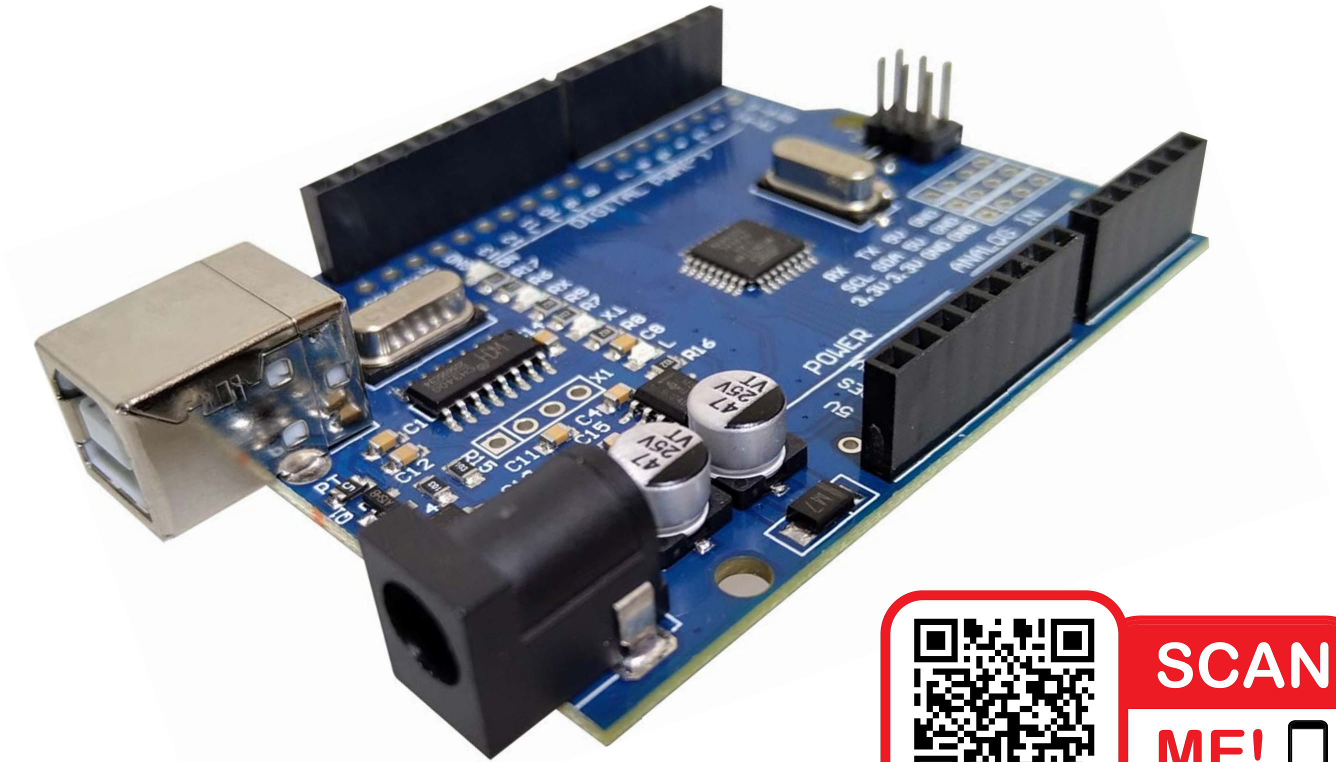
ARD2-1015

ARD2 Arduino Compatible Expanded Kit

ARD 2

Using the

UNO R3 Compatible Controller Board



SCAN ME!

A Step by Step Introduction with 24 Projects

Aligned with the Australian F-10 Curriculum
Design and Technologies (Version 8.4)

WILTRONICS ONLINE

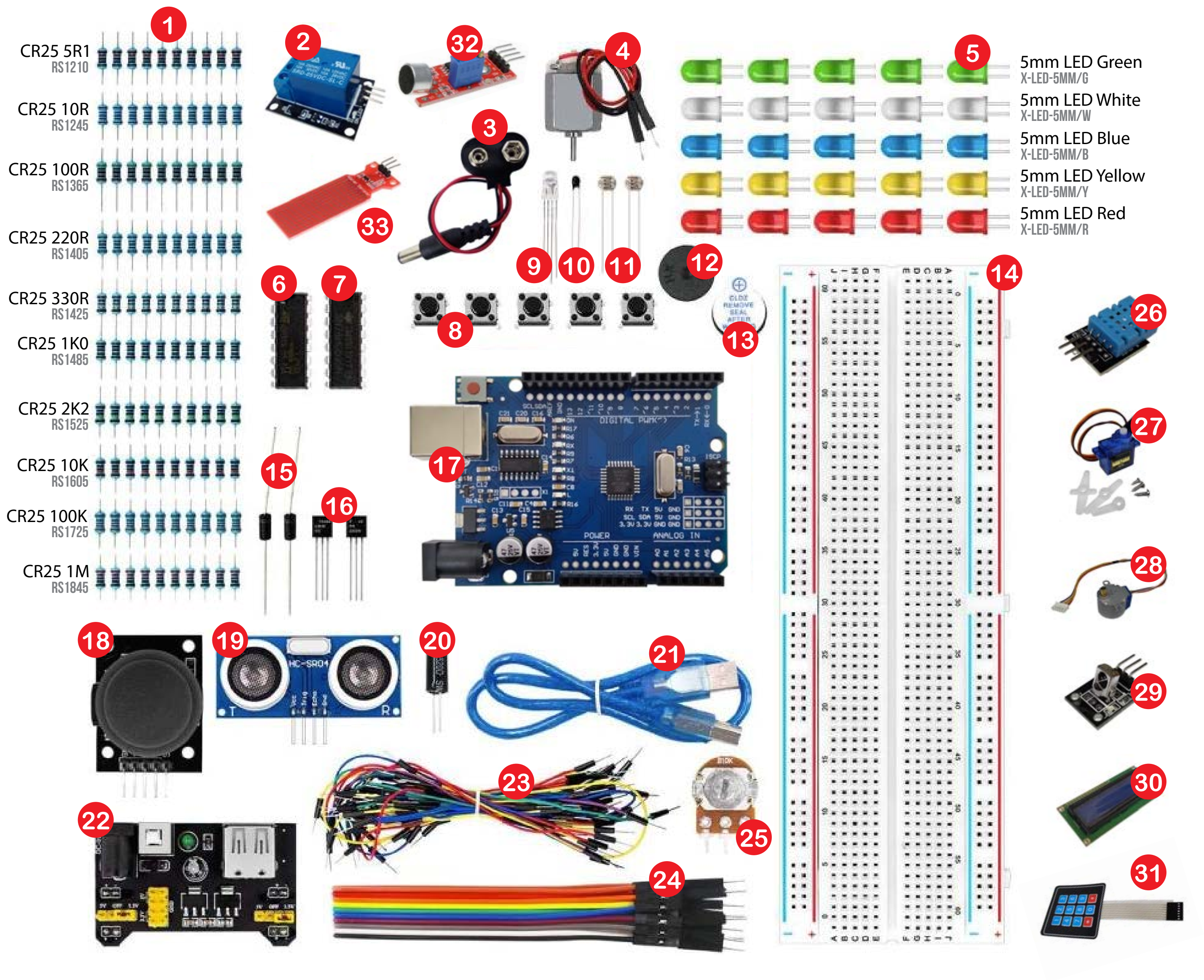
Table of Contents

Aligned with the Australian F-10 Curriculum : Design and Technologies (Version 8.4)

Digital Technologies Knowledge and Understanding
ACTDEK004, ACTDEK011, ACTDEK013, ACTDEK019
Digital Technologies Processes and Production Skills
ACTDEP005, ACTDEP014, ACTDEP016, ACTDEP024

List of Components

1. Resistors x 100 RS5000
2. 5V Relay RE0105
3. 9V Battery Connection BA9006
4. DC Motor M00000
5. LEDs x 25 LED-5MM
6. 74HC595 IC 74HC595
7. L293D IC L293D
8. Push Button Switch x 5 SW0520
9. RGB LED x 1 X-LED-RGBCA
10. Thermistor x 1 RS6472
11. Photoresistor x 2 X-MPB12C39A
12. Passive Buzzer x 1 SP1200
13. Active Buzzer x 1 AC3008
14. 830 Points Breadboard MA4009
15. Diodes x 2 X-1N4007
16. Transistor x 2 X-PH2222A
17. Uno R3 Dev Board ARD2-0066
18. Analog Joystick ARD2-2223
19. UltraSonic Sensor ARD2-2020
20. Tilt Ball Switch ARD2-2220
21. USB Lead CM4058
22. Power Supply Module ARD2-4025
23. M-M Wires x 32 MA3020
24. F-M Wires x 10 CN3602
25. Potentiometer PT076X
26. Temp & Humidity Sensor ARD2-2215
27. Servo (SG90) HA9740
28. Stepper Motor M01735
29. IR Receiver ARD2-2222
30. LCD1602 Module ARD2-1210
31. 4x4 Matrix Switch ARD2-2116
32. Audio Sensor ARD2-2238
33. Water Sensor ARD2-3025



Lesson 1
Installing the IDE

Introduction

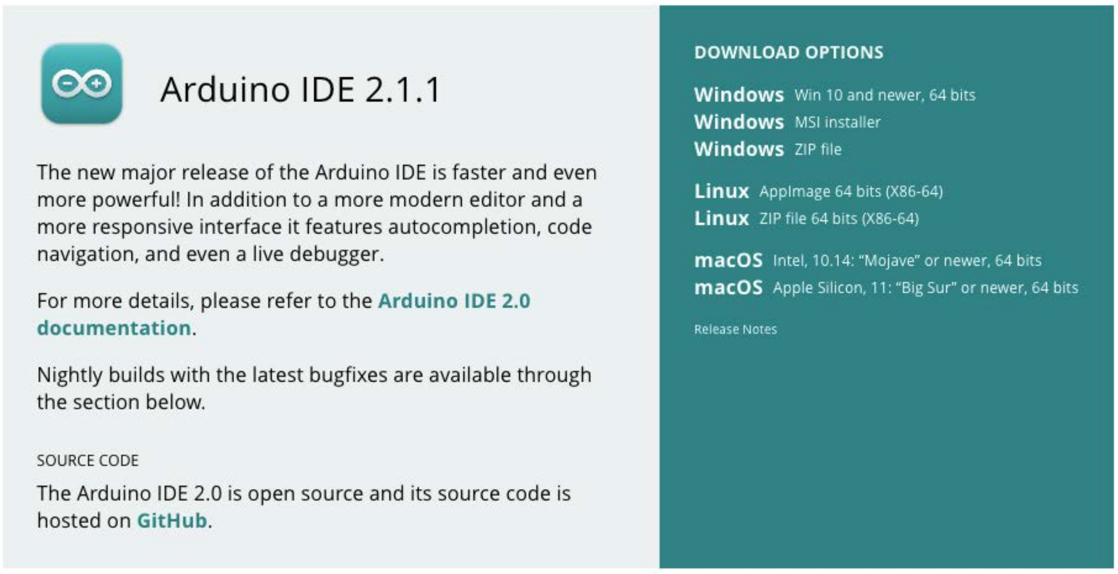
The Arduino Integrated Development Environment (IDE) is the software used in conjunction with the Arduino platform. This lesson aims to guide you in setting up your computer to use Arduino and help you get started with the subsequent lessons.

To program your Arduino board, you'll need to install the Arduino software, which is compatible with Windows, Mac, and Linux. However, the installation process varies for each platform, requiring some manual steps.

Let's walk through the setup process to get you started smoothly.

STEP 1

Visit **https://www.arduino.cc/en/software**
Download the Latest Version of ARDUINO compatible with your Operating System (Fig 1)



Arduino IDE 2.1.1

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the Arduino IDE 2.0 documentation.

Nightly builds with the latest bugfixes are available through the section below.

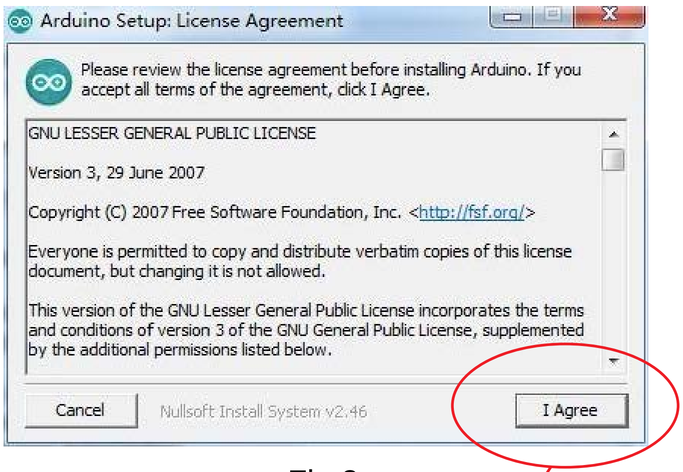
SOURCE CODE
The Arduino IDE 2.0 is open source and its source code is hosted on GitHub.

DOWNLOAD OPTIONS
Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file
Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)
macOS Intel, 10.14: "Mojave" or newer, 64 bits
macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits
Release Notes

Fig 1

Installing Arduino (Windows)

Follow the Installation Procedures (Fig 2, 3, 4, 5)



Arduino Setup: License Agreement

Please review the license agreement before installing Arduino. If you accept all terms of the agreement, click I Agree.

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

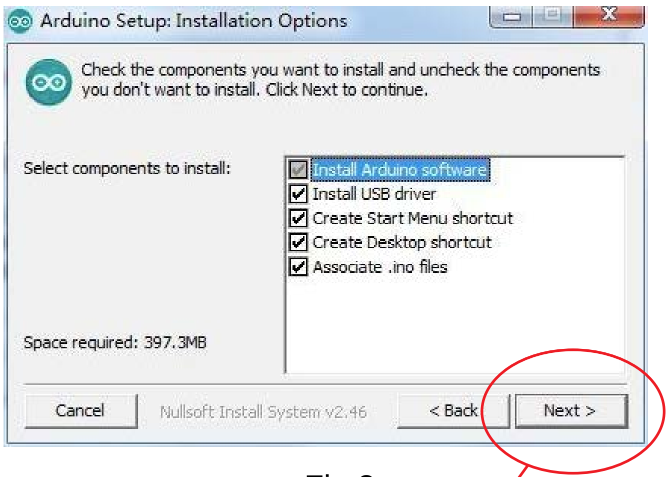
Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

Fig 2

Click **I Agree**



Arduino Setup: Installation Options

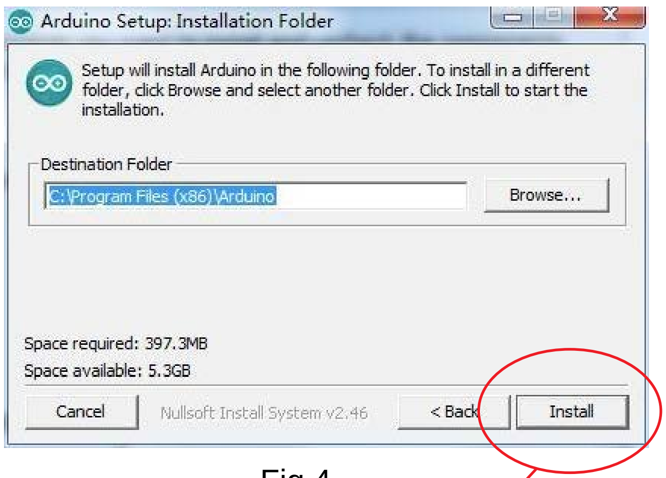
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:
- [x] Install Arduino software
- [x] Install USB driver
- [x] Create Start Menu shortcut
- [x] Create Desktop shortcut
- [x] Associate .ino files

Space required: 397.3MB

Fig 3

Click **Next**



Arduino Setup: Installation Folder

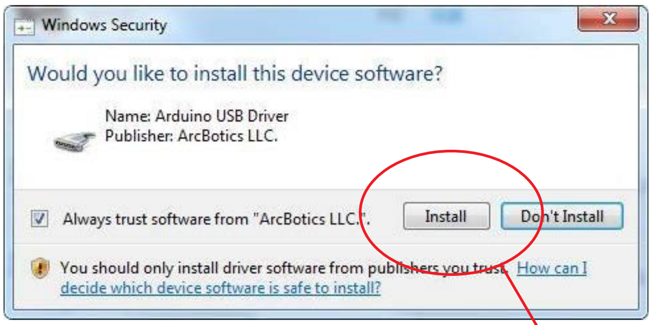
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Arduino

Space required: 397.3MB
Space available: 5.3GB

Fig 4

Click **Install**



Windows Security

Would you like to install this device software?

Name: Arduino USB Driver
Publisher: ArcBotics LLC.

- [x] Always trust software from "ArcBotics LLC.".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Fig 5

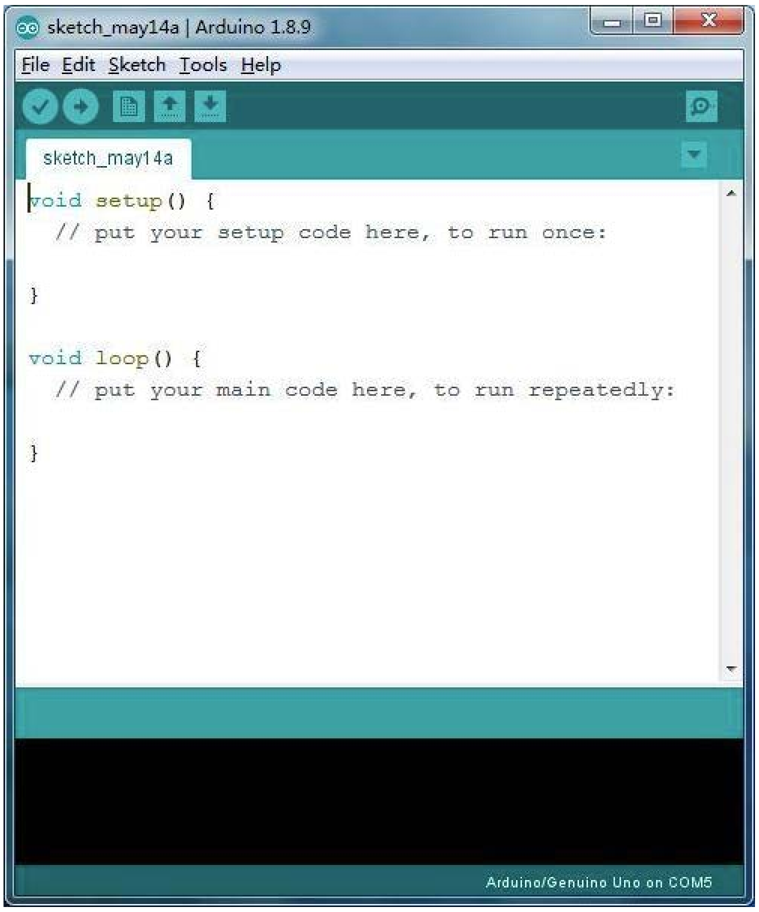
Click **Install**

Installing Arduino (Windows)

After Successful Installation the Arduino IDE Icon will appear on your Desktop (Fig 6)

Fig 6

Double Click The Arduino Icon to enter the Development Environment (Fig 7)


sketch_may14a | Arduino 1.8.9
File Edit Sketch Tools Help
sketch_may14a
Arduino/Genuino Uno on COM5

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Fig 7

Installing Arduino (Mac OS)

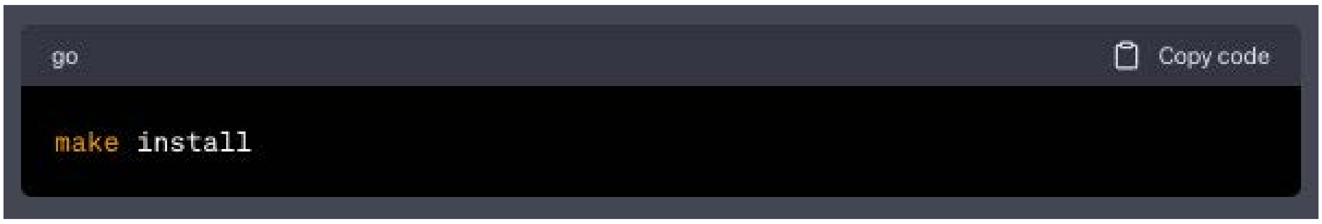
To get started with the Arduino IDE, follow these steps:

1. Download the Arduino software by getting the zip file for your operating system.
2. Unzip the downloaded file to access the Arduino application.
3. Double-click on "Arduino.app" to launch the Arduino IDE.
4. If you don't have Java runtime library installed on your computer, the system will prompt you to install it. Follow the on-screen instructions for Java installation.
5. Once the Java installation is complete, you can run the Arduino IDE and start programming your Arduino board.

Installing Arduino (Linux)

To install Arduino on a Linux system, you can follow these steps:

Open a terminal on your Linux machine.
Use the following command to install Arduino using the **make install** command:



```
make install
```

Please note that this method assumes you have already downloaded and extracted the Arduino software from the official website.

Alternatively, if you are using Ubuntu, it is recommended to install Arduino IDE from the Software Center of Ubuntu. Open the Software Center, search for "Arduino IDE," and click on the install button. Using the Software Center ensures a user-friendly installation process and automatic updates when available, making it more convenient for Ubuntu users.

Lesson 2
Adding Libraries

Introduction

To enhance the capabilities of your Arduino and access features beyond the built-in functions, you can install additional Arduino libraries.

These libraries provide pre-written code and functions that you can easily integrate into your Arduino projects.

Here's how you can install additional Arduino libraries:

1. Open the Arduino IDE on your computer.

2. Go to "Sketch" in the menu bar, then select "Include Library," and choose "Manage Libraries."

3. A Library Manager window will open, showing a list of available libraries.

4. Use the search bar to find the specific library you want to install.
You can search by name or function.

5. Once you find the desired library, click on it, and a "Install" button will appear.
Click on "Install" to add the library to your Arduino IDE.

6. After installation, you can access the new library's functions by including it in your Arduino sketch.
To do this, go to "Sketch" > "Include Library" and select the library you installed.

Now, you have extended the capabilities of your Arduino, and you can use the additional functions provided by the installed library in your projects..

Lesson 3
Blink

Overview

In this lesson, you will learn two essential things:

A. How to program your UNO R3 compatible controller board to make the built-in LED blink.
B. How to download and upload programs to your UNO R3 Compatible board through basic steps.

Let's break down each part:

Part A

Programming the UNO R3 Compatible Board to Blink the Built-in LED
You will be guided through the process of loading a simple sketch using the Arduino IDE.
This will make the LED on your UNO R3 compatible board blink on and off at regular intervals.

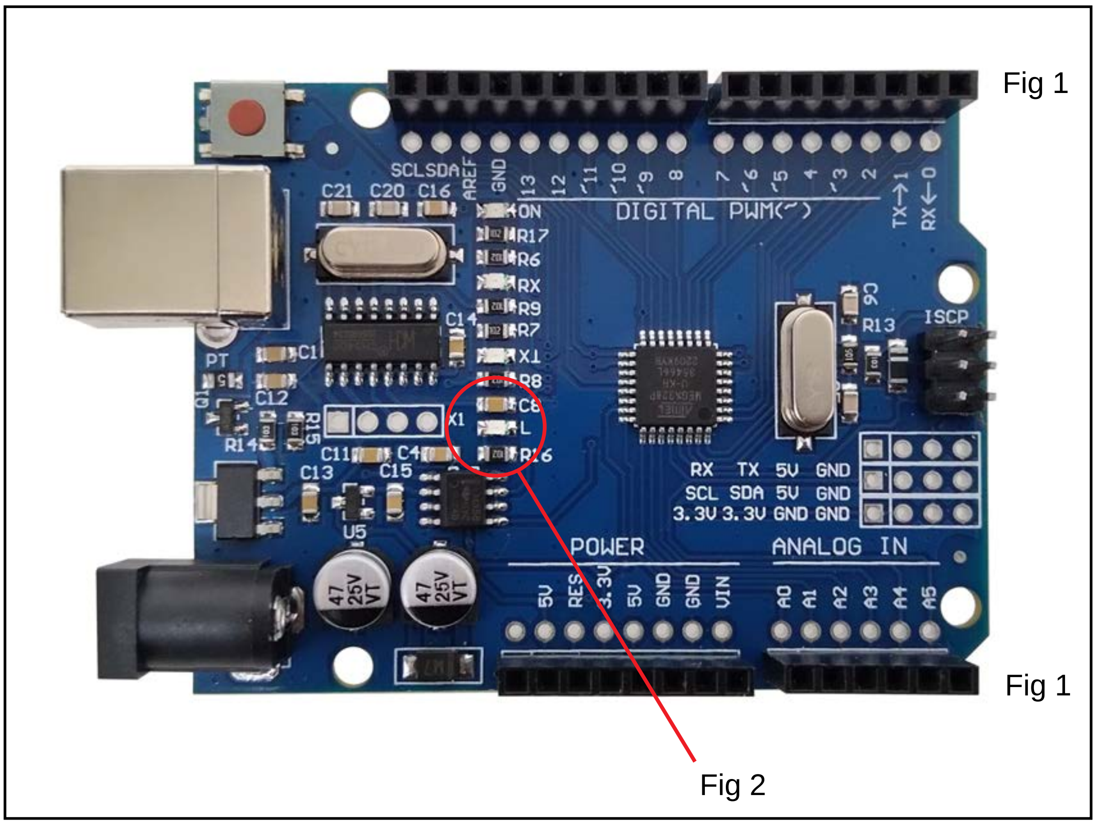
Part B

Uploading Programs to the UNO R3 Compatible Board
You will be introduced to the steps required to upload your programmed code to the
UNO R3 compatible controller board.
This process allows you to transfer your written program from the Arduino IDE on your
computer to the physical board so that it can run the desired function.

These skills will form the foundation for more complex and exciting Arduino projects in the future.

Components Required

1 x UNO R3 Compatible Development Board



Principle

The UNO R3 compatible board features rows of connectors on both sides, allowing you to connect electronic devices and attach 'shields' that extend its functionality. (Fig 1)
Shields are add-on modules that can be plugged into the board to provide additional features.

The UNO R3 compatible board also comes equipped with a single LED that you can control directly from your Arduino sketches. (Fig 2)
This LED is an integral part of the UNO R3compatible board and is commonly referred to as the 'L' LED, as it is labeled as such on the board itself.
With your sketches, you can easily manipulate this LED to turn it on, off, or make it blink, serving as a handy indicator or part of your project's functionality.

You might notice that when you connect your UNO R3 compatible board to a USB plug, the 'L' LED already blinks. This happens because the boards are usually shipped with the 'Blink' sketch pre-installed.
In this lesson, we will reprogram the UNO R3 compatible board with our own version of the Blink sketch and even modify the rate at which the LED blinks.

The Arduino IDE comes equipped with a vast collection of example sketches that you can load and utilize. Among these examples, there is one specifically for making the 'L' LED blink.
We will use this example as a starting point and customize it to suit our needs.

Part A

Loading the Blink Sketch

Load the 'Blink' sketch that you will find in the IDE's menu system (Fig 3

File > Examples > 01.Basics > Blink

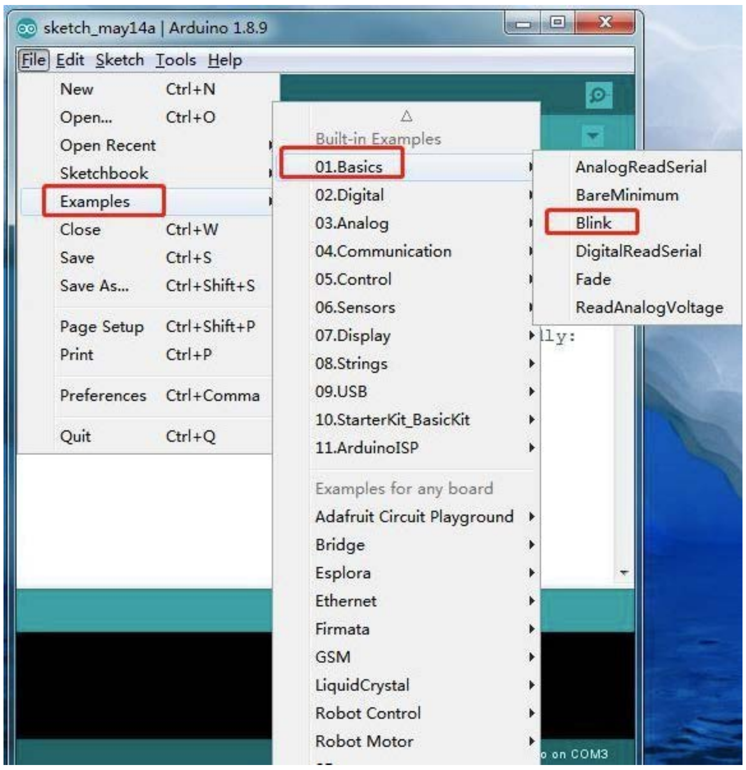
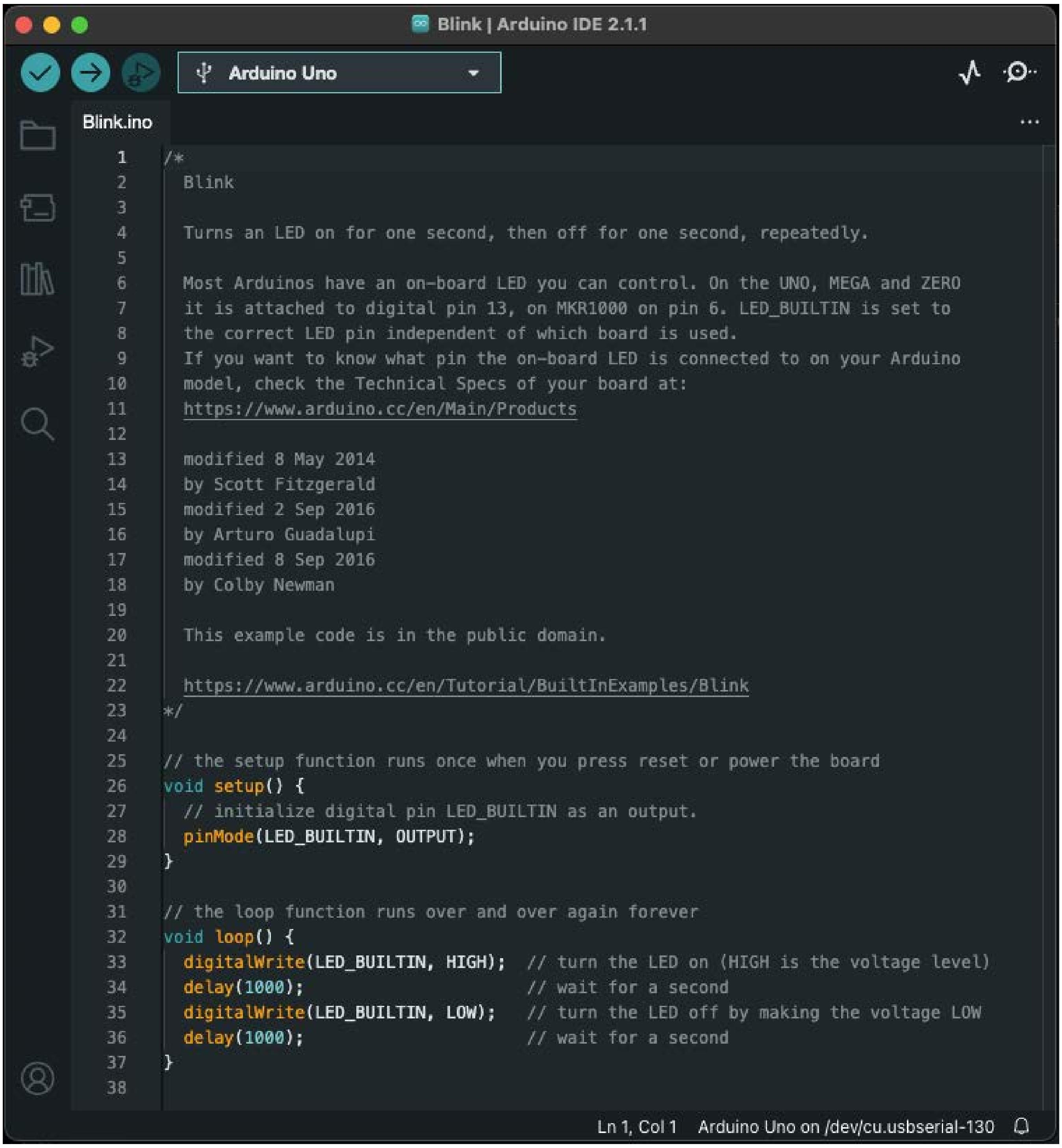


Fig 3

When the BLINK Sketch is loaded the code opens in a new window (Fig 4)



```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}

```

Ln 1, Col 1   Arduino Uno on /dev/cu.usbserial-130

Fig 4

Part B

Uploading Code to the Board

When uploading code to your UNO R3 Compatible Development Board, you need to ensure that you have selected the **correct board model** and **port number** in the Arduino IDE.

Board Model Selection (Fig 1)

Tools > Board > Arduino/Genuino Uno

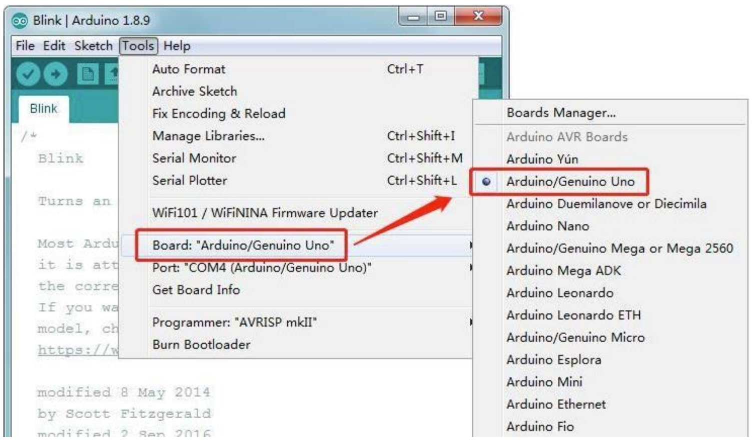


Fig 1

Board Port Number Selection (Fig 2)

Tools > Port > ...

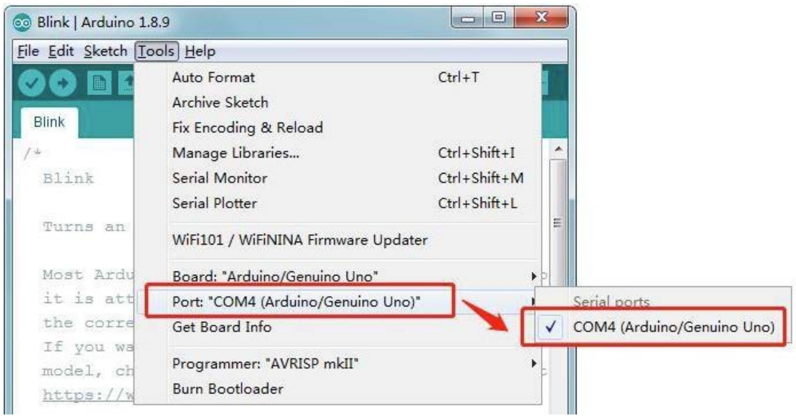

Blink | Arduino 1.8.9
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM4 (Arduino/Genuino Uno)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
✓ COM4 (Arduino/Genuino Uno)

Fig 2

Look for the port that corresponds to your UNO R3 Compatible Development Board.
It will be labeled with a format like "COMX (Arduino/Genuino Uno)" or similar.
Select this port to establish the connection for code uploading.

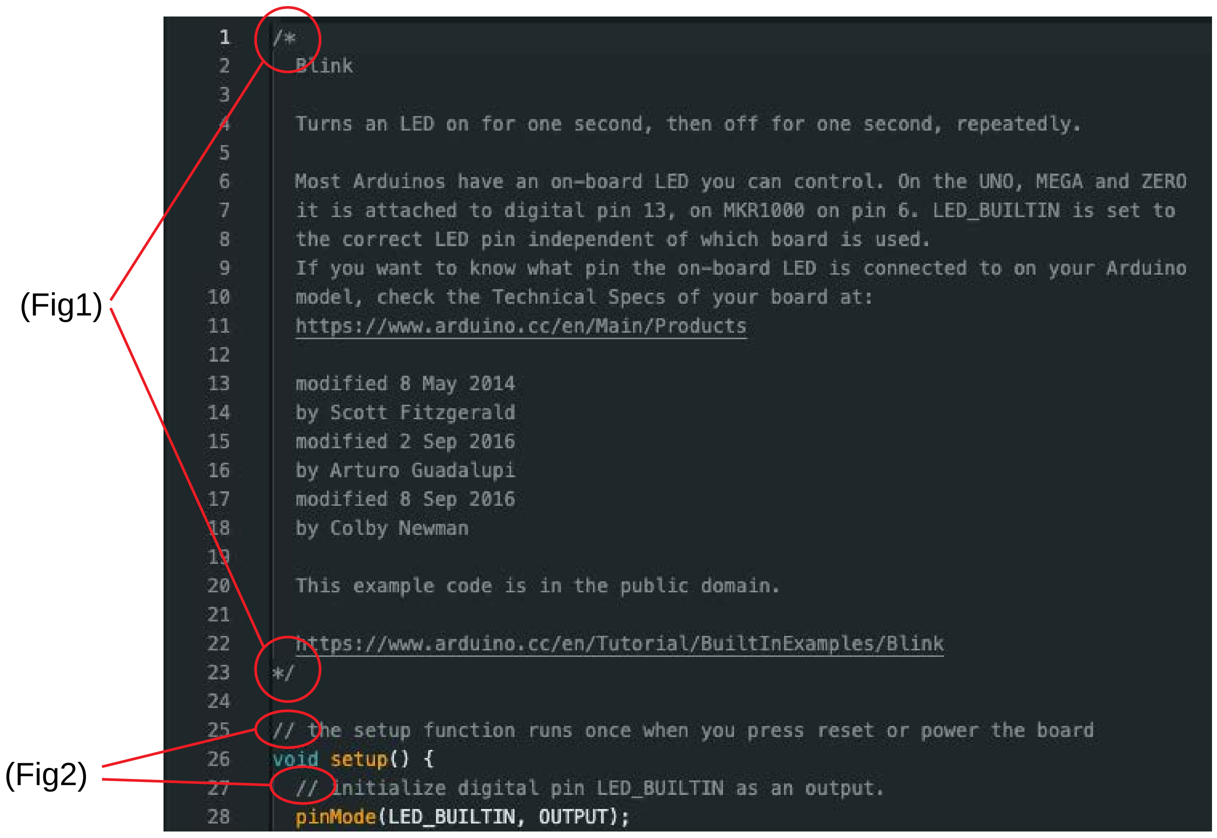
Understanding The Code

In this sketch, you'll notice a significant portion is comprised of comments.
These comments serve as explanatory notes and do not act as actual program instructions.
Their purpose is to help you understand how the program works.

