ELECROW

All-in-one Starter Kit for Arduino User Manual

- STEAM Education
- Open-source Hardware

20+
Lessons

15
Built-in modules

www.elecrow.com

Table of Contents

Introduction

Welcome to read the user manual of All-in-one Starter Kit for Arduino. Let's start learning about the All-in-one Starter Kit for Arduino development board and how to use the sensors now!

Don't worry, this development board comes with 21 easy-to-follow, fun and inspiring lessons that take you step by step through your knowledge. Here, you will be taken to familiarize yourself with the electronic modules, exercise your logical thinking, enhance your creative ideas, and program the functions of these electronic modules.

From the very beginning to understand the installation of Arduino software, then introduce the Arduino development board and various sensors, then learn the programming functions of these sensors and the programming language used, and finally how to realize the specific applications of these sensors. Each step of the explanation is very clear, even if you are a beginner, you can quickly master Arduino programming.

The all-in-one Starter Kit for Arduino product offers 15 electronic modules, each module has its own unique features and functionality, designed for beginners and ideal for getting started. In short, by learning this development board, you will learn the basics and principles of sensors, understand important concepts such as digital and analog signals, analog-to-digital conversion, programming logic, and learn how to use some of the complex electronic modules. Most importantly, you will start learning Arduino programming, which will help improve your logical thinking skills.

In the programming section, we will use Arduino software for programming. Arduino platform is one of the most popular open source platforms and easy to use. It contains global hardware resources including hardware (various models of Arduino boards) and software (Arduino IDE), which is one of the best choices for programming learning.

Getting Started

Installing Arduino IDE

Download Arduino in Windows system

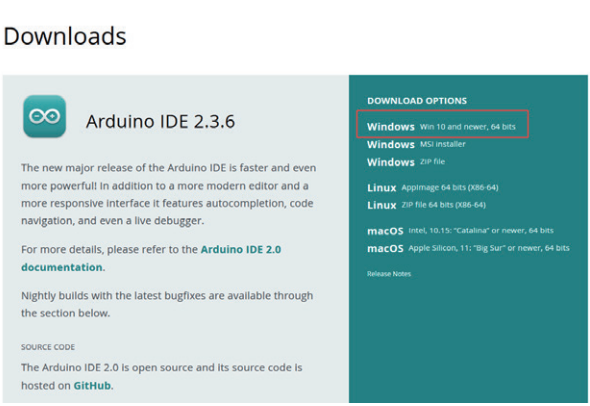
• STEP 1:

Login to Arduino official website, download Arduino.

Arduino official website: https://www.arduino.cc/en/software/

• STEP 2:

Select your computer's corresponding system to download, such as Window system.

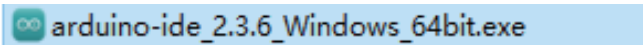


• STEP 3:

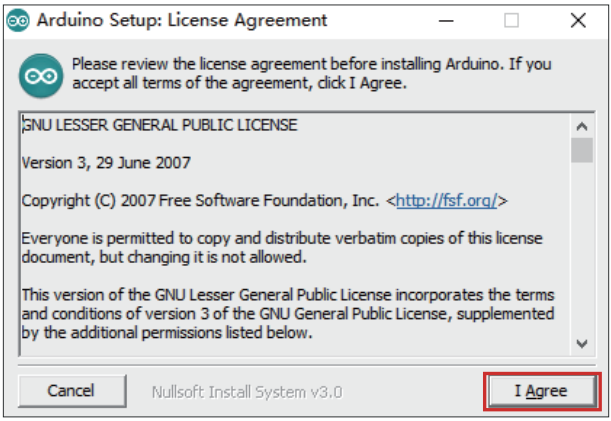
Click ***JUST DOWNLOAD*** and select the save location to start the download.

• STEP 4:

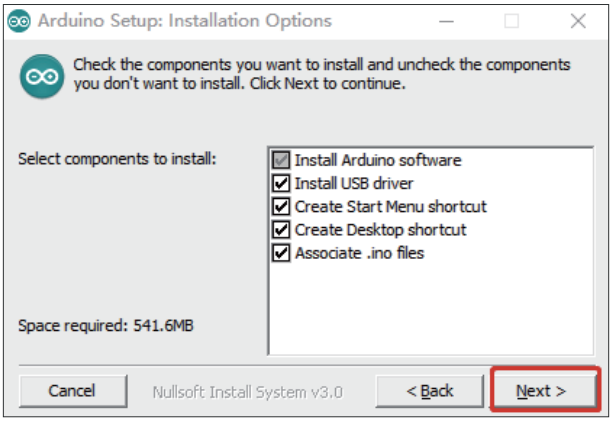
1. When installing Arduino, please locate the executable file with the .exe extension within the folder where you previously saved, which is the Arduino installation package.



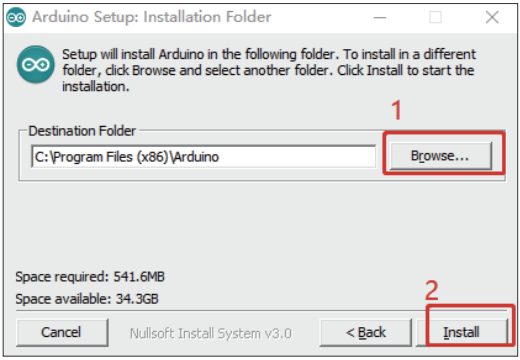
2. After double-clicking the installation package, this page will appear. Click on '***I Agree***'.

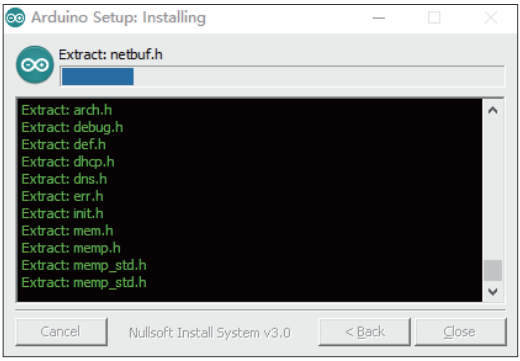


3. Check all options by default and click ***Next***.

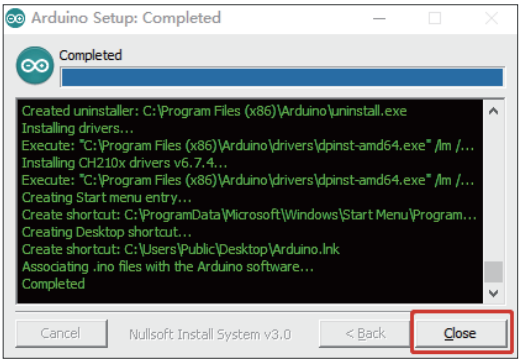


4. Click on '***Browse***' to select the installation location, it is recommended to install it on any drive other than the C: drive. Then click '***Install*** '.

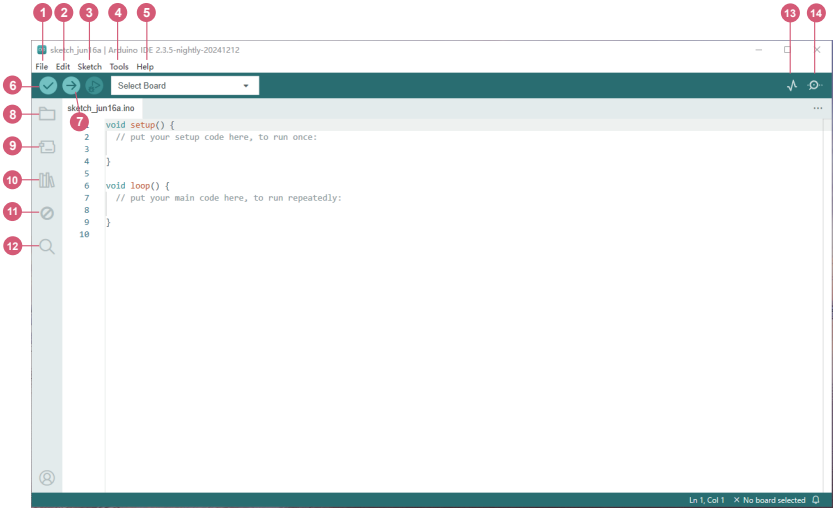




5. Installation Complete,click '***Close***'.



Arduino IDE Introduction



1 Flie: This lets you create, open, save, and manage sketch files, access sample code, adjust editor preferences, export compiled files, and exit the Arduino IDE—making it easy to handle project files and configure your workspace.

2 Edit: This is for editing your code, including undo, redo, cut, copy, paste, find and replace, select all, as well as commenting and uncommenting code—helping you modify your program quickly and efficiently.

3 Sketch: This handles compiling and uploading your code, allowing you to verify syntax, upload programs to your board, manage libraries, and export build results—streamlining your development and debugging process.

4 Tools: Choose your board model, serial port, and programmer, open the serial monitor and plotter, manage libraries, and check board details—essential for hardware setup and debugging.

5 **Help:** Provide official Arduino reference materials, FAQs, troubleshooting guides, and software version information to help users learn, use, and resolve issues they may encounter during development.

6 **Verify:** Compile the Arduino code to check for syntax errors and issues without uploading it, ensuring the code is correct and executable.

7 **Upload:** Upload the compiled code to the Arduino board to run and test the program on the actual hardware.

8 **Sketchbook:** Used for managing and quickly accessing all saved Arduino sketches, making it easy for users to open, edit, and organize their project code.

9 **Boards manager:** Used to install, update, and manage various Arduino board support packages, extending the IDE's compatibility with different hardware.

10 **Library manager:** Used to search for, install, and manage Arduino libraries, helping users easily integrate various functional modules and streamline the development process.

11 **Debug:** Used to assist with code debugging by printing messages and errors through the serial port, helping identify issues in the program and improving development efficiency.

12 **Search:** The Search feature allows quick find-and-replace within the code, making it easier to locate and edit specific content and boosting editing efficiency.

13 **Serial Plotter:** Used to plot numerical data sent from the Arduino board via the serial port in real time, helping users visually analyze sensor readings and variable changes.

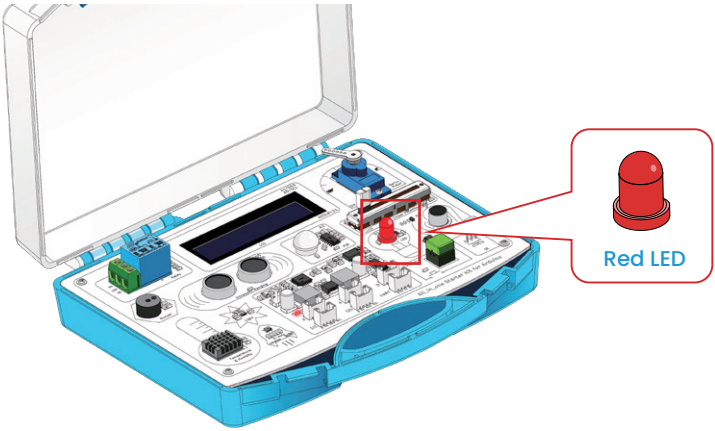
14 **Serial Monitor:** Used for serial communication with the Arduino board, allowing real-time sending and displaying of text to facilitate debugging and monitoring program status.

Lesson 1 - LED Control

Introduction

In this lesson, we will focus on the programming practice of LED control. By writing code to configure pins and design logical sequences, we will achieve alternating LED on/off behavior at fixed intervals. Through this process, you will not only learn fundamental methods of hardware control but also gain a deeper understanding of time management and loop logic in embedded programming, laying a solid foundation for developing more complex projects in the future.

Hardware Used in This Lesson:

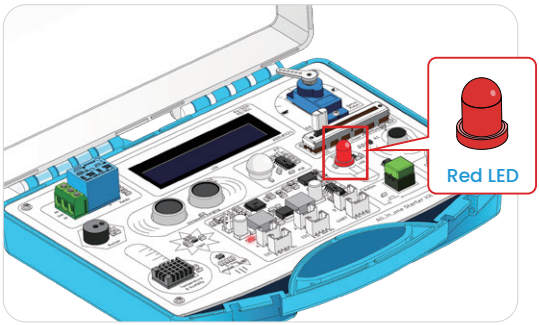


Working Principle of LED

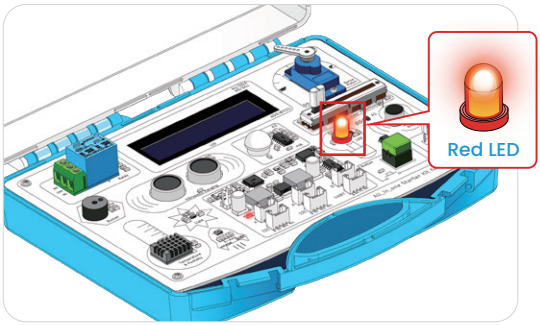
The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region and holes from the P-type region recombine near the junction. During this recombination process, electrons transition from a higher energy level to a lower one, releasing the excess energy in the form of photons—producing light. The color (wavelength) of the emitted light is determined by the energy bandgap of the semiconductor material. This is a direct application of the phenomenon known as electroluminescence.

Operation Effect Diagram

Before Running



After Operation



Once the program runs successfully, you will see the LED on the All-in-one Starter Kit for Arduino blinking—turning on for one second and then off for one second in a continuous loop.

Key Explanations

1. Variable Definition

```
int LedPin = 10;
```

- An integer variable named LedPin is defined with a value of 10, indicating that the LED is connected to pin 10.

2. setup() Function

```
void setup() {
  pinMode(LedPin, OUTPUT);
}
```

- The setup() function is executed once when the Arduino board starts up or is reset.

  The statement pinMode(LedPin, OUTPUT) configures pin 10 as an output, allowing the Arduino to send voltage to this pin in order to control the LED's state (on or off).

3. loop() Function

```
void loop() {
  digitalWrite(LedPin, HIGH);
  delay(1000);
  digitalWrite(LedPin, LOW);
  delay(1000);
}
```

- The loop() function runs repeatedly after setup() has finished. It performs the following operations:
  - digitalWrite(LedPin, HIGH): Sets pin 10 to a high voltage level (5V or 3.3V), turning the LED on.
  - delay(1000): Pauses the program for 1000 milliseconds (1 second) while the LED remains lit.
  - digitalWrite(LedPin, LOW): Sets pin 10 to a low voltage level (0V), turning the LED off.
  - delay(1000): Pauses again for 1 second while the LED remains off.

Complete Code

Click the link below to open the complete code:

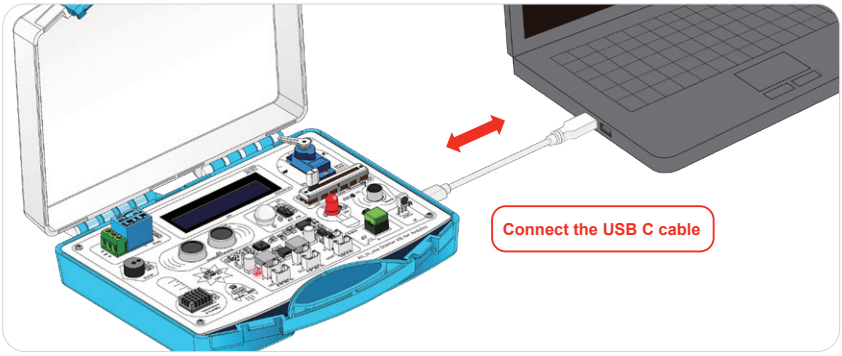
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After you finish learning the above code, you can adjust the functions, such as modifying the blinking interval time or adjusting the on-off mode of the bulb.

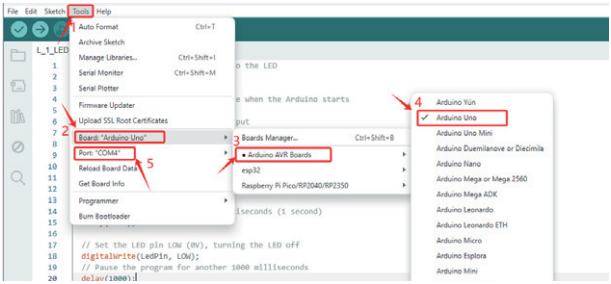
Uploading the Code

The code explanation is now complete. Next, we need to upload the code to the All-in-one Starter Kit for Arduino so we can observe the hardware functions in action.

1. First, connect the All-in-one Starter Kit for Arduino to your computer using a USB cable.

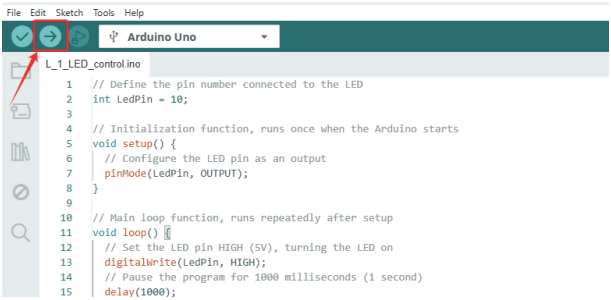


2. Open the Arduino IDE, click on “Tools,” and select the correct board option: “Arduino Uno.” Also, choose the appropriate port number.

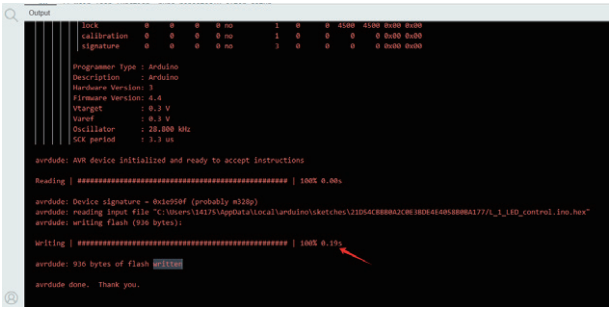


(The Arduino Uno board option is built into the Arduino IDE and will be visible by default.)
(In this example, the port is COM4, but yours may be different.)

3. Finally, click the “Upload” button.



4. Once the upload is complete, you will see a “Upload successful” message displayed in the Arduino IDE.



5. Now, you should be able to see the intended functionality demonstrated on your All-in-one Starter Kit for Arduino.

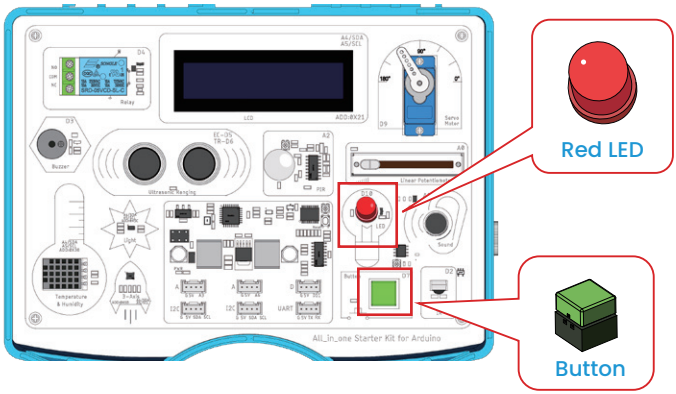
Lesson 2 - LED Control

Introduction

In this lesson, we will use the All-in-one Starter Kit for Arduino to learn how to control an LED using button input. The final goal is to achieve the following functionality: when the button is pressed, it outputs a HIGH signal; when the button is released, it outputs a LOW signal.

In this lesson, we will use the button to control the LED's state: the LED will turn on when the button is pressed and turn off when the button is released.

Hardware Used in This Lesson:



Working Principle of Button Control

A button generates signals through mechanical contacts or capacitive sensing. Mechanical buttons rely on a metal dome or spring mechanism to create HIGH or LOW logic levels when pressed or released, often requiring a debounce circuit to eliminate signal fluctuations. Capacitive buttons detect changes in capacitance between electrodes. The microcontroller (MCU) scans matrix buttons by reading the voltage levels of rows and columns, or by reading ADC values, to determine which button is pressed. The system distinguishes between short and long presses through software-based timing.

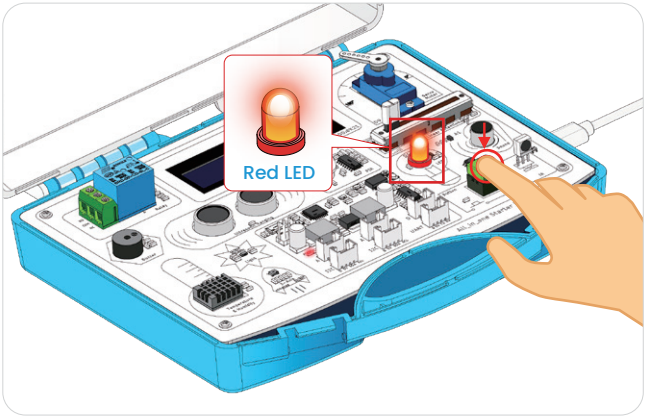
Working Principle of an LED

The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from

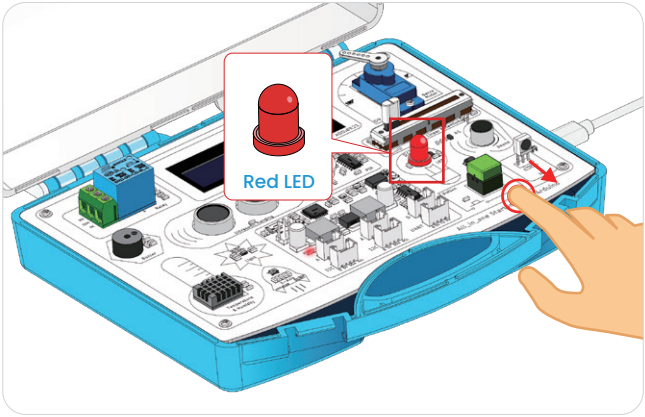
a higher to a lower energy level, releasing excess energy in the form of photons—producing visible light. The color (wavelength) of the light depends on the bandgap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

• Button Pressed:



• Button Released:



When you press and hold the button, the LED remains on. When you release the button, the LED turns off.

Key Explanations

1. Variable Definition

```
int buttonPin = 7;
int LedPin = 10;
```

- **Two variables are defined:**
  - buttonPin — the pin connected to the button (digital pin 7)
  - LedPin — the pin connected to the LED (digital pin 10)

2. setup() Function

```
void setup() {
  pinMode(LedPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}
```

- The setup() function runs once when the Arduino starts up or is reset.

  In this function, the two pins are initialized:

  - pinMode(LedPin, OUTPUT): sets pin 10 as an output to control the LED.
  - pinMode(buttonPin, INPUT): sets pin 7 as an input to read the button's state.

3. loop() Function

```
void loop() {
  if (digitalRead(buttonPin))
    digitalWrite(LedPin, LOW);
  else
    digitalWrite(LedPin, HIGH);
  delay(100);
}
```

- **Read Button State**

  Uses digitalRead(buttonPin) to check the voltage level of the button pin.

  - If you've set pinMode(buttonPin, INPUT_PULLUP) (common practice for buttons without external resistors):
    - **Button NOT pressed:** The pin stays at HIGH (pulled up to 5V/3.3V).
    - **Button pressed:** The pin connects to GND, so it reads LOW.

- **Control LED Based on Button State**

  The LED behavior is inverted relative to the button's physical state (because of INPUT_PULLUP):

  - When the button is **pressed** (circuit connects to GND):

    digitalRead(buttonPin) returns LOW → Execute digitalWrite(ledPin, HIGH) to turn the LED ON.

- When the button is **not pressed** (circuit open):

  digitalRead(buttonPin) returns HIGH → Execute digitalWrite(ledPin, LOW) to turn the LED OFF.

- **Debounce & Timing**

  delay(100) pauses the program for 100 milliseconds. This helps:

  - **Debounce the button:** Fixes “jitter” (false signals from a button bouncing during a press).
  - **Stabilize behavior:** Ensures the LED doesn’t flicker unpredictably, but note—this simple delay isn’t the most advanced debounce method (libraries like Bounce2 are better for complex cases).

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, you can modify the functionality—for example, change it so that pressing the button turns the LED on, and pressing it again turns the LED off. Alternatively, after learning more advanced concepts later, you can use button inputs to control other hardware components.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so that we can see the hardware functions in action. The code upload steps for this lesson are the same as in Lesson 1—please refer to the first lesson for detailed instructions.

After a successful upload, you will be able to use the button on the All-in-one Starter Kit for Arduino to control the LED.

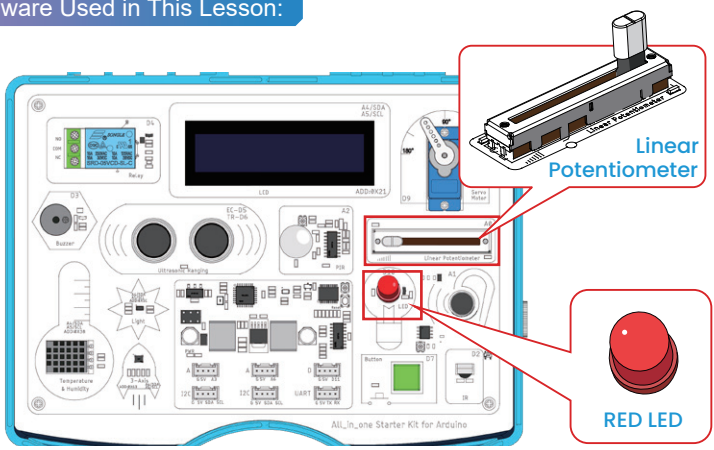
Functionality: When you press and hold the button, the LED stays on; when you release the button, the LED turns off. If the LED does not work as expected, please ensure the program is running correctly.

