

---

# **SunFounder Ultimate Sensor Kit**

**[www.sunfounder.com](http://www.sunfounder.com)**

**Apr 01, 2024**





# CONTENTS

<b>1</b>	<b>About the display language</b>	<b>3</b>
<b>2</b>	<b>Table of contents</b>	<b>5</b>
2.1	Get Started with Arduino . . . . .	5
2.2	Arduino Video Course . . . . .	29
2.3	Download the Code . . . . .	31
2.4	Component Basics . . . . .	32
2.5	IoT Projects . . . . .	127
2.6	Fun Projects . . . . .	322
2.7	FAQ . . . . .	351
2.8	Thank You . . . . .	363
<b>3</b>	<b>Copyright Notice</b>	<b>365</b>



Thanks for choosing our Ultimate Sensor Kit.

---

**Note:** This document is available in the following languages.

- 
- 
- 

Please click on the respective links to access the document in your preferred language.

---



## SunFounder Ultimate Sensor Kit with Arduino UNO R4 Minima

Ever been disappointed by an online learning kit that came with just a basic guide? Have you dreamed of delving into the world of electronics, but felt overwhelmed by the complexity?

Seen brilliant electronic projects showcased by enthusiasts and wondered how they got started?

Introducing: The Ultimate Sensor Kit with Arduino Uno R4 Minima. The answer to all your creative challenges in one compact package.

Driven by the revolutionary Arduino R4, this kit signifies a new era for open-source enthusiasts. With a powerful 32-bit processor, ample memory, and more, it's designed to seamlessly integrate with both new and existing projects.

Not only does it come with over 30 state-of-the-art modules like the Ultrasonic Sensor, Flame Sensor, Accelerometer & Gyroscope, and even the Pulse Oximeter and Heart Rate Sensor, but each component is paired with a beginner-friendly code example. This ensures you not only possess the tools but also the initial know-how.

Dive deep into the world of IoT with tailored projects that link the Arduino to platforms like Blynk via the ESP8266 WiFi module. Unleash your innovation by designing projects such as the Flame Alert System, Intrusion Alert System, and even a Bluetooth-controlled environmental monitor.

But that's not all; dive into an array of engaging projects! Construct a smart trash can, develop an automatic soap dispenser, or perhaps a motion-triggered relay. The kit expands your horizons, allowing you to translate imagination into tangible creations.

This isn't just about following steps; it's about understanding, experimenting, and inventing. Instead of merely replicating, you'll be crafting projects that are uniquely yours.

Why just be a spectator when you can be a creator? Begin your journey into the captivating world of electronics with the Ultimate Sensor Kit. Your adventure starts here!

If you have any questions or other interesting ideas, feel free to send an email to [service@sunfounder.com](mailto:service@sunfounder.com).

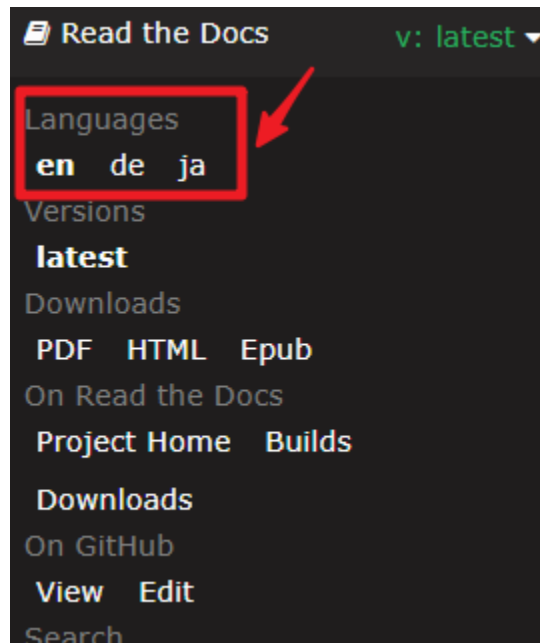
- *About the display language*
- *Table of contents*
- *Copyright Notice*

## ABOUT THE DISPLAY LANGUAGE

**Note:** In addition to English, we are working on other languages for this course. Please contact [service@sunfounder.com](mailto:service@sunfounder.com) if you are interested in helping, and we will give you a free product in return.

---

Currently the online tutorial supports English, German and Japanese. Please click the **Read the Docs** icon in the lower left corner of the page to change the display language.





## TABLE OF CONTENTS

### 2.1 Get Started with Arduino

If you have no idea about Arduino. There are several words I would like to show you: electronics, design, programming, and even Maker. Some of you may think these words are quite far away from us, but in fact, they are not far at all. Because Arduino can take us into the world of programming and help us realize the dream of being a Maker. In this session we will learn:

- What is Arduino?
- What can Arduino do?
- How to build an Arduino Project?

#### 2.1.1 What is Arduino?

First of all, I will give you a brief introduction to Arduino.

Arduino is a convenient, flexible, and easy-to-use open-source electronic prototyping platform, including hardware Arduino boards of various models and software Arduino IDE. It is not only suitable for engineers for rapid prototyping, but also artists, designers, hobbyists, while it is almost a must-have tool for modern Makers.

Arduino is quite a large system. It has software, hardware, and a very huge online community of people who have never met each other but are able to work together because of a common hobby. Everyone in the Arduino family is using their wisdom, making with their hands, and sharing one great invention after another. And you can also be a part of it.

#### 2.1.2 What can Arduino do?

Speaking of which, you may have doubts about what Arduino can actually do. Suffice it to say, Arduino will solve all your problems.

Technically speaking, Arduino is a programmable logic controller. It is a development board that can be used to create many exciting and creative electronic creations: such as remote-controlled cars, robotic arms, bionic robots, smart homes, etc.

Arduino boards are straightforward, simple, and powerful, suitable for students, makers and even professional programmers.

To this day, electronics enthusiasts worldwide continue to develop creative electronic creations based on Arduino development boards.

## 2.1.3 How to build an Arduino Project

Follow these steps to learn how to use Arduino from zero!

### Download and Install Arduino IDE 2.0

The Arduino IDE, known as Arduino Integrated Development Environment, provides all the software support needed to complete an Arduino project. It is a programming software specifically designed for Arduino, provided by the Arduino team, that allows us to write programs and upload them to the Arduino board.

The Arduino IDE 2.0 is an open-source project. It is a big step from its sturdy predecessor, Arduino IDE 1.x, and comes with revamped UI, improved board & library manager, debugger, autocomplete feature and much more.

In this tutorial, we will show how to download and install the Arduino IDE 2.0 on your Windows, Mac, or Linux computer.

### Requirements

- Windows - Win 10 and newer, 64 bits
- Linux - 64 bits
- Mac OS X - Version 10.14: “Mojave” or newer, 64 bits

### Download the Arduino IDE 2.0

1. Visit .
2. Download the IDE for your OS version.



## Arduino IDE 2.0.0

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)

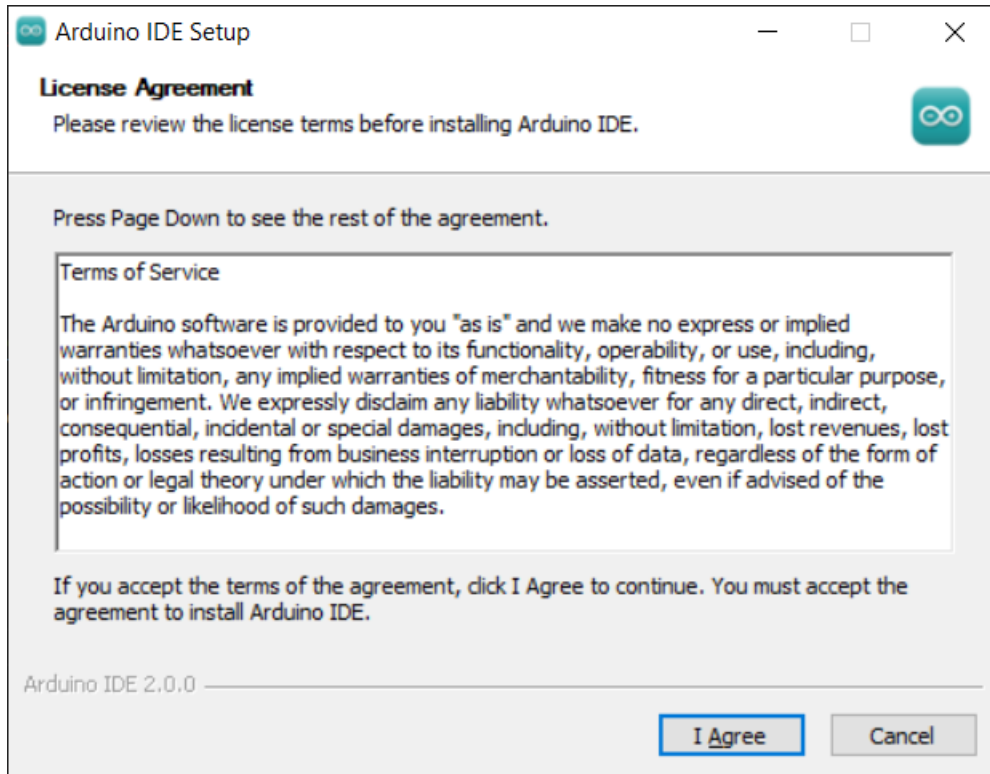
**macOS** 10.14: “Mojave” or newer, 64 bits



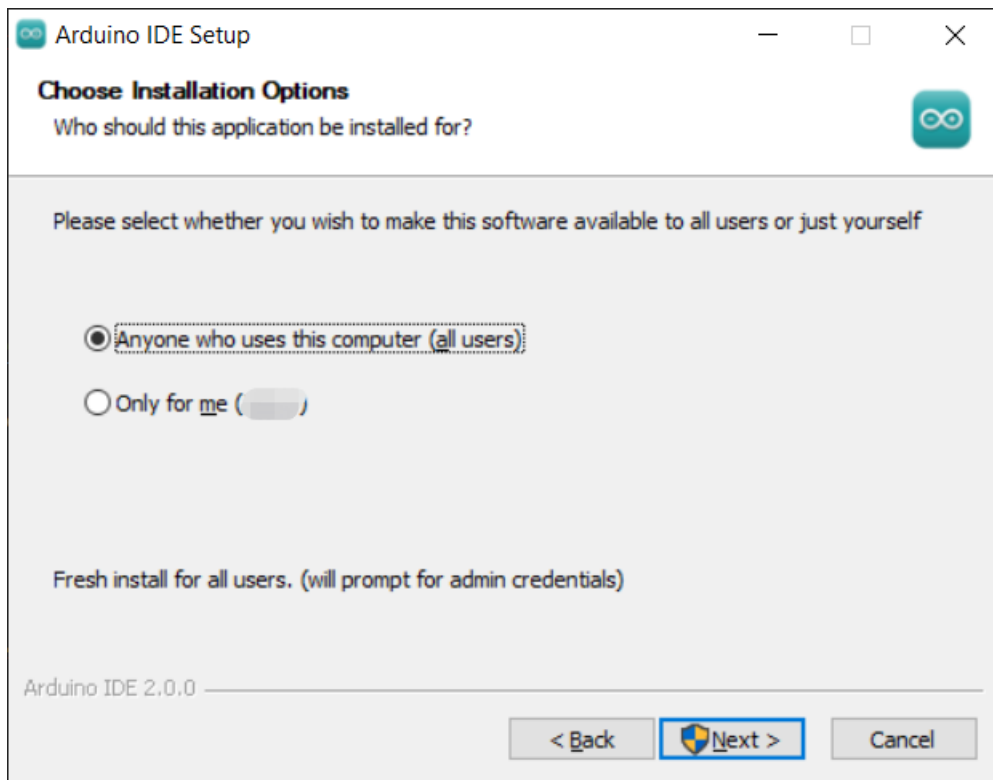
## Installation

### Windows

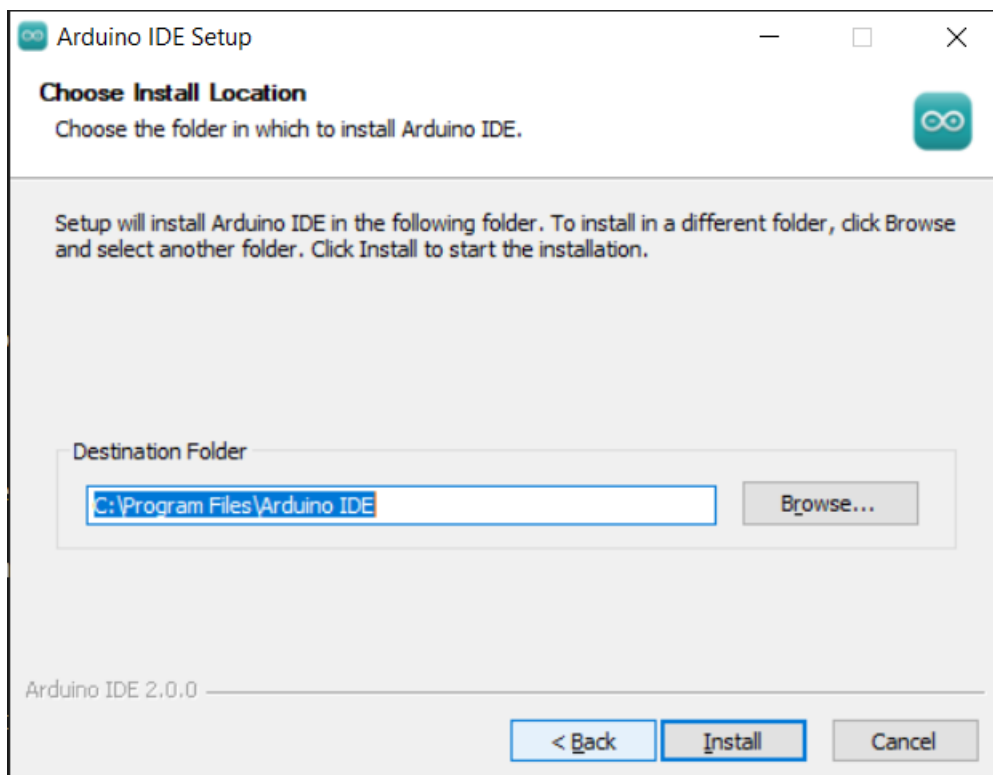
1. Double click the `arduino-ide_xxxx.exe` file to run the downloaded file.
2. Read the License Agreement and agree it.



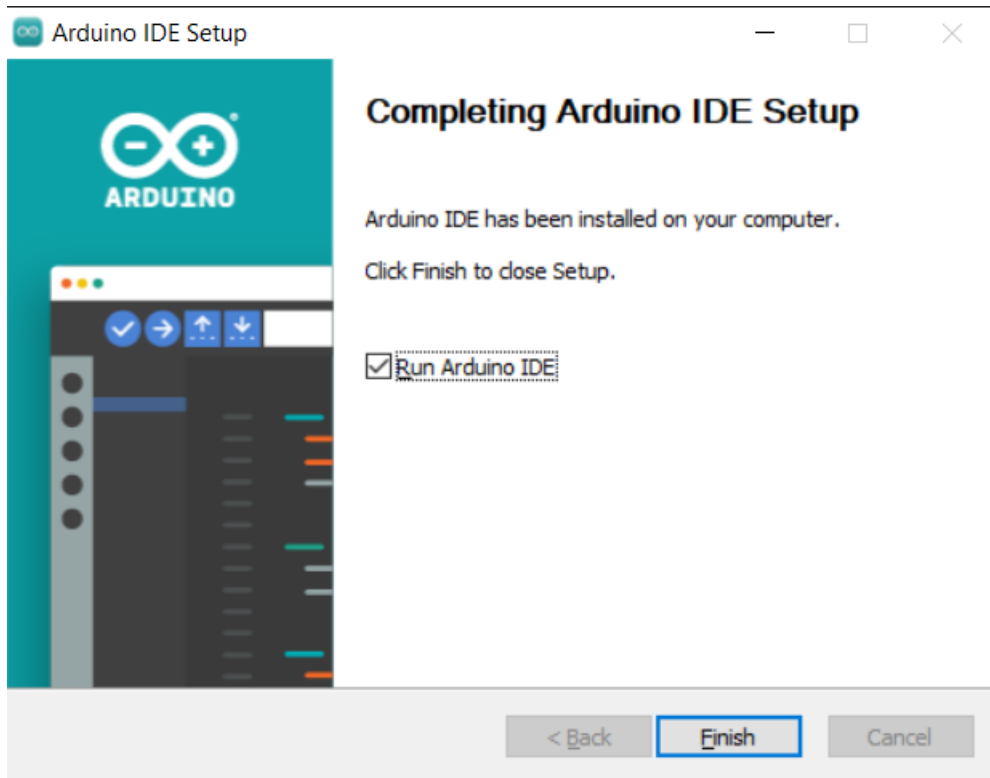
3. Choose installation options.



4. Choose install location. It is recommended that the software be installed on a drive other than the system drive.

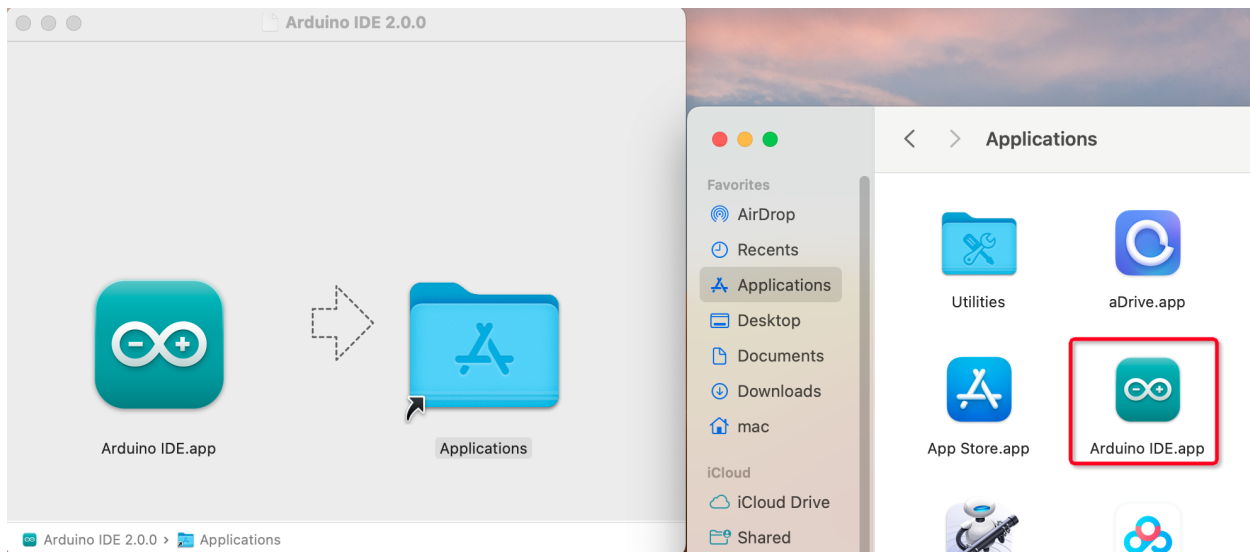


5. Then Finish.



## macOS

Double click on the downloaded `arduino_ide_xxxx.dmg` file and follow the instructions to copy the **Arduino IDE.app** to the **Applications** folder, you will see the Arduino IDE installed successfully after a few seconds.

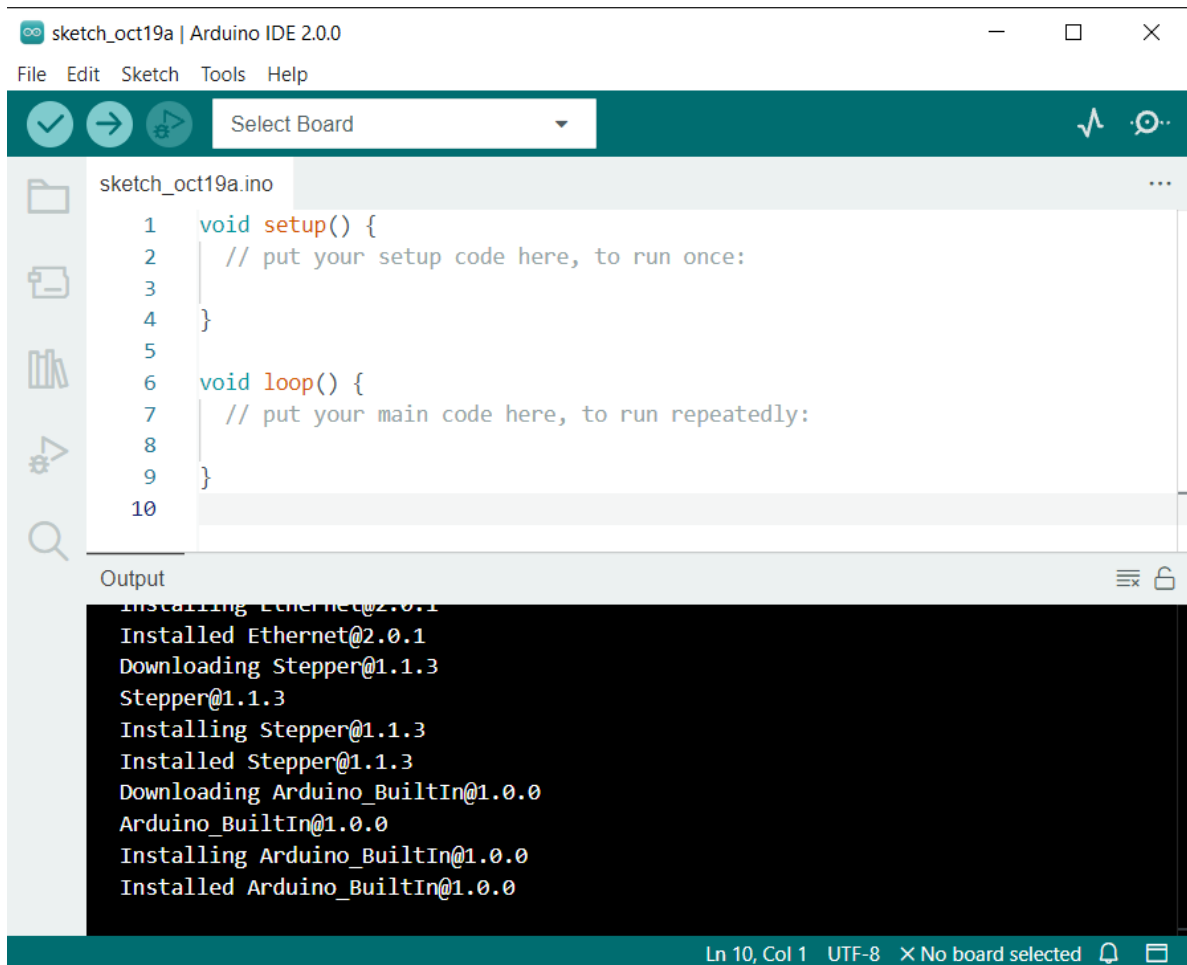


### Linux

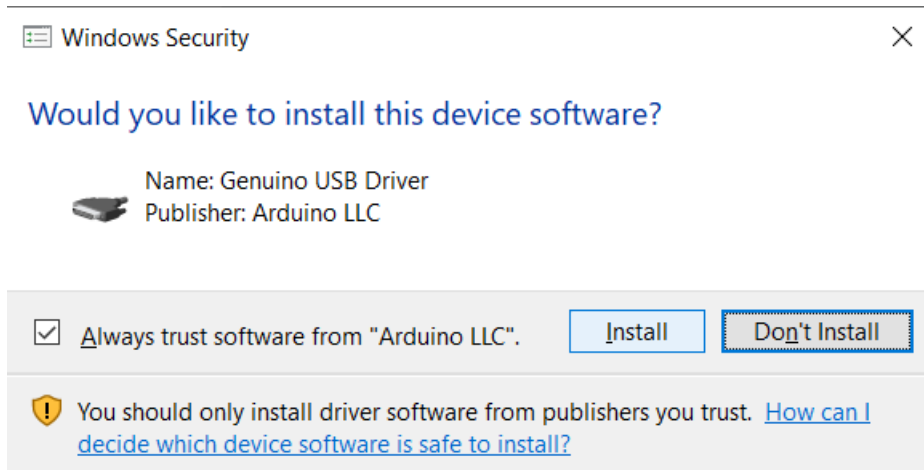
For the tutorial on installing the Arduino IDE 2.0 on a Linux system, please refer

### Open the IDE

1. When you first open Arduino IDE 2.0, it automatically installs the Arduino AVR Boards, built-in libraries, and other required files.



2. In addition, your firewall or security center may pop up a few times asking you if you want to install some device driver. Please install all of them.



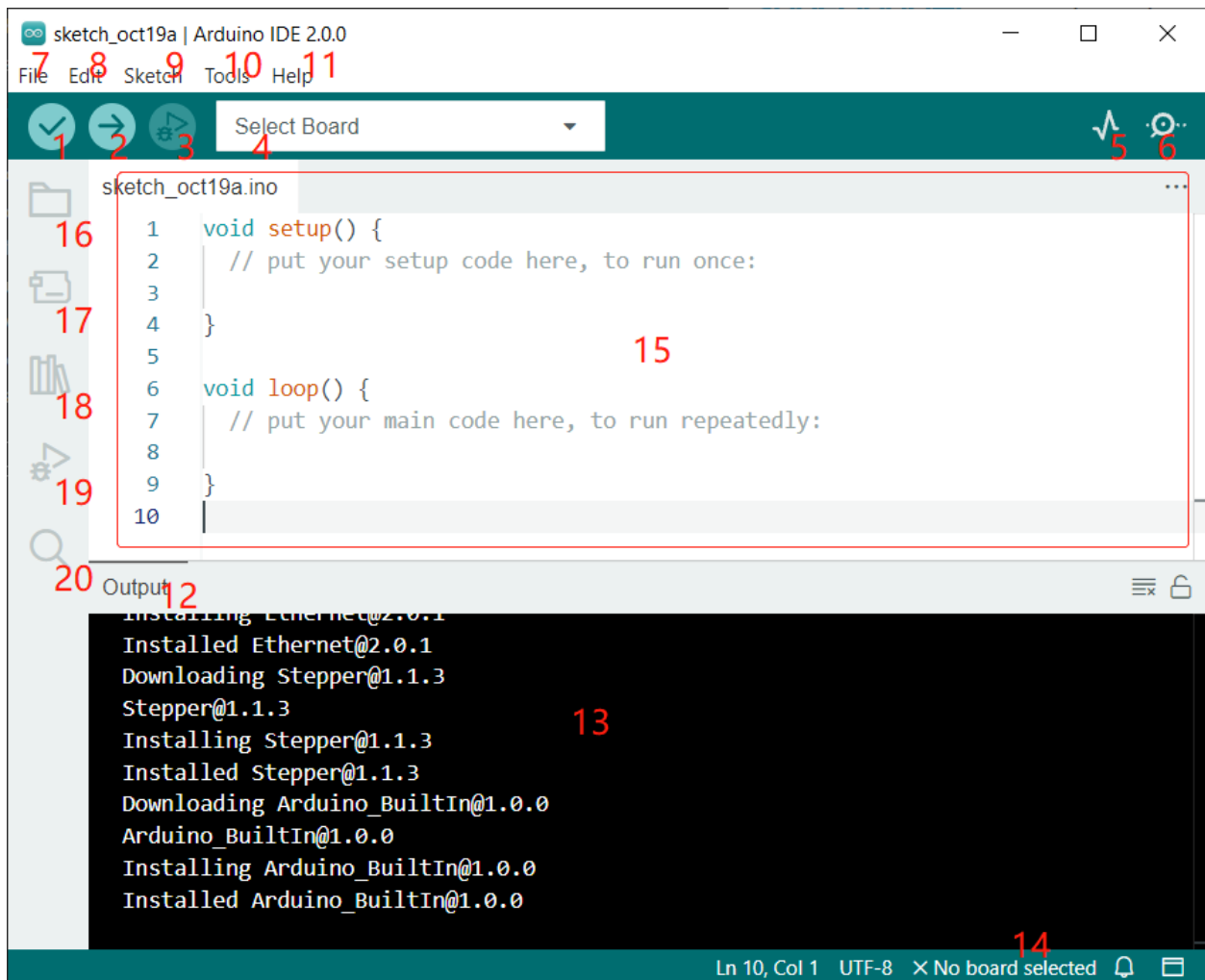
3. Now your Arduino IDE is ready!

---

**Note:** In the event that some installations didn't work due to network issues or other reasons, you can reopen the Arduino IDE and it will finish the rest of the installation. The Output window will not automatically open after all installations are complete unless you click Verify or Upload.

---

## Introduce of Arduino IDE

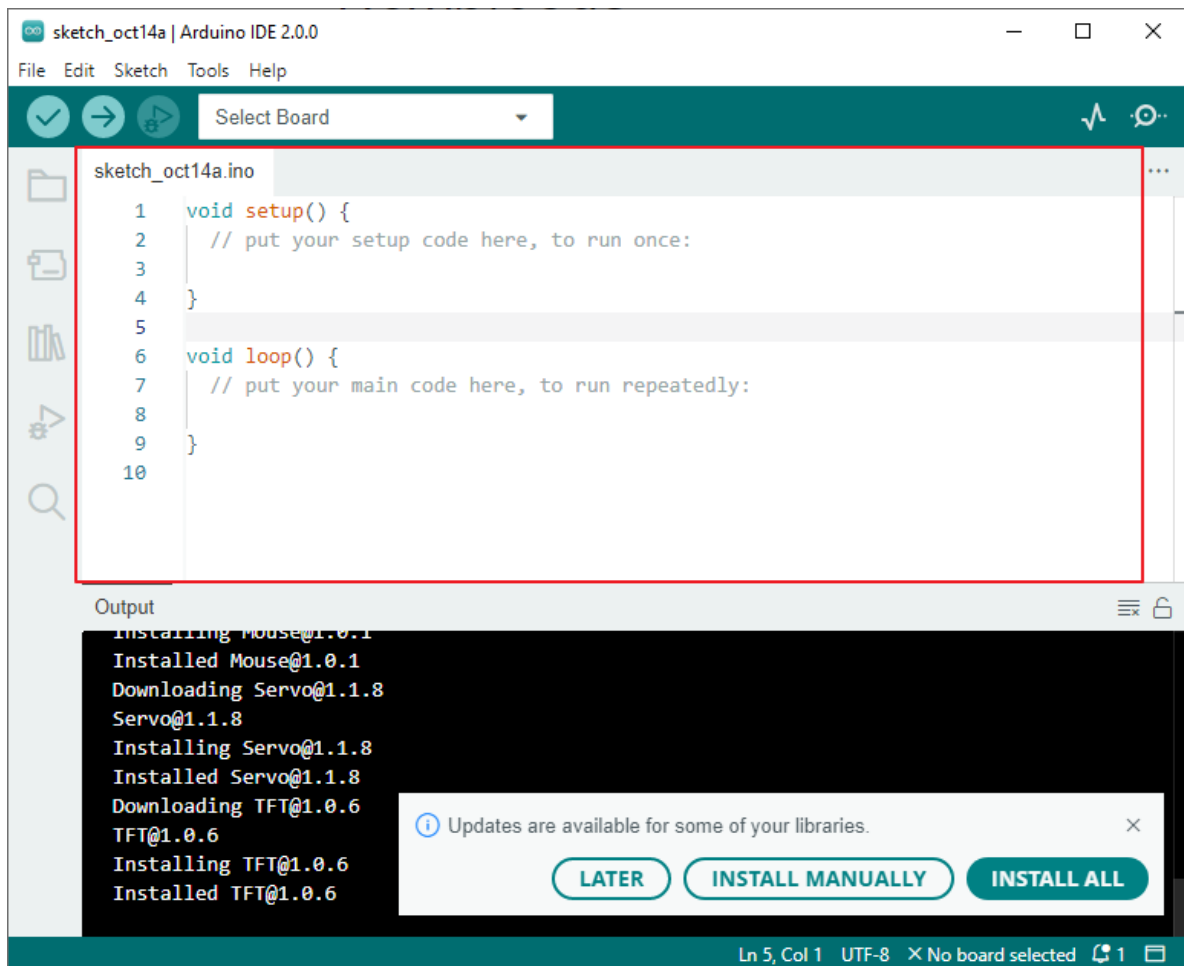


1. **Verify:** Compile your code. Any syntax problem will be prompted with errors.
2. **Upload:** Upload the code to your board. When you click the button, the RX and TX LEDs on the board will flicker fast and won't stop until the upload is done.
3. **Debug:** For line-by-line error checking.
4. **Select Board:** Quick setup board and port.
5. **Serial Plotter:** Check the change of reading value.
6. **Serial Monitor:** Click the button and a window will appear. It receives the data sent from your control board. It is very useful for debugging.
7. **File:** Click the menu and a drop-down list will appear, including file creating, opening, saving, closing, some parameter configuring, etc.
8. **Edit:** Click the menu. On the drop-down list, there are some editing operations like **Cut**, **Copy**, **Paste**, **Find**, and so on, with their corresponding shortcuts.
9. **Sketch:** Includes operations like **Verify**, **Upload**, **Add** files, etc. A more important function is **Include Library** – where you can add libraries.

10. **Tool:** Includes some tools – the most frequently used Board (the board you use) and Port (the port your board is at). Every time you want to upload the code, you need to select or check them.
11. **Help:** If you're a beginner, you may check the options under the menu and get the help you need, including operations in IDE, introduction information, troubleshooting, code explanation, etc.
12. **Output Bar:** Switch the output tab here.
13. **Output Window:** Print information.
14. **Board and Port:** Here you can preview the board and port selected for code upload. You can select them again by **Tools** -> **Board** / **Port** if any is incorrect.
15. The editing area of the IDE. You can write code here.
16. **Sketchbook:** For managing sketch files.
17. **Board Manager:** For managing board driver.
18. **Library Manager:** For managing your library files.
19. **Debug:** Help debugging code.
20. **Search:** Search the codes from your sketches.

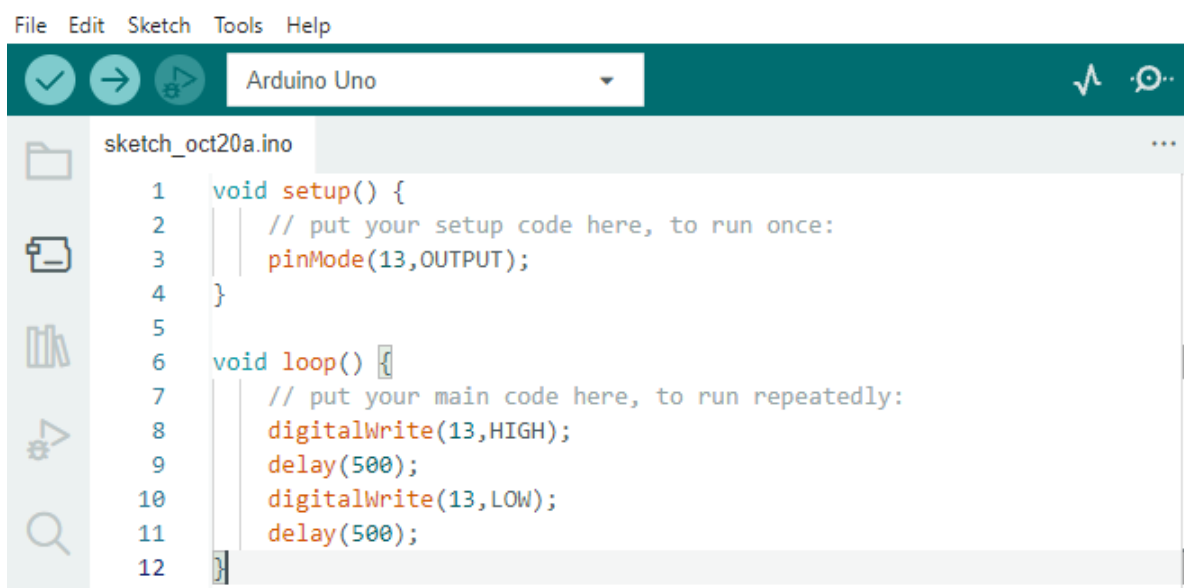
### **How to create, open or Save the Sketch?**

1. When you open the Arduino IDE for the first time or create a new sketch, you will see a page like this, where the Arduino IDE creates a new file for you, which is called a “sketch”.



These sketch files have a regular temporary name, from which you can tell the date the file was created. sketch\_oct14a.ino means October 14th first sketch, .ino is the file format of this sketch.

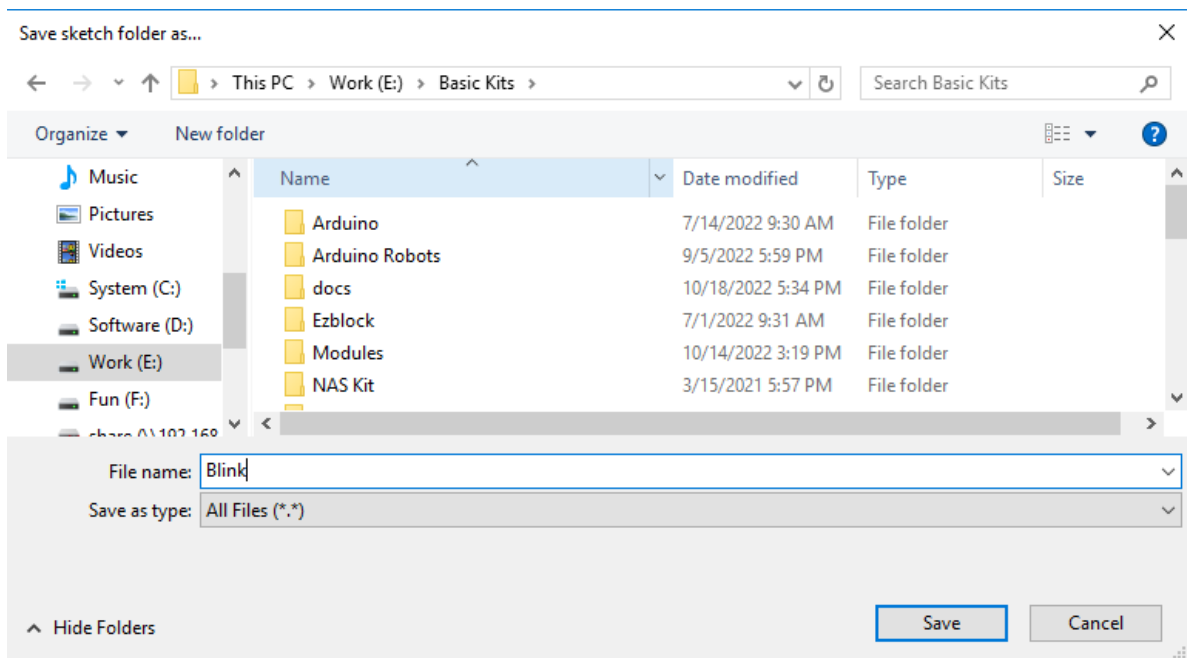
2. Now let's try to create a new sketch. Copy the following code into the Arduino IDE to replace the original code.



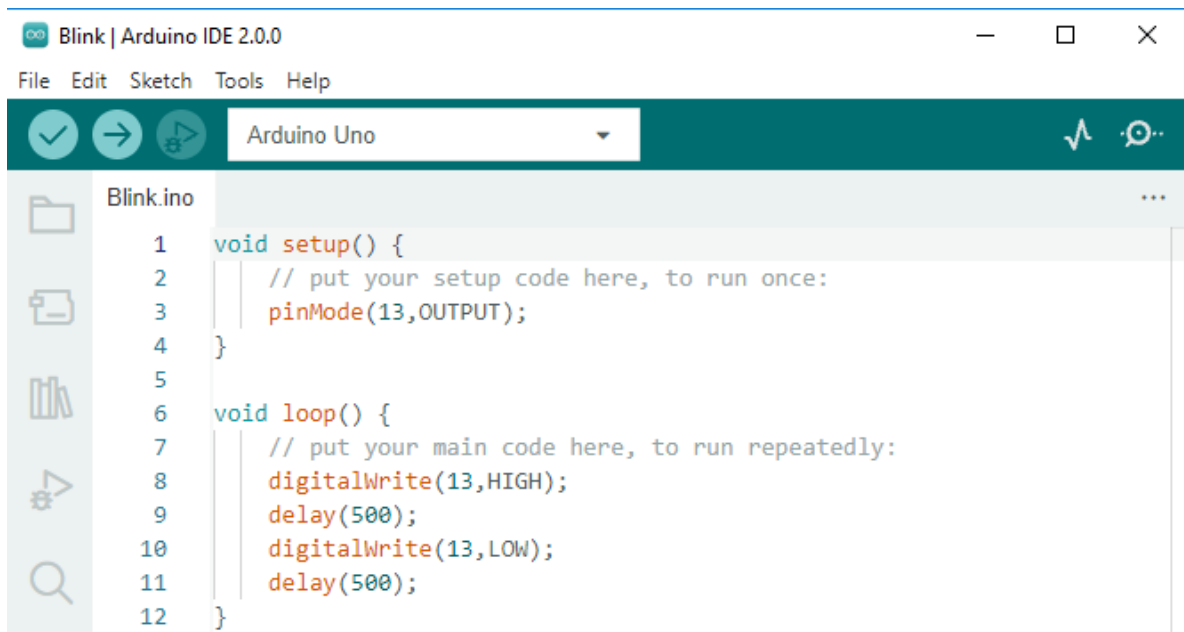


```
void setup() {  
  // put your setup code here, to run once:  
  pinMode(13,OUTPUT);  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  digitalWrite(13,HIGH);  
  delay(500);  
  digitalWrite(13,LOW);  
  delay(500);  
}
```

3. Press Ctrl+S or click **File -> Save**. The Sketch is saved in: C:\Users\{your\_user}\Documents\Arduino by default, you can rename it or find a new path to save it.



4. After successful saving, you will see that the name in the Arduino IDE has been updated.



Please continue with the next section to learn how to upload this created sketch to your Arduino board.

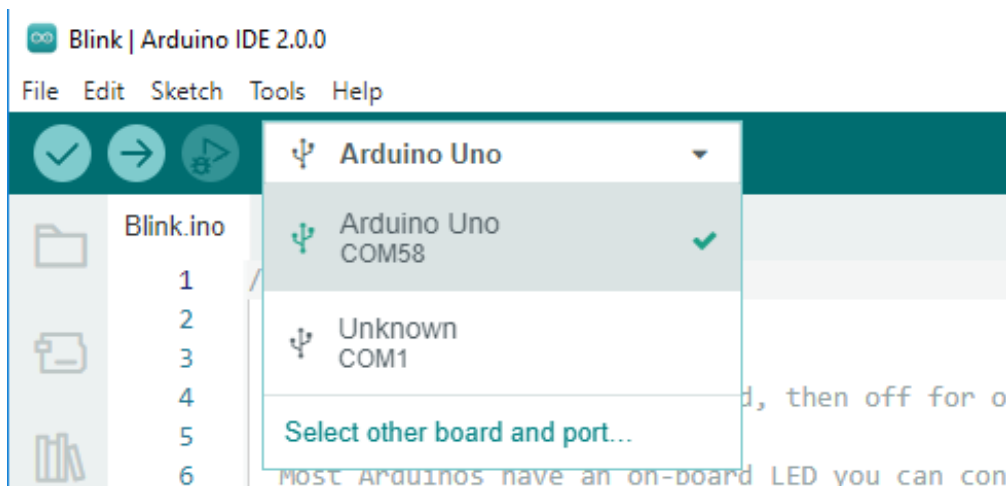
## How to upload Sketch to the Board?