The comments are categorized into two types:

Block Comments (Fig1)
These comments are enclosed between **/*** and ***/** at the top of the sketch.
They serve to explain the overall purpose and functionality of the entire sketch.

Single Line Comments (Fig 2)
These comments start with **//** and extend to the end of the line.
They provide specific explanations for individual lines of code, guiding you through the program's logic step by step.



```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
```

Setup Function

Every Arduino sketch must include a SETUP function.
It is triggered when you manually press the reset button, and when the board resets due to power cycles or sketch uploads.

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

The 'setup' function is enclosed within **curly braces { }**
This is where you can add your own instructions to configure and initialize project settings.

In this sketch you'll find only one command within the 'setup' function.

```
// initialize the digital pin LED_BUILTIN as an output
 pinMode(LED_BUILTIN, OUTPUT);
```

This command informs the Arduino board that the LED pin will be used as an output.

Loop Function

In every Arduino sketch, there must also be a LOOP function.
Unlike the 'setup' function, which runs only once after the board is turned on or reset, the 'loop' function keeps repeating its instructions over and over again
This continuous loop allows your program to keep running until you turn off the board.

```
void loop ()
 {
 digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
 delay(1000);                                  // wait for a second
 digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW)
 delay(1000)                                   // wait for a second
}
```

Loop Function (cont)

Inside the 'loop' function, these commands control the LED blinking:

1. `digitalWrite(LED_BUILTIN, HIGH);`	Turn the LED on (HIGH).
2. `delay(1000);`	Pause for 1000 milliseconds (1 second).
3. `digitalWrite(LED_BUILTIN, LOW);`	Turn the LED off (LOW).
4. `delay(1000);`	Pause for another 1000 milliseconds (1 second).

To make your LED blink faster, you can adjust the parameter inside the parentheses of the 'delay' command. The value inside the parentheses determines how long the delay is in milliseconds. By reducing this value, you can make the LED blink more rapidly.

```
void loop ()
 {
 digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
 delay(500);                                 // wait for a second
 digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW)
 delay(500)                                  // wait for a second
}
```

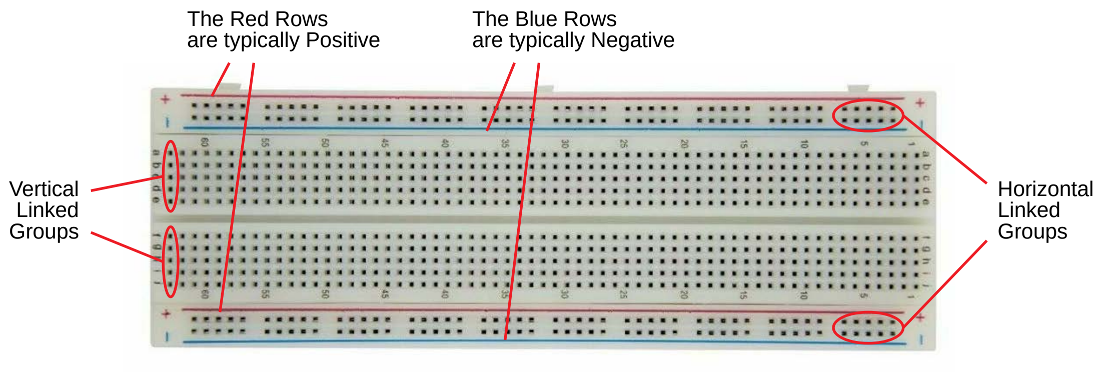
For example, if you set the delay to 500 milliseconds, the LED will blink twice as fast, and if you set it to 250 milliseconds, it will blink even faster.

Experiment with different delay values to find the blinking speed that suits your project.

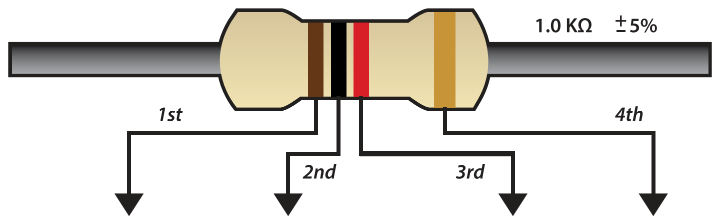
Upload the sketch again and you should see the LED start to blink more quickly.

About Breadboards

The 830 Tie Points Breadboard is used for electronics prototyping without soldering, with "1 x 830" indicating its tie points. Tie points, essential for component insertion, are categorized into vertical and horizontal groups.
Vertical groups in the board's center accommodate integrated circuits (ICs), connecting top to bottom but not side-by-side.
Horizontal groups or "power rails" run on the board's edges, color-coded for positive (red) and negative (blue/black) connections. Proper connections prevent short circuits and damage.


The Red Rows are typically Positive
The Blue Rows are typically Negative
Vertical Linked Groups
Horizontal Linked Groups

Resistor Colour Code Chart



Colour	*1st Band*	*2nd Band*	*Decimal Multiplier*		*Tolerance*
Black	0	0	1	1	
Brown	1	1	10	10	± 1%
Red	2	2	100	100	± 2%
Orange	3	3	1K	1000	
Yellow	4	4	10K	10,000	
Green	5	5	100K	100,000	
Blue	6	6	1M	1,000,000	
Violet	7	7	10M	10,000,000	
Grey	8	8	100M	100,000,000	
White	9	9	1000M	1,000,000,000	
Gold				0.1	± 5%
Silver				0.01	± 10%
None					± 20%

Resistor color coding is a way to tell the value of a resistor by looking at its colored bands.
Each color represents a number:

We read the colors from left to right, so the value of the resistor is 1, 0, followed by 2. Combining these numbers, we get 10 * 10^2, which equals 1000 ohms. So, the resistor with brown, black, and red bands has a value of 1000 ohms.

Lesson 4
Flashing External LED Light

Overview

In the previous lesson we worked with the built-in LED on the development board, which does not require any external connections. To further enhance our hands-on abilities, we will now connect an external LED to one of the interfaces on the development board.

By doing this, we can control the external LED's brightness and behavior through our own circuit that we build ourselves.

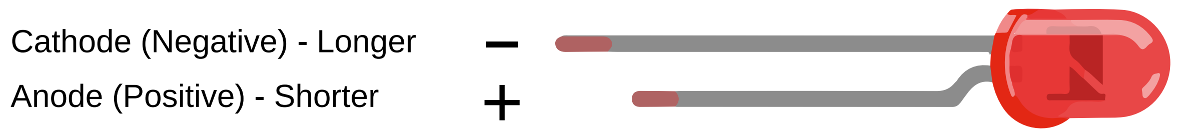
Componets

UNO R3 Compatible Development Board X 1 ARD2-0066
5mm Red LED X 1 X-LED-5MM/R
220 ohm Resistor X 1 RS1405
830 Tie Points Breadboard X 1 MA4009
M-M wires (Male to Male jumper wires) X 2 MA3020

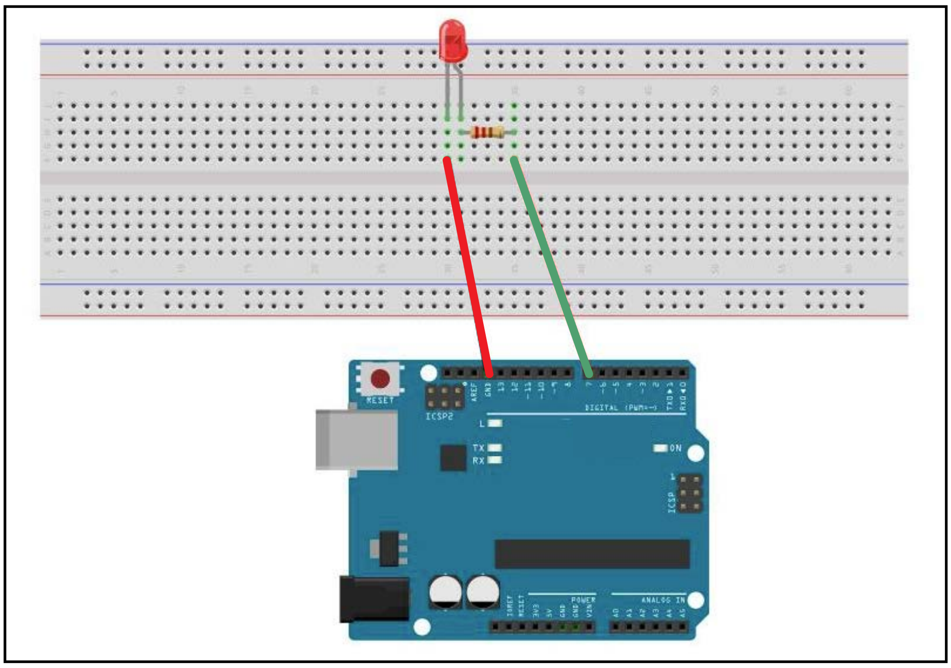
The Relationship between LEDs and Resistors

The relationship between LEDs and resistors is crucial for proper LED operation and to prevent damage to the LED. LEDs are semiconductor devices that require a specific forward voltage to emit light.
If connected directly to a voltage source without a resistor, the LED will draw excessive current and can be permanently damaged.

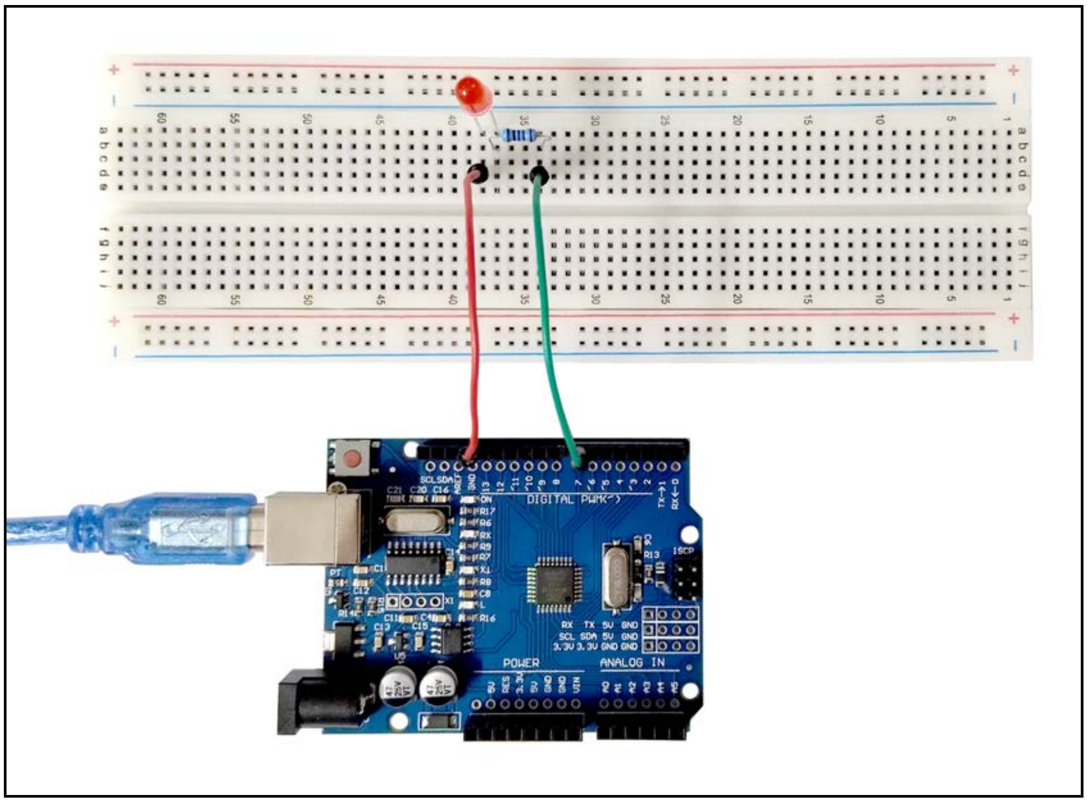
To identify the anode and cathode of an LED, typically the longer leg is the anode (positive) and the shorter leg is the cathode (negative).



Wiring Schematic



Physical Wiring Diagram



Code

```
void setup() {
  // This function runs once when the Arduino is powered on or reset.
  // It is used to initialize settings and set up the pins.

  // Set pin 7 as an OUTPUT to control the LED.
  pinMode(7, OUTPUT);
  // This line tells the Arduino that pin 7 will be used to output signals
  // to control external devices, such as an LED.
}

void loop() {
  // This function runs repeatedly in a loop after the setup() function executes
  // Turn the LED on (HIGH) for 200 milliseconds (0.2 seconds).
  digitalWrite(7, HIGH);
  // This line sends a high voltage (5V) signal to pin 7, turning the LED on.
  // The LED will remain on for the specified delay time.

  delay(200);
  // Pause the program for 200 milliseconds (0.2 seconds).
  // The LED will stay on for 0.2 seconds before the next instruction.

  // Turn the LED off (LOW) for 200 milliseconds (0.2 seconds).
  digitalWrite(7, LOW);
  // This line sends a low voltage (0V) signal to pin 7, turning the LED off.
  // The LED will remain off for the specified delay time.

  delay(200);
  // Pause the program for 200 milliseconds (0.2 seconds).
  // The LED will stay off for 0.2 seconds before the loop starts again.
}
```

Summary

The code controls an external LED connected to pin 7 of the Development Board.
The LED is turned on and off alternately every 0.2 seconds (200 milliseconds) in an infinite loop.
The setup() function initializes the pin 7 as an output, and the loop() function repeatedly executes the commands to control the LED's state, creating a flashing effect.

Lesson 5
Push Button Controlled LED Light

Overview

In this lesson you will learn how to use a button to control an LED and implement the 'delay control function'. By understanding how to interact with buttons and LEDs, you'll be able to open up various possibilities for creating interactive and responsive Arduino projects.

Componets

UNO R3 Compatible Development Board ARD2-0066
5mm Red LED X 1 X-LED-5MM/R
220 ohm Resistor X 1 RS1405
10k ohm Resistor X 1 RS1605
M-M wires (Male to Male jumper wires) X 5 MA3020
830 Tie Points Breadboard X 1 MA4009
Push Button X 1 SW0520

Push Button

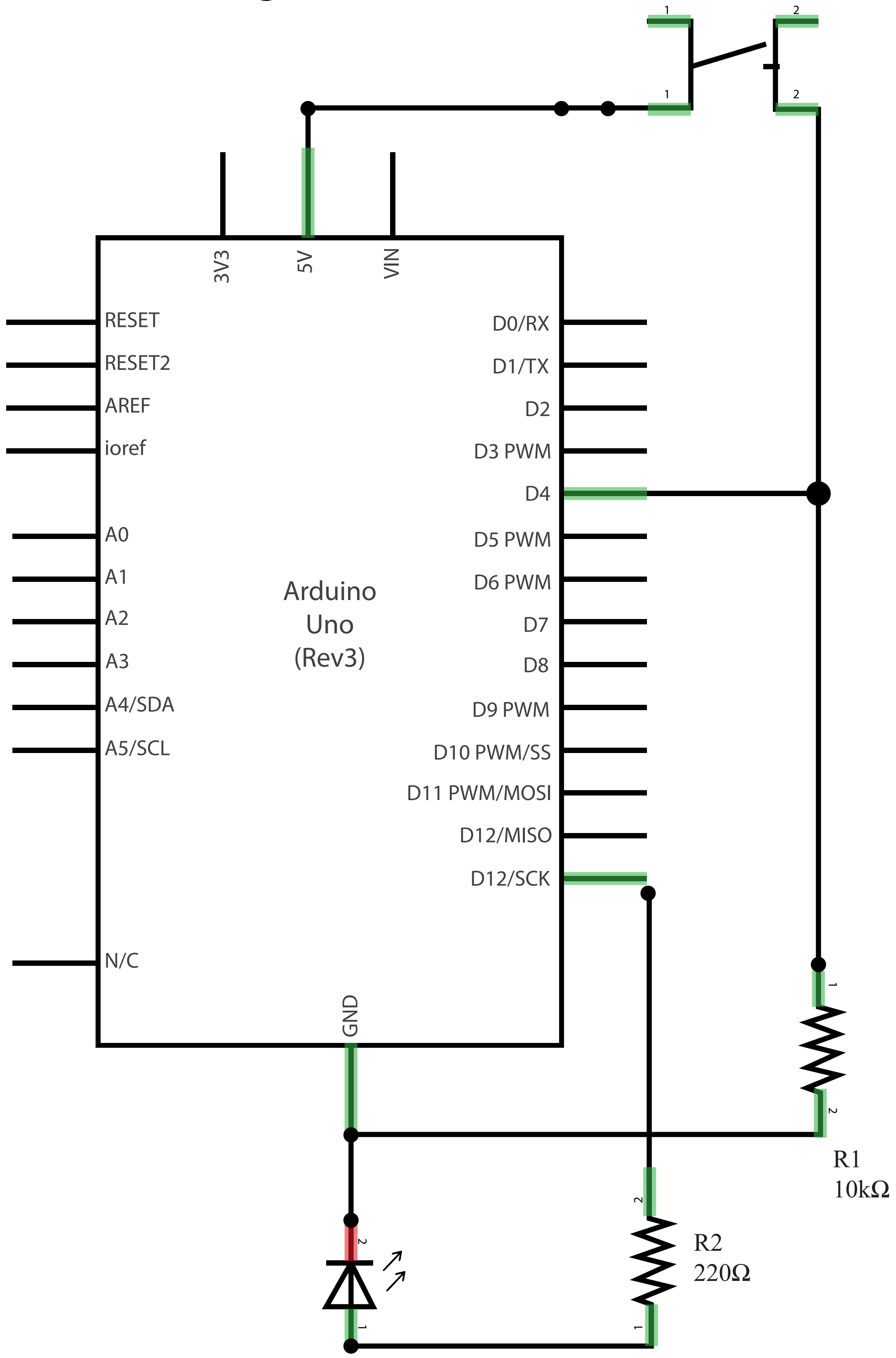
About Buttons

A push button, also known as a momentary switch, is a common input device in electronic circuits. It allows users to provide input to the circuit by pressing and releasing it.

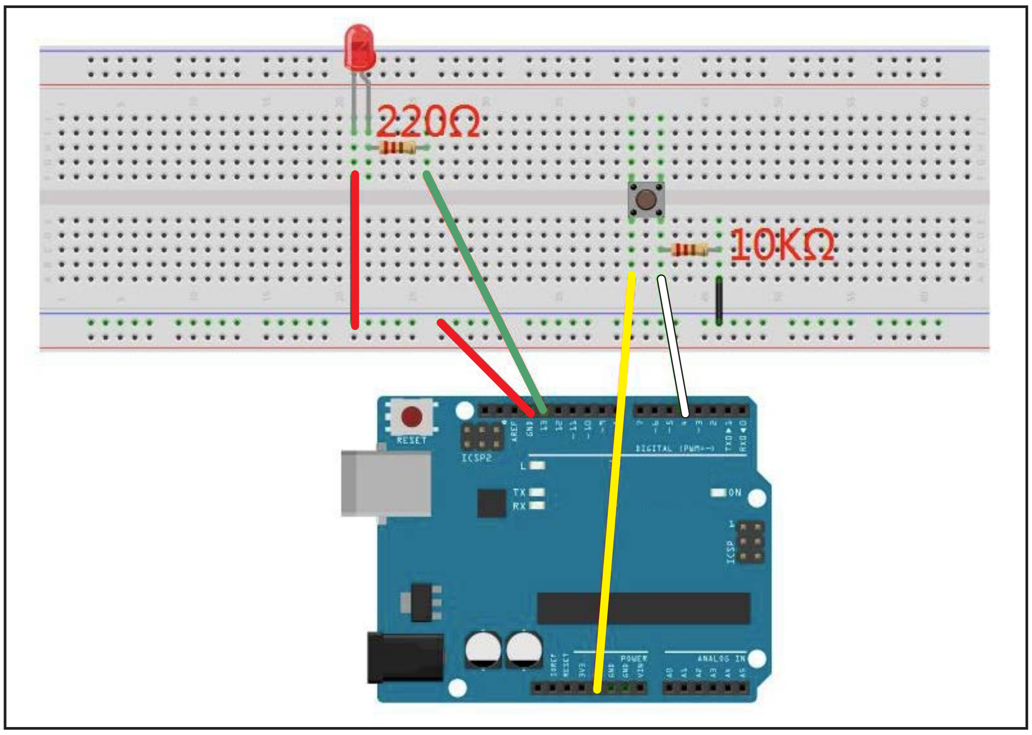
When the button is pressed, it creates a connection between its two terminals, allowing current to flow through the circuit.

When it is released, the connection is broken, interrupting the flow of current.
For the UNO R3 Compatible Development Board, the state of a button is typically read using one of its digital input/output (I/O) pins.
The I/O pin can detect whether the button is pressed or not by reading the voltage level on the corresponding pin. When the button is pressed, the voltage level is considered low (0V or GND), and when the button is not pressed, the voltage level is considered high (5V or Vcc).
The Arduino uses this information to respond to the button's state and perform specific actions.

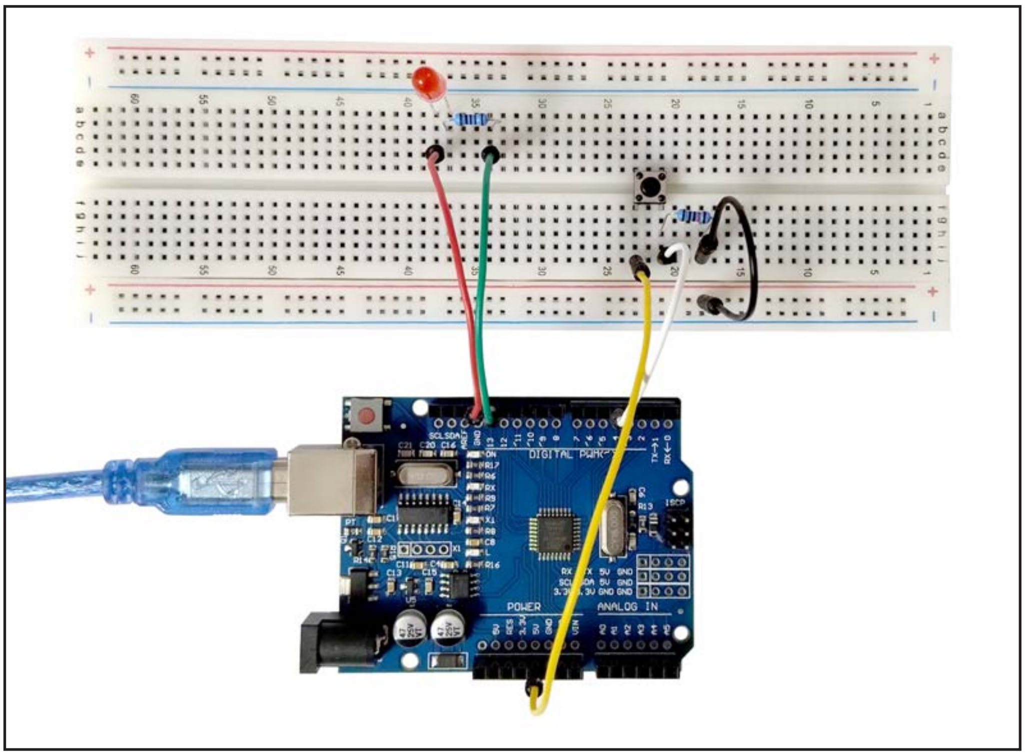
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



Code

```
void setup() {
  // Set pin 4 as input (to read button state) and pin 13 as output (to control LED).
  pinMode(4, INPUT);
  pinMode(13, OUTPUT);
}

void loop() {
  // Read the state of pin 4 (button) and store it in the variable 'n'.
  int n = digitalRead(4);

  // If the button is pressed (HIGH state), proceed with the following actions.
  if (n == HIGH) {
    // Wait for 1 second (1000 milliseconds).
    delay(1000);

// Turn on the LED connected to pin 13.
  digitalWrite(13, HIGH);

  // Wait for 5 seconds (5000 milliseconds).
  delay(5000);

  // Turn off the LED connected to pin 13.
  digitalWrite(13, LOW);
 }
}
```

New Statement “If”

The "if" statement is a conditional statement in programming that allows the code to make decisions based on a particular condition.
If the condition specified in the "if" statement evaluates to true, the code inside the "if" block will be executed; otherwise, it will be skipped, and the program will move on to the next part of the code.

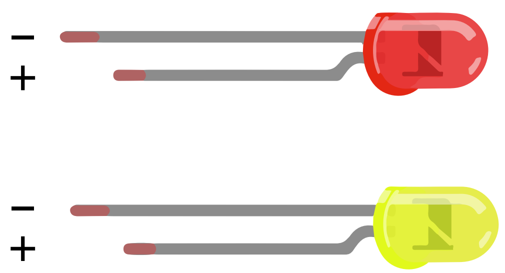
Lesson 6
8 x Cascading LED Lights

Overview

A cascading LED refers to a series of LEDs arranged in such a way that the illumination appears to flow from one LED to the next, creating a visually captivating effect.
Cascading LED effects can produce stunning visual displays that can be synchronized with music or other elements for enhanced impact and engagement.

Componets

UNO R3 Compatible Development Board	ARD2-0066
5mm LEDs X 8	LED-5MM
220 ohm Resistor X 8	RS1425
830 Tie Points Breadboard X 1	MA4009
M-M wires (Male to Male jumper wires) X 17	MA3020



LEDs

About LEDs

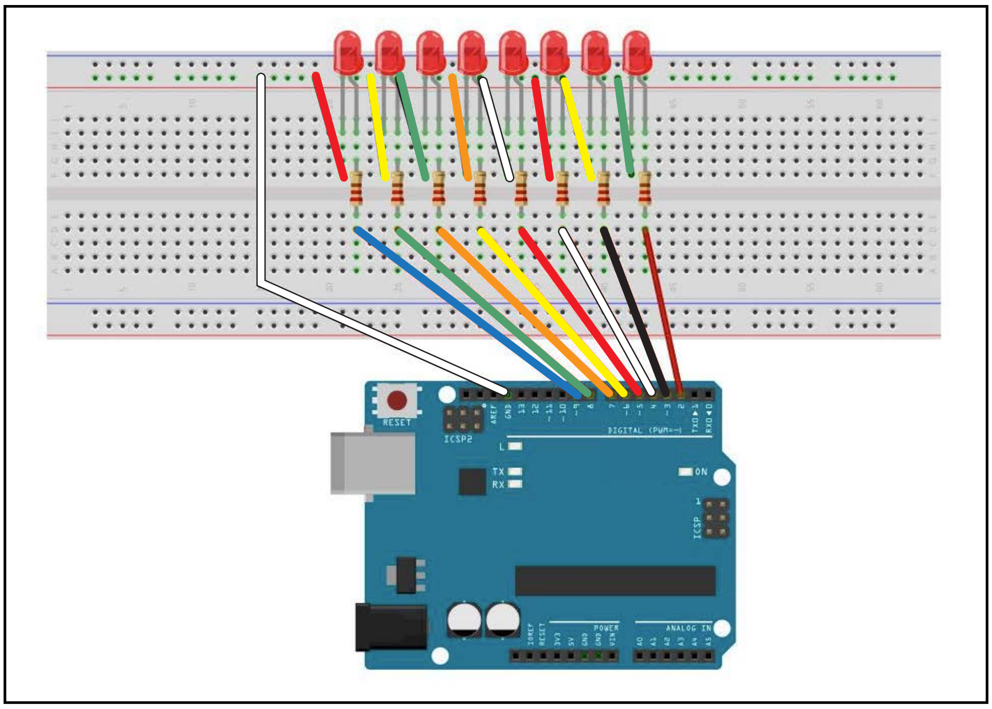
LEDs are polarized components, meaning they have a specific orientation for proper functioning. The two leads of an LED are called the anode and the cathode, and their polarity determines how the LED should be connected in a circuit.

Anode (+): The anode is the longer lead of the LED, and it is the positive (+) side.
It is also referred to as the "p+" or "A+" lead. The anode is where the current enters the LED.

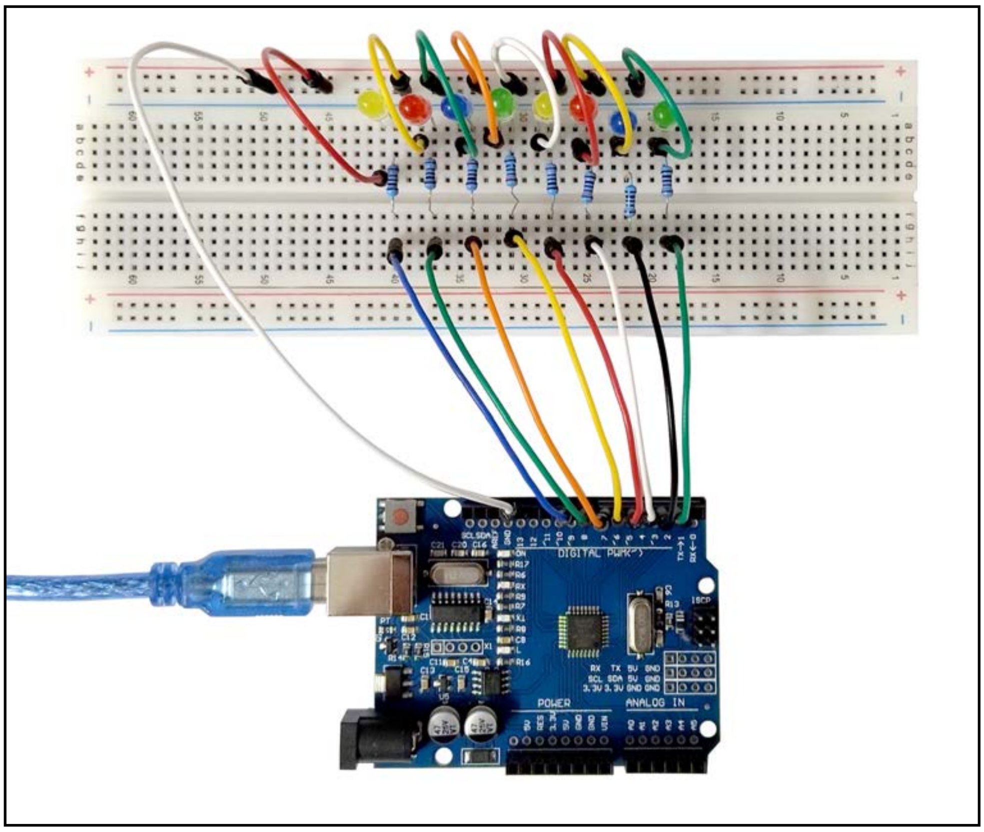
Cathode (-): The cathode is the shorter lead of the LED, and it is the negative (-) side.
It is also referred to as the "p-" or "K" lead. The cathode is where the current exits the LED.

When connecting an LED to a power source, it is crucial to connect the anode to the positive voltage (usually 5V or 3.3V) and the cathode to the negative voltage (ground or GND).
If the LED is connected in reverse, it will not emit light and may become damaged due to excessive reverse voltage.

Wiring Schematic



Physical Wiring Diagram



Code

```
void setup() {
  // Set up pins 1 to 9 as OUTPUT to control external components (e.g., LEDs).
  for (int x = 1; x <= 9; x++) {
    pinMode(x, OUTPUT);
  }
}

void loop() {
  // Loop through pins 1 to 9, turning on each pin (LED) one by one with a delay
  //of 200 milliseconds.
  for (int i = 1; i <= 9; i++) {
    digitalWrite(i, LOW);
    delay(200);
  }

// After turning on all pins, wait for 100 milliseconds.
  delay(100);

// Loop through pins 1 to 9, turning off each pin (LED) one by one with a delay
//of 200 milliseconds.
for (int i = 1; i <= 9; i++) {
  digitalWrite(i, HIGH);
  delay(200);
}

// After turning off all pins, wait for 100 milliseconds before the loop repeats.
delay(100);
}
```

Summary

The 'setup' function prepares pins 1 to 9 to control external components (e.g., LEDs).

In the 'loop' function, it turns on each LED one by one in sequence with a short delay, then turns them all off together with another short delay, creating a cascading effect. This process repeats.

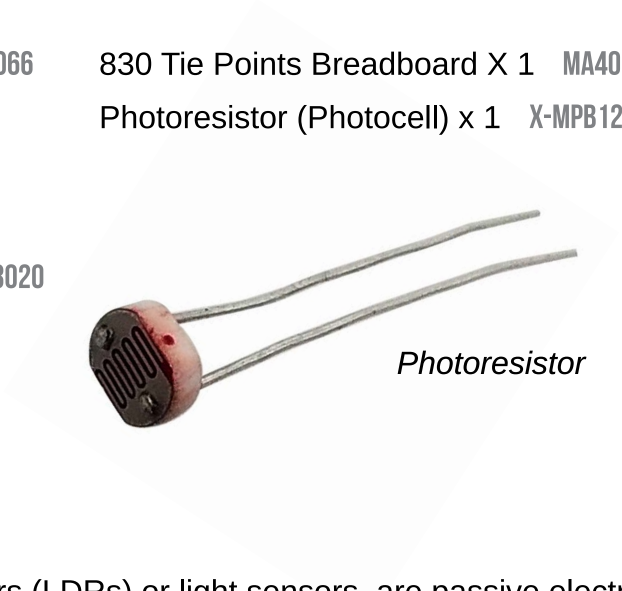
Lesson 7
Photocell LED Experiment

Overview

In this lesson, you will discover how to measure light intensity using an Analog Input. By utilizing the varying level of light, you'll control the number of LEDs that will be illuminated.
This hands-on exercise will help you understand how to use analog sensors to gather data, and how to translate that data into actions, by controlling the brightness or quantity of LEDs

Componets

UNO R3 Compatible Development Board ARD2-0066
5mm LEDs X 8 LED-5MM
220 ohm Resistor X 8 RS1405
10k ohm Resistor X 1 RS1605
M-M wires (Male to Male jumper wires) X 20 MA3020
830 Tie Points Breadboard X 1 MA4009
Photoresistor (Photocell) x 1 X-MPB12C39A

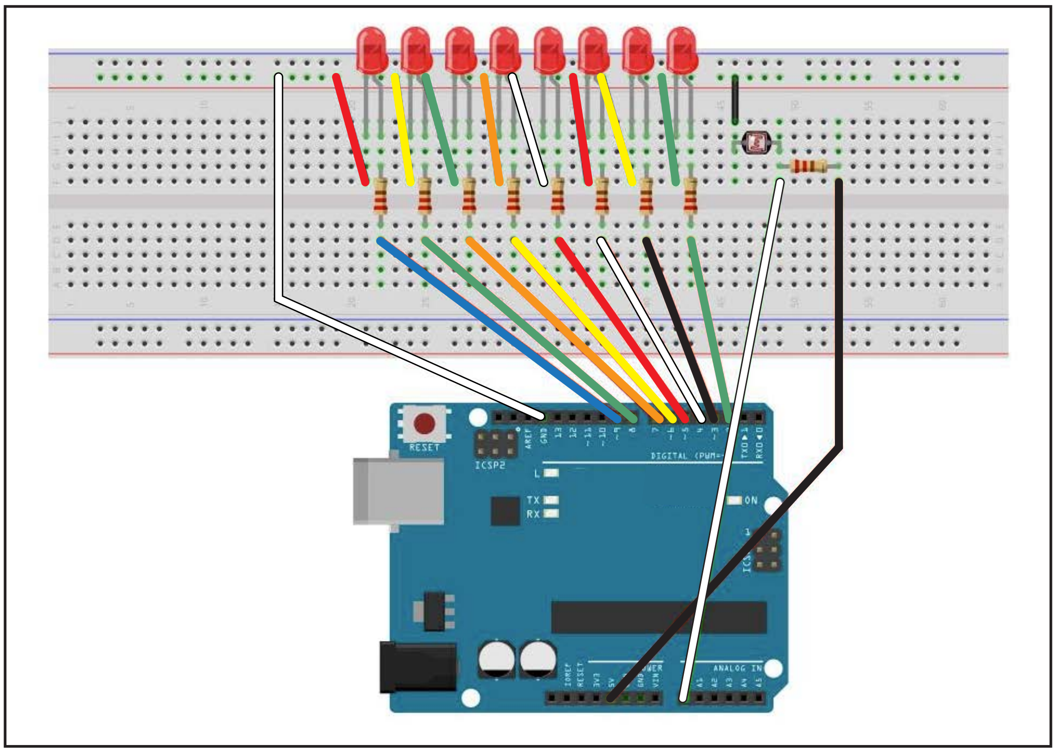

Photoresistor

About Photocells

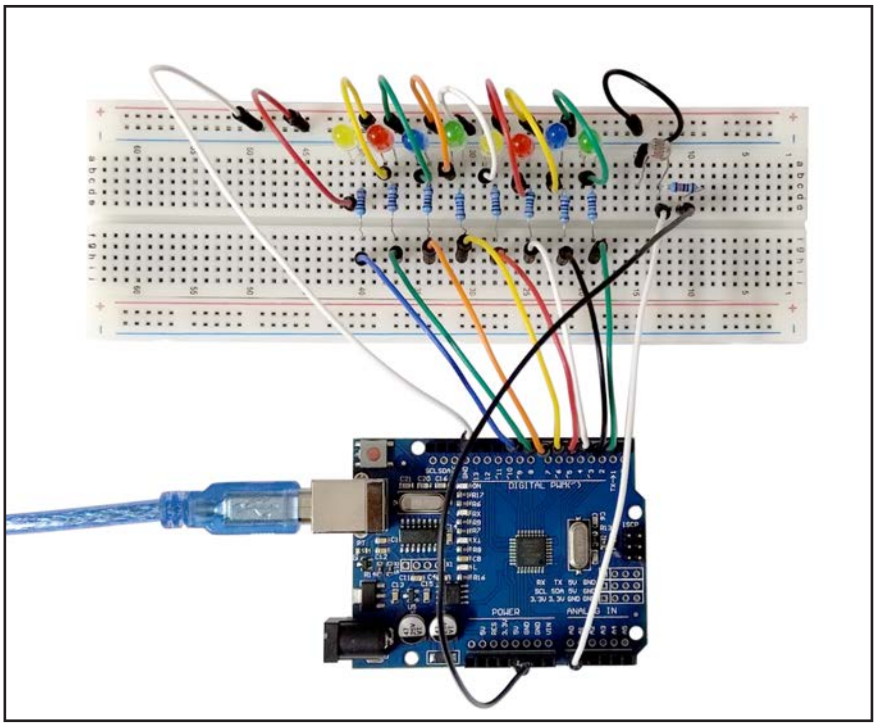
Photocells, also known as light-dependent resistors (LDRs) or light sensors, are passive electronic components that change their resistance based on the amount of light falling on their surface. When exposed to light, the resistance of the photocell decreases, and when in darkness, the resistance increases. This property allows photocells to be used for detecting light levels and controlling circuits based on ambient light conditions.