Lesson 3 - Breathing Led

Introduction

In this lesson, we will use a potentiometer with a maximum resistance of 10kΩ to create a breathing LED effect. When you turn the potentiometer knob from left to right, its output voltage varies between 0V and 5V (VCC). We will use this varying voltage to adjust the LED brightness and achieve the breathing light effect!

Hardware Used in This Lesson:



Working Principle of a Slide Potentiometer

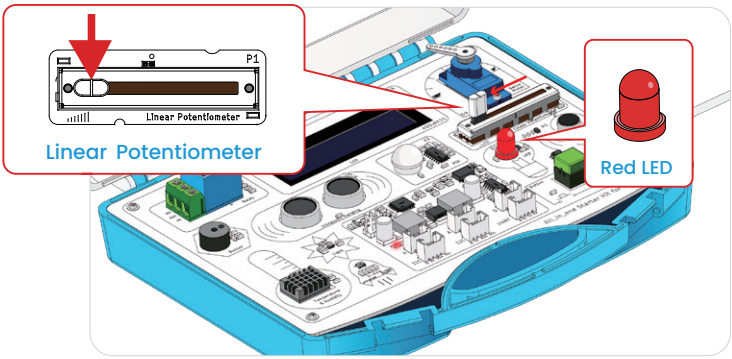
A slide potentiometer adjusts resistance by changing the effective contact length on its resistive element. As the sliding contact (wiper) moves along the resistive track, the length of the resistive material in the current path varies, resulting in a change in total resistance. In a linear potentiometer, resistance changes proportionally with the slider position, while in a logarithmic type, it follows a nonlinear curve. The structure must ensure reliable contact to avoid signal jitter or noise.

Working Principle of an LED

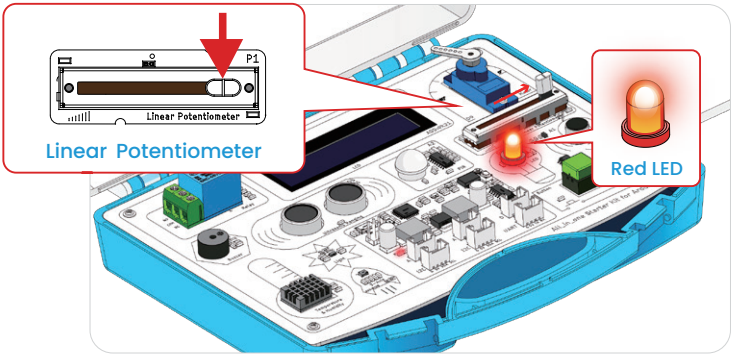
The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from a higher to a lower energy level, releasing excess energy in the form of photons—producing visible light. The color (wavelength) of the light depends on the bandgap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

• **Slide Potentiometer at the Far Left:**



• **Slide Potentiometer at the Far Right:**



As the potentiometer slider moves, the brightness of the LED changes accordingly. When you slide it all the way to the right (maximum position), the LED reaches its brightest state; when you slide it all the way to the left (minimum position), the LED turns off.

Key Explanations

1. Global Variables

```
int LinearPin = A0;
int LedPin = 10;
```

Use int to define pin numbers:

- The slide potentiometer is connected to analog pin A0 (analog input 0).
- The LED is connected to digital pin 10.

2. setup() Function

```
void setup() {
  Serial.begin(115200);
  Serial.println("Backpack init'd.");
  pinMode(LedPin, OUTPUT);
  pinMode(LinearPin, INPUT);
}
```

- **Serial Communication:** Establishes USB communication with the computer for debugging output.

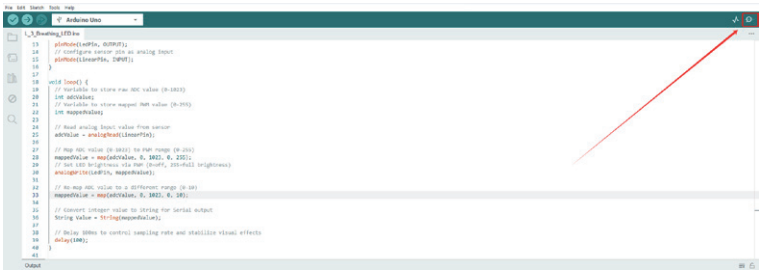
Pin Mode Configuration:

• OUTPUT: Allows the Arduino to output voltage to the LED.

• INPUT: Allows the Arduino to read voltage from the sensor (potentiometer).

***Note:* To open the Serial Monitor and view relevant data:**

1. Click the Serial Monitor button in the top-right corner of the Arduino IDE.



2. Set the baud rate to match the value specified in your code.

3. loop() Function

```
void loop() {
  int adcValue;
  int mappedValue;
  adcValue = analogRead(LinearPin);
  mappedValue = map(adcValue, 0, 1023, 0, 255);
  analogWrite(LedPin, mappedValue);
  mappedValue = map(adcValue, 0, 1023, 0, 10);
  String Value = String(mappedValue);
  delay(100);
}
```

Core logic:

- Read sensor value: analogRead(LinearPin) returns an integer from 0 to 1023.
- Map value range:
  - The map() function linearly maps the input range (fromLow to fromHigh) to the output range (toLow to toHigh).
  - For example, map(512, 0, 1023, 0, 255) returns 127 (512 is roughly half of 1023, corresponding to half of 255).
- PWM output: analogWrite(LedPin, mappedValue) adjusts the LED brightness using PWM:
  - mappedValue= 0: LED is completely off.
  - mappedValue= 255: LED is at maximum brightness.
  - Intermediate values adjust brightness proportionally (e.g., 127 is about 50% brightness).
- Delay control: delay(100) makes the loop run every 100 milliseconds to avoid excessive reading frequency.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, you can customize the functionality—for example, use the potentiometer to control additional hardware components. By combining multiple controls, you can deepen your understanding and create more complex interactive projects.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so that we can see the hardware functions in action.

The code upload steps for this lesson are the same as in Lesson 1—please refer to the first lesson for detailed instructions.

After a successful upload, you will be able to use the slide potentiometer on the All-in-one Starter Kit for Arduino to control the LED brightness.

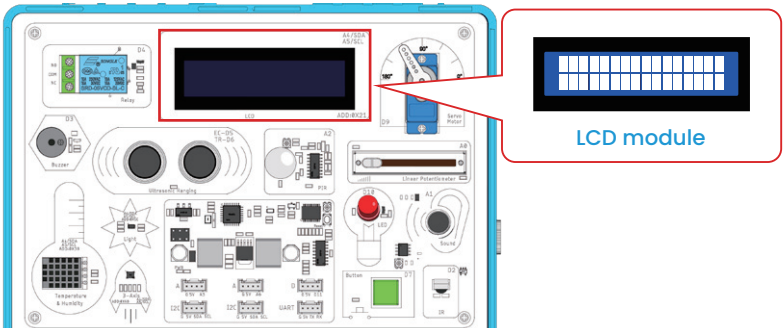
Functionality: As you move the potentiometer slider, the LED brightness will change accordingly. When slid all the way to the right (maximum position), the LED will be at its brightest; when slid all the way to the left (minimum position), the LED will turn off. If the LED does not work as expected, please ensure the program is running correctly.

Lesson 4 - LCD Display

Introduction

In this lesson, we will use the LCD1602 module on the development board to implement text display functionality. This module communicates with the board via the I2C protocol, requiring only two signal lines (SDA and SCL) for data transmission, which simplifies wiring and reduces power consumption. The LCD supports a 16-column by 2-row character display and can show letters, numbers, and symbols using standard ASCII codes. With its dedicated driver library, it enables features such as cursor positioning, screen clearing, and backlight control. Its intuitive interface and ease of operation make it ideal for information visualization in embedded systems.

Hardware Used in This Lesson:



Working Principle of the LCD Screen

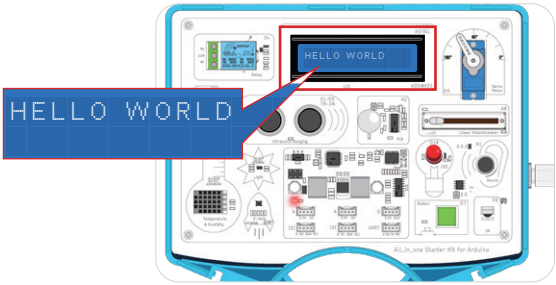
The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices. Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

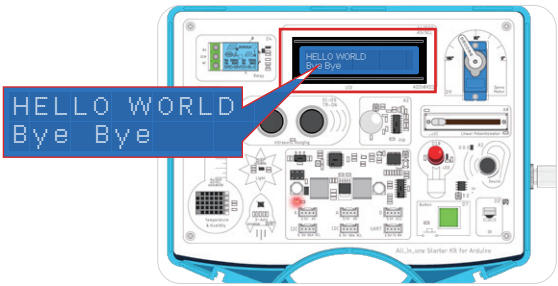
The backlight module (usually LEDs) provides illumination to ensure clear visibility even in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Operation Effect Diagram

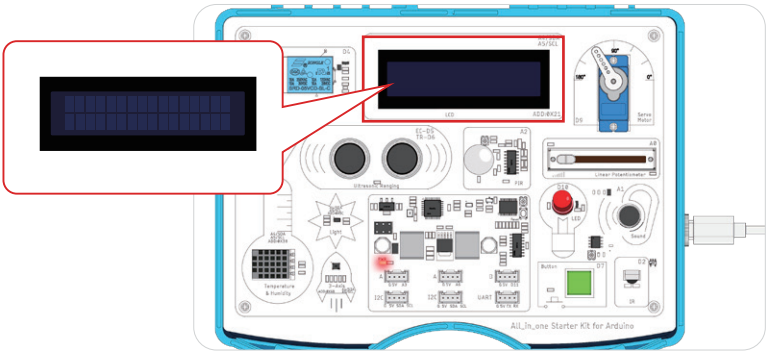
First,
display “HELLO WORLD”:



Second,
display “Bye Bye”:



Finally, turn off the LCD screen:



Once running successfully, you will see the LCD light up and display “HELLO WORLD” on the first line. After one second, “Bye Bye” will appear on the second line. Another second later, the LCD will clear the screen and turn off. If the LCD does not work as expected, please ensure the program is running correctly.

Key Explanations

1. Library and Object Initialization

```
#include "Adafruit_LiquidCrystal.h"
Adafruit_LiquidCrystal lcd(1);
```

- The Adafruit_LiquidCrystal library supports multiple LCD interfaces, including parallel, I2C, and SPI.

  The parameter 1 in lcd(1) represents the I2C bus number or channel ID being used.

2. setup() Function

```
void setup() {
  Serial.begin(115200);
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  Serial.println("Backpack init'd.");
  lcd.setCursor(0, 0);
  lcd.print("HELLO WORLD");
  delay(1000);
```

```
  lcd.setCursor(0, 1);
  lcd.print("Bye Bye");
  delay(1000);
  lcd.clear();
  lcd.setBacklight(0);
}
```

- **nitialization Process:**
  - Use lcd.begin(16, 2) to configure the LCD for 16 columns and 2 rows. This initializes the LCD and sets the display dimensions.
  - Use a while loop to check if the initialization is successful; if it fails, prompt the user to check the wiring.
- **Display Control:**
  - lcd.setCursor(col, row): Sets the cursor position, where col (column) and row (row) both start from 0.
  - lcd.print(): Prints a string at the current cursor position. Supports ASCII characters such as letters, numbers, and symbols.
- **Backlight Control:**
  - lcd.clear(): Clears the screen and resets the cursor to the top-left corner (0,0).
  - lcd.setBacklight(0): Turns off the backlight (0 = off, non-zero = on).

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, you can customize the functionality—for example, by integrating other hardware discussed later, you can display sensor data collected from various modules directly on the LCD screen. This will help you create more interactive and informative embedded systems.

Uploading the Code

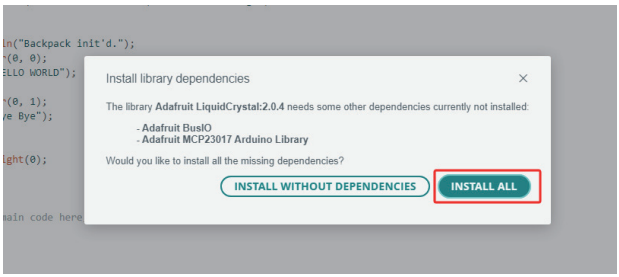
The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

Since this lesson uses an LCD screen that requires additional library files for proper operation, you need to install the necessary libraries before uploading the code.

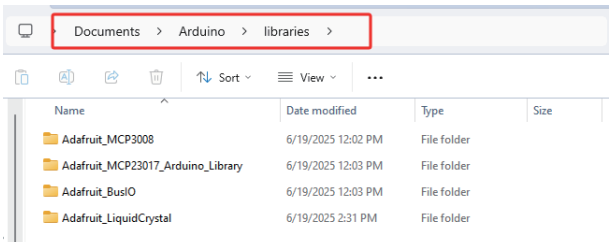
Library Installation Steps:

Follow the illustrated steps to download the Adafruit_LiquidCrystal library (version 2.0.4), which is needed to drive the LCD screen.

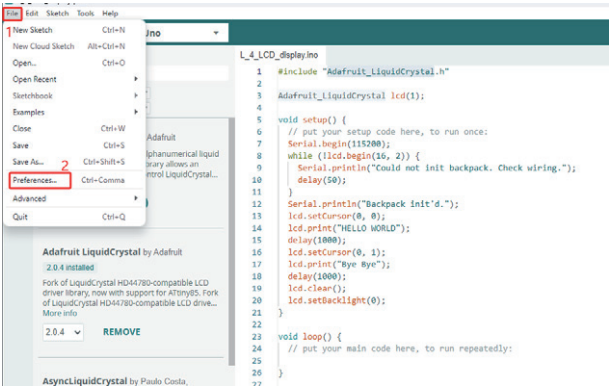
When downloading, also ensure that all dependent libraries are installed alongside it.

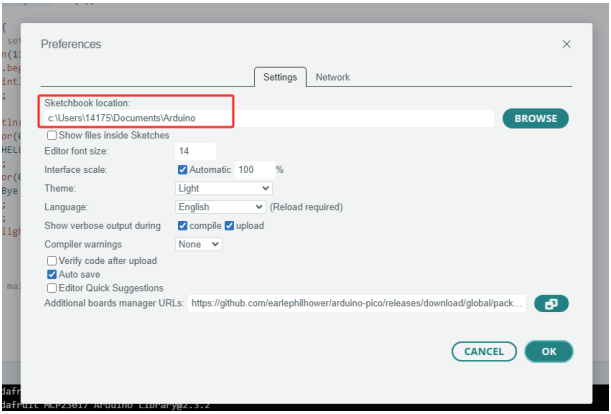


After installation, you can find the downloaded libraries in the directory set by the Arduino IDE.



(The download path is determined by the path set in the Arduino IDE.)





With the libraries installed, you can now upload the code.

The code upload process for this lesson is the same as in Lesson 1 — please refer to that for detailed instructions.

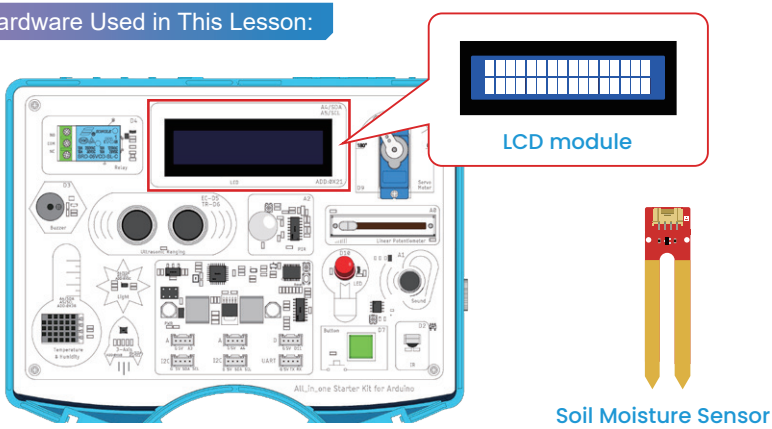
Expected Result:
After a successful upload, the LCD on the All-in-one Starter Kit for Arduino will light up, displaying “HELLO WORLD” on the first line. After one second, “Bye Bye” will appear on the second line. Another second later, the LCD will clear the display and turn off.
If the LCD does not behave as expected, please double-check that the libraries are properly installed and the program is running correctly.

Lesson 5 - Moisture Monitor

Introduction

In this lesson, we will use an LCD module together with a soil moisture sensor to monitor soil moisture levels. When the sensor is inserted into the soil, the LCD will continuously display the moisture readings in real time. These values can be used to assess the condition of the soil.

Hardware Used in This Lesson:



Working Principle of the LCD Screen

The LCD1602 screen (16×2 character display) operates based on the electro-optical effect of liquid crystals. By applying an electric field, the orientation of the liquid crystal molecules is altered, resulting in visible display changes. Internally, the module consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller (MCU) and, via the driver circuits, applies electrical signals to the segment and common electrodes of the LCD. Under the influence of the electric field, the liquid crystal molecules twist or align in specific ways, changing how much light passes through. This creates light or dark pixels, which combine to form characters or symbols.

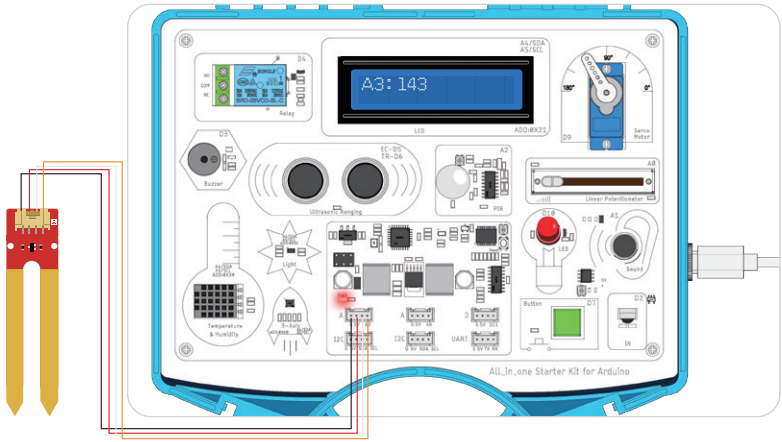
The backlight module (usually an LED) provides illumination to ensure visibility in low-light conditions. Data transmission to the LCD is done via parallel or serial interfaces (such as I2C or SPI). The MCU sends commands (e.g., to set display mode or cursor position) and content (ASCII character codes) according to protocol, and the controller processes these to light up the corresponding pixels, rendering text in the 16-column by 2-row display area.

Working Principle of the Soil Moisture Sensor

A soil moisture sensor determines water content by detecting changes in the soil's physical or electrical properties. The core principle is based on the influence of moisture on the soil's conductivity and dielectric constant. For example, resistive sensors measure moisture by detecting resistance changes between metal electrodes—higher moisture leads to lower resistance. Capacitive sensors detect variations in dielectric constant, which alter the sensor's capacitance. More advanced methods such as TDR (Time Domain Reflectometry) and FDR (Frequency Domain Reflectometry) calculate moisture levels based on the propagation characteristics of electromagnetic signals in the soil. These physical changes are ultimately converted into electrical signals, providing quantitative moisture data for applications such as agricultural irrigation and environmental monitoring.

Operation Effect Diagram

When the soil moisture sensor is properly connected to the designated A3 interface, the system will begin collecting data from the sensor and display it on the LCD screen.



You should observe the LCD continuously showing the values obtained from the soil moisture sensor.If this does not occur, please ensure that the program is running correctly.

Key Explanations

1. Global Variables

```
#include "Adafruit_LiquidCrystal.h"
Adafruit_LiquidCrystal lcd(1);
int sensorPin1 = A3;
int Pin1Value = 0;
```

- **LCD Initialization:** The LCD is controlled via the I2C interface, and it's important to ensure that the I2C address matches the one used by the I2C adapter module.
- **Sensor Pin:** Analog pin A3 is used to read the analog signal from the soil moisture sensor.

2. Global Variables

```
void setup() {
  Serial.begin(115200);
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  Serial.println("Backpack init'd.");
  pinMode(sensorPin1, INPUT);
}
```

- **LCD Initialization:** The function lcd.begin(16, 2) sets the LCD to 16×2 character mode. A while loop is used to ensure the initialization is successful before proceeding.
- **Sensor Configuration:** pinMode(sensorPin1, INPUT) declares A3 as an input pin for reading sensor data.

Note: To learn how to set the baud rate and view Serial Monitor output, please refer back to Lesson 3 for a detailed review!

LCD Screen Initialization:while (!lcd.begin(16, 2))

lcd.begin(16, 2): Attempts to initialize the LCD screen. The parameters 16 and 2 specify that the display has 16 columns and 2 rows.

"!": Logical NOT — !lcd.begin(...) means "if initialization fails."

while loop: If initialization fails, the loop continues to execute the code inside it repeatedly (until the LCD initializes successfully or the program halts).

Action When Initialization Fails

```
Serial.println("Could not init backpack. Check wiring.");
```

Serial Output: The program prints an error message to the Serial Monitor:"Could not init backpack. Check wiring."

This indicates that LCD initialization failed and prompts the user to check the wiring connections.

Action When Initialization Succeeds

```
Serial.println("Backpack init'd.");
```

When lcd.begin(16, 2) returns true (indicating successful initialization), the program exits the while loop and prints "Backpack init'd." to the Serial Monitor, confirming that the LCD has been successfully initialized.

3. loop() Function

```
void loop() {
  Pin1Value = analogRead(sensorPin1);
  Serial.print("sensor1 = " );
  Serial.println(Pin1Value);
  String Pin1String = "A3:" + String(Pin1Value);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(Pin1String);
  delay(1000);
}
```

- Sensor Reading:analogRead(sensorPin1) returns an ADC value between 0 and 1023, corresponding to an input voltage range of 0 to 5V.
- **LCD Display:**

  lcd.clear(): Clears the LCD screen.

  lcd.setCursor(0, 0): Positions the cursor at the top-left corner (column 0, row 0).

  lcd.print(): Displays a formatted string, such as "A3:512", showing the current sensor reading.
- **Delay Control:**

  delay(1000): Pauses the program for 1 second to update the display once per second, preventing excessive refresh rates.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, you should have a deeper understanding of how to display information on the LCD. Moving forward, you can apply this knowledge to show data collected from various sensors on the LCD screen.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so that we can observe the hardware functions in action.

This lesson also requires importing the **Adafruit_LiquidCrystal** library. The download and installation process is the same as in Lesson 4—please refer to that lesson for detailed steps.

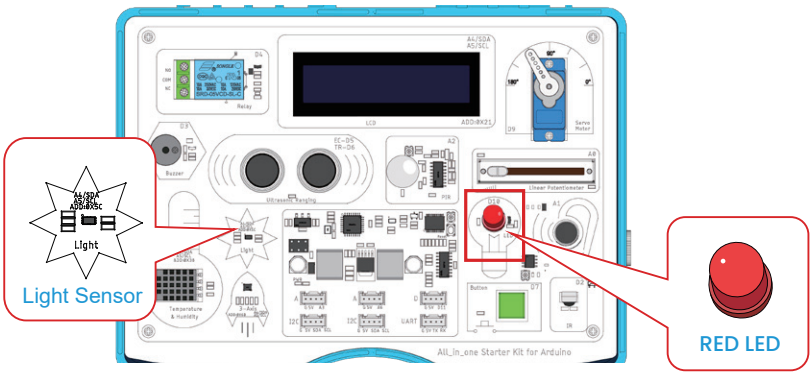
After a successful upload, you will be able to see the soil moisture sensor data displayed on the LCD screen of the All-in-one Starter Kit for Arduino.

Lesson 6 - Intelligent Street Light

Introduction

In this lesson, you will learn how to obtain light intensity data from a light sensor module and use this information to control the LED's on/off state. By defining different brightness thresholds, you can achieve intelligent LED control—activating or deactivating the LED as needed to prevent unnecessary power consumption and promote energy efficiency.

Hardware Used in This Lesson:



Working Principle of the Light Sensor

The light sensor operates based on the photoelectric effect in semiconductor materials. When light photons strike the photosensitive element (such as a photoresistor or photodiode), the photon energy excites electrons in the semiconductor, generating free electrons and holes. This changes the electrical characteristics of the component—such as resistance, current, or voltage—effectively converting light signals into electrical signals. For example, a photoresistor's resistance decreases as illumination increases, while a photodiode under reverse bias exhibits an increase in reverse current proportional to light intensity. These changes are processed by circuits to produce analog or digital outputs.