In this section, you will learn how to upload the sketch created previously to the Arduino board, as well as learn about some considerations.

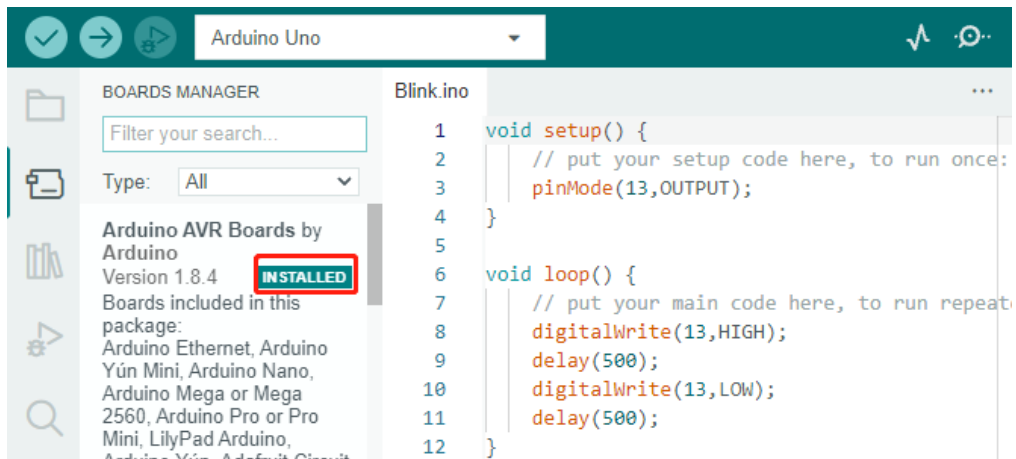
### 1. Choose Board and port

Arduino development boards usually come with a USB cable. You can use it to connect the board to your computer.

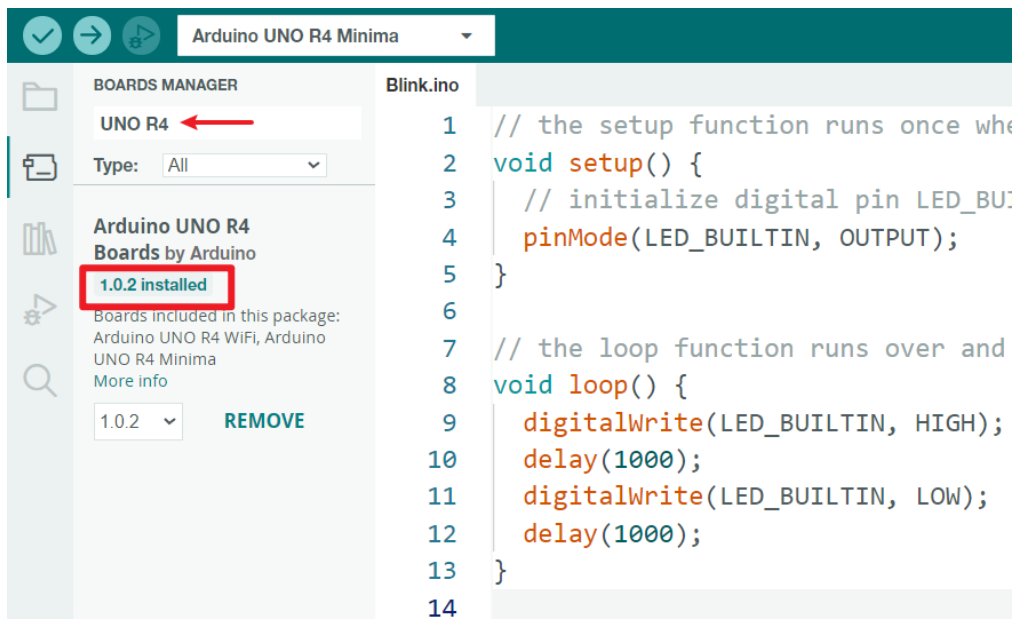
Select the correct **Board** and **Port** in the Arduino IDE. Normally, Arduino boards are recognized automatically by the computer and assigned a port, so you can select it here.



If your board is already plugged in, but not recognized, check if the **INSTALLED** logo appears in the **Arduino AVR Boards** section of the **Boards Manager**, if not, please scroll down a bit and click on **INSTALL**.



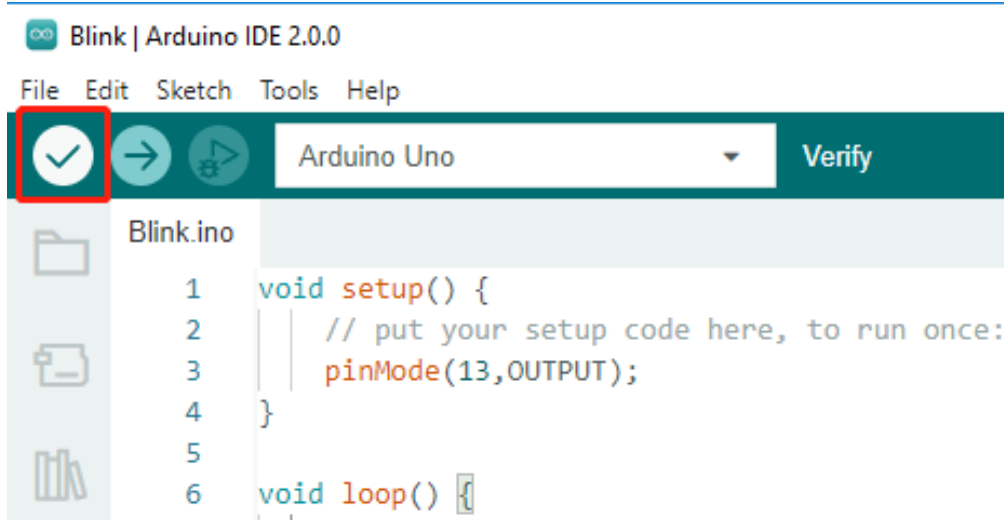
Specifically, for UNO R4, search “UNO R4” in **Boards Manager** and check if the corresponding library is installed.



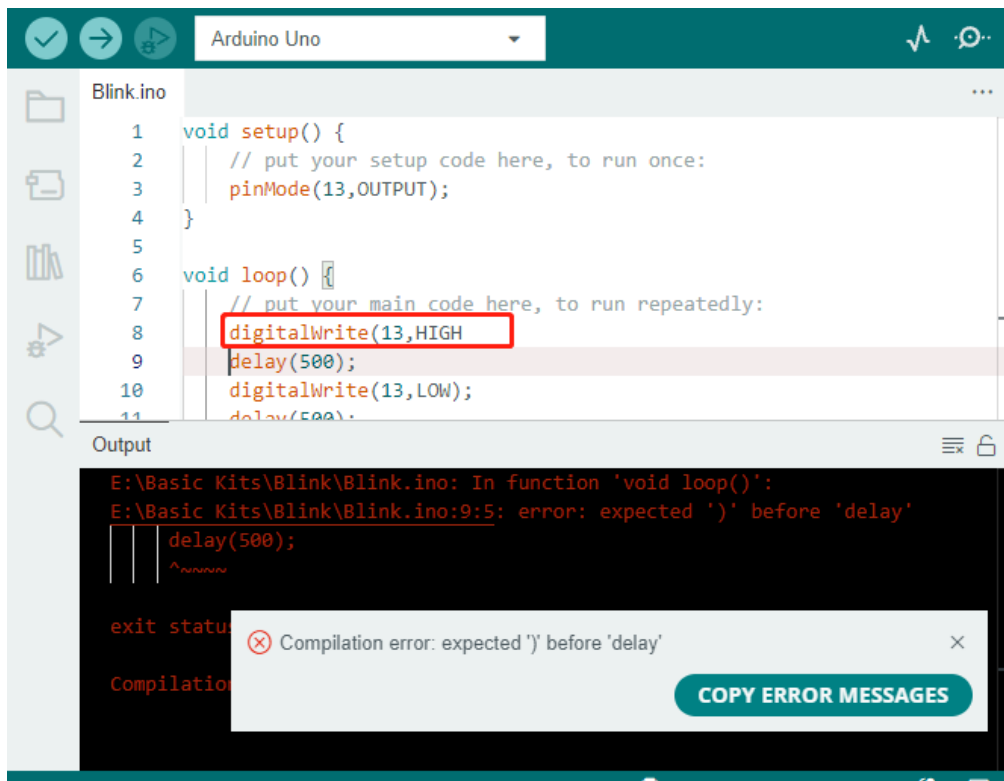
Reopening the Arduino IDE and re-plugging the Arduino board will fix most of the problems. You can also click **Tools** -> **Board** or **Port** to select them.

## 2. Verify the Sketch

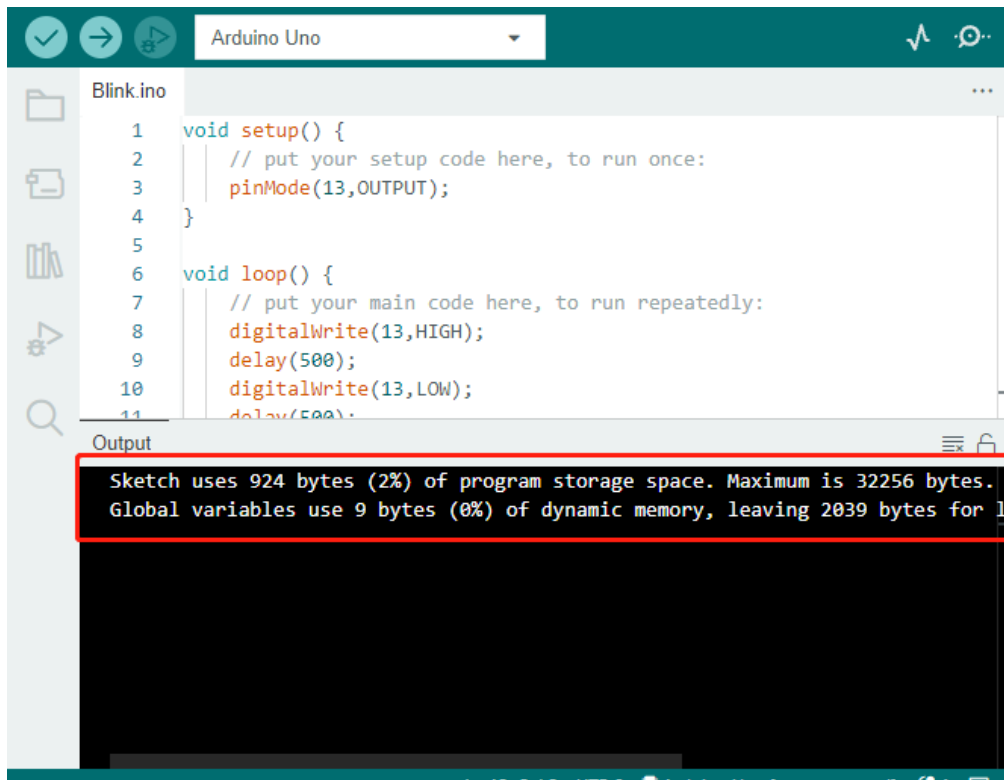
After clicking the Verify button, the sketch will be compiled to see if there are any errors.



You can use it to find mistakes if you delete some characters or type a few letters by mistake. From the message bar, you can see where and what type of errors occurred.

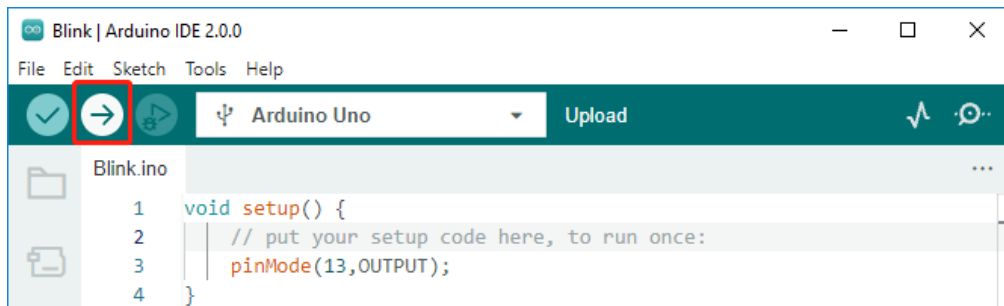


If there are no errors, you will see a message like the one below.

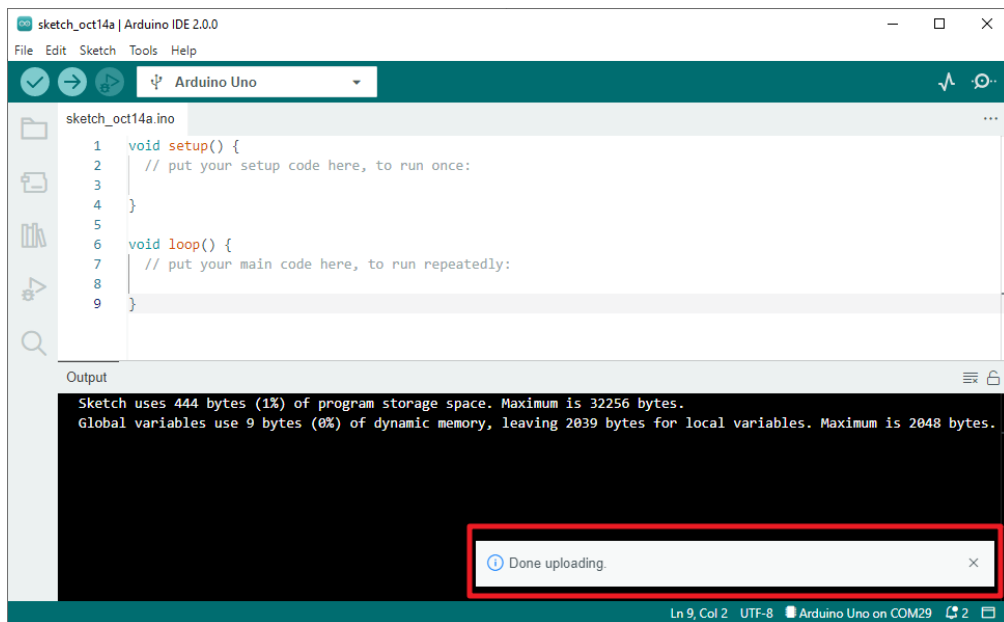


### 3. Upload sketch

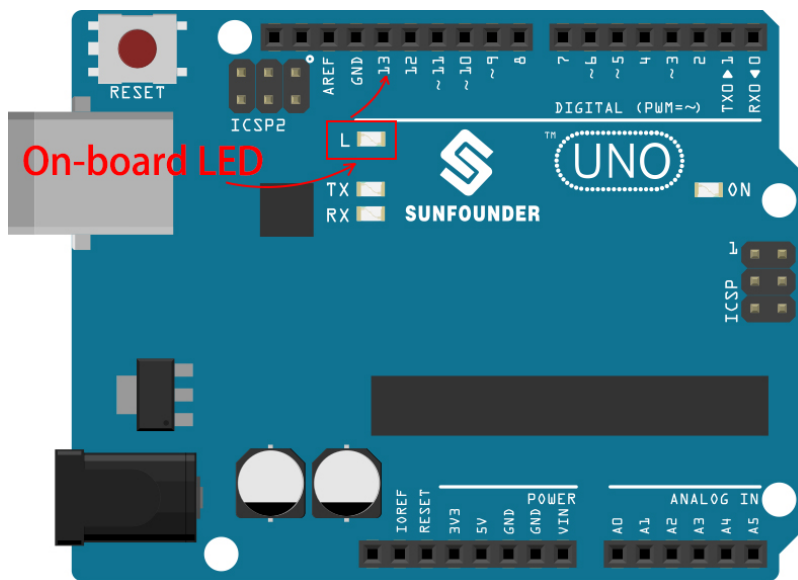
After completing the above steps, click the **Upload** button to upload this sketch to the board.



If successful, you will be able to see the following prompt.



At the same time, the on-board LED blink.



The Arduino board will automatically run the sketch after power is applied after the sketch is uploaded. The running program can be overwritten by uploading a new sketch.

### Arduino Program Structure

Let's take a look at the new sketch file. Although it has a few lines of code itself, it is actually an “empty” sketch. Uploading this sketch to the development board will cause nothing to happen.

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

(continues on next page)

(continued from previous page)

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

If we remove `setup()` and `loop()` and make the sketch a real blank file, you will find that it does not pass the verification. They are the equivalent of the human skeleton, and they are indispensable.

During sketching, `setup()` is run first, and the code inside it (inside `{}`) is run after the board is powered up or reset and only once. `loop()` is used to write the main feature, and the code inside it will run in a loop after `setup()` is executed.

To better understand `setup()` and `loop()`, let's use four sketches. Their purpose is to make the on-board LED of the Arduino blink. Please run each experiment in turn and record their specific effects.

- Sketch 1: Make the on-board LED blink continuously.

```
void setup() {  
  // put your setup code here, to run once:  
  pinMode(13,OUTPUT);  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  digitalWrite(13,HIGH);  
  delay(500);  
  digitalWrite(13,LOW);  
  delay(500);  
}
```

- Sketch 2: Make the on-board LED blink only once.

```
void setup() {  
  // put your setup code here, to run once:  
  pinMode(13,OUTPUT);  
  digitalWrite(13,HIGH);  
  delay(500);  
  digitalWrite(13,LOW);  
  delay(500);  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
}
```

- Sketch 3: Make the on-board LED blink slowly once and then blink quickly.

```
void setup() {  
  // put your setup code here, to run once:  
  pinMode(13,OUTPUT);  
  digitalWrite(13,HIGH);  
  delay(1000);  
  digitalWrite(13,LOW);  
}
```

(continues on next page)

(continued from previous page)

```
    delay(1000);
}

void loop() {
    // put your main code here, to run repeatedly:
    digitalWrite(13,HIGH);
    delay(200);
    digitalWrite(13,LOW);
    delay(200);
}
```

- Sketch 4: Report an error.

```
void setup() {
    // put your setup code here, to run once:
    pinMode(13,OUTPUT);
}

digitalWrite(13,HIGH);
delay(1000);
digitalWrite(13,LOW);
delay(1000);

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

With the help of these sketches, we can summarize several features of `setup-loop`.

- `loop()` will be run repeatedly after the board is powered up.
- `setup()` will run only once after the board is powered up.
- After the board is powered up, `setup()` will run first, followed by `loop()`.
- The code needs to be written within the `{}` scope of `setup()` or `loop()`, out of the framework will be an error.

---

**Note:** Statements such as `digitalWrite(13,HIGH)` are used to control the on-board LED, and we will talk about their usage in detail in later chapters.

---

## Sketch Writing Rule

If you ask a friend to turn on the lights for you, you can say “Turn on the lights.”, or “Lights on, bro.”, you can use any tone of voice you want.

However, if you want the Arduino board to do something for you, you need to follow the Arduino program writing rules to type in the commands.

This chapter contains the basic rules of the Arduino language and will help you understand how to translate natural language into code.

Of course, this is a process that takes time to get familiar with, and it is also the most error-prone part of the process for newbies, so if you make mistakes often, it’s okay, just try a few more times.



## Semicolon ;

Just like writing a letter, where you write a period at the end of each sentence as the end, the Arduino language requires you to use ; to tell the board the end of the command.

Take the familiar “onboard LED blinking” example. A healthy sketch should look like this.

Example:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(13,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(13,HIGH);
  delay(500);
  digitalWrite(13,LOW);
  delay(500);
}
```

Next, let’s take a look at the following two sketches and guess if they can be correctly recognized by Arduino before running them.

Sketch A:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(13,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(13,HIGH)
  delay(500)
  digitalWrite(13,LOW)
  delay(500)
}
```

Sketch B:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(13,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(13,
HIGH); delay
(500
);
  digitalWrite(13,
```

(continues on next page)

(continued from previous page)

```
LOW);
    delay(500)
;
}
```

The result is that **Sketch A** reports an error and **Sketch B** runs.

- The errors in **Sketch A** are missing ; and although it looks normal, the Arduino can't read it.
- **Sketch B**, looks anti-human, but in fact, indentation, line breaks and spaces in statements are things that do not exist in Arduino programs, so to the Arduino compiler, it looks the same as in the example.

However, please don't write your code as **Sketch B**, because it is usually natural people who write and view the code, so don't get yourself into trouble.

## Curlybraces {}

{ } is the main component of the Arduino programming language, and they must appear in pairs. A better programming convention is to insert a structure that requires curly braces by typing the right curly brace directly after typing the left curly brace, and then moving the cursor between the curly braces to insert the statement.

## Comment //

Comment is the part of the sketch that the compiler ignores. They are usually used to tell others how the program works.

If we write two adjacent slashes in a line of code, the compiler will ignore anything up to the end of the line.

If we create a new sketch, it comes with two comments, and if we remove these two comments, the sketch will not be affected in any way.

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

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

Comment is very useful in programming, and several common uses are listed below.

- Usage A: Tell yourself or others what this section of code does.

```
void setup() {
    pinMode(13,OUTPUT); //Set pin 13 to output mode, it controls the onboard LED
}

void loop() {
    digitalWrite(13,HIGH); // Activate the onboard LED by setting pin 13 high
    delay(500); // Status quo for 500 ms
    digitalWrite(13,LOW); // Turn off the onboard LED
}
```

(continues on next page)

(continued from previous page)

```

    delay(500); // Status quo for 500 ms
}

```

- Usage B: Temporarily invalidate some statements (without deleting them) and uncomment them when you need to use them, so you don't have to rewrite them. This is very useful when debugging code and trying to locate program errors.

```

void setup() {
    pinMode(13,OUTPUT);
    // digitalWrite(13,HIGH);
    // delay(1000);
    // digitalWrite(13,LOW);
    // delay(1000);
}

void loop() {
    digitalWrite(13,HIGH);
    delay(200);
    digitalWrite(13,LOW);
    delay(200);
}

```

**Note:** Use the shortcut Ctrl+/ to help you quickly comment or uncomment your code.

### Comment /\*\*/

Same as // for comments. This type of comment can be more than one line long, and once the compiler reads /\*, it ignores anything that follows until it encounters \*/.

Example 1:

```

/* Blink */

void setup() {
    pinMode(13,OUTPUT);
}

void loop() {
    /*
    The following code will blink the onboard LED
    You can modify the number in delay() to change the blinking frequency
    */
    digitalWrite(13,HIGH);
    delay(500);
    digitalWrite(13,LOW);
    delay(500);
}

```

### #define

This is a useful C++ tool.

```
#define identifier token-string
```

The compiler automatically replaces `identifier` with `token-string` when it reads it, which is usually used for constant definitions.

As an example, here is a sketch that uses `define`, which improves the readability of the code.

```
#define ONBOARD_LED 13
#define DELAY_TIME 500

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

void loop() {
    digitalWrite(ONBOARD_LED, HIGH);
    delay(DELAY_TIME);
    digitalWrite(ONBOARD_LED, LOW);
    delay(DELAY_TIME);
}
```

To the compiler, it actually looks like this.

```
void setup() {
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);
    delay(500);
    digitalWrite(13, LOW);
    delay(500);
}
```

We can see that the `identifier` is replaced and does not exist inside the program. Therefore, there are several caveats when using it.

1. A `token-string` can only be modified manually and cannot be converted into other values by arithmetic in the program.
2. Avoid using symbols such as `;`. For example.

```
#define ONBOARD_LED 13;

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

void loop() {
    digitalWrite(ONBOARD_LED, HIGH);
}
```

The compiler will recognize it as the following, which is what will be reported as an error.

```
void setup() {  
    pinMode(13,OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13,HIGH);  
}
```

**Note:** A naming convention for `#define` is to capitalize `identifier` to avoid confusion with variables.

## Variable

The variable is one of the most powerful and critical tools in a program. It helps us to store and call data in our programs.

The following sketch file uses variables. It stores the pin numbers of the on-board LED in the variable `ledPin` and a number “500” in the variable `delayTime`.

```
int ledPin = 13;  
int delayTime = 500;  
  
void setup() {  
    pinMode(ledPin,OUTPUT);  
}  
  
void loop() {  
    digitalWrite(ledPin,HIGH);  
    delay(delayTime);  
    digitalWrite(ledPin,LOW);  
    delay(delayTime);  
}
```

Wait, is this a duplicate of what `#define` does? The answer is NO.

- The role of `#define` is to simply and directly replace text, it is not considered by the compiler as part of the program.
- A **variable**, on the other hand, exists within the program and is used to store and call value. A variable can also modify its value within the program, something that a `define` cannot do.

The sketch file below self-adds to the variable and it will cause the on-board LED to blink longer after each blink.

```
int ledPin = 13;  
int delayTime = 500;  
  
void setup() {  
    pinMode(ledPin,OUTPUT);  
}  
  
void loop() {  
    digitalWrite(ledPin,HIGH);  
    delay(delayTime);
```

(continues on next page)

(continued from previous page)

```
digitalWrite(ledPin,LOW);  
delay(delayTime);  
delayTime = delayTime+200; //Each execution increments the value by 200  
}
```

## Declare a variable

Declaring a variable means creating a variable.

To declare a variable, you need two things: the data type, and the variable name. The data type needs to be separated from the variable by a space, and the variable declaration needs to be terminated by a ;.

Let's use this variable as an example.

```
int delayTime;
```

## Data Type

Here `int` is a data type called integer type, which can be used to store integers from -32768 to 32766. It can also not be used to store decimals.

Variables can hold different kinds of data other than integers. The Arduino language (which, remember, is C++) has built-in support for a few of them (only the most frequently used and useful are listed here):

- `float`: Store a decimal number, for example 3.1415926.
- `byte`: Can hold numbers from 0 to 255.
- `boolean`: Holds only two possible values, `True` or `False`, even though it occupies a byte in memory.
- `char`: Holds a number from -127 to 127. Because it is marked as a `char` the compiler will try to match it to a character from the .
- `string`: Can stores a string of characters, e.g. Halloween.

## Variable Name

You can set the variable to any name you want, such as `i`, `apple`, `Bruce`, `R2D2`, `Sectumsempra`, but there are some basic rules to follow.

1. describe what it is used for. Here, I named the variable `delayTime`, so you can easily understand what it does. It works fine if I name the variable `barryAllen`, but it confuses the person looking at the code.
2. Use regular nomenclature. You can use CamelCase like I did, with the initial T in `delayTime` so that it is easy to see that the variable consists of two words. Also, you can use UnderScoreCase to write the variable as `delay_time`. It doesn't affect the program's running, but it would help the programmer to read the code if you use the nomenclature you prefer.
3. Don't use keywords. Similar to what happens when we type "`int`", the Arduino IDE will color it to remind you that it is a word with a special purpose and cannot be used as a variable name. Change the name of the variable if it is colored.
4. Special symbols are not allowed. For example, space, #, \$, /, +, %, etc. The combination of English letters (case sensitive), underscores, and numbers (but numbers cannot be used as the first character of a variable name) is rich enough.

## Assign a value to a variable

Once we have declared the variable, it is time to store the data. We use the assignment operator (i.e. `=`) to put value into the variable.

We can assign values to the variable as soon as we declare it.

```
int delayTime = 500;
```

It is also possible to assign a new value to it at some time.

```
int delayTime; // no value
delayTime = 500; // value is 500
delayTime = delayTime + 200; // value is 700
```

## 2.2 Arduino Video Course

Embark on a journey through the Arduino world with the comprehensive Arduino Video Course, using SunFounder's Ultimate Sensor Kit. This series begins with an introduction to the Arduino ecosystem and the capabilities of the UNO R4 Minima board, setting the stage for a deep dive into practical applications and programming techniques. You'll learn the basics of controlling LEDs, understanding serial communication, and manipulating various components like RGB LEDs, buttons, and Potentiometer. As long as you follow the course step by step, you will master Arduino, not just copy and paste code, you will write your own code and implement your Arduino projects the way you like.

The is being continuously updated, stay tuned for more!

### Catalogue

#### 2.2.1 Video 1 - Arduino Uno R4 Minima

A step-by-step introduction to Arduino Uno R4 Minima, covering its features, setup, and a basic LED blinking program.

- **Introduction:** Overview of the Arduino Uno R4 Minima training course and its curriculum.
- **Arduino Comparison:** Detailing the improvements and features of the Uno R4 Minima compared to Uno R3.
- **Key Features:** Exploring the advanced features and capabilities of the Uno R4 Minima.
- **Pin Functions:** Comprehensive guide on the various pinouts and their functionalities.
- **IDE Setup:** Instructions on installing and configuring the Arduino IDE for Uno R4 Minima.
- **First Program:** Demonstrating a basic LED blinking program as an introductory project.

### Video

### Related On-line Tutorials

- *Arduino UNO R4 Minima Board*

#### 2.2.2 Video 2 - Introducing the Arduino IDE

Explore the fundamentals of Arduino programming and the new features of Arduino IDE version 2, perfect for beginners venturing into the world of Arduino.

- **Introduction to Arduino:** Aims to bridge the instructional gap for beginners using the Arduino Uno R4.
- **Exploring Arduino IDE:** An in-depth look at the features and tools of Arduino IDE version 2, making it accessible for new users.
- **Programming Basics:** Understanding the structure of Arduino code, including `setup` and `loop` functions.

- **Onboard LEDs Control:** Practical examples of controlling the Arduino's built-in LEDs to demonstrate basic coding techniques.
- **Syntax and Functions:** Emphasizing the importance of syntax in programming and introducing essential functions like `pinMode` and `digitalWrite`.

### Video

#### Related On-line Tutorials

- [\*Get Started with Arduino\*](#)

## 2.2.3 Video 3: Basic Circuits and Breadboarding

A detailed tutorial on building basic circuits and using a breadboard, focused on practical applications with Arduino Uno R4 Minima.

- **Circuit Basics:** Introduction to essential components of an electrical circuit.
- **Ohm's Law:** Explaining the relationship between current, voltage, and resistance.
- **Breadboarding Technique:** Guide on using a breadboard for circuit assembly.
- **Series Circuits:** Understanding voltage division and principles of series connections.
- **Parallel Circuits:** Exploring the characteristics and advantages of parallel circuitry.
- **LED Circuitry:** Practical tips on selecting resistor values and building LED circuits with Arduino.

### Video

#### Related On-line Tutorials

- [\*RGB Module\*](#)

## 2.2.4 Video 4: Variables, Data Types, and Constants

This tutorial dives into the essentials of using variables, data types, constants, and preprocessor directives in Arduino programming, demonstrated through a traffic light simulation.

- **Variables in Arduino:** Demonstrates how to use variables for dynamic programming in Arduino.
- **Data Types:** Explains various data types like `int`, `float`, `double`, `char`, `bool`, and their memory usage.
- **Efficient Memory Usage:** Highlights the significance of selecting appropriate data types for efficient memory management in microcontrollers.
- **const and #define:** Discusses the use of constants and preprocessor directives for creating fixed values in code.
- **Traffic Light Simulation:** Guides through building and programming a traffic light simulation to apply these concepts.

### Video

#### Related On-line Tutorials

- [\*Traffic Light Module\*](#)



## 2.2.5 Video 5: Digital Inputs and Conditional Logic

Discover how to integrate digital inputs, manage floating pins with pull-up/down resistors, and apply conditional statements in Arduino Uno R4 Minima projects, exemplified with a traffic light simulation.

- **Digital Inputs:** Learning to connect push button switches as digital inputs.
- **Pull-Up/Down Resistors:** Understanding the use of resistors to stabilize floating pins in digital inputs.
- **Serial Monitor Tool:** Using the Arduino IDE's serial monitor for textual output from the Arduino board.
- **If-Else Statements:** Implementing if-else logic for decision making in Arduino sketches.
- **Traffic Light Simulation:** Demonstrating the concepts with a traffic light circuit using LEDs and push buttons.
- **digitalRead() Function:** Utilizing this function to read digital inputs and control outputs accordingly.

### Video

#### Related On-line Tutorials

- [\*Button Module\*](#)
- [\*Traffic Light Module\*](#)

## 2.2.6 Video 6: Analog to Digital Conversion

This tutorial delves into analog to digital conversion, elucidating binary numbers, bits, and bytes, showcased through the use of a potentiometer with Arduino Uno R4 Minima.

- **Analog to Digital Conversion:** Demonstrating conversion principles using a potentiometer.
- **Digital Representation:** Illustrating how analog signals are digitized.
- **Programming for Conversion:** Writing Arduino sketches to convert decimal to binary numbers.
- **Bits and Bytes:** Exploring the basics of binary numbers and their importance in digital computing.
- **analogRead() Function:** Using this function to read and convert analog signals.
- **ADC Resolution:** Understanding how resolution affects the precision of analog input measurements.

### Video

#### Related On-line Tutorials

- [\*Potentiometer Module\*](#)
- [\*Potentiometer scale value\*](#)

## 2.3 Download the Code

Download the relevant code from the link below.

- SunFounder Ultimate Sensor Kit Code
- Or check out the code at [SunFounder Ultimate Sensor Kit - GitHub](#)

## 2.4 Component Basics

After opening the package, please check whether the quantity of components is compliance with product description and whether all components are in good condition.

- SunFounder Ultimate Sensor Kit Components List

The following is an introduction to each component, including its working principle and corresponding project. **Each component has a simple code example to help you get started quickly.**

### Control Board

#### 2.4.1 Arduino UNO R4 Minima Board

The **Arduino UNO R4 Minima** is a development board with the classic UNO form factor, based on the RA4M1() microcontroller from Renesas. It is faster and has more memory than the previous versions of the board. It has a number of built-in features such as a DAC, RTC and HID. The UNO R4 Minima is a **5 V only board**. It has 14 digital I/O, 6 analog inputs with up to 14-bit resolution, a clock speed of 48 MHz, and 32 kB SRAM, 256 kB flash memory & 8 kB of EEPROM.

#### Technical Parameters

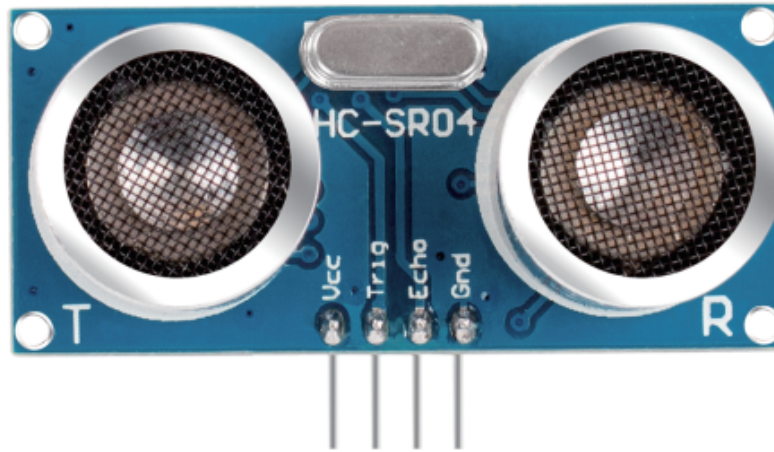
Board	Name	Arduino® UNO R4 Minima
Microcontroller	Renesas RA4M1 (Arm® Cortex®-M4)	
USB	USB-C®	Programming Port
Pins	Digital I/O Pins	14
Pins	Analog input pins	6
	DAC	1
	PWM pins	6
Communication	UART	Yes, 1x
	I2C	Yes, 1x
	SPI	Yes, 1x
	CAN	Yes 1 CAN Bus
Power	Circuit operating voltage	5 V
	Input voltage (VIN)	6-24 V
	DC Current per I/O Pin	8 mA
Clock speed	Main core	48 MHz
Memory	RA4M1	256 kB Flash, 32 kB RAM
Dimensions	Width	68.85 mm
	Length	53.34 mm

#### What's More

- [Arduino IDE](#)
- [Download and Install Arduino IDE 2.0](#)
- [Arduino Programming Language Reference](#)
- 
- 
-

## Sensor

### 2.4.2 Ultrasonic Sensor Module (HC-SR04)



#### Introduction

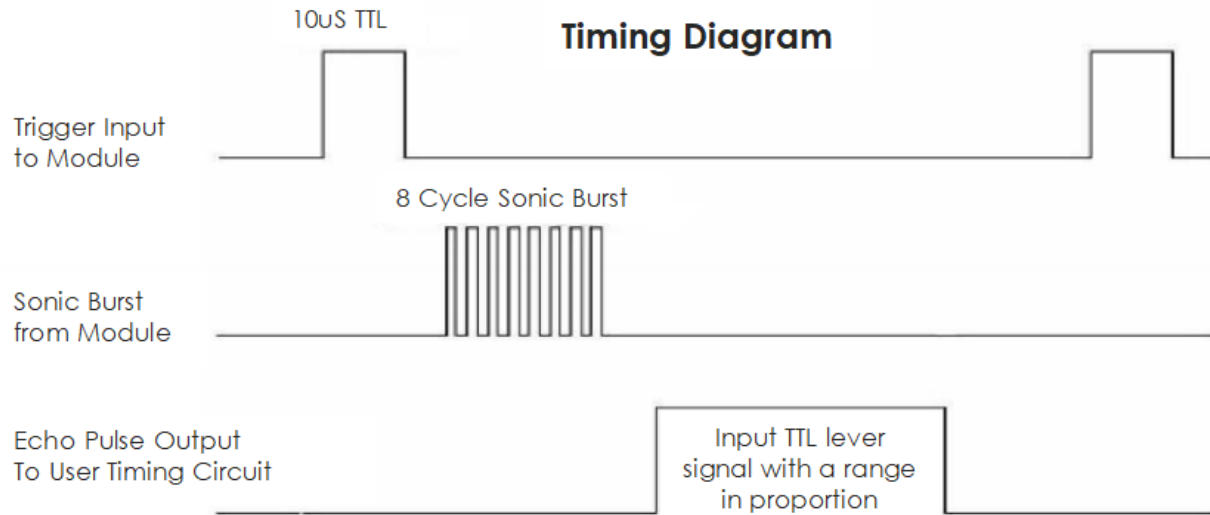
The Ultrasonic Module(HC-SR04) is a sensor that can measure distances between 2cm and 400cm using ultrasonic waves. It is commonly used in robotics and automation projects to detect objects and measure distances. The module consists of an ultrasonic transmitter and receiver, which work together to send and receive ultrasonic waves.

#### Principle

The module includes ultrasonic transmitters, receiver and control circuit. The basic principles are as follows:

1. Use an IO flip-flop to process a high level signal of at least 10us.
2. The module automatically sends eight 40khz and detects if there is a pulse signal return.
3. If the signal returns, passing the high level, the high output IO duration is the time from the transmission of the ultrasonic wave to the return of it. Here, test distance = (high time x sound speed (340 m / s) / 2.

The timing diagram is shown below.



You only need to supply a short 10µs pulse for the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

---

**Note:** It is recommended to use measurement cycle over 60ms in order to prevent signal collisions of trigger signal and the echo signal.

---

### Formula:

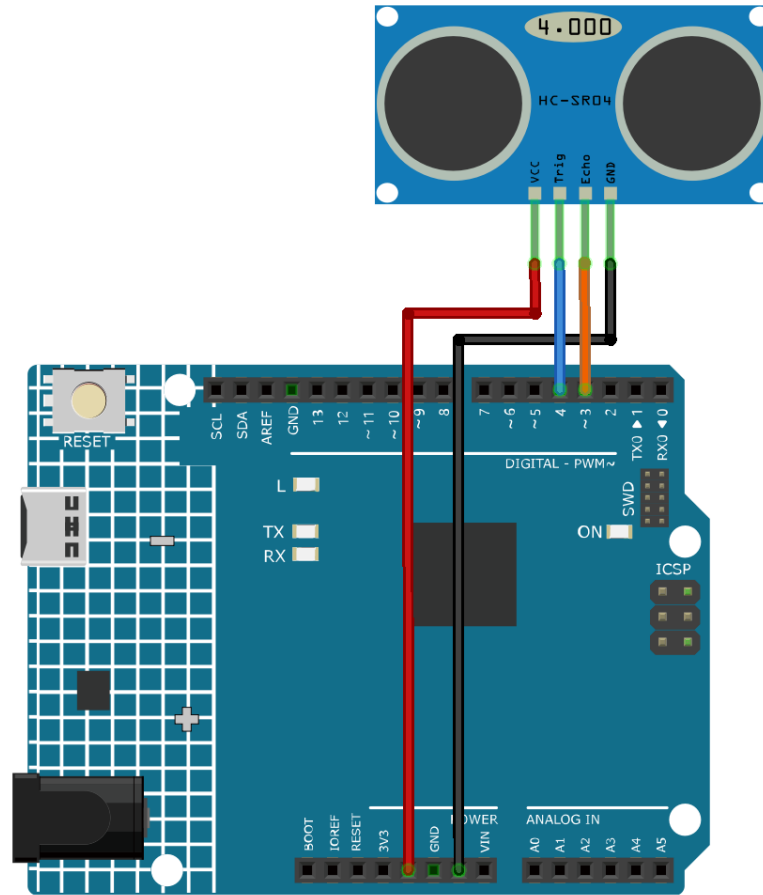
- $\text{us} / 58 = \text{centimeters}$
- $\text{us} / 148 = \text{inch}$
- $\text{distance} = \text{high level time} * \text{speed of sound (340m/s)} / 2;$

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Ultrasonic Sensor Module \* 1
- Jumper Wires

### Circuit Assembly



## Code

### Code explanation

#### 1. Pin declaration:

Start by defining the pins for the ultrasonic sensor. `echoPin` and `trigPin` are declared as integers and their values are set to match the physical connection on the Arduino board.

```
const int echoPin = 3;
const int trigPin = 4;
```

#### 2. `setup()` function:

The `setup()` function initializes the serial communication, sets the pin modes, and prints a message to indicate the ultrasonic sensor is ready.

```
void setup() {
  Serial.begin(9600);
  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);
  Serial.println("Ultrasonic sensor:");
}
```

#### 3. `loop()` function:

The `loop()` function reads the distance from the sensor and prints it to the serial monitor, then delays for

400 milliseconds before repeating.

```
void loop() {  
  float distance = readDistance();  
  Serial.print(distance);  
  Serial.println(" cm");  
  delay(400);  
}
```

#### 4. readDistance() function :

The readDistance() function triggers the ultrasonic sensor and calculates the distance based on the time it takes for the signal to bounce back.

```
float readDistance() {  
  digitalWrite(trigPin, LOW); // Set trig pin to low to ensure a clean pulse  
  delayMicroseconds(2);      // Delay for 2 microseconds  
  digitalWrite(trigPin, HIGH); // Send a 10 microsecond pulse by setting trig  
  ↪ pin to high  
  delayMicroseconds(10);  
  digitalWrite(trigPin, LOW); // Set trig pin back to low  
  float distance = pulseIn(echoPin, HIGH) / 58.00; // Formula: (340m/s * 1us) /  
  ↪ 2  
  return distance;  
}
```

### Additional Ideas

- Display the distance on an LCD screen instead of serial monitor
- Add LEDs that light up when object is within a threshold distance

### More Projects

- *Smart trashcan*

### 2.4.3 Gas/Smoke Sensor Module (MQ2)



#### Introduction

The MQ-2 sensor is a versatile gas sensor capable of detecting a wide range of gases including alcohol, carbon monoxide, hydrogen, isobutene, liquefied petroleum gas, methane, propane, and smoke. It is popular among beginners due to its low cost and easy-to-use features.