Photocells are widely used in various applications, such as automatic lighting control systems, outdoor lighting, burglar alarms, and camera exposure control.
They are simple sensors that provide an analog signal representing the light intensity, making them suitable for interfacing with microcontrollers like Arduino for light-based projects.

Wiring Schematic



Physical Wiring Diagram



Code

```
int val = 0;        // Initialize a variable 'val' to store the analog reading.

void setup() {
  int x;
  for (x = 1; x <= 9; x++) {
    pinMode(x, OUTPUT);
// Set up pins 1 to 9 as OUTPUT to control external components (e.g., LEDs).
  }
  Serial.begin(9600);    // Start serial communication at a baud rate of 9600.
}

void loop() {
  int i;
  val = analogRead(A0); // Read the analog value from pin A0 and store it in 'val'.
  Serial.println(val);  // Print the analog value to the Serial Monitor.

if (val > 200) { // If the analog value is greater than 200, execute the following:
  for (i = 1; i <= 9; i++) {
    digitalWrite(i, LOW); // Turn off all LEDs connected to pins 1 to 9.
  }
  delay(200); // Wait for 200 milliseconds.
} else { // If the analog value is not greater than 200, execute the following:
  for (i = 1; i <= 9; i++) {
    digitalWrite(i, HIGH); // Turn on all LEDs connected to pins 1 to 9.
  }
  delay(100); // Wait for 100 milliseconds.
 }
}
```

Summary

The code reads light intensity using an analog photocell sensor and displays the values on the Serial Monitor. Based on the sensor reading, it controls a set of LEDs, turning them off if the light is bright (analog value > 200) and turning them on if the light is dim (analog value ≤ 200).
The loop continuously adjusts LED illumination according to the detected light level.

Lesson 8
Powering 8 LEDs with a 74HC595 Serial to Parallel Converter

Overview

In this lesson, you'll discover a clever way to use eight LEDs with your UNO board without having to sacrifice eight output pins. Typically, you could connect each LED to a separate UNO pin, but that would quickly deplete the available pins, especially when you have other components like buttons, sensors, and servos connected.

To overcome this limitation, we will utilize a special integrated circuit called the 74HC595 **Serial to Parallel Converter.** This chip provides eight outputs, which is perfect for controlling our eight LEDs, and it requires only three inputs to transfer data to it bit by bit.

By using the 74HC595 chip, we can efficiently control multiple LEDs using fewer pins.
Although it may be slightly slower than directly driving the LEDs with individual pins
(about 500,000 times per second compared to 8,000,000 times per second), it's still exceptionally fast, far beyond what humans can detect. Therefore, the slight reduction in speed is well worth the advantage of conserving precious pins for other components like buttons, sensors, and servos.

This technique allows you to expand the capabilities of your UNO board and build more complex projects without worrying about running out of pins.

Componets

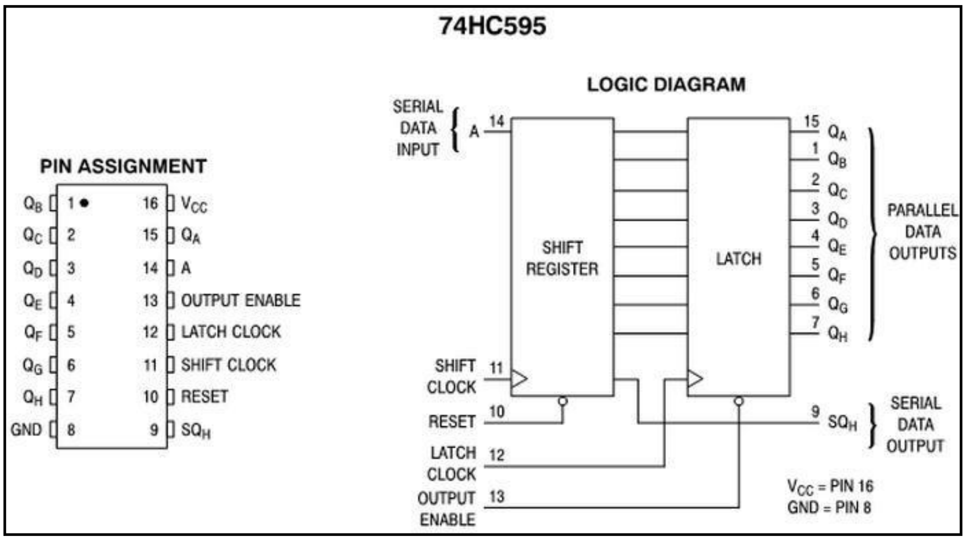
UNO R3 Compatible Development Board ARD2-0066
830 Tie Points Breadboard x 1 MA4009
5mm LEDs X 8 LED-5MM
220 ohm Resistor X 8 RS1405
74HC595 IC X 1 74HC595
M-M wires (Male to Male jumper wires) X 20 MA3020

About the 74HC595

The shift register is a chip that acts like eight memory locations, each storing a 1 or 0.
We use the 'Data' and 'Clock' pins to set these values.

Data Pin: This is where you input the data, bit by bit, which you want to store in the shift register. You shift in the data one bit at a time.

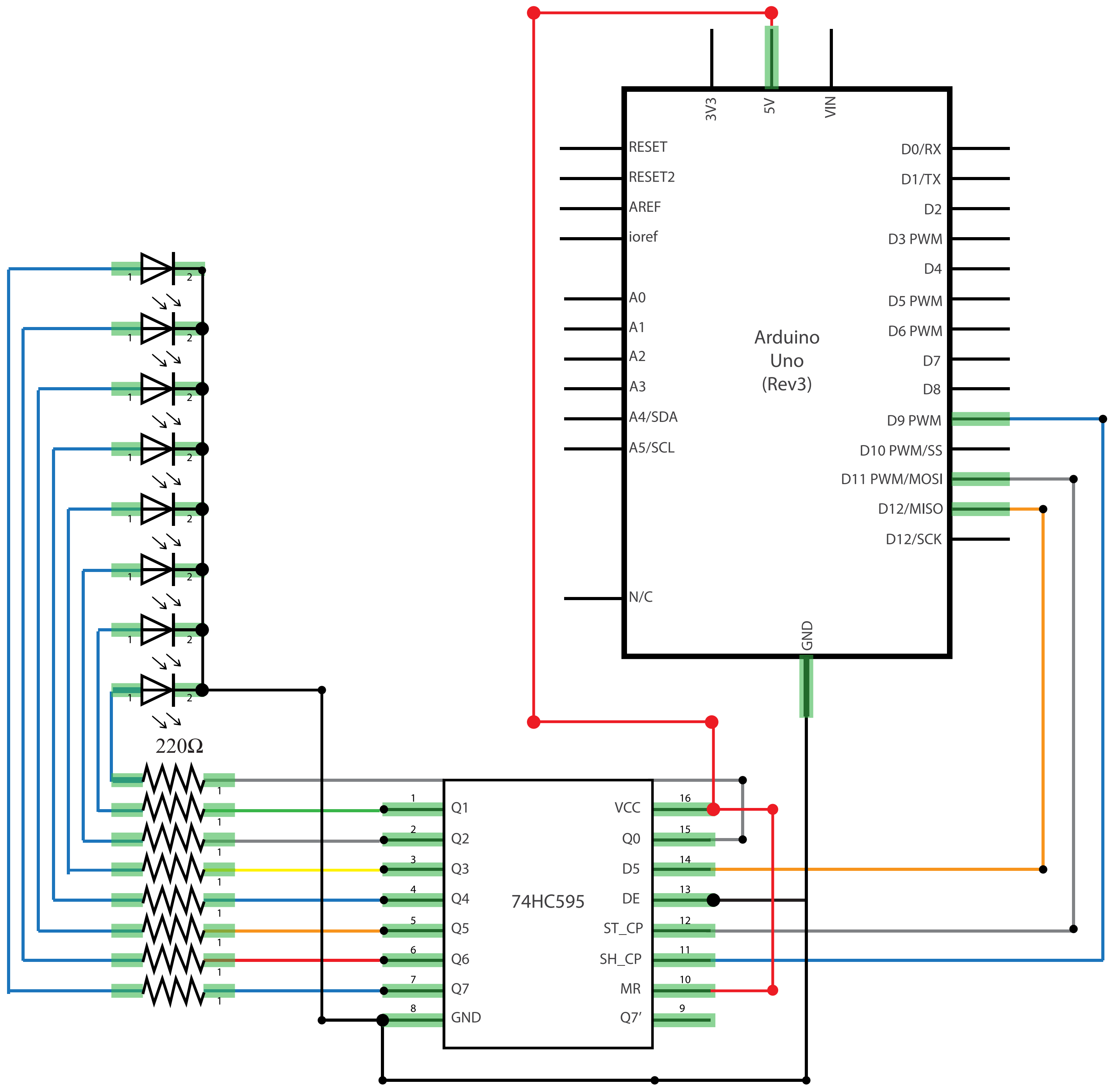
Clock Pin: The "Clock" pin is responsible for synchronizing the shifting process.
For each pulse received on the "Clock" pin, the shift register moves the data one step forward, effectively moving it from one register to the next.



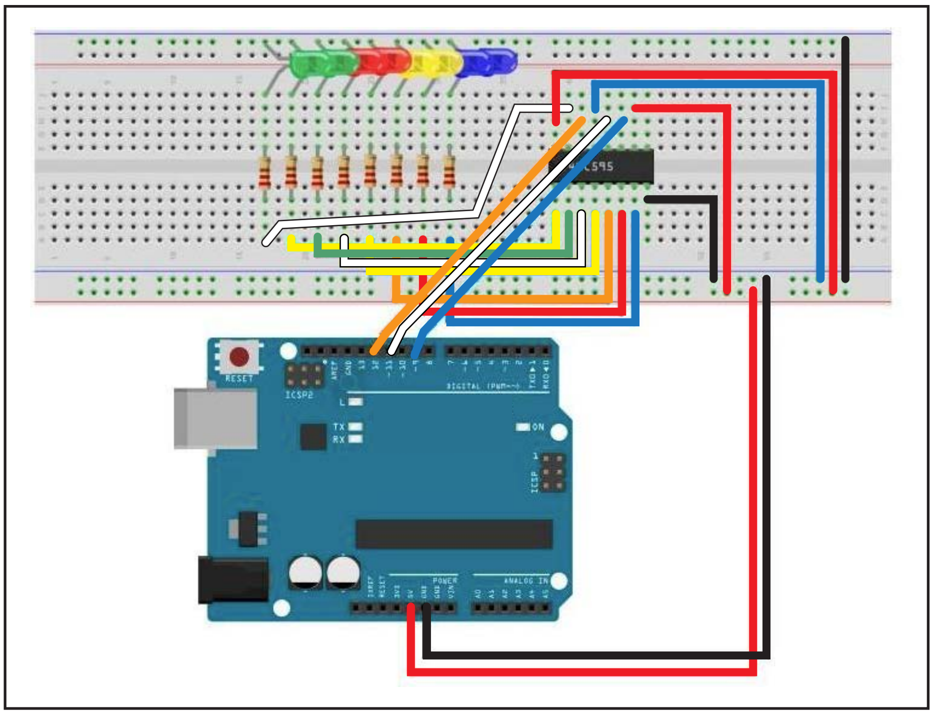
The clock pin must receive eight pulses. Each pulse, when the data pin is high, inputs a 1 into the shift register; otherwise, it inputs a 0.
After all eight pulses are received, activating the 'Latch' pin copies these eight values to the latch register. This step prevents incorrect LED flickering during data loading into the shift register.

The chip also includes an 'Output Enable' (OE) pin, which allows you to enable or disable all the outputs simultaneously. You can attach this pin to a PWM-capable UNO pin and use 'analogWrite' to control the brightness of the LEDs. Since the OE pin is active low, we connect it to GND to enable it.

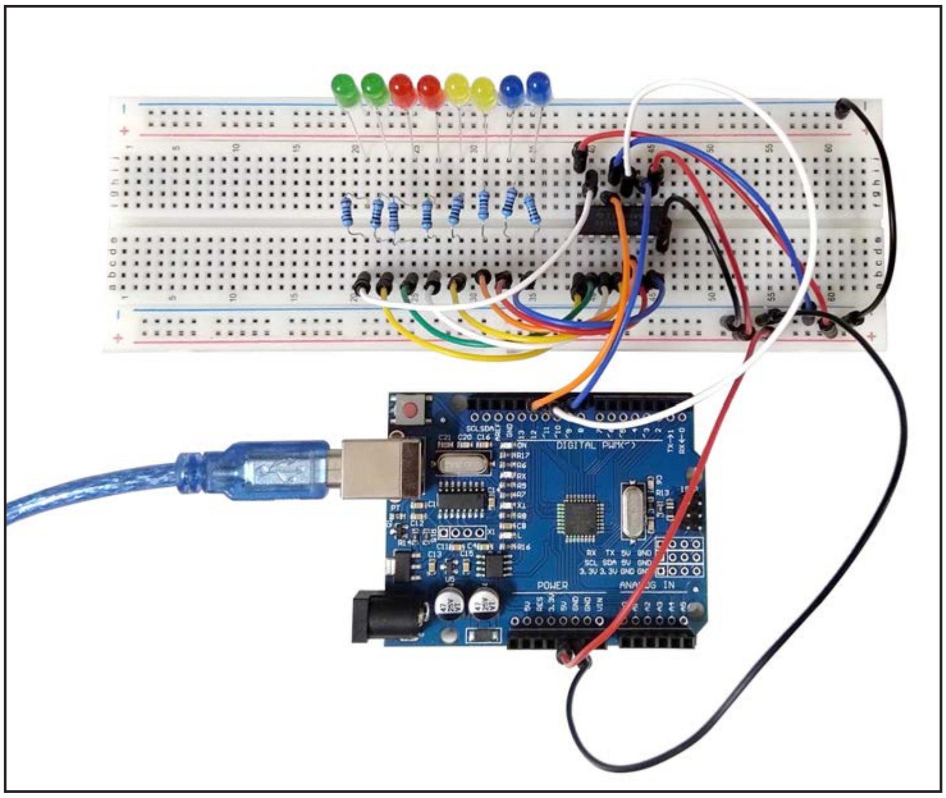
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



Setup Procedure

To connect the components for this project, follow these steps:

Place the 74HC595 chip on the breadboard, ensuring that the U-shaped notch is at the top and pin 1 is to the left of this notch.

Connect the following between the Arduino UNO and the 74HC595 chip:

Digital 12 from UNO to pin #14 of the shift register.
Digital 11 from UNO to pin #12 of the shift register.
Digital 9 from UNO to pin #11 of the shift register.
As most outputs from the 74HC595 chip are on the left side, position the LEDs on the same side for easier connection.

Insert the resistors, being careful not to let their leads touch each other.
Consider shortening the leads if they cause difficulties.

Place the LEDs on the breadboard with their longer positive leads facing towards the 74HC595 chip, regardless of which side of the breadboard they are on.

Connect jumper wires according to the diagram provided. Don't forget the one that goes from pin 8 of the 74HC595 chip to the GND column of the breadboard.

Upload the provided sketch to your Arduino UNO. The LEDs should light up one by one until all are on, then turn off, and the cycle repeats.

With these connections and the uploaded sketch, your setup should be complete, and you should see the LEDs blinking in the desired sequence.

Code

```
// Define the pin numbers for the shift register
int latchPin = 11;          // Connected to the latch pin of the shift register
int clockPin = 9;           // Connected to the clock pin of the shift register
int dataPin = 12;           // Connected to the data pin of the shift register

byte leds = 0;              // Variable to store the pattern of LEDs to be displayed

void setup() {
  // Set the pin modes for the shift register
  pinMode(latchPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
}

void loop() {
  leds = 0;                        // Clear the LED pattern
  updateShiftRegister();           // Update shift register with the current LED pattern
  delay(500);                      // Wait for 500 milliseconds

for (int i = 0; i < 8; i++) {
  bitSet(leds, i);                 // Set one LED at a time in the LED pattern
  updateShiftRegister();           // Update the shift register with updated LED pattern
  delay(500);                      // Wait for 500 milliseconds

  // The loop will repeat for all LEDs, creating a scrolling effect
 }
}
// Function to update the shift register with the LED pattern
void updateShiftRegister() {
  digitalWrite(latchPin, LOW); // Set latch pin LOW to start data transmission
  shiftOut(dataPin, clockPin, LSBFIRST, leds);  // Send the LED pattern to the
                                                //shift register
  digitalWrite(latchPin, HIGH); // Set latch pin HIGH to update the shift register
}
```

Summary

This code controls a shift register to display patterns of LEDs.
It uses three pins (latchPin, clockPin, and dataPin) to interface with the shift register.
The leds variable stores the LED pattern to be displayed.

In the setup() function, the pin modes for the shift register control pins are set to OUTPUT.

In the loop() function, the code displays different LED patterns.
It starts by clearing the LED pattern (leds = 0), updates the shift register, and waits for 500 milliseconds.
Then, it goes through a loop to set one LED at a time in the LED pattern, updates the shift register with the updated pattern, and waits for 500 milliseconds.

This loop creates a scrolling effect by displaying LEDs one after the other.

Lesson 9
Powering an RGB LEDs

Overview

RGB LEDs are colorful and easy to use in projects.
They have three LEDs in one package: Red, Green, and Blue.
You can connect them like regular LEDs using either 9V (Common Anode)
or Ground (Common Cathode) type. This lesson uses a Common Cathode LED

To avoid damage, use one of the three resistors with each LED’s Anode Connection (+).
In this lesson, the LED smoothly transitions from Red to Green, then Blue, and back to Red, creating a cycling effect that showcases different colors.

Componets

UNO R3 Compatible Development Board ARD2-0066
830 Tie Points Breadboard x 1 MA4009
220 ohm Resistor X 3 RS1405
RGB (Common Cathode -CC) LED X 1 X-LED-RGBCA
M-M wires (Male to Male jumper wires) X 4 MA3020

About RGB LEDs

RGB LEDs are special because they contain three LEDs in one package: red, green, and blue. By adjusting the brightness of each LED, you can create any color you want, just like mixing paint on a palette.

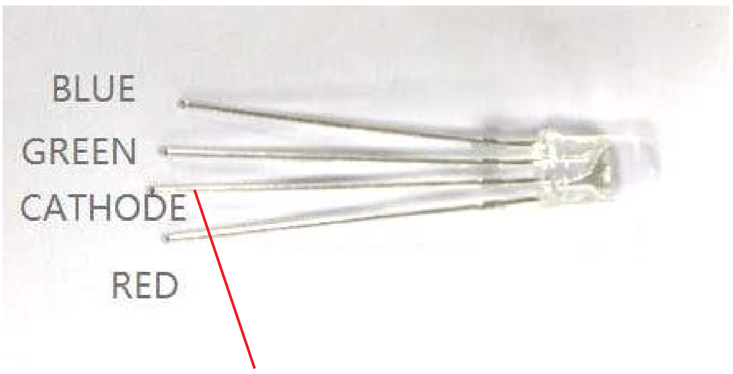
Instead of using different resistors, you can use the **analogWrite** function with **~ marked pins** on the UNO R3 board to control the brightness of each LED easily.

This allows you to create various colors without much effort.

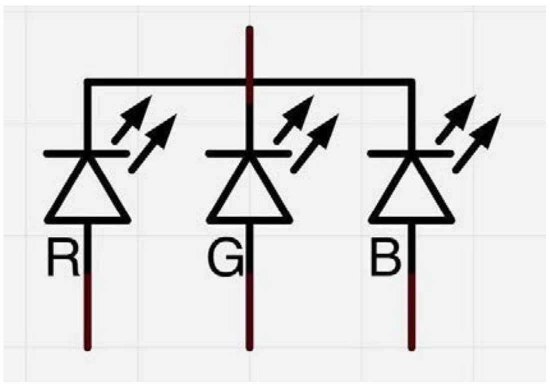
About RGB LEDs (Cont)

RGB LEDs have four leads.
Three leads are for the positive connections of the individual red, green, and blue LEDs inside the package.

The fourth lead is for the common negative side of all three LEDs.
By controlling the voltage applied to the positive leads and the common cathode pppnegative lead, you can mix colors and create various lighting effects using RGB LEDs.



The Common Cathode (-)
has a longer lead

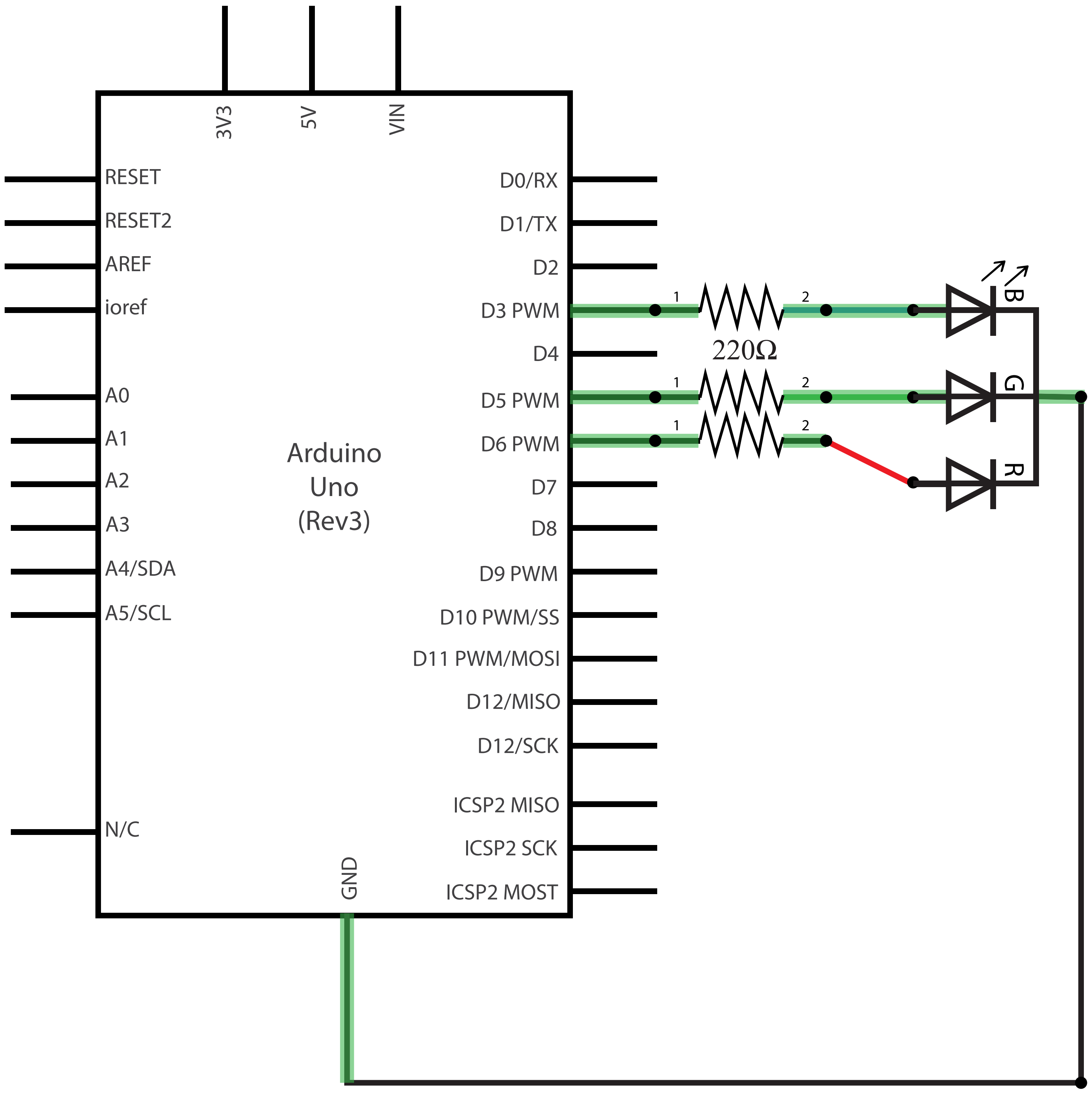


Each separate pin for green, blue, or red is called an Anode and should be connected to the positive side.
The common cathode (-) connection is the second pin from the flat side and is connected to ground.

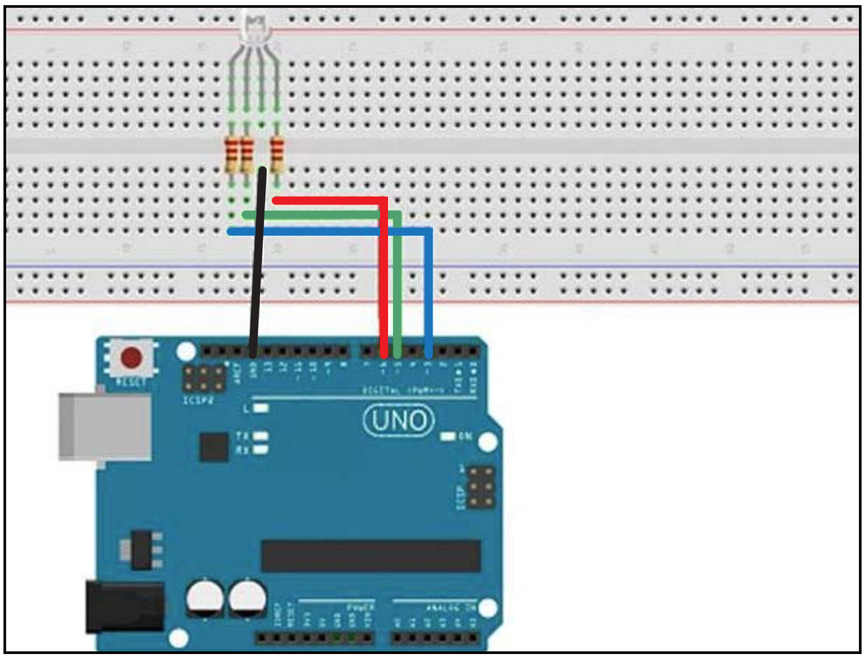
To prevent excessive current, each LED inside the package requires its own 220Ω resistor.
The positive leads of the LEDs (one red, one green, and one blue) are connected to the UNO output pins using these resistors.

Connecting the LED the wrong way round will prevent it from lighting up.

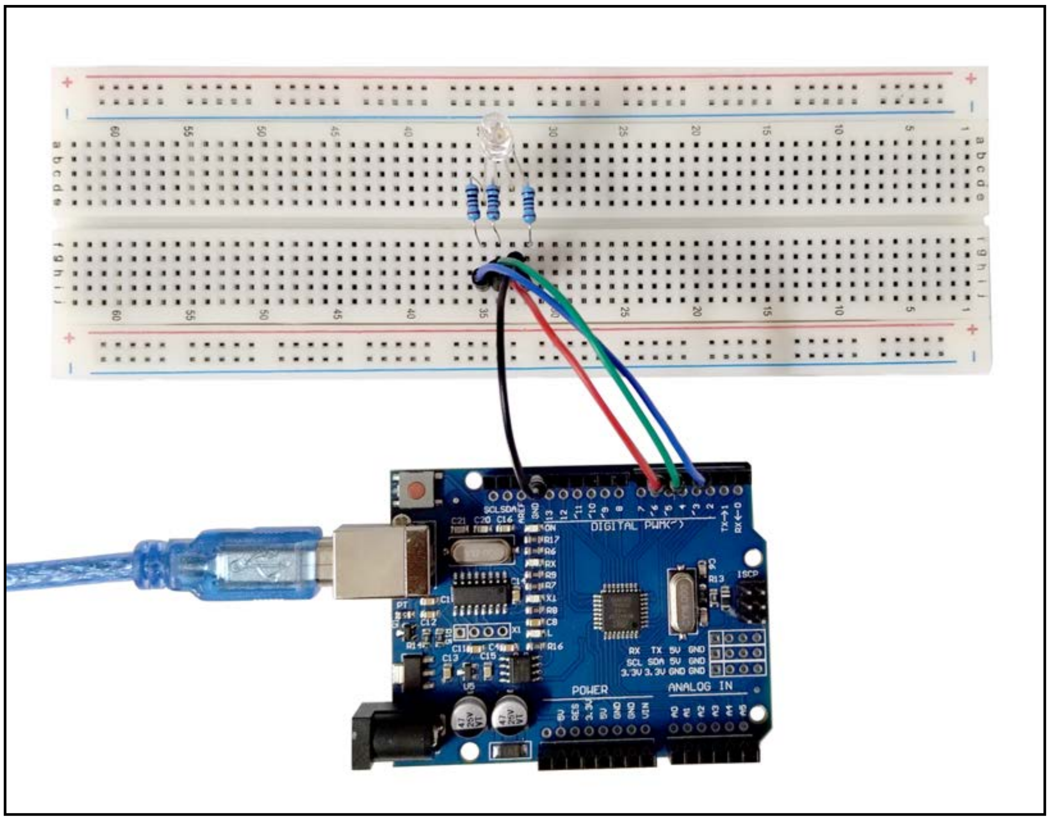
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



Code

```
// Define the pin numbers for the RGB LED
#define BLUE 3      // Pin number for the blue LED
#define GREEN 5     // Pin number for the green LED
#define RED 6       // Pin number for the red LED

void setup() {
  // Set the pin modes for the RGB LED
  pinMode(RED, OUTPUT);       // Set the red pin as an output
  pinMode(GREEN, OUTPUT);     // Set the green pin as an output
  pinMode(BLUE, OUTPUT);      // Set the blue pin as an output

  // Turn on the red LED and turn off the green and blue LEDs
  digitalWrite(RED, HIGH);    // Set the red pin to HIGH (ON)
  digitalWrite(GREEN, LOW);   // Set the green pin to LOW (OFF)
  digitalWrite(BLUE, LOW);    // Set the blue pin to LOW (OFF)

  // put your setup code here, to run once:
}

void loop() {
  // Initialize the RGB values
  int redValue = 255;         // Full intensity (255) for red color
  int greenValue = 0;         // No intensity (0) for green color
  int blueValue = 0;          // No intensity (0) for blue color

  // put your main code here, to run repeatedly:
  int delayTime = 10;         // Delay time in milliseconds

  // Insert any additional code to control the RGB LED colors here

  delay(delayTime);           // Wait for specified delay time before looping again
}
```

Summary

The code defines three constants (RED, GREEN, and BLUE) representing the pin numbers for each color of the RGB LED. In the setup() function, it sets these pins as outputs and sets the initial state to have the red LED on and the green and blue LEDs off.

In the loop() function, it initializes the intensity values for each color (redValue, greenValue, and blueValue).

You can insert additional code in the loop() to control the RGB LED colors by adjusting the intensity values.

After any additional code, it adds a delay of 10 milliseconds before looping again.

This allows you to control the colors and create various effects using the RGB LED.

Lesson 10
Active Buzzer

Overview

In this lesson, you will learn how to generate a sound with an active buzzer.
You will also install the library file “piches.h”

Componets

UNO R3 Compatible Development Board ARD2-0066
830 Tie Points Breadboard x 1 MA4009
Active Buzzer X 1 AC3008
M-M wires (Male to Male jumper wires) X 2 MA3020

Active Buzzer

About Buzzers

Electronic buzzers are devices powered by direct current (DC) and consist of an integrated circuit. They find extensive applications in various electronic products, such as computers, printers, alarms, electronic toys, and more, where sound alerts are necessary.

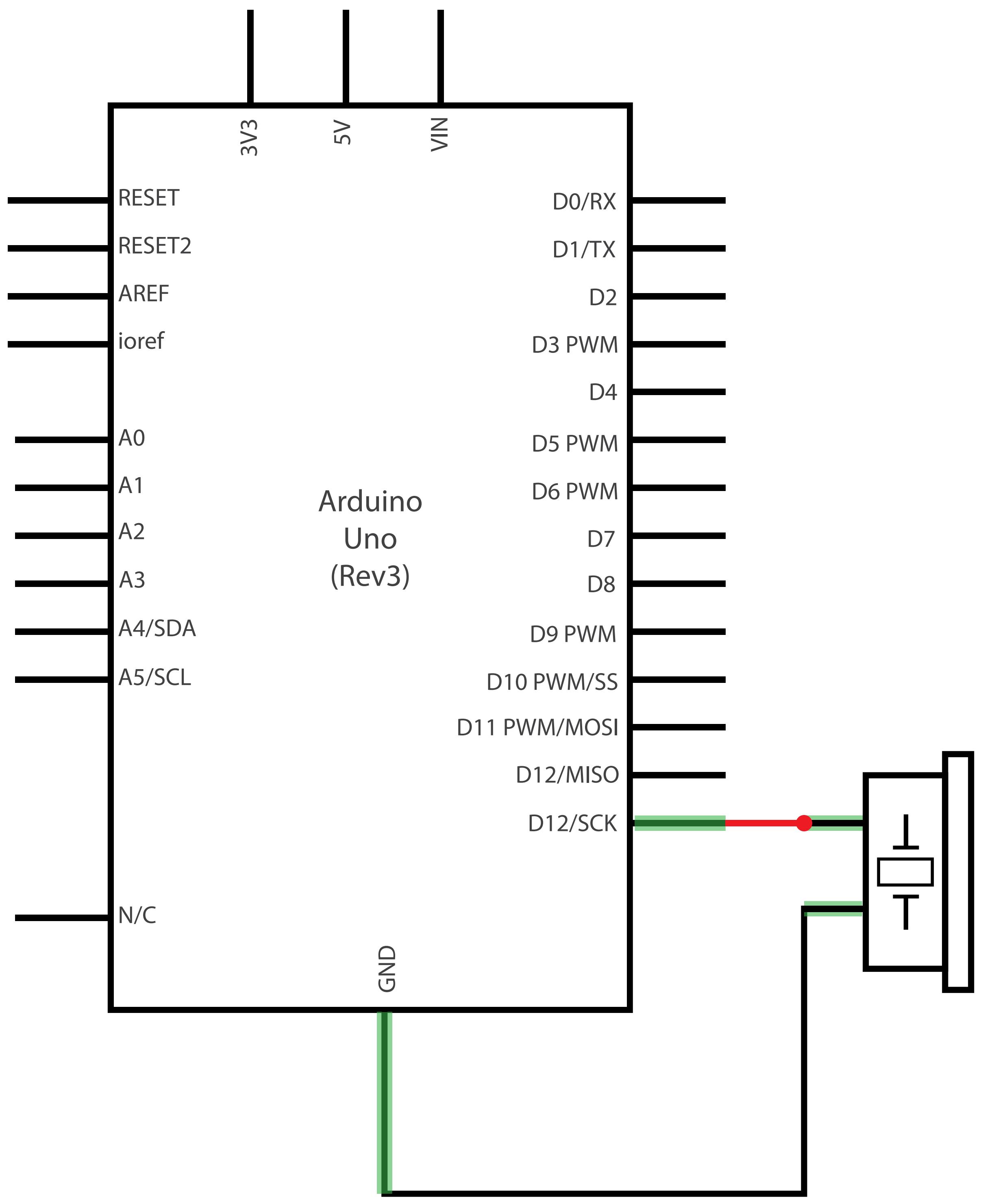
There are two main types of buzzers: active and passive.

To distinguish between the two types, you can examine their appearance.
Turn the pins of two buzzers face up.
The buzzer with a green circuit board is a passive buzzer, whereas the one enclosed with a black tape is an active buzzer.

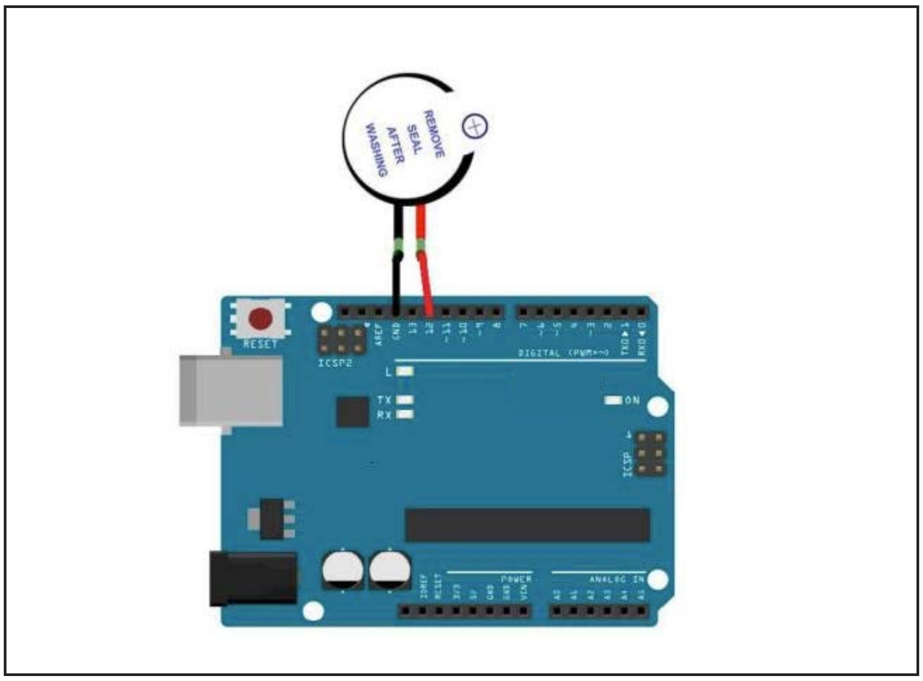
The active buzzer has a built-in oscillator circuit, so it can generate sound by itself when provided with a DC signal.

On the other hand, the passive buzzer requires an external oscillating signal from a microcontroller or other source to produce sound.