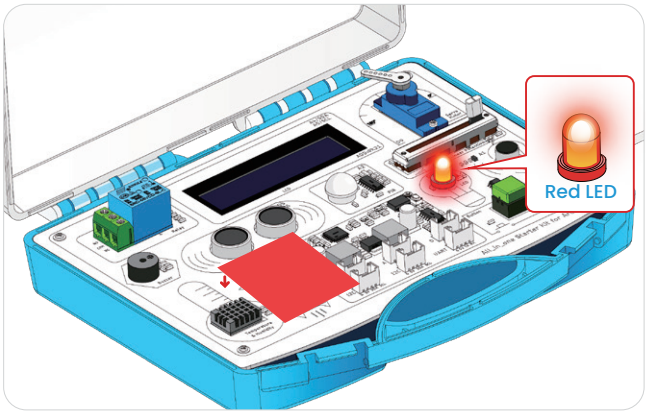
Working Principle of the LED

The LED (Light Emitting Diode) is based on a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from

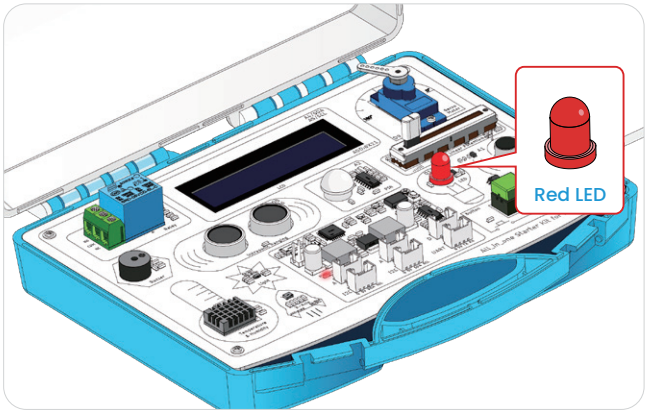
a higher energy level to a lower one, releasing excess energy as photons, which is perceived as light emission. The color (wavelength) of the emitted light is determined by the semiconductor material's bandgap. This is a direct application of electroluminescence.

Operation Effect Diagram

• When an object blocks the light sensor, the LED lights up.



• When the light sensor is exposed to light, the LED turns off.



After running successfully, you will see the LED turn on when you cover the top of the light sensor to simulate low-light conditions. When you remove your hand to simulate a well-lit environment, the LED will turn off.

Key Explanations

1. Global Variables and Initialization

```
#include <BH1750.h>
BH1750 lightMeter(0x5c);
float lux;
```

Sensor Initialization: Communicate with the sensor via the I2C address 0x5C.

BH1750 lightMeter(0x5c); — Creates a BH1750 light sensor object named lightMeter and sets its I2C address to 0x5C.

I2C (Inter-Integrated Circuit) transfers data using only two bidirectional signal lines:

- **SDA (Serial Data Line):** carries the data.
- **SCL (Serial Clock Line):** synchronizes the timing of data transfer.

I2C supports communication with multiple devices on the same bus, where each device has a unique address. The master device selects the target slave device by its address to perform two-way data exchange. Its main advantages are simple wiring, low cost, support for hot-swapping, and operation across low to high speeds.

In simple terms:

I2C is like a “chat channel” built with two wires: one wire (SDA) for sending messages, and another wire (SCL) for keeping the rhythm. Every device connected to these two wires has its own “house number” — the I2C address.

Why is the address needed?

Imagine a group of people talking in the same room. To get someone’s attention, you need to call their name. Similarly, the I2C bus may have multiple devices such as temperature sensors and displays. When the master controller wants to send a command to a particular device, it calls out that device’s “address” (e.g., 0x5C). Only the device with the matching address “responds,” while others ignore the message.

2. setup() Function

```
void setup() {
  Serial.begin(115200);
  Wire.begin();
  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire)) {
    Serial.println(F("BH1750 Advanced begin"));
```

```
  } else {
    Serial.println(F("Error initialising BH1750"));
  }
  pinMode(LedPin, OUTPUT);
}
```

- I2C Communication: Wire.begin() initializes the I2C bus, enabling the Arduino to communicate with the sensor.
- Sensor Configuration:

BH1750::CONTINUOUS_HIGH_RES_MODE

• sets the operating mode to continuous high-resolution mode.

• Continuous mode: The sensor continuously measures light without needing repeated triggers.

• High resolution: Provides high precision with 1 lux accuracy.

• Common Modes:

• CONTINUOUS_HIGH_RES_MODE — high precision (1 lx) continuous measurement.

• CONTINUOUS_LOW_RES_MODE — lower precision (4 lx) but faster continuous measurement.

• ONE_TIME_HIGH_RES_MODE—single high-precision measurement.

• lightMeter.begin() is used to detect whether the sensor is properly connected and ready.

3. loop() Function

```
void loop() {
  if (lightMeter.measurementReady(true)) {
    lux = lightMeter.readLightLevel();
    Serial.print("[-] Light: [");
    Serial.print(lux);
    Serial.println("] lx");
  }
  delay(10);
```

Light Measurement Logic:

• measurementReady(true): Checks whether the measurement is complete; if not, it waits automatically until the data is ready.

• readLightLevel(): Returns the current light intensity value in lux (lx).

Note: The reason for adding serial print statements is to facilitate data monitoring and debugging.

```
Serial.print("[-] Light: [");
Serial.print(lux);
Serial.println("] lx");
```

These three lines of code enable you to view the obtained light intensity data in the Serial Monitor, preparing for subsequent program logic that makes decisions based on the light levels.

LED Control Logic:

```
if (lux <= 100)
  digitalWrite(LedPin, HIGH);
 else
  digitalWrite(LedPin, LOW);
}
```

• When the light intensity is below 100 lux (e.g., at night or in a dark room), the LED turns on.

• When the light intensity is above 100 lux, the LED turns off.

Note:

```
if (condition) {
  // Code executed when conditions are met
} else {
  // Code executed when the condition is not met
}
```

This is similar to a real-life scenario: "If it rains tomorrow, take an umbrella; otherwise, don't."

Execution Logic:

• First, check if the condition is true (non-zero).

• If true, execute the code inside the if block.

• If false, skip the if block and execute the code inside the else block.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

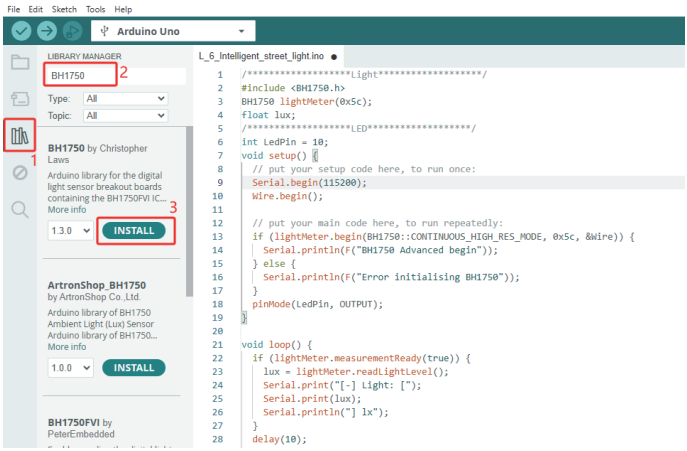
After mastering the above code, you can expand the functionality by using the light sensor to control and interact with more hardware devices.

Uploading the Code

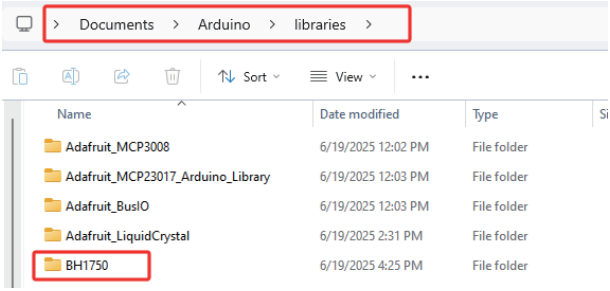
The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so that we can see the hardware functions in action.

Since the photosensitive sensor used in this lesson requires additional library files for driving, we need to add appropriate library files to ensure the code runs normally before uploading it.

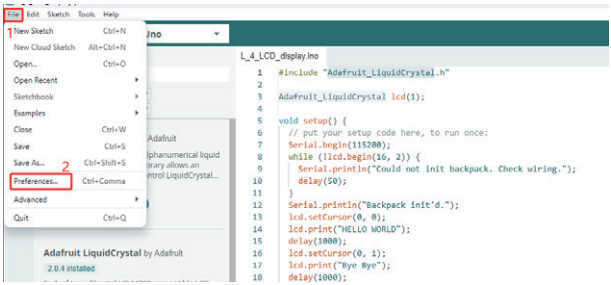
Follow the steps shown in the figure to download the BH1750 library (version 1.3.0). This will allow you to use the library's interfaces to drive the photosensitive sensor.

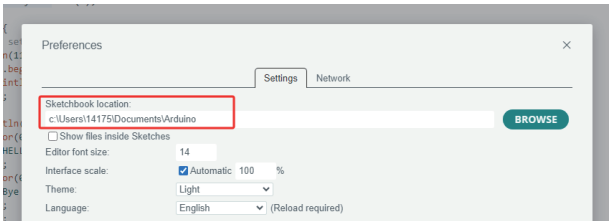


After installation, you can find the downloaded library files in this path



(The download path is determined by the path set in the Arduino IDE.)





Now that the library is downloaded, you can upload the code.

The upload steps are the same as in Lesson 1—please refer to it for guidance.

After a successful upload, you can control the photosensitive sensor on the All_in_one Starter Kit for Arduino to turn the LED on/off.

Implementation:

When you cover the light sensor to simulate low-light conditions, the LED will turn on.

When you remove your hand to simulate adequate lighting, the LED will turn off.

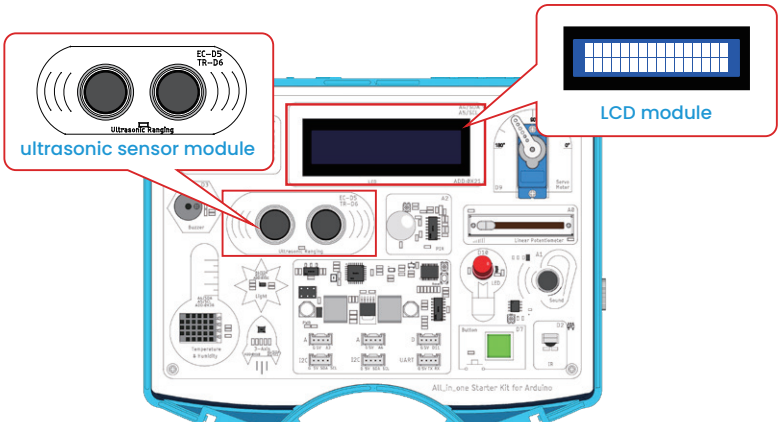
If this does not occur, check to ensure the program is executing correctly.

Lesson 7 - Ultrasonic Ranging Display

Introduction

In this lesson, we will learn how to use an ultrasonic module. With this module, we can measure the distance between a flat surface in front of the module and the module itself. We will create an ultrasonic distance meter and display the measured distance on an LCD module.

Hardware Used in This Lesson:



Working Principle of the LCD Screen

The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices. Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

The backlight module (usually LEDs) provides illumination to ensure clear visibility even

in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Working Principle of the Ultrasonic Sensor

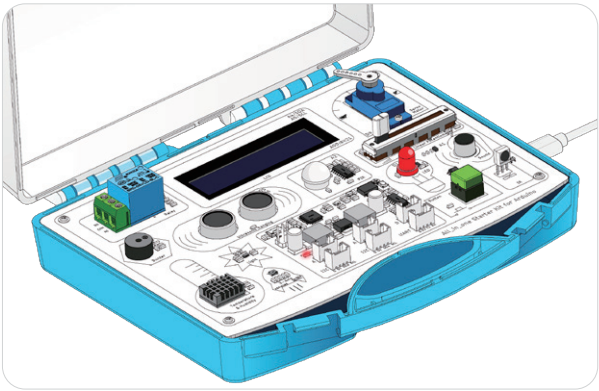
The ultrasonic sensor operates based on the emission and reception of reflected ultrasonic waves. Inside the sensor, a piezoelectric transducer is excited by an electrical signal to produce high-frequency mechanical vibrations, emitting ultrasonic waves at frequencies above 20 kHz. When these ultrasonic waves encounter an obstacle, they reflect back as echoes. The transducer receives the echo and converts the mechanical vibrations back into electrical signals.By measuring the time interval between emitting and receiving the signal (the transit time), and knowing the speed of sound in the medium (approximately 340 m/s in air), the distance to the obstacle is calculated using the formula:

Distance = (Speed of Sound × Time) ÷ 2

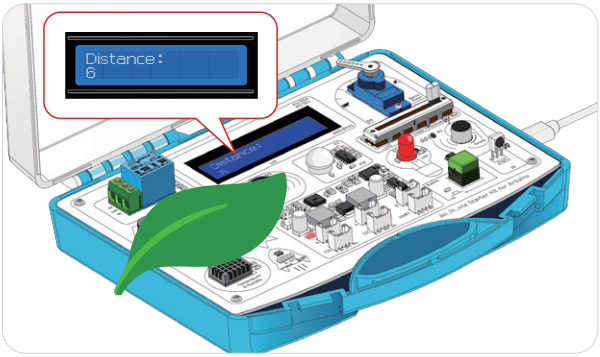
This principle is similar to echolocation. The sensor enables non-contact distance measurement and is widely used in obstacle avoidance, liquid level detection, industrial flaw detection, and more. Thanks to the ultrasonic waves' strong directionality and good penetration, it achieves accurate distance measurements.a higher energy level to a lower one, releasing excess energy as photons, which is perceived as light emission. The color (wavelength) of the emitted light is determined by the semiconductor material's bandgap. This is a direct application of electroluminescence.

Operation Effect Diagram

• No object blocking:



• **Object blocking:**



After successful operation, you will observe the phenomena described above.

You will see the LCD screen continuously updating the distance data measured by the ultrasonic sensor. As the flat surface in front of the ultrasonic module moves, the measured distance value will change accordingly.

Key Explanations

1. LCD Section

```
#include "Adafruit_LiquidCrystal.h"
Adafruit_LiquidCrystal lcd(1);
```

The Adafruit LCD library is included here, and an LCD object is initialized with the parameter 1, which indicates that the LCD display is connected via the I2C interface.

```
String NULL_TXT = "                ";
```

A string named NULL_TXT is defined to help clear the LCD screen. This string contains 16 spaces because the LCD has 16 columns, and writing it effectively blanks out a full line on the display.

```
void LCD_print(String txt1, String txt2)
{
  lcd.setCursor(0, 0);
  lcd.print("                ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
```

```
  lcd.setCursor(0, 0);
  lcd.print(txt1);
  lcd.setCursor(0, 1);
  lcd.print(txt2);
}
```

This is a custom function named LCD_print, used to display two lines of text on the LCD screen.

First, lcd.setCursor(0, 0) and lcd.setCursor(0, 1) position the cursor at the beginning of the first and second lines, respectively.

Then, NULL_TXT is printed to each line to clear any previous content.

Finally, the input strings txt1 and txt2 are printed to the first and second lines of the display.

2. Ultrasonic Section

```
#include <HCSR04.h>
const byte triggerPin = 6;
const byte echoPin = 5;
UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);
```

The HCSR04 library is included to operate the ultrasonic sensor.

The trigger pin (triggerPin) and echo pin (echoPin) are defined.

An UltraSonicDistanceSensor object is initialized using these pins.

```
unsigned long previousMillis = 0;
const long interval = 500;
```

The variable previousMillis is defined to record the last time the LCD was updated.

The constant interval is defined to specify the LCD update interval, which is set to 500 milliseconds.

```
float distance = distanceSensor.measureDistanceCm();
String Value = String((int)distance);
```

The method measureDistanceCm is called to measure the distance, and the result is stored in the distance variable.

The distance value is then converted to an integer, and subsequently converted to a string named Value so it can be displayed on the LCD.

```
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
  previousMillis = currentMillis;
  lcd.setCursor(0, 1);
  lcd.print("   ");
  lcd.setCursor(0, 1);
  lcd.print(Value);
}
```

The current time **currentMillis** is retrieved and compared with **previousMillis** to determine whether the LCD update interval has been reached.

If the interval has passed, **previousMillis** is updated, and the cursor is moved to the **beginning of the second line** of the LCD.

A space is printed first to help clear any old content, followed by printing the **new distance value**.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, you can further enhance its functionality — for example, by using ultrasonic distance detection to trigger other hardware when a certain distance threshold is reached. You can also optimize the provided code to improve measurement accuracy or responsiveness.

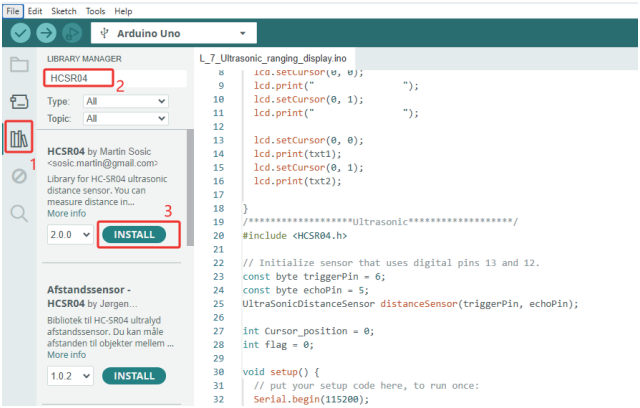
Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

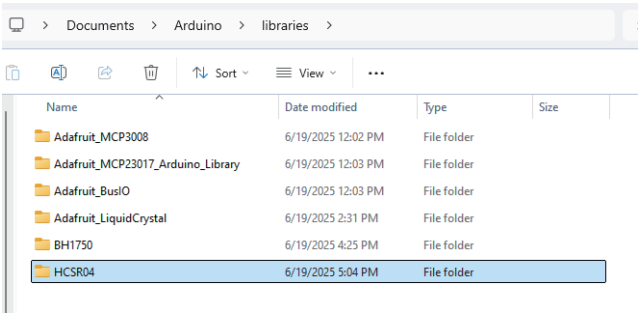
Since the ultrasonic sensor used in this lesson requires an additional driver library, you need to install the appropriate library before uploading the code to ensure it runs correctly.

Follow the steps shown in the diagram to download the HCSR04 library (version 2.0.0). This will allow you to call the library's interfaces to drive the ultrasonic sensor.

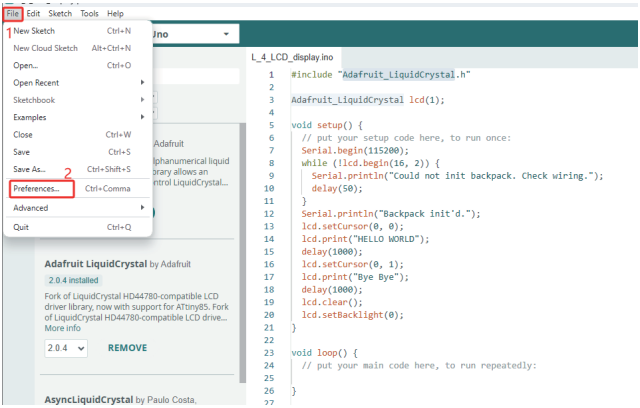
need to install the appropriate library before uploading the code to ensure it runs correctly.

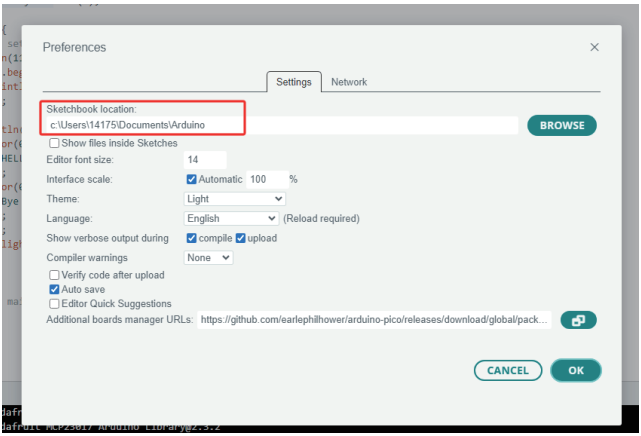


After installation, you can find the downloaded library files in this path.



(The download path is determined by the path set in the Arduino IDE.)





Now that the library is downloaded, you can upload the code.

The upload steps are the same as in Lesson 1—please refer to it for guidance.

After a successful upload, the LCD screen on the All_in_one Starter Kit for Arduino will display distance data collected by the ultrasonic sensor.

Implementation: You will observe the LCD screen continuously updating with distance measurements from the ultrasonic sensor. As an object in front of the ultrasonic module moves, the displayed distance value will change accordingly. If this does not occur, check to ensure the program is executing correctly.

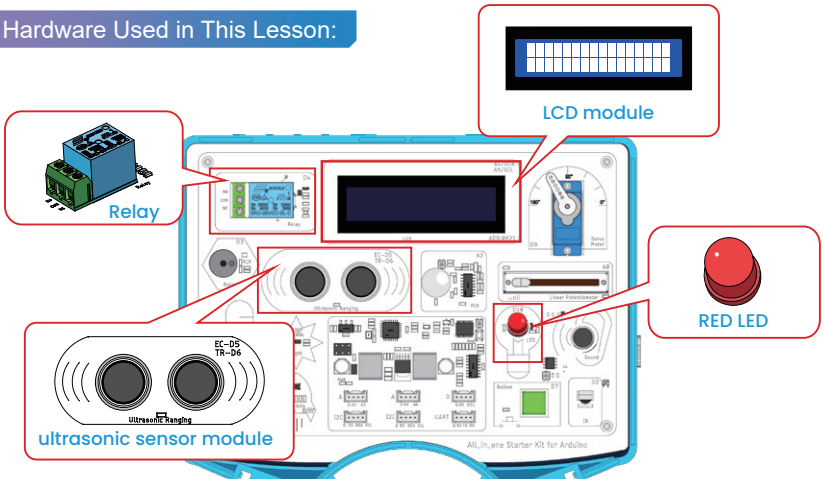
Lesson 8 - Obstacle Close Range Alarm

Introduction

In this section, we will delve deeper into the ultrasonic module and learn how to coordinate its operation with other modules. We will use the distance data obtained from the ultrasonic module to control the activation and deactivation of the relay module and LED module, thereby implementing an ultrasonic obstacle avoidance function.

Note: This lesson builds upon the previous lesson to expand functionality!

Hardware Used in This Lesson:



Working Principle of the LCD Screen

The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices. Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

The backlight module (usually LEDs) provides illumination to ensure clear visibility even

in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Working Principle of the Ultrasonic Sensor

The ultrasonic sensor operates based on the emission and reception of reflected ultrasonic waves. Inside the sensor, a piezoelectric transducer is excited by an electrical signal to produce high-frequency mechanical vibrations, emitting ultrasonic waves at frequencies above 20 kHz. When these ultrasonic waves encounter an obstacle, they reflect back as echoes. The transducer receives the echo and converts the mechanical vibrations back into electrical signals.By measuring the time interval between emitting and receiving the signal (the transit time), and knowing the speed of sound in the medium (approximately 340 m/s in air), the distance to the obstacle is calculated using the formula:

Distance = (Speed of Sound × Time) ÷ 2

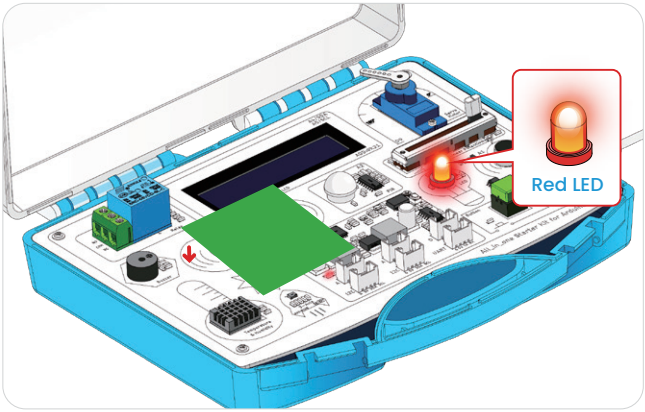
This principle is similar to echolocation. The sensor enables non-contact distance measurement and is widely used in obstacle avoidance, liquid level detection, industrial flaw detection, and more. Thanks to the ultrasonic waves' strong directionality and good penetration, it achieves accurate distance measurements.a higher energy level to a lower one, releasing excess energy as photons, which is perceived as light emission. The color (wavelength) of the emitted light is determined by the semiconductor material's bandgap. This is a direct application of electroluminescence.