#### Principle

The MQ-2 sensor works on the principle of resistance changes in the presence of different gases. When the target gas comes in contact with the heated MOS(Metal Oxide Semiconductor) material, it undergoes oxidation or reduction reactions that change the resistance of the MOS material. It is noteworthy that the MQ2 gas sensor is capable of detecting multiple gases, but lacks the ability to differentiate between them. This is a common characteristic of most gas sensors.

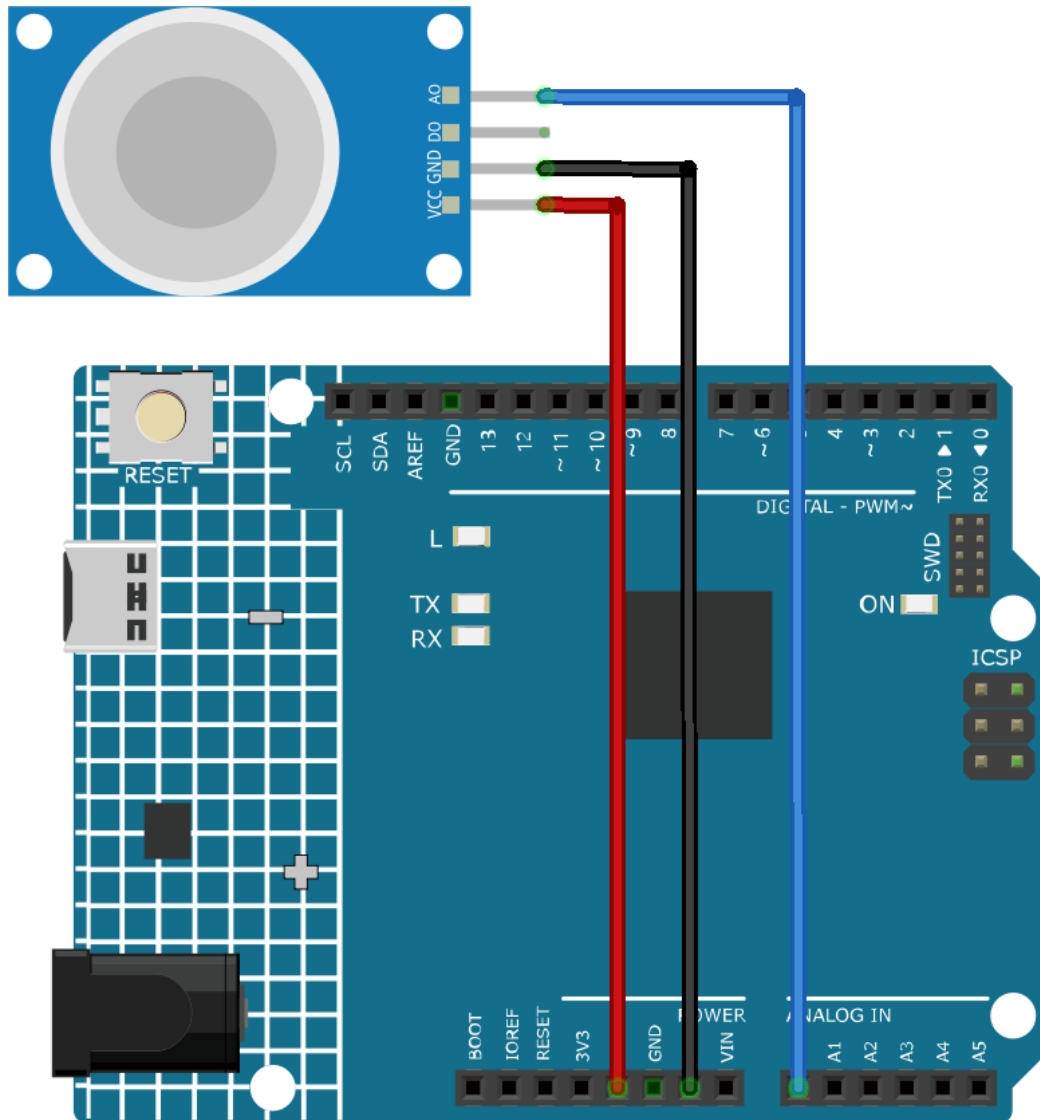
The sensor has a built-in potentiometer that allows you to adjust the sensor digital output (D0) threshold. When the concentration of gas in the air exceeds a certain threshold value, the resistance of the sensor changes. This change in resistance is then converted into an electrical signal that can be read by an Arduino board.

#### Usage

##### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Gas Sensor Module(MQ2) \* 1
- Jumper Wires

##### Circuit Assembly



## Code

### Code explanation

1. The first line of code is a constant integer declaration for the gas sensor pin. We use the analog pin A0 to read the output from the gas sensor.

```
const int sensorPin = A0;
```

2. The `setup()` function is where we initialize our serial communication at a baud rate of 9600. This is necessary to print the readings from the gas sensor to the serial monitor.

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



3. The `loop()` function is where we continuously read the analog value from the gas sensor and print it to the serial monitor. We use the `analogRead()` function to read the analog value from the sensor. We then wait for 50 milliseconds before the next reading. This delay gives some breathing space for the serial monitor to process the data.

```
void loop() {  
  Serial.print("Analog output: ");  
  Serial.println(analogRead(sensorPin)); // Read the analog value of the gas  
  sensor and print it to the serial monitor  
  delay(50); // Wait for 50 milliseconds  
}
```

**Note:** MQ2 is a heating-driven sensor that usually requires preheating before use. During the preheating period, the sensor typically reads high and gradually decreases until it stabilizes.

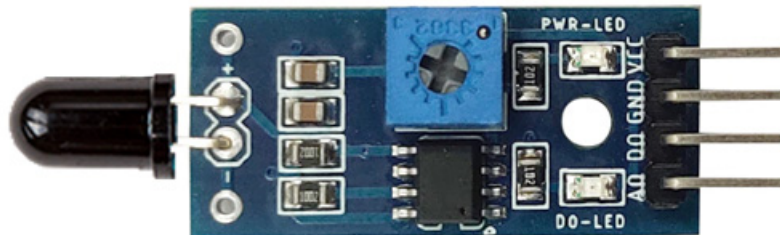
### Additional Ideas

- add a visual or auditory warning system (using LEDs or a buzzer) that triggers when gas concentrations exceed certain thresholds.

### More Projects

- *Gas leak alarm*

## 2.4.4 Flame Sensor Module



### Introduction

The Flame sensor is a sensor that can detect the presence of fire or flames. It detects fires mainly by sensing infrared radiation emitted by fires or flames. It is widely used in fire detection systems in homes and industries.

### Principle

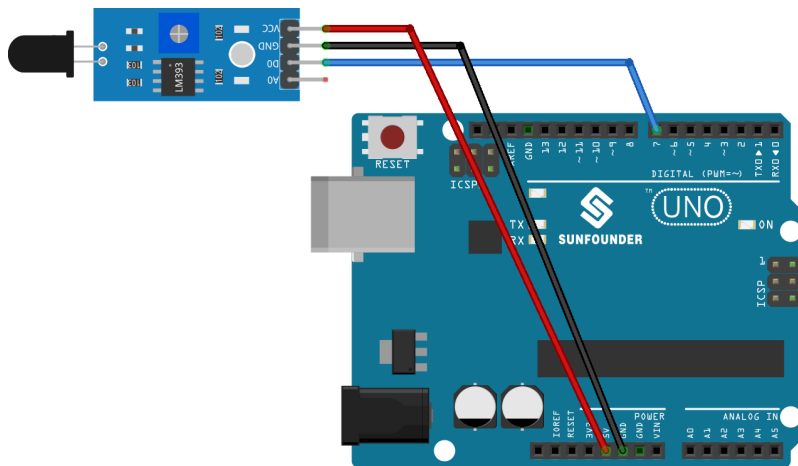
The Flame sensor works on the principle of infrared (IR) detection. The sensor has an IR receiver that detects the IR radiation emitted by flames. When fire burns it emits a small amount of Infra-red light, this light will be received by the Photodiode (IR receiver) on the sensor module. Then we use an Op-Amp to check for a change in voltage across the IR Receiver, so that if a fire is detected the output pin (DO) will give 0V(LOW), and if there is no fire the output pin will be 5V(HIGH).

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Flame Sensor Module \* 1
- Jumper Wires

#### Circuit Assembly



### Code

#### Code explanation

1. The first line of code is a constant integer declaration for the flame sensor pin. We use the digital pin 7 to read the output from the flame sensor.

```
const int sensorPin = 7;
```

2. The setup() function initializes the flame sensor pin as an input and the built-in LED pin as an output. It also starts the serial communication at a baud rate of 9600 for printing messages to the serial monitor.

```
void setup() {  
  pinMode(sensorPin, INPUT);    // Set the flame sensor pin as input  
  pinMode(LED_BUILTIN, OUTPUT); // Set the built-in LED pin as output  
  Serial.begin(9600);           // Initialize the serial monitor at a baud rate of  
  ↪ 9600  
}
```

3. The `loop()` function is where we continuously check the status of the flame sensor. If the sensor detects a flame, the built-in LED is turned on and a message is printed to the serial monitor. If no flame is detected, the LED is turned off and a different message is printed. The process repeats every 100 milliseconds.

```
void loop() {
  // Check if the sensor is detecting a fire
  if (digitalRead(sensorPin) == 0) {
    digitalWrite(LED_BUILTIN, HIGH); // Turn on the built-in LED
    Serial.println("*** Fire detected!!! ***");
  } else {
    digitalWrite(LED_BUILTIN, LOW); // Turn off the built-in LED
    Serial.println("No Fire detected");
  }
  delay(100);
}
```

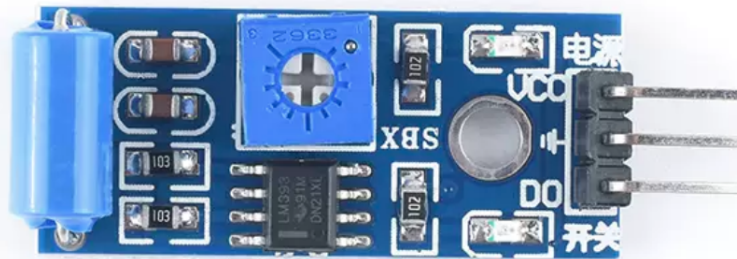
### Additional Ideas

- Modify the code to trigger a buzzer or sound an alarm when fire is detected.
- Incorporate a smoke sensor in addition to the flame sensor to enhance fire detection capabilities.
- Plot the **analog output** instead of just digital HIGH/LOW. Use **AO** pin.

### More Projects

- *Flame Alert System with Blynk*

## 2.4.5 Vibration Sensor Module (SW-420)



### Introduction

SW-420 vibration sensor is a module that can detect vibrations or shocks on a surface. It can be used for various purposes, such as detecting door knocks, machine malfunctions, car collisions, or alarm systems. It operates from 3.3 V to 5 V. The module has three peripherals, two LEDs, one for the power status and the other for the sensor output. In addition, there is a potentiometer that can be further used to control the threshold point of the vibration.

### Principle

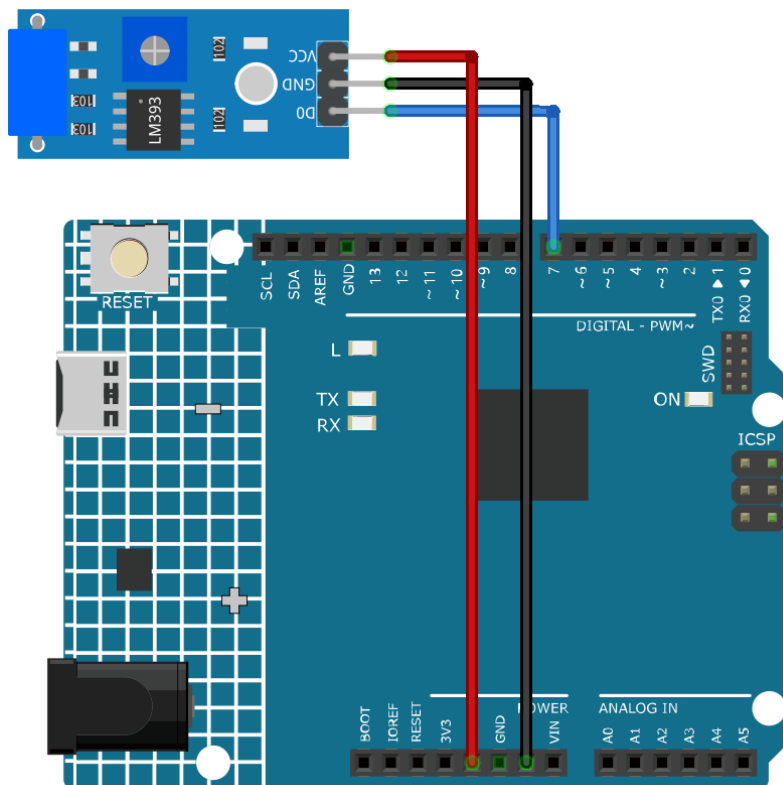
SW-420 vibration sensor module consists of a SW-420 vibration switch and an LM393 voltage comparator. A SW-420 vibration switch is a device that has a spring and a rod inside a tube. When the switch is exposed to a vibration, the spring touches the rod and closes the circuit. The vibration sensor in the module detects these oscillations and converts them into electrical signals. The LM393 comparator chip then compares these signals with a reference voltage set by the potentiometer. If the amplitude of the signal exceeds this reference voltage, the output of the comparator goes high (1), otherwise it goes low (0).

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Vibration Sensor Module(SW-420) \* 1
- Jumper Wires

#### Circuit Assembly



## Code

### Code explanation

1. The first line of code is a constant integer declaration for the vibration sensor pin. We use digital pin 7 to read the output from the vibration sensor.

```
const int sensorPin = 7;
```

2. In the setup() function, we initialize the serial communication at a baud rate of 9600 to print readings from the vibration sensor to the serial monitor. We also set the vibration sensor pin as an input.

```
void setup() {
  Serial.begin(9600);           // Start serial communication at 9600 baud rate
  pinMode(sensorPin, INPUT);    // Set the sensorPin as an input pin
}
```

3. The loop() function is where we continuously check for any vibrations detected by the sensor. If the sensor detects a vibration, it prints "Detected vibration..." to the serial monitor. If no vibration is detected, it prints "...". The loop repeats every 100 milliseconds.

```
void loop() {
  if (digitalRead(sensorPin)) {           // Check if there is any vibration
    ↳detected by the sensor
    Serial.println("Detected vibration..."); // Print "Detected vibration..." if
    ↳vibration detected
  }
  else {
    Serial.println("..."); // Print "..." otherwise
  }
  // Add a delay to avoid flooding the serial monitor
  delay(100);
}
```

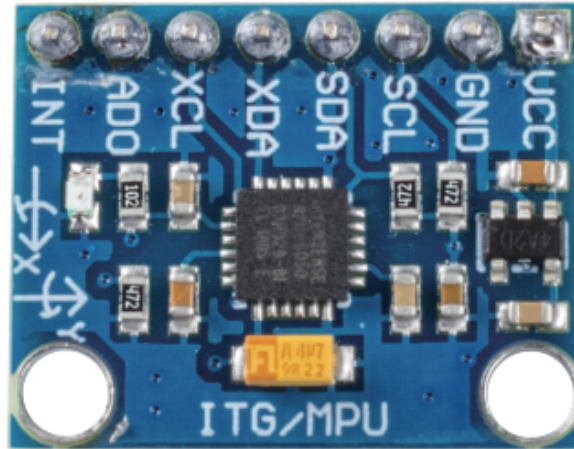
### Additional Ideas

- Could connect an LED to turn ON when vibration is detected
- An alarm sound or buzzer could be triggered on vibration sense

### More Projects

- *Vibration Alert System with IFTTT*

## 2.4.6 Accelerometer & Gyroscope Module (MPU6050)



### Introduction

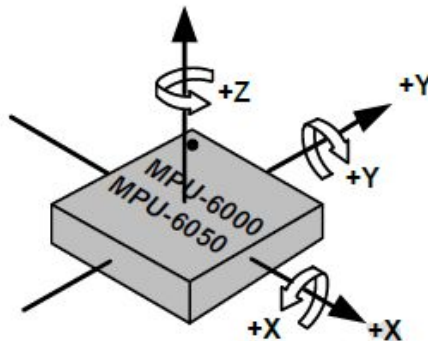
The MPU-6050 is a 6-axis (combines 3-axis Gyroscope, 3-axis Accelerometer) motion tracking device. Changes in motion, acceleration and rotation can be detected. It is commonly used in robotics, gaming controllers, and other electronic devices that require motion detection. Its high accuracy and cheap cost make it very popular among the DIY community.

### Principle

An MPU-650 sensor module consists of a 3-axis accelerometer and a 3-axis gyroscope.

Its three coordinate systems are defined as follows:

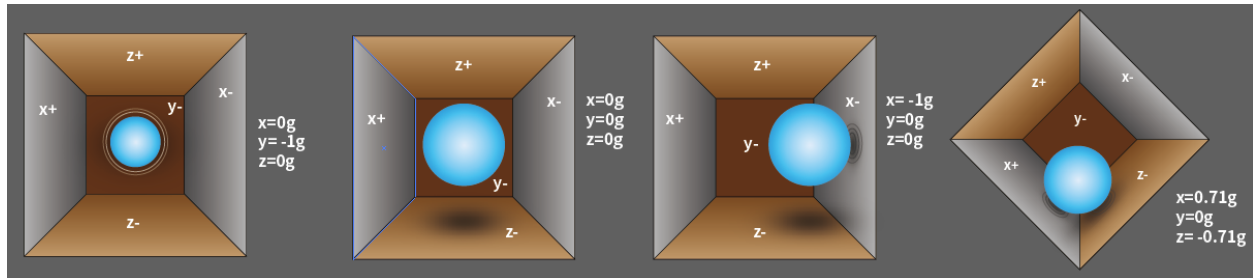
Put MPU6050 flat on the table, assure that the face with label is upward and a dot on this surface is on the top left corner. Then the upright direction upward is the z-axis of the chip. The direction from left to right is regarded as the X-axis. Accordingly the direction from back to front is defined as the Y-axis.



### 3-axis Accelerometer

The accelerometer works on the principle of piezo electric effect, the ability of certain materials to generate an electric charge in response to applied mechanical stress.

Here, imagine a cuboidal box, having a small ball inside it, like in the picture above. The walls of this box are made with piezo electric crystals. Whenever you tilt the box, the ball is forced to move in the direction of the inclination, due to gravity. The wall with which the ball collides, creates tiny piezo electric currents. There are totally, three pairs of opposite walls in a cuboid. Each pair corresponds to an axis in 3D space: X, Y and Z axes. Depending on the current produced from the piezo electric walls, we can determine the direction of inclination and its magnitude.



We can use the MPU6050 to detect its acceleration on each coordinate axis (in the stationary desktop state, the Z-axis acceleration is 1 gravity unit, and the X and Y axes are 0). If it is tilted or in a weightless/overweight condition, the corresponding reading will change.

There are four kinds of measuring ranges that can be selected programmatically:  $\pm 2g$ ,  $\pm 4g$ ,  $\pm 8g$ , and  $\pm 16g$  ( $2g$  by default) corresponding to each precision. Values range from -32768 to 32767.

The reading of accelerometer is converted to an acceleration value by mapping the reading from the reading range to the measuring range.

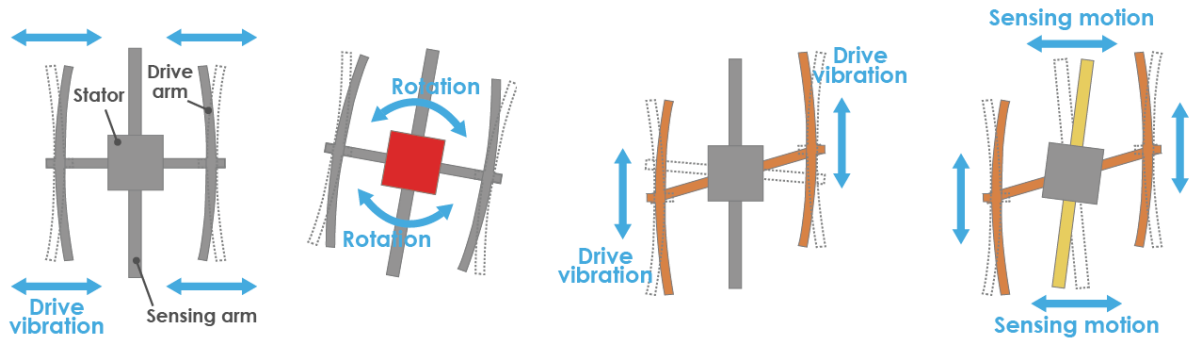
Acceleration = (Accelerometer axis raw data / 65536 \* full scale Acceleration range) g

Take the X-axis as an example, when Accelerometer X axis raw data is 16384 and the range is selected as  $\pm 2g$ :

Acceleration along the X axis =  $(16384 / 65536 * 4) g = 1g$

### 3-axis Gyroscope

Gyroscopes work on the principle of Coriolis acceleration. Imagine that there is a fork like structure, that is in constant back and forth motion. It is held in place using piezo electric crystals. Whenever, you try to tilt this arrangement, the crystals experience a force in the direction of inclination. This is caused as a result of the inertia of the moving fork. The crystals thus produce a current in consensus with the piezo electric effect, and this current is amplified.



1. Normally, a drive arm vibrates in a certain direction.
2. Direction of rotation
3. When the gyro is rotated, the Coriolis force acts on the drive arms, producing vertical vibration.
4. The stationary part bends due to vertical drive arm vibration, producing a sensing motion in the sensing arms.

The Gyroscope also has four kinds of measuring ranges: +/- 250, +/- 500, +/- 1000, +/- 2000. The calculation method and Acceleration are basically consistent.

The formula for converting the reading into angular velocity is as follows:

Angular velocity = (Gyroscope axis raw data / 65536 \* full scale Gyroscope range) °/s

The X axis, for example, the Accelerometer X axis raw data is 16384 and ranges + / - 250°/s:

Angular velocity along the X axis = (16384 / 65536 \* 500)°/s = 125°/s

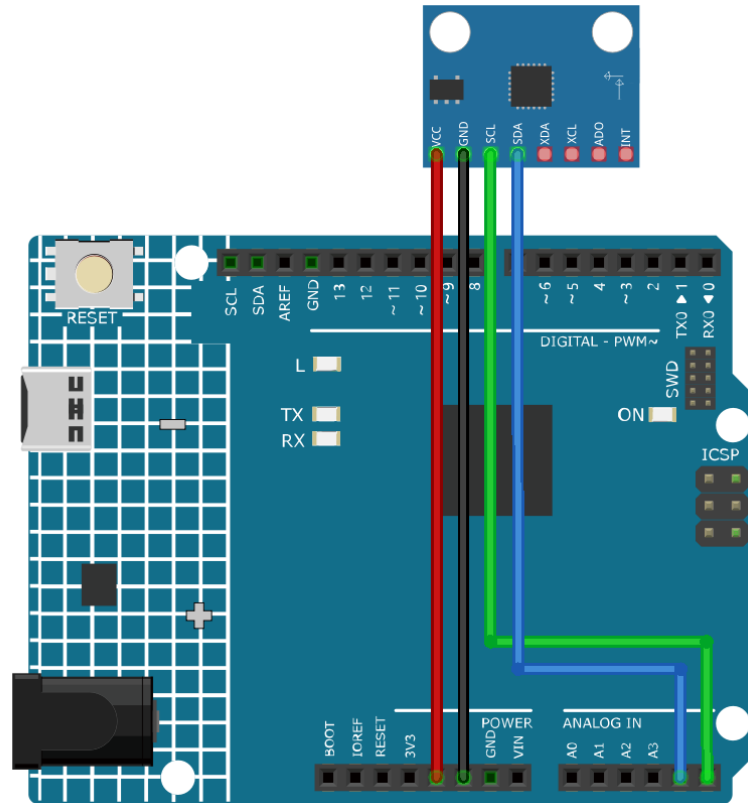
## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Accelerometer & Gyroscope Module(MPU6050) \* 1
- Jumper Wires

### Circuit Assembly





## Code

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit MPU6050**” and install it.

## Code explanation

1. The code starts by including the necessary libraries and creating an object for the MPU6050 sensor. This code uses the Adafruit\_MPU6050 library, Adafruit\_Sensor library, and Wire library. The Adafruit\_MPU6050 library is used to interact with the MPU6050 sensor and retrieve acceleration, rotation, and temperature data. The Adafruit\_Sensor library provides a common interface for various types of sensors. The Wire library is used for I2C communication, which is necessary to communicate with the MPU6050 sensor.

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit MPU6050**” and install it.

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
Adafruit_MPU6050 mpu;
```

2. The setup() function initializes the serial communication and checks if the sensor is detected. If the sensor is not found, the Arduino enters an infinite loop with a “Failed to find MPU6050 chip” message. If found, the

accelerometer range, gyro range, and filter bandwidth are set, and a delay is added for stability.

```
void setup(void) {  
  // Initialize the serial communication  
  Serial.begin(9600);  
  
  // Check if the MPU6050 sensor is detected  
  if (!mpu.begin()) {  
    Serial.println("Failed to find MPU6050 chip");  
    while (1) {  
      delay(10);  
    }  
  }  
  Serial.println("MPU6050 Found!");  
  
  // set accelerometer range to +-8G  
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);  
  
  // set gyro range to +- 500 deg/s  
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);  
  
  // set filter bandwidth to 21 Hz  
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);  
  
  // Add a delay for stability  
  delay(100);  
}
```

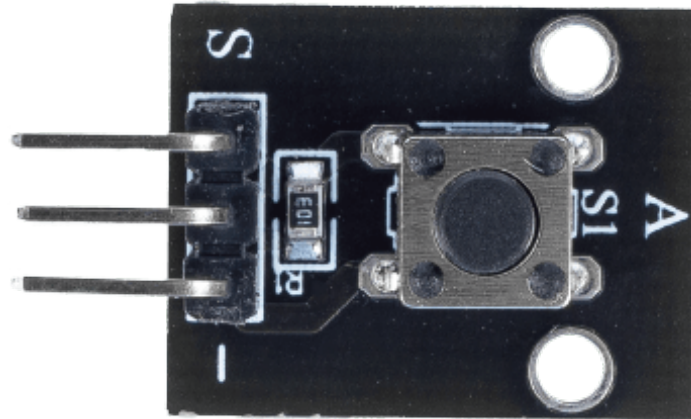
3. In the loop() function, the program creates events to store the sensor readings and then retrieves the readings. The acceleration, rotation, and temperature values are then printed to the serial monitor.

```
void loop() {  
  // Get new sensor events with the readings  
  sensors_event_t a, g, temp;  
  mpu.getEvent(&a, &g, &temp);  
  
  // Print out the acceleration, rotation, and temperature readings  
  // ...  
  
  // Add a delay to avoid flooding the serial monitor  
  delay(1000);  
}
```

## Additional Ideas

- Visualize sensor data in graphical format on an LCD or OLED

### 2.4.7 Button Module



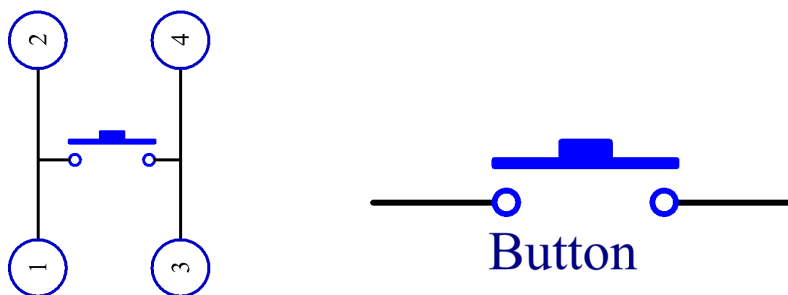
#### Introduction

The button module is an electronic device that detects the state of a button. They are usually used as switches to connect or break circuits. Buttons are used in many scenarios, such as doorbells, desk lamps, remote controls, elevators, fire alarms, etc.

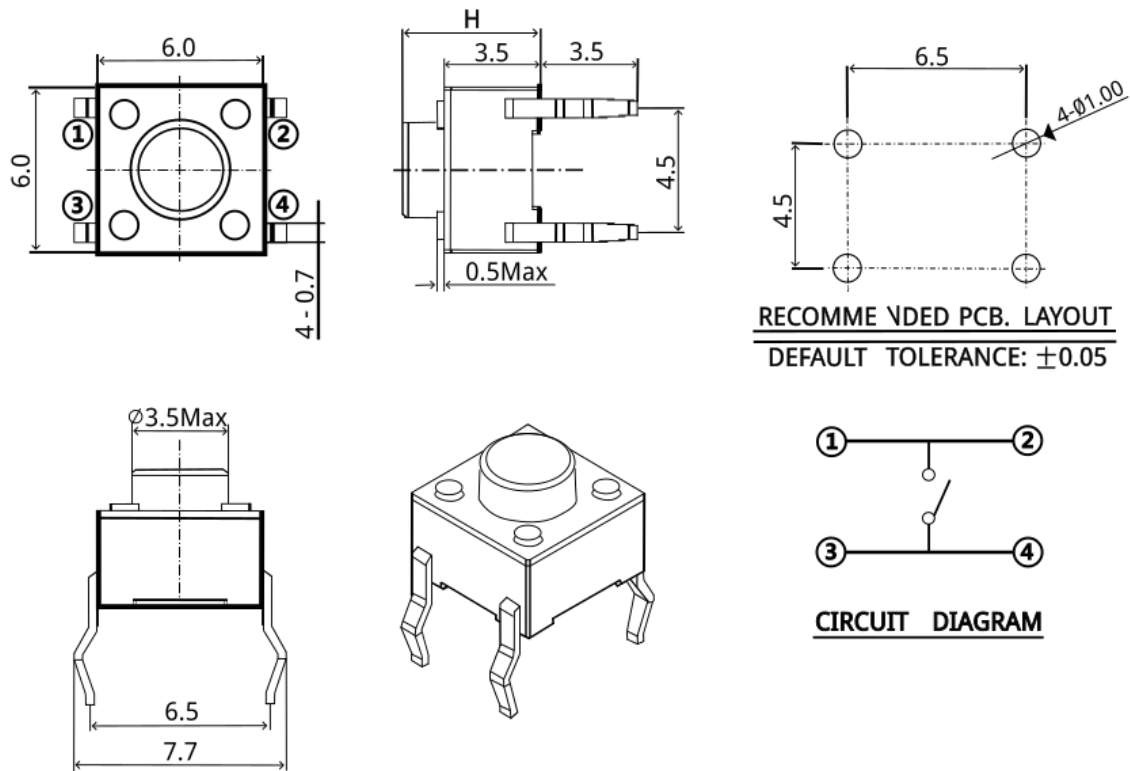
#### Principle

The button module works on the principle of a switch. A switch is an electrical component that can be used to open or close a circuit.

The following is the internal structure of a button. The symbol on the right below is usually used to represent a button in circuits.



Since the pin 1 is connected to pin 2, and pin 3 to pin 4, when the button is pressed, the 4 pins are connected, thus closing the circuit.

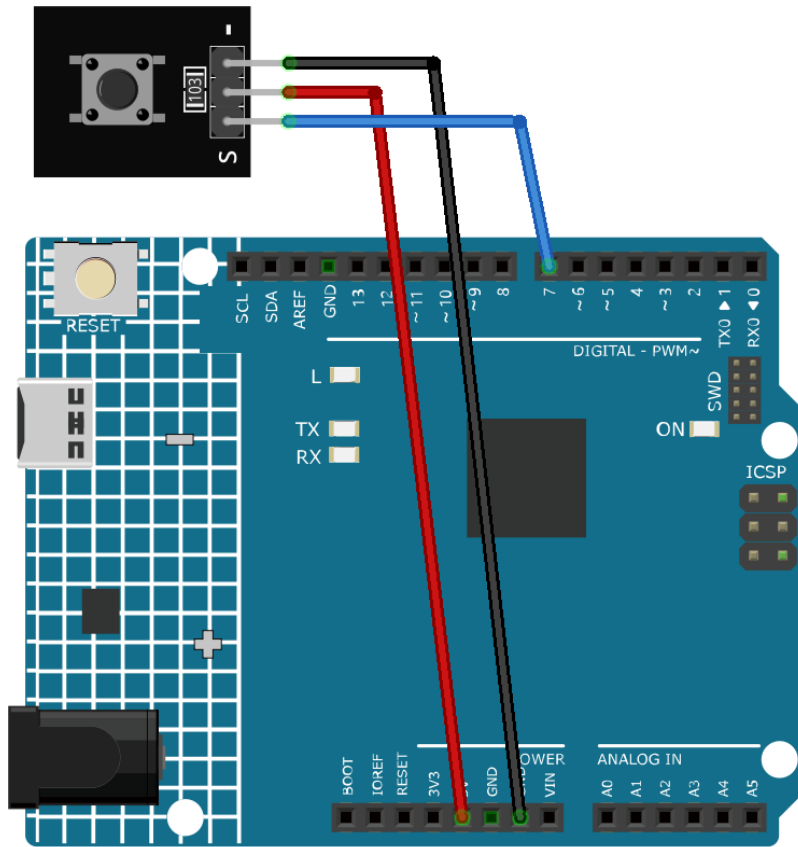


## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Button Module \* 1
- Jumper Wires

### Circuit Assembly



## Code

### Code explanation

1. **Setting Up:** In this part of the code, we first declare the `sensorPin` as a constant integer and assign it the pin number we will connect our button to on the Arduino board. The `setup()` function sets the mode of the `sensorPin` as `INPUT`, meaning we'll be receiving data in through this pin from the button. The `Serial.begin()` function initiates serial communication at a baud rate of 9600.

```
const int sensorPin = 7;

void setup() {
  pinMode(sensorPin, INPUT);
  Serial.begin(9600);
}
```

2. **The Loop:** The `loop()` function contains the main logic of the program. It continuously reads the button state and prints it to the serial monitor every 50 milliseconds. The `digitalRead()` function reads the state of the button, and the `Serial.println()` function prints this value to the serial monitor. The `delay()` function then pauses the execution for 50 milliseconds before the next reading. The button outputs a low level when pressed, and a high level when released.

```
void loop() {
  Serial.println(digitalRead(sensorPin));
}
```

(continues on next page)

(continued from previous page)

```
    delay(50);  
}
```

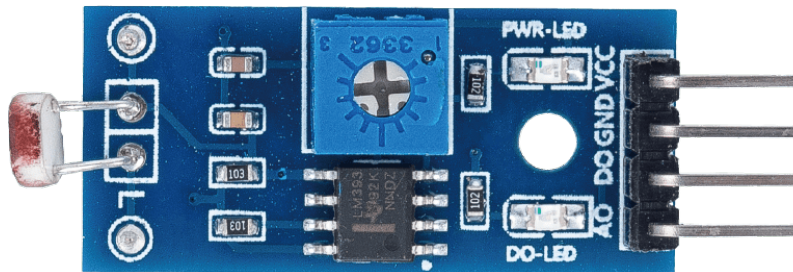
### Additional Ideas

- Use the button with if statements to control different outcomes in a program.
- Make the button toggle an LED on and off instead of just printing to serial monitor.

### More Projects

- *Doorbell*

## 2.4.8 Photoresistor Module



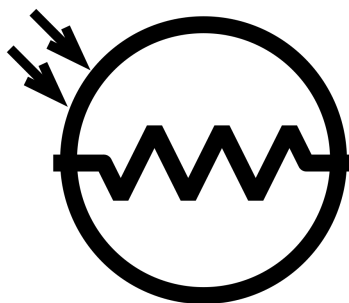
### Introduction

A photoresistor module is a device that can detect the intensity of light in the environment. It can be used for various purposes, such as adjusting the brightness of a device, detecting day and night, or activating a light switch.

An important component of the photoresistor module is the photoresistor. A photoresistor is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity.

A photoresistor can be applied in light-sensitive detector circuits and light-activated and dark-activated switching circuits acting as a resistance semiconductor. In the dark, a photoresistor can have a resistance as high as several megaohms (M), while in the light, a photoresistor can have a resistance as low as a few hundred ohms.

Here is the electronic symbol of photoresistor.



## Principle

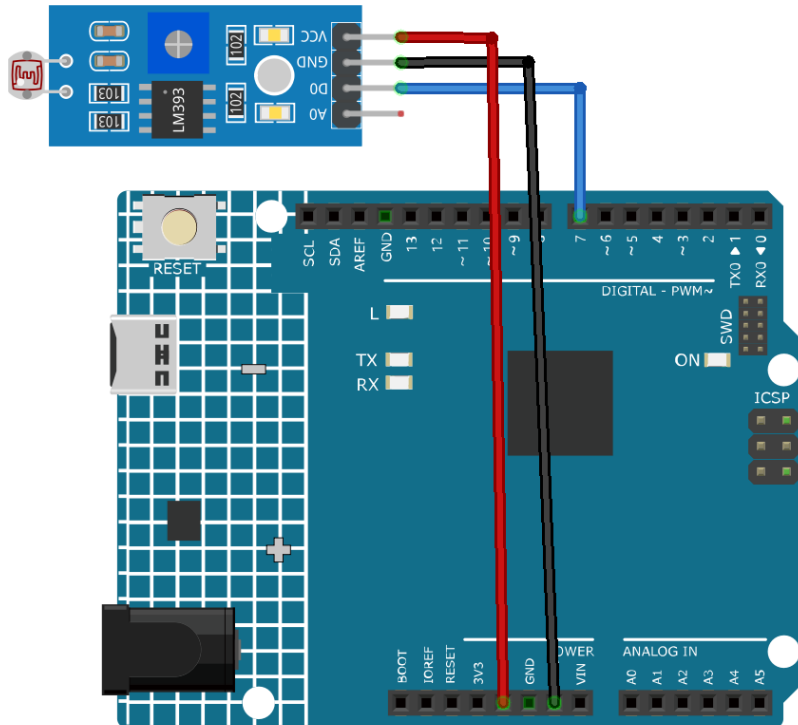
The photoresistor module works on the principle of changing resistance in response to different light intensities. The sensor has a built-in potentiometer that adjusts the sensor's digital output (D0) threshold. When the intensity of light exceeds a certain threshold, the resistance of the sensor changes. This change in resistance is then converted to an electrical signal that can be read by the Arduino board.

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Photoresistor Module \* 1
- Jumper Wires

### Circuit Assembly



## Code

### Code explanation

1. This line of code defines the pin number to which the photoresistance sensor is connected on the Arduino board.

```
const int sensorPin = 7;
```

2. The `setup()` function is a special function in Arduino that is executed only once when the Arduino is powered on or reset. In this project, the `sensorPin` is set as `INPUT` because we are reading values from it. The `Serial.begin(9600)` command initiates serial communication at a baud rate of 9600.

```
void setup() {  
  pinMode(sensorPin, INPUT);  
  Serial.begin(9600);  
}
```

3. The `loop()` function is the main function where the program runs repeatedly. In this function, the `digitalRead` function reads the digital value from the photoresistor sensor and prints it to the serial monitor using `Serial.println`. The `delay(50)` command makes the program wait for 50 milliseconds before taking the next reading.

```
void loop() {  
  Serial.println(digitalRead(sensorPin));  
  delay(50);  
}
```

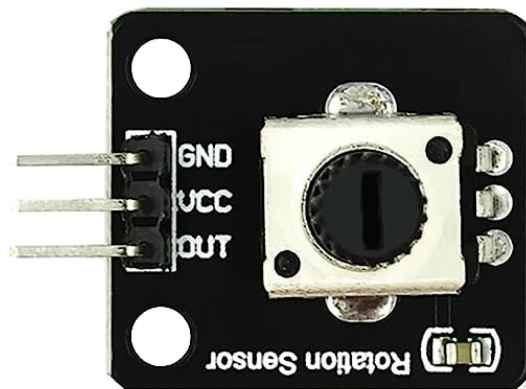
### Additional Ideas

- Use the sensor to turn on/off an LED or relay.
- Plot the **analog output** instead of just digital HIGH/LOW. Use AO pin.

### More Projects

- *Light control switch*

## 2.4.9 Potentiometer Module





## Introduction

The potentiometer module is an electronic component that changes its resistance depending on the position of the twist knob. It can be used for various purposes, such as controlling the volume of a speaker, the brightness of a LED, or the speed of a motor.

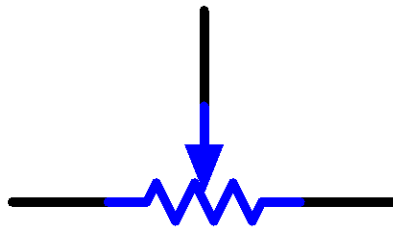
## Principle

Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation.

Potentiometers come in various shapes, sizes, and values, but they all have the following things in common:

- They have three terminals (or connection points).
- They have a knob, screw, or slider that can be moved to vary the resistance between the middle terminal and either one of the outer terminals.
- The resistance between the middle terminal and either one of the outer terminals varies from 0 to the maximum resistance of the pot as the knob, screw, or slider is moved.

Here is the circuit symbol of potentiometer.



The functions of the potentiometer in the circuit are as follows:

### 1. Serving as a voltage divider

Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the travel it has made.

### 2. Serving as a rheostat

When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.

### 3. Serving as a current controller

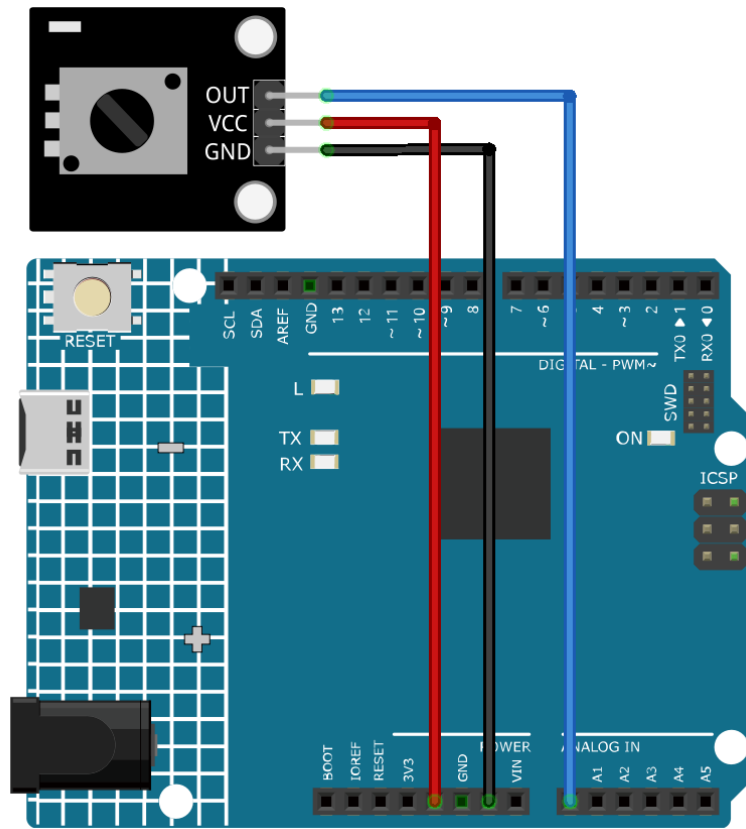
When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Potentiometer Module \* 1
- Jumper Wires

#### Circuit Assembly



### Code

#### Code explanation

1. This line of code defines the pin number to which the potentiometer is connected on the Arduino board.