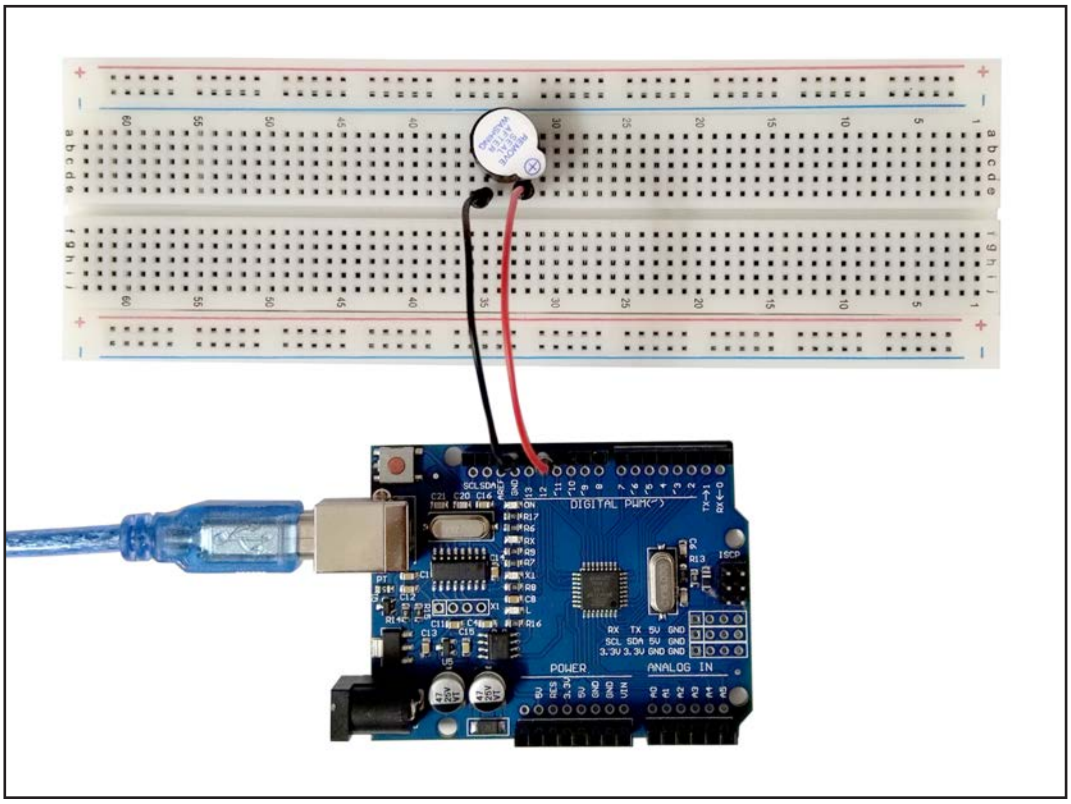
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



Downloading and Installing “pitches.h” Library

The "pitches.h" library is not a built-in library in the Arduino IDE.
However, it is commonly used for working with musical notes and frequencies in Arduino projects.

To obtain the "pitches.h" library, you can follow these steps:

1. Visit the Wiltronics Website
   https://www.wiltronics.com.au/product/72832/ard2-arduino-compatible-starter-kit-uno-r3-with-16-projects/
2. Download the **pitches.h** ZIP file

3. Extract the downloaded ZIP file. Inside, you will find a folder named "arduino-songs-master".

4. Open your Arduino IDE and go to "Sketch" -> "Include Library" -> "Add .ZIP Library".

5. Navigate to the extracted "arduino-songs-master" folder and select it.

The Arduino IDE will then import the library, and you should be able to use the "pitches.h" library in your sketches.

Alternatively, you can manually copy the "pitches.h" file from the library's "pitches" folder into your Arduino project's folder. In this case, make sure the file is named "pitches.h" and placed in the same directory as your Arduino sketch (.ino) file.

After importing the library, you can include the "pitches.h" header in your code using #include "pitches.h", as you did in your original code snippet.

Code

```
// Include the "pitches.h" library, which contains musical note definitions
#include "pitches.h"

// Define the melody as an array of notes
int melody[] = {
  NOTE_C5, NOTE_D5, NOTE_E5, NOTE_F5, NOTE_G5, NOTE_A5, NOTE_B5, NOTE_C6
};

// Define the duration (time in milliseconds) for each note
int duration = 500;

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:

  // Loop through each note in the melody
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    // Play the current note using the tone function
    // Arguments:
    // - Pin number (12) where the piezo buzzer is connected
    // - Frequency of the current note from the melody array
    // - Duration of the note
    tone(12, melody[thisNote], duration);

    // Wait for the duration of the note before playing the next one
    delay(1000);
  }

  // After playing the entire melody, wait for 2 seconds before starting again
  delay(2000);
}
```

Lesson 11
Passive Buzzer

Overview

In this lesson, you will learn how to use a passive buzzer to generate eight different sounds, each lasting 0.5 seconds.
The buzzer will produce musical notes ranging from "Alto Do" (523Hz) to "Treble Do" (1047Hz).

You will be programming the Arduino to play these notes one after the other using the passive buzzer. Each note will play for 0.5 seconds before moving on to the next one, creating a sequence of musical sounds.

Componets

UNO R3 Compatible Development Board	ARD2-0066
Passive Buzzer X 1	SP1200
M-M wires (Male to Male jumper wires) X 1	MA3020

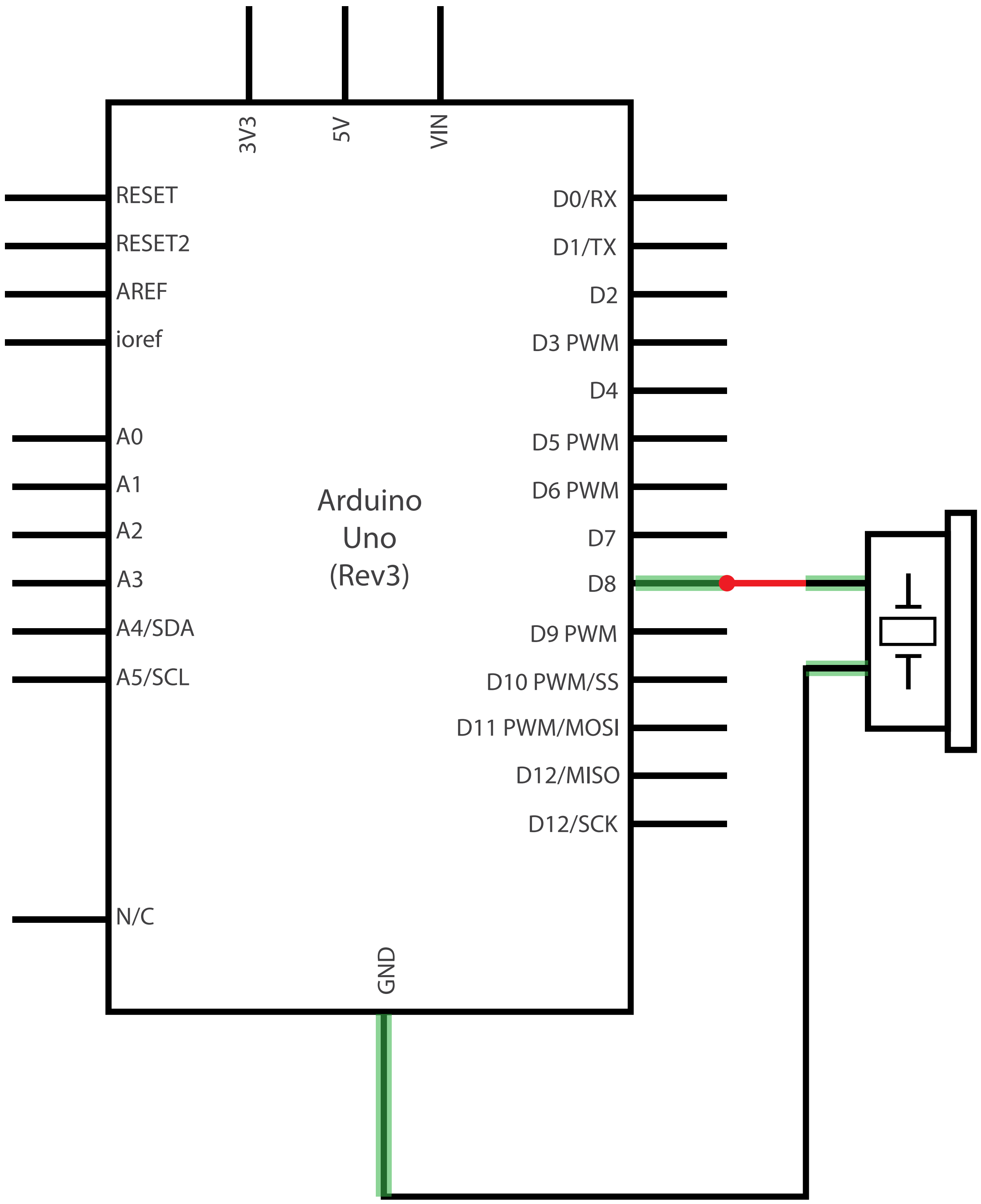
Passive Buzzer

About Passive Buzzers

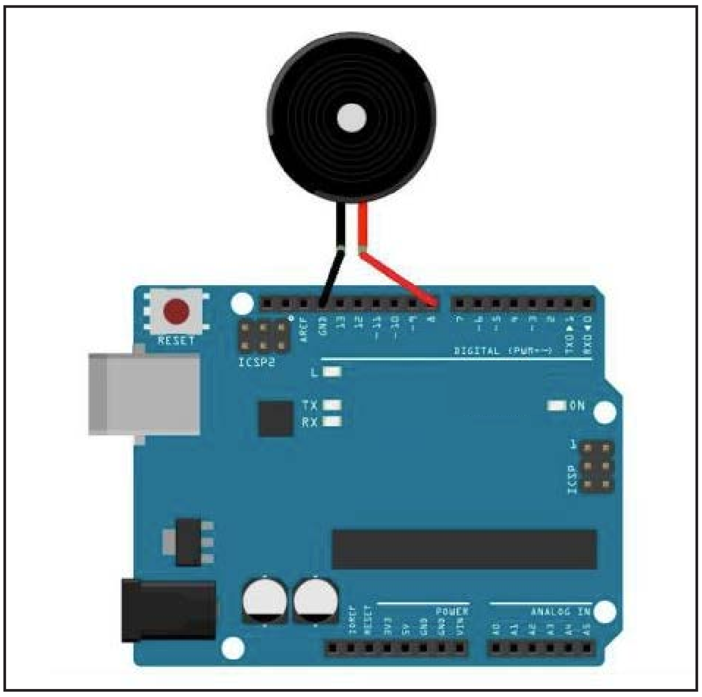
The passive buzzer works by using Pulse Width Modulation (PWM) to generate audio and create vibrations in the air. By changing the frequency of the vibrations, different sounds can be produced. For example, a pulse with a frequency of 523Hz generates "Alto Do," 587Hz produces "midrange Re," and 659Hz results in "midrange Mi." By controlling the buzzer's pulses, you can play a song.

It is essential not to use the UNO R3 Compatible Development Board’s analogWrite() function to generate pulses for the buzzer because this function provides a fixed pulse output at 500Hz. Instead, you need to use other methods, such as direct control of digital pins, to achieve the desired frequencies and play different musical notes with the passive buzzer.

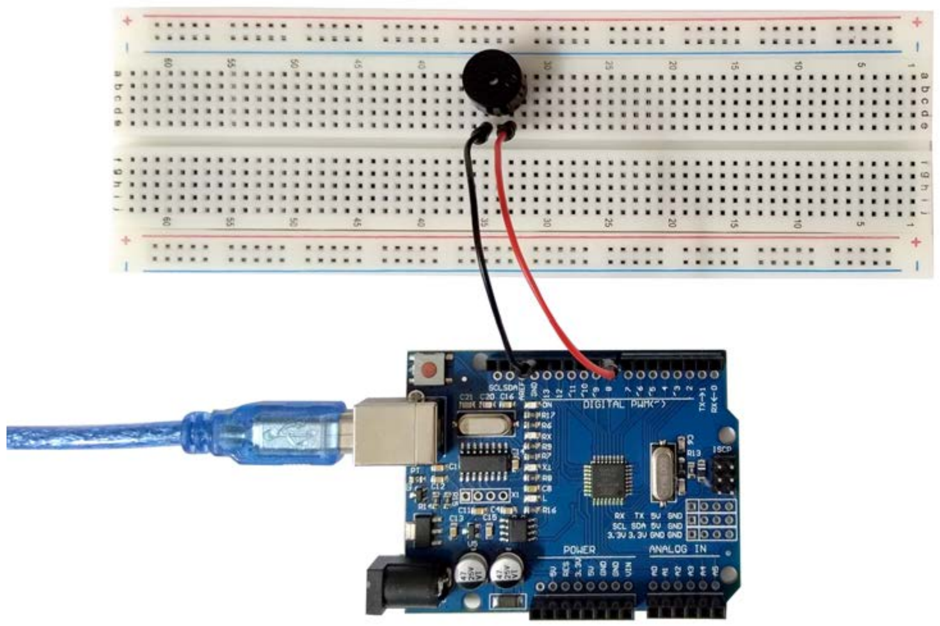
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



Code

```
void setup() {
  // The setup function is empty in this code
  // Put any one-time setup code here (if needed), to run once:
}
void loop() {
  // The loop function is where the main code executes repeatedly

  // Increase the frequency from 200Hz to 800Hz in a loop
  for (int i = 200; i <= 800; i++) {
    // Set pin 8 as an output to connect the passive buzzer
    pinMode(8, OUTPUT);
    // Output the current frequency (i) to the buzzer on pin 8
    tone(8, i);
    // This frequency is maintained for 5 milliseconds
    delay(5);
  }
  // Hold for 4 seconds at the highest frequency (800Hz)
  delay(4000);

  // Decrease the frequency from 800Hz to 200Hz in a loop
  for (int i = 800; i >= 200; i--) {
    // Set pin 8 as an output to connect the passive buzzer
    pinMode(8, OUTPUT);
    // Output the current frequency (i) to the buzzer on pin 8
    tone(8, i);
    // A short delay of 10 milliseconds between each frequency change
    delay(10);
  }
}
```

New Function “tone”

The "tone" command can be used to produce various musical notes or simple sounds with the help of a passive buzzer or piezo speaker connected to the UNO R3 Compatible Development Board. Additionally, there is a corresponding "noTone" command to stop the sound output on the specified pin.

Lesson 12
Tilt Ball Switch

Overview

In this lesson, you will learn how to use a tilt ball switch to detect small angles of inclination.
By connecting the tilt ball switch to an UNO R3 Compatible Development Board, you can monitor its state and detect when the switch is tilted.
This can be useful for various applications, such as detecting the orientation of a device or creating interactive projects that respond to tilting movements.

Componets

UNO R3 Compatible Development Board ARD2-0066
Tilt Ball Switch X 1
F-M wires (Female to Male jumper wires) X 1 CN3602

Tilt Ball Switch

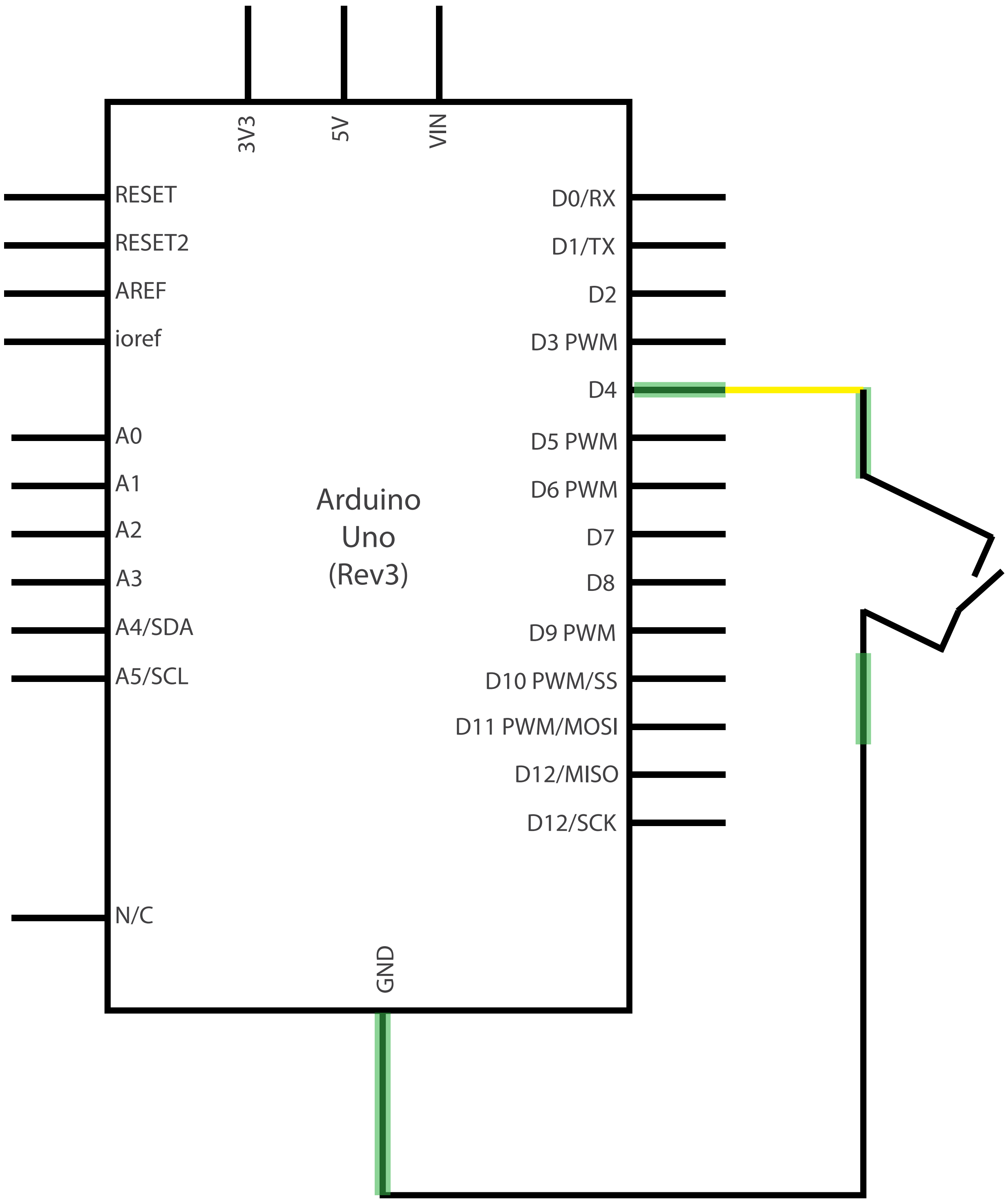
About Tilt Ball Switches

Tilt ball switches, are useful devices for detecting orientation or inclination.
Due to their simple design, they are commonly found in toys, gadgets, and appliances.
Tilt sensors are sometimes referred to as "mercury switches," or "rolling ball sensors" because they often contain a conductive mass like mercury or a rolling ball.

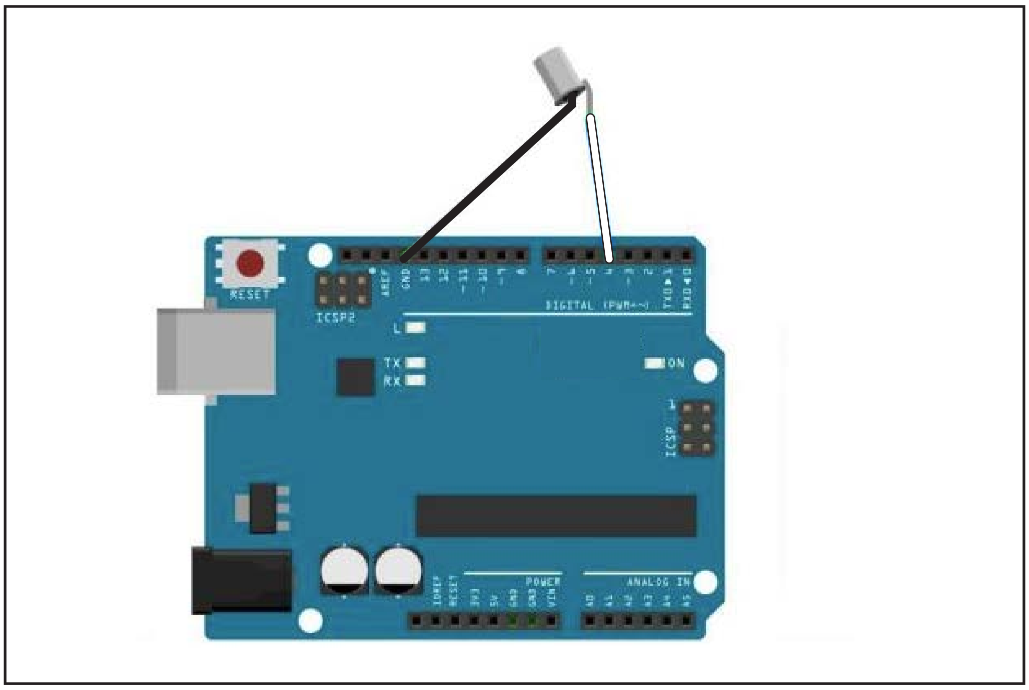
The basic structure of a tilt sensor includes a cavity with a conductive mass inside, and two conductive elements (poles) at one end. When the sensor is tilted in a certain direction, the conductive mass rolls onto the poles and creates a short circuit, acting as a switch.
While not as precise as accelerometers, tilt switches can still detect motion or orientation.

One advantage of tilt switches is that some larger ones can switch power on their own, making them self-sufficient in certain applications.

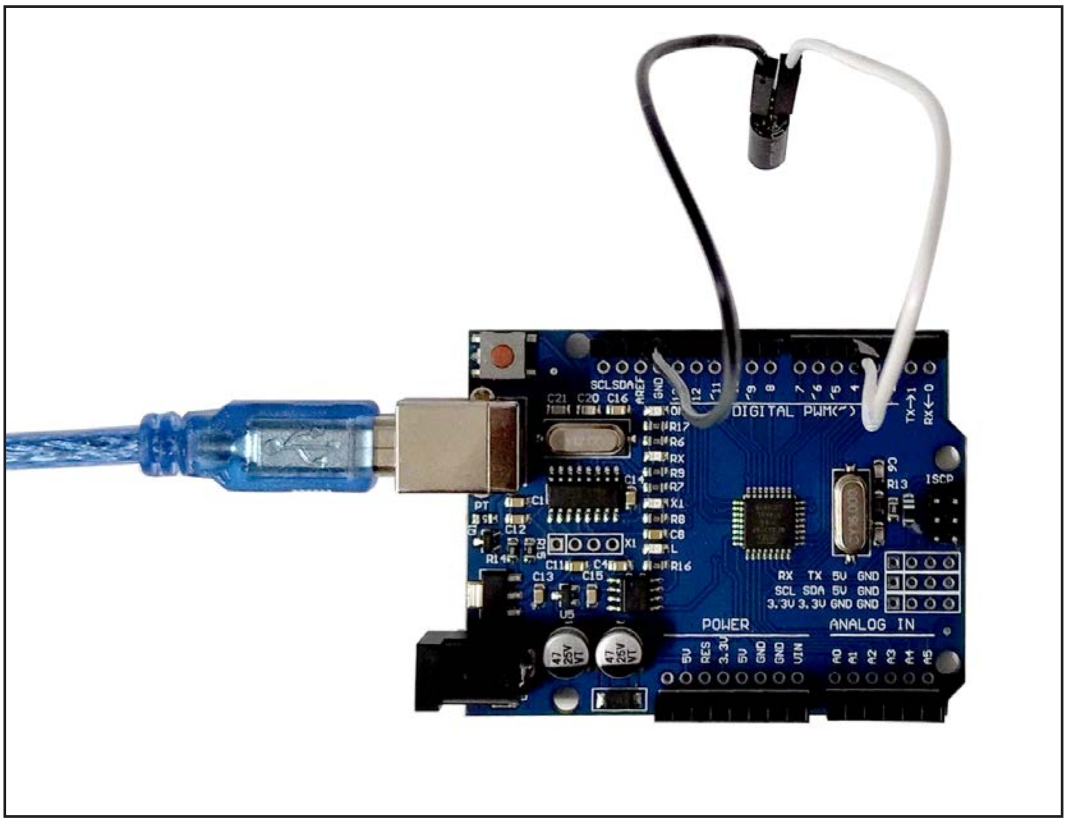
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



Code

```
const int ledPin = 13;        // The LED is connected to pin 13

void setup() {
  pinMode(ledPin, OUTPUT);    // Set pin 13 as an output for the LED
  pinMode(4, INPUT);          // Set pin 4 as an input to read tilt ball switch state
  digitalWrite(4, HIGH);      // Enable internal pull-up resistor on pin 4 set to HIGH
}

/*******************************************/
void loop() {
  int digitalVal = digitalRead(4); // Read the state of pin 4 (tilt ball switch)

  // If the tilt ball switch is not tilted (LOW state), turn the LED off
  if (LOW == digitalVal) {
    digitalWrite(ledPin, LOW); // Turn the LED off
  }
  // If the tilt ball switch is tilted (HIGH state), turn the LED on
  else {
    digitalWrite(ledPin, HIGH); // Turn the LED on
  }
}
```

New Function “if...else”

The "if...else" statement allows the execution of different code blocks based on a condition.
It checks if a specified condition is true, and if it is, the code within the "if" block is executed.
If the condition is false, the code within the "else" block is executed instead.

In the provided code, the "if...else" statement is used to check the state of the tilt ball switch.
If the switch is not tilted (LOW state), the LED is turned off by executing the code inside the "if" block.
If the switch is tilted (HIGH state), the LED is turned on by executing the code inside the "else" block.
This way, the LED's behavior is controlled by the tilt ball switch's orientation.

Lesson 13

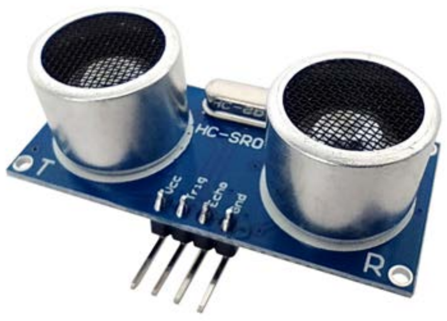
HC-SR04 Ultrasonic Distance Sensor

Overview

The Ultrasonic sensor, specifically the HC-SR04 model, is a versatile component widely used in projects that require distance measurements or obstacle avoidance.
It operates by emitting ultrasonic waves and then measuring the time it takes for the waves to bounce back after hitting an object.
It can be easily integrated into robotics projects, smart devices, and many other applications.

Componets

Uno R3 Development Board X 1	ARD2-0066
Ultrasonic Sensor Module X 1	ARD2-2020
F-M wires (Female to Male jumper wires) X 4	CN3602



Ultrasonic Sensor Module

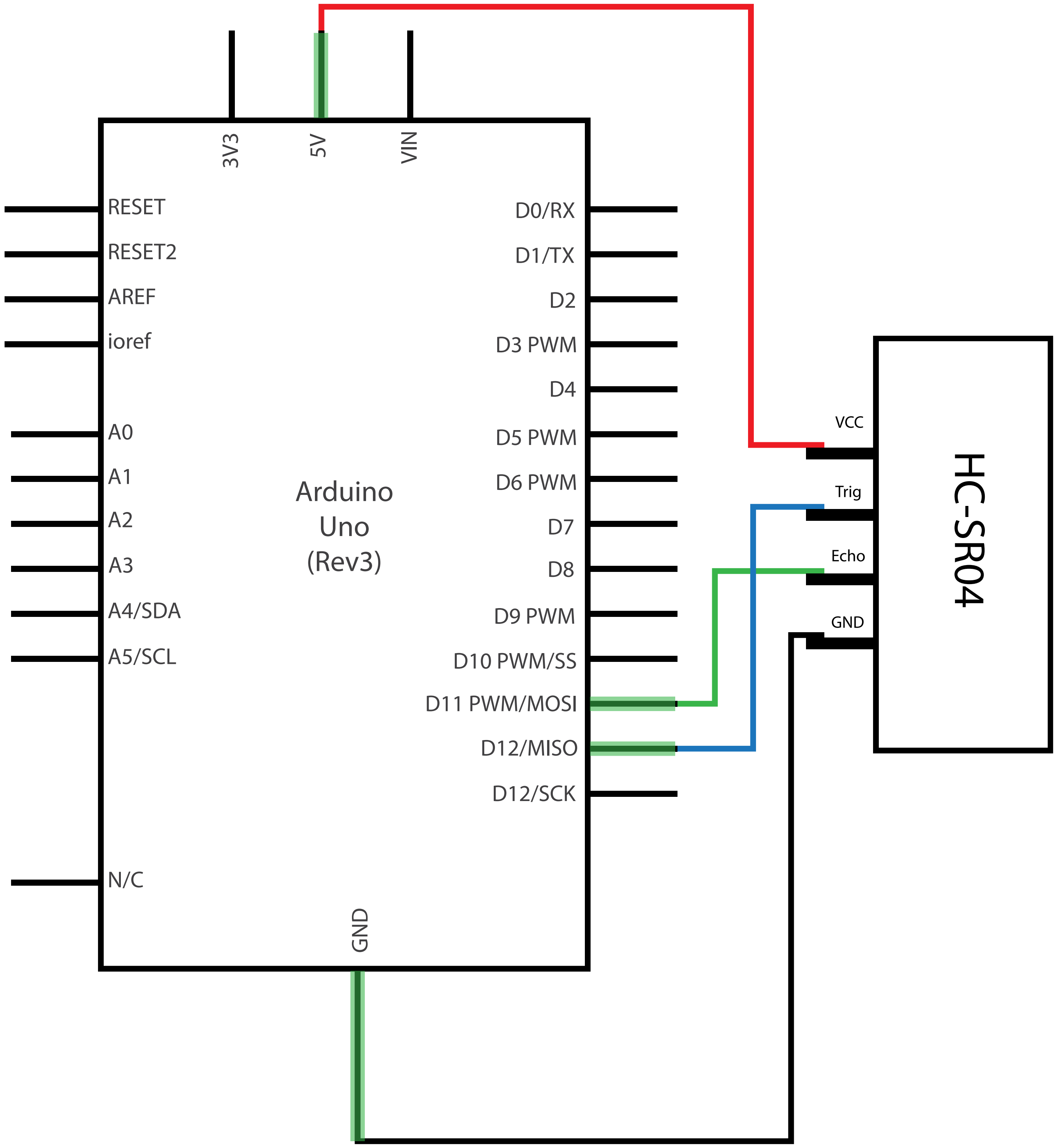
About HC-SR04

The sensor module consists of ultrasonic transmitters, a receiver, and a control circuit.
It operates based on the principle of emitting a 40 kHz ultrasonic pulse and detecting the echo signal that bounces back from an object.

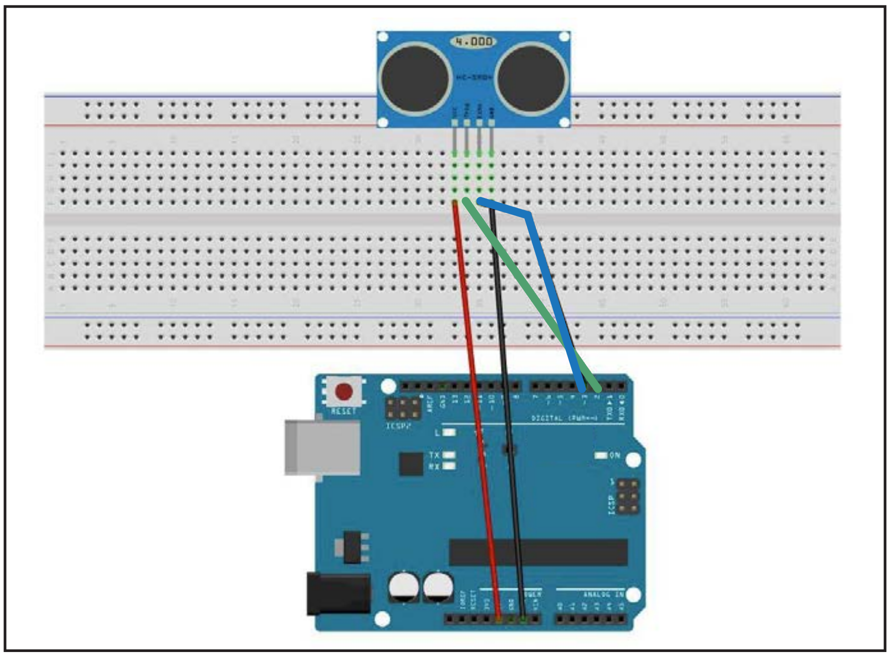
To measure distance, a 10us high-level signal is applied to the trigger input, initiating the ranging process. The module then emits an 8-cycle burst of ultrasound at 40 kHz and waits for the echo. The distance is calculated by measuring the time interval between sending the trigger signal and receiving the echo signal.

The formula to calculate the distance in centimeters is:
distance = (high level time * velocity of sound (340m/s)) / 2
For converting the time interval into inches, the formula is:
distance = high level time * velocity of sound (340m/s) / 148.

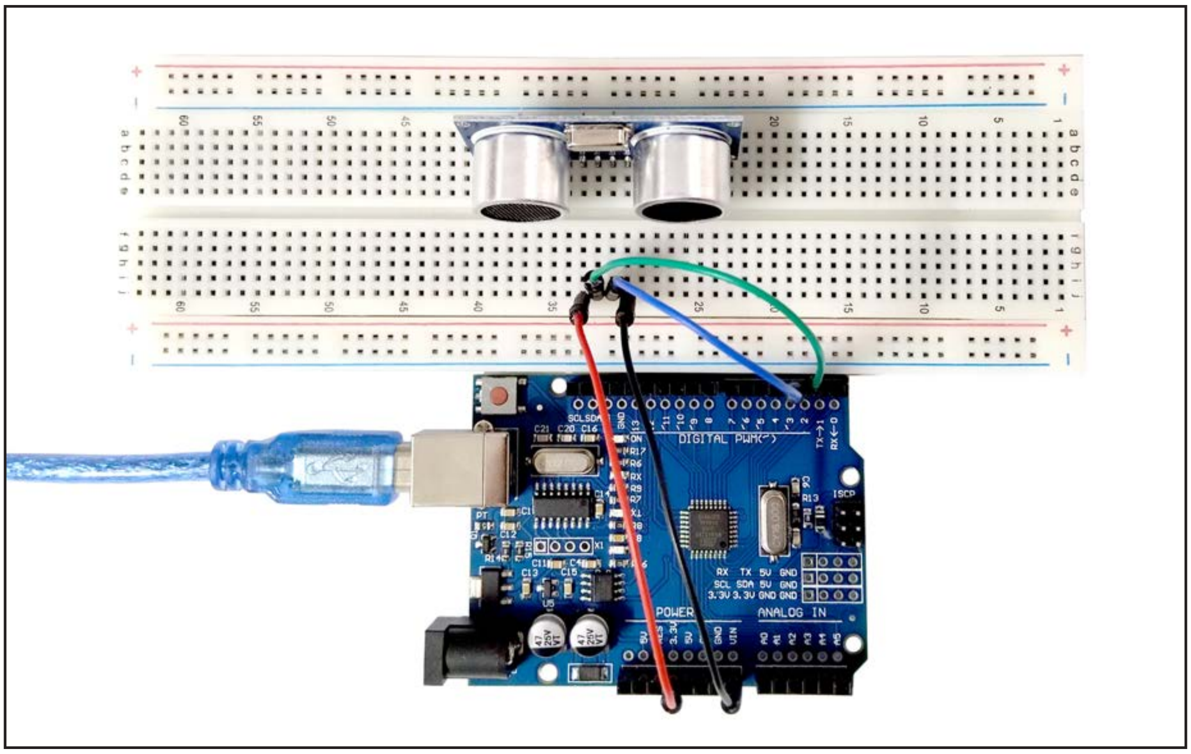
Connection Diagram



Wiring Schematic



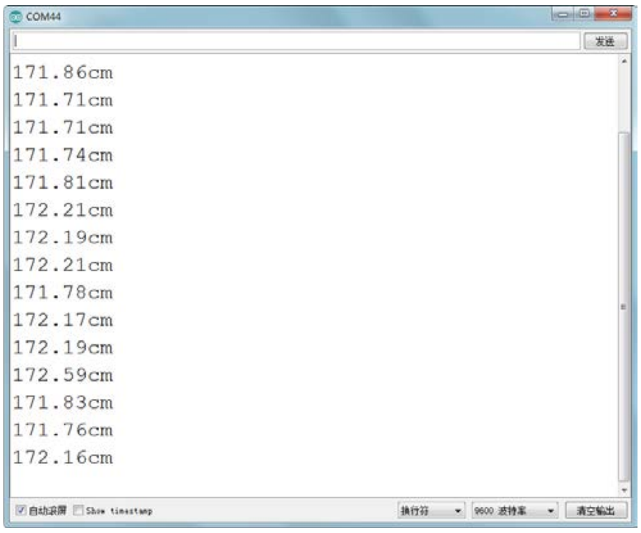
Physical Wiring Diagram



Code

```
const int TrigPin = 2;  // TrigPin connected to pin 2 for triggering the ultrasonic sensor
const int EchoPin = 3;  // EchoPin connected to pin 3 for receiving the echo signal
float distance;         // Variable to store the measured distance in centimeters

void setup() {
  Serial.begin(9600); // Initialize serial communication at 9600 baud rate
  pinMode(TrigPin, OUTPUT); // Set TrigPin as an output for sending trigger signal
  pinMode(EchoPin, INPUT); // Set EchoPin as an input to receive the echo signal
  Serial.println("Ultrasonic sensor:"); // Print a message to the serial monitor
}
void loop() {
  digitalWrite(TrigPin, LOW);     // Set TrigPin to LOW state to ensure a clean start
  delayMicroseconds(2);           // Short delay for stabilization
  digitalWrite(TrigPin, HIGH);    // Send a 10us high-level pulse to the ultrasonic sensor
  delayMicroseconds(10);          // Keep the TrigPin HIGH for 10us
  digitalWrite(TrigPin, LOW);     // Turn off the trigger pulse

distance = pulseIn(EchoPin, HIGH) / 58.00; // Measure the time taken for the echo
                             //signal to return and calculate the distance in centimeters
Serial.print(distance);      // Print the measured distance to the serial monitor
Serial.print("cm");          // Print the unit (centimeters)
Serial.println();            // Move to the next line in the serial monitor
delay(1000);                 // Wait for 1 second before taking the next measurement
}
```

Open the Serial Monitor to check the Readings



```
171.86cm
171.71cm
171.71cm
171.74cm
171.81cm
172.21cm
172.19cm
172.21cm
171.78cm
172.17cm
172.19cm
172.59cm
171.83cm
171.76cm
172.16cm
```

Summary

The code sets up the Arduino by defining the necessary pins for the ultrasonic sensor (TrigPin and EchoPin), initializes serial communication, and prints a message to the serial monitor. In the loop, the code triggers the ultrasonic sensor to send a pulse, measures the time it takes for the echo signal to return, calculates the distance, and displays the measured distance in centimeters on the serial monitor. The process is repeated every 1 second to continuously measure and display the distance.