Working Principle of Relays

A relay is an electrically controlled switch that uses electromagnetic induction or other physical effects to control circuit connections. Its core components include a coil, iron core, armature, and contacts (normally open/closed). When current flows through the coil, it generates a magnetic field that magnetizes the iron core, attracting the armature and moving the contacts. This action closes normally open contacts and opens normally closed contacts. When the coil is de-energized, the magnetic field dissipates, and the armature returns to its original position via a reset spring, restoring the contacts to their initial state. Solid-State Relays (SSR) use semiconductor devices (e.g., thyristors) instead of mechanical contacts. They achieve electrical isolation between input and output circuits through optoelectronic or magnetic coupling. Relays excel at using low-voltage, low-current control signals (e.g., microcontroller pins) to drive high-voltage, high-current loads. They are widely used in automation control, power systems, and household appliances for circuit protection, logical control, and signal amplification.

Operation Effect Diagram

When the distance is less than 30 centimeters, both the relay and the LED light will be activated simultaneously.



If the distance reaches or exceeds 30 centimeters, the relay and the LED light will be turned off.

Key Explanations

1. Pin Definitions

```
int relayPin = 4;
int LedPin = 10;
```

The relayPin and LedPin are defined to connect the relay and LED to digital pins 4 and 10, respectively.

2. Ultrasonic Section Section

```
#include <HCSR04.h>
const byte triggerPin = 6;
const byte echoPin = 5;
UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);
```

The HCSR04 library is included to operate the ultrasonic sensor.

The trigger pin (triggerPin) and echo pin (echoPin) are defined, and an UltraSonicDistanceSensor object is initialized using these pins.

3. Initial LCD Display

```
LCD_print("Distance    30", "");
```

The LCD_print function is called to set the first line to display “Distance 30” and clear the second line.

Here, “30” is a threshold value indicating that when the measured distance is greater than or equal to 30 cm, the relay and LED will be turned off; when the distance is less than 30 cm, the relay and LED will be turned on.

4. Distance Judgment and Control

```
if ((int)distance >= 30)
{
  lcd.setCursor(9, 0);
  lcd.print(">=");
  digitalWrite(relayPin, LOW);
  digitalWrite(LedPin, LOW);
}
else
{
  lcd.setCursor(9, 0);
  lcd.print("< ");
  digitalWrite(relayPin, HIGH);
  digitalWrite(LedPin, HIGH);
}
```

- Convert the measured distance value to an integer and compare it with the threshold of 30.

  distance >= 30： （(distance is greater than or equal to 30 cm）：

  • lcd.setCursor(9, 0)：Display "≥" at column 9 of the first row on the LCD.

  • digitalWrite(relayPin, LOW);

  digitalWrite(LedPin, LOW);

  Set the relay pin relayPin and LED pin LedPin to low level (turn off).

- else: （(if distance is less than 30 cm)：

  • lcd.setCursor(9, 0)：Display "<" at column 9 of the first row on the LCD.

  • digitalWrite(relayPin, HIGH);

  digitalWrite(LedPin, HIGH);

  Set the relay pin relayPin and LED pin LedPin to high level (turn on).

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, you have further enhanced your ability to control multiple hardware components. You now have a solid understanding of trigger mechanisms and hardware control.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

The code upload procedure for this lesson is the same as that in Lesson 7. Please refer to the upload steps outlined in Lesson 7!

After the upload is successful, you will be able to see the ultrasonic sensor on the All_in_one Starter Kit for Arduino continuously measuring distance data in real time. When the measured distance reaches a certain trigger condition, it will activate other hardware devices.

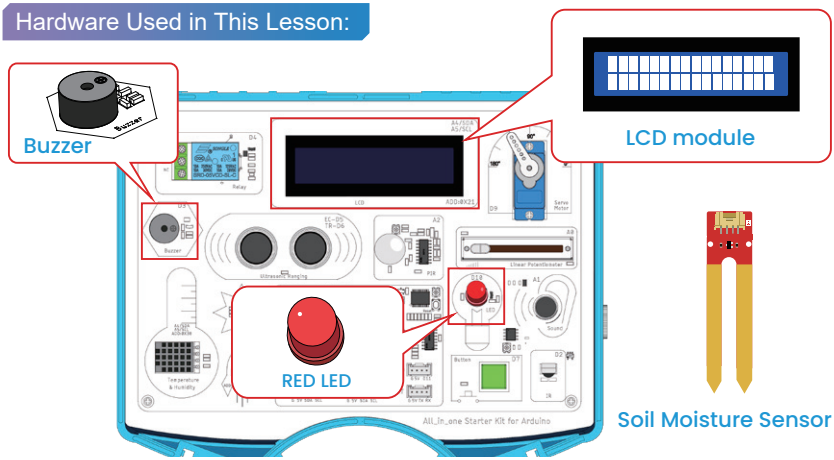
Implementation: You will observe that as the measured distance changes, when it is less than 30 cm, both the relay and LED will be activated, indicating an obstacle has been detected. When the distance reaches or exceeds 30 cm, the relay and LED will turn off, indicating the surrounding environment is safe and free of obstacles.If this behavior does not occur, please verify that the program is running correctly.

Lesson 9 - Plant Watering Reminder System

Introduction

In this lesson, we will learn how to use a soil moisture sensor to monitor changes in soil moisture levels. When the moisture drops to 10% or below, a buzzer will sound an alert to remind you to water the plants. When the moisture level is between 10% and 20%, an LED will blink to indicate mild water deficiency, prompting you to water the plants soon. When the moisture content exceeds 20%, it means the plants are in good condition and do not require additional watering.

Hardware Used in This Lesson:



Working Principle of the LCD Screen

The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices. Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

The backlight module (usually LEDs) provides illumination to ensure clear visibility even

in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Working Principle of Soil Moisture Sensors

Soil moisture sensors detect changes in the physical or electrical properties of the soil to reflect its water content. The core principle is based on how moisture affects the soil's conductivity and dielectric constant. For example, resistive sensors measure moisture by detecting changes in resistance between metal electrodes, while capacitive sensors measure changes in capacitance caused by variations in the dielectric constant. More advanced methods such as TDR (Time Domain Reflectometry) and FDR (Frequency Domain Reflectometry) calculate moisture levels based on the propagation characteristics of electromagnetic signals in the soil. These physical changes are ultimately converted into electrical signals, providing quantifiable moisture data for applications such as agricultural irrigation and environmental monitoring.

Working Principle of an LED

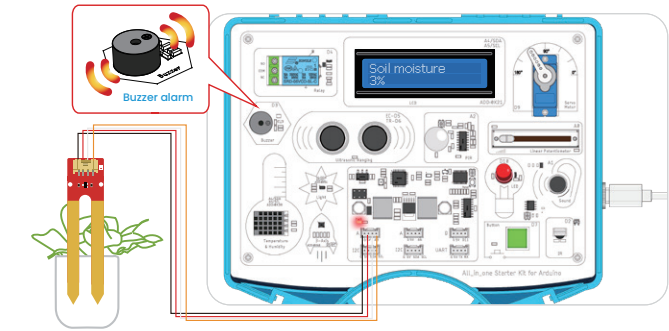
The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from a higher to a lower energy level, releasing excess energy in the form of photons—producing visible light. The color (wavelength) of the light depends on the bandgap of the semiconductor material. This process is a direct application of electroluminescence.

Working Principle of a Buzzer

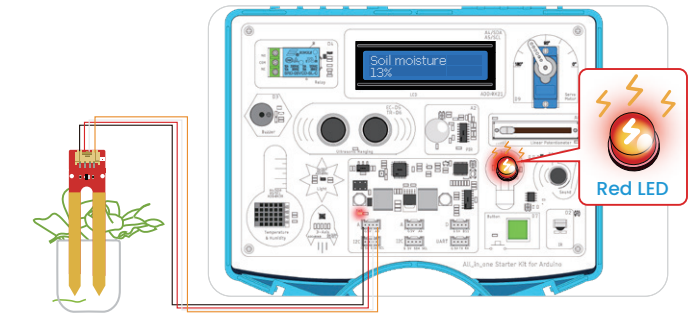
A buzzer is an electronic component that converts electrical signals into sound signals, operating based on either electromagnetic induction or the piezoelectric effect. An electromagnetic buzzer contains a coil, a magnet, and a vibrating diaphragm. When current flows through the coil, it generates a magnetic field that interacts with the permanent magnet, causing the diaphragm to vibrate and produce sound. The presence and pitch of the sound can be controlled by modulating the current's frequency and duration. In contrast, a piezoelectric buzzer uses piezoelectric materials (such as piezoceramics) that deform mechanically when an alternating voltage is applied—a phenomenon known as the inverse piezoelectric effect. This deformation drives the diaphragm to vibrate and emit sound at a specific frequency. Both types of buzzers require external circuitry for proper operation and are commonly used in alarms, electronic alerts, and notification systems.

Operation Effect Diagram

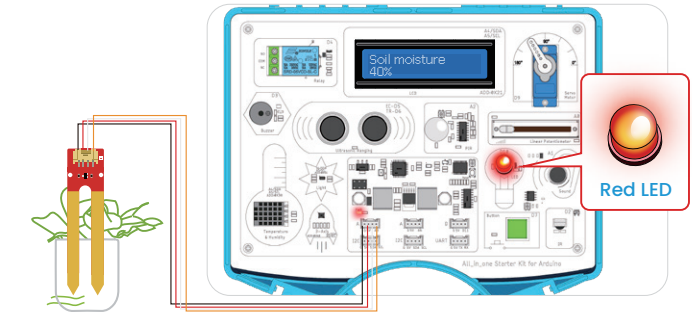
When the soil moisture drops below 10%, the buzzer will sound an alarm to remind you to water the plant.



When the moisture is between 10% and 20%, the LED will blink, indicating mild water deficiency and prompting timely watering.



When the moisture level exceeds 20%, it indicates that the plant is in good condition, and the red LED remains steadily on.



Once the program runs successfully, you will observe that the LED and buzzer respond to changes in soil moisture levels:

• When the buzzer sounds, it indicates that the plant is severely lacking water and requires immediate watering.

• When the LED blinks, it means the plant is moderately lacking water and needs to be watered soon.

• When the LED remains steadily on, it shows that the plant has sufficient moisture and is growing well.

Key Explanations

1. Moisture Sensor

```
int sensorPin1 = A3;
int Pin1Value = 0;
int mappedValue;
```

• Defined the input pin sensorPin1 for the soil moisture sensor, which is connected to analog input pin A3.

• Defined the variable Pin1Value to store the raw value read from the sensor.

• Defined the variable mappedValue to store the mapped percentage value representing the soil moisture level.

2. LCD Display

```
void LCD_print(String txt1, String txt2)
{
  lcd.setCursor(0, 0);
  lcd.print("                ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
  lcd.setCursor(0, 0);
  lcd.print(txt1);
  lcd.setCursor(0, 1);
  lcd.print(txt2);
}
```

• A custom function LCD_print is used to print two lines of text on the LCD screen.

• It first clears the old content from the screen, then prints the new text lines.

3. Initialization Section

```
void setup() {
  Serial.begin(115200);
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  Serial.println("Backpack init'd.");
  pinMode(sensorPin1, INPUT);
  pinMode(buzzerPin, OUTPUT);
  LCD_print("Soil moisture", "");
}
```

• Initializes serial communication with a baud rate of 115200.

• Initializes the LCD screen; if initialization fails, it prints an error message and delays repeatedly until successful.

• Sets the soil moisture sensor pin as input and the buzzer pin as output.

• Calls LCD_print to display "Soil moisture" on the first line of the LCD, and clears the second line.

4. Main Loop Section

```
void loop() {
  Pin1Value = analogRead(sensorPin1);
  Serial.print("sensor1 = " );
  Serial.println(Pin1Value);
  mappedValue = map(Pin1Value, 0, 1023, 0, 100);
  String Value = String(mappedValue) + "%";
  lcd.setCursor(0, 1);
  lcd.print("    ");
  lcd.setCursor(0, 1);
  lcd.print(Value);
  delay(500);
  if (mappedValue < 10)
  {
    digitalWrite(LedPin, LOW);
    tone(buzzerPin, 1300);
```

```
    delay(250);
    noTone(buzzerPin);
  } else if (mappedValue >= 10 && mappedValue < 20)
  {
    digitalWrite(LedPin, HIGH);
    delay(100);
    digitalWrite(LedPin, LOW);
  } else if (mappedValue >= 20)
  {
    digitalWrite(LedPin, HIGH);
  }
}
```

Reading Soil Moisture Sensor Value:

```
Pin1Value = analogRead(sensorPin1);
```

The analogRead function is used to read the analog value from the soil moisture sensor and store it in the variable Pin1Value. The raw value is printed to the serial monitor for debugging purposes.

Mapping the Moisture Value:

```
mappedValue = map(Pin1Value, 0, 1023, 0, 100);
```

The map function is used to convert the raw analog reading (ranging from 0 to 1023) into a percentage value between 0 and 100, representing the soil moisture level.

Updating the LCD Display:

```
String Value = String(mappedValue) + "%";
lcd.setCursor(0, 1);
lcd.print("    ");
lcd.setCursor(0, 1);
lcd.print(Value);
```

The mapped soil moisture value is converted into a string and appended with a percentage sign ("%").

The second row of the LCD is cleared, and the new moisture value is displayed.

Controlling the Buzzer and LED Based on Moisture Level:

```
if (mappedValue < 10)
{
  tone(buzzerPin, 1300);
  delay(250);
  noTone(buzzerPin);
} else if (mappedValue >= 10 && mappedValue < 20)
{
  digitalWrite(LedPin, HIGH);
  delay(100);
  digitalWrite(LedPin, LOW);
} else if (mappedValue >= 20)
{
  digitalWrite(LedPin, HIGH);
}
```

mappedValue < 10: If the moisture level is below 10%, the buzzer is triggered and sounds for 250 milliseconds as a critical low moisture warning.

mappedValue >= 10 && mappedValue < 20: If the moisture level is between 10% and 20%, the LED blinks once (set HIGH for 100 milliseconds, then set LOW).

mappedValue >= 20: If the moisture level is 20% or above, the LED remains steadily ON, indicating healthy soil conditions.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After going through the code, you'll have a much better handle on how different sensors and hardware work together, and how to use logic to control more devices.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

This lesson also requires importing the Adafruit_LiquidCrystal library. The download and installation process is the same as in Lesson 4, so please refer to that for guidance!

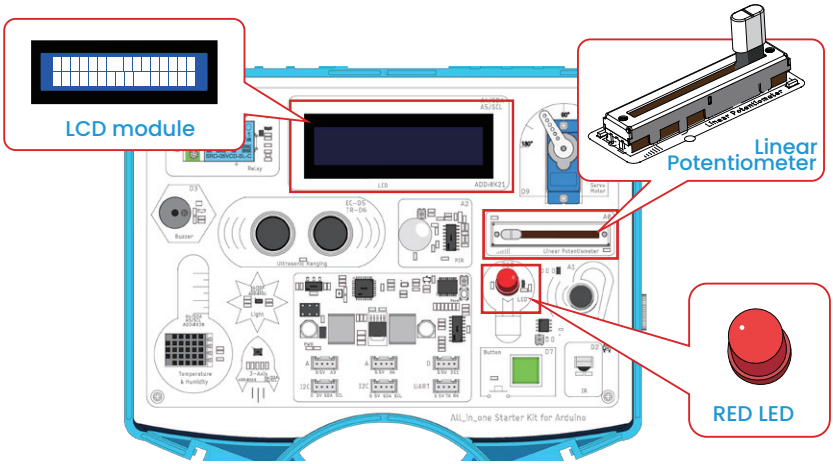
After the upload is successful, you will be able to see the data collected by the soil moisture sensor displayed on the LCD screen of the All_in_one Starter Kit for Arduino, allowing real-time monitoring of the current soil moisture levels. When the moisture reaches the preset threshold, the system will trigger an alert to notify the user to take appropriate action promptly.

Lesson 10 - Brightness Display

Introduction

In this lesson, we will use a sliding sensor (potentiometer) to adjust the brightness of an LED, allowing you to experience interactive manual control of light intensity. By simply moving the slider, you can change the LED's brightness in real time—from soft, dim light to bright, intense illumination—providing an intuitive understanding of how analog signals and PWM (Pulse Width Modulation) technology work together to control brightness.

Hardware Used in This Lesson:



Working Principle of the LCD Screen

The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices. Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

The backlight module (usually LEDs) provides illumination to ensure clear visibility even in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Working principle of a sliding potentiometer

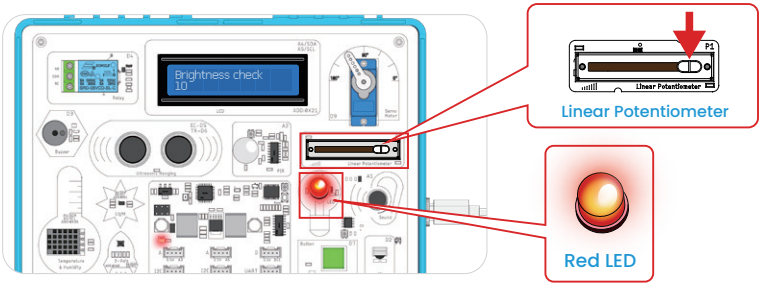
A sliding potentiometer adjusts its resistance by changing the effective contact length on the resistive element. As the sliding contact (wiper) moves along the resistive track, the length of the resistive material in the current path varies, causing the total resistance to change. Linear potentiometers have resistance proportional to the wiper position, while logarithmic types follow a nonlinear curve. The design must ensure reliable contact to avoid noise or erratic behavior.

Working Principle of an LED

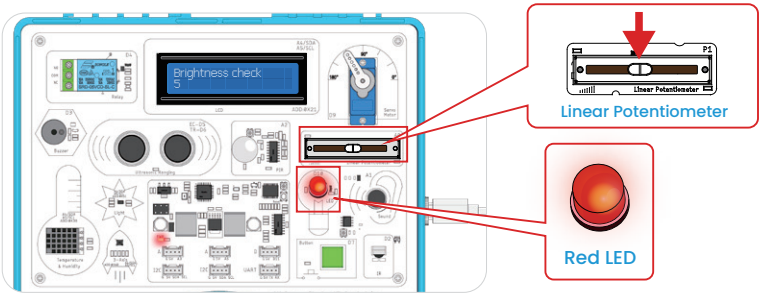
The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from a higher to a lower energy level, releasing excess energy in the form of photons—producing visible light. The color (wavelength) of the light depends on the bandgap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

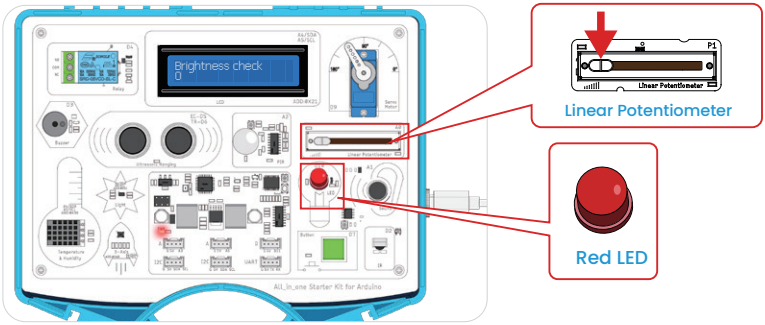
When the sliding module is all the way to the right, the brightness is at its maximum.



When the sliding module is in the middle, the brightness is at a medium level.



When the sliding module is all the way to the left, the brightness is at its minimum.



Once the program runs successfully, you'll be able to use the sliding sensor (potentiometer) to control the LED brightness, and the LCD will display a brightness level from 0 to 10, allowing for intuitive brightness adjustment.

Key Explanations

1. Variable Definition

```
int LinearPin = A0;
int LedPin = 10;
```

LinearPin: Connected to analog input pin A0, used to read the value from the linear sensor.

LedPin: Connected to digital pin 10, used to control the brightness of the LED.

2. Initialization Function

```
void setup() {
  Serial.begin(115200);
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  Serial.println("Backpack init'd.");
  pinMode(LedPin, OUTPUT);
  pinMode(LinearPin, INPUT);
  lcd.setCursor(0, 0);
  lcd.print("Brightness check");
}
```

Initializes serial communication for debugging and monitoring purposes.

Initializes the LCD screen; if initialization fails, it continuously prints error messages in a loop until successful.

Sets the LED pin and the linear sensor pin to the appropriate pin modes.

Displays “Brightness check” on the first line of the LCD, indicating this is a brightness detection program.

3. Initialization Function

```
void loop() {
  int adcValue;
  int mappedValue;
  adcValue = analogRead(LinearPin);
```

```
  mappedValue = map(adcValue, 0, 1023, 0, 255);
  analogWrite(LedPin, mappedValue);
  mappedValue = map(adcValue, 0, 1023, 0, 10);
  String Value = String(mappedValue);
  delay(100);
  lcd.setCursor(0, 1);
  lcd.print("    ");
  lcd.setCursor(0, 1);
  lcd.print(Value);
}
```

- **Sensor Value Reading:**

  The analogRead function is used to read the analog value from the linear sensor and store it in adcValue.

- **Value Mapping:**

  The map function is used to map adcValue from a range of 0–1023 to 0–255, which is used to control the PWM duty cycle of the LED.

  The value is mapped again from 0–1023 to 0–10 for display purposes on the LCD.

- **LED Brightness Control:**

  The analogWrite function writes the mapped PWM value to the LED pin, adjusting the LED brightness accordingly.

- **LCD Display Update:**

  The mapped brightness level is converted to a string and displayed on the second line of the LCD.

  lcd.print(" ") is used to clear any previous content on the second line, ensuring the display remains clean and accurate.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, you should now have a stronger grasp of how multiple sensors and hardware components can work together. By applying logical control, you can coordinate and manage various devices, enabling functions like the brightness level control and display demonstrated in this lesson.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

This lesson also requires importing the Adafruit_LiquidCrystal library. The download and installation process is the same as in Lesson 4—please refer to Lesson 4 for detailed steps!

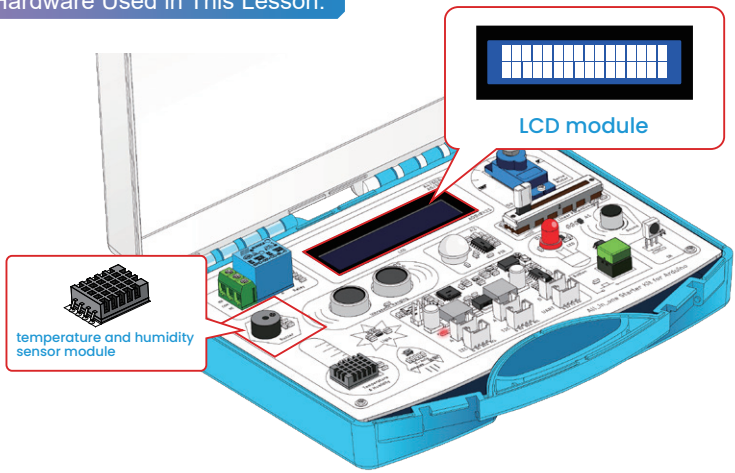
Once the upload is successful, you will see the current LED brightness level displayed on the LCD screen of the All_in_one Starter Kit for Arduino. You can then control the LED brightness in real time by adjusting the sliding potentiometer, allowing for an intuitive and interactive lighting experience.

Lesson 11 - Temperature&Humidity Detecting System

Introduction

In this lesson, we will introduce how to use a temperature and humidity sensor module to obtain temperature and humidity data and display it on a screen. With this module, we can collect real-time local temperature and humidity readings, which will be beneficial for our future experiments and research.

Hardware Used in This Lesson:



Working Principle of the LCD Screen

The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices. Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

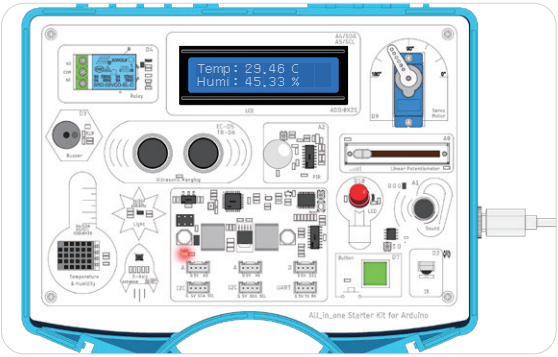
The backlight module (usually LEDs) provides illumination to ensure clear visibility even in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Working Principle of Temperature and Humidity Sensors

A temperature and humidity sensor is an electronic device used to measure ambient temperature and humidity levels. Its working principle relies on the response characteristics of sensitive elements to environmental physical quantities:For temperature measurement, components such as thermistors, thermocouples, or semiconductor-based temperature sensors are commonly used as sensing elements. A thermistor, for instance, exhibits a predictable change in resistance as temperature varies — for example, a negative temperature coefficient (NTC) thermistor decreases in resistance as temperature increases. This change in resistance is typically converted into a voltage signal using a Wheatstone bridge or similar circuit, then processed through analog-to-digital conversion (ADC) and calibration algorithms to produce an accurate temperature reading.

Operation Effect Diagram

The temperature and humidity sensor is continuously acquiring real-time environmental data.



