (e.g., pin A1).
    - Connect the **ECHO** pin of the HC-SR04 to another digital pin on the microcontroller (e.g., pin A2).

- Connect the **VCC** pin of the OLED display to the **5V** pin of the microcontroller.
- Connect the **GND** pin of the OLED display to the **GND** pin of the microcontroller.
- Connect the **SDA** pin of the OLED display to the **SDA** pin on the microcontroller (A4 for Arduino Uno).
- Connect the **SCL** pin of the OLED display to the **SCL** pin on the microcontroller (A5 for Arduino Uno).
- Connect one terminal of the buzzer to a digital pin on the microcontroller (e.g., pin 12) and the other terminal to **GND**.

Sample Code

```
#include <Wire.h>               // Required for I2C
communication with OLED
#include <Adafruit_GFX.h>      // Core graphics library for
OLED
#include <Adafruit_SSD1306.h> // Library for SSD1306 OLED
displays

// --- Ultrasonic Sensor Definitions ---
#define TRIG_PIN A1 // Trigger pin connected to Analog Pin A1
#define ECHO_PIN A2 // Echo pin connected to Analog Pin A2

// --- Buzzer Definition ---
#define BUZZER_PIN 12 // Buzzer connected to Digital Pin 9

// --- OLED Display Definitions ---
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA,
SCL pins)
// The I2C address for most SSD1306 OLEDs is 0x3C or 0x3D
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino
reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

// --- Global Variables ---
long duration;
```

```
float distanceCm;

void setup() {
  // --- Serial Communication Setup ---
  Serial.begin(9600);
  Serial.println("Ultrasonic Sensor with OLED and Buzzer");

  // --- Pin Mode Setup ---
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT); // Set buzzer pin as output

  // --- OLED Display Initialization ---
  // If the display doesn't begin, try changing 0x3C to 0x3D
or vice-versa
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Clear the display buffer.
  display.clearDisplay();
  display.setTextSize(1);       // Smallest text size
  display.setTextColor(SSD1306_WHITE); // White text on black
background
  display.setCursor(0,0);       // Set cursor to top-left
corner
  display.println("Initializing...");
  display.display();            // Show initial message
  delay(1000);
}

void loop() {
  // --- Ultrasonic Sensor Reading ---
  // Clear the trigPin by setting it LOW for 2 microseconds
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  // Set the trigPin HIGH for 10 microseconds to send a pulse
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
```

```
  // Read the echo pin: returns the duration of the pulse in
microseconds
  duration = pulseIn(ECHO_PIN, HIGH);

  // Calculate the distance in centimeters
  // Speed of sound is 343 meters/second or 0.0343
cm/microsecond.
  // The sound travels to the object and back, so divide by 2.
  distanceCm = duration * 0.0343 / 2;

  // --- Serial Monitor Output ---
  Serial.print("Distance: ");
  Serial.print(distanceCm);
  Serial.println(" cm");

  // --- OLED Display Output ---
  display.clearDisplay();       // Clear the display buffer
  display.setTextSize(2);       // Set text size for distance
  display.setCursor(0,0);       // Set cursor for "Distance:"
  display.print("Distance:");

  display.setCursor(0, 30);     // Move cursor down for value
  display.setTextSize(3);       // Larger text for the actual
distance
  display.print(distanceCm);
  display.print(" cm");
  display.display();            // Update the OLED display

  // --- Buzzer Control (Example: buzz if less than 20cm) ---
  if (distanceCm < 10 && distanceCm > 0) { // Check distance
and ensure it's not a false reading of 0
    tone(BUZZER_PIN, 1000); // Play a tone of 1000 Hz
  } else {
    noTone(BUZZER_PIN); // Stop the tone
  }

  // Add a small delay to avoid excessive readings and
flickering
  delay(1000); // Update every 1000 milliseconds (1 seconds)
}
```

Explanation of the Code

1. **Libraries**: The code includes the necessary libraries for the OLED display (Adafruit_GFX and Adafruit_SSD1306).
2. **Pin Definitions**: The code defines the pins for the TRIG, ECHO, and buzzer.
3. **Setup**: In the setup() function, serial communication is initialized, and the pins are configured. The OLED display is also initialized.
4. **Loop**: In the loop() function, the code sends a pulse to the TRIG pin, measures the duration of the echo received on the ECHO pin, calculates the distance, and displays it on the OLED screen. If the distance is below the specified threshold, the buzzer is activated.

Conclusions

In this lesson, students will gain hands-on experience with ultrasonic sensors, displays, and buzzers. They will learn how to measure distance, provide visual feedback on an OLED display, and trigger auditory alerts with a buzzer. This project demonstrates the integration of multiple components in a single application, reinforcing concepts in electronics, programming, and sensor technology. Students are encouraged to experiment with different distance thresholds and observe how the system responds.

Chapter 16: NeoPixel Ring with 8 Pixels

Introduction

In this lesson, students will learn how to control a NeoPixel ring with 8 pixels using a microcontroller (e.g., Arduino). NeoPixels are individually addressable RGB LEDs that can be programmed to display a wide range of colors and effects. This project will introduce students to concepts of digital lighting, color mixing, and programming.

Background

- **NeoPixel**: NeoPixels are RGB LEDs that can be controlled individually. Each LED contains a red, green, and blue diode, allowing for a full spectrum of colors. They are controlled using a single data line, making them easy to integrate into various projects.
- **Microcontroller**: A microcontroller (e.g., Arduino) will be used to send data to the NeoPixel ring, controlling the color and brightness of each LED.