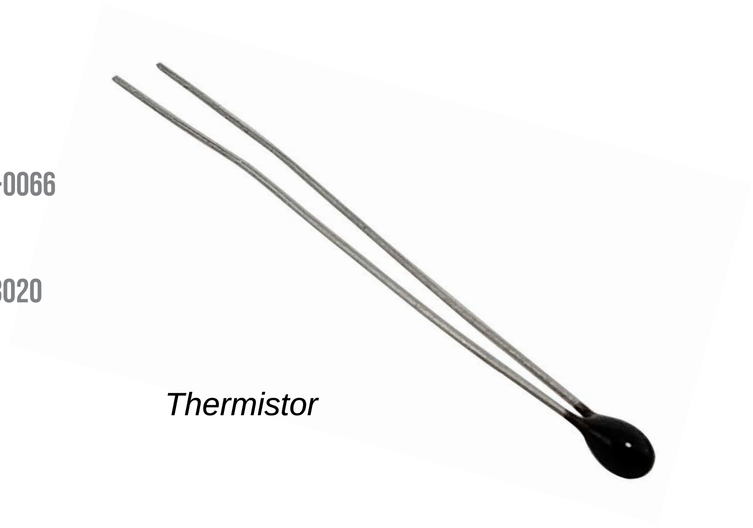
Lesson 14
Thermometer Sensor

Overview

In this lesson you will use NTC thermistors to measure temperature and display the readings on a serial monitor or a connected display. This allows you to create temperature monitoring projects and applications using the thermistor's resistance-temperature relationship.

Componets

UNO R3 Compatible Development Board	ARD2-0066
830 tie-points Breadboard X 1	MA4009
M-M wires (Male to Male jumper wires) X 3	MA3020
220 ohm Resistor x 1	RS1405
Thermistor x 1	RS6472



About Thermistors

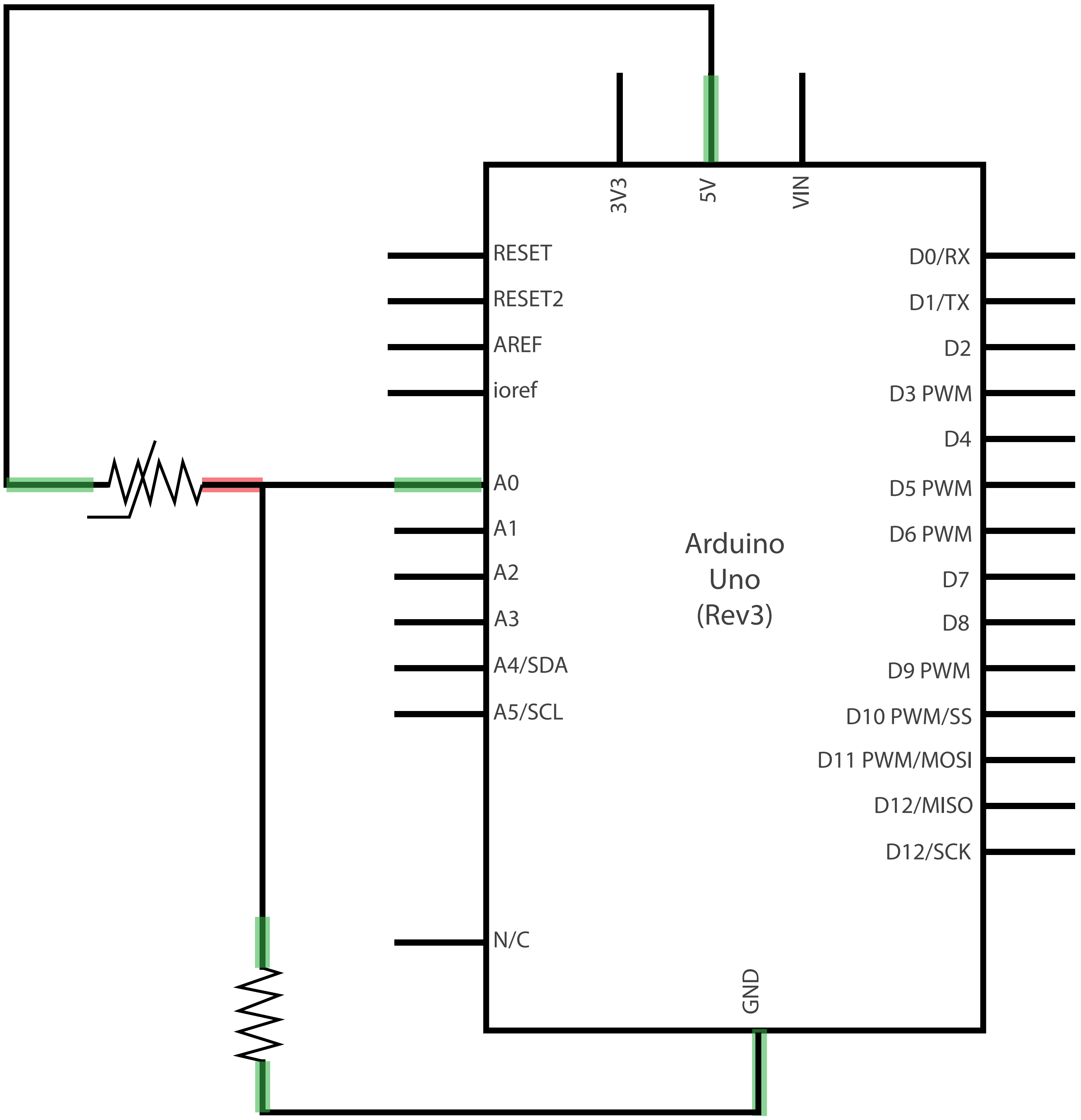
A thermistor is a type of thermal resistor that exhibits a significant change in resistance with temperature. While all resistors experience some change in resistance with temperature, thermistors are designed to have a much larger and more noticeable change, often around 100 ohms or more per degree of temperature change.

There are two main types of thermistors:
NTC (negative temperature coefficient) and PTC (positive temperature coefficient).

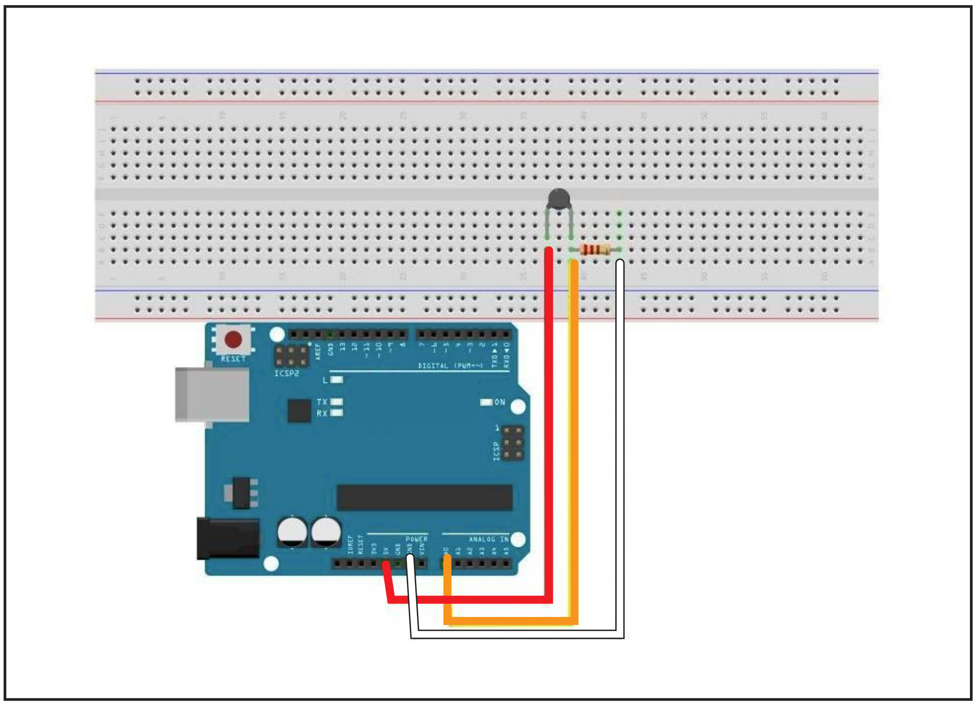
NTC thermistors have a decrease in resistance as temperature increases, and they are commonly used for temperature measurement applications.

On the other hand, PTC thermistors experience an increase in resistance with temperature and are often used as resettable fuses to protect circuits by limiting current flow as they heat up.

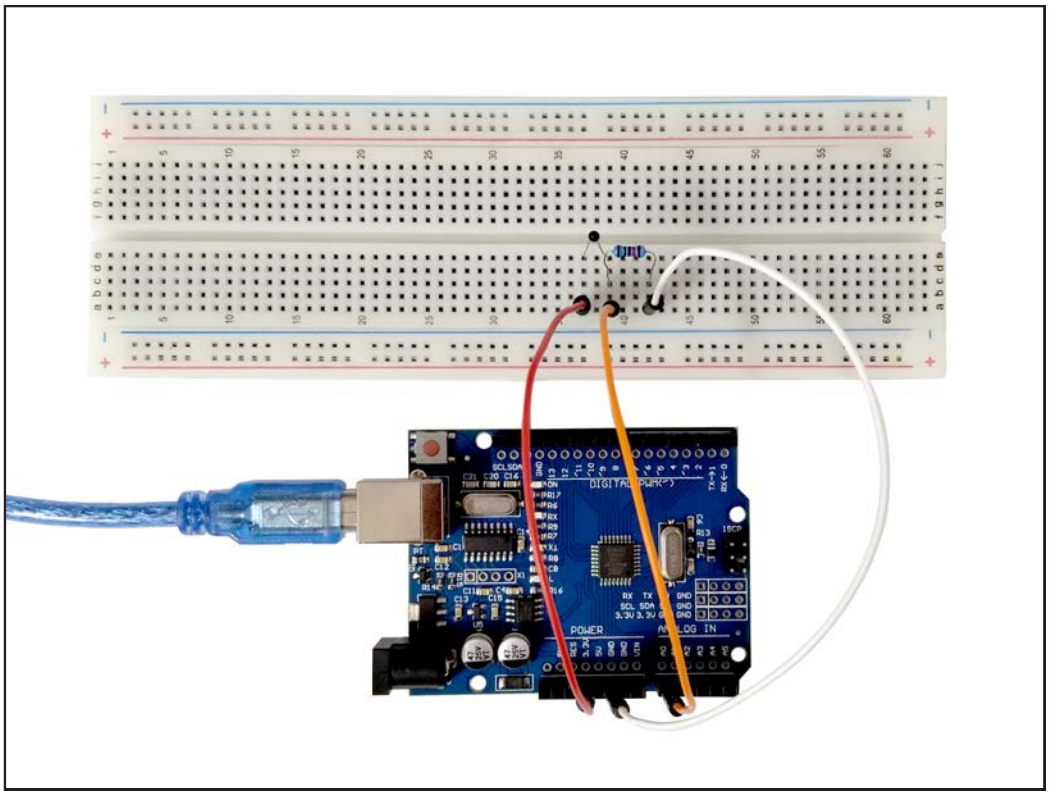
Connection Diagram



Wiring Schematic



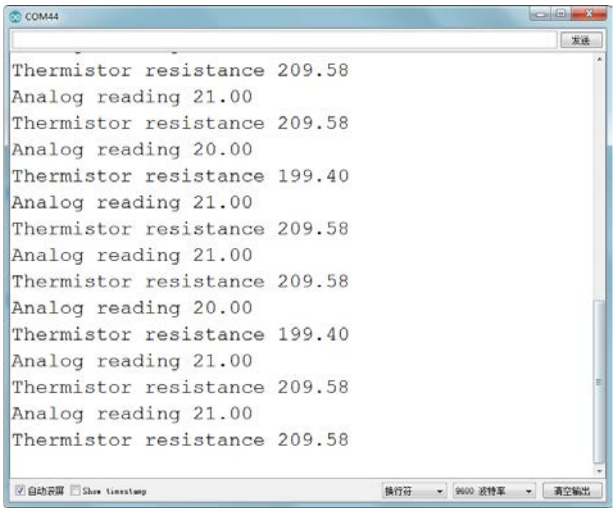
Physical Wiring Diagram



Code

```
// The value of the 'other' resistor
#define SERIESRESISTOR 10000

// What pin to connect the thermistor to
#define THERMISTORPIN A0

void setup(void) {
Serial.begin(9600); // Start serial communication at 9600 baud rate
}

void loop(void) {
float reading; // Variable to store the analog reading from the thermistor
reading = analogRead(THERMISTORPIN); // Read the analog value from the thermistor

Serial.print("Analog reading: "); // Print a label to identify the analog reading value
Serial.println(reading); // Print the analog reading value on the serial monitor

// Convert the analog reading to thermistor resistance using the provided formula
reading = (1023 / reading) - 1; // (1023/ADC - 1)
reading = SERIESRESISTOR / reading; // 10K / (1023/ADC - 1)

Serial.print("Thermistor resistance: ");    // Print a label to identify the
                                             // thermistor resistance value

Serial.println(reading); // Print the thermistor resistance value on the serial monitor

delay(1000); // Add a 1-second delay between readings to avoid excessive data output
}
```

Open the Serial Monitor to check the Readings



```
Thermistor resistance 209.58
Analog reading 21.00
Thermistor resistance 209.58
Analog reading 20.00
Thermistor resistance 199.40
Analog reading 21.00
Thermistor resistance 209.58
Analog reading 21.00
Thermistor resistance 209.58
Analog reading 20.00
Thermistor resistance 199.40
Analog reading 21.00
Thermistor resistance 209.58
Analog reading 21.00
Thermistor resistance 209.58
```

Summary

The code reads the analog value from a thermistor connected to pin A0. It calculates the thermistor resistance using the provided formula and prints the analog reading and resistance values on the serial monitor.

The code enables monitoring of the thermistor's output for temperature sensing applications.

Lesson 15
Analog Joystick Module

Overview

In this lesson we will learn how to interface and use the analog joystick module.
This will enable us to add user-controlled input to our projects, opening up possibilities for creating games, robotic control systems, and various interactive applications.

Componets

UNO R3 Compatible Development Board	ARD2-0066
Joystick Module X 1	ARD2-2223
F-M wires (Female to Male jumper wires) X 5	CN3602

About Analog Joysticks

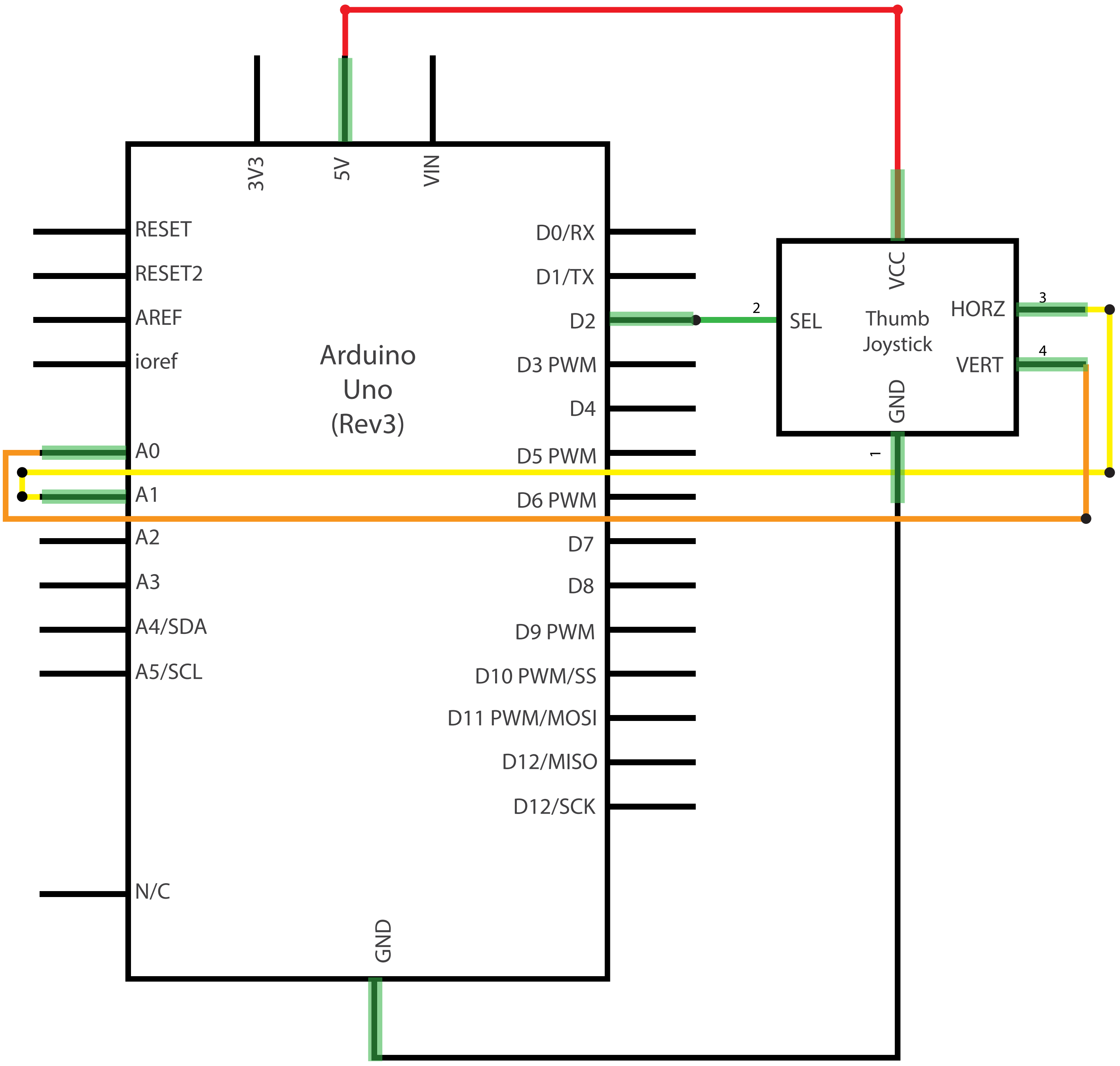
The analog joystick module has five pins: VCC (power supply), Ground (common ground), X and Y (analog output pins representing the joystick's position along the X and Y axes), and Key (digital input pin for the joystick's press-to-select push-button).

The X and Y pins provide analog voltage signals corresponding to the joystick's position in the X and Y directions. We need analog pins on the Arduino to read theanalog signals accurately.

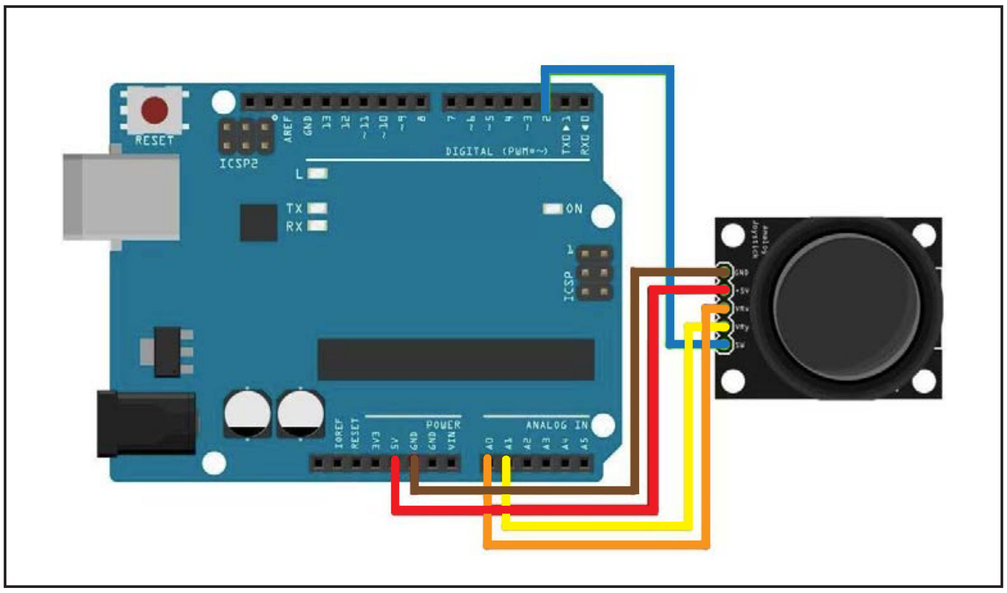
The Key pin is connected to ground when the joystick is pressed down and is floating (not connected to anything) otherwise. To obtain stable readings from the Key/Select pin, we need to connect it to VCC (power supply) via a pull-up resistor.
The built-in resistors on the Arduino's digital pins can be used to activate the pull-up resistors for the Key pin configured as an input.

In summary, to use the analog joystick module, we connect VCC and Ground to their respective power and ground sources. We connect the X and Y pins to analog input pins on the Arduino to read the joystick's position, and the Key pin to a digital input pin on the Arduino to detect the press of the joystick's push-button.

Connection Diagram

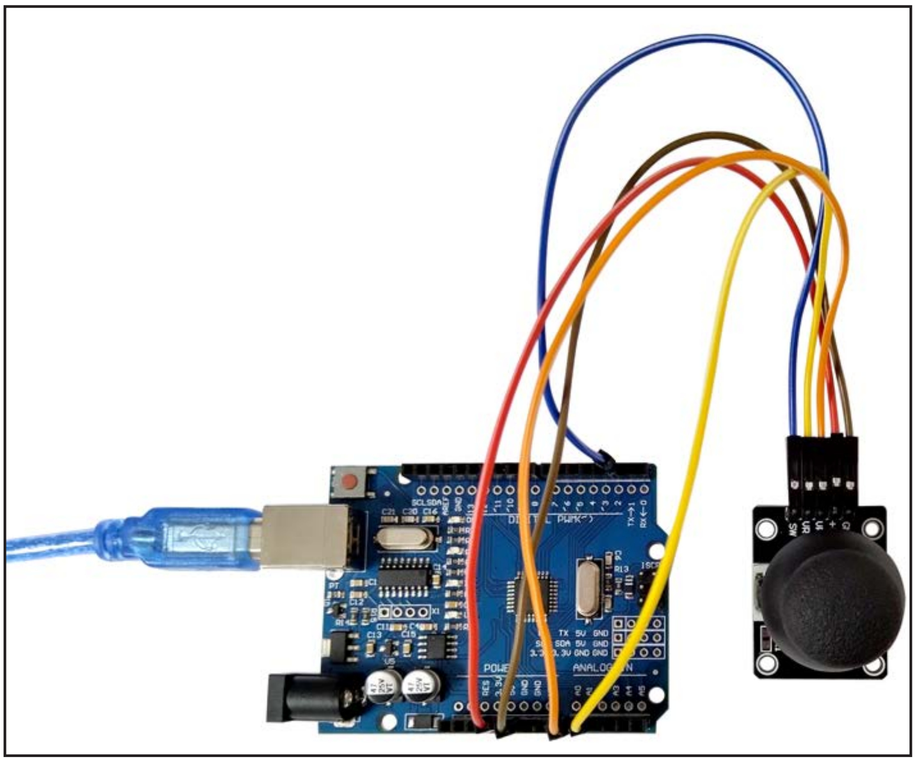


Wiring Schematic



We need 5 connections to the joystick.
The connections are: Key, Y, X, Voltage and Ground.
"Y and X" are Analog and "Key" is Digital. If you don't need the switch then you can use only 4 pins.

Physical Wiring Diagram



Code

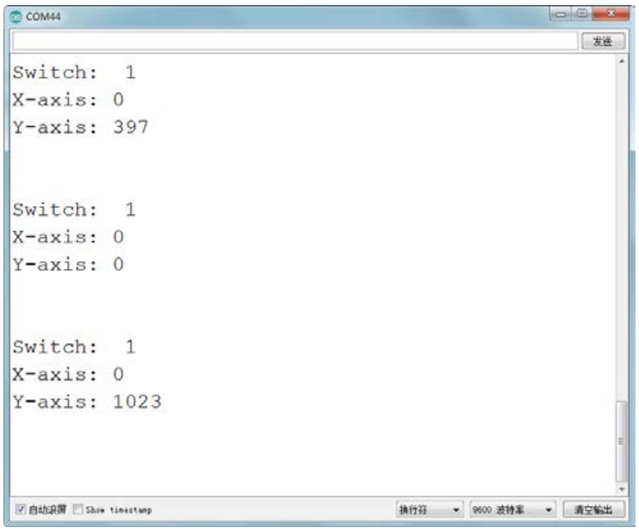
```
/* Arduino pin numbers */
const int SW_pin = 2;     // Pin number for the switch
const int Y_pin = 1;      // Pin number for the Y-axis analog sensor
const int X_pin = 0;      // Pin number for the X-axis analog sensor

bool switchState = HIGH;        // Variable to store the current state of the switch
bool lastSwitchState = HIGH;    // Variable to store the previous state of the switch

void setup() {
pinMode(SW_pin, INPUT);         // Set the switch pin as INPUT
digitalWrite(SW_pin, HIGH);     // Enable the internal pull-up resistor for the switch pin
Serial.begin(9600);             // Initialize the serial communication at 9600 baud rate
// put your setup code here, to run once:
}

void loop() {
  // Read the state of the switch
  switchState = digitalRead(SW_pin);

  // Check if the switch state has changed
  if (switchState != lastSwitchState) {
    if (switchState == LOW) {
      Serial.println("Switch pressed!"); // Print a message when the switch is pressed
    } else {
      Serial.println("Switch released!"); // Print a message when the switch is released
    }
    delay(50); // Add a small debounce delay to avoid false triggering
  }
// Update the last switch state
lastSwitchState = switchState;
```

Code Continued on Next page

Code (Cont)

```
// Read and print the analog values
Serial.print("Switch: ");
Serial.print(switchState);        // Print the state of the switch (LOW or HIGH)
Serial.print("\n");
Serial.print("Y-axis: ");
Serial.print(analogRead(Y_pin));  // Read and print the analog value from the Y-axis sensor
Serial.print("\n");
Serial.print("X-axis: ");
Serial.print(analogRead(X_pin));  // Read and print the analog value from the X-axis sensor
Serial.print("\n\n");

delay(100); // Add a small delay for smoother operation
// put your main code here, to run repeatedly:
}
```

Open the Serial Monitor to check the Readings



```
Switch:  1
X-axis: 0
Y-axis: 397


Switch:  1
X-axis: 0
Y-axis: 0


Switch:  1
X-axis: 0
Y-axis: 1023
```

Summary

The code allows users to control and monitor the joystick's input.
It reads the state of a push-button switch and the analog values from the Y-axis and X-axis sensors of the joystick module. It then prints the state of the switch and the analog readings from the Y-axis and X-axis sensors on the serial monitor.
The code also includes a small debounce delay to prevent false triggering when the switch state changes. The main loop continuously reads and prints the sensor values with a small delay for smoother operation.

Lesson 16
L293D Chip & DC Motors

Overview

In this lesson, you will learn how to control a small DC motor using an UNO R3 Compatible Development Board, an L293D Chip, and a transistor.
By the end of the lesson, you'll be equipped with the knowledge to control DC motors effectively and use them in your own projects and applications.

Componets

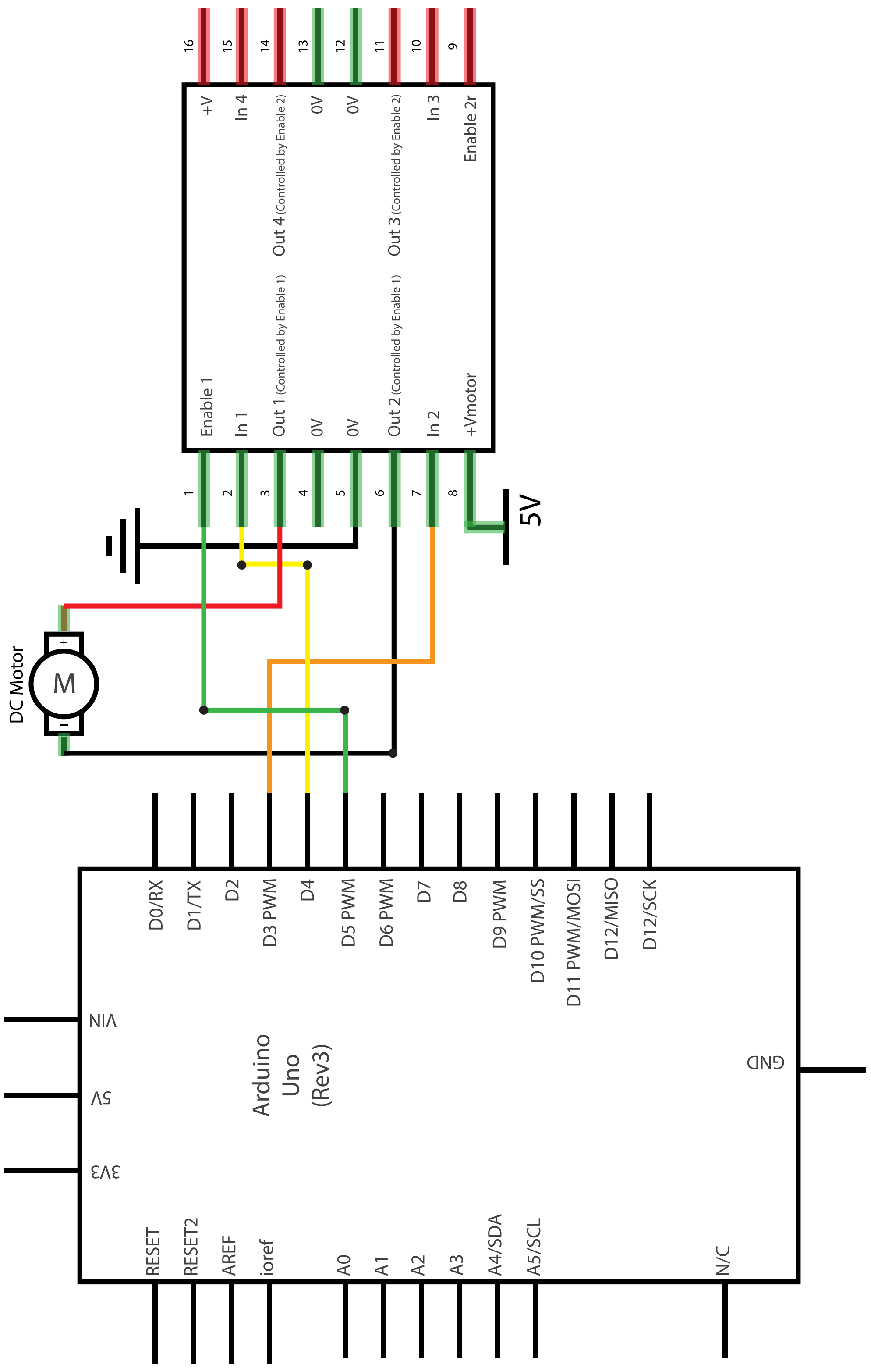
UNO R3 Compatible Development Board	ARD2-0066
830 tie-points Breadboard X 1	MA4009
M-M wires (Male to Male jumper wires) X 5	MA3020
DC Motor x 1	M00000
L293D IC x 1	L293D

About the L293D IC

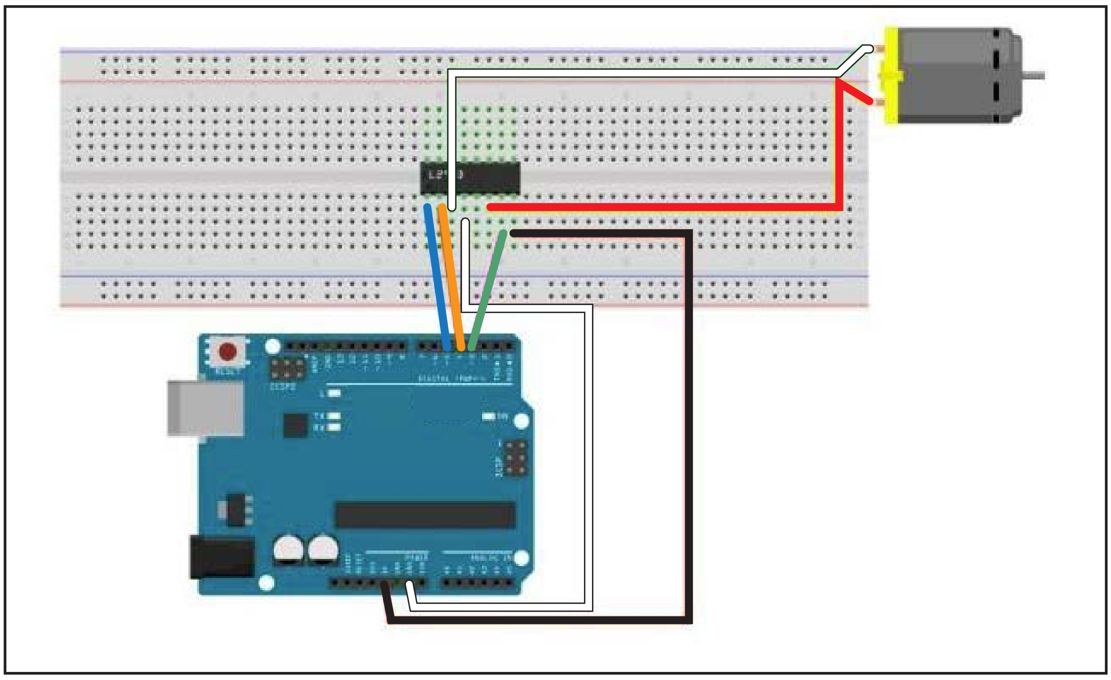
The L293D is a quadruple high-current half-H driver chip designed to provide bidirectional drive currents of up to 600mA at voltages from 4.5V to 36V.
It can control two motors independently and is ideal for driving inductive loads such as relays, solenoids, DC and bipolar stepping motors, and other high-current and high-voltage loads in positive-supply applications.

Its versatility and reliability make it a valuable component for projects requiring precise motor control. The L293D features TTL compatible inputs and provides a totem-pole drive circuit for each output, using Darlington transistor sinks and pseudo-Darlington sources. The drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN.
When an enable input is high, the associated drivers are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are in the high-impedance state. Each pair of drivers can function as a full-H or bridge reversible drive, making it suitable for controlling solenoids or motors.

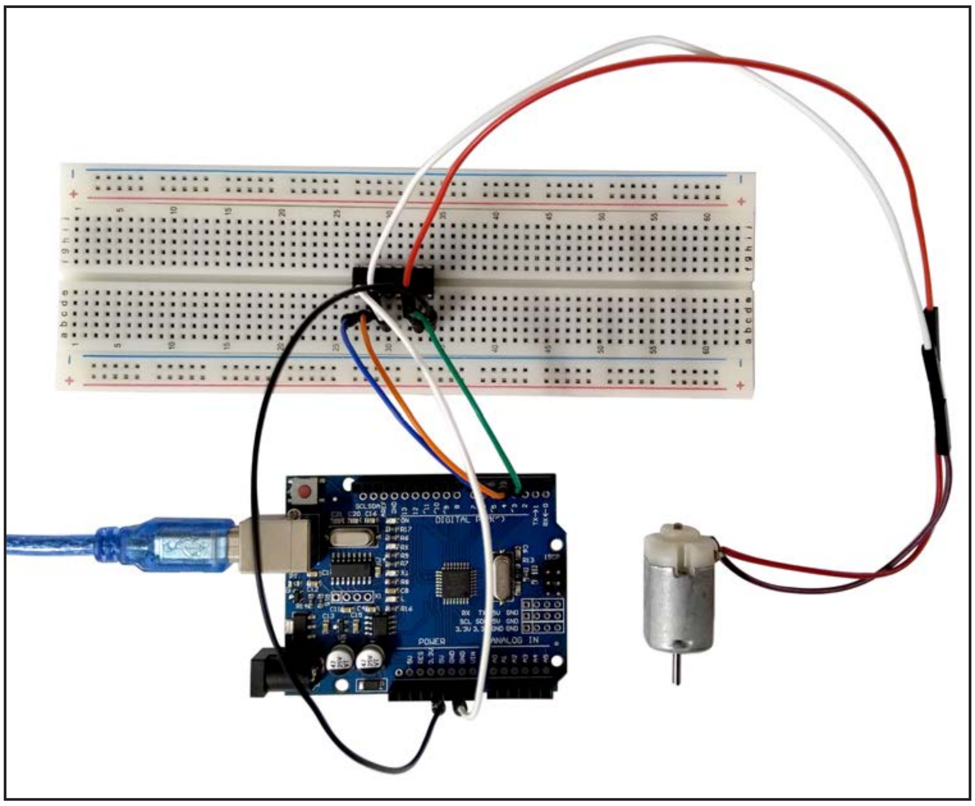
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



This code demonstrates various motor control techniques using the L293D motor driver. It enables the motor driver, controls the motor in different directions and speeds, and then disables the motor driver.