Once the program runs successfully, you will observe that the temperature and humidity values from the sensor are periodically updated and displayed on the LCD screen.

Key Explanations

1. DHT20 Temperature and Humidity Sensor

```
#include "DHT20.h"
DHT20 DHT(&Wire);
```

• The DHT20 library is included to operate the DHT20 temperature and humidity sensor.

• A DHT20 object is initialized, configured to use the I2C interface.

```
uint8_t count_DHT20 = 0;
```

A variable named count_DHT20 is defined to keep track of the number of readings, with debugging information printed every 10 reads.

2. LCD Display

```
#include "Adafruit_LiquidCrystal.h"
Adafruit_LiquidCrystal lcd(1);
```

• The Adafruit LCD library is included, and an LCD object is initialized using the I2C interface.

• The Adafruit_LiquidCrystal library supports multiple LCD interfaces (parallel, I2C, SPI).

• The parameter 1 in lcd(1) specifies the I2C bus number or channel.

```
void LCD_print(String txt1, String txt2)
{
  lcd.setCursor(0, 0);
  lcd.print("                ");
  lcd.setCursor(0, 1);
  lcd.print("                ");

  lcd.setCursor(0, 0);
  lcd.print(txt1);
  lcd.setCursor(0, 1);
  lcd.print(txt2);
}
```

• A custom function LCD_print is defined to print two lines of text on the LCD screen. It first clears the previous content on the screen and then prints the new text.

3. Initialization Functions

```
void setup() {
  Serial.begin(115200);
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  Serial.println("Backpack init'd.");
  Wire.begin();
  DHT.begin();
}
```

• Serial.begin(115200)：Initializes serial communication for debugging and monitoring.

• lcd.begin(16, 2)：Initializes the LCD screen with 16 columns and 2 rows; if initialization fails, it continuously prints error messages until successful.

• Wire.begin()：Initializes the I2C interface.

• DHT.begin()：Initializes the DHT20 temperature and humidity sensor.

4. Main Loop Function

LCD_print("Temp:", "Humi:");

```
LCD_print("Temp:", "Humi:");
```

• Call the LCD_print function to display labels on the LCD—“Temp:” on the first line and “Humi:” on the second line. This sets up temperature and humidity labels on the screen.

Time Interval Control

```
if (millis() - DHT.lastRead() >= 1000)
```

• Use the millis() function to get the current time and compare it with the last data read time from the DHT20 sensor (DHT.lastRead()). This ensures data is read and updated at appropriate intervals.

• If the time difference is greater than or equal to 1000 milliseconds (1 second), the reading operation is executed. This ensures that temperature and humidity data are read once every second.

Read temperature and humidity data

```
int status = DHT.read();
```

• Call the DHT.read() method to read temperature and humidity data. This method updates the internal state of the DHT20 sensor, including temperature and humidity values.

Read temperature and humidity data

```
if ((count_DHT20 % 10) == 0) {
  count_DHT20 = 0;
  Serial.println();
  Serial.println("Type\tHumidity (%)\tTemp (°C)");
}
count_DHT20++;
```

• Use the variable count_DHT20 to keep track of the number of readings taken.

• Every 10 readings, print a debug information header to the serial monitor.

• After each reading, increment the count_DHT20 variable by one.

Construct display strings

```
String TemValue = "Temp:" + String(DHT.getTemperature()) + " C";
String HumValue = "Humi:" + String(DHT.getHumidity()) + " %";
```

• Construct display strings for temperature and humidity to be shown on the LCD.

Update LCD display

```
lcd.setCursor(5, 0);
lcd.print(String(DHT.getTemperature()) + "
C");lcd.setCursor(5, 1);
lcd.print(String(DHT.getHumidity()) + " %");
```

- Use lcd.setCursor to set the cursor position, displaying humidity on the first line and temperature on the second line.
- Call lcd.print to output the humidity and temperature values onto the LCD screen.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

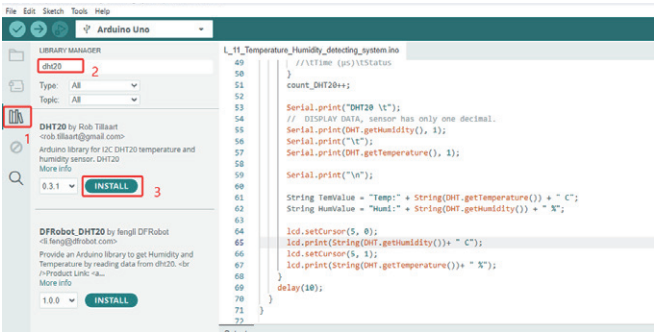
After studying the above code, you can further customize the functionality—for example, use the data collected from the temperature and humidity sensor to set threshold values that control other hardware devices.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

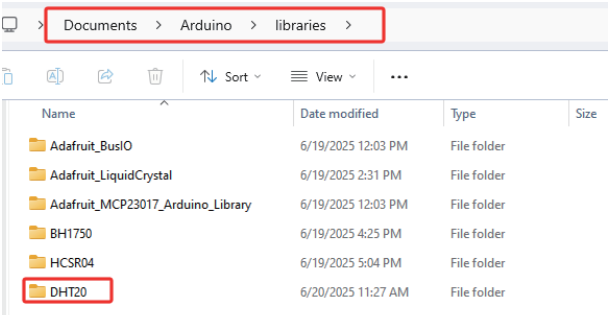
Since the DHT20 temperature and humidity sensor used in this lesson requires additional library support, you need to install the appropriate driver libraries before uploading the code to ensure it runs correctly.

Follow the steps shown in the figure to download the DHT20 library (version 0.3.1), which is necessary to drive the temperature and humidity sensor properly.

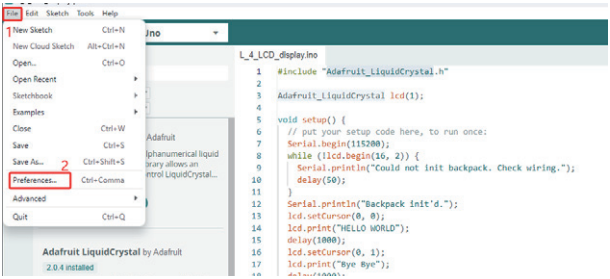


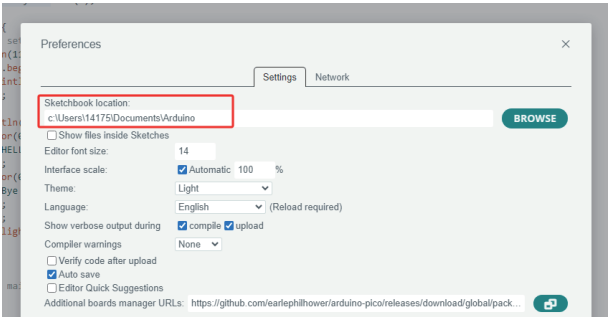
This lesson also requires importing the Adafruit_LiquidCrystal library (version 2.0.4). The download and installation process is the same as in Lesson 4—please refer to Lesson 4 for details!

After installation, you can find the downloaded library files in this path.



(The download path is determined by the path set in the Arduino IDE.)





Now that the library is downloaded, you can upload the code.

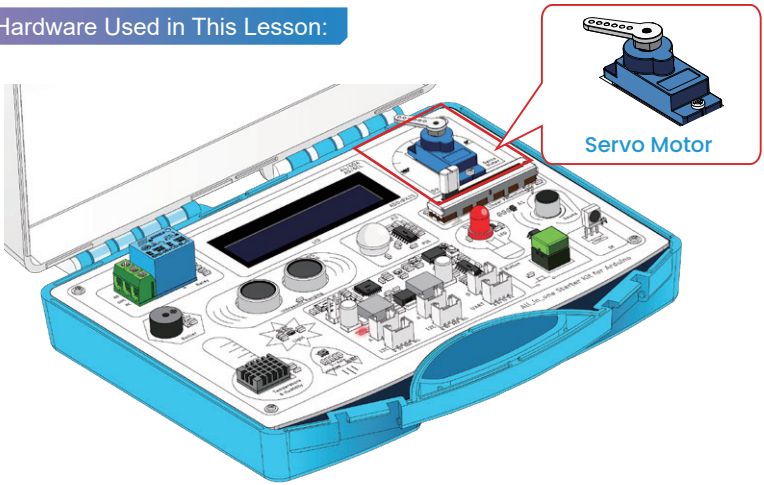
After the upload is successful, you will be able to see the temperature and humidity data from the sensor displayed on the LCD screen of the All_in_one Starter Kit for Arduino.

Lesson 12 - Servo Control

Introduction

In this lesson, we will systematically study the basic operating principles and control methods of servo motors. Through programming, you will achieve precise angle control of the servo motor, allowing it to rotate smoothly from 0 degrees to 180 degrees, then reverse back to 0 degrees. This will demonstrate the full range of reciprocating motion, providing a clear understanding of how servo motors are typically used in position control applications.

Hardware Used in This Lesson:

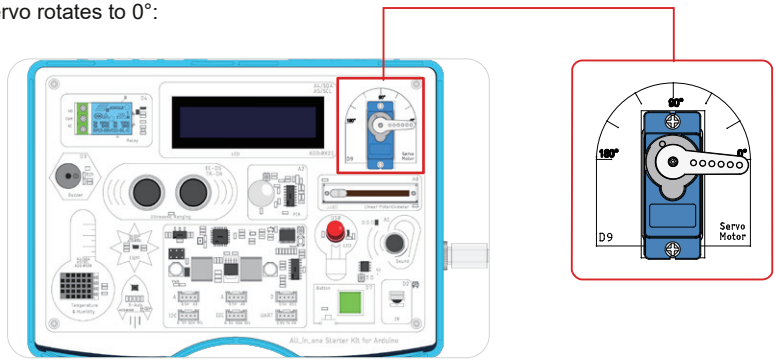


Working Principle of the LCD Screen

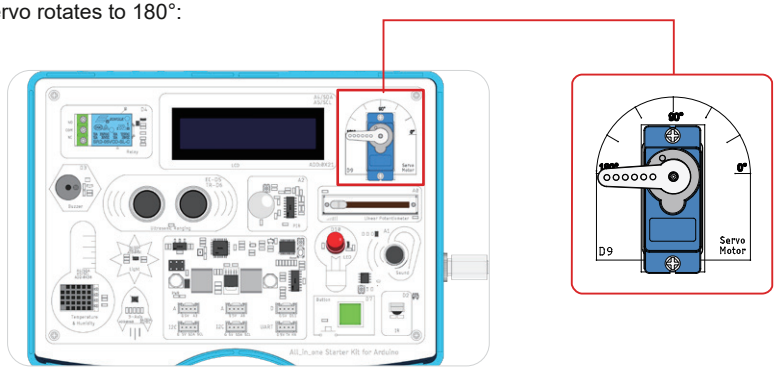
A servo motor is an angular position control actuator that integrates the principles of a servo system, operating based on a closed-loop feedback control mechanism. When a control signal—usually a Pulse Width Modulation (PWM) signal—is input, the pulse width corresponds to a target angle (for example, 1 ms pulse width corresponds to 0°, 1.5 ms to 90°, and 2 ms to 180°). The control circuit compares the input signal with the real-time angle feedback from a potentiometer. If there is a deviation, it drives the motor to rotate. The motor, through a gear reduction mechanism, turns the output shaft, while the potentiometer rotates synchronously with the output shaft, changing its resistance to reflect the current angle and feeding this back to the control circuit. This closed-loop system continuously adjusts the motor's direction and speed until the actual output shaft angle matches the target angle, stopping movement to achieve precise angular control within ±0.5°.

Operation Effect Diagram

Servo rotates to 0°:



Servo rotates to 180°:



Key Explanations

1. Servo Initialization and Parameter Configuration

```
Servo myservo;
myservo.attach(9, 600, 2520);
```

- Servo class: Built-in Arduino library used for controlling servo motors.
- attach(pin, min, max):

• pin：The Arduino pin connected to the servo control signal (pin 9 in this example).

min/max：The pulse width in microseconds corresponding to the servo angles.

- Default range: 544μs (0°) to 2400μs (180°).
- Adjusted range: 600μs (0°) to 2520μs (180°), tailored for specific servo mechanical limits.

2. Core Logic for Angle Control

```
for (pos = 0; pos <= 180; pos += 1) {
  myservo.write(pos);
  delay(15);
```

- write(angle): Sends an angle command to the servo.

  Angle mapping:

  • write(0) → Outputs a 600μs pulse (corresponding to 0°).

  • write(90) → Outputs a 1560μs pulse (midpoint).

  • write(180) → Outputs a 2520μs pulse (corresponding to 180°)。

- **delay(15):**

  • The servo needs time to physically move; the delay ensures enough time before the next command.

  • Too short a delay may cause the servo to jitter or fail to reach the target position.

3. Implementation of Reciprocal Motion

```
for (pos = 0; pos <= 180; pos += 1) { ... }
delay(2000);
for (pos = 180; pos >= 0; pos -= 1) { ... }
delay(2000);
```

- Loop structure: Uses two for-loops to achieve back-and-forth movement.

  Delay function: Adds a pause at the endpoint positions to make the servo's status easier to observe.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

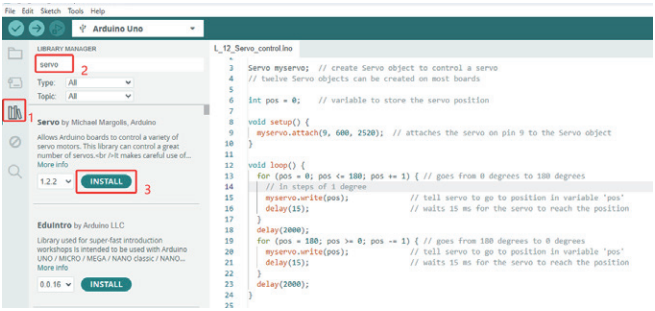
After studying the above code, you can further enhance the functionality—for example, by displaying the servo's current rotation angle on the screen in real time.

Uploading the Code

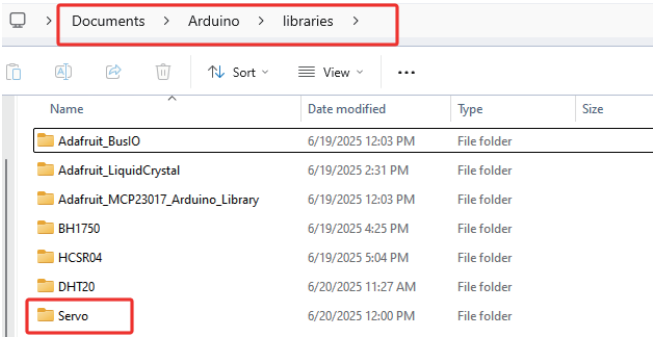
The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

Since this lesson involves using a servo motor that requires additional library support, before uploading the code, we need to install the appropriate library to ensure the code runs correctly.

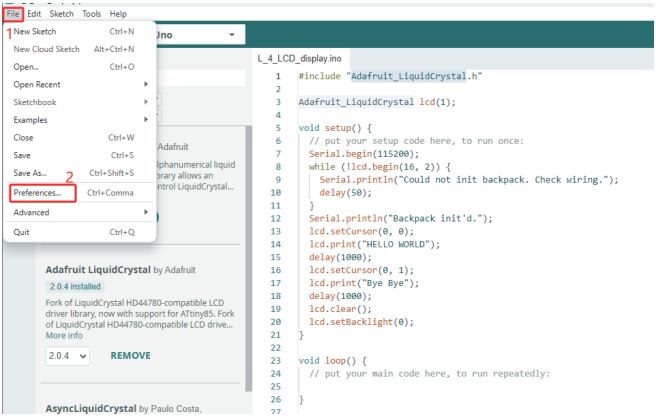
Follow the steps shown in the diagram to download the Servo library (version 1.2.2) so that the servo motor can function properly.

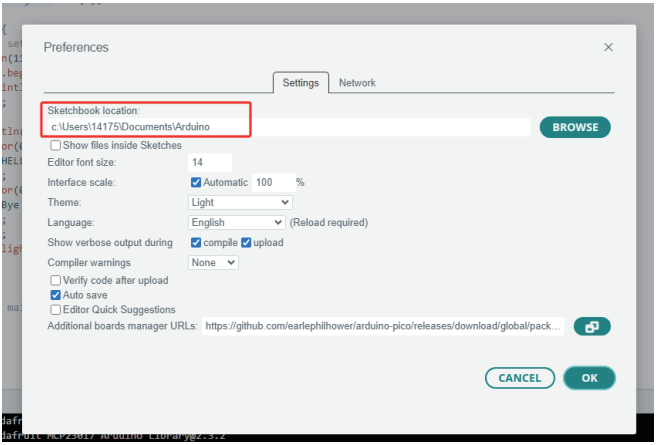


After installation, you can find the downloaded library files in this path.



(The download path is determined by the path set in the Arduino IDE.)





Now that the library is downloaded, you can upload the code.

The upload steps are the same as in Lesson 1—please refer to it for guidance.

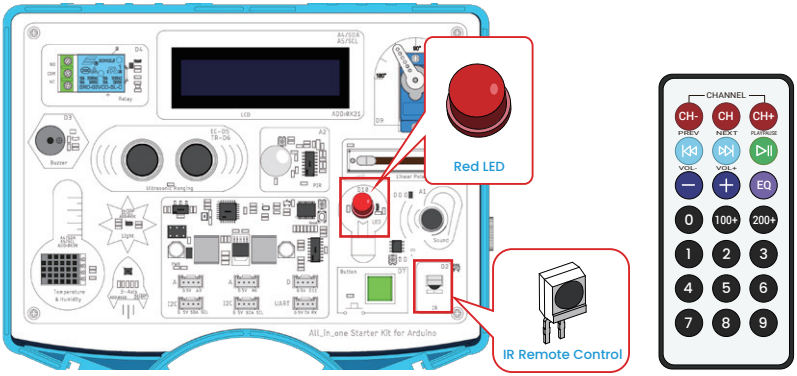
Once the upload is successful, you will see the servo motor on the All_in_one Starter Kit for Arduino in action—rotating smoothly from 0° to 180°, and then reversing direction from 180° back to 0°.

Lesson 13 - IR Control LED

Introduction

In this lesson, we will delve into some advanced applications of infrared (IR) remote control technology. By identifying distinct button signals, we can use an IR remote to control the state of an LED. When Button 1 is pressed, the LED remains steadily on; pressing Button 2 causes the LED to blink; and pressing Button 3 turns the LED off.

Hardware Used in This Lesson:



Working Principle of Infrared Remote Control and Receiver

The working principle of an infrared (IR) remote control and receiver is based on the transmission of encoded infrared signals and their subsequent decoding and execution. When a button is pressed on the remote control, the internal encoding circuit converts the button input into a specific binary code format (such as NEC or RC-5 protocol). This code is then modulated—typically using a 38kHz carrier frequency to reduce interference—and transmitted as infrared light pulses (around 940nm wavelength) via an IR LED.On the receiving side, an IR receiver module (e.g., HS0038) detects the incoming IR signal using a photodiode. The received signal is then amplified, filtered, and demodulated to recover the original binary code. This decoded signal is interpreted by a microcontroller or main processing unit, which then performs the corresponding action—such as adjusting a servo angle or switching an appliance on or off. The entire process follows a closed loop: **button encoding → infrared modulation and transmission → photodetection and signal decoding → command execution**, enabling effective wireless control of electronic devices.

Working Principle of an LED

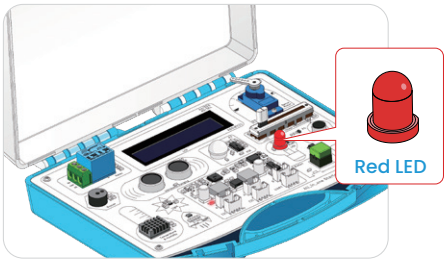
The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from a higher to a lower energy level, releasing excess energy in the form of photons—producing visible light. The color (wavelength) of the light depends on the bandgap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

When button 1 is pressed, the LED remains steadily ON.

When button 2 is pressed, the LED flashes.

When button 3 is pressed, the LED turns OFF.



Key Explanations

1. IR Receiver Initialization

```
#include <IRSendRev.h>
#include <TimerOne.h>
#define IR_PIN 2
```

- The IRSendRev library is included to handle infrared signal reception.
- The TimerOne library is included to implement timer functionality.
- The IR receiver pin IR_PIN is defined and connected to digital pin 2.

2. Variable Definitions

```
int LedPin = 10;
bool ledState = true;
bool timerRunning = false;
```

- LedPin: The control pin for the LED.
- ledState: Tracks the current state of the LED; true indicates the LED is ON, and false means it is OFF.
- timerRunning: A boolean used to track whether the timer is currently running.

3. Initialization Function

```
void setup() {
  Serial.begin(115200);
  while (!Serial);
  IR.Init(IR_PIN);
  Serial.println("init over");
  pinMode(LedPin, OUTPUT);
  Timer1.initialize(1000000);
  Timer1.attachInterrupt(toggleLED);
  Timer1.stop();
  timerRunning = false;
}
```

- **Initialize Serial Communication:**

```
Serial.begin(115200);
while (!Serial);
```

Serial.begin(115200); initializes the serial communication with a baud rate of 115200. The baud rate determines the data transmission speed over the serial port, here set to 115,200 bits per second.

while (!Serial); is a loop that waits for the serial connection to be established. This is necessary for some boards (like Arduino Leonardo) where the serial port takes some time to initialize. The loop blocks further execution until the serial connection is ready.

- **Initializing the Infrared Receiver Module**

```
IR.Init(IR_PIN);
Serial.println("init over");
```

IR.Init(IR_PIN);: calls the Init function from the IRSendRev library to initialize the infrared receiver module. IR_PIN specifies the digital pin connected to the infrared receiver, which is pin 2 in this case.

Serial.println("init over");: prints the message "init over" to the serial monitor, helping with debugging and confirming that the initialization was successful.

- **Timer initialization**

```
Timer1.initialize(1000000);
Timer1.attachInterrupt(toggleLED);
Timer1.stop();
timerRunning = false;
```

Timer1.initialize(1000000);: initializes the timer with a period of 1 second (1,000,000 microseconds). The TimerOne library allows you to set the timer interval in microseconds.

Timer1.attachInterrupt(toggleLED);: attaches the timer interrupt to the toggleLED function, meaning that every time the timer reaches the set period, the toggleLED function is automatically called.

Timer1.stop();: Stops the timer. At initialization, the timer does not start immediately but waits for further instructions from the program.

timerRunning = false;: Sets a global variable timerRunning to false, indicating that the timer is currently not running.

4. Main Loop Function

- **Check Infrared Data:**

```
if (IR.IsDta())
```

Use the IR.IsDta() function to check whether infrared data has been received. This function returns true if data is received; otherwise, it returns false.

- **Receive Infrared Data:**

```
byte length = IR.Recv(dta);
```

Use the IR.Recv(dta) function to receive the infrared data, storing the received data in the array dta. The variable length holds the length of the received data.

- **Execute Actions Based on Key Codes**

```
switch (dta[8])
```

Use a switch statement to perform corresponding actions based on the 9th byte of the received data (dta[8]), which usually represents the key code to distinguish different button presses.

- **Button 1**

```
case 48:
  Serial.println("[1]");
  digitalWrite(LedPin, HIGH);
  if (timerRunning) {
    stopTimer();
  }
  break;
```

If the received key code is 48 (Button 1):

• Print [1] to the serial monitor.

• Turn on the LED (set LedPin to HIGH).

• If the timer is running, call stopTimer() to stop the timer.

- **Button 2**

```
case 24:
  Serial.println("[2]");
  if (!timerRunning) {
    startTimer();
  }
  break;
```

If the received key code is 24 (Button 2):

• Print [2] to the serial monitor.

• If the timer is not running, call startTimer() to start the timer.

- **Button 3**

```
case 122:
  Serial.println("[3]");
  digitalWrite(LedPin, LOW);
  if (timerRunning) {
    stopTimer();
  }
  break;
```

If the received key code is 122 (Button 3):

- Print [3] to the serial monitor.
- Turn off the LED (set LedPin to LOW).
- If the timer is running, call stopTimer() to stop the timer.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the above code, you can further customize the functionality—for example, use the infrared remote control to operate and control other hardware devices based on different remote signals.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

Since this lesson involves using an infrared module that requires additional library files to operate properly, you need to add the appropriate libraries before uploading the code.

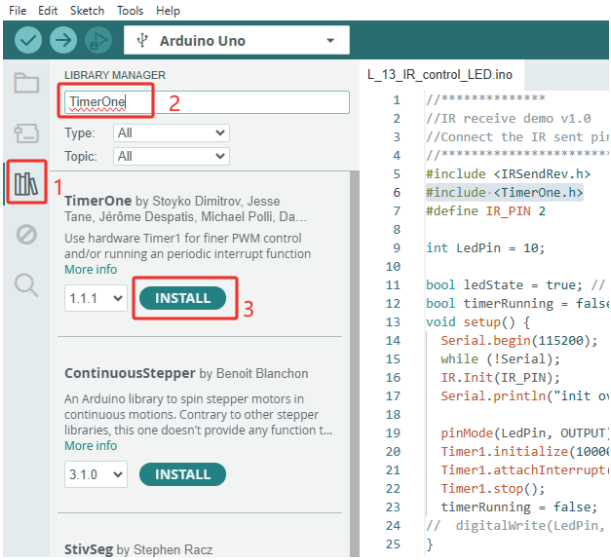
Note: The library for this infrared module is not available through the Arduino IDE's built-in library manager. You will need to download it from GitHub. However, we have prepared the necessary library for you — just click the link below to download it.