```
const int sensorPin = A0;
```

2. The `setup()` function is a special function in Arduino that is executed only once when the Arduino is powered on or reset. In this project, the `Serial.begin(9600)` command initiates serial communication at a baud rate of 9600.

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

3. The `loop()` function is the main function where the program runs repeatedly. In this function, the `analogRead()` function reads the analog value from the potentiometer and prints it to the serial monitor using `Serial.println()`. The `delay(50)` command makes the program wait for 50 milliseconds before taking the next reading.

```
void loop() {  
  Serial.println(analogRead(sensorPin));  
  delay(50);  
}
```

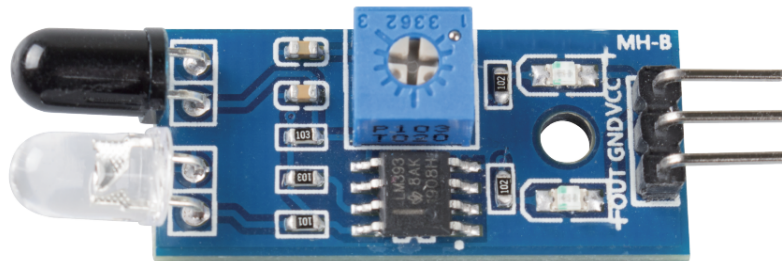
### Additional Ideas

- Control an LED's brightness: The potentiometer's analog value could be used to control the brightness of an LED connected to a PWM-enabled pin on the Arduino.
- Control a Servo Motor's Position: By mapping the analog value to the range of the servo's position (usually 0 to 180 degrees), the potentiometer could be used as a controller for the servo motor.

### More Projects

- *Potentiometer scale value*

## 2.4.10 IR Obstacle Avoidance Sensor Module

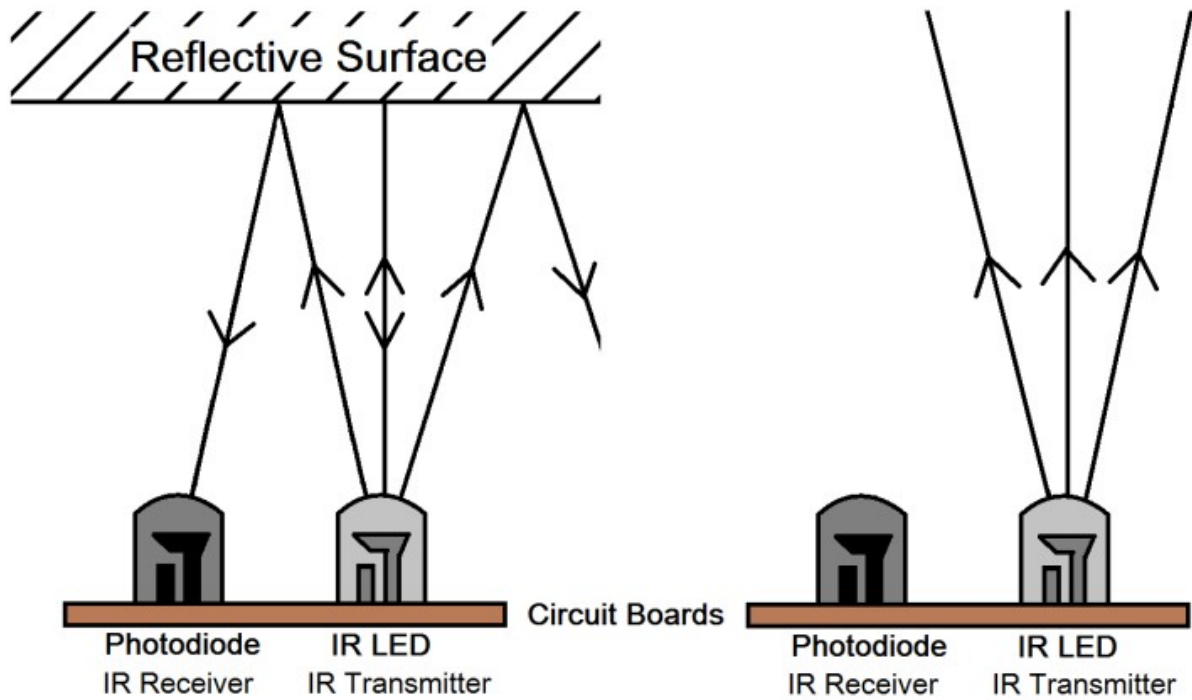


### Introduction

An IR Obstacle Sensor works in accordance with the infrared reflection principle to detect obstacles. When there is no object, the infrared receiver receives no signals; when there is an object ahead which blocks and reflects the infrared light, the infrared receiver will receive signals.

### Principle

An obstacle avoidance sensor mainly consists of an infrared transmitter, an infrared receiver and a potentiometer. According to the reflecting character of an object, if there is no obstacle, the emitted infrared ray will weaken with the distance it spreads and finally disappear. If there is an obstacle, when the infrared ray encounters it, the ray will be reflected back to the infrared receiver. Then the infrared receiver detects this signal and confirms an obstacle in front. The detection range can be adjusted by the built-in potentiometer.

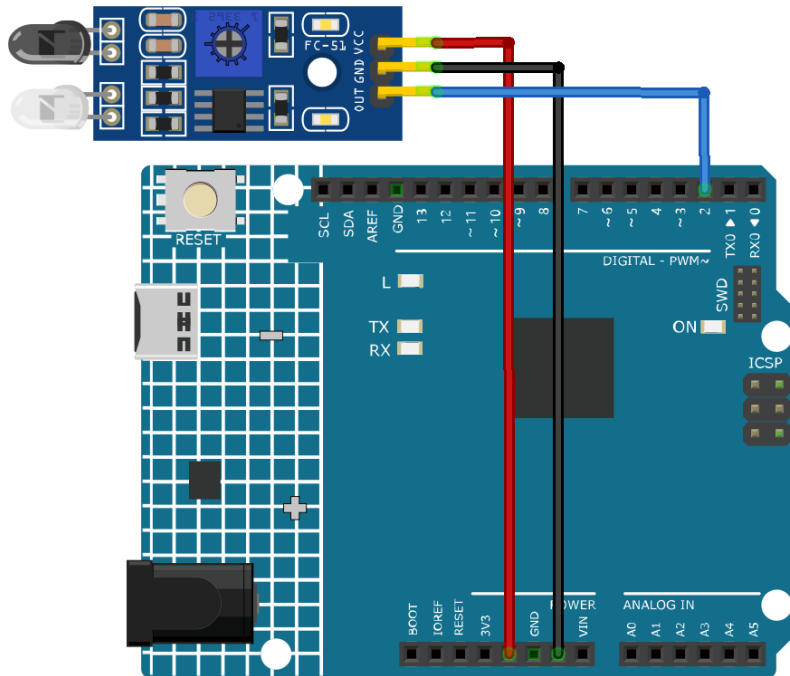


## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- IR Obstacle Avoidance Sensor Module \* 1
- Jumper Wires

### Circuit Assembly



## Code

### Code explanation

1. Define pin number for sensor connection:

```
const int sensorPin = 2;
```

Connect the sensor's output pin to Arduino pin 2.

2. Setup serial communication and define sensor pin as input:

```
void setup() {
  pinMode(sensorPin, INPUT);
  Serial.begin(9600);
}
```

Initialize serial communication at 9600 baud rate to print to serial monitor. Set sensor pin as input to read input signal.

3. Read sensor value and print to serial monitor:

```
void loop() {
  Serial.println(digitalRead(sensorPin));
  delay(50);
}
```

Continuously read digital value from sensor pin using `digitalRead()` and print value to serial monitor using `Serial.println()`. Add 50ms delay between prints for better viewing.

**Note:** If the sensor is not working properly, adjust the IR transmitter and receiver to make them parallel. Additionally,

you can adjust the detection range using the built-in potentiometer.

---

### Additional Ideas

- Add buzzer that beeps when obstacle is detected

### More Projects

- *Automatic soap dispenser*

## 2.4.11 Capacitive Soil Moisture Module



### Introduction

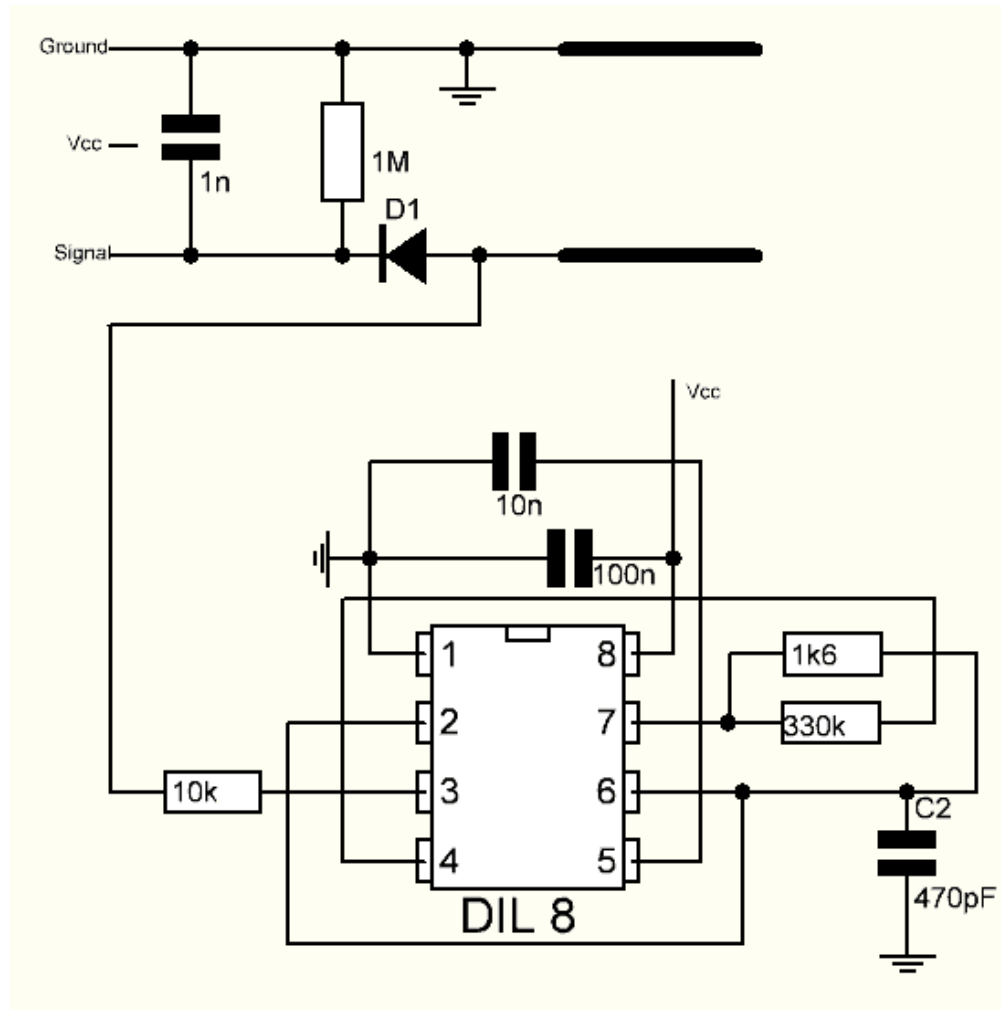
The Soil Moisture Module is a sensor that measures the moisture content of soil. It is used in agriculture to monitor soil moisture levels and help farmers determine when to water their crops.

### Principle

This capacitive soil moisture sensor is different from most of the resistive sensors on the market, using the principle of capacitive induction to detect soil moisture. It avoids the problem that resistive sensors are highly susceptible to corrosion and greatly extends its working life.

It is made of corrosion-resistant materials and has an excellent service life. Insert it into the soil around plants and monitor real-time soil moisture data. The module includes an on-board voltage regulator that allows it to operate over a voltage range of 3.3 ~ 5.5 V. It is ideal for low-voltage microcontrollers with 3.3 V and 5 V supplies.

The hardware schematic of the capacitive soil moisture sensor is shown below.



There is a fixed frequency oscillator, which is built with a 555 timer IC. The generated square wave is then fed to the sensor like a capacitor. However, for the square wave signal, the capacitor has a certain reactance or, for the sake of argument, a resistor with a pure ohmic resistor (10k resistor on pin 3) to form a voltage divider.

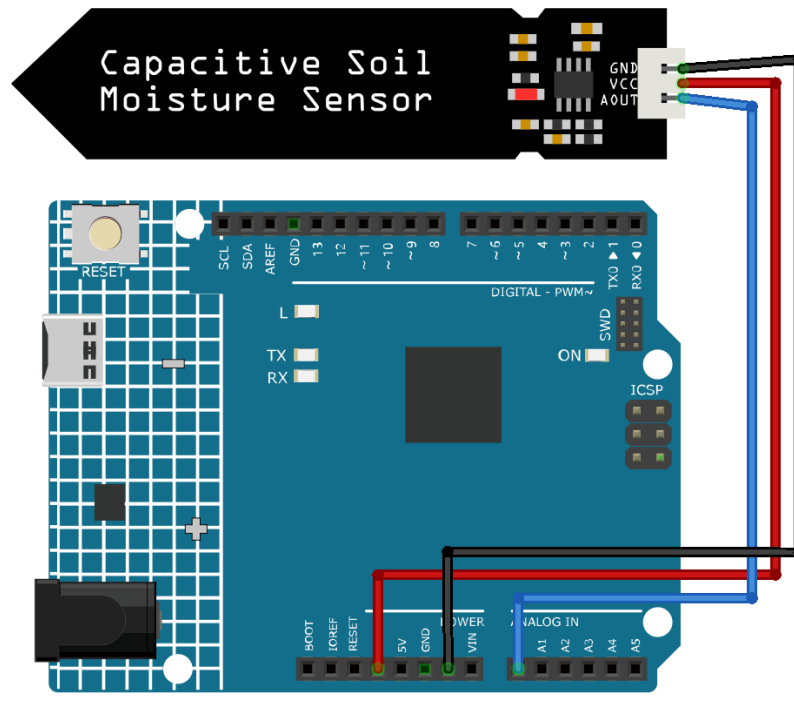
The higher the soil moisture, the higher the capacitance of the sensor. As a result, the square wave has less reactance, which reduces the voltage on the signal line, and the smaller the value of the analog input through the microcontroller.

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Soil Moisture Module \* 1
- Jumper Wires

### Circuit Assembly



## Code

### Code explanation

1. Defining sensor pin In this part of the code, a constant integer named `sensorPin` is defined and assigned the value `A0`. `A0` is the analog input pin on the Arduino board where the soil moisture sensor is connected.

```
const int sensorPin = A0;
```

2. Setting up the serial communication The `setup()` function is called once when the Arduino is powered on or reset. Here, we initialize the Serial library at 9600 baud rate. The baud rate is the rate at which information is transferred. In this case, it's 9600 bits per second (bps).

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

3. Reading data and printing to the serial monitor The loop function is where the main logic of the program resides. This function loops indefinitely once the program starts. Inside the loop, we use the `analogRead()` function to read the data from the moisture sensor and print it to the Serial Monitor. We then pause the program for 500 milliseconds using the `delay()` function before taking the next reading.

```
void loop() {  
  Serial.println(analogRead(sensorPin));  
  delay(500);  
}
```

---

**Note:** The smaller the value, the higher the soil moisture level.

---



## Additional Ideas

- Integrate a buzzer or LED that activates if the moisture level goes below a certain threshold. This way, you'll have a physical alert to water your plants.
- You could connect a water pump and automate the watering process. When the moisture level drops below a threshold, the Arduino can activate the pump to water the plants.

## More Projects

- *Plant Monitor with Blynk*
- *Auto Watering System with Blynk*

### 2.4.12 Raindrop Detection Module



#### Introduction

A raindrop sensor, or raindrop detection sensor, is used to detect whether it is raining and also the rainfall. It is widely used in the automatic wiper system, smart lighting system and sunroof system of automobiles.

#### Principle

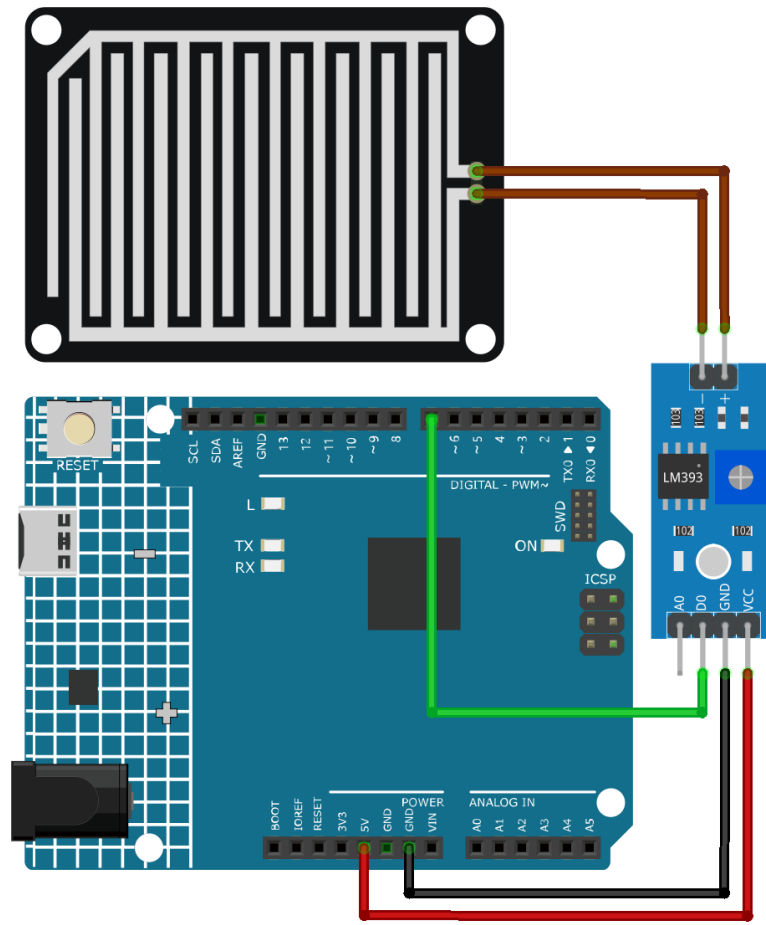
Raindrop sensor is basically a board on which nickel is coated in the form of lines. It works on the principal of resistance. When there is no rain drop on board. Resistance is high so we get high voltage according to  $V=IR$ . When rain drop is present it reduces the resistance because water is a conductor of electricity and the presence of water connects nickel lines in parallel so resistance is reduced and voltage drop across it is reduced. The more intense the rainfall the lower the resistance.

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Raindrop Detection Module \* 1
- Jumper Wires

#### Circuit Assembly



### Code

#### Code explanation

1. Defining sensor pin Here, a constant integer named `sensorPin` is defined and assigned the value 7. This corresponds to the digital pin on the Arduino board where the raindrops detection sensor is connected.

```
const int sensorPin = 7;
```

2. Setting up the pin mode and initiating serial communication In the `setup()` function, two essential steps are performed. Firstly, `pinMode()` is used to set the `sensorPin` as an input, enabling us to read digital values from the raindrops sensor. Secondly, serial communication is initialized with a baud rate of 9600.

```
void setup() {  
  pinMode(sensorPin, INPUT);  
  Serial.begin(9600);  
}
```

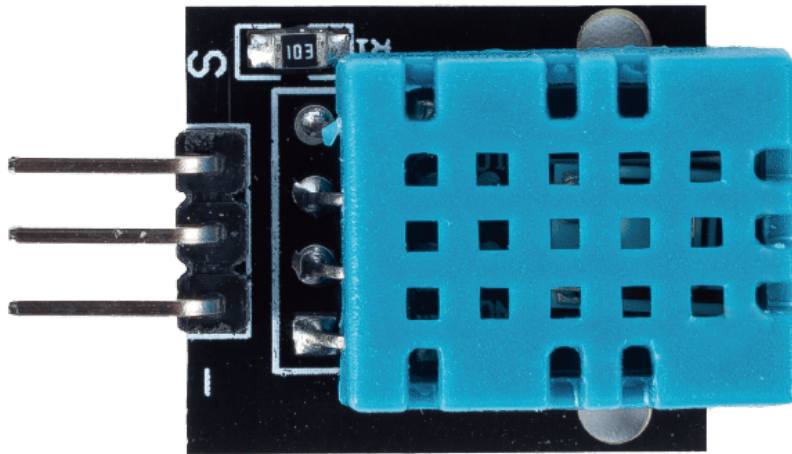
3. Reading the digital value and sending it to the serial monitor. The `loop()` function reads the digital value from the raindrops sensor using `digitalRead()`. This value (either HIGH or LOW) is printed to the Serial Monitor. The program then waits for 50 milliseconds before the next reading.

```
void loop() {  
  Serial.println(digitalRead(sensorPin));  
  delay(50);  
}
```

### Additional Ideas

- Add an LED indicator that lights up when rain is detected
- Connect a buzzer to the Arduino to sound an alert when rain is detected, which can act as an early warning system for events like picnics or outdoor activities.

## 2.4.13 Temperature and Humidity Sensor Module (DHT11)



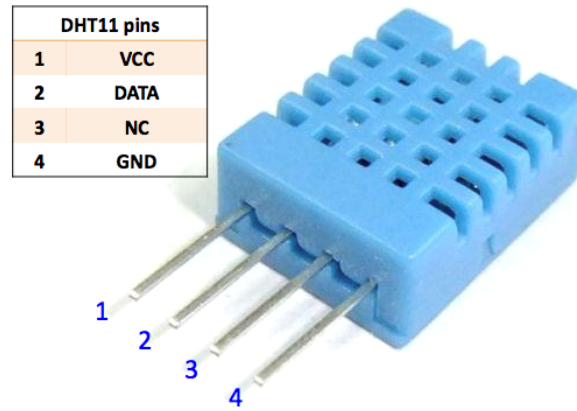
### Introduction

The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the temperature and humidity sensing technology are applied to ensure that the product has high reliability and excellent long-term stability.

The sensor includes a resistive sense of wet component and an NTC temperature measurement device, and is connected with a high-performance 8-bit microcontroller.

### Principle

Only three pins are available for use: VCC, GND, and DATA. The communication process begins with the DATA line sending start signals to DHT11, and DHT11 receives the signals and returns an answer signal. Then the host receives the answer signal and begins to receive 40-bit humidity and temperature data (8-bit humidity integer + 8-bit humidity decimal + 8-bit temperature integer + 8-bit temperature decimal + 8-bit checksum).

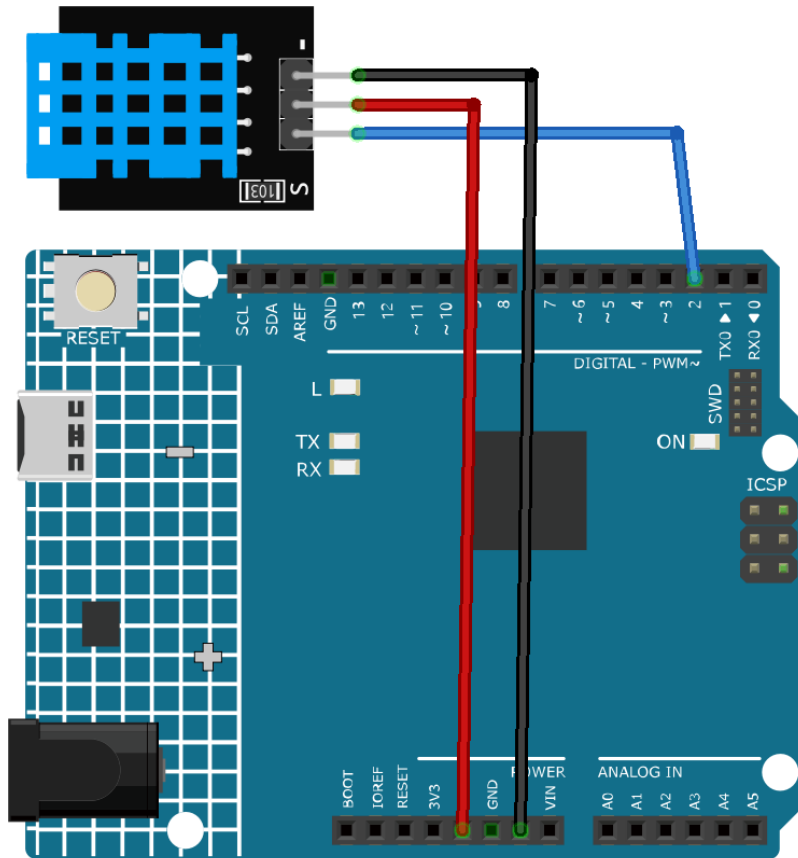


### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Temperature and Humidity Sensor Module(DHT11) \* 1
- Jumper Wires

#### Circuit Assembly



## Code

**Note:** To install the library, use the Arduino Library Manager and search for “**DHT sensor library**” and install it.

## Code explanation

1. Inclusion of necessary libraries and definition of constants. This part of the code includes the DHT sensor library and defines the pin number and sensor type used in this project.

**Note:** To install the library, use the Arduino Library Manager and search for “**DHT sensor library**” and install it.

```
#include <DHT.h>
#define DHTPIN 2           // Define the pin used to connect the sensor
#define DHTTYPE DHT11     // Define the sensor type
```

2. Creation of DHT object. Here we create a DHT object using the defined pin number and sensor type.

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

3. This function is executed once when the Arduino starts. We initialize the serial communication and the DHT sensor in this function.

```
void setup() {  
  Serial.begin(9600);  
  Serial.println(F("DHT11 test!"));  
  dht.begin(); // Initialize the DHT sensor  
}
```

4. Main loop. The loop() function runs continuously after the setup function. Here, we read the humidity and temperature values, calculate the heat index, and print these values to the serial monitor. If the sensor read fails (returns NaN), it prints an error message.

---

**Note:** There is a way to measure how hot it feels outside by combining the air temperature and the humidity. It is also called the “felt air temperature” or “apparent temperature”.

---

```
void loop() {  
  delay(2000);  
  float h = dht.readHumidity();  
  float t = dht.readTemperature();  
  float f = dht.readTemperature(true);  
  if (isnan(h) || isnan(t) || isnan(f)) {  
    Serial.println(F("Failed to read from DHT sensor!"));  
    return;  
  }  
  float hif = dht.computeHeatIndex(f, h);  
  float hic = dht.computeHeatIndex(t, h, false);  
  Serial.print(F("Humidity: "));  
  Serial.print(h);  
  Serial.print(F("% Temperature: "));  
  Serial.print(t);  
  Serial.print(F("°C "));  
  Serial.print(f);  
  Serial.print(F("°F Heat index: "));  
  Serial.print(hic);  
  Serial.print(F("°C "));  
  Serial.print(hif);  
  Serial.println(F("°F"));  
}
```

### Additional Ideas

- Display readings on an LCD or OLED display

### More Projects

- *Plant Monitor with Blynk*
- *Bluetooth Environmental Monitor*

## 2.4.14 PIR Motion Module (HC-SR501)

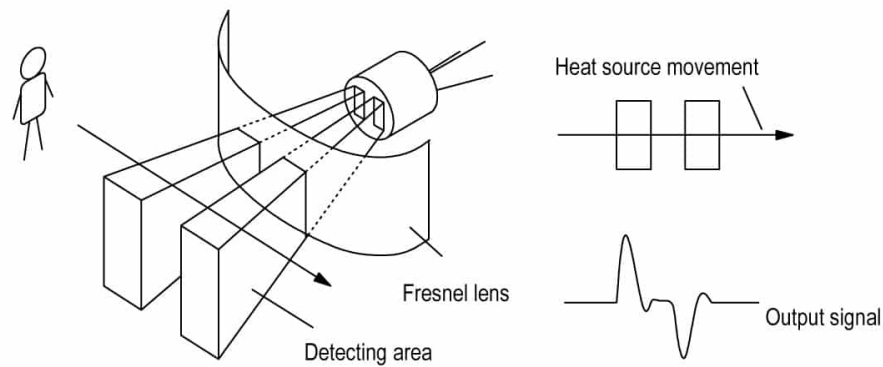


### Introduction

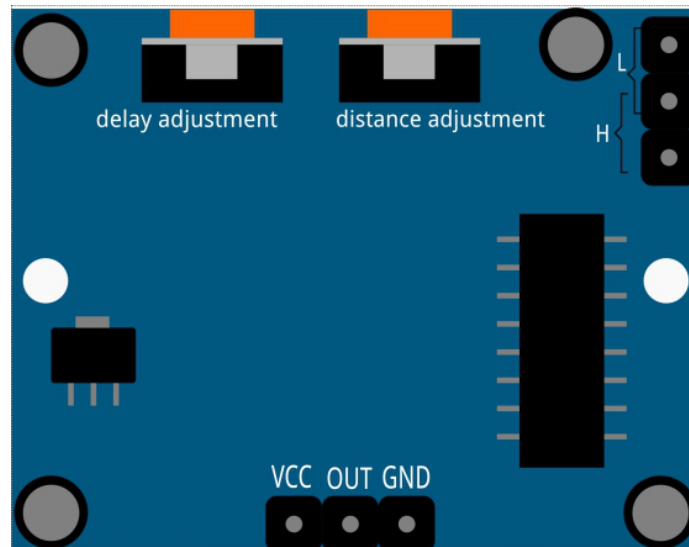
The Passive Infrared (PIR) Motion Sensor is a sensor that detects motion. It is commonly used in security systems and automatic lighting systems. The sensor has two slots that detect infrared radiation. When an object, such as a person, passes in front of the sensor, it detects a change in the amount of infrared radiation and triggers an output signal.

### Principle

The PIR sensor is split into two slots that are connected to a differential amplifier. Whenever a stationary object is in front of the sensor, the two slots receive the same amount of radiation and the output is zero. Whenever a moving object is in front of the sensor, one of the slots receives more radiation than the other, which makes the output fluctuate high or low. This change in output voltage is a result of detection of motion.



After the sensing module is wired, there is a one-minute initialization. During the initialization, module will output for 0~3 times at intervals. Then the module will be in the standby mode. Please keep the interference of light source and other sources away from the surface of the module so as to avoid the misoperation caused by the interfering signal. Even you'd better use the module without too much wind, because the wind can also interfere with the sensor.



## Distance Adjustment

Turning the knob of the distance adjustment potentiometer clockwise, the range of sensing distance increases, and the maximum sensing distance range is about 0-7 meters. If turn it anticlockwise, the range of sensing distance is reduced, and the minimum sensing distance range is about 0-3 meters.



## **Delay adjustment**

Rotate the knob of the delay adjustment potentiometer clockwise, you can also see the sensing delay increasing. The maximum of the sensing delay can reach up to 300s. On the contrary, if rotate it anticlockwise, you can shorten the delay with a minimum of 5s.

## **Two Trigger Modes**

Choosing different modes by using the jumper cap.

- H: Repeatable trigger mode, after sensing the human body, the module outputs high level. During the subsequent delay period, if somebody enters the sensing range, the output will keep being the high level.
- L: Non-repeatable trigger mode, outputs high level when it senses the human body. After the delay, the output will change from high level into low level automatically.

## **Usage**

### **Hardware components**

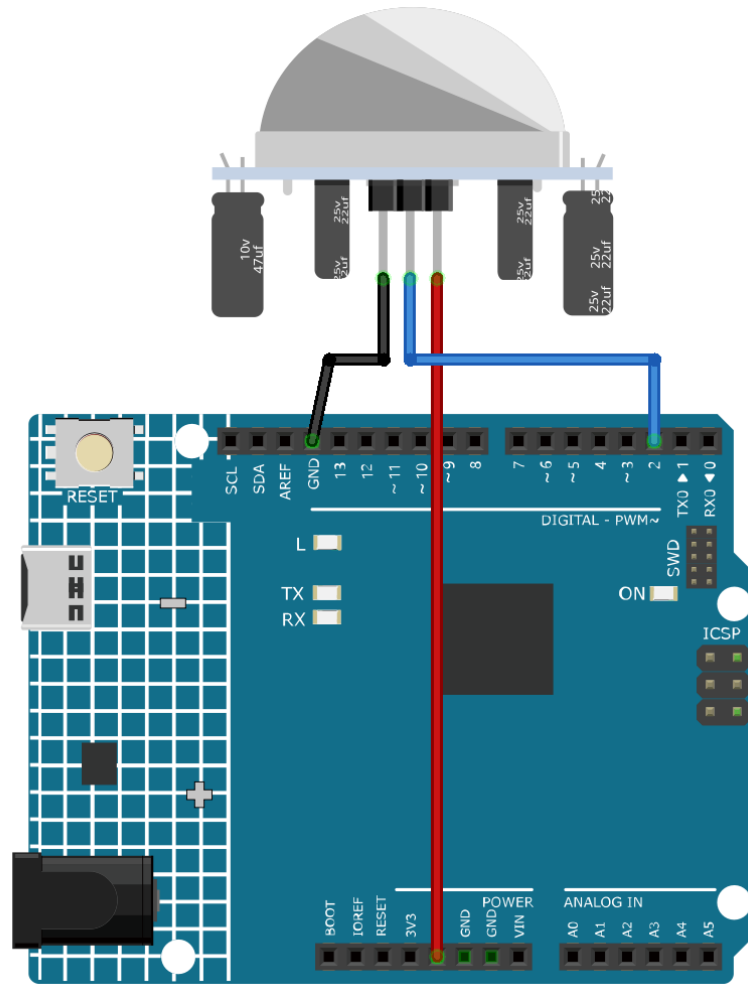
- Arduino Uno R4 or R3 board \* 1
- PIR Motion Module \* 1
- Jumper Wires

### **Circuit Assembly**

---

**Note:** The pin markings are hidden by the Fresnel lens. You can open it to view.

---



## Code

### Code explanation

1. Setting up the PIR Sensor Pin. The pin for the PIR sensor is defined as pin 2.

```
const int pirPin = 2;  
int state = 0;
```

2. Initializing the PIR Sensor. In the `setup()` function, the PIR sensor pin is set as an input. This allows the Arduino to read the state of the PIR sensor.

```
void setup() {  
  pinMode(pirPin, INPUT);  
  Serial.begin(9600);  
}
```

3. Reading from the PIR Sensor and Displaying the Results. In the `loop()` function, the state of the PIR sensor is continuously read.

```
void loop() {  
  state = digitalRead(pirPin);  
  if (state == HIGH) {  
    Serial.println("Somebody here!");  
  } else {  
    Serial.println("Monitoring...");  
    delay(100);  
  }  
}
```

If the state is HIGH, meaning motion is detected, a message “Somebody here!” is printed to the serial monitor. Otherwise, “Monitoring...” is printed.

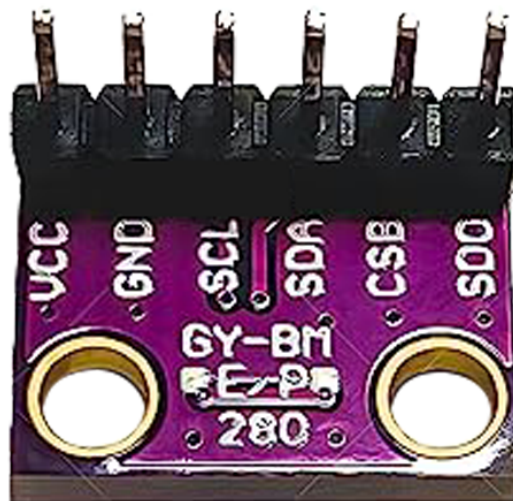
### Additional Ideas

- You can add an LED that turns on or blinks when motion is detected. This provides visual feedback in addition to the serial monitor message.
- Integrate a buzzer to sound an alert when motion is detected.

### More Projects

- *Motion triggered relay*
- *Vibration Alert System with IFTTT*

## 2.4.15 Temperature, Humidity & Pressure Sensor (BMP280)



### Introduction

A GY-BMP280-3.3 high precision atmospheric pressure sensor module is a device that can measure the air pressure and temperature with high accuracy. It can help you monitor the weather conditions and create projects that use altitude or barometric pressure data.

### Principle

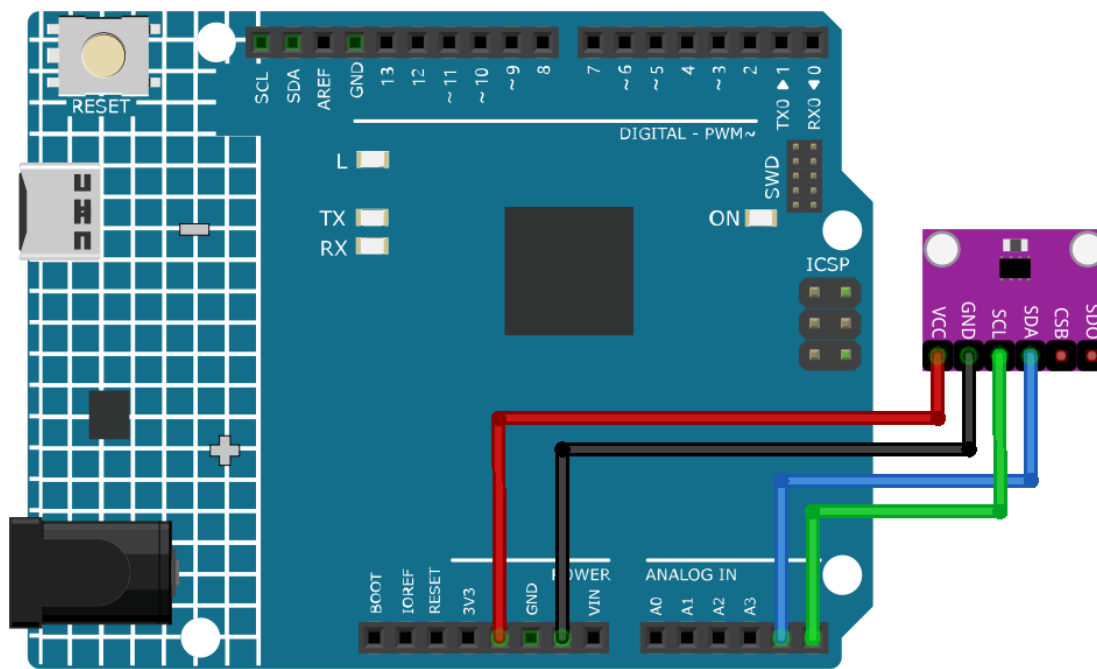
A GY-BMP280-3.3 high precision atmospheric pressure sensor module works by using a BMP280 sensor from Bosch that can measure both pressure and temperature. The BMP280 sensor has a piezoresistive pressure sensor and a thermistor inside a sealed metal chamber. The piezoresistive sensor changes its resistance according to the pressure applied to the chamber. The thermistor changes its resistance according to the temperature inside the chamber. The module has an integrated circuit that converts the resistance values into digital signals and sends them to the Arduino through either I2C or SPI interface.

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Temperature, Humidity & Pressure Sensor(GY-BMP280-3.3) \* 1
- Jumper Wires

#### Circuit Assembly



## Code

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit BMP280**” and install it.

## Code explanation

1. Including Libraries and Initialization. Necessary libraries are included and the BMP280 sensor is initialized for communication using the I2C interface.

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit BMP280**” and install it.

- Adafruit BMP280 Library: This library provides an easy-to-use interface for the BMP280 sensor, allowing the user to read temperature, pressure, and altitude.
- Wire.h: Used for I2C communication.

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
#define BMP280_ADDRESS 0x76
Adafruit_BMP280 bmp; // use I2C interface
```

2. The setup() function initializes the Serial communication, checks for the BMP280 sensor, and sets up the sensor with default settings.

```
void setup() {
  Serial.begin(9600);
  while (!Serial) delay(100);
  Serial.println(F("BMP280 test"));
  unsigned status;
  status = bmp.begin(BMP280_ADDRESS);
  // ... (rest of the setup code)
```

3. The loop() function reads data from the BMP280 sensor for temperature, pressure, and altitude. This data is printed to the Serial Monitor.

```
void loop() {
  // ... (read and print temperature, pressure, and altitude data)
  delay(2000); // 2-second delay between readings.
}
```

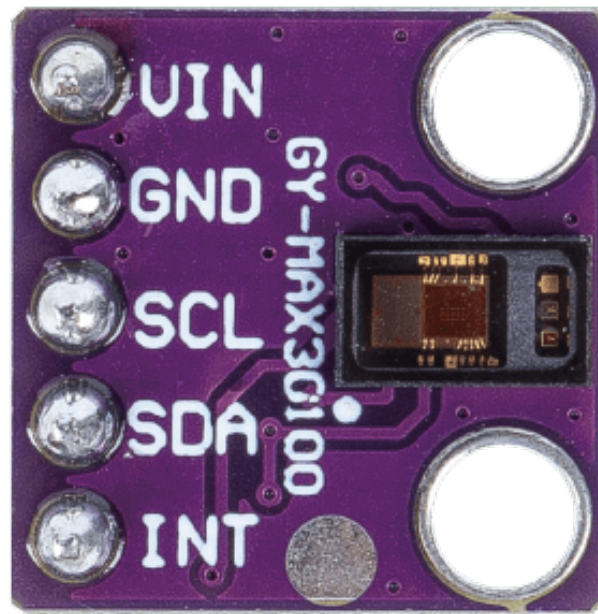
### Additional Ideas

- Integrate an LCD display module to show the readings instead of or in addition to the Serial Monitor.
- Set threshold values for temperature and pressure. Use a buzzer or LED to alert when these thresholds are exceeded.

### More Projects

- *Weather Monitor with ThingSpeak*

### 2.4.16 Pulse Oximeter and Heart Rate Sensor (MAX30102)



#### Introduction

MAX30102 is a sensor that combines a pulse oximeter and a heart rate monitor. It's an optical sensor that measures the absorbance of pulsating blood through a photodetector after emitting two wavelengths of light from two LEDs - a red and an infrared one. This particular LED colour combination is designed to allow data to be read with the tip of one's finger.

#### Principle

The MAX30102 works by shining both lights onto the finger or earlobe (or essentially anywhere where the skin isn't too thick, so both lights can easily penetrate the tissue) and measuring the amount of reflected light using a photodetector. This method of pulse detection through light is called Photoplethysmogram.

The working of MAX30102 can be divided into two parts: Heart Rate Measurement and Pulse Oximetry (measuring the oxygen level of the blood).

## Heart Rate Measurement

The oxygenated hemoglobin (HbO<sub>2</sub>) in the arterial blood has the characteristic of absorbing IR light. The redder the blood (the higher the hemoglobin), the more IR light is absorbed. As the blood is pumped through the finger with each heartbeat, the amount of reflected light changes, creating a changing waveform at the output of the photodetector. As you continue to shine light and take photodetector readings, you quickly start to get a heart-beat (HR) pulse reading.