Learning Objectives

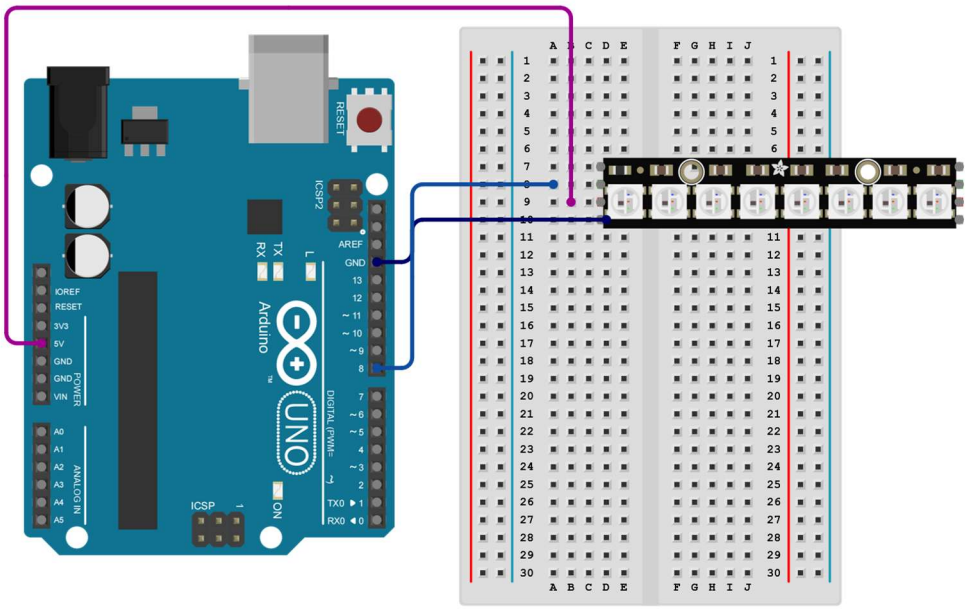
By the end of this lesson, students will be able to:

1. Understand the working principle of NeoPixels.
2. Interface a NeoPixel ring with a microcontroller.
3. Write code to control the color and effects of the NeoPixel ring.
4. Apply knowledge of programming and electronics in practical applications.

Functionality

- The NeoPixel ring will display various colors and effects based on the programmed code.
- Students will learn how to change colors, create patterns, and implement simple animations.

Circuit Diagram



Step-by-Step Wiring Instructions

1. **Components Needed**:
    - NeoPixel Ring (8 pixels)
    - Microcontroller (e.g., Arduino Uno)
    - 470 Ohm resistor (optional, for data line protection)
    - 1000 µF capacitor (optional, for power stabilization)
    - Jumper wires
    - Breadboard (optional)
2. **Wiring Instructions**:
    - Connect the **VCC** pin of the NeoPixel ring to the **5V** pin of the microcontroller.
    - Connect the **GND** pin of the NeoPixel ring to the **GND** pin of the microcontroller.
    - Connect the **DIN** pin of the NeoPixel ring to **Pin 3** on the microcontroller. (Optionally, place a 470 Ohm resistor in series with the data line for protection.)

- If using a capacitor, connect it across the VCC and GND pins of the NeoPixel ring to help stabilize the power supply.

Sample Code

```
#include <Adafruit_NeoPixel.h>

#define PIN 8 // Pin connected to the NeoPixel ring
#define NUMPIXELS 8 // Number of pixels in the NeoPixel ring

Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB +
NEO_KHZ800);

void setup() {
  pixels.begin(); // Initialize the NeoPixel ring
}

void loop() {
  // Cycle through colors
  for (int i = 0; i < NUMPIXELS; i++) {
    pixels.setPixelColor(i, pixels.Color(255, 0, 0)); // Red
    pixels.show();
    delay(500);
    pixels.setPixelColor(i, pixels.Color(0, 255, 0)); // Green
    pixels.show();
    delay(500);
    pixels.setPixelColor(i, pixels.Color(0, 0, 255)); // Blue
    pixels.show();
    delay(500);
    pixels.setPixelColor(i, pixels.Color(0, 0, 0)); // Off
  }

  // Rainbow effect
  for (int j = 0; j < 256; j++) {
    for (int i = 0; i < NUMPIXELS; i++) {
      pixels.setPixelColor(i, Wheel((i + j) & 255));
    }
    pixels.show();
    delay(50);
  }
}

// Function to generate rainbow colors across 0-255 positions
```

```
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if (WheelPos < 85) {
    return pixels.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  } else if (WheelPos < 170) {
    WheelPos -= 85;
    return pixels.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  } else {
    WheelPos -= 170;
    return pixels.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  }
}
```

Explanation of the Code

1. **Library**: The code includes the Adafruit NeoPixel library, which simplifies controlling the NeoPixel ring.
2. **Pin Definitions**: The code defines the pin connected to the NeoPixel ring and the number of pixels.
3. **Setup**: In the setup() function, the NeoPixel ring is initialized.
4. **Loop**: In the loop() function, the code cycles through red, green, and blue colors for each pixel, creating a simple animation. It also includes a rainbow effect that cycles through colors.

Conclusions

In this lesson, students will gain hands-on experience with NeoPixels and learn how to control them using a microcontroller. They will understand the principles of digital lighting, color mixing, and programming. This project can be expanded further by encouraging students to create their own patterns and effects, fostering creativity and problem-solving skills in electronics and programming.