IRSendRev library files:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/libraries

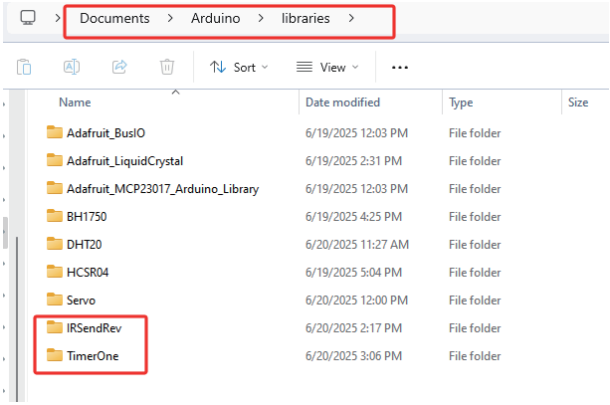
Next, we also need to install the required timer library. The TimerOne library provides support for timer functions. A timer is a hardware resource that can trigger interrupts at specified intervals, enabling timed tasks.

Follow the steps shown in the diagram to download the TimerOne library (version 1.1.1) so that you can use the timer functionality.

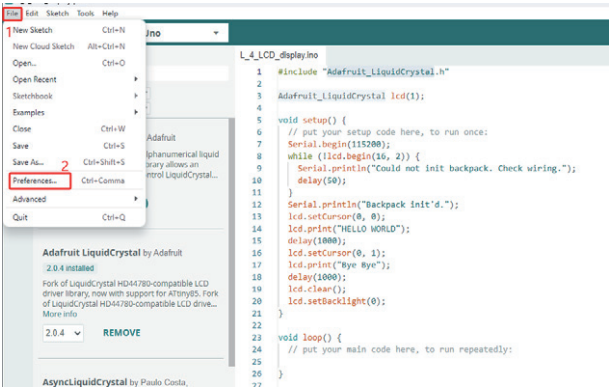


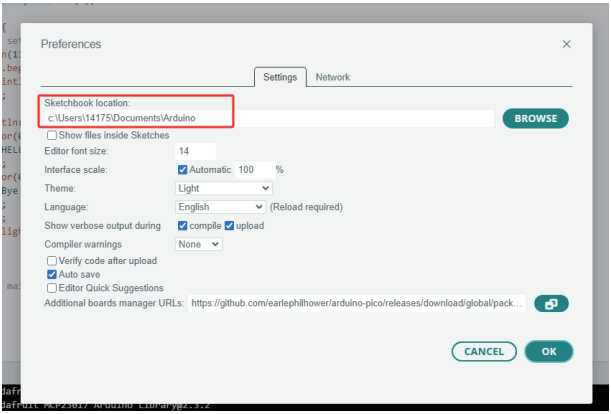
After installation, you can find the downloaded library files in this path.

Remember to move the downloaded infrared receiver library file to the specified path so that the code can recognize the library file you added during the upload process.



(The download path is determined by the path set in the Arduino IDE.)





The library files have now been downloaded. Next, you can upload the code we will be using.

THe code upload steps for this lesson are the same as those in Lesson 1. Please refer to Lesson 1 for detailed instructions!

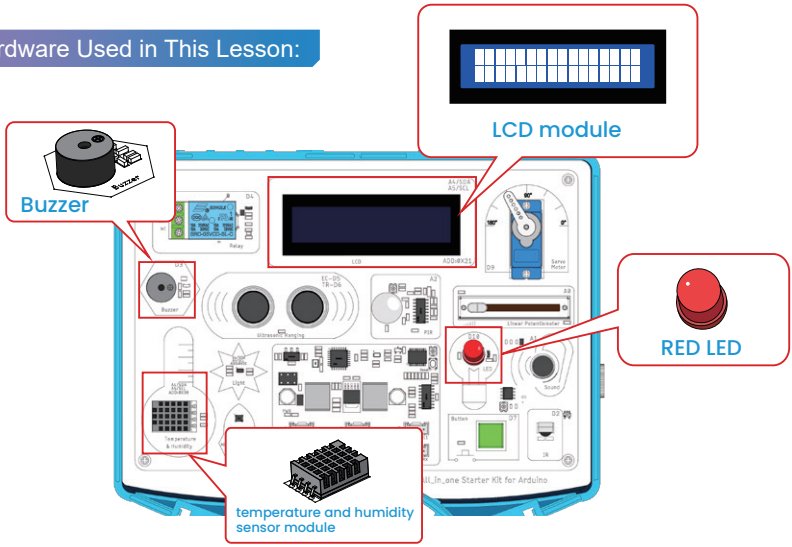
After the upload is successful, you will be able to use the infrared remote control to operate the LED on the All_in_one Starter Kit for Arduino to perform the corresponding functions.

Lesson 14 - Weather Reminder

Introduction

In this lesson, we will learn about environmental monitoring and intelligent control technology based on the DHT20 temperature and humidity sensor. By real-time acquisition of temperature and humidity data, we will achieve three interactive functions: dynamically updating the LCD display content according to temperature and humidity thresholds (such as showing values and status icons), automatically controlling the LED indicator light (lighting it up as a warning under high temperature and humidity), and triggering the buzzer alarm (sounding when preset thresholds are exceeded). The course will thoroughly explain the complete implementation process of sensor data reading, threshold logic judgment, and multi-device coordinated control, enabling you to master the core technologies of data acquisition and actuator coordination in environmental monitoring systems.

Hardware Used in This Lesson:



Working Principle of the LCD Screen

The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices.

Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

The backlight module (usually LEDs) provides illumination to ensure clear visibility even in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Working Principle of Temperature and Humidity Sensors

A temperature and humidity sensor is an electronic device used to measure ambient temperature and humidity levels. Its working principle relies on the response characteristics of sensitive elements to environmental physical quantities:For temperature measurement, components such as thermistors, thermocouples, or semiconductor-based temperature sensors are commonly used as sensing elements. A thermistor, for instance, exhibits a predictable change in resistance as temperature varies — for example, a negative temperature coefficient (NTC) thermistor decreases in resistance as temperature increases. This change in resistance is typically converted into a voltage signal using a Wheatstone bridge or similar circuit, then processed through analog-to-digital conversion (ADC) and calibration algorithms to produce an accurate temperature reading.

Working Principle of an LED

The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from a higher to a lower energy level, releasing excess energy in the form of photons—producing visible light. The color (wavelength) of the light depends on the bandgap of the semiconductor material. This process is a direct application of electroluminescence.

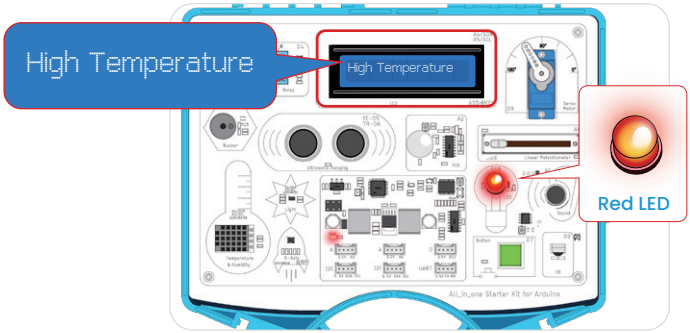
Working Principle of a Buzzer

A buzzer is an electronic component that converts electrical signals into sound signals, operating based on either electromagnetic induction or the piezoelectric effect. An electromagnetic buzzer contains a coil, a magnet, and a vibrating diaphragm. When

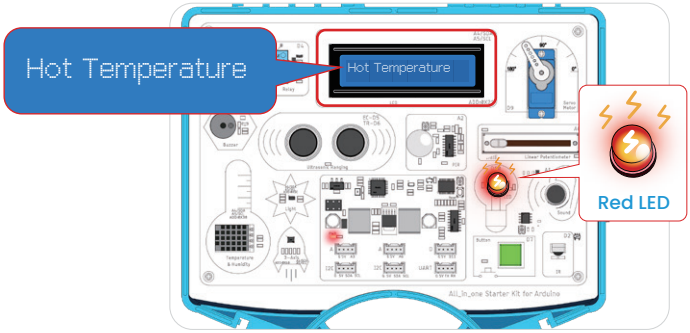
current flows through the coil, it generates a magnetic field that interacts with the permanent magnet, causing the diaphragm to vibrate and produce sound. The presence and pitch of the sound can be controlled by modulating the current's frequency and duration. In contrast, a piezoelectric buzzer uses piezoelectric materials (such as piezoceramics) that deform mechanically when an alternating voltage is applied—a phenomenon known as the inverse piezoelectric effect. This deformation drives the diaphragm to vibrate and emit sound at a specific frequency. Both types of buzzers require external circuitry for proper operation and are commonly used in alarms, electronic alerts, and notification systems.

Operation Effect Diagram

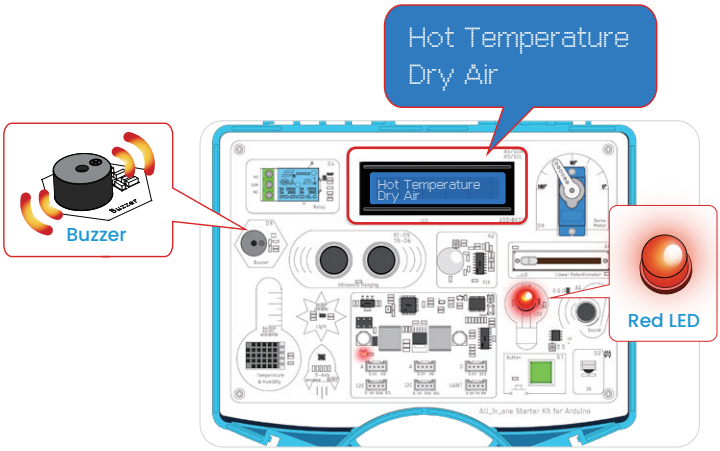
When the temperature exceeds 25°C, the red LED will light up, and the LCD will display a “High Temperature” warning.



When the temperature exceeds 30°C, the red LED will blink, and the LCD will display a “Hot Temperature” warning.



When the humidity drops below 40%, the buzzer will sound, and the LCD will display a “Dry Air” warning.



Key Explanations

1. Initialization Function (setup)

```
void setup() {
  Serial.begin(115200);
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  Serial.println("Backpack init'd.");
  Wire.begin();
  DHT.begin();
}
```

• Initialize serial communication: Set the baud rate to 115200 for debugging and monitoring.

• Initialize the I2C interface: Wire.begin() initializes the I2C interface; the DHT20 sensor communicates with the Arduino via this interface.

• Initialize the DHT20 sensor: DHT.begin() initializes the DHT20 temperature and humidity sensor.

2. Main Loop Function (loop)

- **Read DHT20 Data:**

```
if (millis() - DHT.lastRead() >= 1000) {
  int status = DHT.read();
```

• Use the millis() function together with DHT.lastRead() method to ensure that data from the DHT20 sensor is read once every 1 second.

• Call DHT.read() method to read temperature and humidity data, which also updates the sensor's internal status.

- **Control LED and LCD Display Based on Temperature**

```
if (value2 > 25 && value2 <= 30) {
  LCD_print("High Temperature", " ");
  digitalWrite(LedPin, HIGH);
} else if (value2 > 30) {
  LCD_print("Hot Temperature", " ");
  LED_State = !LED_State;
  digitalWrite(LedPin, LED_State);
} else {
  LCD_print("Tem:" + String(value2) + "C", "Hum:" + String(value1) + "%");
  digitalWrite(LedPin, LOW);
}
```

▸ Control the LED status and LCD display content according to the temperature value.

- **Control LED and LCD Display Based on Temperature**

```
if (value2 > 25 && value2 <= 30) {
  LCD_print("High Temperature", " ");
  digitalWrite(LedPin, HIGH);
}
```

If the temperature is between 25°C and 30°C, display "High Temperature" on the LCD and turn on the LED.

- **Control LED and LCD Display Based on Temperature**

```
if (value2 > 30) {
  LCD_print("Hot Temperature", " ");
  LED_State = !LED_State;
  digitalWrite(LedPin, LED_State);
}
```

If the temperature exceeds 30°C, display "Hot Temperature" and make the LED blink.

```
 else {
  LCD_print("Tem:" + String(value2) + "C", "Hum:" + String(value1) + "%");
  digitalWrite(LedPin, LOW);
}
```

If the temperature is below 25°C, display the normal temperature and humidity readings and turn off the LED.

According to humidity control the buzzer

```
if (value1 < 40) {
  lcd.setCursor(0, 1);
  lcd.print("Dry Air");
  tone(buzzerPin, 1300);
} else {
  noTone(buzzerPin);
}
```

- **Code breakdown:**

```
if (value1 < 40) {
  lcd.setCursor(0, 1);
  lcd.print("Dry Air");
  tone(buzzerPin, 1300);
}
```

If the humidity is below 40%, display "Dry Air" on the LCD and sound the buzzer alarm.

```
 else {
  noTone(buzzerPin);
}
```

If the humidity is greater than or equal to 40%, stop the buzzer.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the above code, you have made great progress—you now know how to set thresholds to control more hardware devices, and your ability to handle control logic has improved significantly!

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

This lesson still requires the use of the DHT20 library (version 0.3.1) and the Adafruit_LiquidCrystal library (version 2.0.4). Please refer to Lesson 11 for details.

(The code upload steps for this lesson are the same as those in Lesson 11; please follow the upload instructions provided there!)

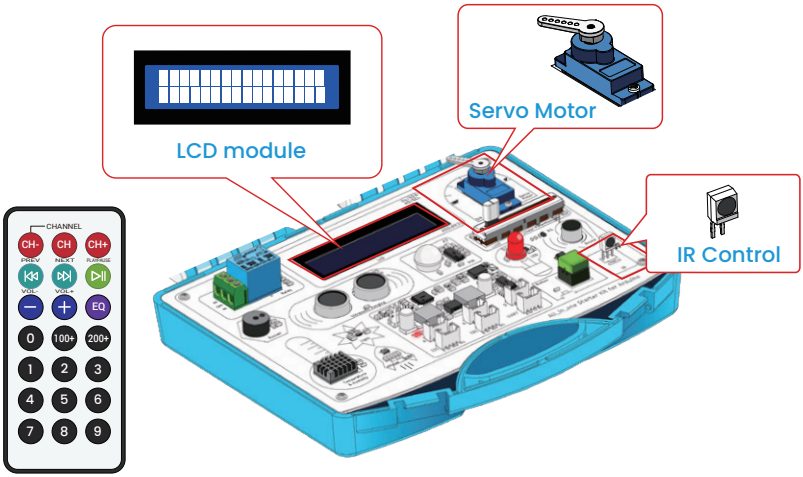
After the upload is successful, you will be able to see the DHT20 temperature and humidity sensor on the All_in_one Starter Kit for Arduino continuously collecting real-time temperature and humidity data. When the preset thresholds are reached, corresponding alert messages will be displayed on the LCD screen.

Lesson 15 - Servo Angle Control

Introduction

In this lesson, we will learn how to precisely control the rotation angle of a servo motor using an infrared remote control and display the angle value in real-time on an LCD screen. The course will provide an in-depth explanation of infrared signal reception and decoding, the principles of servo motor angle control, and methods for visualizing data on the LCD. You will master the complete technical process—from remote command transmission to motor angle adjustment and real-time data display—enabling seamless integration of human-machine interaction control with dynamic data visualization.

Hardware Used in This Lesson:



Working Principle of Infrared Remote Control and Receiver

The working principle of an infrared (IR) remote control and receiver is based on the transmission of encoded infrared signals and their subsequent decoding and execution. When a button is pressed on the remote control, the internal encoding circuit converts the button input into a specific binary code format (such as NEC or RC-5 protocol). This code is then modulated—typically using a 38kHz carrier frequency to reduce interference—and transmitted as infrared light pulses (around 940nm wavelength) via an IR LED. On the receiving side, an IR receiver module (e.g., HS0038) detects the incoming IR signal using a photodiode. The received signal is then amplified, filtered, and demodulated to recover the original binary code. This decoded signal is interpreted by a

microcontroller or main processing unit, which then performs the corresponding action—such as adjusting a servo angle or switching an appliance on or off.The entire process follows a closed loop: **button encoding → infrared modulation and transmission → photodetection and signal decoding → command execution**, enabling effective wireless control of electronic devices.

Working Principle of the LCD Screen

The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices. Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

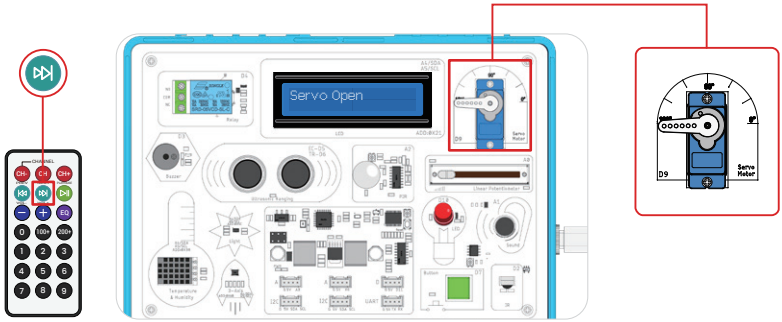
The backlight module (usually LEDs) provides illumination to ensure clear visibility even in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Working principle of a servo motor

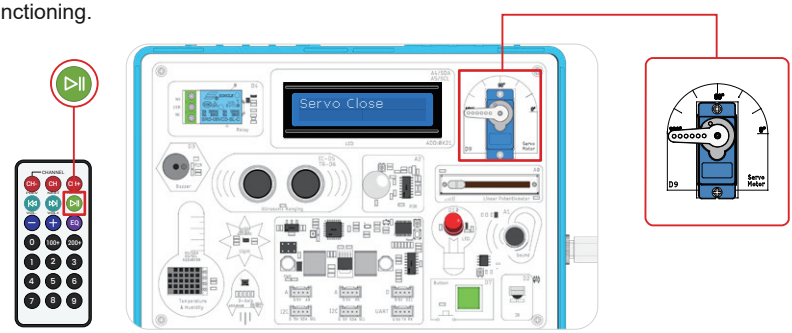
A servo motor is an angular position control actuator that integrates the principles of a servo system, operating based on a closed-loop feedback control mechanism. When a control signal—usually a Pulse Width Modulation (PWM) signal—is input, the pulse width corresponds to a target angle (for example, 1 ms pulse width corresponds to 0°, 1.5 ms to 90°, and 2 ms to 180°). The control circuit compares the input signal with the real-time angle feedback from a potentiometer. If there is a deviation, it drives the motor to rotate. The motor, through a gear reduction mechanism, turns the output shaft, while the potentiometer rotates synchronously with the output shaft, changing its resistance to reflect the current angle and feeding this back to the control circuit. This closed-loop system continuously adjusts the motor's direction and speed until the actual output shaft angle matches the target angle, stopping movement to achieve precise angular control within ±0.5°.

Operation Effect Diagram

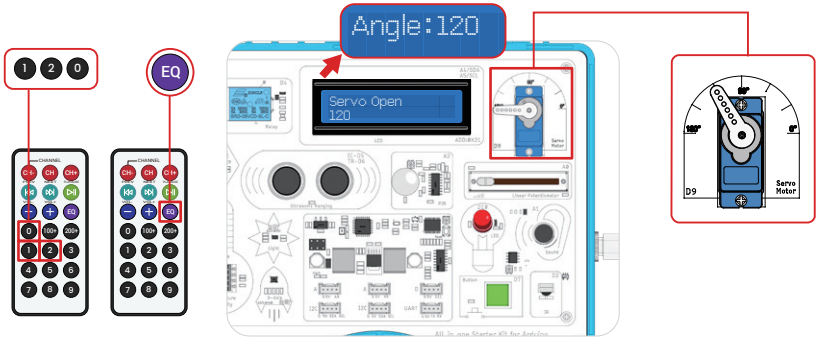
When the [NEXT] button on the remote control is pressed, the servo motor is activated and automatically rotates to 180°.



When the [PLAY/PAUSE] button is pressed, the servo motor is deactivated and stops functioning.



After you input the desired angle to control, pressing the [EQ] button will make the servo motor move to the specified angle (provided the servo motor is currently activated).



Key Explanations

1. Setup function initialization:

```
void setup() {
  Serial.begin(115200);
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  Serial.println("Backpack init'd.");
  IR.Init(IR_PIN);
  Serial.println("init over");
  num1 = "Angle:";
  lcd.setCursor(0, 0);
  lcd.print(num1);
}
```

• Initialize serial communication with a baud rate of 115200 for debugging and monitoring.

• Initialize the LCD screen; if initialization fails, print an error message and delay 50 milliseconds repeatedly until successful.

• Initialize the infrared receiver module by calling IR.Init(IR_PIN), where IR_PIN is defined as digital pin 2.

• Initialize LCD display:Set the LCD to show "Angle:" on the first line, which will be used to display the servo motor's angle.

2. Main Loop Function (loop)

- **Read Infrared Data:**

```
if (IR.IsDta()) {
  byte length = IR.Recv(dta);
```

Use IR.IsDta() to check if there is any infrared data received.

Use IR.Recv(dta) to receive the infrared data and store it in the array dta.

Execute Actions Based on Button Code:

- **Button 2: Control the servo to turn on**

```
case 2:   Serial.println("[NEXT]");
  myservo.attach(9, 600, 2520);
  myservo.write(180);
  lcd.setCursor(0, 0);
  lcd.print("              ");
  lcd.setCursor(0, 0);
  lcd.print("Servo Open");
  break;
```

Serial.println("[NEXT]"): Print the received button info to the serial monitor for debugging and monitoring.

myservo.attach(9, 600, 2520): Attach the servo to digital pin 9 and set the pulse width range from 600 to 2520 microseconds, which is the standard control range for the servo.

myservo.write(180): Set the servo angle to 180 degrees, which usually represents the "open" state of the servo.

lcd.setCursor(0, 0): Position the LCD cursor at the start of the first line.

lcd.print(" "): Clear the old content on the first line.

lcd.print("Servo Open"): Display "Servo Open" on the first line to indicate the servo is now active.

- **Button 194: Control Servo to Turn Off**

```
case 194: Serial.println("[PLAY/PAUSE]");
  myservo.detach();
  lcd.setCursor(0, 0);
  lcd.print("              ");
  lcd.setCursor(0, 0);
  lcd.print("Servo Close");
  break;
```

Serial.println("[PLAY/PAUSE]"): Prints the received button info to the serial monitor for debugging and monitoring.

myservo.detach(): Detaches the servo from the control pin, effectively stopping signal output. The servo will stay at its current angle without active control.

lcd.setCursor(0, 0): Positions the LCD cursor at the beginning of the first row.

lcd.print(" "): Clear the old content on the first line.

lcd.print("Servo Close"): Display "Servo Close" on the first line, indicating that the servo motor has been turned off.

- **Button 144：Set Servo Angle**

```
case 144: Serial.println("[EQ]");
  if (num.toInt() >= 0 && num.toInt() <= 180) {
    Serial.println(num);
    num1 = "Angle:";
    myservo.write(num.toInt());
    num = "";
    value = myservo.read();
    num1 += String(value);
    lcd.setCursor(0, 0);
    lcd.print("              ");
    lcd.setCursor(0, 0);
    lcd.print(num1);
    break;
  } else {
    num1 = "Angle:";
    num1 += "Error";
    num = "";
    lcd.setCursor(0, 0);
    lcd.print("              ");
    lcd.setCursor(0, 0);
    lcd.print(num1);
    break;
  }
```

• Serial.println("[EQ]")： Print the received button information via the serial port for debugging and monitoring.

• if (num.toInt() >= 0 && num.toInt() <= 180)： Check if the input numeric string is within the range of 0 to 180 degrees.

• If input value is valid:

Serial.println(num)： Print the input numeric string via the serial port.

num1 = "Angle:": Initialize the display string.

myservo.write(num.toInt()): Set the servo angle to the input value.

num = "": Clear the input numeric string.

value = myservo.read(): Read the current servo angle.

num1 += String(value): Append the current servo angle to the display string.

lcd.setCursor(0, 0): Position the LCD cursor at the start of the first row.

lcd.print(" "): Clear the old content on the first row.

lcd.print(num1): Display the current servo angle on the first row.