## Pulse Oximetry

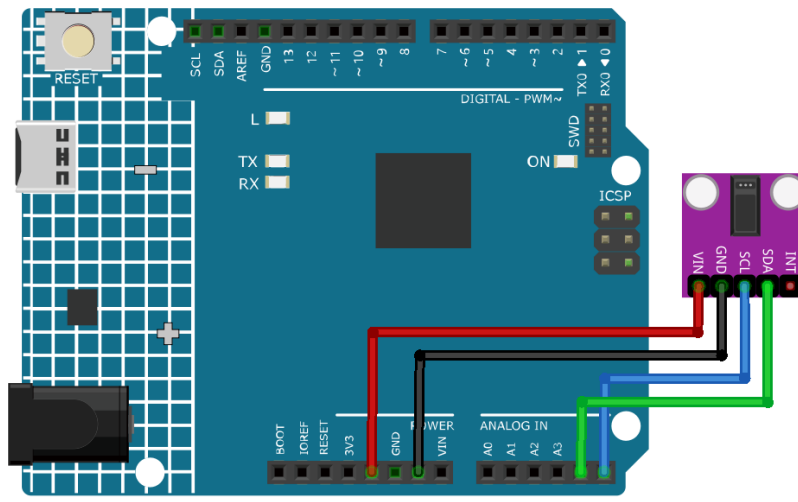
Pulse oximetry is based on the principle that the amount of RED and IR light absorbed varies depending on the amount of oxygen in your blood.

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Pulse Oximeter and Heart Rate Sensor(MAX30102) \* 1
- Jumper Wires

### Circuit Assembly



## Code

**Warning:** This sketch detects heart-rate optically. This method is tricky and prone to give false readings. So please **DO NOT** use it for actual medical diagnosis.

**Note:** To install the library, use the Arduino Library Manager and search for “**SparkFun MAX3010x**” and install it.

## Code explanation

### 1. Including Libraries & Initializing Global Variables:

The essential libraries are imported, the sensor object is instantiated, and global variables for data management are set.

---

**Note:** To install the library, use the Arduino Library Manager and search for “**SparkFun MAX3010x**” and install it.

---

```
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
MAX30105 particleSensor;
// ... (other global variables)
```

### 2. Setup Function & Sensor Initialization:

The Serial communication is initialized at a baud rate of 9600. The sensor’s connection is checked, and if successful, an initialization sequence is run. An error message is displayed if the sensor isn’t detected.

```
void setup() {
  Serial.begin(9600);
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30102 not found.");
    while (1) ; // Infinite loop if sensor not detected.
  }
  // ... (further setup)
```

### 3. Reading IR Value & Checking for Heartbeat:

The IR value, which is indicative of the blood flow, is fetched from the sensor. The `checkForBeat()` function assesses if a heartbeat is detected based on this value.

```
long irValue = particleSensor.getIR();
if (checkForBeat(irValue) == true) {
  // ... (heartbeat detected actions)
}
```

### 4. Calculating Beats Per Minute (BPM):

Upon detecting a heartbeat, the BPM is calculated based on the time difference since the last detected heartbeat. The code also ensures the BPM falls within a realistic range before updating the average.

```
long delta = millis() - lastBeat;
beatsPerMinute = 60 / (delta / 1000.0);
if (beatsPerMinute < 255 && beatsPerMinute > 20) {
  // ... (store and average BPM)
}
```

### 5. Printing Values to the Serial Monitor:

The IR value, current BPM, and average BPM are printed to the Serial Monitor. Additionally, the code checks if the IR value is too low, suggesting the absence of a finger.



```
//Print the IR value, current BPM value, and average BPM value to the serial monitor
Serial.print("IR=");
Serial.print(irValue);
Serial.print(", BPM=");
Serial.print(beatsPerMinute);
Serial.print(", Avg BPM=");
Serial.print(beatAvg);

if (irValue < 50000)
  Serial.print(" No finger?");
```

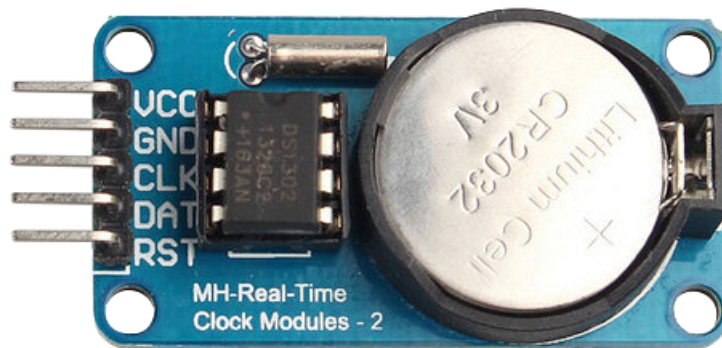
### Additional Ideas

- Add LEDs to flash with each detected beat
- Use a small OLED or LCD screen to display real-time BPM values and other relevant data.

### More Projects

- *Heart rate monitor*

## 2.4.17 Real Time Clock Module (DS1302)



### Introduction

A DS1302 real-time clock module is a device that can keep track of the date and time. It can help you create projects that need accurate timing and scheduling. It can also be used to create a digital clock with Arduino.

### Principle

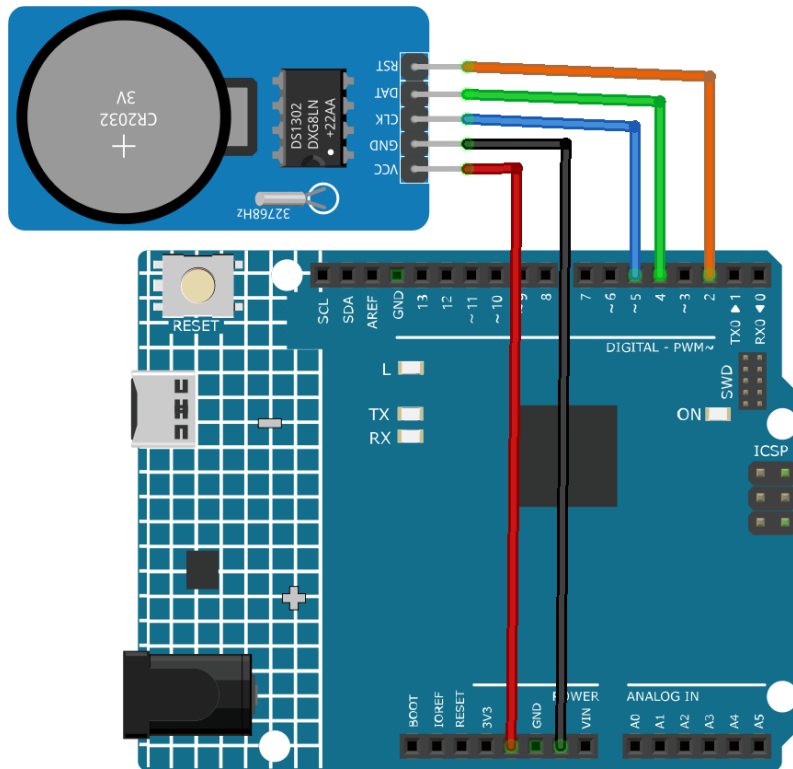
DS1302 is a trickle charging clock chip, launched by DALLAS in America. With a built-in real-time clock/calendar and a 31-byte static RAM, it can communicate with MCU through simple serial ports. The real-time clock/calendar circuit provides information about second, minute, hour, day, week, month, and year. DS1302 can automatically adjust the number of days per month and days in leap year. You can determine to use a 24-hour or 12-hour system by AM/PM selection. It can simply communicate with MCU in synchronous serial way and only needs to use three port cables: Reset (RST) cable, I/O data (SDA) cable and serial clock (SCL) cable.

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Real Time Clock Module(DS1302) \* 1
- Jumper Wires

#### Circuit Assembly



## Code

**Note:** To install the library, use the Arduino Library Manager and search for “**Rtc by Makuna**” and install it.

## Code explanation

### 1. Initialization and library inclusion

**Note:** To install the library, use the Arduino Library Manager and search for “**Rtc by Makuna**” and install it.

Here, necessary libraries are included for the DS1302 RTC module.

```
#include <ThreeWire.h>
#include <RtcDS1302.h>
```

### 2. Define pins and create RTC instance

Pins for communication are defined and an instance of the RTC is created.

```
const int IO = 4;    // DAT
const int SCLK = 5;  // CLK
const int CE = 2;    // RST

ThreeWire myWire(4, 5, 2); // IO, SCLK, CE
RtcDS1302<ThreeWire> Rtc(myWire);
```

### 3. setup() function

This function initializes the serial communication and sets up the RTC module. Various checks are made to ensure the RTC is running correctly.

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

    Serial.print("compiled: ");
    Serial.print(__DATE__);
    Serial.println(__TIME__);

    Rtc.Begin();

    RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
    printDateTime(compiled);
    Serial.println();

    if (!Rtc.IsDateTimeValid()) {
        // Common Causes:
        // 1) first time you ran and the device wasn't running yet
        // 2) the battery on the device is low or even missing

        Serial.println("RTC lost confidence in the DateTime!");
    }
}
```

(continues on next page)

(continued from previous page)

```

    Rtc.SetDateTime(compiled);
}

if (Rtc.GetIsWriteProtected()) {
    Serial.println("RTC was write protected, enabling writing now");
    Rtc.SetIsWriteProtected(false);
}

if (!Rtc.GetIsRunning()) {
    Serial.println("RTC was not actively running, starting now");
    Rtc.SetIsRunning(true);
}

RtcDateTime now = Rtc.GetDateTime();
if (now < compiled) {
    Serial.println("RTC is older than compile time! (Updating DateTime)");
    Rtc.SetDateTime(compiled);
} else if (now > compiled) {
    Serial.println("RTC is newer than compile time. (this is expected)");
} else if (now == compiled) {
    Serial.println("RTC is the same as compile time! (not expected but all is fine)
↪");
}
}

```

#### 4. loop() function

This function periodically fetches the current date and time from the RTC and prints it on the serial monitor. It also checks if the RTC is still maintaining a valid date and time.

```

void loop() {
    RtcDateTime now = Rtc.GetDateTime();

    printDateTime(now);
    Serial.println();

    if (!now.IsValid()) {
        // Common Causes:
        // 1) the battery on the device is low or even missing and the power line
↪was disconnected
        Serial.println("RTC lost confidence in the DateTime!");
    }

    delay(5000); // five seconds
}

```

#### 5. Date and time printing function

A helper function that takes a RtcDateTime object and prints the formatted date and time to the serial monitor.

```

void printDateTime(const RtcDateTime& dt) {
    char datestring[20];

```

(continues on next page)

(continued from previous page)

```

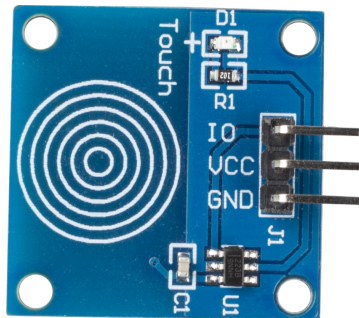
snprintf_P(datestring,
            countof(datestring),
            PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
            dt.Month(),
            dt.Day(),
            dt.Year(),
            dt.Hour(),
            dt.Minute(),
            dt.Second());
Serial.print(datestring);
}

```

### Additional Ideas

- Display time on LCD or serial monitor
- Schedule events/alarms at certain times

## 2.4.18 Touch Sensor Module



### Introduction

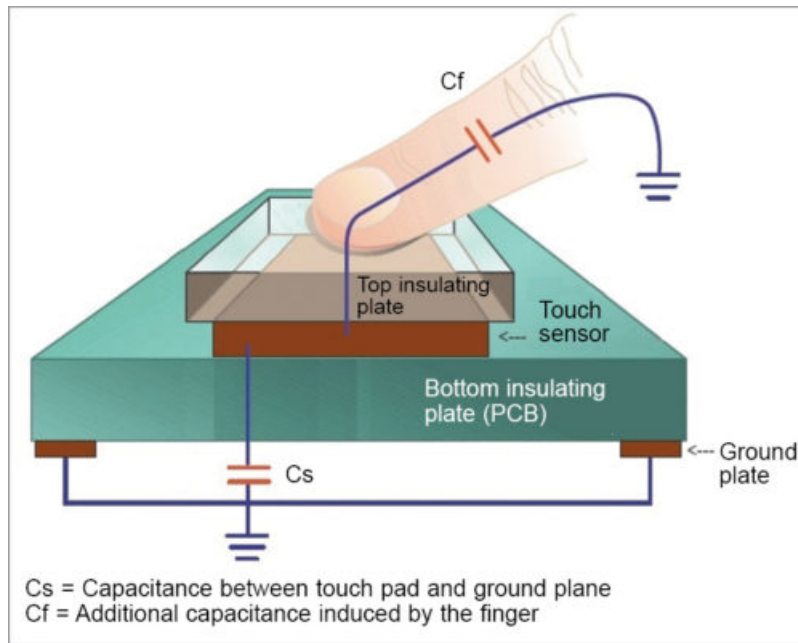
The Touch switch sensor (also called touch button or touch switch) is widely used to control devices (e.g. touchable lamp). It has the same functionality as a button. It is used instead of the button on many new devices because it makes the product look neat.

### Principle

This module is a capacitive touch switch module based on a touch sensor IC (TTP223B). In the normal state, the module outputs a low level with low power consumption; when a finger touches the corresponding position, the module outputs a high level and becomes low level again after the finger is released.

Here is how the capacitive touch switch works:

A capacitive touch switch has different layers—top insulating face plate followed by touch plate, another insulating layer and then ground plate.



In practice, a capacitive sensor can be made on a double-sided PCB by regarding one side as the touch sensor and the opposite side as ground plate of the capacitor. When power is applied across these plates, the two plates get charged. In equilibrium state, the plates have the same voltage as the power source.

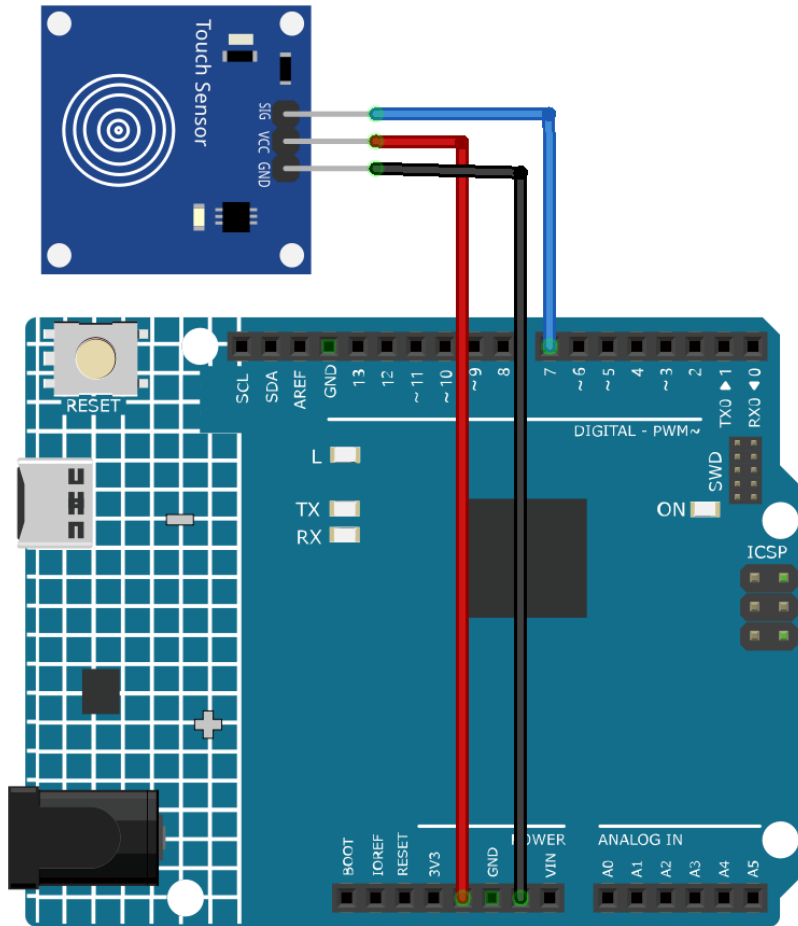
The touch detector circuit has an oscillator whose frequency is dependent on capacitance of the touchpad. When a finger is moved close to the touchpad, additional capacitance causes frequency of this internal oscillator to change. The detector circuit tracks oscillator frequency at timed intervals, and when the shift crosses the threshold change, the circuit triggers a key-press event.

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Touch Sensor Module \* 1
- Jumper Wires

### Circuit Assembly



## Code

### Code explanation

1. Setting up the necessary variables. We start by defining the pin number where the touch sensor is connected.

```
const int sensorPin = 7;
```

2. Initialization in the `setup()` function. Here, we specify that the sensor pin will be used for input, the built-in LED will be used for output, and we start the serial communication to allow messages to be sent to the serial monitor.

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

3. Continuously, the Arduino checks if the touch sensor is activated. If touched, it turns on the LED and sends a "Touch detected!" message. If not touched, it turns off the LED and sends a "No touch detected..." message. A delay is introduced to prevent the sensor from being read too quickly.

```
void loop() {  
  if (digitalRead(sensorPin) == 1) {  
    digitalWrite(LED_BUILTIN, HIGH);  
    Serial.println("Touch detected!");  
  } else {  
    digitalWrite(LED_BUILTIN, LOW);  
    Serial.println("No touch detected...");  
  }  
  delay(100);  
}
```

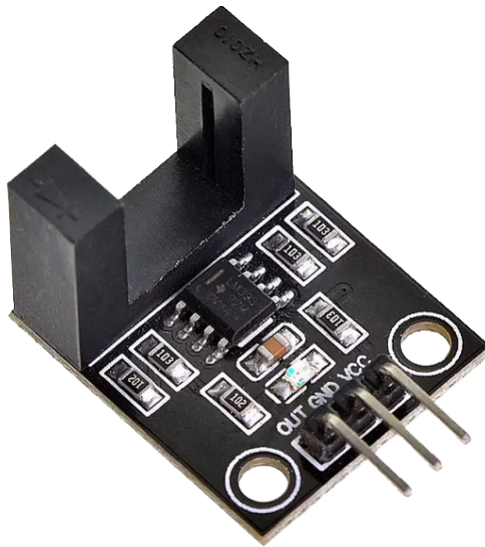
### Additional Ideas

- Use different LED colors to indicate touch
- The touch sensor could be used to control more complex elements, like a motor or a relay.

### More Projects

- *Touch toggle light*

## 2.4.19 Infrared Speed Sensor Module



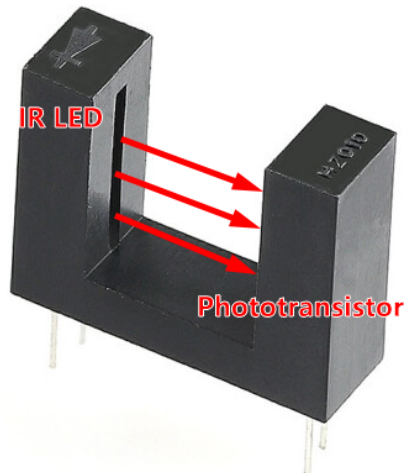


## Introduction

The Infrared Speed Sensor Module is an IR counter that has an IR transmitter and receiver. If any obstacle is placed between these sensors, a signal is sent to the microcontroller. The module can be used in association with a microcontroller for motor speed detection, pulse count, position limit, etc.

## Principle

The Infrared Speed Sensor Module has 1 H2010 photocell, which consists of a phototransistor and an infrared light emitter packaged in a 10 cm wide black plastic housing.



When operating, the infrared light-emitting diode continuously emits infrared light (invisible light), and the photosensitive triode will conduct if it receives it.

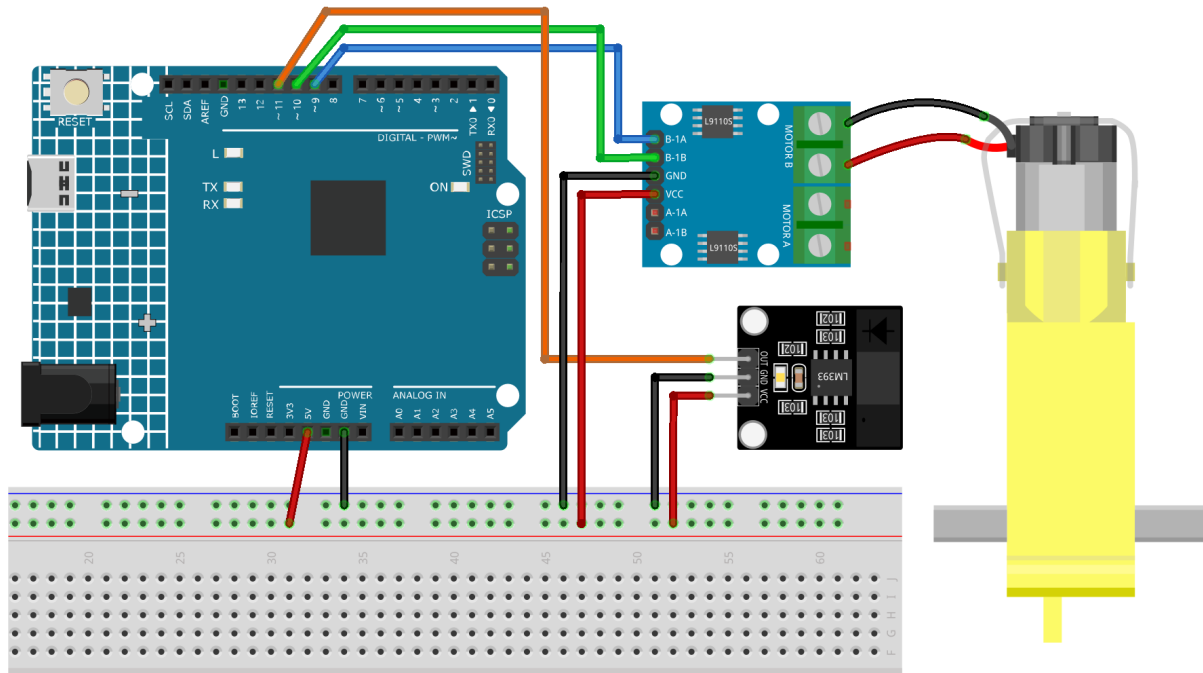


## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Infrared Speed Sensor Module \* 1
- Jumper Wires

### Circuit Assembly



## Code

### Code explanation

1. Setting up the pins and initializing variables. Here, we define the pins for the motor and the speed sensor. We also initialize variables that will be used to measure and calculate the speed of the motor.

```
// Define the sensor and motor pins
const int sensorPin = 11;
const int motorB_1A = 9;
const int motorB_2A = 10;

// Define variables for measuring speed
unsigned long start_time = 0;
unsigned long end_time = 0;
int steps = 0;
float steps_old = 0;
float temp = 0;
float rps = 0;
```

2. Initialization in the setup() function. This section sets up the serial communication, configures the pins' modes, and sets the initial motor speed.

```
void setup() {
  Serial.begin(9600);
  pinMode(sensorPin, INPUT);
  pinMode(motorB_1A, OUTPUT);
  pinMode(motorB_2A, OUTPUT);
  analogWrite(motorB_1A, 160);
```

(continues on next page)

(continued from previous page)

```
analogWrite(motorB_2A, 0);  
}
```

3. Measuring the motor's speed in the loop() function. In this segment, the motor's steps are measured for a duration of 1 second. These steps are then used to calculate the revolutions per second (rps), which is then printed to the serial monitor.

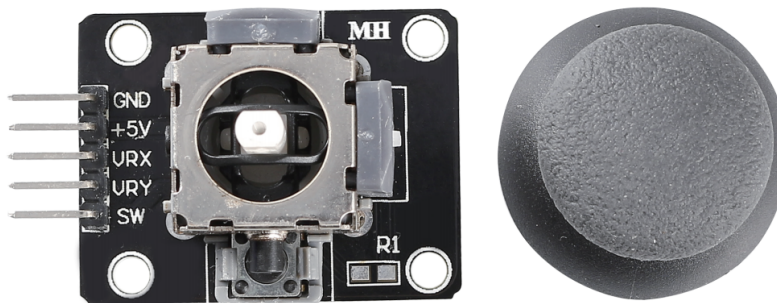
millis() returns the number of milliseconds passed since the Arduino board began running the current program.

```
void loop() {  
  start_time = millis();  
  end_time = start_time + 1000;  
  while (millis() < end_time) {  
    if (digitalRead(sensorPin)) {  
      steps = steps + 1;  
      while (digitalRead(sensorPin))  
        ;  
    }  
  }  
  temp = steps - steps_old;  
  steps_old = steps;  
  rps = (temp / 20);  
  Serial.print("rps:");  
  Serial.println(rps);  
}
```

### Additional Ideas

- Display the rps on an LCD screen for a more user-friendly interface.

### 2.4.20 Joystick Module



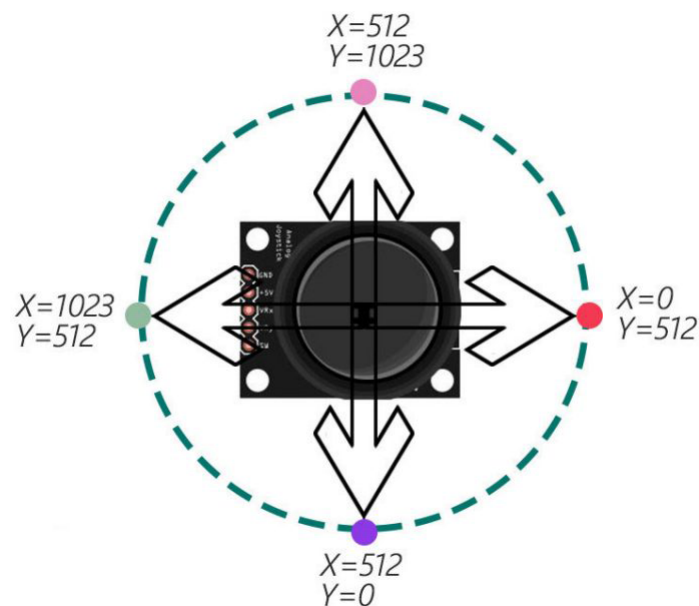
### Introduction

A joystick module is a device that can measure the movement of a knob in two directions: horizontal (X-axis) and vertical (Y-axis). A joystick module can be used to control various things such as games, robots, cameras, etc.

### Principle

Joystick operates based on the resistance change of two potentiometers (usually 10-kilo ohms). By changing resistance in x and y directions, Arduino receives varying voltages which are interpreted to x and y coordinates. The processor needs an ADC unit to change the joystick's analog values into digital values and perform necessary processing.

Arduino boards have six 10-bits ADC channels. It means the Arduino's reference voltage (5 volts) is divided to 1024 segments. When joystick moves along the x-axis, the ADC value rises from 0 to 1023, with the value 512 in the middle. The image below displays the ADC approximate value based on the joystick position.

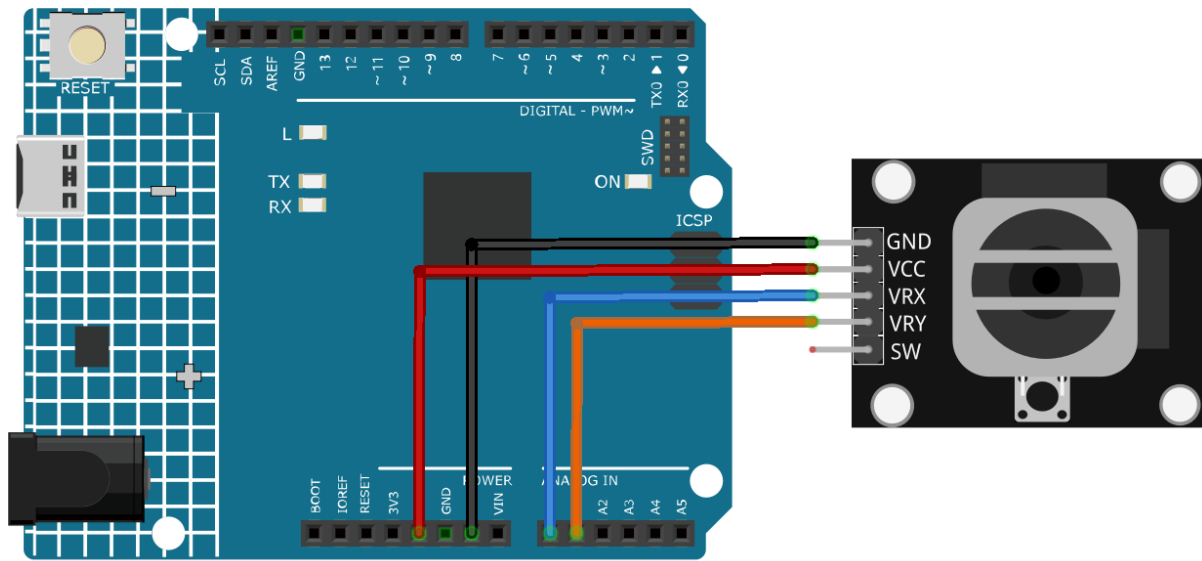


### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Joystick Module \* 1
- Jumper Wires

#### Circuit Assembly



## Code

### Code explanation

1. Setting up the joystick pins. Here, we define which analog pins the X and Y axes of the joystick are connected to.

```
const int xPin = A0;
const int yPin = A1;
```

2. Initialization in the `setup()` function. This section sets up the serial communication, allowing us to send and receive messages from the Arduino through the serial monitor.

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

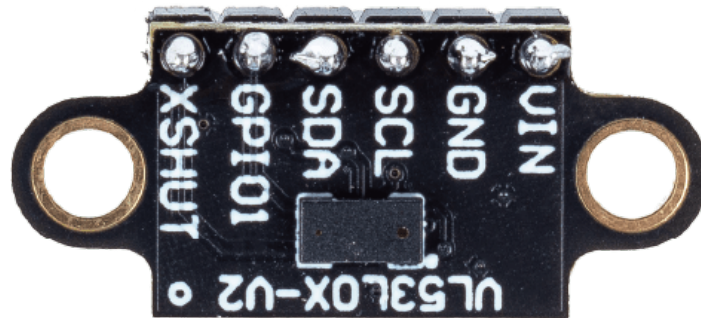
3. Reading the joystick values in the `loop()` function. Continuously, the Arduino reads the X and Y values from the joystick and prints them to the serial monitor. There's a short delay after each print to make the readings more readable and to avoid overwhelming the serial monitor.

```
void loop() {
  Serial.print("X: ");
  Serial.print(analogRead(xPin));
  Serial.print(" | Y: ");
  Serial.println(analogRead(yPin));
  delay(50);
}
```

### Additional Ideas

- Use the joystick values to control a servo motor, making it move in response to joystick movements.

#### 2.4.21 Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)



### Introduction

The VL53L0X module is a Time of Flight (ToF) ranging sensor that can accurately measure distances up to 2 meters using laser technology. It is a multi-sensor module with an integrated laser emitter, detector, and microcontroller. The module has all required components such as pull-up resistors and capacitors. It can handle about 50 - 1200 mm of range distance.

### Principle

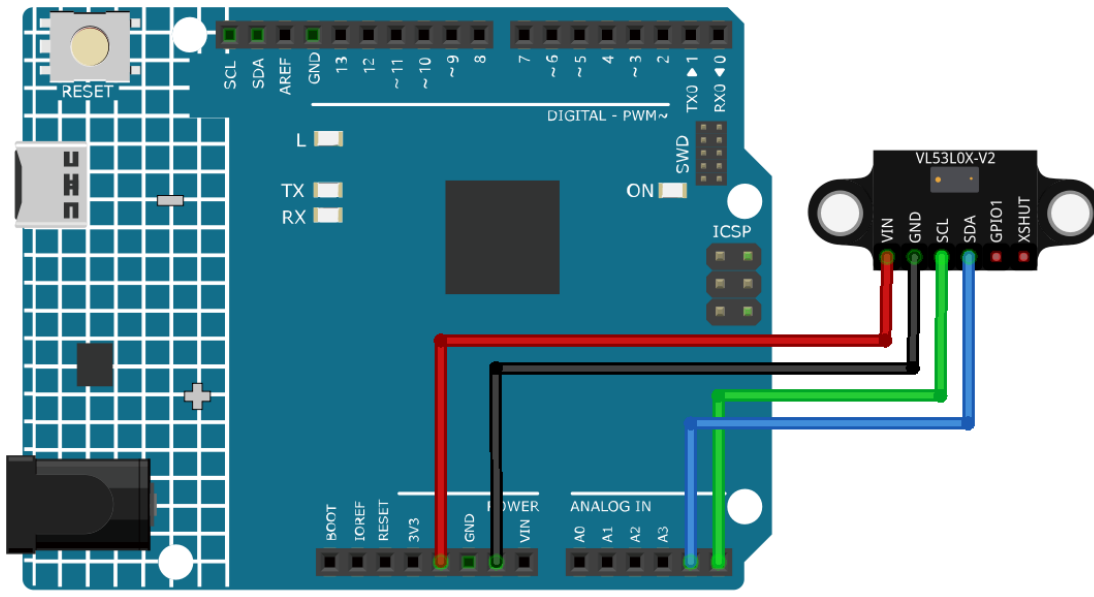
The VL53L0X module works on the principle of Time of Flight (ToF). It sends out a laser pulse and measures the time it takes for the pulse to bounce back. The time it takes for the pulse to return is proportional to the distance between the sensor and the object. The module uses a single photon avalanche diode (SPAD) array to detect the reflected light from the object. The SPAD array is capable of detecting even a single photon of light. The module also has an integrated microcontroller that processes the data from the SPAD array and calculates the distance between the sensor and the object.

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Time of Flight Micro-LIDAR Distance Sensor \* 1
- Jumper Wires

#### Circuit Assembly



## Code

**Note:** To install the library, use the Arduino Library Manager and search for “Adafruit\_VL53L0X” and install it.

## Code explanation

1. Including the necessary library and initializing the sensor object. We start by including the library for the VL53L0X sensor and creating an instance of the Adafruit\_VL53L0X class.

**Note:** To install the library, use the Arduino Library Manager and search for “Adafruit\_VL53L0X” and install it.

```
#include <Adafruit_VL53L0X.h>
Adafruit_VL53L0X lox = Adafruit_VL53L0X();
```

2. Initialization in the setup() function. Here, we set up serial communication and initialize the distance sensor. If the sensor can't be initialized, the program halts.

```
void setup() {
  Serial.begin(115200);
  while (!Serial) {
    delay(1);
  }
  Serial.println("Adafruit VL53L0X test");
  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1)
      ;
  }
}
```

(continues on next page)

(continued from previous page)

```

}
Serial.println(F("VL53L0X API Simple Ranging example\n\n"));
}

```

3. Capturing and displaying the measurements in the loop() function. Continuously, the Arduino captures a distance measurement using the rangingTest() method. If the measurement is valid, it's printed to the serial monitor.

```

void loop() {
  VL53L0X_RangingMeasurementData_t measure;
  Serial.print("Reading a measurement... ");
  lox.rangingTest(&measure, false);
  if (measure.RangeStatus != 4) {
    Serial.print("Distance (mm): ");
    Serial.println(measure.RangeMilliMeter);
  } else {
    Serial.println(" out of range ");
  }
  delay(100);
}

```

## Additional Ideas

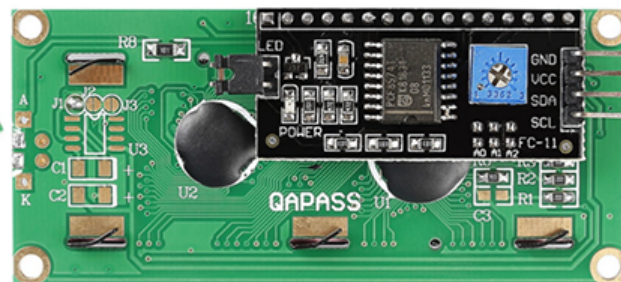
- Integrate the sensor with a display (like an OLED) to show the distance measurements.
- Use the distance data to trigger other components, such as LEDs or buzzers, when an object comes within a specific range.

## More Projects

- *ToF distance monitor*

## Display

### 2.4.22 I2C LCD 1602





## Introduction

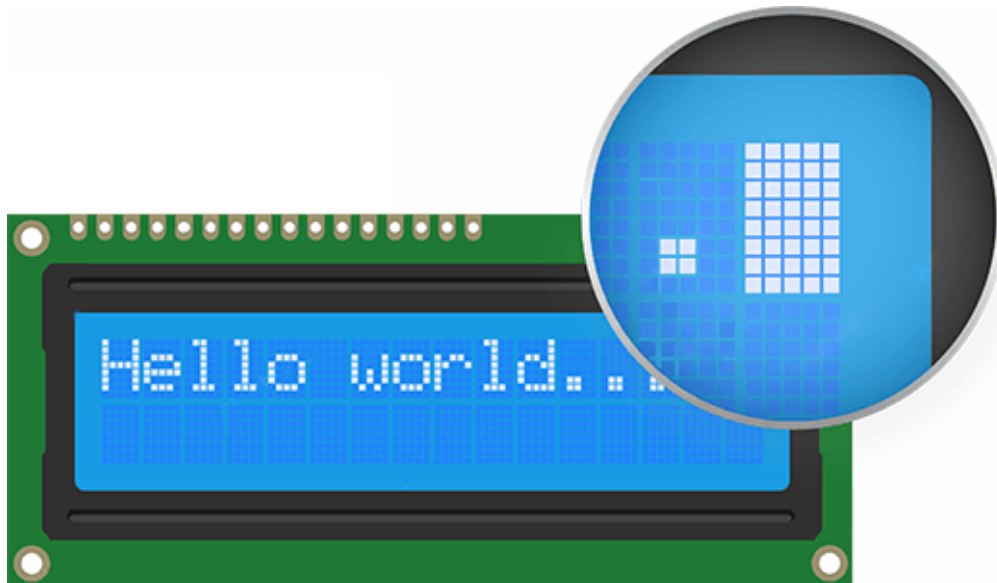
An I2C LCD1602 is a device that can display text and characters on a 16x2 (16 columns and 2 rows) liquid crystal display (LCD) using the I2C protocol. You can use an I2C LCD1602 to show information from your Arduino projects, such as sensor readings, messages, menus, etc. The I2C module has a built-in PCF8574 I2C chip that converts I2C serial data to parallel data for the LCD display.

- 

## Principle

An I2C LCD1602 consists of a normal LCD1602 and an I2C module that is attached to the back of the LCD. The I2C module is a chip that can expand the I/O ports of the Arduino using the I2C protocol. The I2C protocol is a serial communication protocol that uses two wires: SDA (serial data) and SCL (serial clock). The I2C protocol allows multiple devices to communicate with each other using only two wires and unique addresses.

The I2C module converts the signals from the Arduino into commands for the LCD. The LCD has 16x2 cells that can display characters or symbols. Each cell consists of 5x8 dots that can be turned on or off by applying voltage. The LCD can display different characters or symbols by turning on or off different combinations of dots.



## I2C Address

The default address is basically 0x27, in a few cases it may be 0x3F.

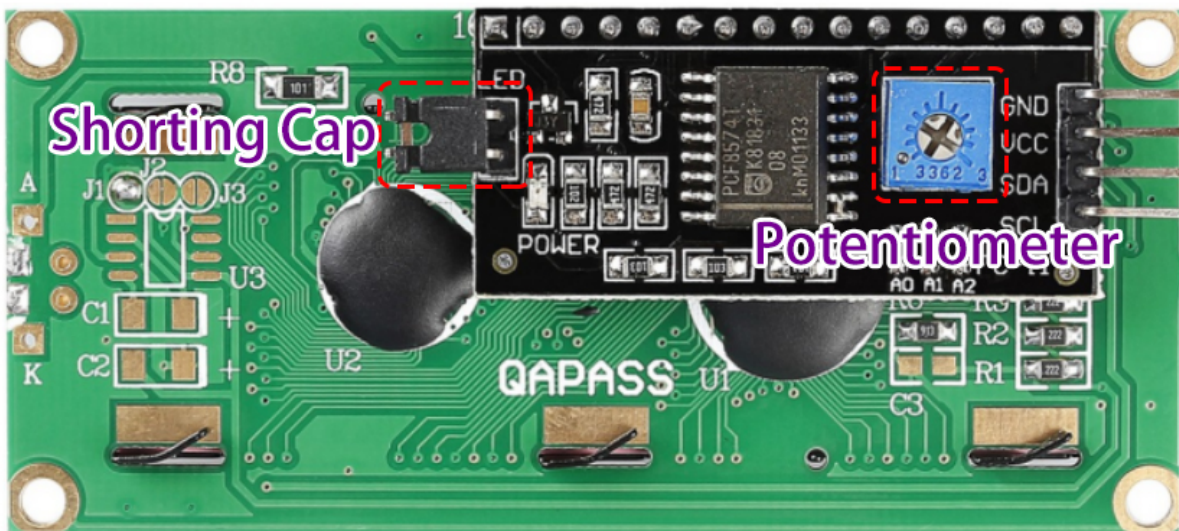
Taking the default address of 0x27 as an example, the device address can be modified by shorting the A0/A1/A2 pads; in the default state, A0/A1/A2 is 1, and if the pad is shorted, A0/A1/A2 is 0.

## Slave Address

0	0	1	0	0	A2	A1	A0	
0	0	1	0	0	1	1	1	0x27
0	0	1	0	0	1	1	0	0x26
0	0	1	0	0	1	0	1	0x25
0	0	1	0	0	0	1	1	0x23
.....								
0	0	1	0	0	0	0	0	0x20

### Backlight/Contrast

Backlight can be enabled by jumper cap, unplug the jumper cap to disable the backlight. The blue potentiometer on the back is used to adjust the contrast (the ratio of brightness between the brightest white and the darkest black).



- **Shorting Cap:** Backlight can be enabled by this cap, unplug this cap to disable the backlight.
- **Potentiometer:** It is used to adjust the contrast (the clarity of the displayed text), which is increased in the clockwise direction and decreased in the counterclockwise direction.

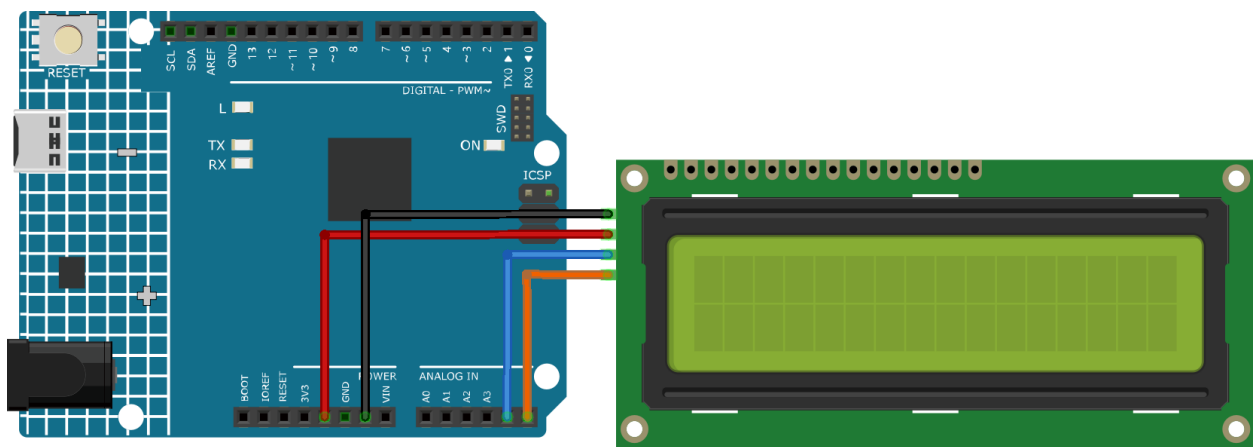
**Note:** After wiring the LCD, you should turn on the Arduino and adjust the contrast by rotating the potentiometer on the I2C module until the first row of rectangles appear to ensure proper LCD operation.

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- I2C LCD1602 \* 1
- Jumper Wires

### Circuit Assembly



## Code

**Note:** To install the library, use the Arduino Library Manager and search for “**LiquidCrystal I2C**” and install it.

### Code explanation

1. Library Inclusion and LCD Initialization: The LiquidCrystal I2C library is included to provide functions and methods for LCD interfacing. Following that, an LCD object is created using the LiquidCrystal\_I2C class, specifying the I2C address, number of columns, and number of rows.

**Note:** To install the library, use the Arduino Library Manager and search for “**LiquidCrystal I2C**” and install it.

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

2. Setup Function: The setup() function is executed once when the Arduino starts. In this function, the LCD is initialized, cleared, and the backlight is turned on. Then, two messages are displayed on the LCD.