Code

```
// Pin definitions
#define ENABLE 5 // Pin to enable motor driver
#define DIRA 3 // Pin for motor direction control A
#define DIRB 4 // Pin for motor direction control B

int i; // Loop counter variable

void setup() {
  // Set pin directions
  pinMode(ENABLE, OUTPUT);
  pinMode(DIRA, OUTPUT);
  pinMode(DIRB, OUTPUT);

  Serial.begin(9600); // Start serial communication for debugging
  // put your setup code here, to run once:
}

void loop() {
  //---back and forth example---
  Serial.println("One way, then reverse");
  digitalWrite(ENABLE, HIGH); // Enable the motor driver
  for (i = 0; i < 5; i++) {
    digitalWrite(DIRA, HIGH); // Set motor direction one way
    digitalWrite(DIRB, LOW);
    delay(500);
    digitalWrite(DIRA, LOW); // Set motor direction in reverse
    digitalWrite(DIRB, HIGH);
    delay(500);
  }
  digitalWrite(ENABLE, LOW); // Disable the motor driver
  delay(2000);
```

Code Continued on Next page

Code (cont)

```
//---fast Slow example---
Serial.println("fast Slow example");
digitalWrite(ENABLE, HIGH);
digitalWrite(DIRA, HIGH);     // Set motor direction one way
digitalWrite(DIRB, LOW);
delay(3000);
digitalWrite(ENABLE, LOW);    // Disable the motor driver
delay(1000);
digitalWrite(ENABLE, HIGH);
digitalWrite(DIRA, LOW);      // Set motor direction in reverse
digitalWrite(DIRB, HIGH);
delay(3000);
digitalWrite(DIRA, LOW);      // Stop the motor
delay(2000);

//---PWM full then slow---
Serial.println("PWM full then slow");
analogWrite(ENABLE, 255);     // Set motor speed to full
digitalWrite(DIRA, HIGH);     // Set motor direction one way
digitalWrite(DIRB, LOW);
delay(2000);
analogWrite(ENABLE, 180);     // Reduce motor speed
delay(2000);
analogWrite(ENABLE, 128);     // Reduce motor speed further
delay(2000);
analogWrite(ENABLE, 50);      // Set motor speed to slow
delay(2000);
analogWrite(ENABLE, 128);     // Increase motor speed
delay(2000);
analogWrite(ENABLE, 180);     // Increase motor speed further
delay(2000);
analogWrite(ENABLE, 255);     // Set motor speed to full
delay(2000);
digitalWrite(ENABLE, LOW);    // Disable the motor driver
delay(100000);                // Delay for a long time before repeating

// put your main code here, to run repeatedly:
};
```

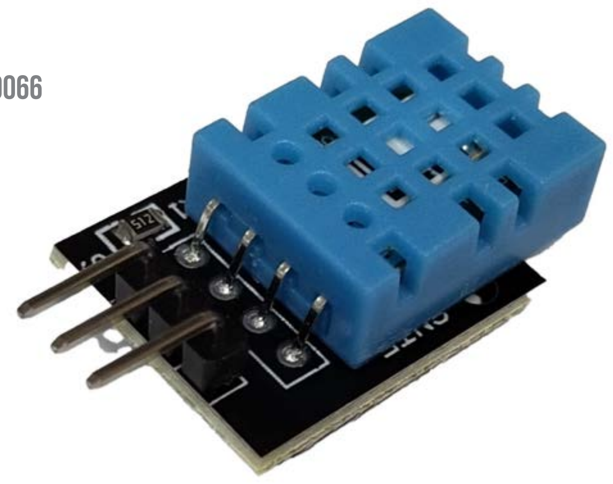
Lesson 17
DHT11 Temperature and Humidity Sensor

Overview

In this lesson, we will learn how to use a DHT11 Temperature and Humidity Sensor.
It's accurate enough for most projects that need to keep track of humidity and temperature readings.
We will use a library specifically designed for these sensors that will make our code short and easy to write.

Componets

UNO R3 Compatible Development Board ARD2-0066
1 x DHT11 Temperature and Humidity Module
3 x Female to Male DuPont Wires

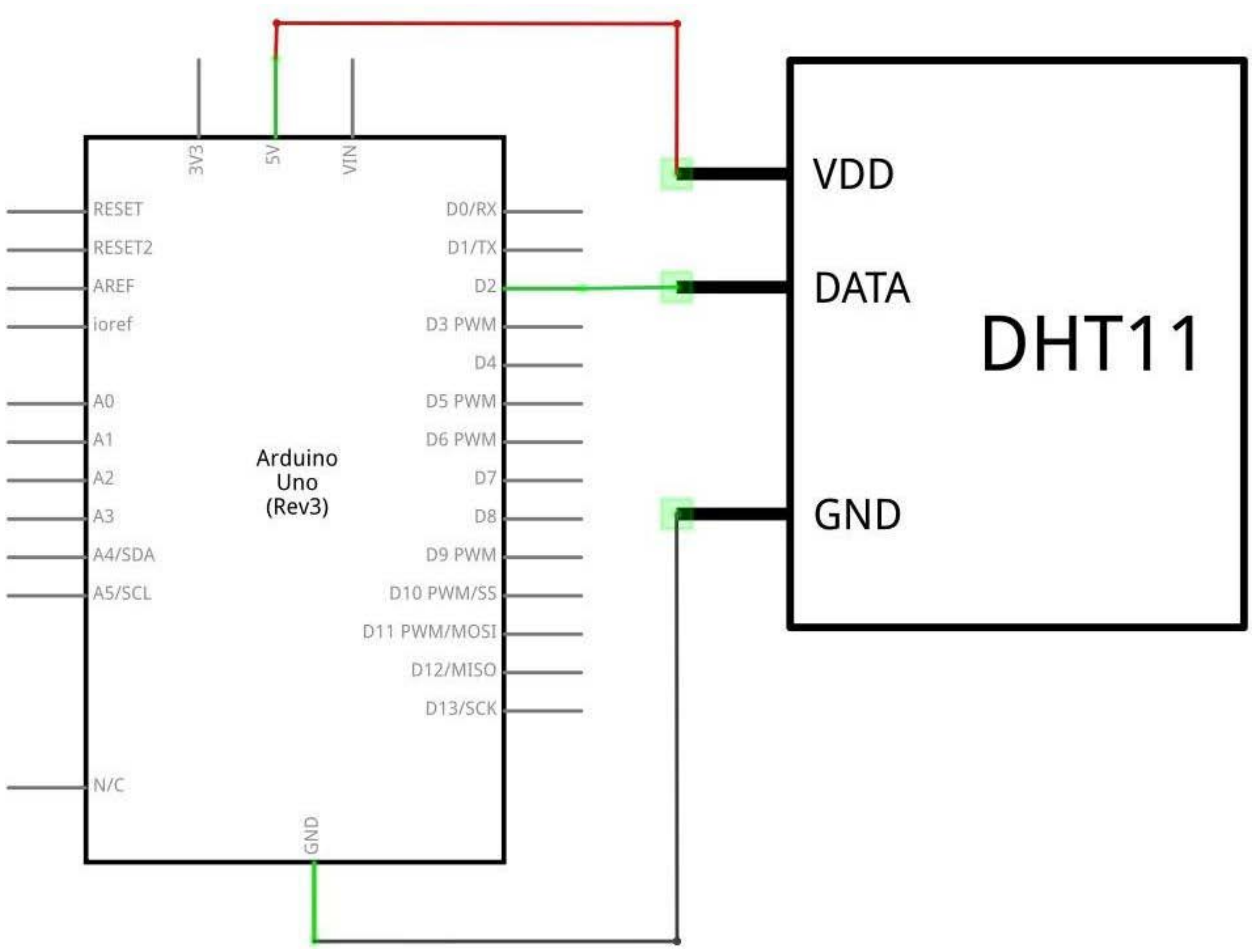


Relationship between Components

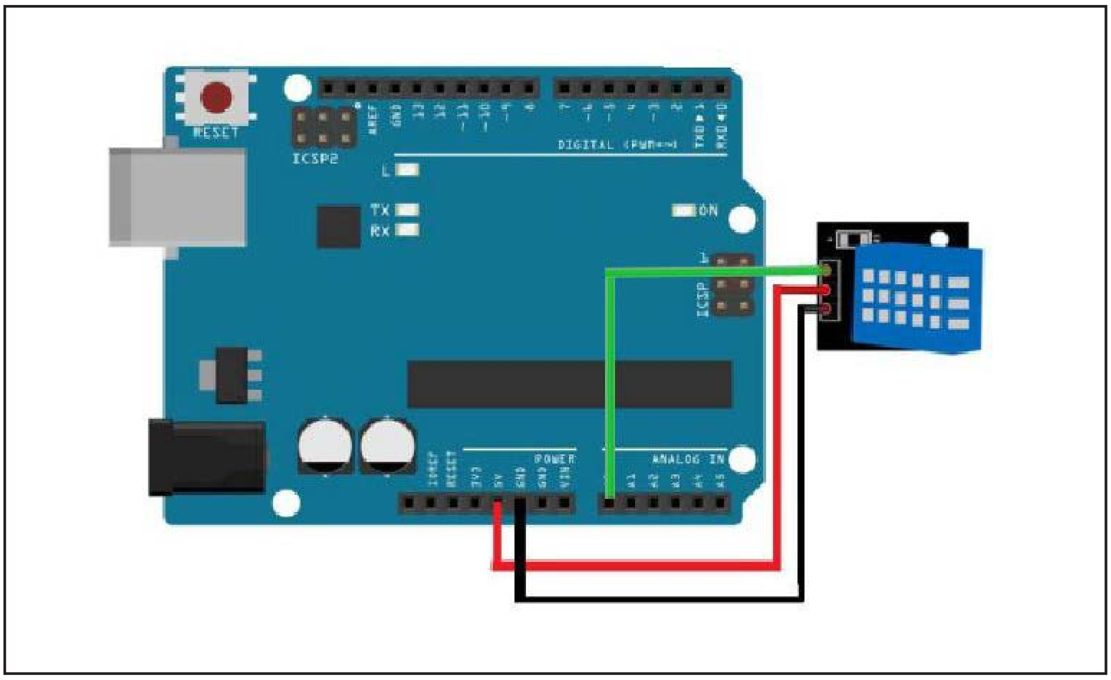
The DHT11 sensor is used to measure temperature and humidity levels.
It provides digital output, making it easy to interface with microcontrollers like the UNO R3.
The sensor has four pins: VDD (power supply), DATA (serial data), NC (not connected), and GND (ground).

Proper wiring and use of the library functions allow the UNO R3 to read temperature and humidity values accurately.

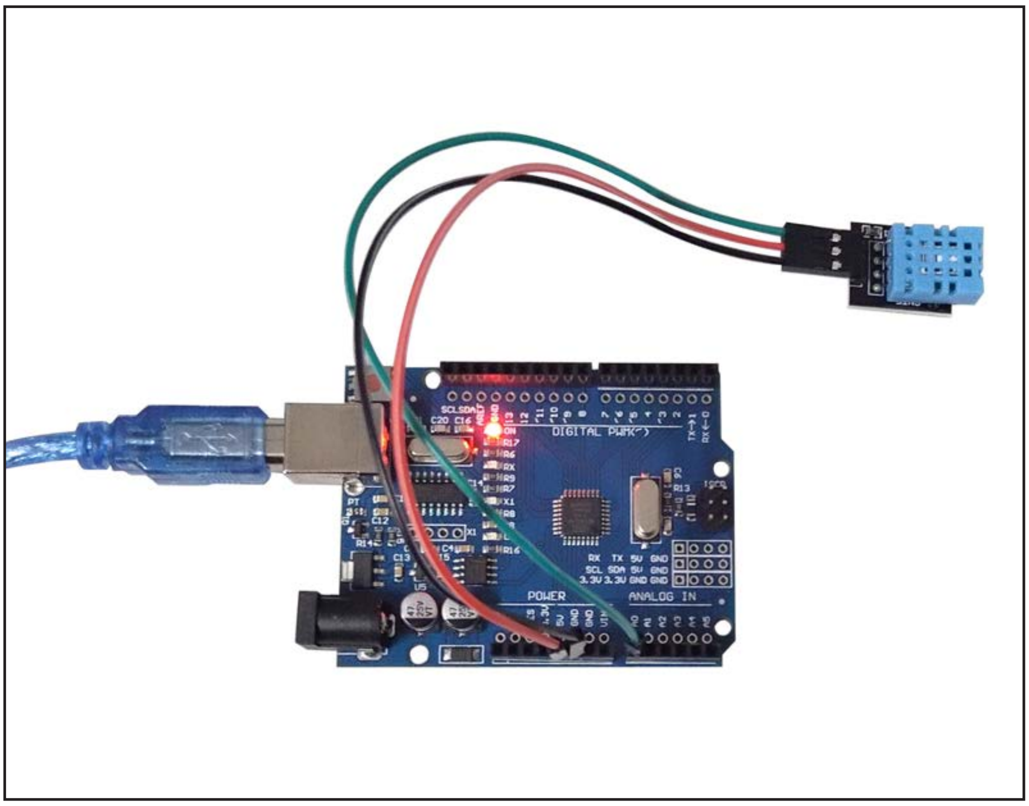
Connection Diagram



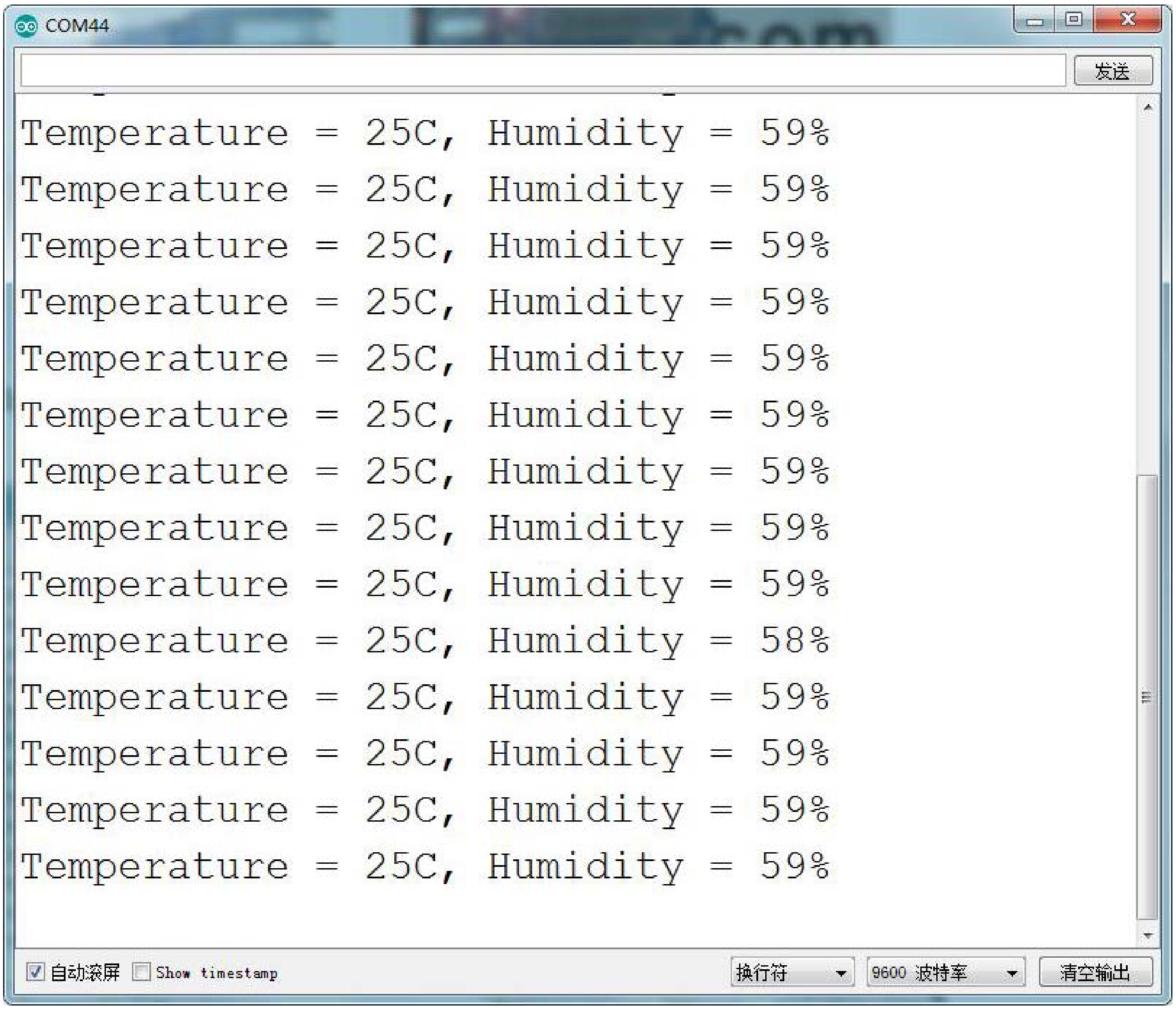
Wiring Schematic



Physical Wiring Diagram



Open the serial monitor as follows:



```
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 58%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
```

Code

```
#include "dht11.h"

DHT sensor;

void setup() {
    // Initialize the serial communication
    Serial.begin(9600);
    // Attach the sensor to pin A0
    sensor.attach(A0);
    // Allow some time for the sensor to stabilize
    delay(1000);
}

void loop() {
    // Update the sensor readings
    sensor.update();
    // Check for errors in reading the sensor data
    switch (sensor.getLastError()) {
        case DHT_ERROR_OK:
            // Create a message with the temperature and humidity values
            char msg[128];
            sprintf(msg, "Temperature = %dC, Humidity = %d%%", sensor.getTemperatureInt(),
sensor.getHumidityInt());
            // Print the message to the serial monitor
            Serial.println(msg);
            break;
        case DHT_ERROR_START_FAILED_1:
            Serial.println("Error: start failed (stage 1)");
            break;
        case DHT_ERROR_START_FAILED_2:
            Serial.println("Error: start failed (stage 2)");
            break;
        case DHT_ERROR_READ_TIMEOUT:
            Serial.println("Error: read timeout");
            break;
        case DHT_ERROR_CHECKSUM_FAILURE:
            Serial.println("Error: checksum error");
            break;
    }
    // Wait for 2 seconds before the next reading
    delay(2000);
}
```

Summary

In this lesson, we learned how to use a DHT11 Temperature and Humidity Sensor with an UNO R3 development board. The sensor is connected to the board using three wires (VDD, DATA, GND).

We utilized a library to simplify reading the sensor data and printed the temperature and humidity values to the serial monitor. The setup function initializes the sensor, and the loop function continuously updates and displays the sensor readings every 2 seconds.

Lesson 18
Servo Motor

Overview

The servo motor has three wires: power, ground, and signal. The power wire is usually red and should be connected to the 5V pin on the Arduino board. The ground wire is usually black or brown and should be connected to the ground pin on the Arduino board.
The signal wire is usually yellow, orange, or white and should be connected to the digital pins on the Arduino board. Please note that the servo system consumes considerable power, so if you need to drive more than one or two, you may need to use a separate power supply (i.e., the +5V pin on the Arduino) to power them.
Be sure to connect the Arduino to the ground of the external power supply.

Componets

UNO R3 Compatible Development Board
1 x Servo (SG90)
3 x Male to Male Jumper Wires

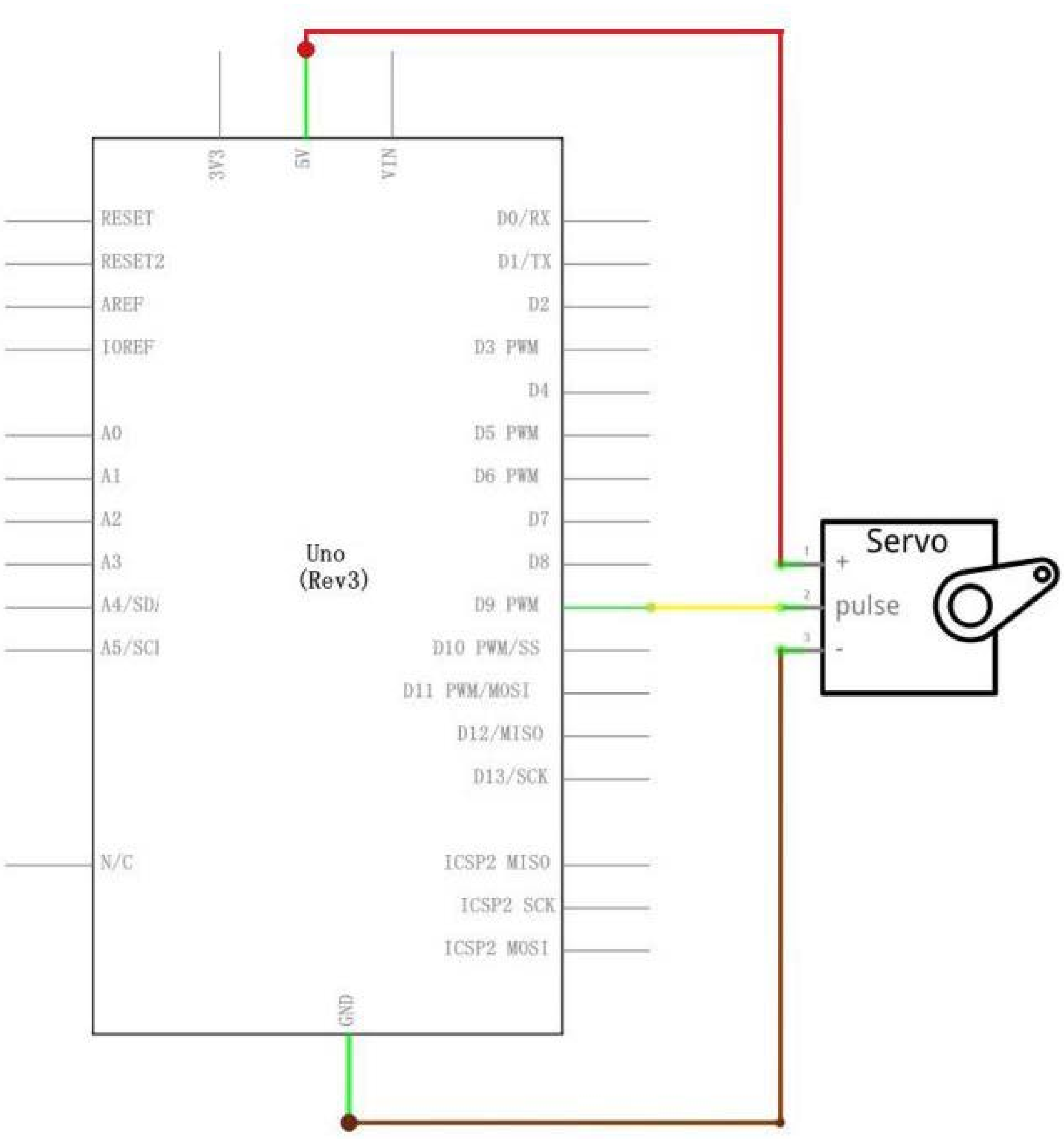
Relationship between Components

The SG90 servo motor is controlled by sending a PWM (Pulse Width Modulation) signal from the Arduino.

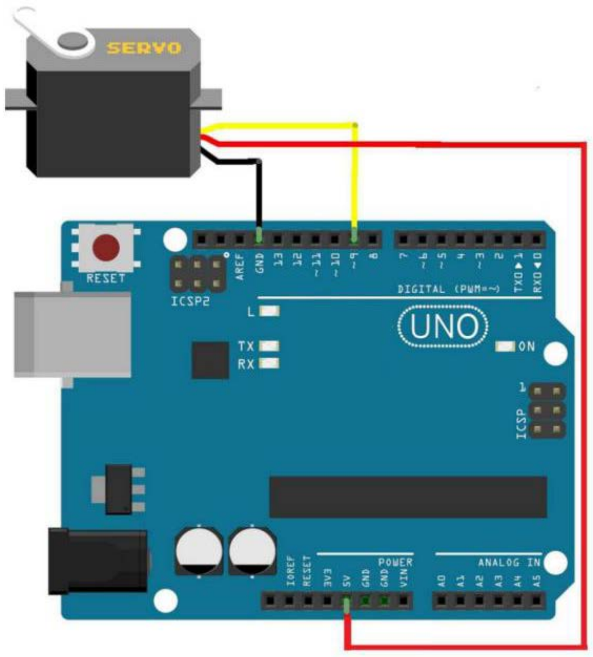
The servo has three wires: the red wire connects to the 5V power pin, the black or brown wire connects to the ground, and the yellow, orange, or white wire connects to a digital PWM pin on the Arduino.

The servo motor rotates to the specified angle based on the PWM signal received from the Arduino.

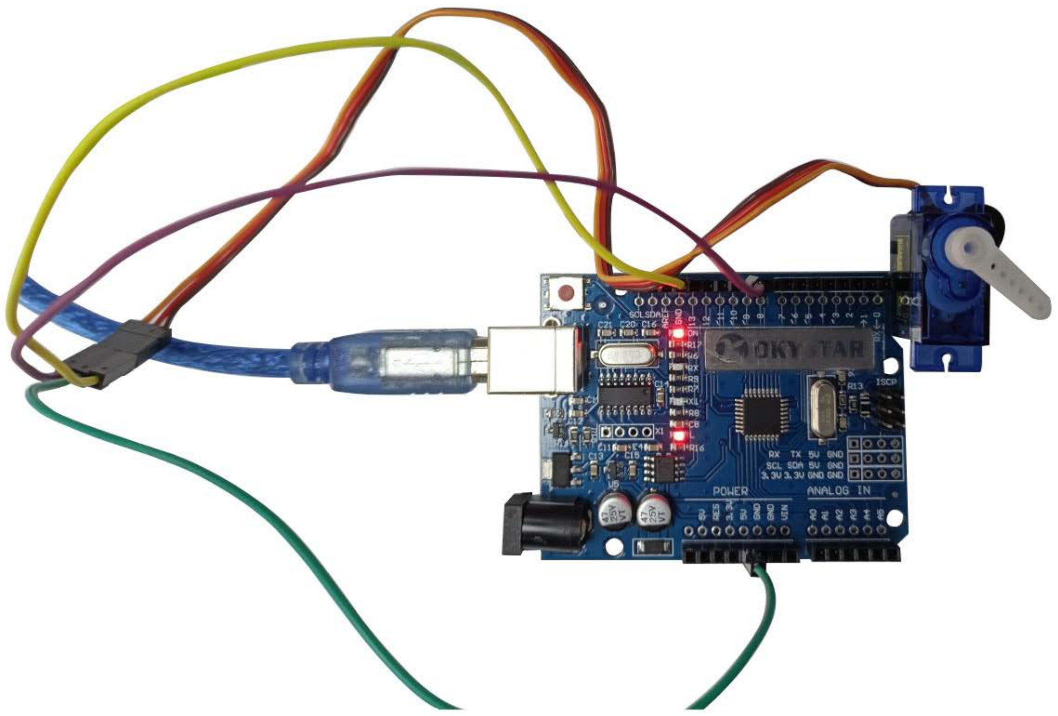
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



Code

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo
                // twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
    myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
    for (pos = 0; pos <= 180; pos += 1) {  // goes from 0 degrees to 180 degrees
        // in steps of 1 degree
        myservo.write(pos);  // tell servo to go to position in variable 'pos'
        delay(15);           // waits 15ms for the servo to reach the position
    }
    for (pos = 180; pos >= 0; pos -= 1) {  // goes from 180 degrees to 0 degrees
        myservo.write(pos);  // tell servo to go to position in variable 'pos'
        delay(15);           // waits 15ms for the servo to reach the position
    }
}
```

Summary

In this lesson, we learned how to use a servo motor (SG90) with an UNO R3 development board. The servo motor is connected to the board using three wires: power (red), ground (black or brown), and signal (yellow, orange, or white). We utilized the Servo library to control the servo's position.

The setup function initializes the servo on pin 9, and the loop function continuously rotates the servo from 0 to 180 degrees and back to 0 degrees, creating a sweeping motion.

Each step of the motion is controlled by a PWM signal sent from the Arduino to the servo.

Lesson 19
Stepper Motor

Overview

In this lesson, you will learn a fun and easy way to drive a stepper motor.

The stepper motor we are using comes with its own driver board, making it easy to connect to our UNO.

Componets

UNO R3 Compatible Development Board

1 x 830 Tie-Points Breadboard

1 x ULN2003 Stepper Motor Driver Module

1 x Stepper Motor

1 x 9V1A Adapter

1 x Power Supply Module

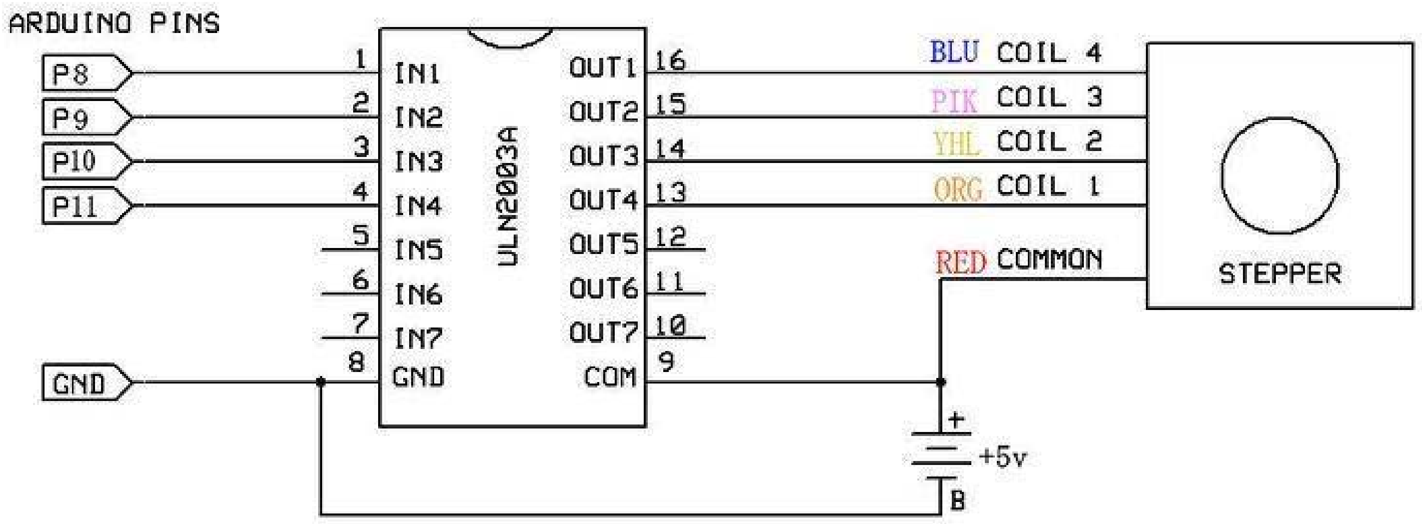
6 x Female to Male DuPont Wires

1 x Male to Male Jumper Wire

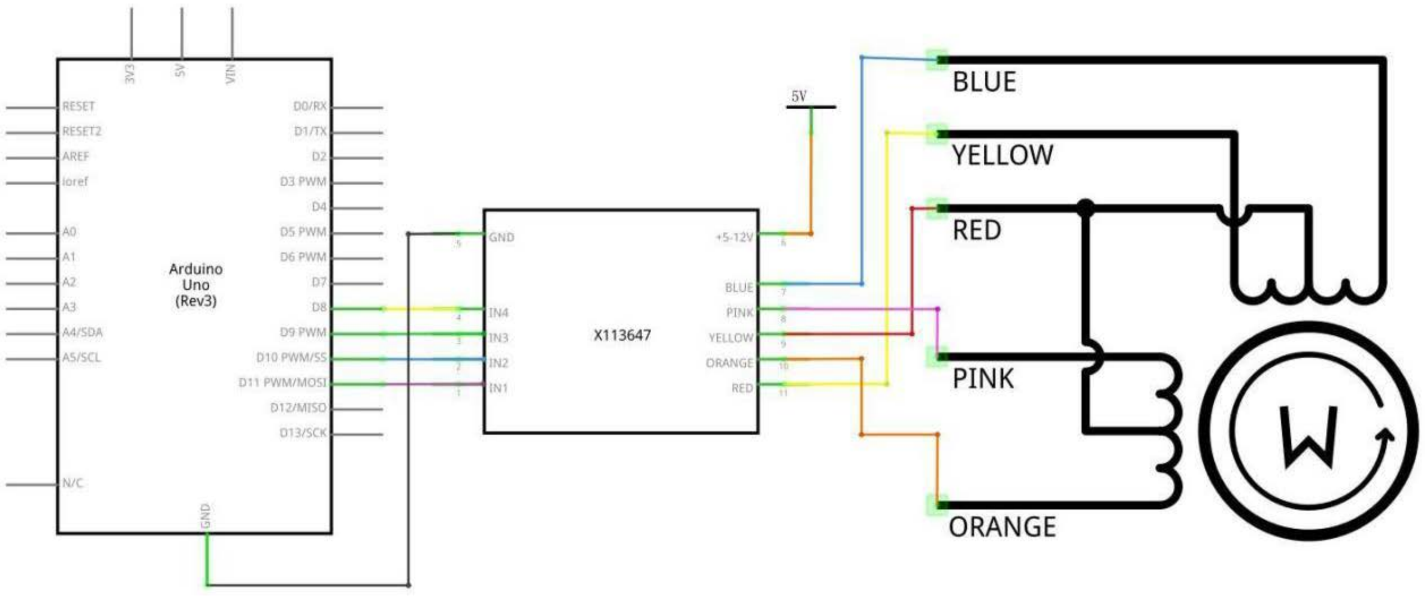
Relationship between Components

A stepper motor is an electromechanical device that converts electrical pulses into discrete mechanical movements. The motor's rotation is directly related to the sequence of the applied pulses, the frequency of these pulses, and the number of pulses applied.
One of the significant advantages of a stepper motor is its ability to be accurately controlled in an open-loop system, eliminating the need for expensive sensing and feedback devices.
The ULN2003 driver board interfaces the stepper motor to the Arduino, making it easy to control the motor using Arduino's digital pins.

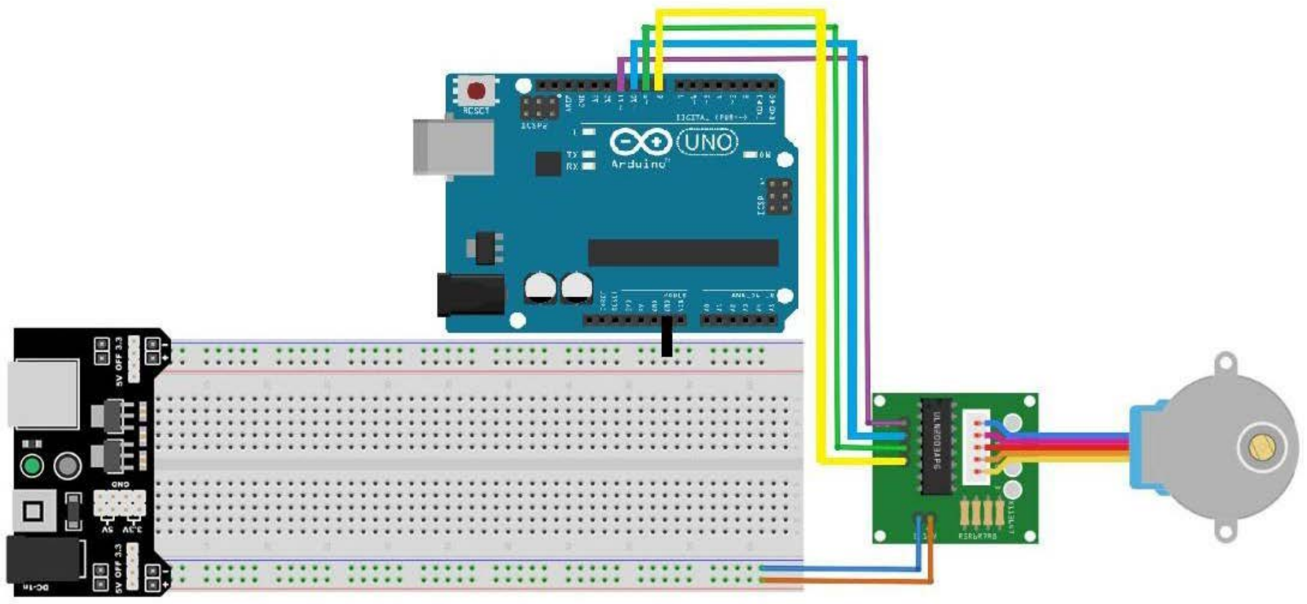
Schematic



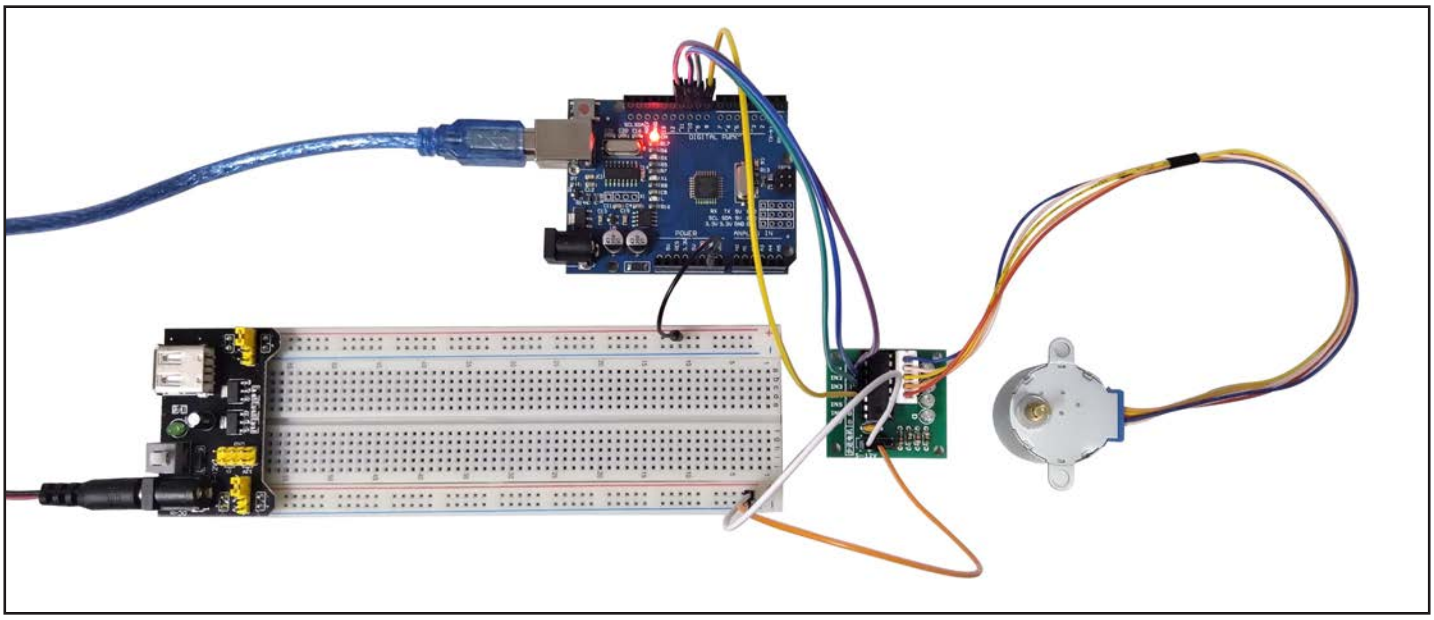
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



Code