If input value is invalid:

num1 = "Angle:": Initialize the display string.

num1 += "Error": Append an error message.

num = "": Clear the input numeric string.

lcd.setCursor(0, 0): Position the LCD cursor at the start of the first row.

lcd.print(" "): Clear the old content on the first row.

lcd.print(num1): Display the error message on the first row.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the above code, it is believed that you have gained a deeper understanding of infrared control and a more intuitive cognition of its working principle.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

Since the LCD screen used in this lesson requires additional library files for driving, we need to add appropriate library files to ensure the code runs normally before uploading it.

Please refer to the library installation method in Lesson 4 to prepare the **Adafruit_LiquidCrystal library (version 2.0.4)**.

Since the infrared module used in this lesson requires additional library files to function, you must add the appropriate libraries before uploading the code.

Note: The library for this infrared module cannot be found via the Arduino IDE's Library Manager. You need to download it from GitHub, but we have prepared it for you. Click the link below to download:

IRSendRev Library:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/libraries

Now that the library files have been downloaded, you can proceed to upload the code used in this lesson.

Remember to move the downloaded infrared receiver library file to the specified path so that the code can recognize the library file you added during the upload process.

The code upload steps for this lesson are consistent with the process in the first lesson. Please refer to the first lesson for guidance!

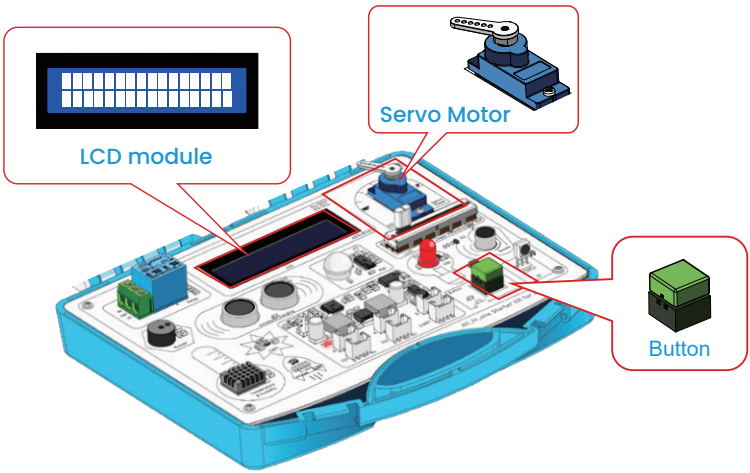
After the upload is successful, you can use the infrared remote control to make the LCD on the All_in_one Starter Kit for Arduino display the key values you input, and you can also intuitively see the servo running at the specified angle.

Lesson 16 - Polite Automatic Door

Introduction

In this lesson, we will delve into interactive control techniques for servo motors, focusing on dynamic angle adjustment through button input. The course will cover key concepts such as button state detection, precise servo angle control, and action logic design. You will implement a complete interactive process: “press the button — servo motor rotates to open the door; release the button — motor resets to close the door.” Through hands-on practice, you will gain a solid understanding of the coordination between mechanical motion and electronic control, as well as the application of momentary triggers and sustained state control in servo systems. This foundation is essential for developing motor control in smart home systems, automation equipment, and similar applications.

Hardware Used in This Lesson:



Working Principle of Button Control

A button generates signals through mechanical contacts or capacitive sensing. Mechanical buttons rely on a metal dome or spring mechanism to create HIGH or LOW logic levels when pressed or released, often requiring a debounce circuit to eliminate signal fluctuations. Capacitive buttons detect changes in capacitance between electrodes. The microcontroller (MCU) scans matrix buttons by reading the voltage levels of rows and columns, or by reading ADC values, to determine which button is pressed. The system distinguishes between short and long presses through software-based timing.

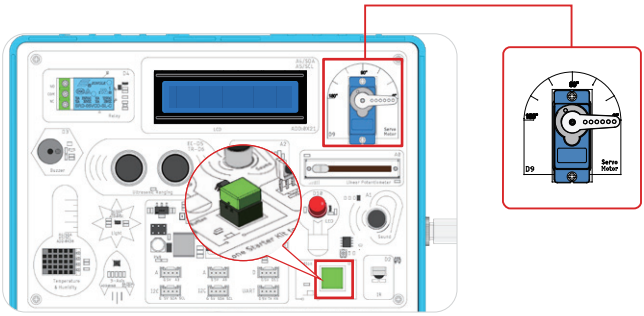
Working principle of a servo motor

A servo motor is an angular position control actuator that integrates the principles of a servo system, operating based on a closed-loop feedback control mechanism. When a control signal—usually a Pulse Width Modulation (PWM) signal—is input, the pulse width corresponds to a target angle (for example, 1 ms pulse width corresponds to 0°, 1.5 ms to 90°, and 2 ms to 180°). The control circuit compares the input signal with the real-time angle feedback from a potentiometer. If there is a deviation, it drives the motor to rotate. The motor, through a gear reduction mechanism, turns the output shaft, while the potentiometer rotates synchronously with the output shaft, changing its resistance to reflect the current angle and feeding this back to the control circuit. This closed-loop system continuously adjusts the motor's direction and speed until the actual output shaft angle matches the target angle, stopping movement to achieve precise angular control within ±0.5°.

Operation Effect Diagram

• When the button is pressed and held, the servo motor rotates to 180°, simulating a door-opening function, and the LCD screen displays "Welcome."

• When the button is released, the servo motor returns to 0°, simulating a door-closing function.



Key Explanations

1. Hardware Initialization and Connections

```
Servo myservo;
int buttonPin = 7;
Adafruit_LiquidCrystal lcd(1);
```

This section of the code defines three core hardware components: the servo motor, the button, and the LCD screen. It also specifies how they are connected to the microcontroller.

2. Servo Motor Parameter Configuration

```
myservo.attach(9, 600, 2520);
```

- **Parameter Explanation:**
  - 9: Signal pin connected to the servo.
  - 600: Minimum pulse width in microseconds, corresponding to the 0° position.
  - 2520: Maximum pulse width in microseconds, corresponding to the 180° position.
- **Extended Angle Principle:** The standard pulse range for servo motors is typically 1000–2000 µs. However, some servos support an extended range (e.g., 600–2520 µs), which allows rotation beyond 180°, depending on the servo's mechanical design.

3. Button Detection and State Control

```
if (!digitalRead(buttonPin))
{
  myservo.write(180);
  lcd.setCursor(0, 0);
  lcd.print("Welcome");
}
else
{
  myservo.write(0);
  lcd.setCursor(0, 0);
  lcd.print("              ");
}
```

This section represents the core logic of the code. It controls the servo motor's rotation and the LCD display based on the button's state:

```
if (!digitalRead(buttonPin)) { ... }
```

- **Voltage Logic:**

  When the button is **not pressed**, pin 7 is held HIGH (logic 1) via an internal pull-up resistor.

  When the button **is pressed**, the pin is pulled LOW (logic 0) to ground, and !digitalRead(buttonPin) evaluates to true.

  • When the button is pressed, the servo rotates to 180°, simulating an "open door" motion, and the LCD displays "Welcome".

  • When the button is released, the servo returns to 0°, simulating a "close door" motion, and the LCD is cleared (by printing a blank space).

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After learning the above code, you can adjust the functions based on the original code: simply press the button to make the servo motor operate (no need to hold the button continuously), and press the button again to restore the servo to its initial angle.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

Since the servo motor used in this lesson requires additional library files to function properly, you need to add the appropriate libraries before uploading the code to ensure it runs correctly.

The Servo library (version 1.2.2) for the servo motor was explained in detail in Lesson 12. You can refer to the installation steps for the servo library in Lesson 12.

Secondly, the LCD screen used in this lesson also requires additional library files for driving. Therefore, before uploading the code, we need to add appropriate library files to ensure the normal operation of the code.

The Adafruit_LiquidCrystal library (version 2.0.4) for the LCD screen was explained in detail in Lesson 4. You can refer to the installation steps for the LCD library in Lesson 4.

Now that the library files have been downloaded, you can proceed to upload the code used in this lesson.

The code upload steps for this lesson are the same as those in the first lesson. Please refer to the first lesson for details!

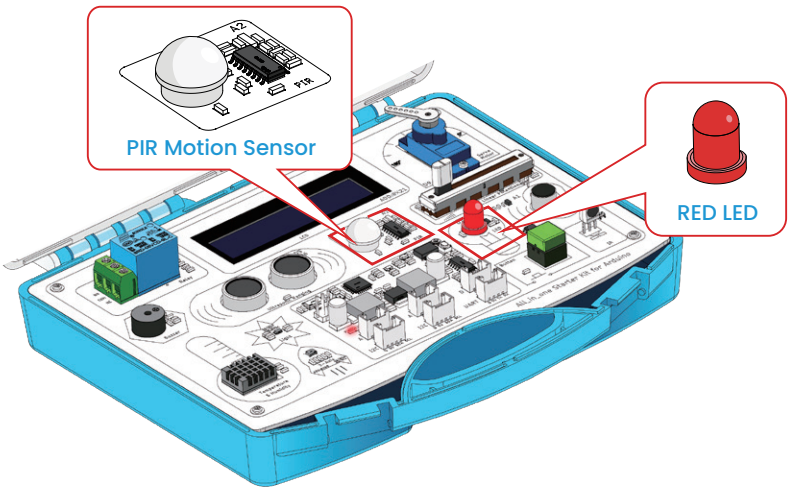
After the upload is successful, pressing the button on the All-in-one Starter Kit for Arduino will make the servo rotate to 180°, and the screen will display "Welcome".

Lesson 17 - PIR Control Light

Introduction

In this chapter, we will delve into advanced application scenarios of the PIR motion sensor, with a focus on implementing intelligent control of an LED lighting system based on motion detection signals. By analyzing the characteristics of sensor data and the logic behind lighting control, you will master the complete closed-loop process—from signal acquisition and status evaluation to execution control—laying a solid technical foundation for building smart environment-sensing systems.

Hardware Used in This Lesson:



Working Principle of PIR Motion Sensor

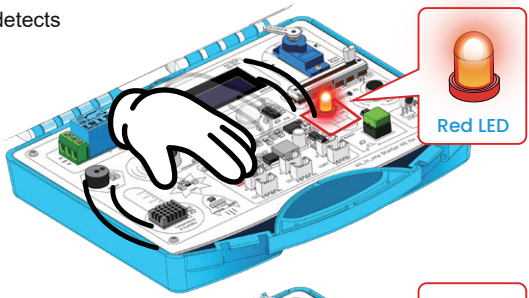
A PIR (Passive Infrared) motion sensor operates by detecting changes in infrared radiation emitted by humans or animals. Its core component is a pyroelectric sensor, which senses variations in infrared levels when an object moves within its detection range. These changes are caused by the difference in thermal radiation between the moving object and the background. The detected signals are then amplified and converted into electrical signals, resulting in a change in output level. This enables non-contact motion detection, making PIR sensors widely used in applications such as security alarms and smart home systems for automatic human presence sensing.

Working Principle of an LED

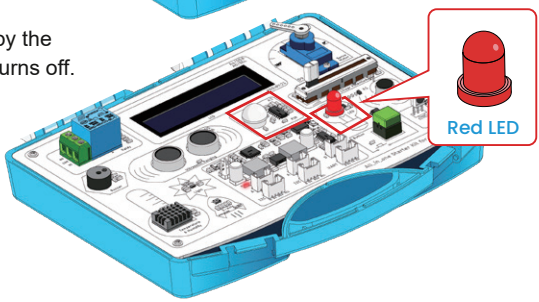
The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from a higher to a lower energy level, releasing excess energy in the form of photons—producing visible light. The color (wavelength) of the light depends on the bandgap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

When the PIR motion sensor detects movement, the LED turns on.



When no motion is detected by the PIR motion sensor, the LED turns off.



Key Explanations

1. Hardware Connection and Variable Definition

```
#define PIR_PIN A2
static int oldState = 0;
int LedPin = 10;
```

PIR Sensor: Connected via analog pin A2 to read signals, though it actually uses digital signals (HIGH/LOW).

State Variable: oldState is used to store the previously detected state, enabling the program to determine whether the current state has changed.

2. Initialization Setup (setup() function)

```
void setup() {
  pinMode(PIR_PIN, INPUT);
  pinMode(LedPin, OUTPUT);
}
```

Pin Mode Configuration: Set the PIR sensor pin as input, and the LED pin as output.

3. Main Loop Logic (loop() function)

```
void loop() {
  byte state = digitalRead(PIR_PIN);
  if( state && oldState != state ) {
    Serial.println("[-] Motion detected!");
    oldState = state;
    digitalWrite(LedPin, HIGH);
    delay(5000);
  }
  else if( !state && oldState != state ) {
    Serial.println("[-] No Motion!");
    digitalWrite(LedPin, LOW);
    oldState = state;
  }
  delay(20);
}
```

- **State Detection:**
  - state = digitalRead(PIR_PIN): Read the PIR sensor status (HIGH = motion detected, LOW = no motion).
  - oldState != state: Check if the state has changed to avoid redundant triggers.
- **Motion Detected Branch:**
  - if(state && oldState != state): Triggered when motion is detected and the state changes.
  - Log output: "Motion detected!"
  - Turn on the LED (HIGH).
  - Critical Delay: delay(5000) keeps the LED on for 5 seconds, ignoring new motion detections during this period.

- **No Motion Branch:**
  - else if(!state && oldState != state): Triggered when motion stops and the state changes.
  - Log output: "No Motion!"
  - Turn off the LED (LOW).

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, you can enhance the functionality by improving the code's responsiveness. This allows the PIR sensor to continuously monitor for motion without being blocked by delays.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so that we can see the hardware functions in action.

The code upload steps for this lesson are the same as those in Lesson 1. Please refer to Lesson 1 for detailed instructions!

Once the upload is successful, you will see that the LED on the All-in-one Starter Kit for Arduino lights up when the PIR sensor detects your movement.

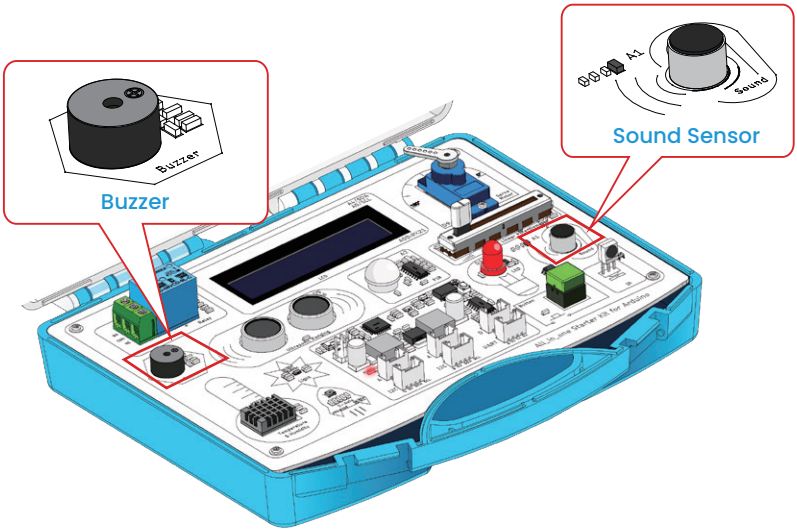
Implementation: You'll observe that when you wave your hand or walk within the sensing range of the PIR motion sensor, the LED will turn on for 5 seconds. If there is no motion detected within the range, the LED will remain off.If this behavior does not occur as expected, please double-check that the code has been uploaded to the board successfully.

Lesson 18 - Sound Reminder

Introduction

In this lesson, we will learn the practical operation of a sound sensor. By using the sound sensor to detect the ambient noise level, the system will trigger a buzzer to sound an alarm whenever a sound is detected. This demonstrates a simple sound-activated alert function.

Hardware Used in This Lesson:



Working Principle of the Sound Sensor

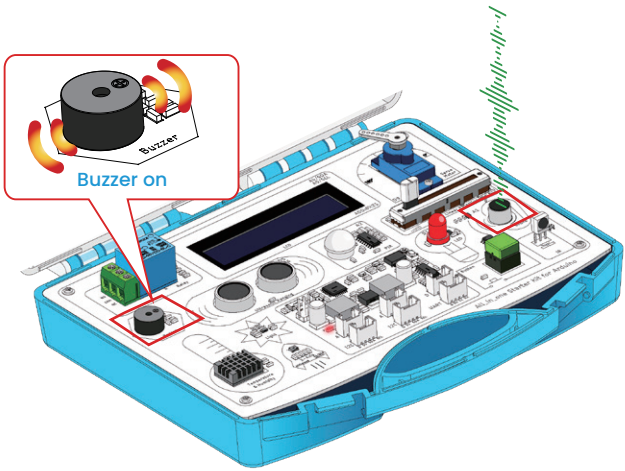
A sound sensor primarily uses a microphone or an electret condenser microphone to capture ambient sound waves. These sound waves cause the diaphragm inside the sensor to vibrate, which results in changes in internal capacitance or resistance. This variation converts the sound signal into an electrical signal. The signal is then amplified by an internal circuit and processed through filtering and conditioning to produce either an analog voltage or a digital signal corresponding to the sound intensity.When the detected sound reaches a preset threshold, the sensor activates a corresponding circuit—such as triggering a buzzer—to respond to the environmental sound. Sound sensors are commonly used in applications like voice-activated switches and alarm systems.

Working Principle of a Buzzer

A buzzer is an electronic component that converts electrical signals into sound signals, operating based on either electromagnetic induction or the piezoelectric effect. An electromagnetic buzzer contains a coil, a magnet, and a vibrating diaphragm. When current flows through the coil, it generates a magnetic field that interacts with the permanent magnet, causing the diaphragm to vibrate and produce sound. The presence and pitch of the sound can be controlled by modulating the current's frequency and duration. In contrast, a piezoelectric buzzer uses piezoelectric materials (such as piezoceramics) that deform mechanically when an alternating voltage is applied—a phenomenon known as the inverse piezoelectric effect. This deformation drives the diaphragm to vibrate and emit sound at a specific frequency. Both types of buzzers require external circuitry for proper operation and are commonly used in alarms, electronic alerts, and notification systems.

Operation Effect Diagram

When the sound sensor detects a value exceeding the threshold of 400, the buzzer will sound:



Once the program runs successfully, you'll observe the following behavior:If the surrounding noise is too loud, or you shout directly at the sound sensor, the buzzer will emit a one-second "beep" to alert you to lower the volume.If the noise continues to stay above the threshold, the buzzer will keep beeping repeatedly.When the environment becomes quiet again and the sound level drops below the threshold, the buzzer will stop automatically.

Key Explanations

1. Hardware Connection and Variable Definition

```
#define SOUND_PIN A1
buzzerPin = 3;
```

Sound Sensor: Outputs an analog signal (usually 0-1023), where a higher value indicates louder sound.

Buzzer: Driven by the tone() function and must be connected to a PWM-capable pin.

2. Initialization (setup function)

```
void setup() {
  pinMode(SOUND_PIN, INPUT);
  pinMode(buzzerPin, OUTPUT);
}
```

Set the pin modes clearly: the sound sensor pin as input and the buzzer pin as output.

3. Main Loop Logic (loop() function)

```
void loop() {
  if (analogRead(SOUND_PIN) >= 400) {
    Serial.println("[-] Detect Sound!");
    tone(buzzerPin, 1300);
    delay(1000);
    while (1) {
      if (analogRead(SOUND_PIN) >= 400) {
        Serial.println(analogRead(SOUND_PIN));
      } else {
        noTone(buzzerPin);
        break;
      }
    }
  }
}
```

- **Sound Detection and Initial Response:**

  analogRead(SOUND_PIN) >= 400: Check if the sound level exceeds the threshold (400).

  tone(buzzerPin, 1300): Activate the buzzer to emit a 1300Hz tone.

  delay(1000): Force a 1-second delay during which other events are not processed.

- **Continuous Monitoring:**

  while (1): Enter an infinite loop to continuously monitor the sound status.

  • When the sound remains above the threshold:

  Output the current sound value via serial communication.

  • When the sound falls below the threshold:

  noTone(buzzerPin): Stop the buzzer.

  break: Exit the loop and wait for the next sound trigger.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the above code, you can modify its functionality—for example, adjust the sound threshold value and customize the trigger mechanism when the threshold is exceeded based on the existing code.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

The code upload steps for this lesson are the same as those in Lesson 1. Please refer to Lesson 1 for detailed instructions!

After the upload is successful, you will be able to hear the buzzer on the All_in_one Starter Kit for Arduino sound an alarm when the sound sensor detects noise above the threshold.

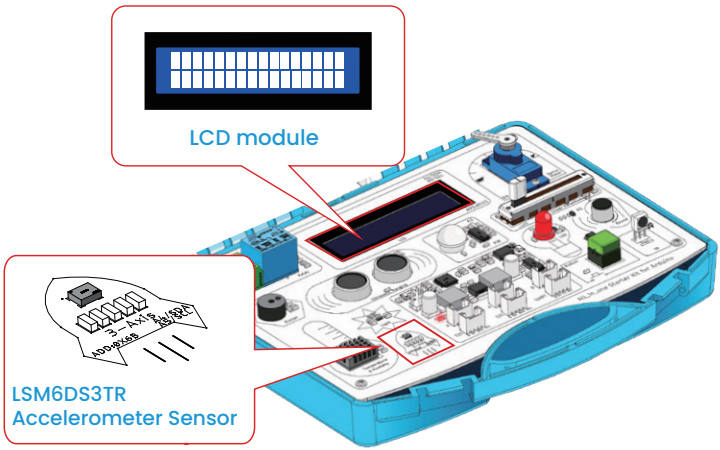
Implementation:You will notice that if the surrounding noise is too loud or you shout directly at the sound sensor, the buzzer will beep for one second to remind you to lower the volume. If the noise remains high, the buzzer will continue beeping. When there is no sound, the buzzer will stop sounding.

Lesson 19 - Calculation of Acceleration

Introduction

In this lesson, we will learn how to operate the LSM6DS3TR sensor. By moving the All_in_one Starter Kit for Arduino, you will see data changes on the display. This sensor can detect and calculate acceleration values in different directions, helping us understand how to acquire and use these acceleration data to implement various interactive functions.

Hardware Used in This Lesson:

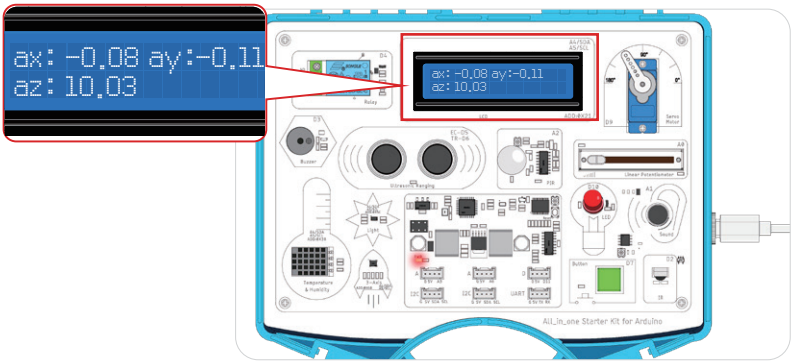


Working Principle of the LSM6DS3TR Accelerometer Sensor

The LSM6DS3TR sensor operates based on Micro-Electro-Mechanical Systems (MEMS) technology. Its internal accelerometer measures linear motion by detecting capacitance changes caused by a proof mass under acceleration. The gyroscope, on the other hand, utilizes the Coriolis effect, calculating angular velocity from phase shifts in vibrating structures during rotation. The sensor converts the analog signals sensed by the MEMS structures into digital signals, which are then processed by internal digital filters. It outputs three-axis acceleration and three-axis angular velocity data via I2C or SPI interfaces. Additionally, its built-in motion detection logic can automatically identify states such as free fall or step counting based on preset thresholds, operating without continuous intervention from the main controller, thereby enabling low-power operation.

Operation Effect Diagram

When you move the All_in_one Starter Kit for Arduino, you will see the acceleration data for the three axes displayed on the screen:



After successful operation, the LCD will show the accelerometer values for the X, Y, and Z axes. When you move the accelerometer quickly along any axis, you will observe the corresponding accelerometer value changing accordingly on that axis.

Key Explanations

1. Data Storage

```
float accel[3];
```

accel[0~2]: Corresponding to acceleration values of X, Y, Z axes (unit: m/s²)

2. LCD Display Control

Screen Clear Function

```
void LCD_print(String txt1, String txt2)
{
  lcd.setCursor(0, 0); lcd.print("                ");
  lcd.setCursor(0, 1); lcd.print("                ");
  lcd.setCursor(0, 0); lcd.print(txt1);
  lcd.setCursor(0, 1); lcd.print(txt2);
}
```

Implementation Principle:

Overwrite original characters with spaces (16 spaces correspond to the LCD width).

```
lcd.setCursor(0, 0); lcd.print("                ");
lcd.setCursor(0, 1); lcd.print("                ");
```

Clear and initialize both the first and second rows of the LCD screen.

```
lcd.setCursor(0, 0); lcd.print(txt1);
lcd.setCursor(0, 1); lcd.print(txt2);
```

Display the desired text content on the first and second rows.