```
void setup() {  
  lcd.init();           // initialize the LCD  
  lcd.clear();          // clear the LCD display  
  lcd.backlight();      // Make sure backlight is on  
  
  // Print a message on both lines of the LCD.  
  lcd.setCursor(2, 0);  //Set cursor to character 2 on line 0  
  lcd.print("Hello world!");  
  
  lcd.setCursor(2, 1);  //Move cursor to character 2 on line 1  
  lcd.print("LCD Tutorial");  
}
```

### Additional Ideas

- Integrate a temperature sensor and display the current room temperature on the LCD.

### More Projects

- *Potentiometer scale value*
- *Bluetooth LCD*

### 2.4.23 OLED Display Module



## Introduction

An OLED (Organic Light-Emitting Diode) display module is a device that can display text, graphics and images on a thin and flexible screen using organic materials that emit light when electric current is applied.

The main advantage of an OLED Display is that it emits its own light and doesn't need another source of backlight. Due to this, OLED Displays often have better contrast, brightness and viewing angles when compared to LCD displays.

Another important feature of OLED Displays is deep black levels. Since each pixel emits its own light in an OLED Display, to produce black color, the individual pixel can be turned OFF.

Due to lower power consumption (only pixels which are lit up draw current), OLED displays are also popular in battery operated devices like Smart Watches, Health Trackers and other wearables.

## Principle

An OLED display module consists of an OLED panel and an OLED driver chip that is mounted on the back of the module. The OLED panel is made of many tiny pixels that can produce different colors of light. Each pixel consists of several layers of organic materials sandwiched between two electrodes (anode and cathode). When electric current flows through the electrodes, the organic materials emit light of different wavelengths depending on their composition.

The OLED driver chip is a chip that can control the pixels of the OLED panel using a serial communication protocol called I2C (Inter-Integrated Circuit).

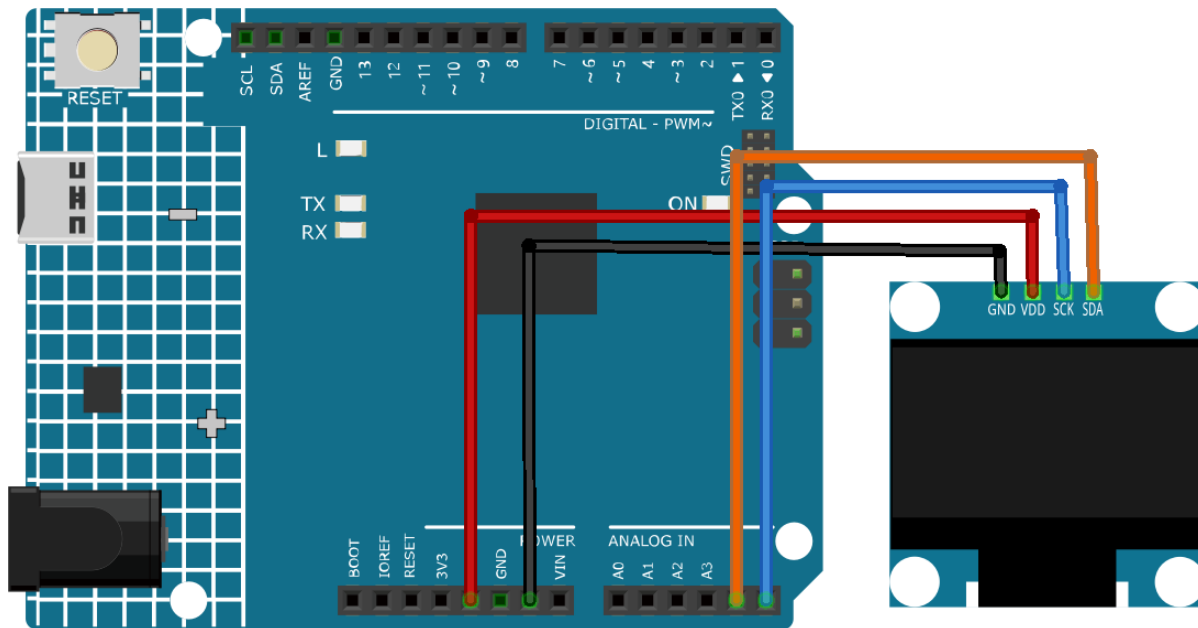
The OLED driver chip converts the signals from the Arduino into commands for the OLED panel. The Arduino can send data to the OLED driver chip using a library that can control the I2C protocol. One such library is the Adafruit SSD1306 library<sup>1</sup>. With this library, you can initialize the OLED display module, set the brightness level, print text, graphics or images, etc.

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- OLED Display Module \* 1
- Jumper Wires

### Circuit Assembly



## Code

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit SSD1306**” and “**Adafruit GFX**” and install it.

## Code explanation

1. **Library Inclusion and Initial Definitions:** The necessary libraries for interfacing with the OLED are included. Following that, definitions regarding the OLED’s dimensions and I2C address are provided.
  - **Adafruit SSD1306:** This library is designed to help with the interfacing of the SSD1306 OLED display. It provides methods to initialize the display, control its settings, and display content.
  - **Adafruit GFX Library:** This is a core graphics library for displaying text, producing colors, drawing shapes, etc., on various screens including OLEDs.

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit SSD1306**” and “**Adafruit GFX**” and install it.

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

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

(continues on next page)

(continued from previous page)

```
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
```

2. **Bitmap Data:** Bitmap data for displaying a custom icon on the OLED screen. This data represents an image in a format that the OLED can interpret.

You can use this online tool called that can turn your image into an array.

The PROGMEM keyword denotes that the array is stored in the program memory of the Arduino microcontroller. Storing data in program memory (PROGMEM) instead of RAM can be helpful for large amounts of data, which would otherwise take up too much space in RAM.

```
static const unsigned char PROGMEM sunfounderIcon[] = {...};
```

3. **Setup Function (Initialization and Display):** The setup() function initializes the OLED and displays a series of patterns, texts, and animations.

```
void setup() {
  ... // Serial initialization and OLED object initialization
  ... // Displaying various text, numbers, and animations
}
```

### Additional Ideas

- Use buttons to change the displayed messages or toggle between different patterns and animations.
- Display sensor readings (like temperature or humidity) on the OLED in real-time.

### More Projects

- *Heart rate monitor*
- *ToF distance monitor*
- *Bluetooth OLED*

## 2.4.24 Traffic Light Module



### Introduction

The traffic light module is a small device that can display red, yellow and green lights, just like a real traffic light. It can be used to make a traffic light system model or to learn how to control LEDs with Arduino. It is featured with its small size, simple wiring, targeted, and custom installation. It can be connected PWM pin to control the brightness of the LED.

### Principle

The traffic light module can be controlled in two primary ways. The more straightforward method involves using digital inputs from the Arduino, where a HIGH or LOW signal directly turns the corresponding LED on or off. Alternatively, PWM (pulse-width modulation) can be used, especially when varying the brightness of the LED is desired. PWM is a technique where the duty cycle of a digital signal is changed to modulate the brightness of the LED. A duty cycle represents the percentage of time that a signal remains on during a specific period. For instance, a 50% duty cycle implies the signal is active for half the duration and inactive for the remainder. Adjusting the duty cycle allows for the LED's brightness modulation.

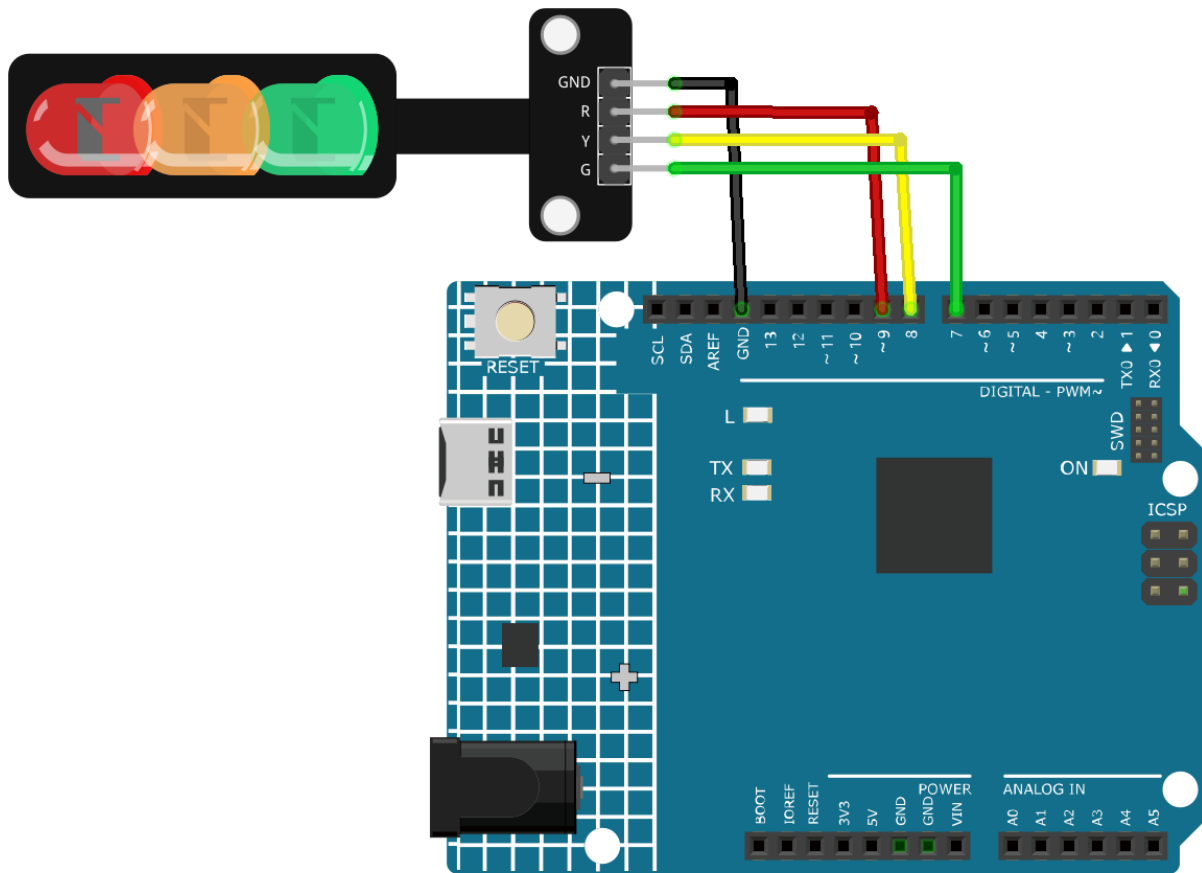
### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Traffic Light Module \* 1
- Jumper Wires

#### Circuit Assembly





## Code

### Code explanation

1. Before any operations, we define constants for the pins where LEDs are connected. This makes our code easier to read and modify.

```
const int rledPin = 9; //red
const int yledPin = 8; //yellow
const int gledPin = 7; //green
```

2. Here, we specify the pin modes for our LED pins. They are all set to OUTPUT because we intend to send voltage to them.

```
void setup() {
  pinMode(rledPin, OUTPUT);
  pinMode(yledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);
}
```

3. This is where our traffic light cycle logic is implemented. The sequence of operations is:

- Turn the green LED on for 5 seconds.
- Blink the yellow LED three times (each blink lasts for 0.5 seconds).

- Turn the red LED on for 5 seconds.

```
void loop() {  
  digitalWrite(gledPin, HIGH);  
  delay(5000);  
  digitalWrite(gledPin, LOW);  
  
  digitalWrite(yledPin, HIGH);  
  delay(500);  
  digitalWrite(yledPin, LOW);  
  delay(500);  
  digitalWrite(yledPin, HIGH);  
  delay(500);  
  digitalWrite(yledPin, LOW);  
  delay(500);  
  digitalWrite(yledPin, HIGH);  
  delay(500);  
  digitalWrite(yledPin, LOW);  
  delay(500);  
  
  digitalWrite(rledPin, HIGH);  
  delay(5000);  
  digitalWrite(rledPin, LOW);  
}
```

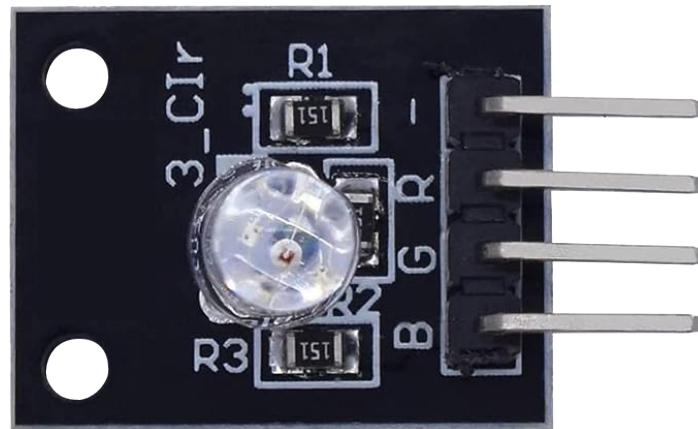
### Additional Ideas

- Integrate a buzzer to give sound alerts during the change from green to red, helping visually impaired individuals.

### More Projects

- *Touch toggle light*
- *Remote Relay Controller with Blynk*
- *Bluetooth Voice-control Relay*
- *Bluetooth Traffic Light*

### 2.4.25 RGB Module



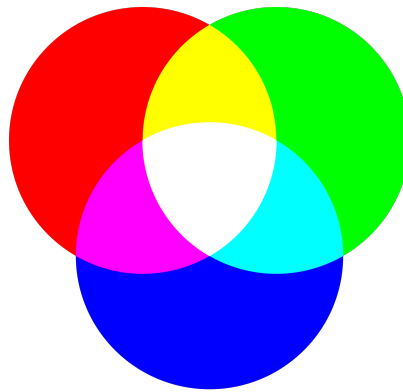
#### Introduction

The RGB Full Color LED module emits a range of colors by mixing red, green, and blue light. Each color is adjusted by using PWM. It can be used to create colorful lighting effects or to learn how to use PWM (pulse-width modulation) with Arduino.

#### Principle

The RGB MODULE works by using a full-color LED that uses R, G, and B pins with adjustable PWM voltage input. Colors from the LED can be combined. For example, mix blue light and green light give cyan light, red light and green light give yellow light. This is called “The additive method of color mixing”.

- [Additive color - Wikipedia](#)



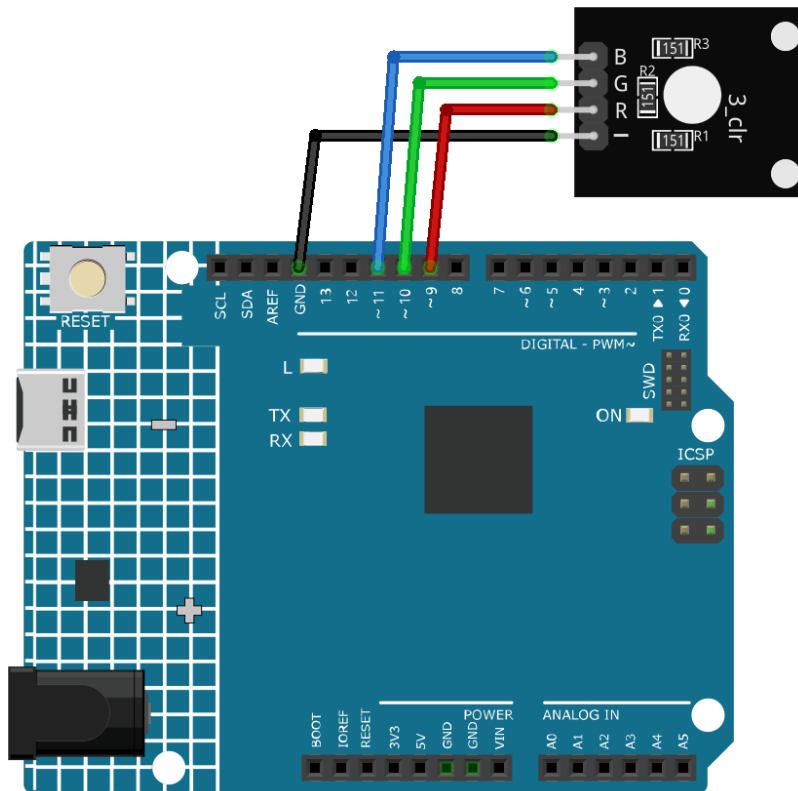
Based on this method, we can use the three primary colors to mix the visible light of any color according to different proportions. For example, orange can be produced by more red and less green. The strength of the primary colors (red, blue, green) is adjusted in order to achieve full color mixing effect. PWM is a technique where the duty cycle of a digital signal is modified, adjusting the percentage of time that the signal remains active within a given period. By changing the duty cycle, we can make the LED appear brighter or dimmer.

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- RGB Module \* 1
- Jumper Wires

#### Circuit Assembly



### Code

#### Code explanation

1. The first segment of the code declares and initializes the pins to which each color channel of the RGB LED module is connected.

```
const int rledPin = 9; // pin connected to the red color channel
const int gledPin = 10; // pin connected to the green color channel
const int bledPin = 11; // pin connected to the blue color channel
```

2. The setup() function initializes these pins as OUTPUT. This means we are sending signals OUT from these pins to the RGB LED module.

```
void setup() {
  pinMode(rledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);
}
```

(continues on next page)

(continued from previous page)

```
pinMode(bledPin, OUTPUT);
}
```

3. In the loop() function, the setColor() function is called with different parameters to display different colors. The delay() function is used after setting each color to pause for 1000 milliseconds (or 1 second) before moving on to the next color.

```
void loop() {
  setColor(255, 0, 0); // Set RGB LED color to red
  delay(1000);
  setColor(0, 255, 0); // Set RGB LED color to green
  delay(1000);
  // The rest of the color sequence...
}
```

4. The setColor() function uses the analogWrite() function to adjust the brightness of each color channel on the RGB LED module. The analogWrite() function employs Pulse Width Modulation (PWM) to simulate varying voltage outputs. By controlling the PWM duty cycle (the percentage of time a signal is HIGH within a fixed period), the brightness of each color channel can be controlled, allowing the mixing of various colors.

```
void setColor(int R, int G, int B) {
  analogWrite(rledPin, R); // Use PWM to control the brightness of the red color_
  ↪ channel
  analogWrite(gledPin, G); // Use PWM to control the brightness of the green color_
  ↪ channel
  analogWrite(bledPin, B); // Use PWM to control the brightness of the blue color_
  ↪ channel
}
```

### Additional Ideas

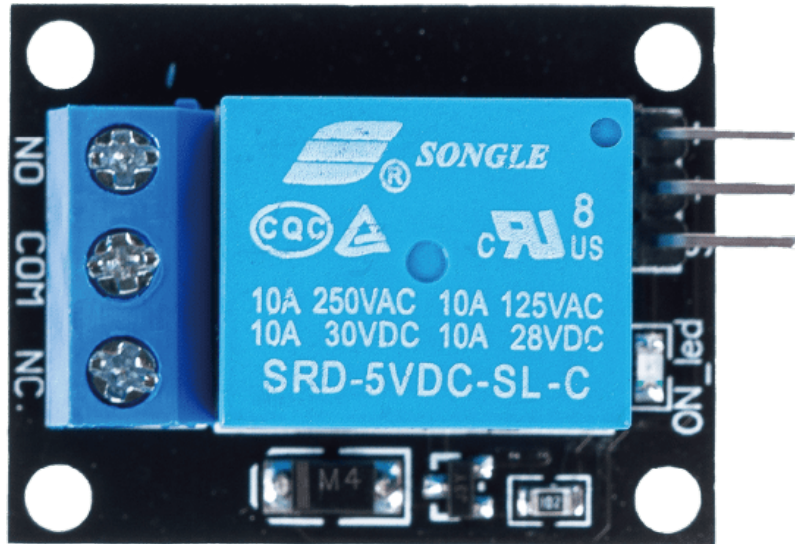
- Try displaying other colors
- Integrate the RGB LED with sensors, and display different colors based on the sensor's value.

### More Projects

- *Gas leak alarm*
- *Light control switch*
- *Motion triggered relay*
- *Bluetooth RGB Controller*
- *Bluetooth Remote Relay*

### Actuator

## 2.4.26 5V Relay Module



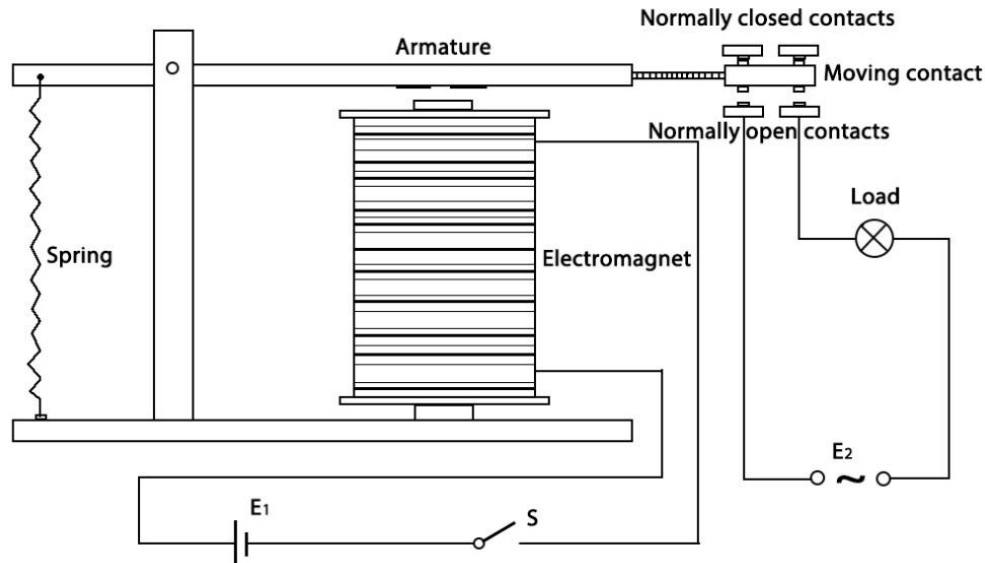
### Introduction

5V relay modules are devices that can switch high voltage or high current devices on and off using a 5V signal from Arduino. They can be used to control devices such as lights, fans, motors, solenoids, etc. 5V relay has three high voltage terminals (NC, C, and NO) which connect to the device you want to control. The other side has three low voltage pins (Ground, Vcc, and Signal) which connect to the Arduino.

### Principle

A relay is a device which is used to provide connection between two or more points or devices in response to the input signal applied. In other words, relays provide isolation between the controller and devices, which may operate on either AC or DC. However, they receive signals from a microcontroller which works on DC hence requiring a relay to bridge the gap. Relay is extremely useful when you need to control a large amount of current or voltage with small electrical signal.

There are 5 parts in every relay:



**Electromagnet** - It consists of an iron core wound by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.

**Armature** - The movable magnetic strip is known as armature. When current flows through them, the coil gets energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).

**Spring** - When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.

**Set of electrical contacts** - There are two contact points:

- Normally open - connected when the relay is activated, and disconnected when it is inactive.
- Normally closed - not connected when the relay is activated, and connected when it is inactive.

**Molded frame** - This is typically made of plastic and provides structural support and protection for the relay.

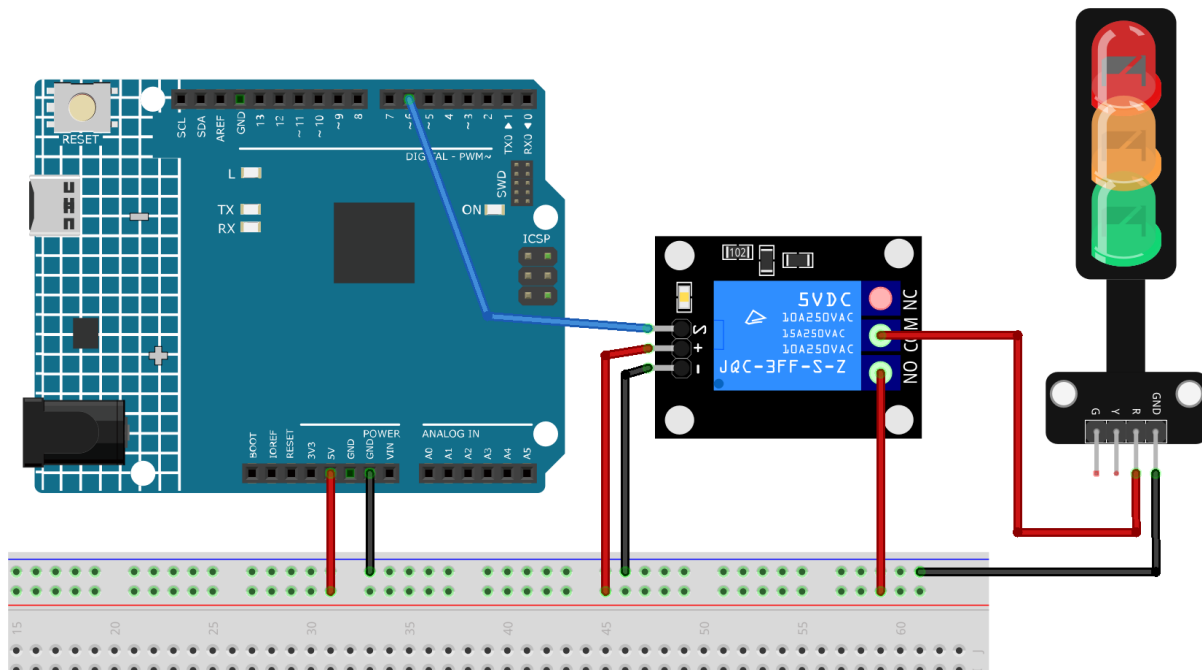
The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open contacts. So the circuit with the load is energized. Then breaking the circuit would be a similar case, as the moving contact will be pulled up to the normally closed contacts under the force of the spring. In this way, the switching on and off of the relay can control the state of a load circuit.

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- 5V Relay Module \* 1
- Jumper Wires

### Circuit Assembly



**Warning:** The following example demonstrates using a relay to control an LED module. **While you can connect the relay to other appliances in actual applications, extreme caution is required when dealing with HIGH AC voltage. Improper or incorrect use can lead to severe injury or even death. Therefore, it is intended for people who are familiar with and knowledgeable about HIGH AC voltage. Always prioritize safety.**

## Code

### Code explanation

1. Setting up the relay pin: - The relay module is connected to pin 6 of the Arduino. This pin is defined as `relayPin` for ease of reference in the code.

```
const int relayPin = 6;
```

2. Configuring the relay pin as an output: - In the `setup()` function, the relay pin is set as an OUTPUT using the `pinMode()` function. This means the Arduino will send signals (either HIGH or LOW) to this pin.

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

3. Toggling the relay ON and OFF: - In the `loop()` function, the relay is first set to the OFF state using `digitalWrite(relayPin, LOW)`. It remains in this state for 3 seconds (`delay(3000)`). - Then, the relay is set to the ON state using `digitalWrite(relayPin, HIGH)`. Again, it remains in this state for 3 seconds. - This cycle repeats indefinitely.

```
void loop() {
  digitalWrite(relayPin, LOW);
  delay(3000);
```

(continues on next page)



(continued from previous page)

```
digitalWrite(relayPin, HIGH);  
delay(3000);  
}
```

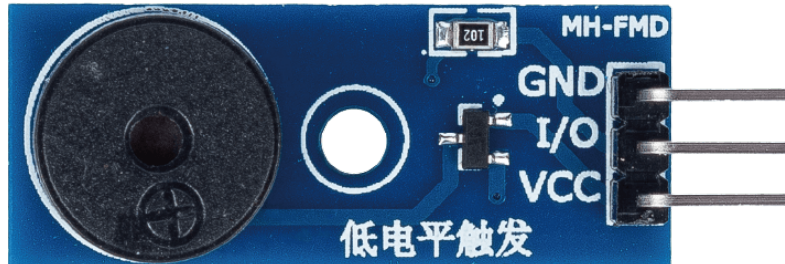
### Additional Ideas

- Introduce a physical button to manually control the relay's state.
- Integrate sensors (like a temperature or light sensor) to trigger the relay based on environmental conditions.

### More Projects

- *Light control switch*
- *Motion triggered relay*
- *Remote Relay Controller with Blynk*
- *Bluetooth Remote Relay*
- *Bluetooth Voice-control Relay*

## 2.4.27 Passive Buzzer Module



### Introduction

A passive buzzer is a device that generates sound when an electrical signal is applied to it. It is called passive because it does not have an internal oscillator to generate sound on its own. Instead, it relies on an external signal from a microcontroller like Arduino to produce sound. The passive buzzer module is a small electronic component that contains a passive buzzer and some additional circuitry that makes it easier to use with Arduino.

### Principle

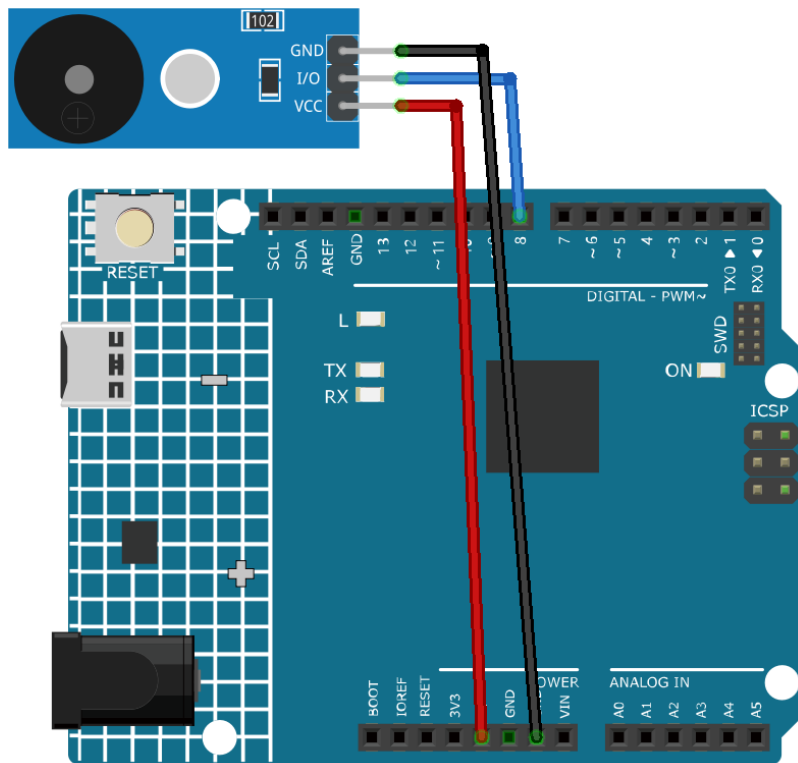
The working principle of the passive buzzer module is based on the piezoelectric effect. When an electrical signal is applied to the buzzer, it causes a piezoelectric crystal inside the buzzer to vibrate at a specific frequency. This vibration produces sound waves that we can hear. The frequency of the sound produced by the buzzer depends on the frequency of the electrical signal applied to it. By changing the frequency of the signal, we can change the pitch of the sound produced by the buzzer.

### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Passive Buzzer Module \* 1
- Jumper Wires

#### Circuit Assembly



## Code

### Code explanation

1. Including the pitches library: This library provides the frequency values for various musical notes, allowing you to use musical notation in your code.

```
#include "pitches.h"
```

2. Defining constants and arrays:

- buzzerPin is the digital pin on the Arduino where the buzzer is connected.
- melody[] is an array that stores the sequence of notes to be played.
- noteDurations[] is an array that stores the duration of each note in the melody.

```
const int buzzerPin = 8;
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};
int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};
```

3. Playing the melody:

- The for loop iterates over each note in the melody.
- The tone() function plays a note on the buzzer for a specific duration.
- A delay is added between notes to distinguish them.
- The noTone() function stops the sound.

```
void setup() {
  for (int thisNote = 0; thisNote < 8; thisNote++) {
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(buzzerPin, melody[thisNote], noteDuration);
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    noTone(buzzerPin);
  }
}
```

4. Empty loop function: Since the melody is played only once in the setup, there's no code in the loop function.

### Additional Ideas

- **Modify the melody:** You can experiment by changing the notes and durations in the `melody[]` and `noteDurations[]` arrays to create your own tunes. If you are interested, there is a repository () on GitHub that provides Arduino code for playing different songs. Although their approach may be different from this project, you can refer to their notes and durations.
- **Add a button:** Integrate a push-button to the circuit and modify the code to play the melody when the button is pressed.

### More Projects

- *Doorbell*
- *Gas leak alarm*
- *Bluetooth Piano*

### 2.4.28 Servo Motor (SG90)



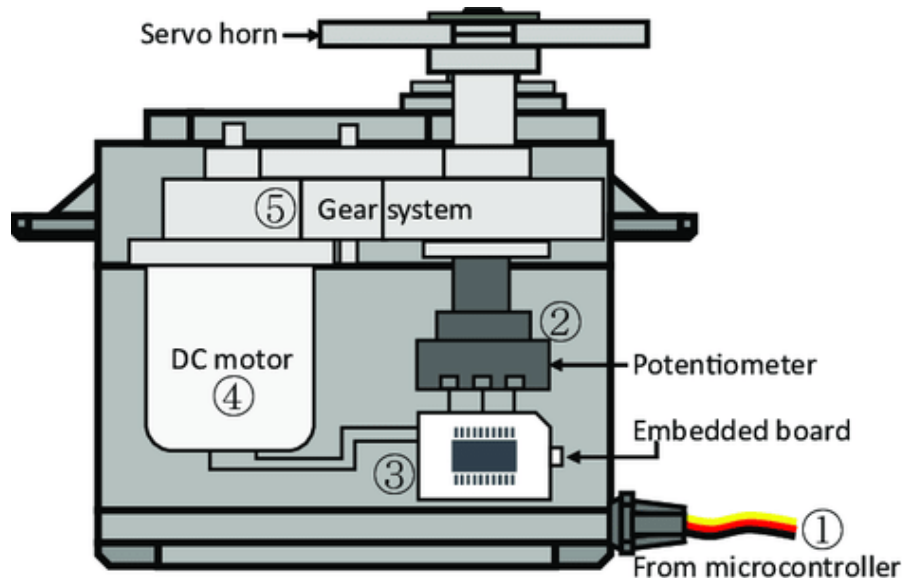
### Introduction

Servo motors are devices that can rotate to a specific angle or position. They can be used to move robotic arms, steering wheels, camera gimbals, etc. Servo motors have 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 a ground pin on the board. The signal wire is usually yellow or orange and should be connected to a PWM pin on the board.

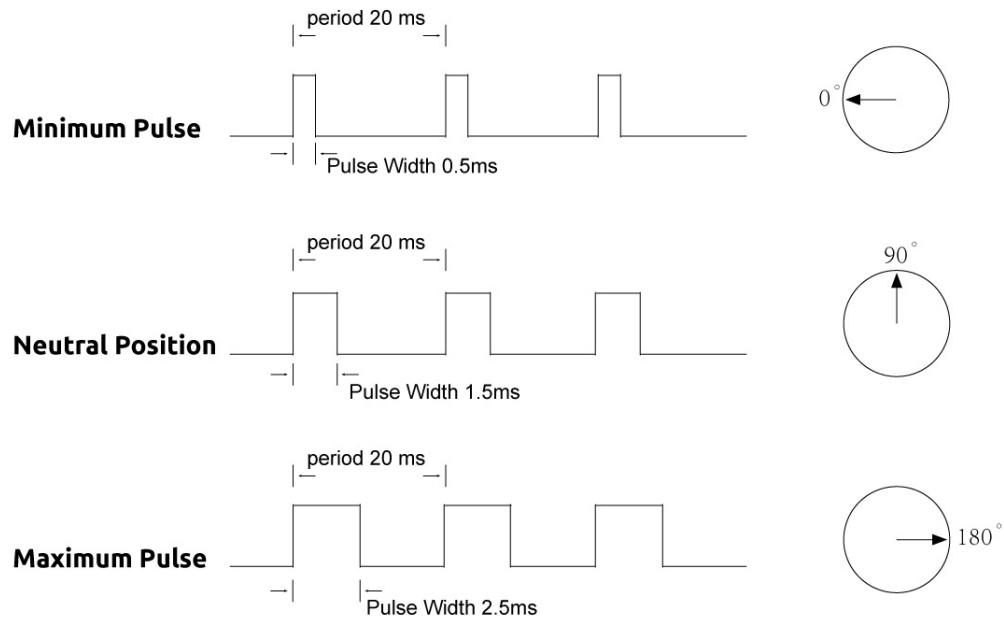
## Principle

A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

It works like this: The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear system and then rotates the shaft after deceleration. The shaft and potentiometer of the servo are connected together. When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded board. Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.



The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position). When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point. When the pulse is wider than 1.5 ms the opposite occurs. The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo. Generally the minimum pulse will be about 0.5 ms wide and the maximum pulse will be 2.5 ms wide.

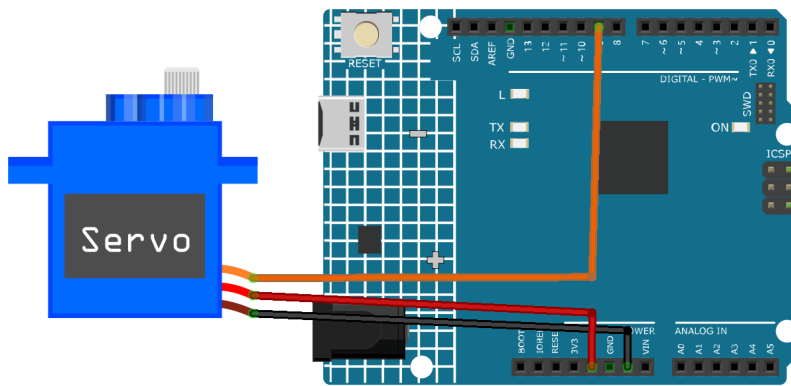


### Usage

#### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Servo Motor \* 1
- Jumper Wires

#### Circuit Assembly



## Code

### Code explanation

1. Here, the Servo library is included which allows for easy control of the servo motor. The pin connected to the servo and the initial angle of the servo are also defined.

```
#include <Servo.h>
const int servoPin = 9; // Define the servo pin
int angle = 0;           // Initialize the angle variable to 0 degrees
Servo servo;             // Create a servo object
```

2. The setup() function runs once when the Arduino starts. The servo is attached to the defined pin using the attach() function.

```
void setup() {
  servo.attach(servoPin);
}
```

3. The main loop has two for loops. The first loop increases the angle from 0 to 180 degrees, and the second loop decreases the angle from 180 to 0 degrees. The servo.write(angle) command sets the servo to the specified angle. The delay(15) causes the servo to wait for 15 milliseconds before moving to the next angle, controlling the speed of the scanning movement.

```
void loop() {
  // scan from 0 to 180 degrees
  for (angle = 0; angle < 180; angle++) {
    servo.write(angle);
    delay(15);
  }
  // now scan back from 180 to 0 degrees
  for (angle = 180; angle > 0; angle--) {
    servo.write(angle);
    delay(15);
  }
}
```

### Additional Ideas

- Control the servo's motion using a potentiometer. The rotation of the potentiometer could be used to directly control the angle of the servo.

### More Projects

- *Smart trashcan*
- *Bluetooth Lock Controller*

### 2.4.29 Centrifugal Pump



#### Introduction

A centrifugal pump is a device that can move liquids from one place to another by using a rotating impeller. It can be used to pump water, oil, chemicals, etc. A centrifugal pump has two main parts: a motor and a pump. The motor provides power to the pump and the pump converts the rotational energy into pressure and flow.

#### Principle

A centrifugal pump operates by using a spinning impeller which increases the velocity of the fluid, drawing it into the pump through an inlet pipe. As the liquid exits the impeller's outer edge, centrifugal force pushes it out through an outlet pipe, resulting in increased pressure. The faster the impeller spins, the higher the pressure and flow of the liquid.

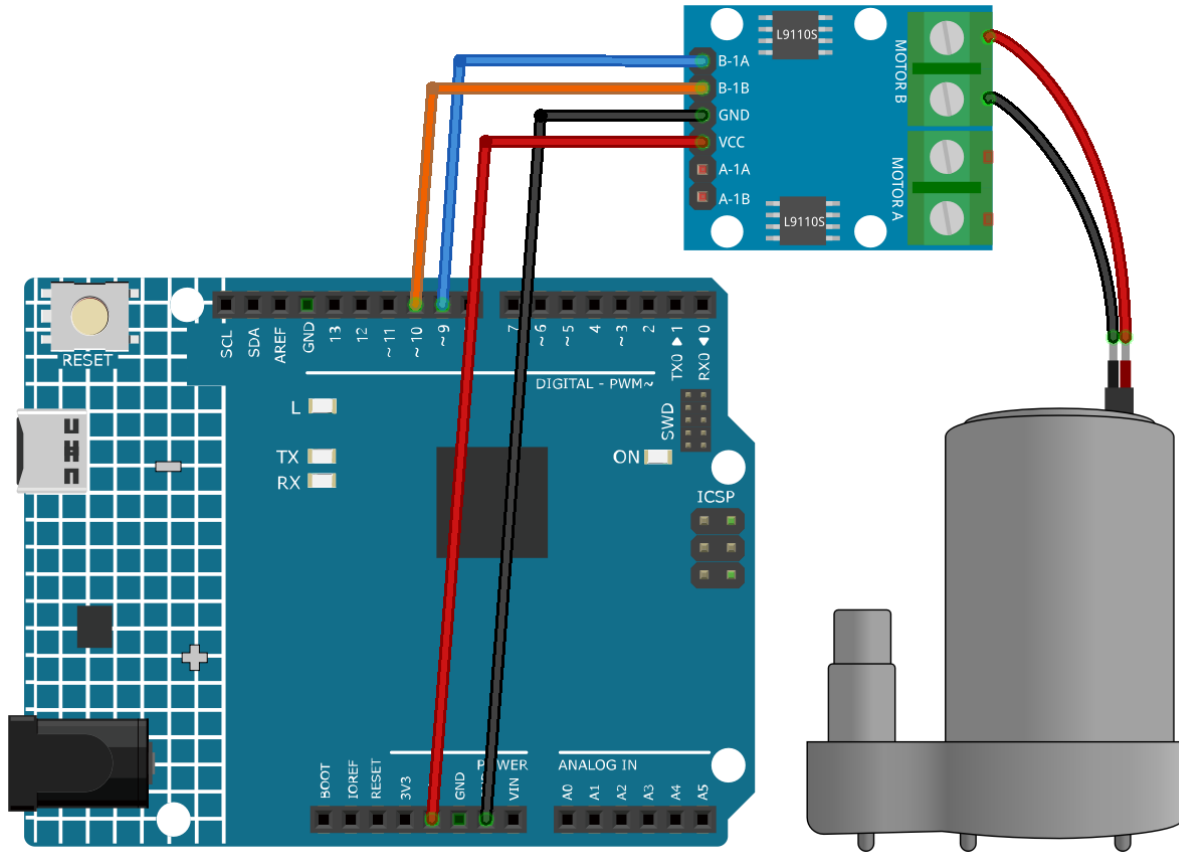
#### Usage

##### Hardware components

- Arduino Uno R4 or R3 board \* 1
- Centrifugal Pump \* 1
- Jumper Wires

##### Circuit Assembly





## Code

### Code explanation

- Two pins are defined for controlling the motor, specifically `motorB_1A` and `motorB_2A`. These pins will connect to the L9110 motor control board to control the direction and speed of the motor.