```
#include <Stepper.h>

const int stepsPerRevolution = 1500;  // change this to fit the number of steps per revolution

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 10, 9, 11);

void setup() {
    // set the speed at 20 rpm:
    myStepper.setSpeed(20);
    // initialize the serial port:
    Serial.begin(9600);
}

void loop() {
    // step one revolution in one direction:
    Serial.println("clockwise");
    myStepper.step(stepsPerRevolution);
    delay(500);
    // step one revolution in the other direction:
    Serial.println("counterclockwise");
    myStepper.step(-stepsPerRevolution);
    delay(500);
}
```

Summary

In this lesson, we learned how to drive a stepper motor using an UNO R3 development board and a ULN2003 driver module. The stepper motor converts electrical pulses into discrete mechanical movements, allowing precise control over the motor’s position.
The ULN2003 driver module makes it easy to connect the stepper motor to the Arduino.
The provided code initializes the stepper motor and rotates it in both clockwise and counterclockwise directions, showcasing the motor’s capability to perform controlled movements based on the input pulses.

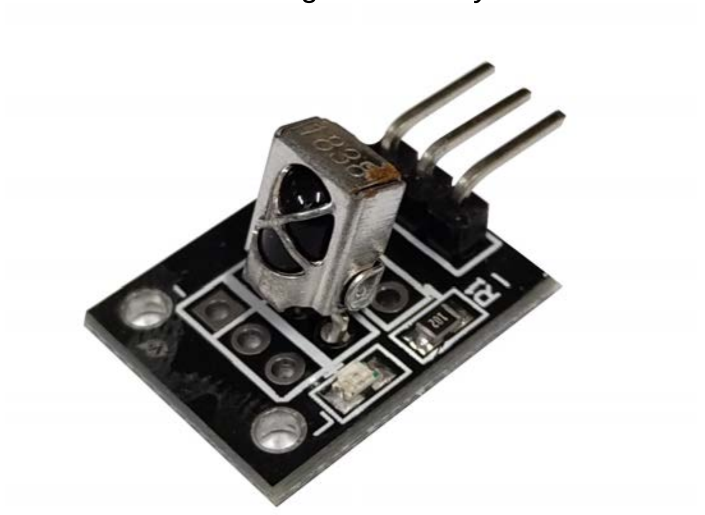
Lesson 20
IR Receiver Module

Overview

Using an IR Remote is a great way to have wireless control of your project.
Infrared remotes are simple and easy to use. In this tutorial, we will be connecting the IR receiver to the UNO and then using a library specifically designed for this sensor.
In our sketch, we will include all the IR Hexadecimal codes available on this remote.
We will also detect if the code was recognized and if we are holding down a key.

Componets

UNO R3 Compatible Development Board
1 x IR Receiver Module
1 x IR Remote
3 x Female to Male DuPont Wires



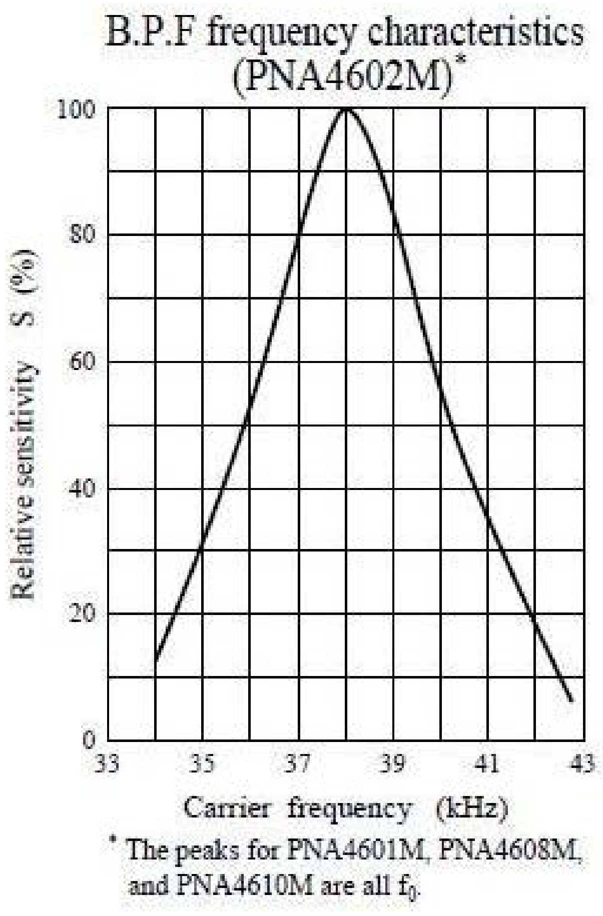
Relationship between Components

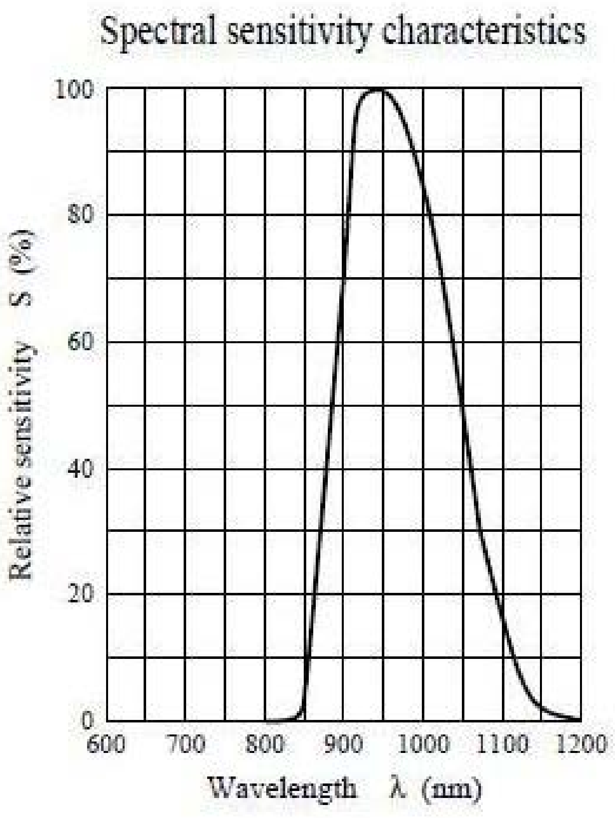
IR detectors are small microchips with a photocell that are tuned to listen to infrared light.
They are typically used for remote control detection. Inside the remote control is a matching IR LED, which emits IR pulses to communicate with the IR receiver.

The IR receiver detects the modulated IR light at 38 KHz and converts it into a digital signal that the Arduino can process.

The connections to the IR receiver are signal (S), voltage (middle pin, 5V), and ground (GND).

What You Can Measure

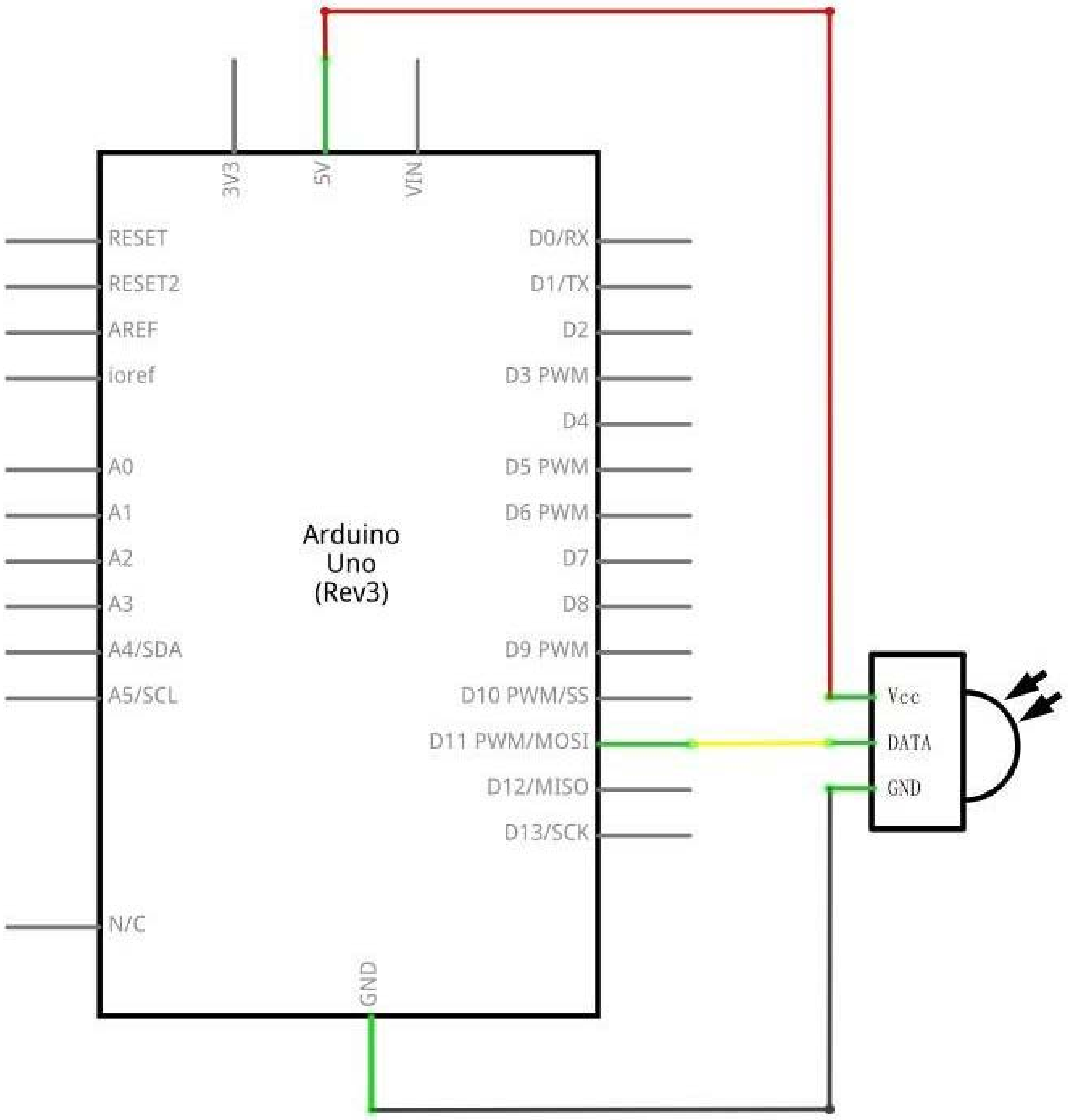




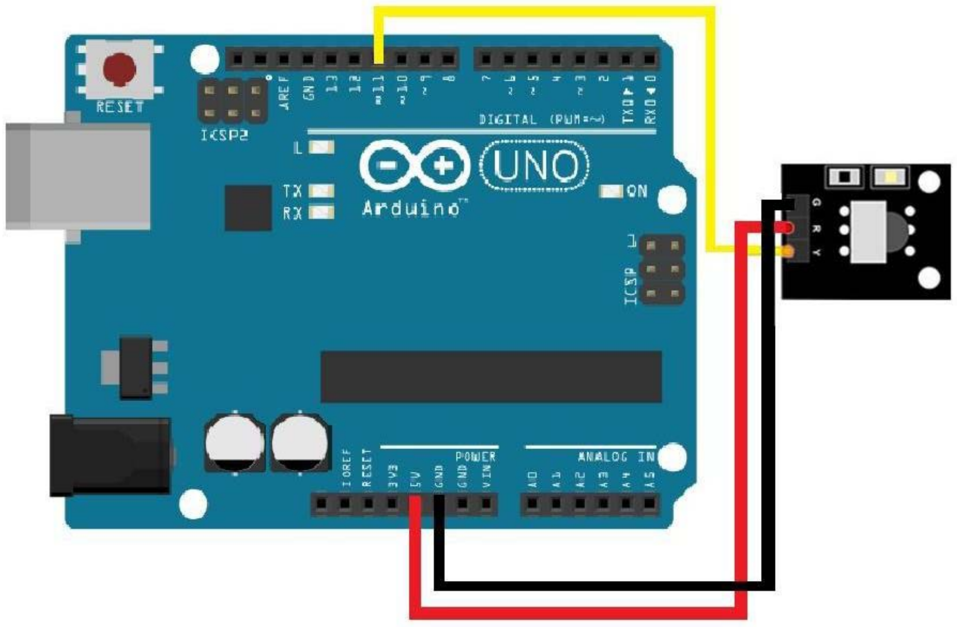
As you can see from these datasheet graphs, the peak frequency detection is at 38 KHz and the peak LED color is 940 nm. You can use from about 35 KHz to 41 KHz but the sensitivity will drop off so that it won't detect as well from afar. Likewise, you can use 850 to 1100 nm LEDs but they won't work as well as 900 to 1000nm so make sure to get matching LEDs! Check the datasheet for your IR LED to verify the wavelength.

Try to get a 940nm - remember that 940nm is not visible light!

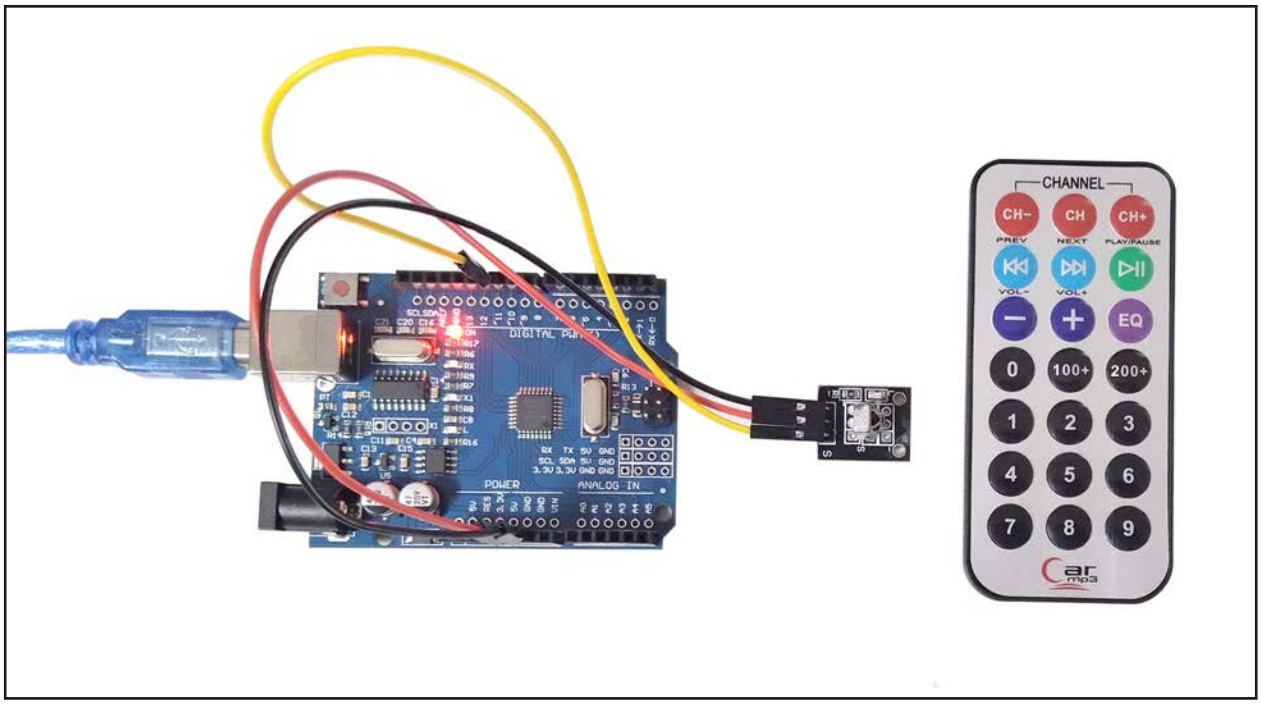
Connection Diagram



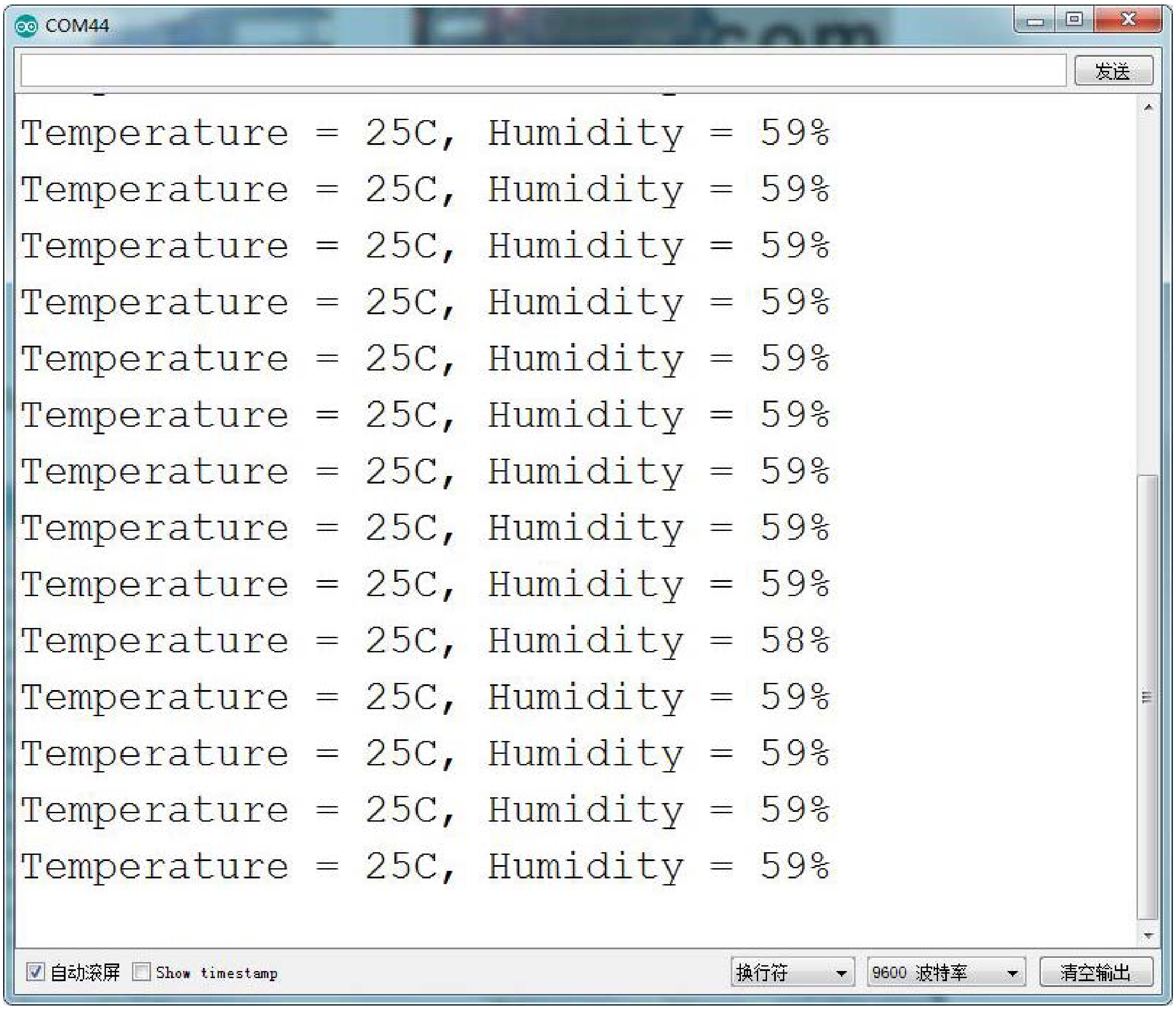
Wiring Schematic



Physical Wiring Diagram



Open the serial monitor as follows:



```
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 58%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
Temperature = 25C, Humidity = 59%
```

Code

```
#include "IRremote.h"

int receiver = 11; // Signal Pin of IR receiver to Arduino Digital Pin 11

// Declare objects
IRrecv irrecv(receiver);  // create instance of 'irrecv'
decode_results results;   // create instance of 'decode_results'

// Function to translate IR codes
void translateIR() {
    switch(results.value) {
        case 0xFFA25D: Serial.println("CH-"); break;
        case 0xFFE21D: Serial.println("CH"); break;
        case 0xFF629D: Serial.println("CH+"); break;
        case 0xFF22DD: Serial.println("VOL-"); break;
        case 0xFF02FD: Serial.println("VOL+"); break;
        case 0xFFC23D: Serial.println("FAST FORWARD"); break;
        case 0xFFE01F: Serial.println("-"); break;
        case 0xFFA857: Serial.println("+"); break;
        case 0xFF906F: Serial.println("EQ"); break;
        case 0xFF9867: Serial.println("100+"); break;
        case 0xFFB04F: Serial.println("200+"); break;
        case 0xFF6897: Serial.println("0"); break;
        case 0xFF30CF: Serial.println("1"); break;
        case 0xFF18E7: Serial.println("2"); break;
        case 0xFF7A85: Serial.println("3"); break;
        case 0xFF10EF: Serial.println("4"); break;
        case 0xFF38C7: Serial.println("5"); break;
        case 0xFF5AA5: Serial.println("6"); break;
        case 0xFF42BD: Serial.println("7"); break;
        case 0xFF4AB5: Serial.println("8"); break;
        case 0xFF52AD: Serial.println("9"); break;
        case 0xFFFFFFFF: Serial.println("REPEAT"); break;
        default: Serial.println("other button"); break;
    }
    delay(500); // Do not get immediate repeat
}
```

Code (cont.)

```
void setup() {
    Serial.begin(9600);
    Serial.println("IR Receiver Button Decode");
    irrecv.enableIRIn(); // Start the receiver
}

void loop() {
    if (irrecv.decode(&results)) { // Have we received an IR signal?
        translateIR();
        irrecv.resume(); // Receive the next value
    }
}
```

Summary

In this lesson, we learned how to use an IR receiver module with an UNO R3 development board. The IR receiver detects modulated IR light at 38 KHz and converts it into a digital signal.

We used a library to decode the signals from an IR remote and printed the corresponding button presses to the serial monitor.

The setup function initializes the IR receiver, and the loop function continuously checks for incoming IR signals, translating and displaying the recognized codes.

Lesson 21
LCD Display

Overview

In this lesson, you will learn how to wire up and use an alphanumeric LCD display.

The display has an LED backlight and can show two rows with up to 16 characters on each row.

It is designed for displaying text, with white characters on a blue background.

We will run an Arduino example program for the LCD library to display text on the screen.

In the next lesson, we will use the display to show temperature readings using sensors.

Componets

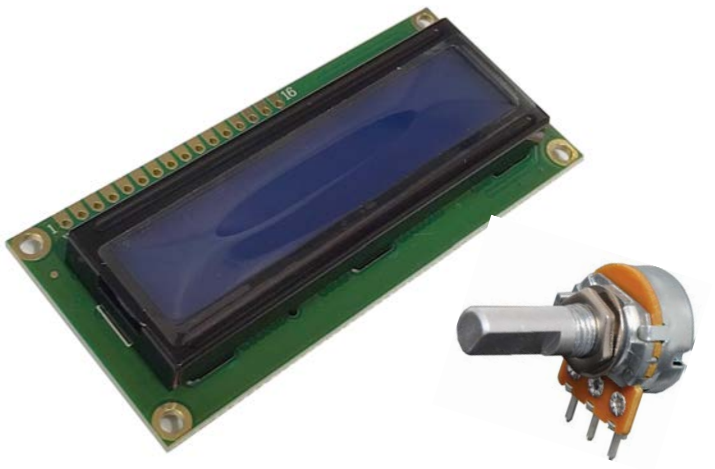
UNO R3 Compatible Development Board

1 x LCD1602 Modules

1 x Potentiometer (10k)

1 x 830 Tie-Points Breadboard

16 x Male to Male Jumper Wires



Relationship between Components

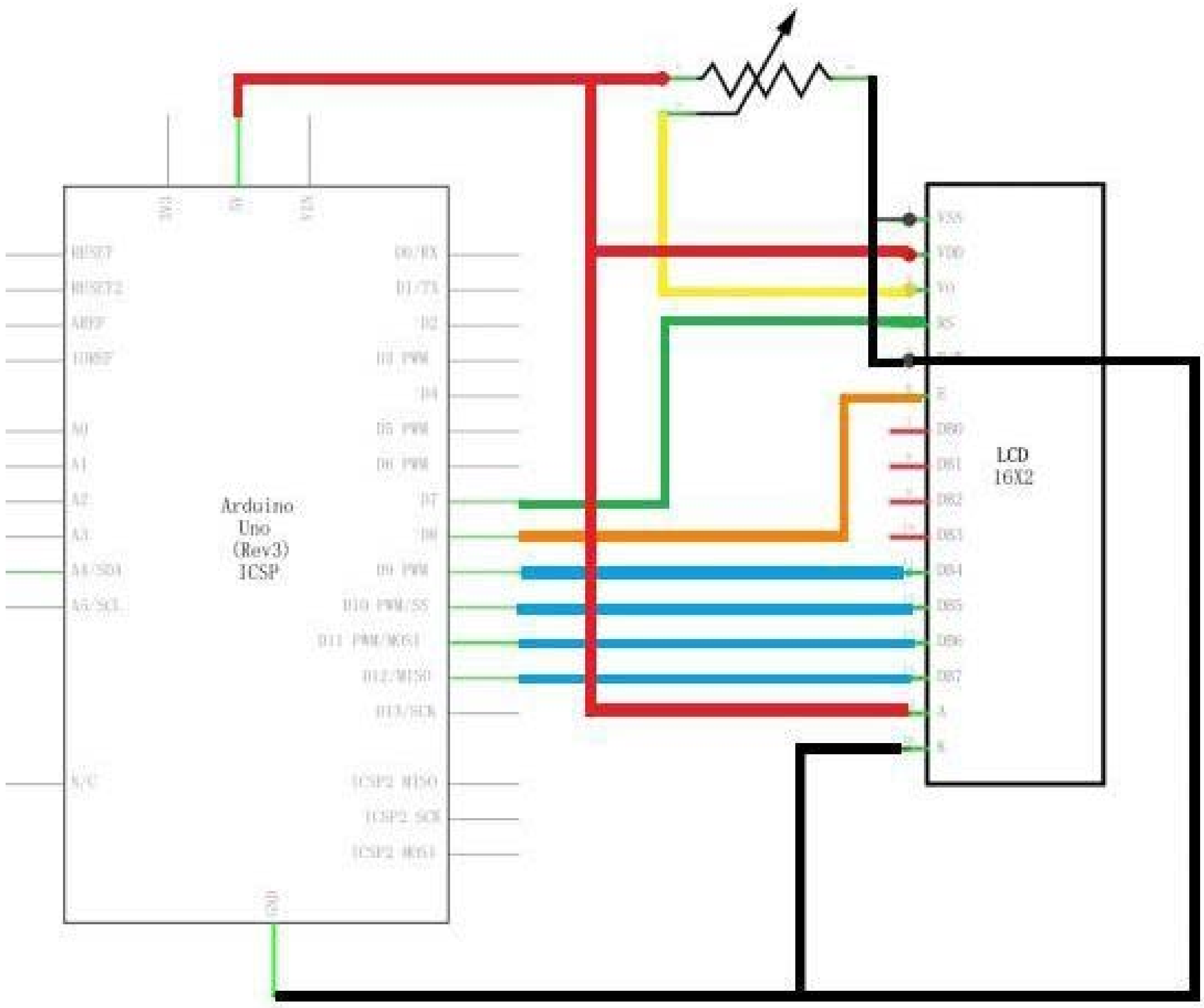
The LCD1602 module is used to display alphanumeric characters.

It requires multiple connections to the Arduino, including power (5V and GND), control pins (RS, RW, E), and data pins (D0-D7).

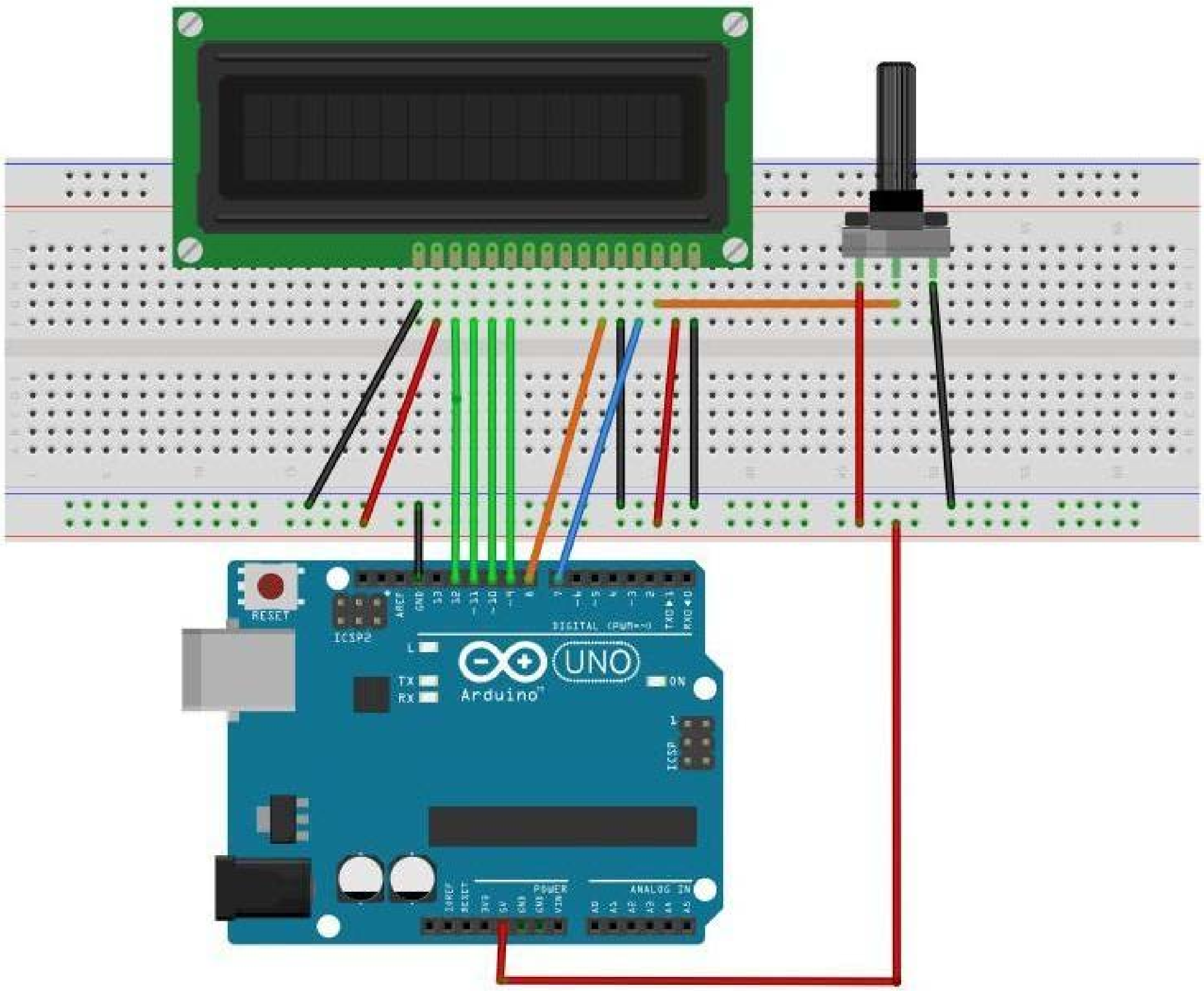
The potentiometer is used to adjust the contrast of the display.

Proper wiring and initialization in the code allow the Arduino to control what is displayed on the screen.

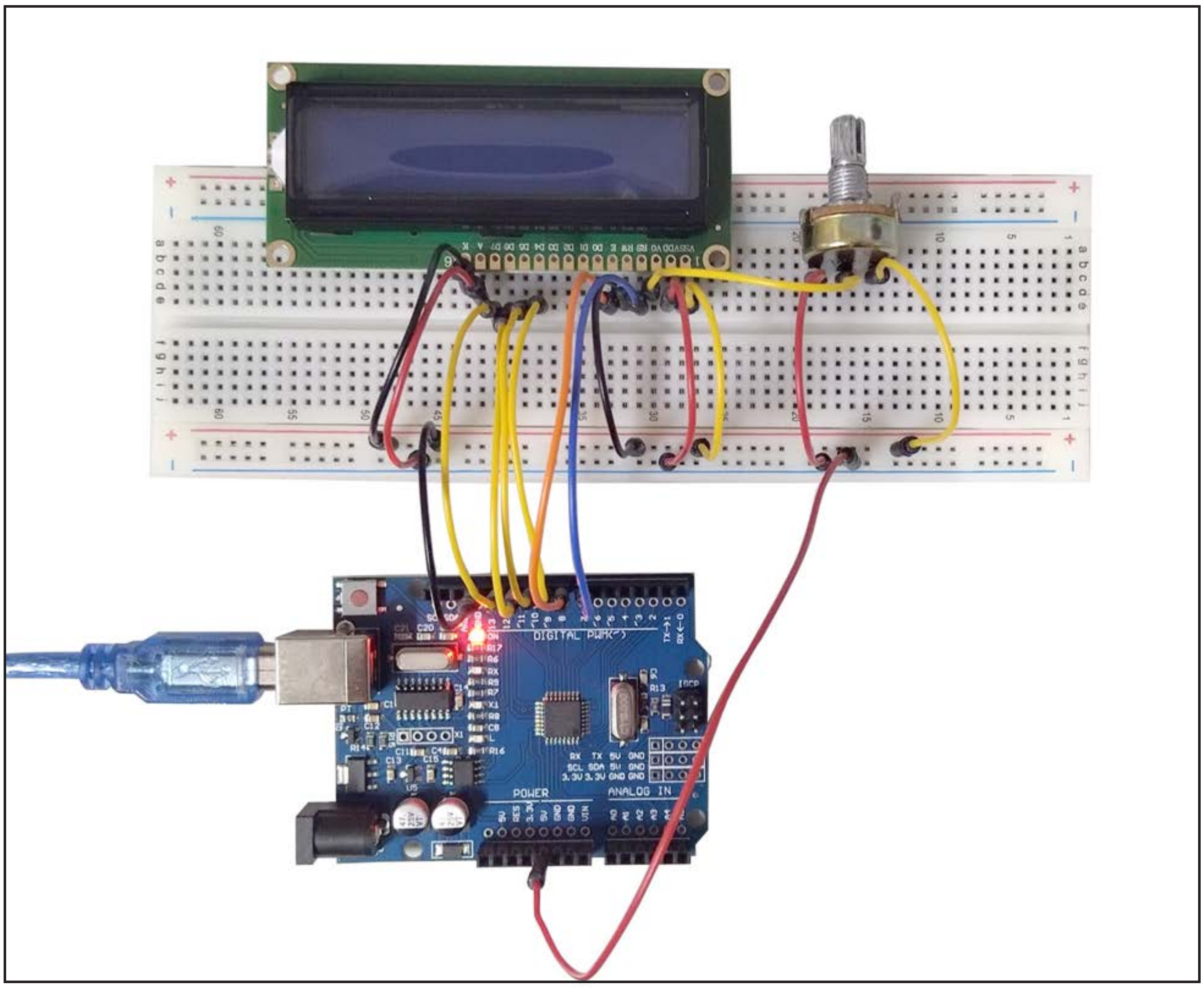
Connection Diagram



Wiring Schematic



Physical Wiring Diagram



The LCD display needs six Arduino pins, all set to be digital outputs, as well as 5V and GND connections. There are several connections to be made, and aligning the display with the top of the breadboard helps to identify its pins.

The potentiometer controls the contrast of the display. If your display is supplied without header pins, follow the instructions in the next section to attach them.

Code