3. Initialization Function (setup)

```
void setup() {
  Wire.begin();
  Serial.begin(115200);
  Wire.begin();
  writeRegister(CTRL1_XL, 0x48);
  writeRegister(CTRL2_G, 0x40);
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  lcd.setCursor(0, 0);
  lcd.print("ax:");
  lcd.setCursor(8, 0);
  lcd.print("ay:");
  lcd.setCursor(0, 1);
  lcd.print("az:");
}
```

Code Decomposition Explanation:

- **Initialize I2C communication:**

```
Wire.begin();
```

Wire.begin(): Initializes the I2C bus to ensure communication with the LSM6DS3TR sensor.

Configure Accelerometer and Gyroscope

```
writeRegister(CTRL1_XL, 0x48);
writeRegister(CTRL2_G, 0x40);
```

writeRegister(CTRL1_XL, 0x48): Configures the accelerometer output data rate to 104Hz and range to 4g.

The writeRegister function is pre-defined in the "LSM6DS3TR.h" file and is called here to prepare for acquiring acceleration values.

writeRegister(CTRL2_G, 0x40): Configures the gyroscope output data rate to 104Hz and range to 250dps.

Set LCD Display Content

```
lcd.setCursor(0, 0);
lcd.print("ax:");
lcd.setCursor(8, 0);
lcd.print("ay:");
lcd.setCursor(0, 1);
lcd.print("az:");
```

Display ax:, ay:, az: on the first and second rows of the LCD to show the X, Y, Z axis data of the accelerometer.

4. Getting Accelerometer Data (Get_Value)

```
void Get_Value() {
  uint8_t data[6];
  readRegister(OUTX_L_XL, data, 6);
  for (int i = 0; i < 3; i++) {
    accel[i] = (int16_t)(data[i * 2] | (data[i * 2 + 1] << 8)) * ACCEL_SENSITIVITY * 9.80;
  }
}
```

- Define data array: uint8_t data[6] is used to store the 6 bytes of data read from the sensor.
- Read accelerometer data:

  • readRegister(OUTX_L_XL, data, 6): Reads 6 bytes of data starting from the accelerometer register OUTX_L_XL.

- • for (int i = 0; i < 3; i++): Converts the 6 bytes of data into 3 16-bit integers representing acceleration values along the X, Y, and Z axes.
- • (int16_t)(data[i * 2] | (data[i * 2 + 1] << 8)): Combines two bytes into a single 16-bit integer.
- • ACCEL_SENSITIVITY * 9.80: Converts the raw acceleration value into meters per second squared (m/s²), where ACCEL_SENSITIVITY is the sensor's sensitivity constant, typically 0.061 mg/LSB.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

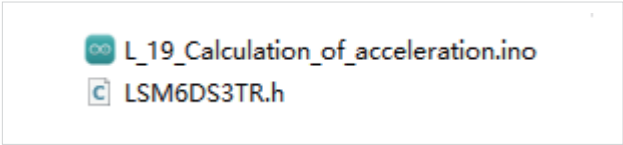
After studying the above code, you can further enhance the functionality by, for example, setting acceleration thresholds and using changes in acceleration states to control the responses of other hardware devices.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

This lesson still requires the use of the Adafruit_LiquidCrystal library (version 2.0.4); please refer to Lesson 4 for installation instructions.

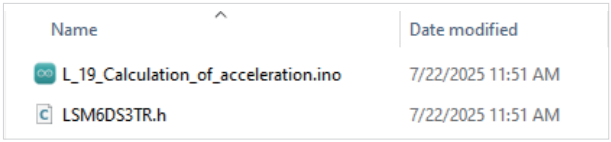
In addition, we need to put the LSM6DS3TR.h library file and the main code file in the same path so that they can be recognized during compilation.



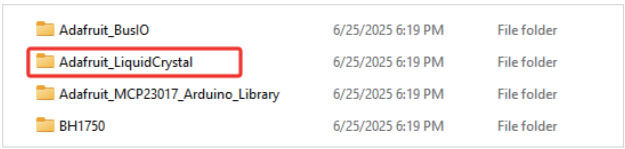
Here, you need to click the link below to download the LSM6DS3TR.h library file:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/libraries

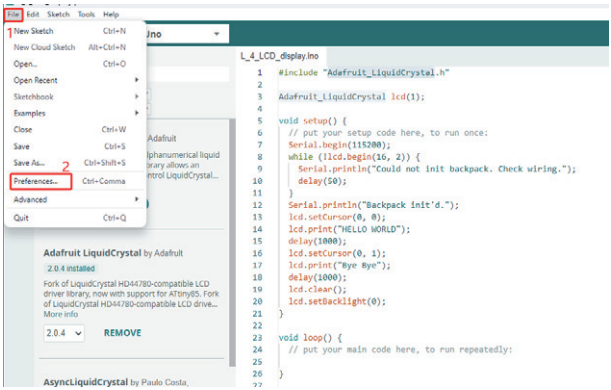
After downloading, remember to put the LSM6DS3TR.h library file together with your main code.

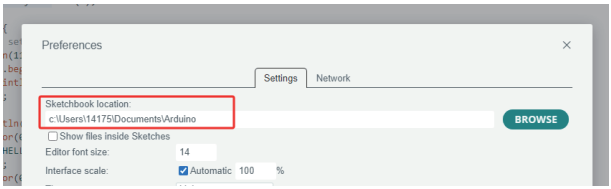


Then check whether there is Adafruit_LiquidCrystal in the specified library file compilation path.



Arduino IDE Library Folder Path Setup:





The code upload procedure for this lesson is the same as in Lesson 1; please refer to Lesson 1 for detailed steps.

After successful uploading, you will see the LCD screen on the All_in_one Starter Kit for Arduino display acceleration data for the three axes.

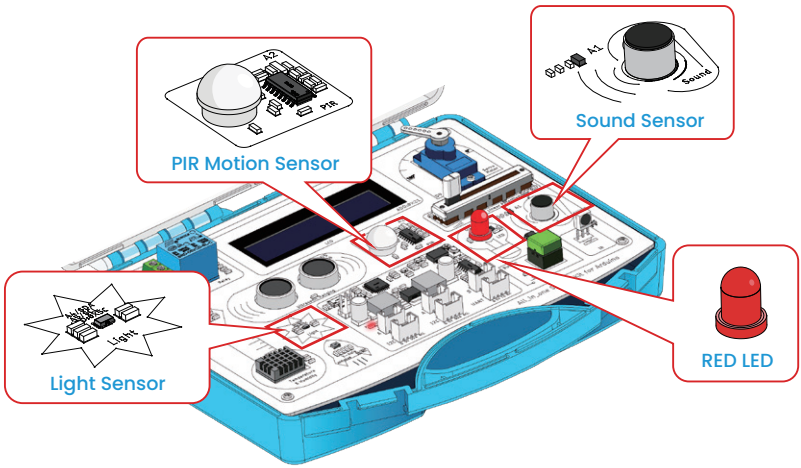
Implementation: When running the program, the LCD will show the accelerometer values for the X, Y, and Z axes. As you quickly move the accelerometer along an axis, you will observe the corresponding accelerometer value change accordingly.

Lesson 20 - Smart Corridor Light

Introduction

In this lesson, we will study advanced control techniques for the integration of sound sensors, PIR motion sensors, light sensors, and LEDs. By fusing data from multiple sensors and coordinating their logic, we will realize an intelligent corridor lighting system that combines human presence detection, ambient light assessment, and sound-triggered activation. This will help you master hardware connections and programming methods for multi-sensor collaborative operation.

Hardware Used in This Lesson:



Working Principle of Photoresistor Sensor

The working principle of a photoresistor sensor is based on the photoelectric effect in semiconductor materials. When light irradiates its photosensitive element (such as a photoresistor or photodiode), the photon energy excites electrons in the semiconductor, generating free electrons and holes. This alters the electrical properties of the element (such as resistance, current, or voltage), thereby converting light signals into electrical signals. For example, the resistance of a photoresistor decreases as light intensity increases, while the reverse current of a photodiode under reverse bias increases with light intensity. These variations are processed by circuits to produce analog or digital output signals.

Working Principle of the Sound Sensor

A sound sensor primarily uses a microphone or an electret condenser microphone to capture ambient sound waves. These sound waves cause the diaphragm inside the sensor to vibrate, which results in changes in internal capacitance or resistance. This variation converts the sound signal into an electrical signal. The signal is then amplified by an internal circuit and processed through filtering and conditioning to produce either an analog voltage or a digital signal corresponding to the sound intensity.When the detected sound reaches a preset threshold, the sensor activates a corresponding circuit—such as triggering a buzzer—to respond to the environmental sound. Sound sensors are commonly used in applications like voice-activated switches and alarm systems.

Working Principle of PIR Motion Sensor

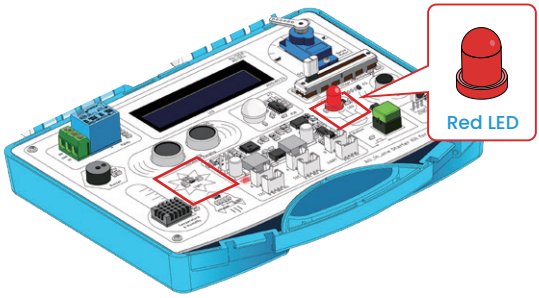
A PIR (Passive Infrared) motion sensor operates by detecting changes in infrared radiation emitted by humans or animals. Its core component is a pyroelectric sensor, which senses variations in infrared levels when an object moves within its detection range. These changes are caused by the difference in thermal radiation between the moving object and the background. The detected signals are then amplified and converted into electrical signals, resulting in a change in output level. This enables non-contact motion detection, making PIR sensors widely used in applications such as security alarms and smart home systems for automatic human presence sensing.

Working Principle of an LED

The core of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During recombination, electrons transition from a higher to a lower energy level, releasing excess energy in the form of photons—producing visible light. The color (wavelength) of the light depends on the bandgap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

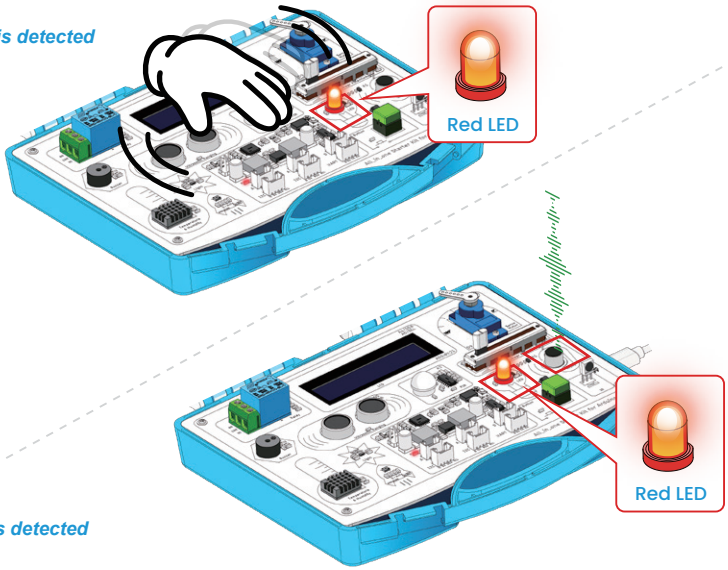
When the ambient light is strong, the LED will remain off regardless of any motion or sound detected.



When the ambient light is dim or the photoresistor sensor is covered, the LED is initially off by default.

At this time, if motion or sound is detected, the LED will turn on for 10 seconds and then turn off.

- ***motion is detected***



- ***sound is detected***

If there is continuous motion or continuous sound, the LED will stay on continuously until no activity is detected. After that, it will remain on for an additional 10 seconds before turning off.

Key Explanations

1. Initialization Function (setup)

```
void setup() {
  Serial.begin(115200);
  Wire.begin();
  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire)) {
    Serial.println(F("BH1750 Advanced begin"));
  } else {
    Serial.println(F("Error initialising BH1750"));
  }
  pinMode(PIR_PIN, INPUT);
  pinMode(LedPin, OUTPUT);
  pinMode(SOUND_PIN, INPUT);
}
```

- Initialize the I2C bus: Wire.begin() is used to initialize the I2C bus to ensure communication with the BH1750 sensor.
- Initialize the BH1750 light intensity sensor:

  if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire))

  • Use lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c, &Wire) to initialize the sensor in continuous high-resolution mode (BH1750::CONTINUOUS_HIGH_RES_MODE).

  • If initialization is successful, print “BH1750 Advanced begin”; otherwise, print an error message.
- Set pin modes:

  • PIR_PIN (A2) is set as input, connected to the PIR motion sensor.

  • LedPin (10) is set as output to control the LED.

  • SOUND_PIN (A1) is set as input, connected to the sound sensor.

2. Loop Function (loop)

- **Read light intensity value:**

```
if (lightMeter.measurementReady(true))
```

Use lightMeter.measurementReady(true) to check if the light sensor data is ready.

```
int lux = lightMeter.readLightLevel();
```

Use lightMeter.readLightLevel() to read the light intensity value (in lux).

- **Control LED and sensors based on light intensity:**

```
if (lux < 100)
```

• If the light intensity is less than 100 lux:

```
while (1) {
    int state = digitalRead(PIR_PIN);
    if (state == HIGH || digitalRead(SOUND_PIN)) {
```

Enter an infinite loop to continuously monitor the state of the PIR sensor and sound sensor.

```
      digitalWrite(LedPin, HIGH);
      delay(10000);
```

If motion or sound is detected, turn on the LED and delay for 10 seconds.

```
else if (state == LOW && digitalRead(SOUND_PIN) == LOW) {
      digitalWrite(LedPin, LOW);
    }
```

If no motion or sound is detected, turn off the LED.

```
    lux = lightMeter.readLightLevel();
    if (lux >= 100)
      break;
```

Re-read the light intensity value in each loop iteration; if the light intensity reaches or exceeds 100 lux, break out of the loop.

```
else if (lux >= 100) {
    digitalWrite(LedPin, LOW);
  }
```

Else if the light intensity is greater than or equal to 100 lux, turn off the LED.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After studying the code above, we trust that your logical thinking abilities have been further enhanced, enabling you to coordinate and control multiple hardware resources simultaneously. Next, we recommend optimizing the logic design to achieve clear and systematic control over the entire hardware system.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

Since this lesson uses the light sensor again, a reminder: please refer to Lesson 6 to install the BH1750 library (version 1.3.0).

Now that the library has been successfully installed, we can proceed to upload the code used in this lesson.

The code upload steps for this lesson are the same as those in Lesson 1. Please refer to Lesson 1 for detailed instructions.

Once the upload is successful, you can start the experiment:

Cover the light sensor with your hand, then wave your hand to simulate motion detection by the PIR sensor—you should see the LED turn on.

Alternatively, make a sound—if the sound sensor detects it, the LED will also turn on.

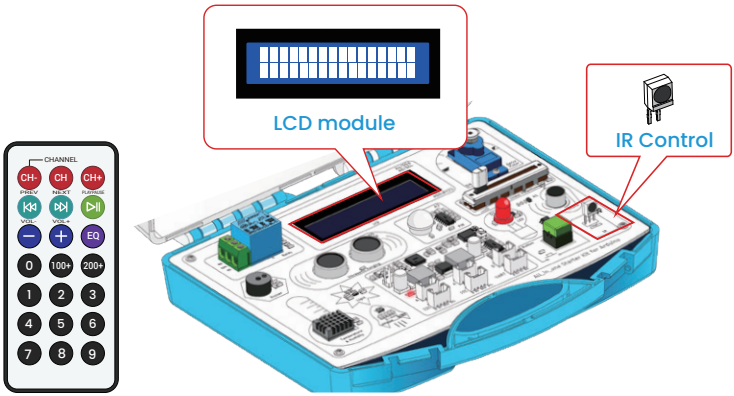
Implementation: When the ambient light is strong, the LED will remain off regardless of any motion or sound. When the ambient light is dim or the light sensor is covered, the LED will default to the off state. At this time, if motion or sound is detected, the LED will light up for 10 seconds and then turn off. If motion or sound continues, the LED will remain on until all activity stops, and then it will stay lit for another 10 seconds before turning off. If no behavior is observed, please double-check to ensure your code matches exactly.

Lesson 21 - Simple Calculator

Introduction

In this chapter, we will explore advanced application scenarios of infrared remote control, learning how to use the remote to input numerical values and expressions, with real-time display of the input content on the screen. When the confirmation button is pressed, the system will automatically parse and evaluate the expression, and finally present the calculation result on the screen. This lesson aims to equip you with a complete understanding of infrared remote interaction and data processing workflows.

Hardware Used in This Lesson:



Working Principle of Infrared Remote Control and Receiver

The working principle of an infrared (IR) remote control and receiver is based on the transmission of encoded infrared signals and their subsequent decoding and execution. When a button is pressed on the remote control, the internal encoding circuit converts the button input into a specific binary code format (such as NEC or RC-5 protocol). This code is then modulated—typically using a 38kHz carrier frequency to reduce interference—and transmitted as infrared light pulses (around 940nm wavelength) via an IR LED. On the receiving side, an IR receiver module (e.g., HS0038) detects the incoming IR signal using a photodiode. The received signal is then amplified, filtered, and demodulated to recover the original binary code. This decoded signal is interpreted by a microcontroller or main processing unit, which then performs the corresponding action—such as adjusting a servo angle or switching an appliance on or off. The entire process follows a closed loop: **button encoding → infrared modulation and transmission → photodetection and signal decoding → command execution**, enabling effective wireless c

Working Principle of the LCD Screen

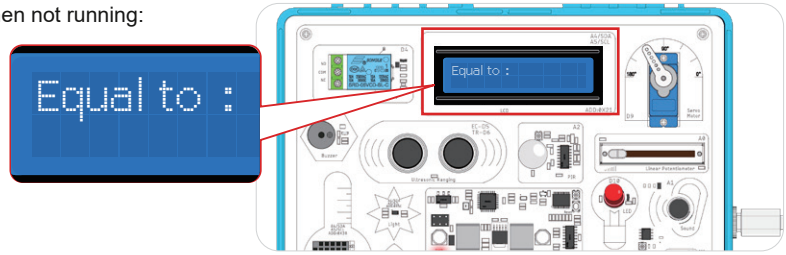
The LCD1602 screen (a 16×2 character liquid crystal display) operates based on the electro-optical effect of liquid crystals. By controlling the electric field, it changes the alignment of liquid crystal molecules to produce visual display effects. Internally, it mainly consists of the LCD panel, a controller (such as the HD44780 or a compatible chip), driver circuits, and a backlight module.

The controller receives commands and data from a microcontroller or other devices. Through the driver circuits, electrical signals are applied to the segment electrodes and common electrodes of the LCD. Under the electric field, the liquid crystal molecules twist and bend at corresponding positions, altering the amount of light passing through. This causes pixels to appear either bright or dark, which combine to form characters or patterns.

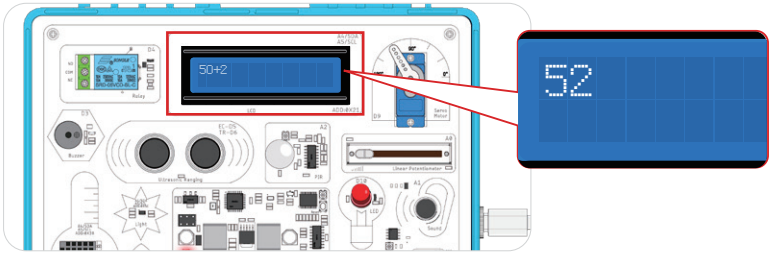
The backlight module (usually LEDs) provides illumination to ensure clear visibility even in low-light environments. Data transmission occurs via parallel or serial interfaces (such as I2C or SPI). The microcontroller sends commands (e.g., setting display mode, cursor position) and display content (ASCII character codes) according to the communication protocol. The controller interprets these instructions and drives the corresponding pixels to light up, ultimately displaying characters within the 16-column by 2-row display area.

Operation Effect Diagram

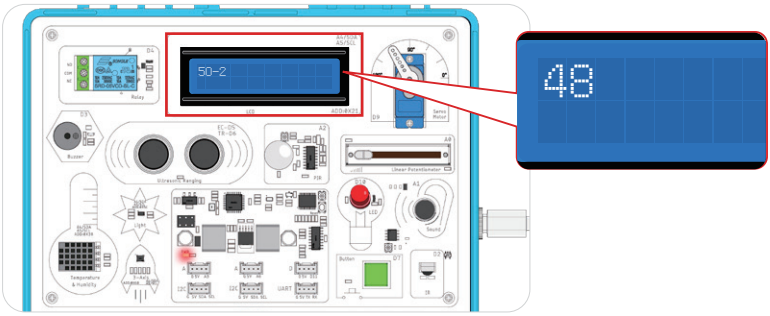
When not running:



When a button is pressed on the infrared remote to perform an addition operation:



When a button is pressed on the infrared remote to perform a subtraction operation:



Key Explanations

1. Initialization Function (setup)

```
void setup() {
  Serial.begin(115200);
  while (!Serial);
  IR.Init(IR_PIN);
  Serial.println("init over");
  while (!lcd.begin(16, 2)) {
    Serial.println("Could not init backpack. Check wiring.");
    delay(50);
  }
  lcd.setCursor(0, 0);
  lcd.print("Equal to:");
  lcd.setCursor(0, 1);
}
```

- Initialize Serial Communication: Sets the baud rate to 115200 for debugging and monitoring.
- Initialize IR Receiver Module: IR.Init(IR_PIN) initializes the infrared receiver module, where IR_PIN is defined as digital pin 2.
- Initialize LCD Screen: lcd.begin(16, 2) attempts to initialize the LCD screen. If initialization fails, an error message is printed, and the system delays 50 milliseconds before retrying until success.
- Set Initial LCD Display Content: The first line of the LCD displays “Equal to:”, while the second line is left blank to display the calculation result.

2. Main Loop Function (loop)

Read Infrared Data

```
if (IR.IsDta()) {
  byte length = IR.Recv(dta);
```

IR.IsDta() is used to check whether any infrared data has been received.

IR.Recv(dta) receives the infrared data and stores it in the array dta.

Execute Operation Based on Key Code

```
switch (dta[8]) {
```

dta[8] typically contains the key code used to distinguish different buttons.

Button 144（EQ）

```
case 144: Serial.println("[EQ]");
    num += '=';
    num3 = "Equal to:";
    if (sliceString(num, num1, num2)) {
```

- Add the equal sign = to the string num.
- Attempt to parse the expression：

  • sliceString(num, num1, num2)： Slice the string num into two operands num1 and num2.

  • If parsing is successful:

    • Check if the operands are within the valid range (0 to 100000000).

    • If operands exceed the range, display an error message.

    • Otherwise, perform addition or subtraction based on the operator and display the result on the LCD.

  • If parsing fails, display an error message.

3. Auxiliary Functions

sliceString Function
Locating Operator and Equals Sign Positions

```
int plusIndex = input.indexOf('+');
int equalIndex = input.indexOf('=');
```

input.indexOf('+'): Searches for the first occurrence of the plus sign + in the string and returns its index. Returns -1 if no plus sign is found.

input.indexOf('='): Searches for the first occurrence of the equals sign = in the string and returns its index. Returns -1 if no equals sign is found.

Validating Operator and Equals Sign

```
if (plusIndex != -1 && equalIndex != -1 && equalIndex > plusIndex) {
```

Checks if both the plus sign + and equals sign = are found.

Verifies that the equals sign = appears after the plus sign + to ensure the expression format is correct (e.g., 123+456=).

Extracting Operands

```
part1 = input.substring(0, plusIndex);
part2 = input.substring(plusIndex + 1, equalIndex);
```

input.substring(0, plusIndex): Extracts the first operand from the start of the string to the position before the plus sign +.

input.substring(plusIndex + 1, equalIndex): Extracts the second operand from the position after the plus sign + to the position before the equals sign =.

The **sliceString** function parses two operands and an operator from the input string by locating the positions of + and =, extracting operands, and returning the parsing result. This function is a critical component for implementing mathematical expression parsing and calculation.

Complete Code

Click the link below to open the complete code:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Arduino-Common-Board-Design-Kit/tree/master/example/all%20in%20one%20Arduino

After learning the above code, you have done a great job and successfully completed the most difficult part of the code in this course. If you are still interested, you can add some other calculation processing logics by yourself, such as multiplication and so on.

Uploading the Code

The code explanation is complete. Next, we need to upload the above code to the All-in-one Starter Kit for Arduino so we can see the hardware functions in action.

Since this lesson uses the **IRSendRev** library, please refer to Lesson 13 if you haven't installed it!

Additionally, the **Adafruit_LiquidCrystal** library (version 2.0.4) is used for screen display. Please refer to Lesson 4 if you haven't installed it!

After completing the above operations to install the library files, you can now upload the code we are using.

The code upload steps for this lesson are the same as those in the first lesson. Please refer to the first lesson!

Once the upload is successful, you will be able to use the infrared remote control to press the buttons on the remote to perform addition and subtraction calculations, and observe the calculation process and results on the LCD screen.

MAKE YOUR MAKING EASIER