```
const int motorB_1A = 9;
const int motorB_2A = 10;
```

- Configuring the pins and controlling the motor:

- The `setup()` function initializes the pins as `OUTPUT` which means they can send signals to the motor control board.
- The `analogWrite()` function is used to set the motor speed. Here, setting one pin to `HIGH` and the other to `LOW` makes the pump spin in one direction. After a 5-second delay, both pins are set to 0, turning off the motor.

```
void setup() {
  pinMode(motorB_1A, OUTPUT); // set pump pin 1 as output
  pinMode(motorB_2A, OUTPUT); // set pump pin 2 as output
  analogWrite(motorB_1A, HIGH);
  analogWrite(motorB_2A, LOW);
  delay(5000); // wait for 5 seconds
```

(continues on next page)

(continued from previous page)

```
analogWrite(motorB_1A, 0); // turn off the pump
analogWrite(motorB_2A, 0);
}
```

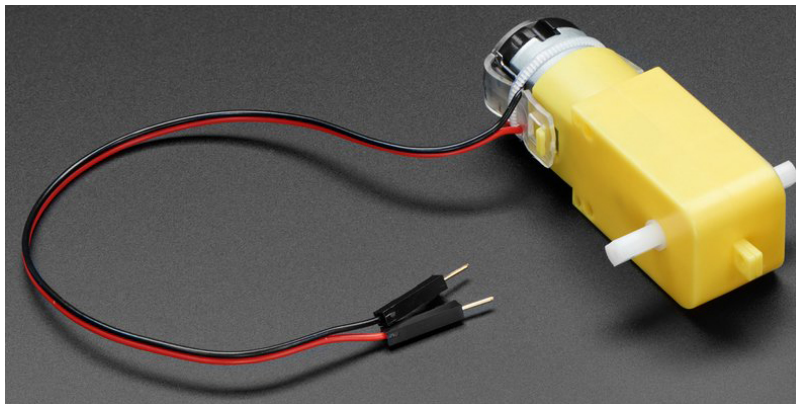
### Additional Ideas

- Reverse the pump's direction by switching the HIGH and LOW values between the pins.
- Implement a system where the pump toggles its state (on/off) using a button press.
- Implement a potentiometer to control the speed of the pump using PWM.
- Include sensors to automate the pump's operation based on certain conditions, e.g., turning the pump on/off depending on water level in a tank.

### More Projects

- *Automatic soap dispenser*
- *Auto Watering System with Blynk*

### 2.4.30 TT Motor



#### Introduction

A TT motor is a type of DC motor that has a gearbox attached to it. The gearbox reduces the speed of the motor and increases its torque. A TT motor is commonly used in applications such as driving wheels, propellers, fans, among others. A TT motor has two wires: a positive wire and a negative wire. The positive wire is usually red and the negative wire is usually black.

A TT DC gearbox motor with a 1:48 gear ratio is used in the product, it comes with 2 x 200mm wires with 0.1" male connectors that fit into a breadboard. Perfect for plugging into a breadboard or terminal block.

You can power these motors with 3 ~ 6VDC, but of course, they will go a little faster at higher voltages.

## Principle

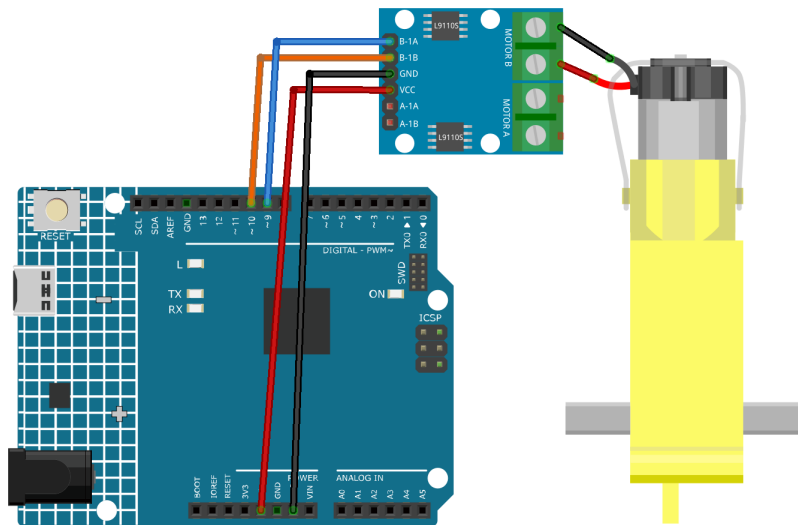
A TT motor works by converting electrical energy into mechanical energy. When a voltage is applied to the wires of the motor, it creates a magnetic field that causes the motor to spin. The speed and direction of the motor depend on the voltage and polarity of the power supply. The higher the voltage, the faster the motor spins. Reversing the polarity will cause the motor to spin in the opposite direction.

## Usage

### Hardware components

- Arduino Uno R4 or R3 board \* 1
- TT Motor \* 1
- Jumper Wires

### Circuit Assembly



## Code

### Code explanation

1. The first part of the code defines the motor control pins. These are connected to the L9110 motor control board.

```
// Define the motor pins
const int motorB_1A = 9;
const int motorB_2A = 10;
```

2. The setup() function initializes the motor control pins as output using the pinMode() function. Then it uses analogWrite() to set the speed of the motor. The value passed to analogWrite() can range from 0 (off) to 255 (full speed). A delay() function is then used to pause the code for 5000 milliseconds (or 5 seconds), after which the motor speed is set to 0 (off).

```
void setup() {
  pinMode(motorB_1A, OUTPUT); // set motor pin 1 as output
```

(continues on next page)

(continued from previous page)

```
pinMode(motorB_2A, OUTPUT); // set motor pin 2 as output

analogWrite(motorB_1A, 255); // set motor speed (0-255)
analogWrite(motorB_2A, 0);

delay(5000);

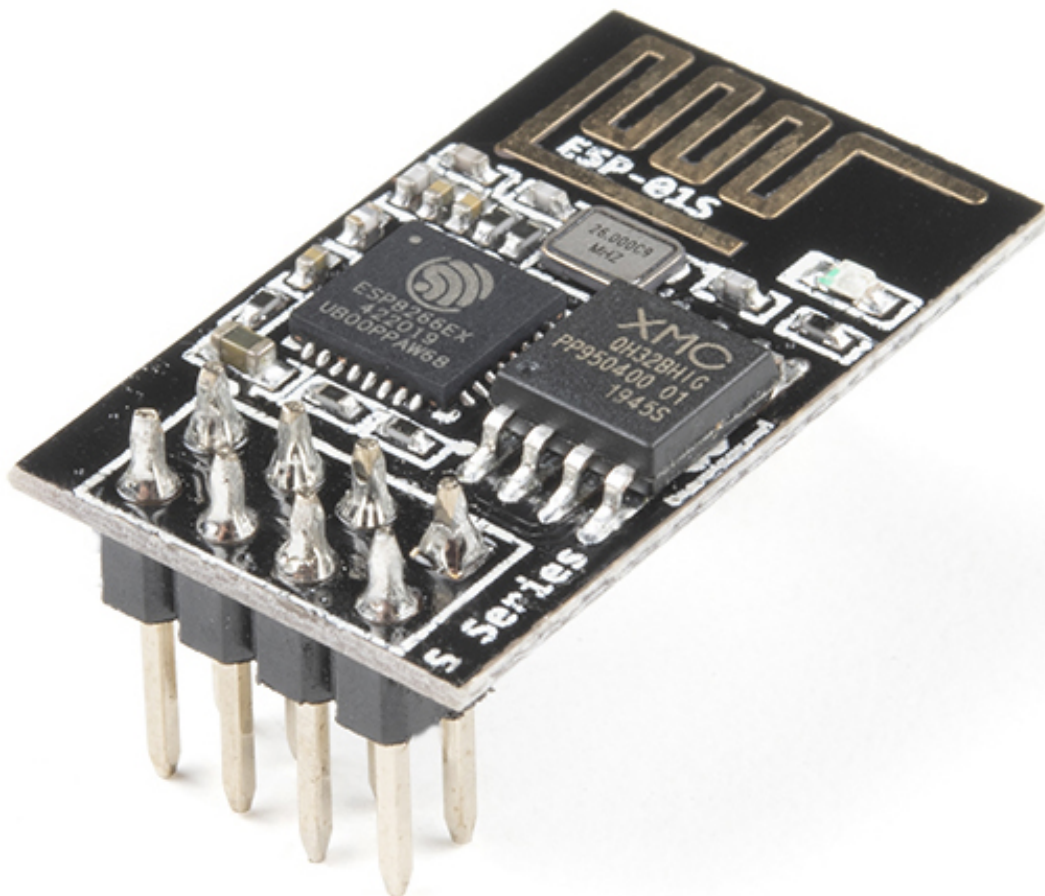
analogWrite(motorB_1A, 0);
analogWrite(motorB_2A, 0);
}
```

### Additional Ideas

- Control Motor Speed with a Potentiometer: Instead of hardcoding the motor speed, you could use a potentiometer to dynamically control the speed of the motor.

### Wireless & IoT

#### 2.4.31 ESP8266 Module



The ESP8266 is a low-cost Wi-Fi microchip, with built-in TCP/IP networking software, and microcontroller capability, produced by Espressif Systems in Shanghai, China.

The chip first came to the attention of Western makers in August 2014 with the ESP-01 module, made by a third-party manufacturer Ai-Thinker. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using Hayes-style commands. However, at first, there was almost no English-language documentation on the chip and the commands it accepted. The very low price and the fact that there were very few external components on the module, which suggested that it could eventually be very inexpensive in volume, attracted many hackers to explore the module, the chip, and the software on it, as well as to translate the Chinese documentation.

Pins of ESP8266 and their functions:

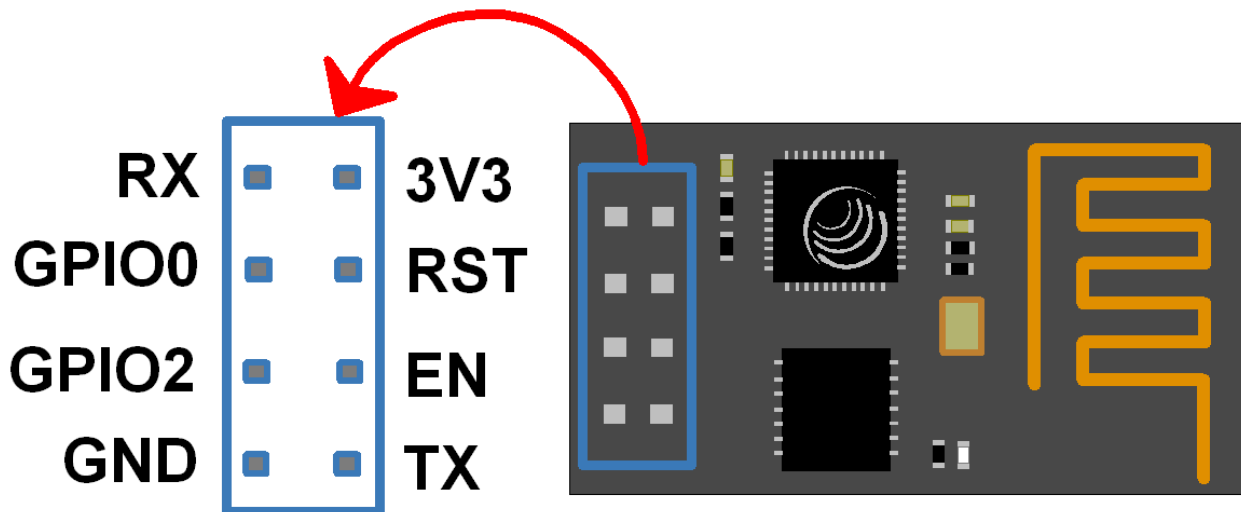
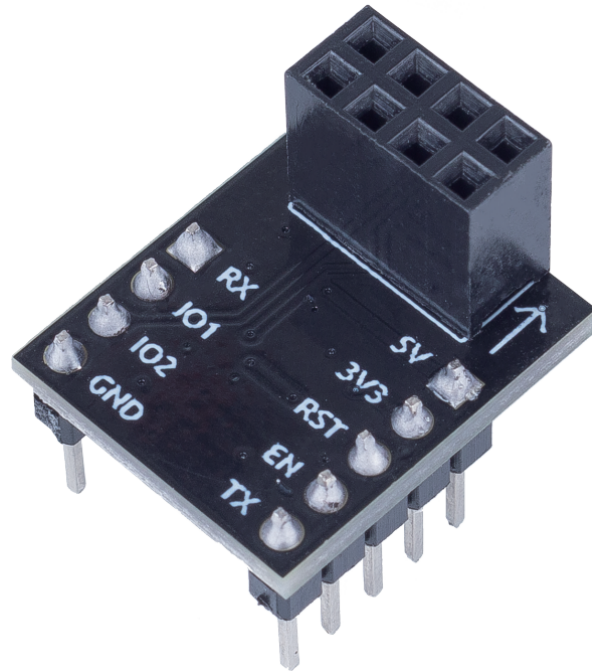


Table 1: ESP8266-01 Pins

Pin	Name	Description
1	TXD	UART_TXD, sending; General Purpose Input/Output: GPIO1; Pull-down is not allowed when startup.
2	GND	GND
3	CU_PD	Working at high level; Power off when low level is supplied.
4	GPIO2	It should be high level when power on, hardware pull-down is not allowed; Pull-up by default;
5	RST	External Reset signal, reset when low level is supplied; work when high level is supplied (high level by default);
6	GPIO0	WiFi Status indicator; Operation mode selection: Pull-up: Flash Boot, operation mode; Pull-down: UART Download, download mode
7	VCC	Power Supply(3.3V)
8	RXD	UART_RXDReceiving; General Purpose Input/Output: GPIO3;

- [ESP8266 - Espressif](#)
-

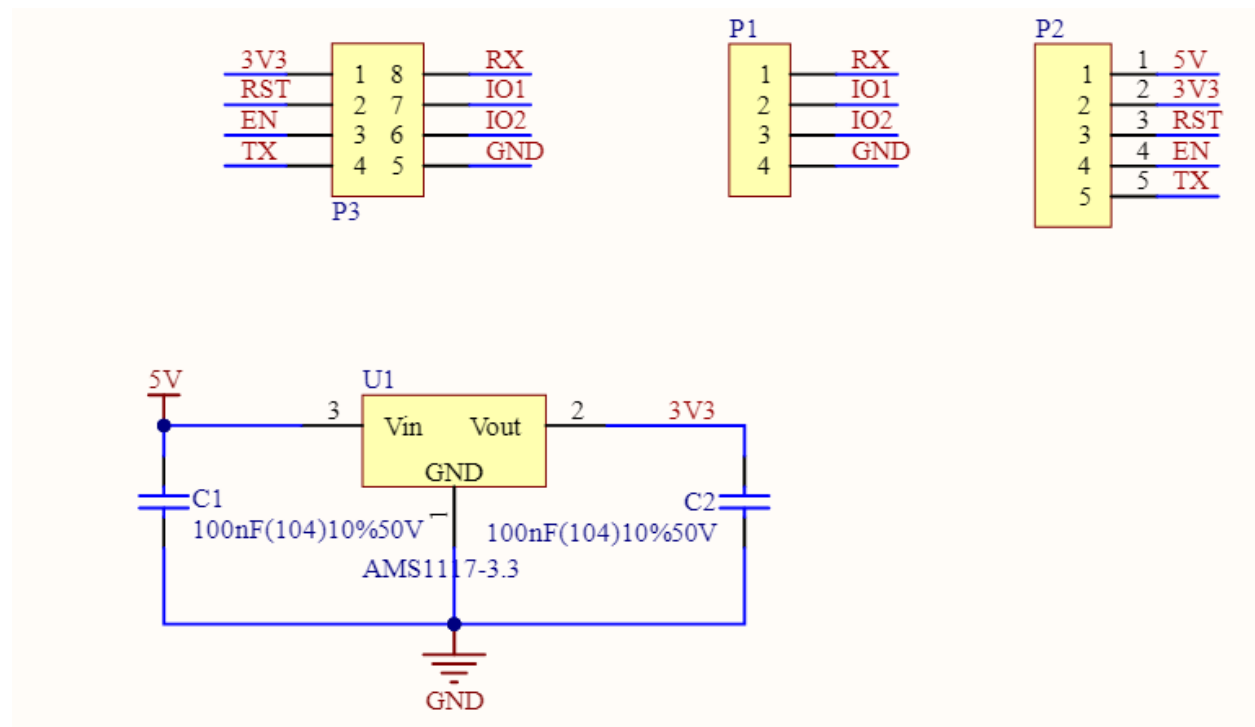
## ESP8266 Adapter



The ESP8266 adapter is an expansion board that allows the ESP8266 module to be used on a breadboard.

It perfectly matches the pins of the ESP8266 itself, and also adds a 5V pin to receive the voltage from the Arduino board. The integrated AMS1117 chip is used to drive the ESP8266 module after dropping the voltage to 3.3V.

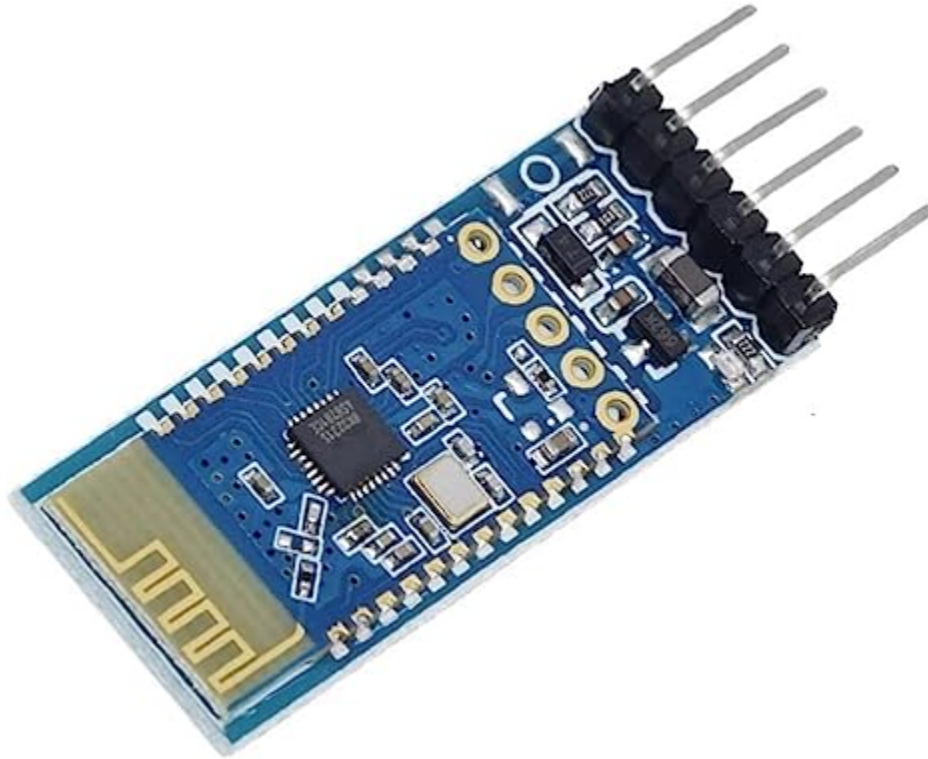
The schematic diagram is as follows:



## More Projects

- *IoT Projects*

### 2.4.32 JDY-31 Bluetooth Module



**Warning:** This module **does not support Apple device** connections, so tutorials involving this module require an Android phone or tablet.

The JDY-31 Bluetooth module is a pin-compatible replacement for the HC-06 Bluetooth module. It is simpler and easier to use than the HC-06 and is often available at a slightly lower cost.

The JDY-31 Bluetooth module is based on Bluetooth 3.0 SPP design and can support Windows, Linux, and Android data transmission. The working frequency of the JDY-31 Bluetooth module is 2.4 GHz with modulation mode GFSK. The maximum transmission power is 8 dB, and the maximum transmission distance is 30 meters. Users can modify the device name through AT command, baud rate, and other instructions.

Pins of JDY-31 and their functions:



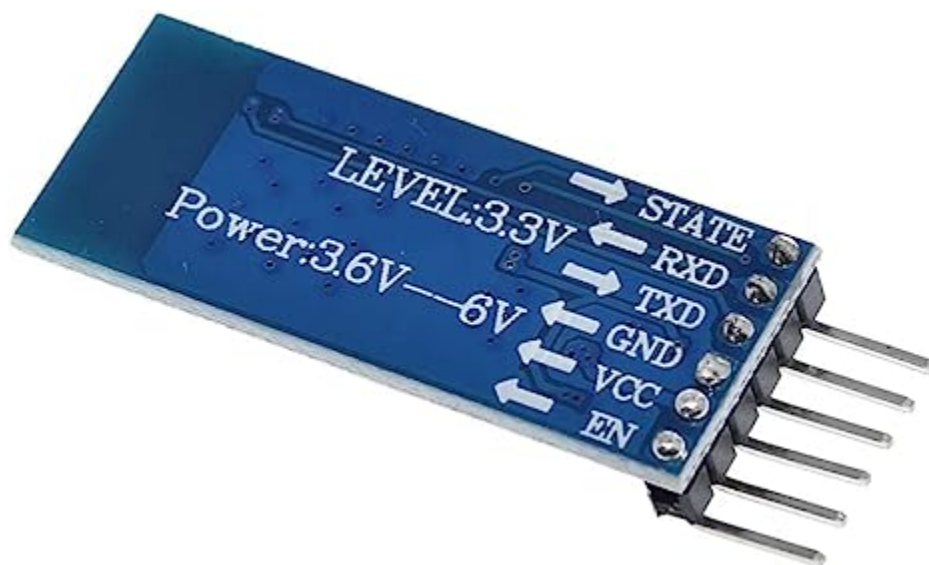


Table 2: JDY-31 Pins

Pin	Name	Description
1	STATE	Connection status pin (not connected low level, output high level after connectio)
2	RXD	Receiver pin, this pin must connect to TX pin of next device.
3	TXD	Transmitter pin, this pin must connect to RX pin of next device.
4	GND	GND
5	VCC	Power Supply(1.8-3.6V, 3.3v recommended)
6	EN	enable or disable the module. When this pin is held high, the module is enabled and begins transmitting and receiving data.

patch application: general application only need to connect VCC, GND, RXD, TXD 4 pins, if you need to actively disconnect in the connection state, send AT+DISC in the connection state.

### AT Command Set

Command	Function	Default
AT+VERSION	Version Number	JDY-31-V1.2
AT+RESET	Soft reset	
AT+DISC	Disconnect (valid when connected)	
AT+LADDR	Query the MAC address of the module	
AT+PIN	Set or query connection password	1234
AT+BAUD	Set or query baud rate	9600
AT+NAME	Set or query broadcast name	JDY-31-SPP
AT+DEFAULT	Factory reset	
AT+ENLOG	Serial port status output	1



## More Projects

- *Get Started with Bluetooth*
- *Bluetooth LCD*
- *Bluetooth Traffic Light*
- *Bluetooth Lock Controller*
- *Bluetooth RGB Controller*
- *Bluetooth Environmental Monitor*
- *Bluetooth Piano*
- *Bluetooth OLED*
- *Bluetooth Remote Relay*
- *Bluetooth Voice-control Relay*

## 2.5 IoT Projects

This kit features the ESP8266 Wifi module and the JDY-31 Bluetooth Module. The ESP8266 allows your Arduino to connect to the internet for IoT experiments, while the Bluetooth module facilitates short-range wireless communication.

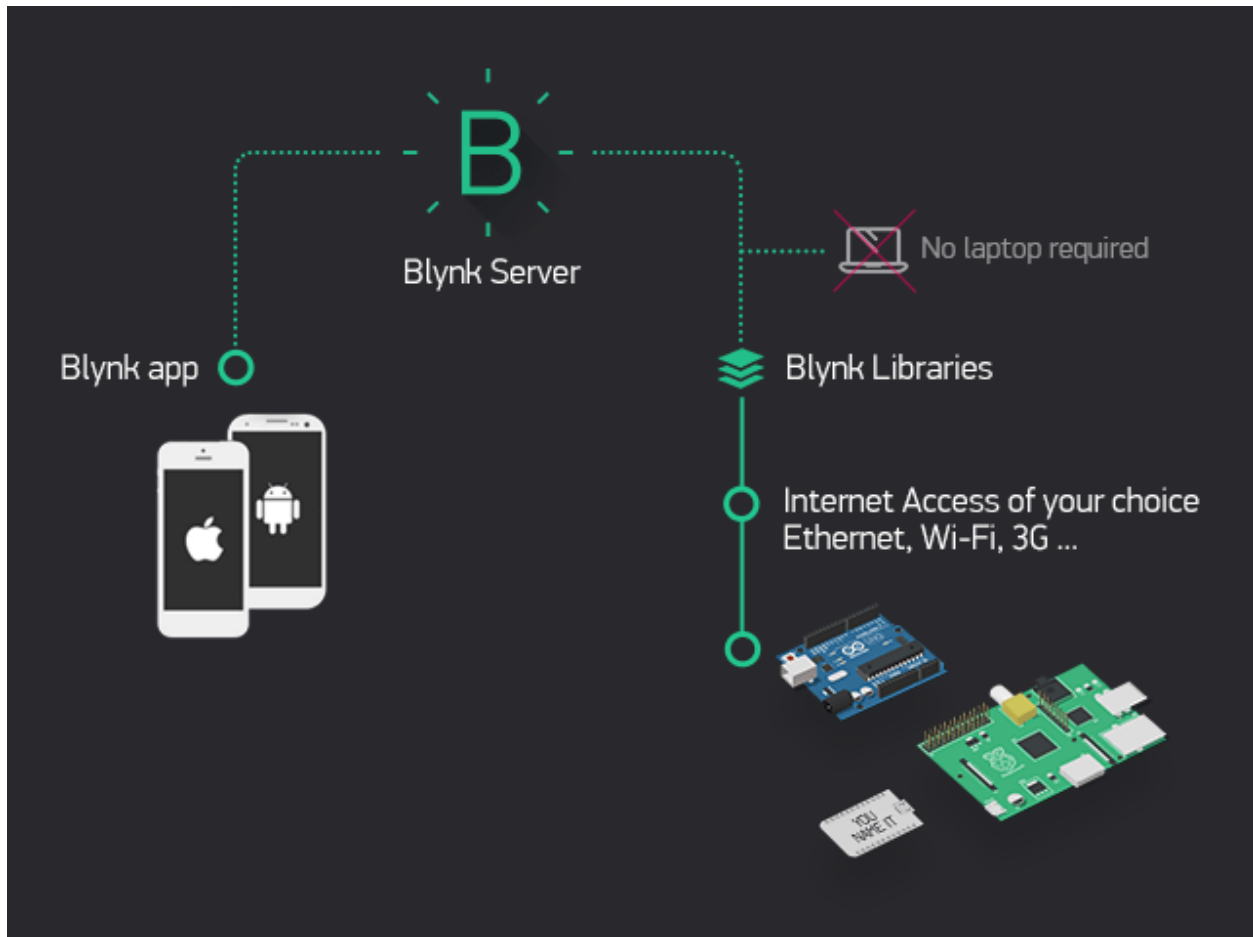
With the ESP8266 WiFi module, you'll explore a range of IoT projects that allow you to connect your Arduino to the internet. We provide step-by-step guides to work with platforms like [and](#) , enabling you to set up alert systems, remote controllers, and environmental monitors, among other applications.

On the other hand, the JDY-31 Bluetooth module opens doors to localized wireless communication. We will guide you through pairing your Arduino with smartphones or other Bluetooth-enabled devices for various controls and monitoring tasks. And you can create projects using apps developed with [.](#)

### Wi-Fi

#### 2.5.1 Get Started with Blynk

Blynk is a full suite of software required to prototype, deploy, and remotely manage connected electronic devices at any scale: from personal IoT projects to millions of commercial connected products. With Blynk anyone can connect their hardware to the cloud and build a no-code iOS, Android, and web applications to analyze real-time and historical data coming from devices, control them remotely from anywhere in the world, receive important notifications, and much more.

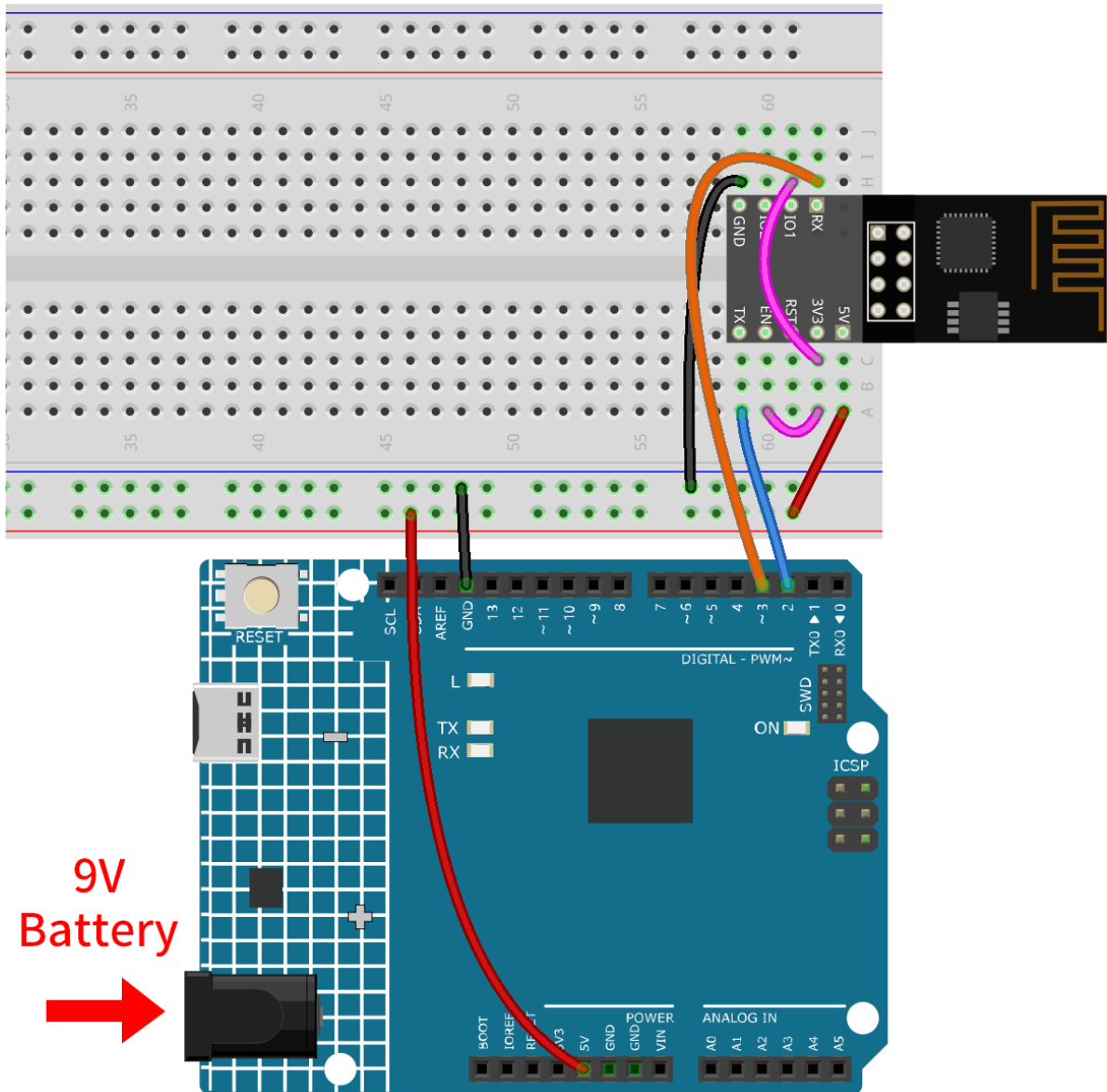


Getting the R4 board to communicate with Blynk requires some configuration when you first use Blynk. Follow the steps below, and note that you must do them in order and not skip any chapters.

### 1.1 Configuring the ESP8266

The ESP8266 module that comes with the kit is already pre-burned with AT firmware, but you still need to modify its configuration by following the steps below.

1. Build the circuit.



2. Open the 00-Set\_software\_serial.ino file under the path of ultimate-sensor-kit\iot\_project\wifi\00-Set\_software\_serial. Or copy this code into Arduino IDE. And upload the code.

The code establishes a software serial communication using Arduino's SoftwareSerial library, allowing the Arduino to communicate with the ESP8266 module through its digital pins 2 and 3 (as Rx and Tx). It checks for data transfer between them, forwarding received messages from one to the other at a baud rate of 115200. **With this code, you can use the Arduino's serial monitor to send AT firmware commands to the ESP8266 module and receive its responses.**

```
#include <SoftwareSerial.h>
SoftwareSerial espSerial(2, 3); //Rx,Tx

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  espSerial.begin(115200);
```

(continues on next page)

(continued from previous page)

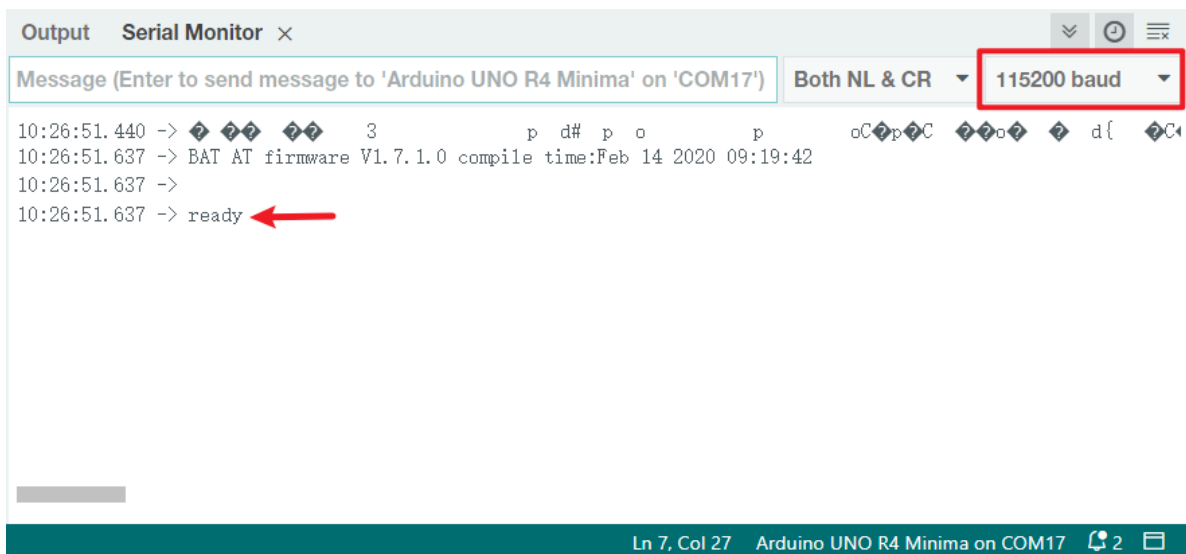
```