```
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void setup() {
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // Print a message to the LCD.
    lcd.print("Hello, World!");
}

void loop() {
    // set the cursor to column 0, line 1
    // (note: line 1 is the second row, since counting begins with 0):
    lcd.setCursor(0, 1);
    // print the number of seconds since reset:
    lcd.print(millis() / 1000);
}
```

Summary

In this lesson, we learned how to use an LCD1602 display with an UNO R3 development board. The display is connected to the Arduino using six digital pins and requires a potentiometer to adjust its contrast. We used the LiquidCrystal library to initialize the display and print text.

The setup function configures the LCD, and the loop function continuously updates the display to show the number of seconds since the Arduino was reset.

This setup provides a basic understanding of how to interface an LCD display with the Arduino for displaying text and real-time data.

Lesson 22
Membrane Switch Module

Overview

In this project, we will integrate a keypad with an UNO R3 board to read the keys pressed by a user. Keypads are used in various devices, including cell phones, fax machines, microwaves, ovens, door locks, etc. Knowing how to connect a keypad to a microcontroller like the UNO R3 board is valuable for building many different types of commercial products.
When all is connected and programmed correctly, pressing a key will show the corresponding character on the Serial Monitor on your computer.

For this project, we will use a matrix keypad, which has 16 keys (0-9, A-D, *, #) but only 8 output pins. This encoding scheme allows for fewer output pins and connections,
making it more efficient than linear keypads.

Componets

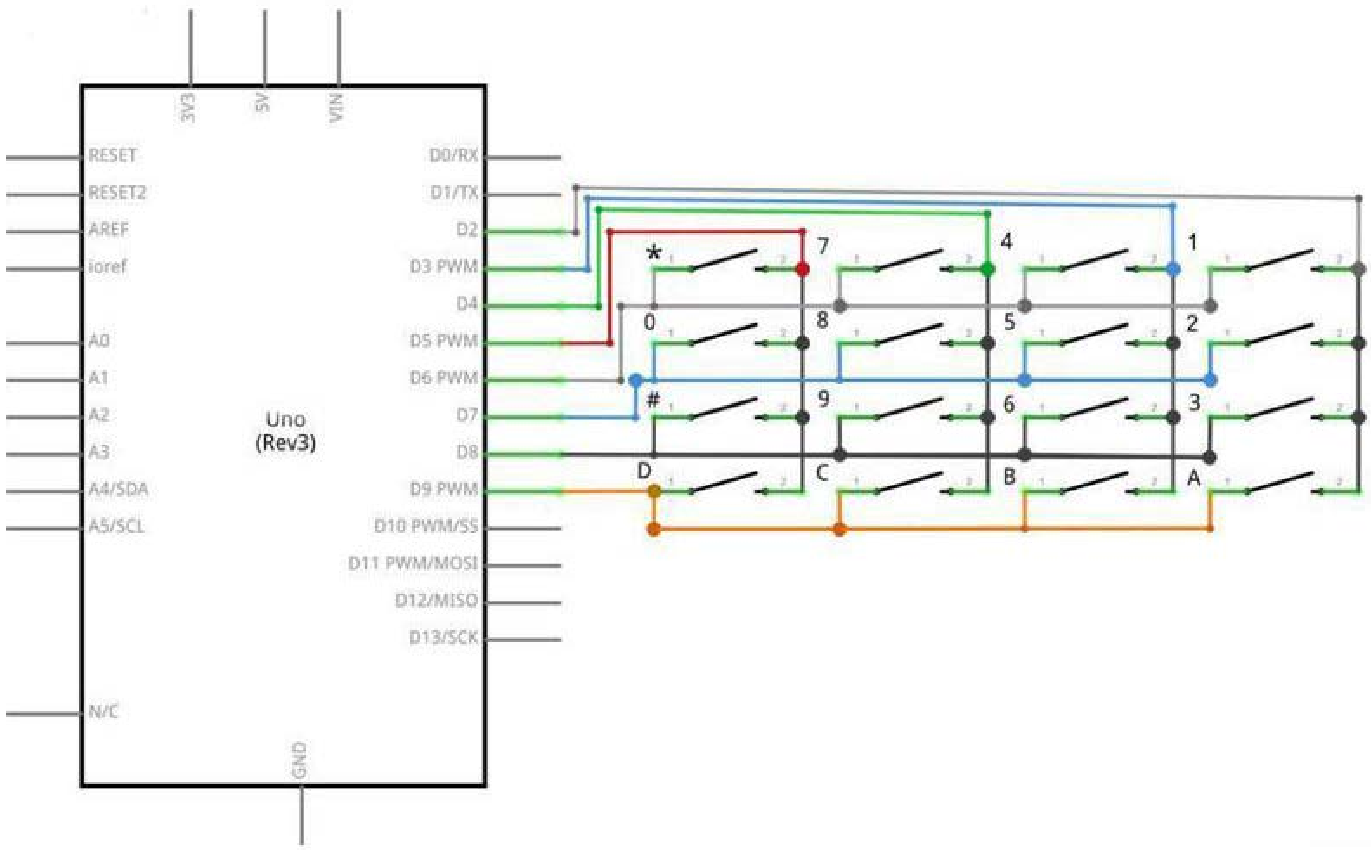
UNO R3 Compatible Development Board
1 x 4x4 Matrix Keypad
15 x Male to Male Jumper Wires
4 x 5mm Red LED
4 x 220 Ohm Resistors

Relationship between Components

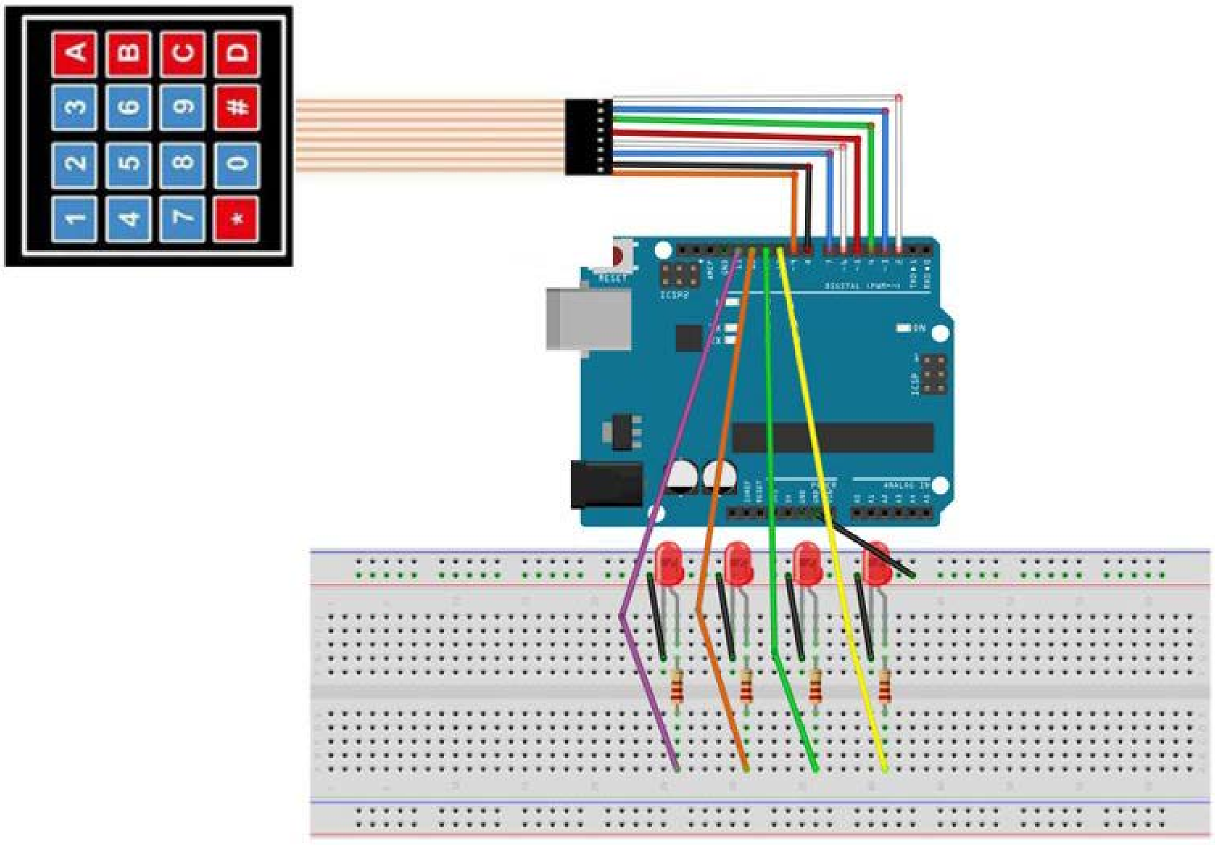
The matrix keypad connects to the UNO R3 through its digital output pins (D9-D2).
The first pin of the keypad connects to D9, the second pin to D8, the third pin to D7,
the fourth pin to D6, the fifth pin to D5, the sixth pin to D4, the seventh pin to D3,
and the eighth pin to D2.

This setup allows the Arduino to read which key is pressed by scanning the rows and columns of the keypad matrix.

Connection Diagram



Wiring Schematic



Keypad Pin	Connects to UNO R3 Pin...
1	D9
2	D8
3	D7
4	D6
5	D5
6	D4
7	D3
8	D2

Code

```
#include <Keypad.h>

const byte ROWS = 4; // four rows
const byte COLS = 4; // four columns

// define the symbols on the buttons of the keypads
char hexaKeys[ROWS][COLS] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6}; // connect to the row pinouts of the keypad
byte colPins[COLS] = {5, 4, 3, 2}; // connect to the column pinouts of the keypad

// initialize an instance of class NewKeypad
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);

void setup() {
    Serial.begin(9600);
    pinMode(10, OUTPUT);
    pinMode(11, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);
}

void loop() {
    char customKey = customKeypad.getKey();
    if (customKey) {
        Serial.println(customKey);
    }
```

Code (cont.)

```
if (customKey == '1') {
    digitalWrite(10, HIGH);
    digitalWrite(11, LOW);
    digitalWrite(12, LOW);
    digitalWrite(13, LOW);
    Serial.println("aaa");
} else if (customKey == '2') {
    digitalWrite(10, LOW);
    digitalWrite(11, HIGH);
    digitalWrite(12, LOW);
    digitalWrite(13, LOW);
    Serial.println("bbb");
} else if (customKey == '3') {
    digitalWrite(10, LOW);
    digitalWrite(11, LOW);
    digitalWrite(12, HIGH);
    digitalWrite(13, LOW);
    Serial.println("ccc");
} else if (customKey == '4') {
    digitalWrite(10, LOW);
    digitalWrite(11, LOW);
    digitalWrite(12, LOW);
    digitalWrite(13, HIGH);
} else if (customKey == '5') {
    digitalWrite(10, HIGH);
    digitalWrite(11, HIGH);
    digitalWrite(12, LOW);
    digitalWrite(13, LOW);
} else if (customKey == '6') {
    digitalWrite(10, LOW);
    digitalWrite(11, LOW);
    digitalWrite(12, HIGH);
    digitalWrite(13, HIGH);
} else if (customKey == '7') {
    digitalWrite(10, HIGH);
    digitalWrite(11, LOW);
```

Code (cont.)

```
        digitalWrite(12, HIGH);
        digitalWrite(13, LOW);
    } else if (customKey == '8') {
        digitalWrite(10, LOW);
        digitalWrite(11, HIGH);
        digitalWrite(12, LOW);
        digitalWrite(13, HIGH);
    } else if (customKey == '9') {
        digitalWrite(10, HIGH);
        digitalWrite(11, HIGH);
        digitalWrite(12, HIGH);
        digitalWrite(13, HIGH);
    } else if (customKey == '0') {
        digitalWrite(10, HIGH);
        delay(400);
        digitalWrite(11, HIGH);
        delay(400);
        digitalWrite(12, HIGH);
        delay(400);
        digitalWrite(13, HIGH);
    } else if (customKey == 'A') {
        digitalWrite(10, LOW);
        digitalWrite(11, LOW);
        digitalWrite(12, LOW);
        digitalWrite(13, LOW);
    }
}
```

Summary

In this lesson, we learned how to integrate a 4x4 matrix keypad with an UNO R3 development board. The matrix keypad allows for efficient wiring and fewer output pins. We used the Keypad library to read key presses from the keypad and display them on the Serial Monitor.
The setup function initializes the keypad and sets the LED pins as outputs.
The loop function reads the key pressed and performs corresponding actions, such as lighting up specific LEDs and printing messages to the Serial Monitor.

Lesson 23
Sound Detection Sensor Module

Overview

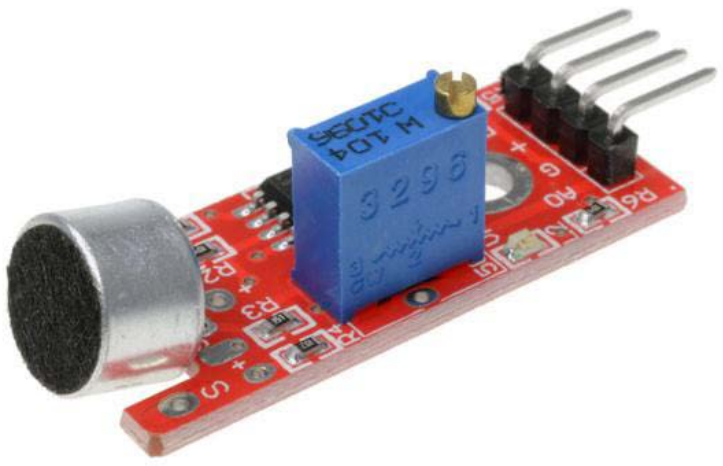
The sound detection sensor is a small board that combines a microphone and some processing circuits, capable of detecting sound volume.

The sensor can be used for various purposes, from industry to simple hobbies or play.

In this course, we will guide you on how to connect and use this sound detection module. It will explain how the circuit works, detail some tips for getting the best performance from the sound sensor, and introduce some projects demonstrating its use.

Componets

UNO R3 Compatible Development Board
1 x Sound Detection Sensor
1 x Breadboard
1 x USB Cable
1 x PC
Jumper Wires



Relationship between Components

The sound detection sensor module has a built-in capacitive electret microphone that is highly sensitive to sound. The sound wave vibrates the electret film, resulting in a corresponding change in voltage, which the sensor can detect.

The LM393 is used as a power amplifier to enhance this weak signal.
The sensitivity can be adjusted using the potentiometer on the module.
When the sound level exceeds the set point, an LED indicator on the sensor module illuminates, and the output is sent low.

The sound sensor module has four pins that need to be connected to the Arduino:

AO (Analog Output): Connect this to analog input pin A0 on the Arduino.
GND (Ground): Connect this to the ground (GND) pin on the Arduino.
VCC (Power): Connect this to the +5V pin on the Arduino.
DO (Digital Output): Connect this to digital pin 2 on the Arduino.

Calibration

At the top of the sound sensor module, there is a small flathead screw that can be used to adjust the sensitivity and the analog output. To calibrate the sensor:

Make some noise while adjusting the screw until the sensor's LED indicator flashes in rhythm with the sound.

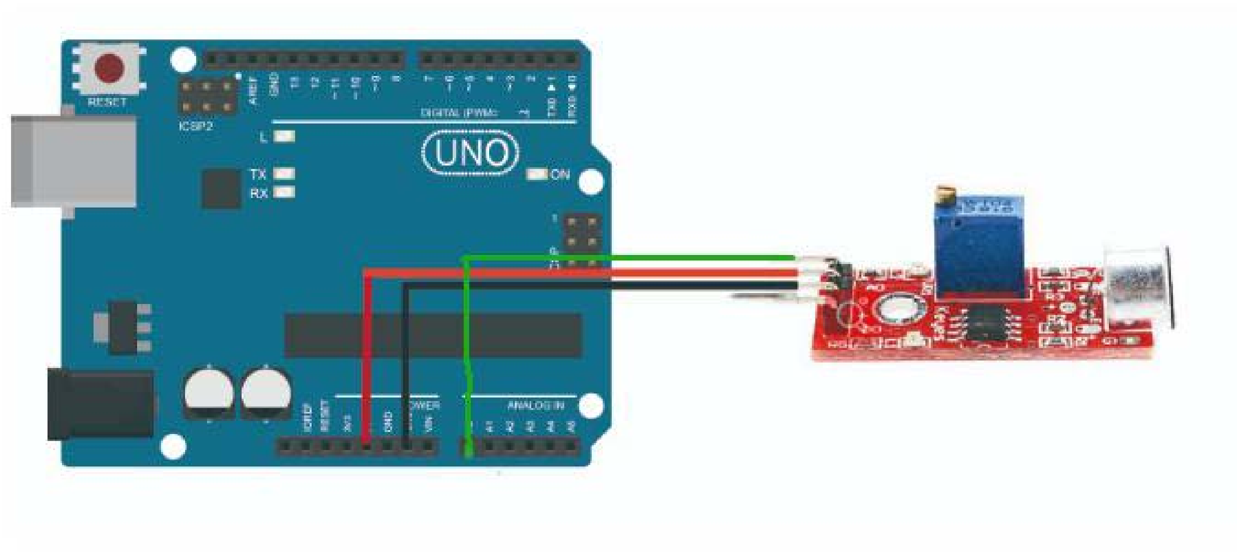
Examples

Digital Detection: Sound-Sensitive Lamp

In this setup, the sound detection sensor module is connected to the Arduino's digital pin.
The onboard LED will illuminate whenever the sensor detects sound.
Analog Detection: Sound Sensor Light

This example demonstrates using the analog pin to detect sound intensity.
The microphone sensor detects the surrounding sound intensity, and the LED lights up when the sound exceeds a specific threshold.

Wiring Schematic



Code

```
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    int soundValue = analogRead(A0);
    Serial.println(soundValue);
    if (soundValue > 30) {
        digitalWrite(LED_BUILTIN, HIGH);
        delay(3000);
    } else {
        digitalWrite(LED_BUILTIN, LOW);
    }
}
```

Summary

In this lesson, we learned how to use a sound detection sensor module with an UNO R3 development board. The sensor detects the intensity of the sound in the surrounding environment and outputs a corresponding signal. By adjusting the potentiometer, we can set the sensitivity of the sensor.
The provided code reads the analog value from the sensor and illuminates the onboard LED when the sound level exceeds a certain threshold. The analog and digital values from the sensor can be viewed on the serial monitor, helping to calibrate the sensor for various applications.

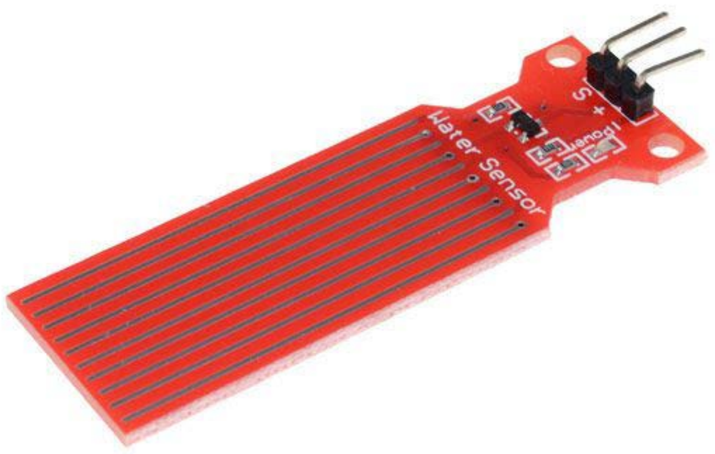
Lesson 24
Water Level Sensor Module

Overview

Water sensor bricks are designed for water detection and can be widely used to detect rainfall, water levels, and even liquid leaks. In this lesson, we will learn what a water sensor is and how it can be used with the UNO R3 board to detect water.

Componets

UNO R3 Compatible Development Board
1 x Water Sensor
1 x Red LED
1 x Breadboard
1 x USB Cable
1 x PC
Jumper Wires



Relationship between Components

The water sensor is a small, lightweight module that detects water levels by measuring the size of water droplets through a series of exposed parallel lines. The water quantity is converted into an analog signal, which the Arduino can use to perform functions like triggering a water level alarm.
The sensor works by using a weak pull-up resistor and a series of exposed traces that connect to the ground and sensor traces. When water bridges these traces, the sensor detects it and sends a signal to the Arduino.

The sensor has three pins:

VCC: Connected to +5V
GND: Connected to ground
S: Signal pin connected to the analog input pin (A0) on the Arduino

Working Principle

The sensor operates by connecting a series of exposed traces to the ground and interleaving them between the ground trace and the sensor trace. A 1 MΩ weak pull-up resistor is used to connect the sensor trace, pulling up the sensor's voltage until water droplets cause a short circuit between the sensor and grounding traces. This change can be detected using ARDUINO digital I/O pins, or with analog pins to measure the water-induced contact between the grounding and sensor wiring.

Using the Sensor as a Water Level Detector

To use the sensor as a water level detector, it should be installed inside a tank at the desired water level. Ensure that the sensor's parallel lines are perpendicular to the water level.
When the sensor is submerged, pin S will produce a higher value, indicating the presence of water.

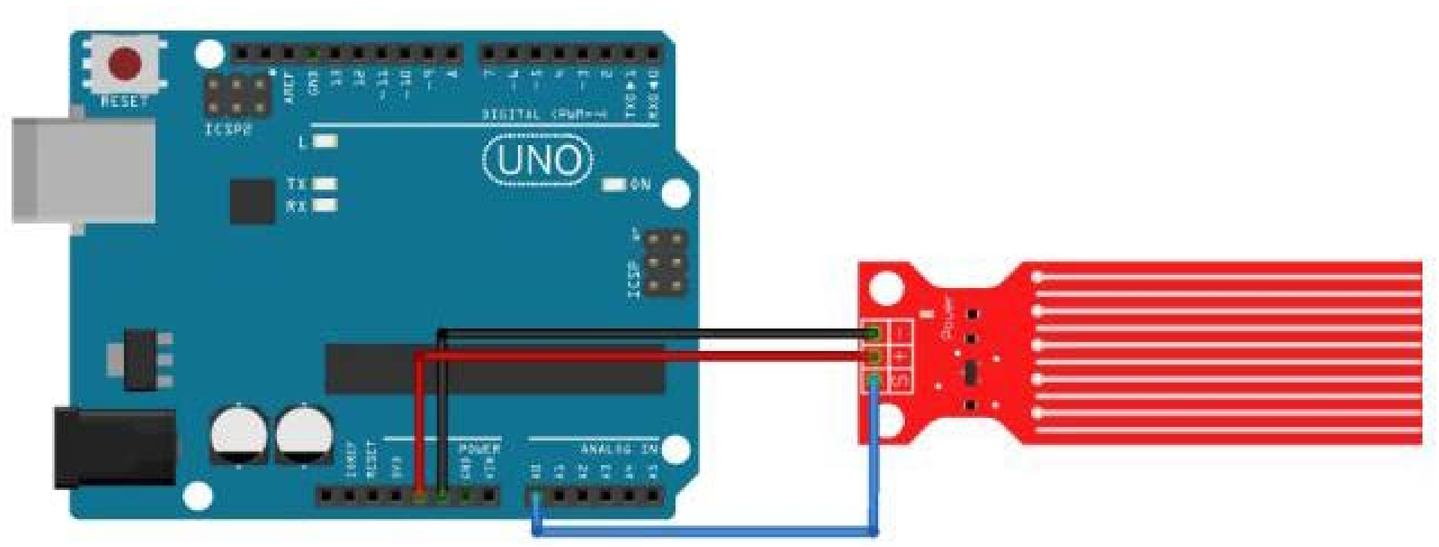
Using the Sensor as a Rain Detector

To detect rain, place the sensor horizontally so that raindrops fall directly onto it.
When raindrops form on the sensor's surface, they create a water film that increases the value of pin S, allowing you to determine if it is raining.

ARDUINO Liquid Level Detector

In this example, we will demonstrate how to use a water sensor to measure the water level in a tank. The S pin of the sensor will be connected to an analog input on the ARDUINO. As the sensor's surface becomes more submerged in water, the value read by the ARDUINO will increase.
This occurs because the water in the tank, typically not pure (H_2O), acts as a conductor, allowing the sensor to detect the presence of water. However, this type of sensor is less effective for measuring the water level in tanks containing purified water, as pure water is not conductive.

Wiring Schematic



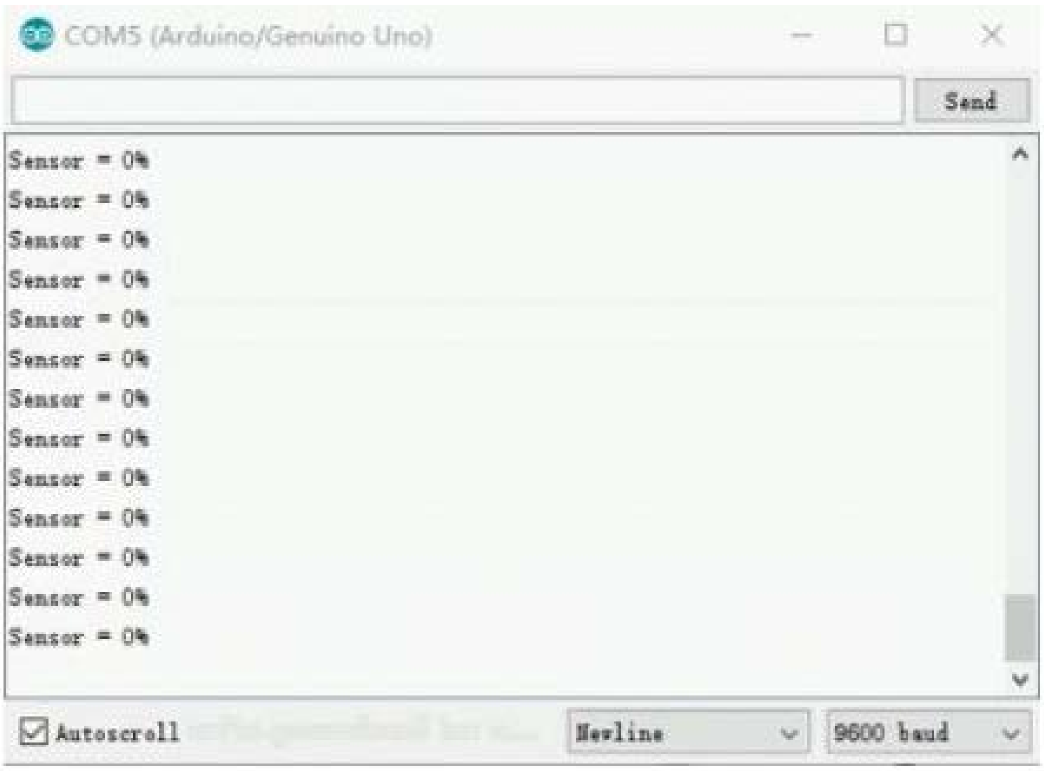
Here, we connect the signal pin (S) to analog pin A0. This allows the ARDUINO board to read analog voltage values.

Code

```
const int analogInPin = A0;
int sensorValue = 0;

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

void loop() {
    sensorValue = analogRead(analogInPin);
    Serial.print("Sensor = ");
    Serial.print(sensorValue * 100 / 1024);
    Serial.println("%");
    delay(1000);
}
```

After the uploaded code is complete the sensor is ready to immerse in the water.

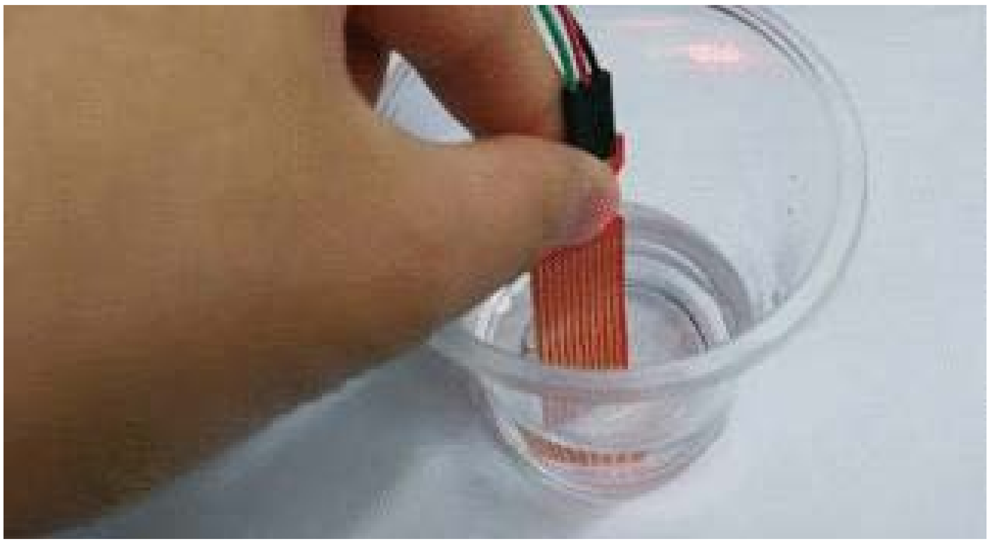
The serial monitor shows:



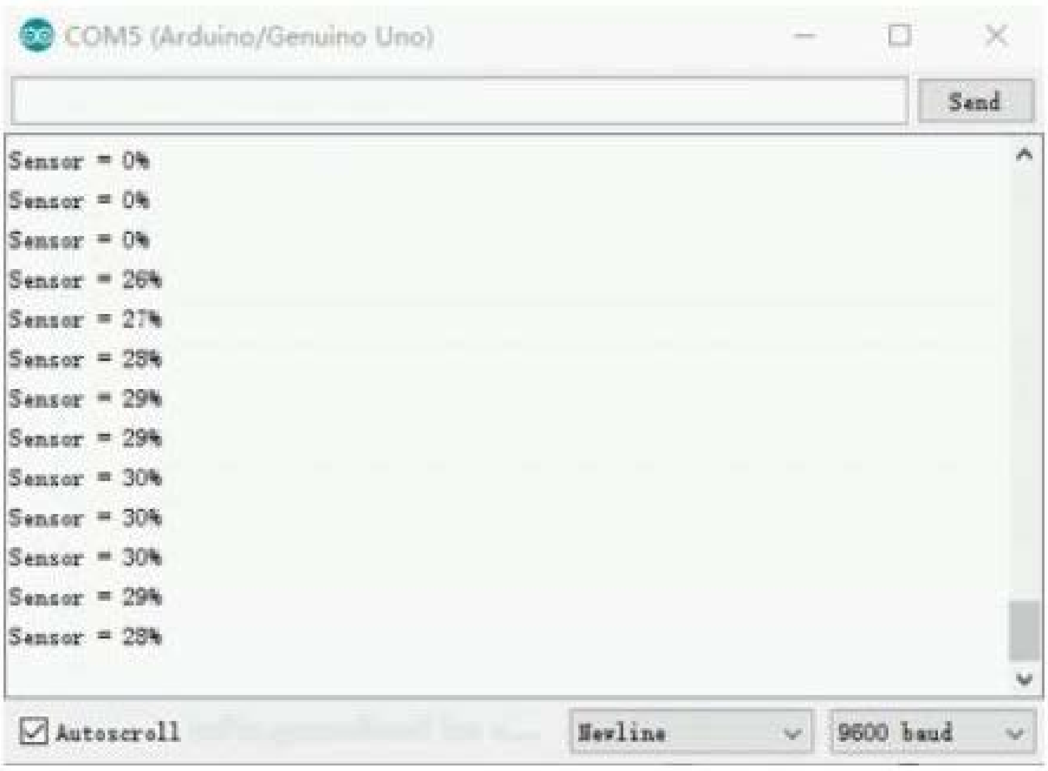
```
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 0%
```

Newline   9600 baud

Immerse the sensor in water



The serial monitor shows:



```
Sensor = 0%
Sensor = 0%
Sensor = 0%
Sensor = 26%
Sensor = 27%
Sensor = 28%
Sensor = 29%
Sensor = 29%
Sensor = 30%
Sensor = 30%
Sensor = 30%
Sensor = 29%
Sensor = 28%
```

Summary

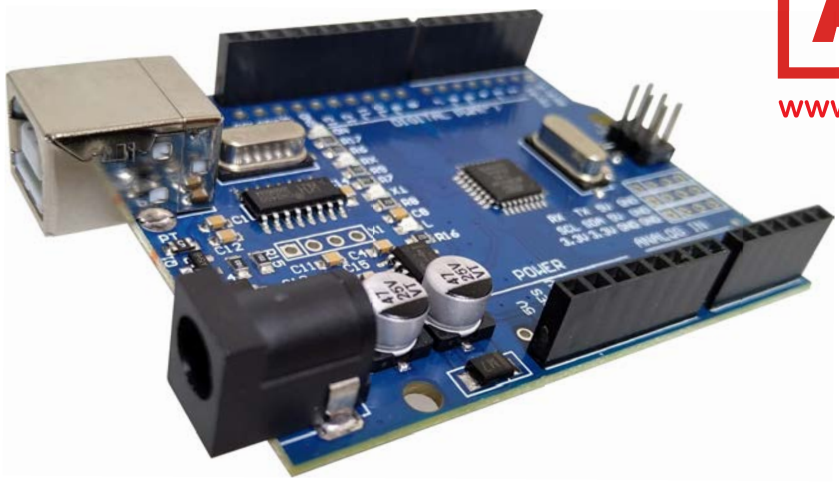
In this lesson, we learned how to use a water sensor with an UNO R3 development board to detect water levels. The sensor detects water by measuring the resistance between exposed traces, which changes when water bridges the gap between them.

The Arduino reads this value as an analog input and prints the percentage of the sensor covered by water to the serial monitor. This setup can be used for various applications, such as rainfall detection, leakage detection, and tank overflow alarms.

The provided code reads the sensor value and outputs it to the serial monitor, helping to monitor and manage water levels in different scenarios.

Well Done!

ARD 2
www.wiltronics.com.au



Congratulations on completing "A Step by Step Guide with 26 projects"!

You've embarked on an exciting journey into the world of Arduino and electronics. Throughout this guide, you've learned valuable skills and built an impressive range of projects, from basic LEDs and buttons to more advanced modules like ultrasonic sensors and DC motors.

Your dedication and perseverance in completing these lessons showcase your growing expertise in the field of electronics and programming.

As you reach the end of this guide, take a moment to reflect on your achievements and the knowledge you've gained. With these newfound skills, you're now equipped to create even more innovative projects and bring your ideas to life. Whether you're pursuing a hobby or exploring a future career path, your passion for electronics and Arduino will undoubtedly lead to exciting opportunities.

Remember, this is just the beginning of your journey. Keep exploring, experimenting, and honing your skills.

The world of Arduino and electronics is vast and ever-evolving, and you're now a part of this dynamic community. Congratulations once again, and we look forward to seeing the incredible projects you'll create in the future! Happy tinkering!

Appendixes

Code Glossary

analogRead(pin)

Reads the analog value from an analog pin and returns a value between 0 and 1023.

The Arduino has 10-bit analog-to-digital converters (ADC), it can read analog voltages in this range.

analogWrite(pin, value)

Provides a PWM (Pulse Width Modulation) signal on a specific pin.

"pin" is the pin number capable of PWM, and "value" can be an integer between 0 (0% duty cycle) and 255 (100% duty cycle) to control the analog output.

attachInterrupt(digitalPin, ISR, mode)

Attaches an interrupt to a specific digital pin. "digitalPin" is the pin number,

"ISR" is the Interrupt Service Routine function that will be executed when the interrupt occurs,

and "mode" specifies the trigger condition (RISING, FALLING, CHANGE, or LOW).

delay(milliseconds)

Pauses the execution of the Arduino sketch for the specified number of milliseconds.

Useful for creating delays in the program flow.

detachInterrupt(digitalPin)

Detaches the interrupt from the specified digital pin.

digitalWrite(pin, value)

Sets the output of a digital pin to HIGH or LOW.

"pin" is the pin number, and "value" can be either HIGH (5V) or LOW (0V).

digitalRead(pin)

Reads the state of a digital pin and returns either HIGH (if the pin is receiving 5V) or LOW (if the pin is receiving 0V).

Code Glossary

else:

An optional block of code executed when the condition of the preceding "if" statement is false.

if (condition):

A conditional statement that executes the code inside the block if the specified condition is true.

noTone(pin):

Stops the generation of a tone on the specified pin.
Used to turn off sound output after using the "tone()" function.

pinMode(pin, mode):

Configures a specific digital pin as INPUT or OUTPUT.
"pin" is the pin number, and "mode" can be either INPUT or OUTPUT.

Serial.begin(baudRate):

Initializes serial communication with the specified baud rate.

Serial.print(data):

Prints data to the serial port for debugging or communication with a computer.

Serial.println(data):

Prints data to the serial port and adds a newline character ('\n') at the end.

shiftOut(dataPin, clockPin, bitOrder, value):

Sends a byte of data serially to a shift register or another device that supports the shiftOut protocol.
"dataPin" is the pin used for data transmission,
"clockPin" is the pin used for the clock signal, "bitOrder" specifies the order of the bits (either MSBFIRST or LSBFIRST), and "value" is the byte of data to be sent.

tone(pin, frequency):

Generates a square wave of the specified

ARD2 Arduino Compatible Expanded Kit Project Manual –

Part No. ARD2-1015

Brought to you by

Phone: (03) 5334 2513
Fax: (03) 5334 1845
Freecall: 1800 067 674
Email: sales@wiltronics.com.au

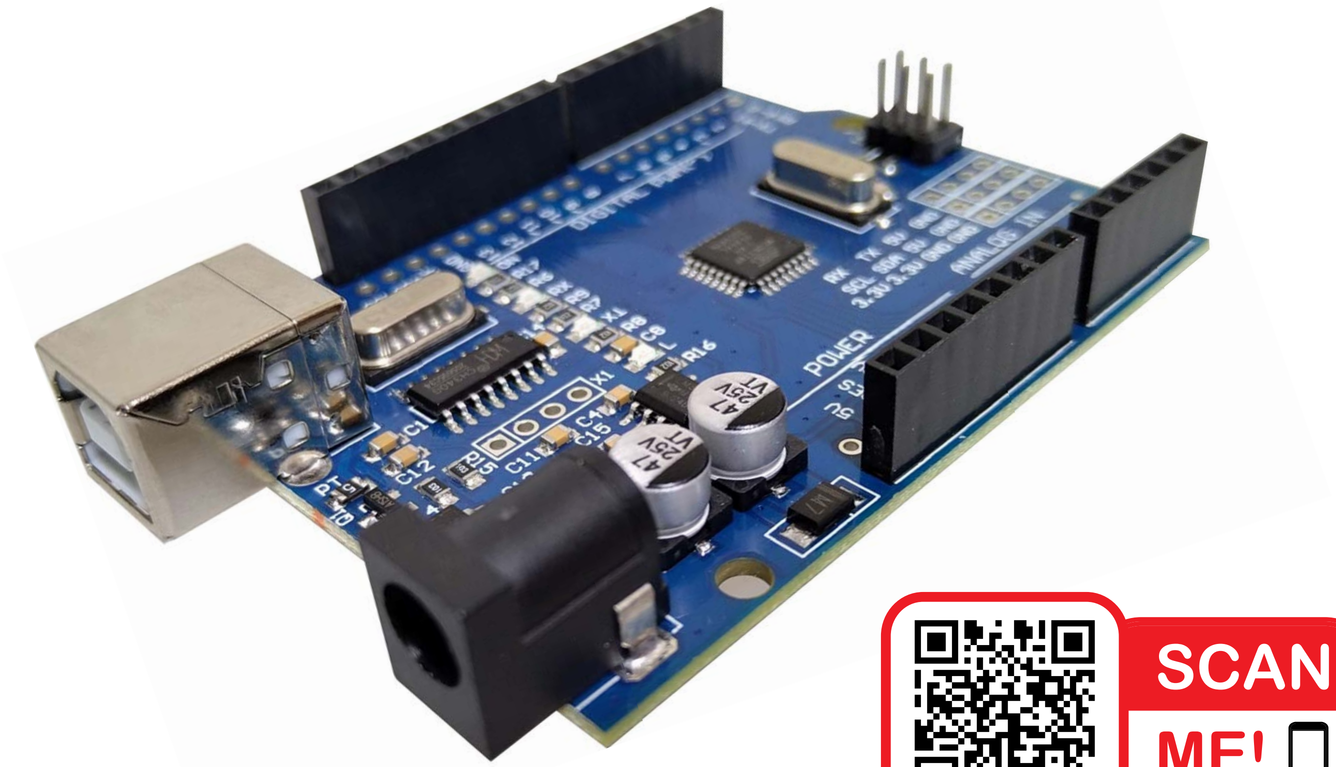
ARD2-1015

ARD2 Arduino Compatible Expanded Kit

ARD 2

Using the

UNO R3 Compatible Controller Board



SCAN ME!

A Step by Step Introduction with 26 Projects

Aligned with the Australian F-10 Curriculum
Design and Technologies (Version 8.4)

WILTRONICS ONLINE