}

void loop() {
  if (espSerial.available()) {
    Serial.write(espSerial.read());
  }
  if (Serial.available()) {
    espSerial.write(Serial.read());
  }
}

```

3. Click the magnifying glass icon (Serial Monitor) in the upper right corner and set the baud rate to **115200**. (You may have some printed information like me, or you may not, it doesn't matter, just go to the next step.)

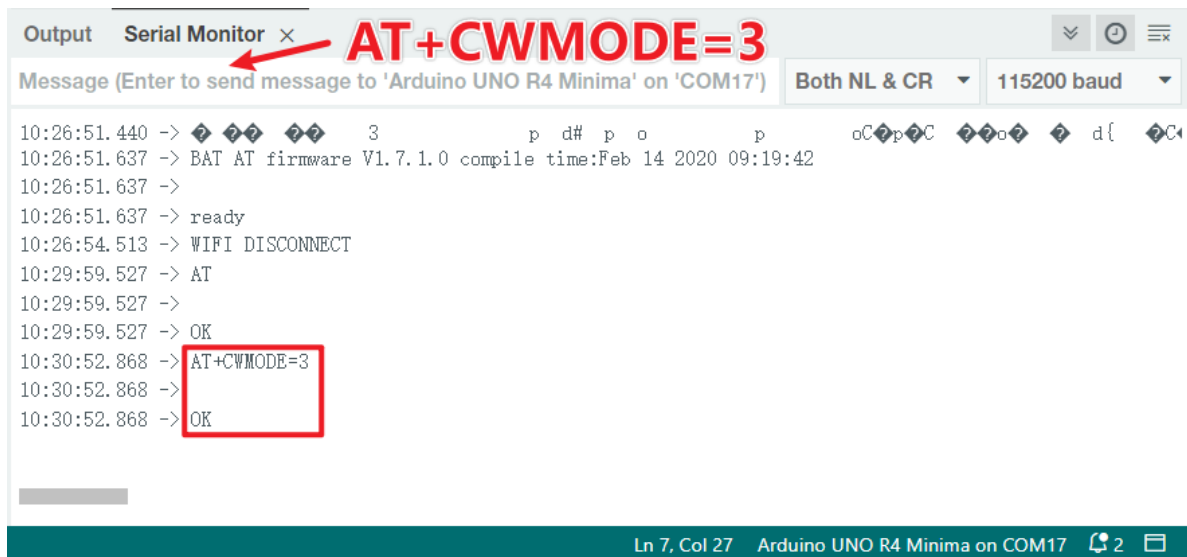
**Warning:**

- If ready doesn't appear, you can try to reset the ESP8266 module(connect RST to GND) and re-open the Serial Monitor.
- In addition, if the result is OK, you may need to re-burn the firmware, please refer to [How to re-burn the firmware for ESP8266 module?](#) for details. If you still can't solve it, please take a screenshot of the serial monitor and send it to [service@sunfounder.com](mailto:service@sunfounder.com), we will help you solve the problem as soon as possible.

4. Click on **NEWLINE DROPDOWN BOX**, select both NL & CR in the drop down option, enter AT, if it returns OK, it means ESP8266 has successfully established connection with R4 board.



5. Enter AT+CWMODE=3 and the managed mode will be changed to **Station and AP** coexistence.



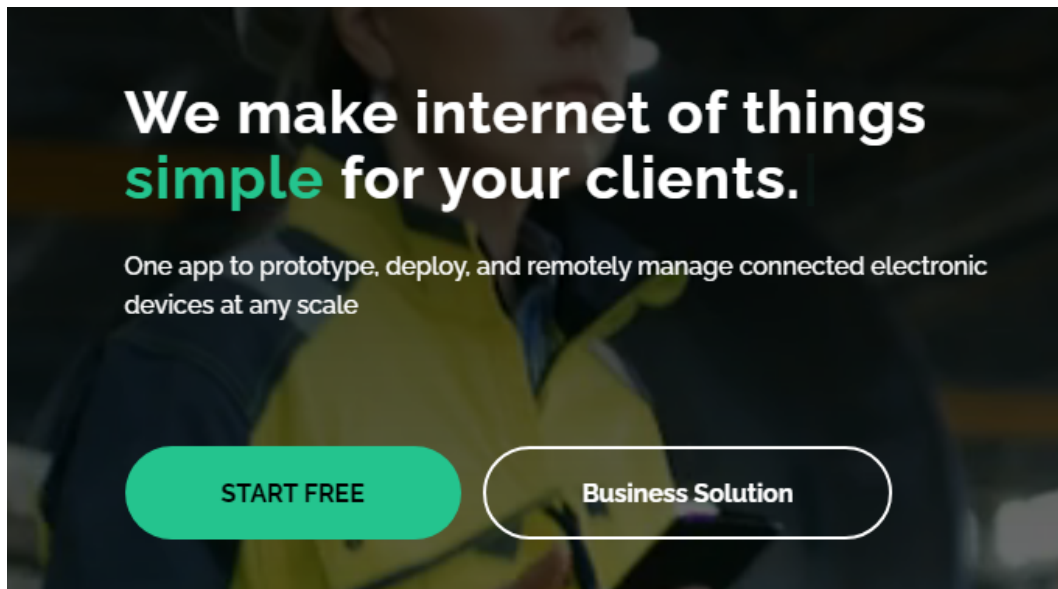
```
Output Serial Monitor x AT+CWMODE=3
Message (Enter to send message to 'Arduino UNO R4 Minima' on 'COM17') Both NL & CR 115200 baud
10:26:51.440 -> 3 p d# p o p oC d{
10:26:51.637 -> BAT AT firmware V1.7.1.0 compile time:Feb 14 2020 09:19:42
10:26:51.637 ->
10:26:51.637 -> ready
10:26:54.513 -> WIFI DISCONNECT
10:29:59.527 -> AT
10:29:59.527 ->
10:29:59.527 -> OK
10:30:52.868 -> AT+CWMODE=3
10:30:52.868 ->
10:30:52.868 -> OK
```

## Reference


- 

## 1.2 Configuring the Blynk

1. Go to the [BLYNK](#) and click **START FREE**.



2. Fill in your email address to register an account.



## Sign Up

Welcome! Fill in your email address and we will send an account activation link.

EMAIL

☐ I agree to [Terms and Conditions](#) and accept [Privacy Policy](#)

**Sign Up**

[Back to Login](#)

3. Go to your email address to complete your account registration.



Welcome!

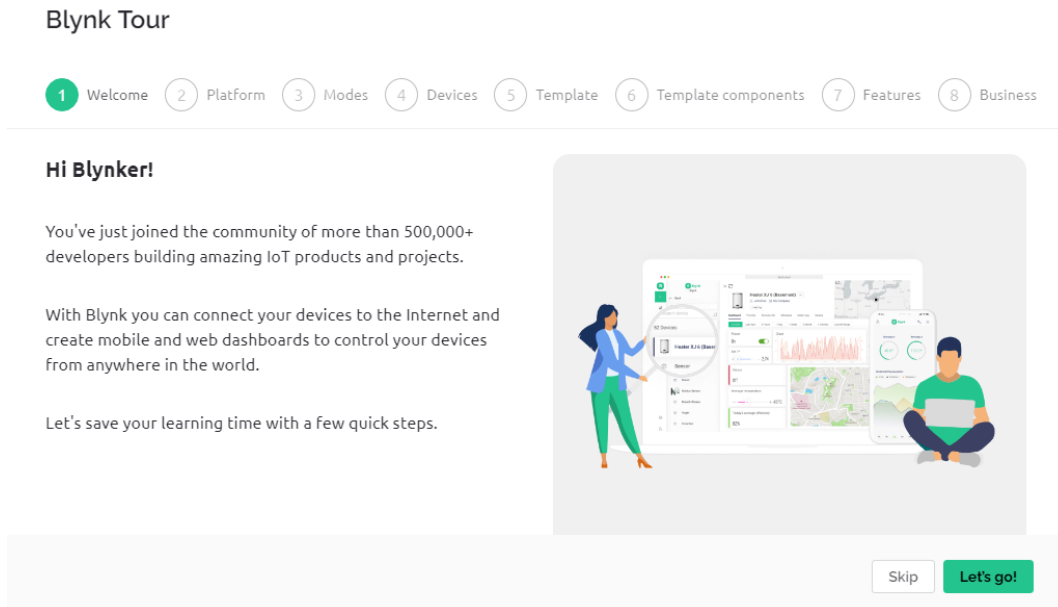
We're excited to see you on board.

To get started, you'll need to create a password for your account.

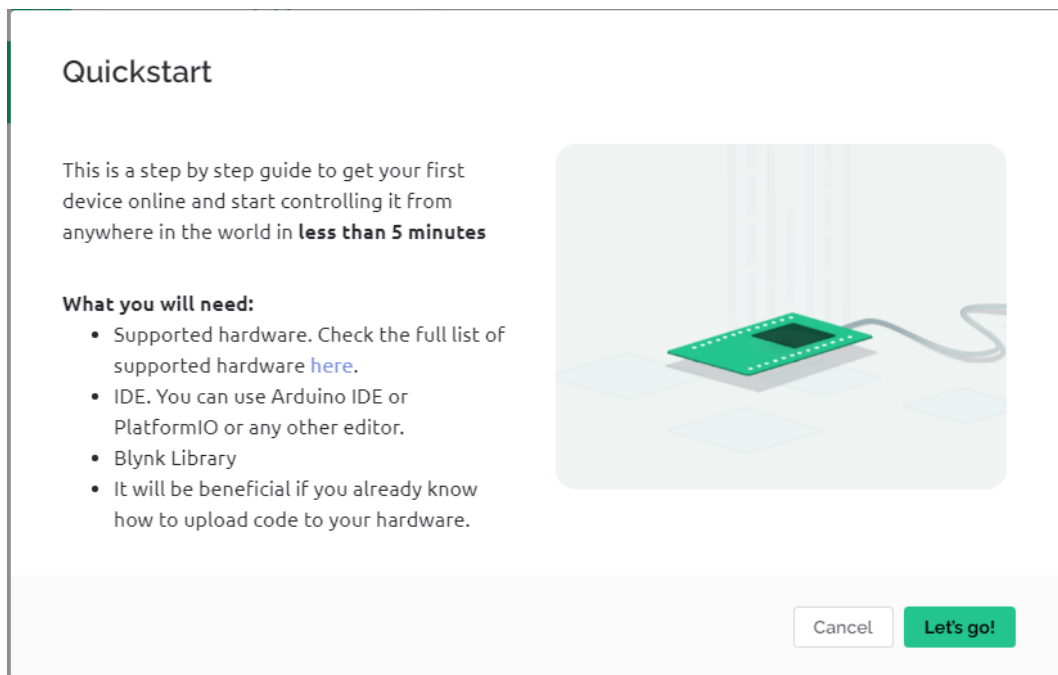
**Create Password**

The link will expire in 30 days.

4. Afterwards, **Blynk Tour** will appear and you can read it to learn the basic information about the Blynk.



5. Next, we need to create a template and device with this **Quick Start**, click **Let's go**.



6. Select the hardware and connection type.



**Quickstart**

1 Hardware — 2 IDE — 3 Blynk Library — 4 Code — 5 Device activation

**Which hardware are you using?**

We will help you prepare the code for you board

ESP8266

**What is your device connectivity type**

Blynk supports various connection types (BLE is not supported yet).

WiFi

Cancel Next →

7. Here you are told which IDE you need to prepare, we recommend the **Arduino IDE**.

**Quickstart**

✓ Hardware — 2 IDE — 3 Blynk Library — 4 Code — 5 Device activation

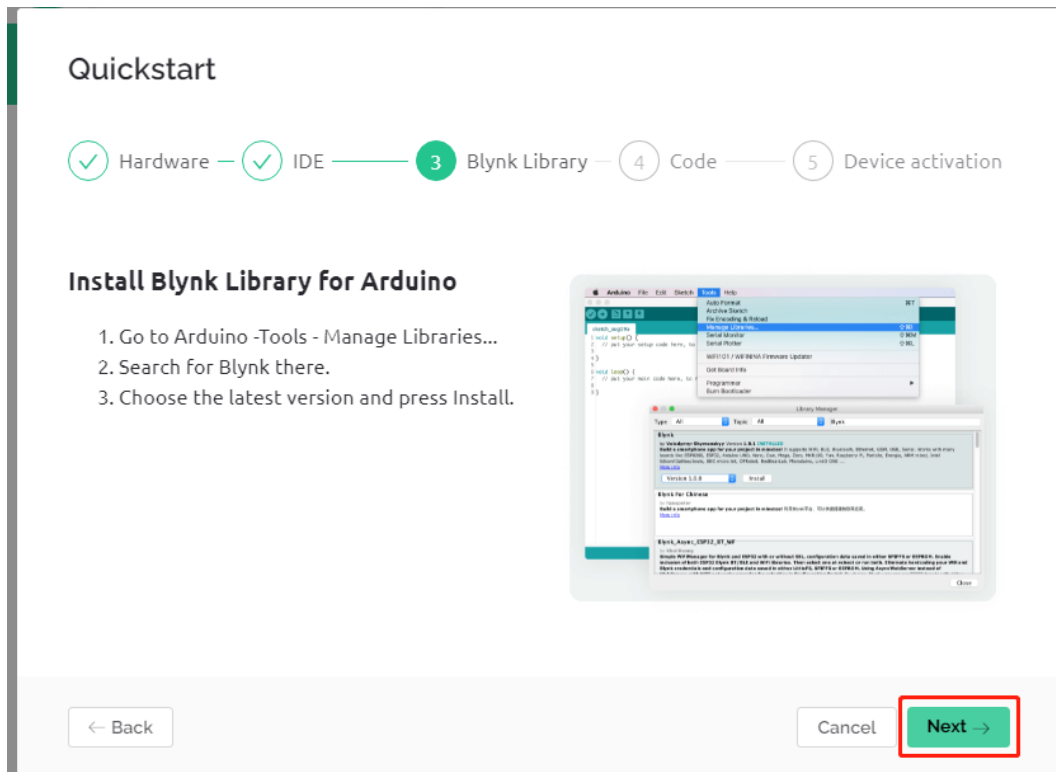
**Which IDE do you use?**

Arduino PlatformIO Other

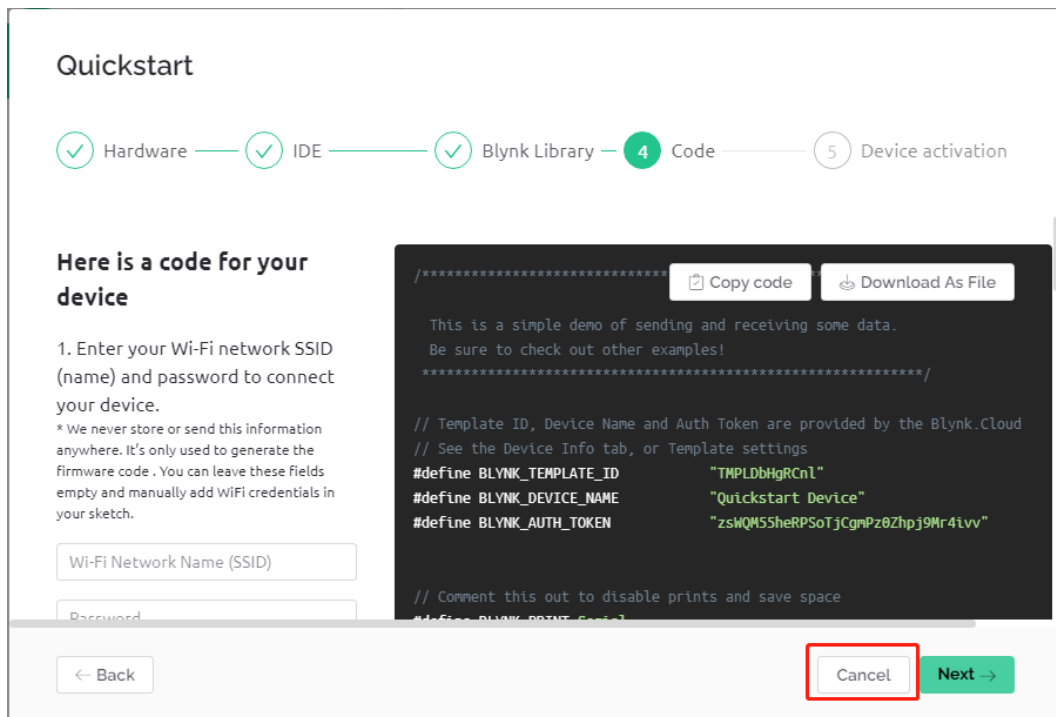
Download → Download →

← Back Cancel Next →

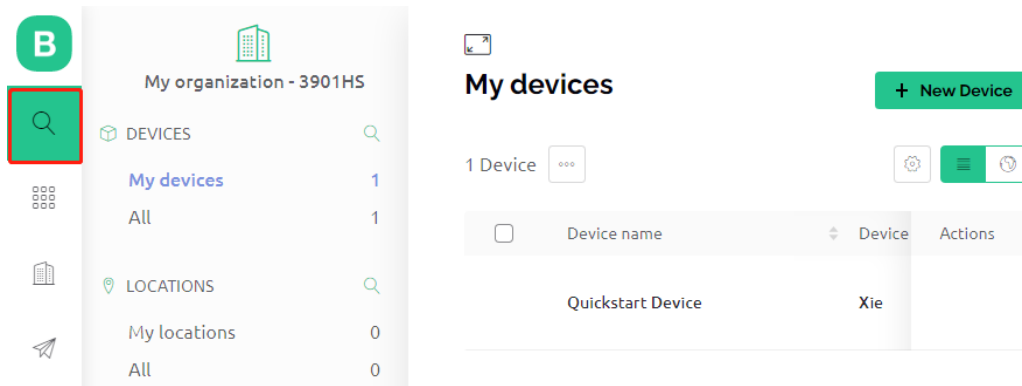
8. Here is the library you need to add, but the recommended library here is a bit problematic, we need to add other libraries manually (we will mention it later). Click **Next** here, and a new template and device will be created.



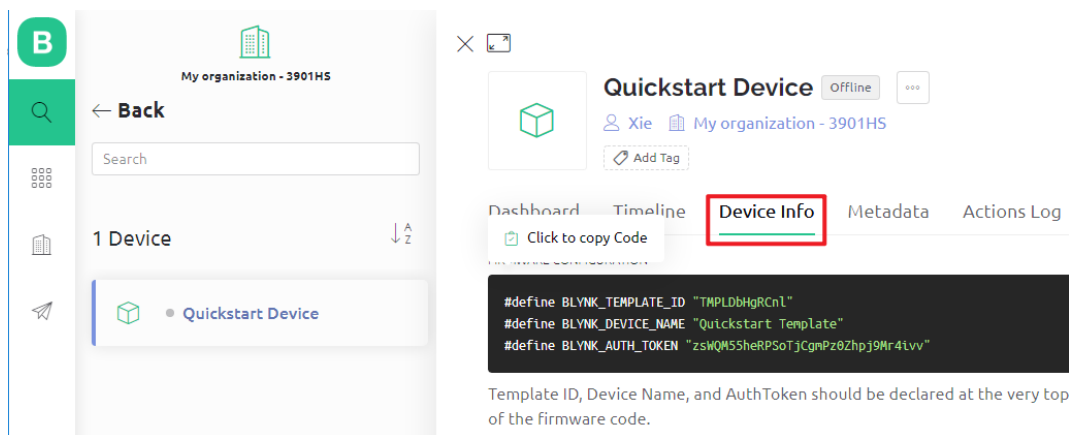
9. The next steps are to upload the relevant code and connect your board to Blynk, but since there is a problem with the library provided earlier, you need to add other libraries again. So click **Cancel** here to stop **Quick Start**.



10. Click the **Search** button and you will see the new device you just created.



11. Go to this **Quickstart Device** and click **Device Info**, you will see `TEMPLATE_ID`, `DEVICE_NAME`, and `AUTH_TOKEN` on the **Device info** page, and you will need to copy them later.

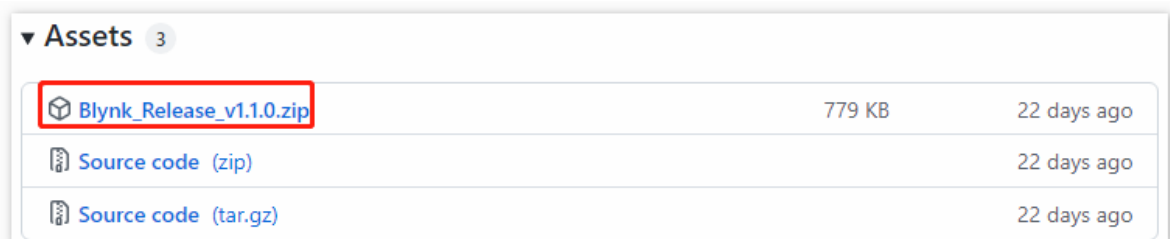


### 1.3 Adding the required libraries

You need to add the correct libraries for the Arduino IDE to use Blynk.

1. Click , scroll down to “**Assets**” and download the first .zip file.

**Note:** Please note that the version number shown in the image below may be outdated. We highly recommend downloading and installing the latest version available.

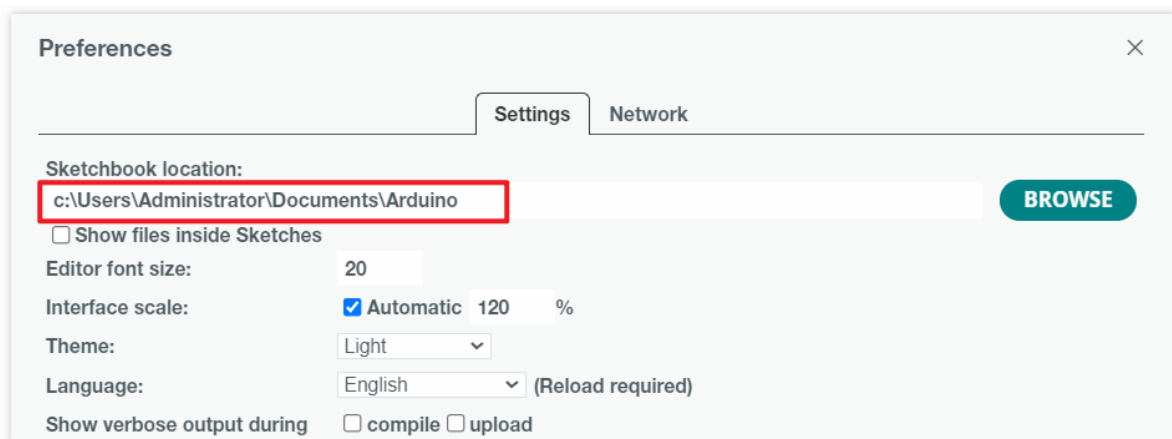


2. Unzip this file and then enter the `libraries` folder to see the following folders.

Name	Date modified	Type
Blynk	7/26/2023 3:51 PM	File folder
BlynkESP8266_Lib	7/26/2023 3:51 PM	File folder
BlynkNcpDriver	7/26/2023 3:51 PM	File folder
Time	7/26/2023 3:51 PM	File folder
TinyGSM	7/26/2023 3:51 PM	File folder

3. Copy them all and add them to the **libraries** folder of your sketchbook.

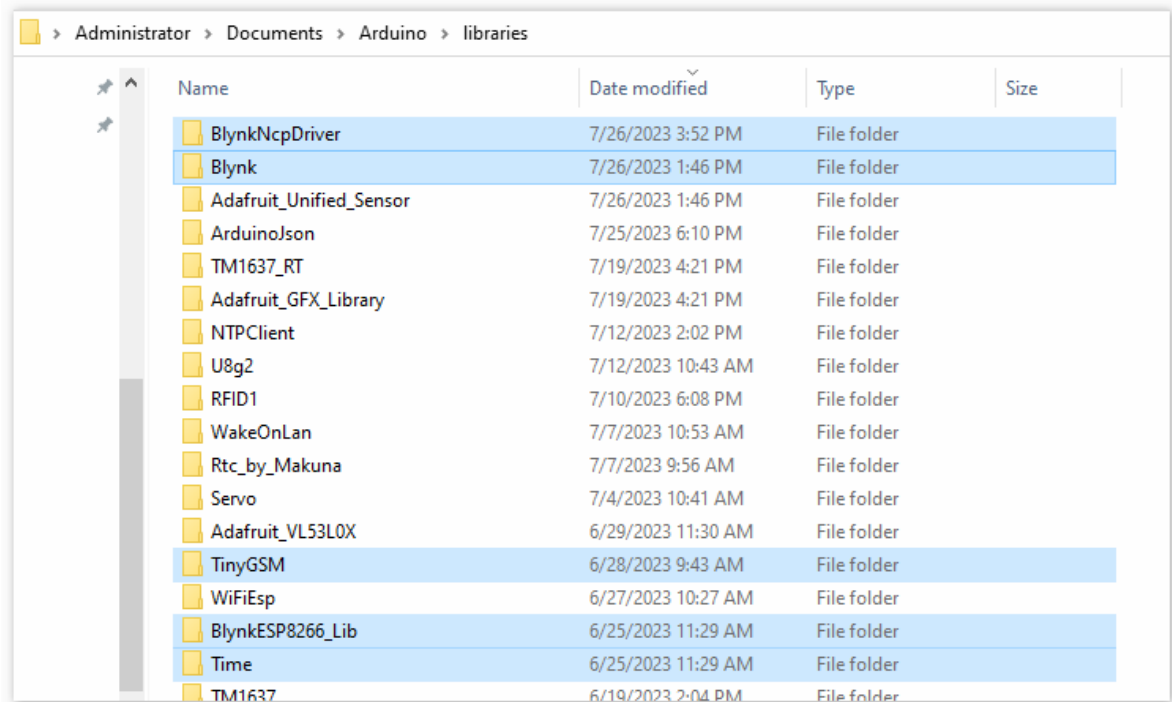
**Step 1:** You can find or change the location of your libraries folder at **File > Preferences > Sketchbook location**.



**Step 2:** Go to the location of your Sketchbook location(find from Arduino IDE). And find **libraries** folder, click to open it.

C:\Users\Administrator\Documents\Arduino\libraries				
<div><div><div></div><div></div><div></div></div></div>				
	Name	Date modified	Type	Size
	<div>Adafruit_BMP280_Library</div>	6/15/2023 11:40 AM	File folder	
	<div>Adafruit_BusIO</div>	6/13/2023 3:38 PM	File folder	
	<div>Adafruit_GFX_Library</div>	7/19/2023 4:21 PM	File folder	
	<div>Adafruit_MPU6050</div>	6/13/2023 3:38 PM	File folder	
	<div>Adafruit_SSD1306</div>	6/13/2023 3:38 PM	File folder	
	<div>Adafruit_Unified_Sensor</div>	7/26/2023 1:46 PM	File folder	

**Step 3:** Paste all the unzipped folders of **Blynk\_Release\_vx.x.x\libraries** into the libraries folder.



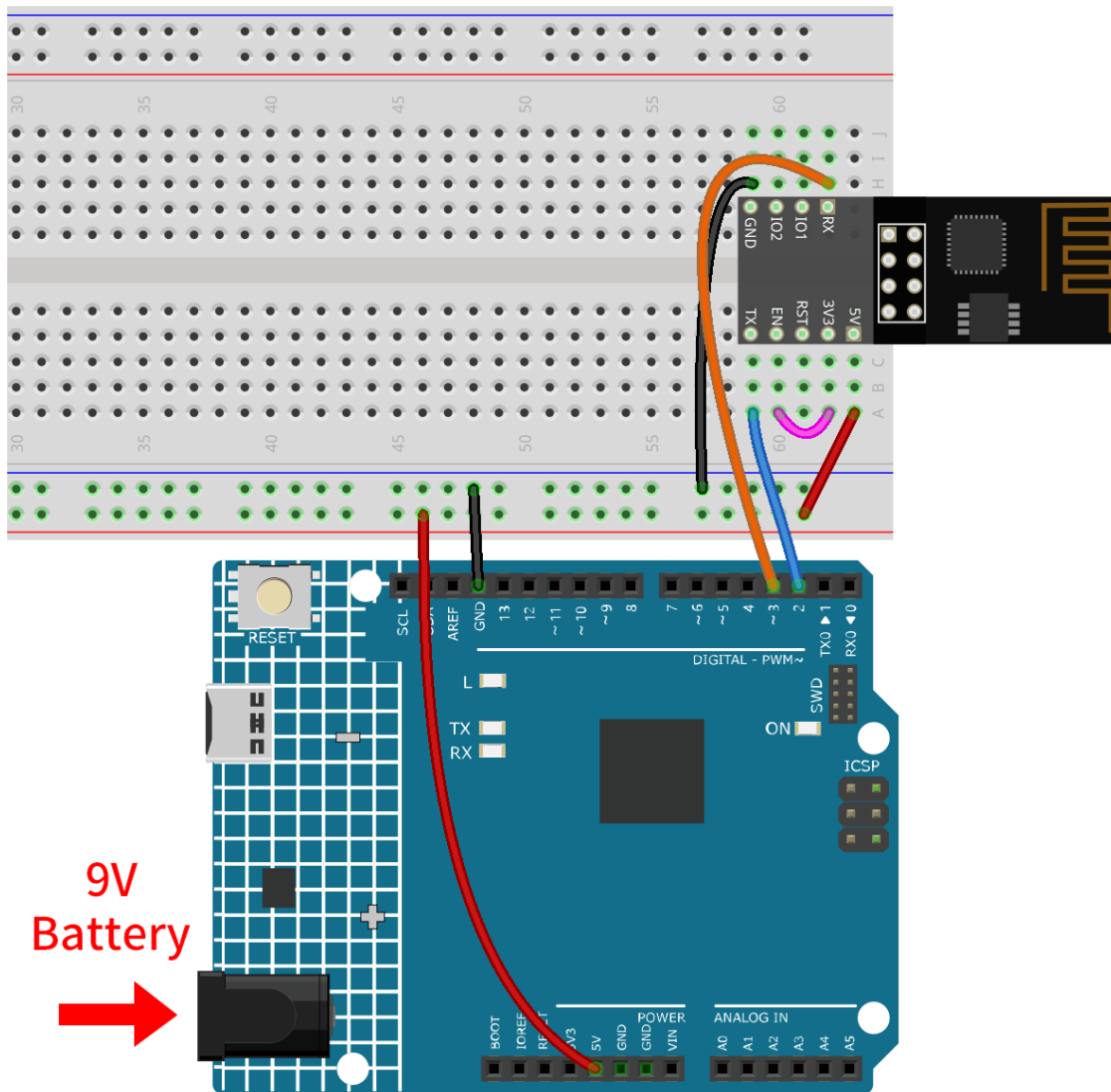
## 1.4 Connecting the R4 board to Blynk

1. Reconnect the ESP8266 module and R4 board, here the software serial is used, so TX and RX are connected to pins 2 and 3 of R4 board respectively.

---

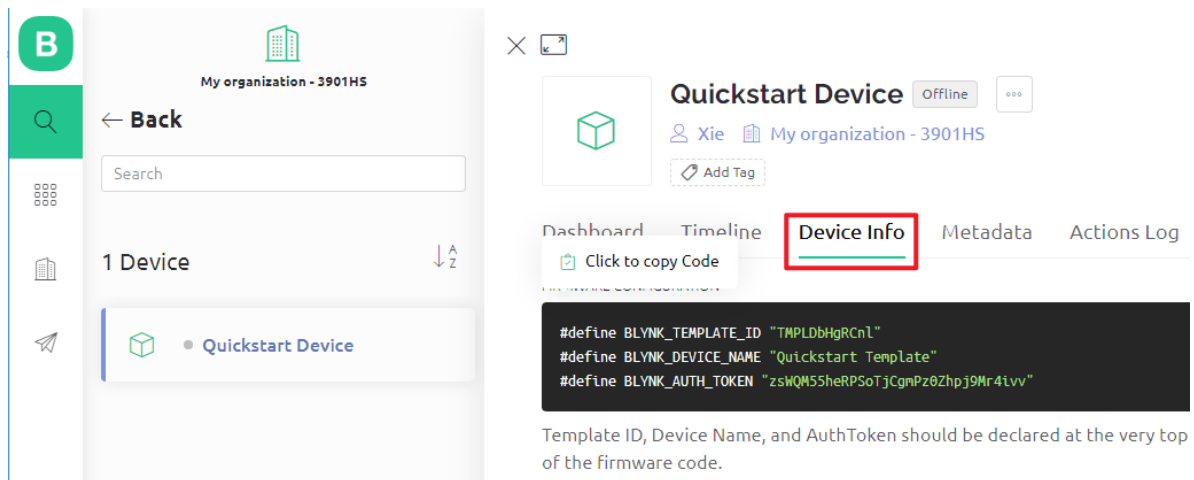
**Note:** The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.

---



1. Open the `00-Blynk_quick_start.ino` file under the path of `ultimate-sensor-kit\iot_project\wifi\00-Blynk_quick_start`. Or copy this code into **Arduino IDE**.
2. Replace the following three lines of code that you can copy from your account's **Device info** page. These three lines of code will allow your R4 board to find your blynk account.

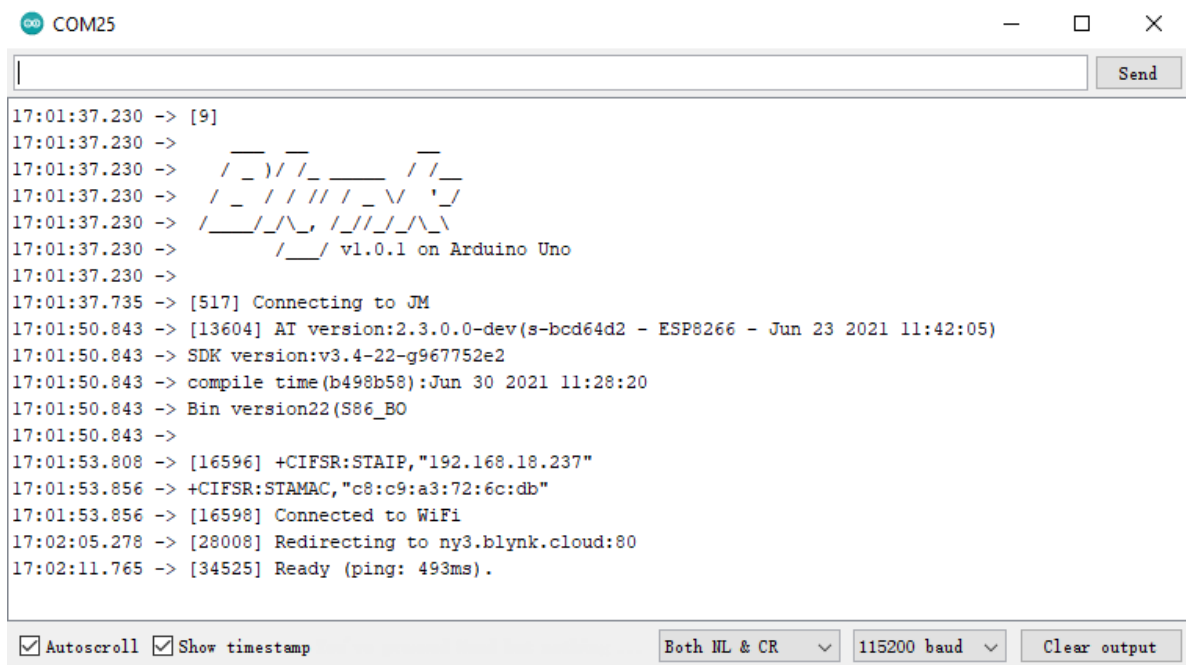
```
#define BLYNK_TEMPLATE_ID "TMPLxxxxxx"
#define BLYNK_DEVICE_NAME "Device"
#define BLYNK_AUTH_TOKEN "YourAuthToken"
```



3. Fill in the ssid and password of the WiFi you are using.

```
char ssid[] = "ssid";
char pass[] = "password";
```

4. Upload the code to the R4 board, then open the serial monitor and set the baud rate to 115200. when the R4 board communicates with Blynk successfully, the serial monitor will show the ready character.

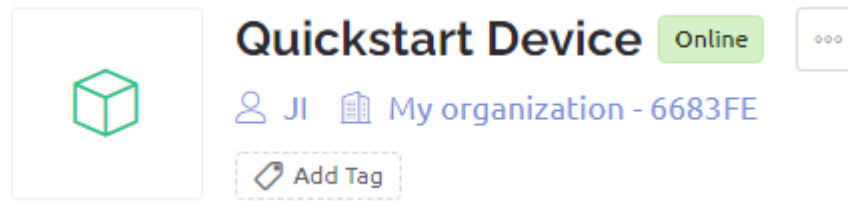


**Note:** If the message ESP is not responding appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

5. The status of Blynk will change from **offline** to **online**.



### 2.5.2 Flame Alert System with Blynk

In this chapter, we will guide you through the process of creating a home flame alarm system demo using Blynk. By utilizing a flame sensor, you can detect potential fires in your home. Sending the detected values to Blynk allows for remote monitoring of your home via the internet. In case of a fire, Blynk will promptly notify you via email.

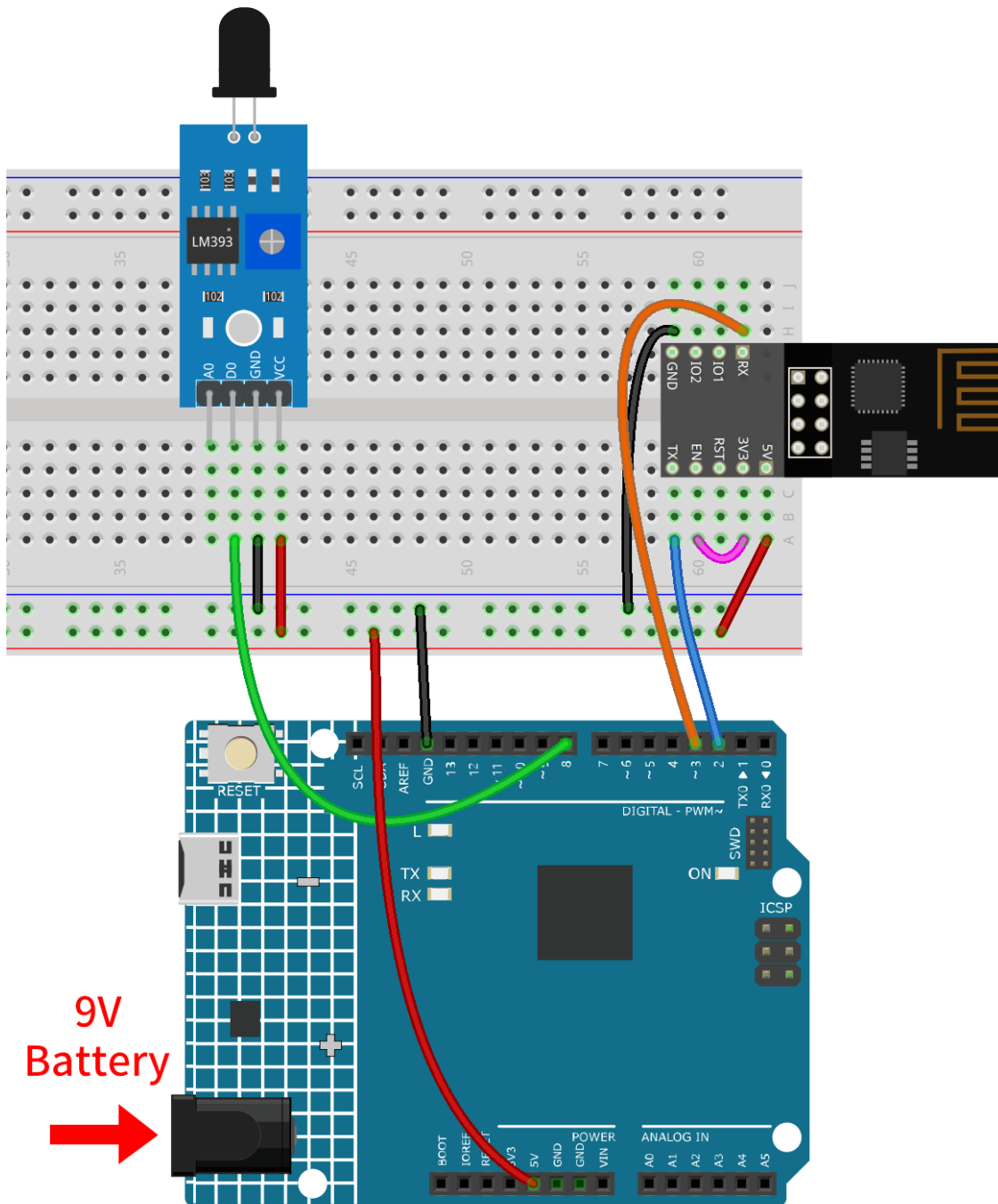
#### 1. Build the Circuit

---

**Note:** The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.

---



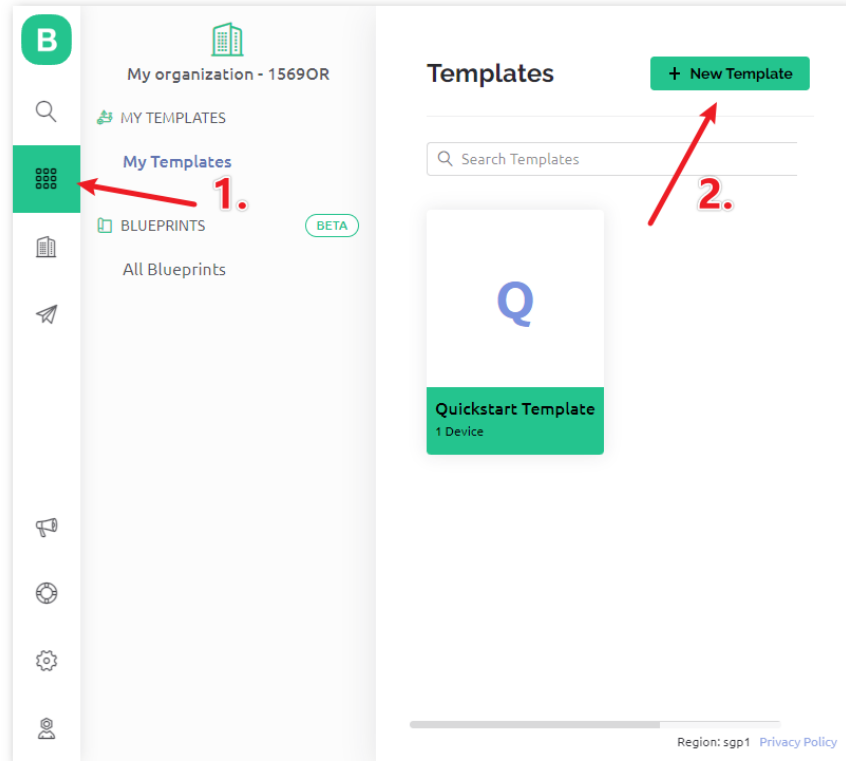


- *Arduino UNO R4 Minima Board*
- *Flame Sensor Module*
- *ESP8266 Module*

## 2. Configure Blynk

### 2.1 Create template

Firstly, we need to establish a template on Blynk. Follow the steps below to create a “**Flame Alert System**” template.



Ensure that the **HARDWARE** is configured as **ESP8266** and the **CONNECT TYPE** is set to **WiFi**.

**Create New Template**

NAME **1.**  
Flame Alert System

HARDWARE **2.** ▼ CONNECTION TYPE **3.** ▼  
ESP8266 WIFI

DESCRIPTION  
This is my template  
19 / 128

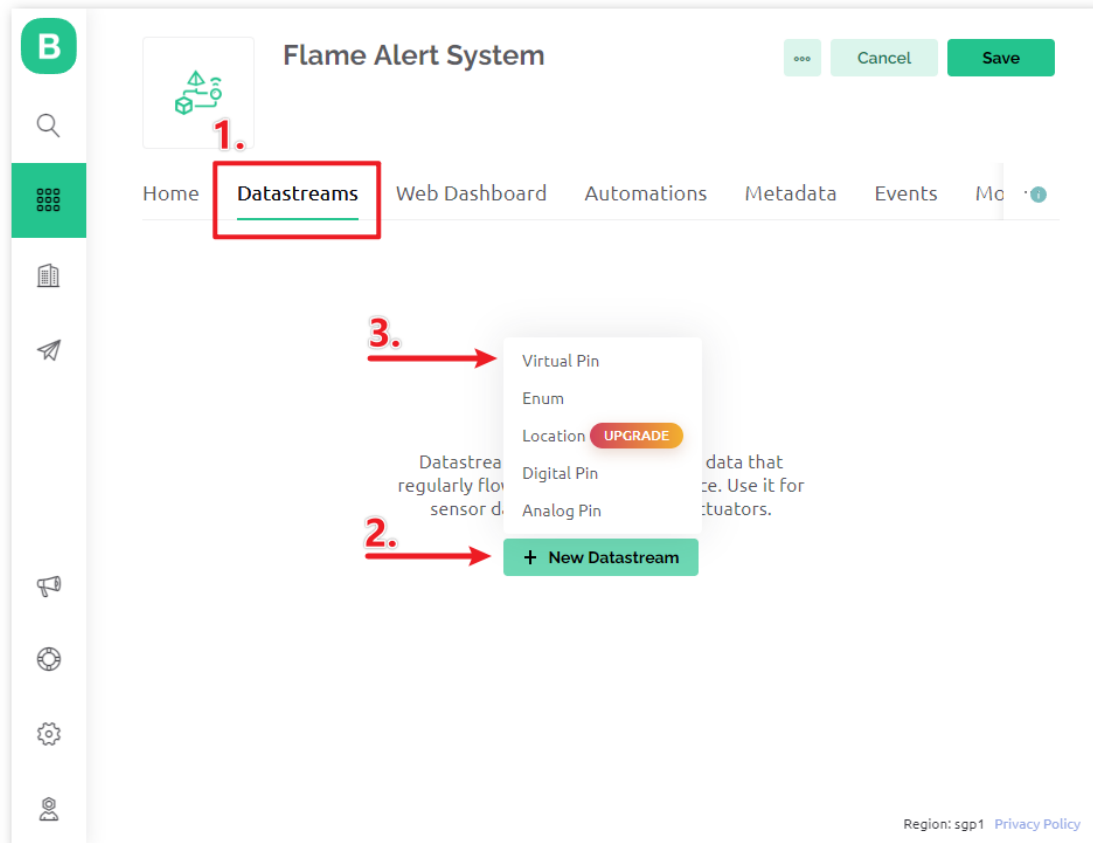
**4.** →

Cancel Done

Region: sgp1 [Privacy Policy](#)

## 2.2 Datastream

Create a **Datastream** of type **Virtual Pin** in the **Datastream** page to get the value of Flame sensor module.

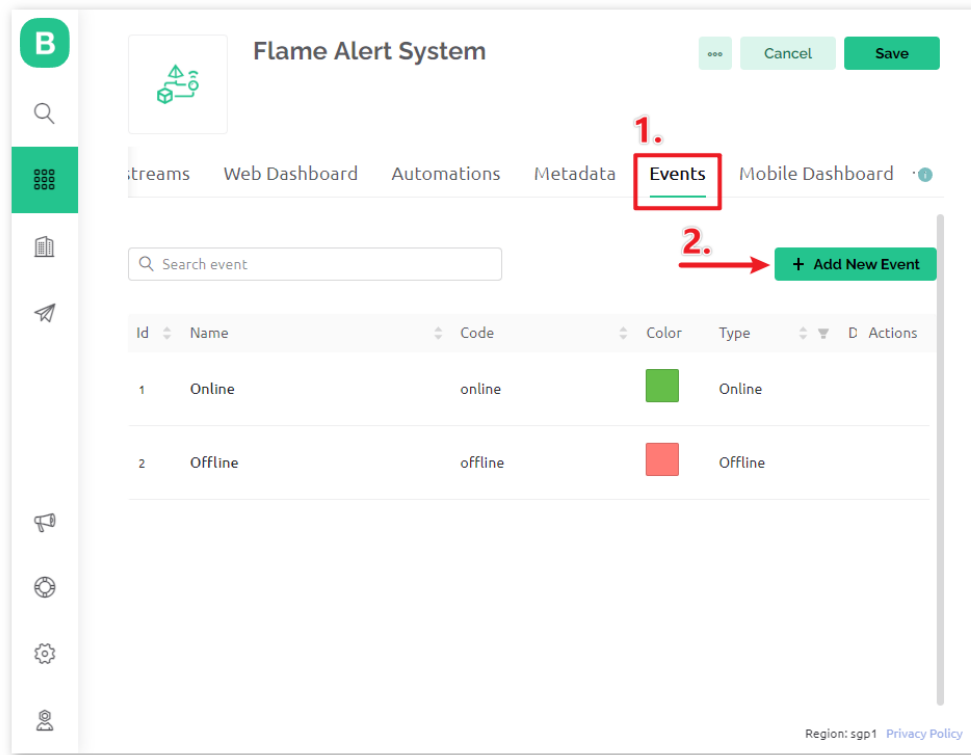


Set the name of the **Virtual Pin** to `flame_sensor_value`. Set the **DATA TYPE** to **Integer** and MIN and MAX to **0** and **1**.

The screenshot shows a web interface for a 'Flame Alert System'. A modal window titled 'Virtual Pin Datastream' is open. It contains several input fields: 'NAME' (highlighted with a red box and containing 'flame\_sensor\_value'), 'ALIAS' (containing 'flame sensor value'), 'PIN' (a dropdown menu showing 'V0'), 'DATA TYPE' (a dropdown menu showing 'Integer'), 'UNITS' (a dropdown menu showing 'None'), 'MIN' (containing '0'), 'MAX' (containing '1'), and 'DEFAULT VALUE' (containing '0'). Below these fields is a link for '+ ADVANCED SETTINGS'. At the bottom right of the modal are 'Cancel' and 'Create' buttons. The background interface includes a sidebar with icons and a top bar with 'Cancel' and 'Save' buttons.

## 2.3 Event

Next, we will create an **event** that logs the detection of flames and sends an email notification.



**Note:** It is recommended to keep it consistent with my settings, otherwise you may need to modify the code to run the project.

Set **EVENT NAME** to `flame_detection_alert`. At the same time, you can customize the content of email sent by setting **DESCRIPTION** for event triggering. You can also set frequency limits for event triggering below.

**Add New Event**

General Notifications

EVENT NAME **1.**  EVENT CODE

TYPE **2.** ☐ Info ☒ Warning ☐ Critical ☐ Content

DESCRIPTION (OPTIONAL) **3.**

Limit

Every  message will trigger the event

Event will be sent to user only once per  **4.**

Region: sgp1 [Privacy Policy](#)

Go to the **Notifications** page and configure email settings.

**Add New Event**

General **Notifications** **5.**

☒ **6.** Enable notifications

**Default recipients**

E-MAIL TO  **7.**

PUSH NOTIFICATIONS TO  **8.**

SMS TO

☐ Deliver push notifications as alerts

When turned on, push notifications will use critical alert sounds. End-users will need to turn this setting on in their app settings. They can also change a sound.

**Notifications Management**

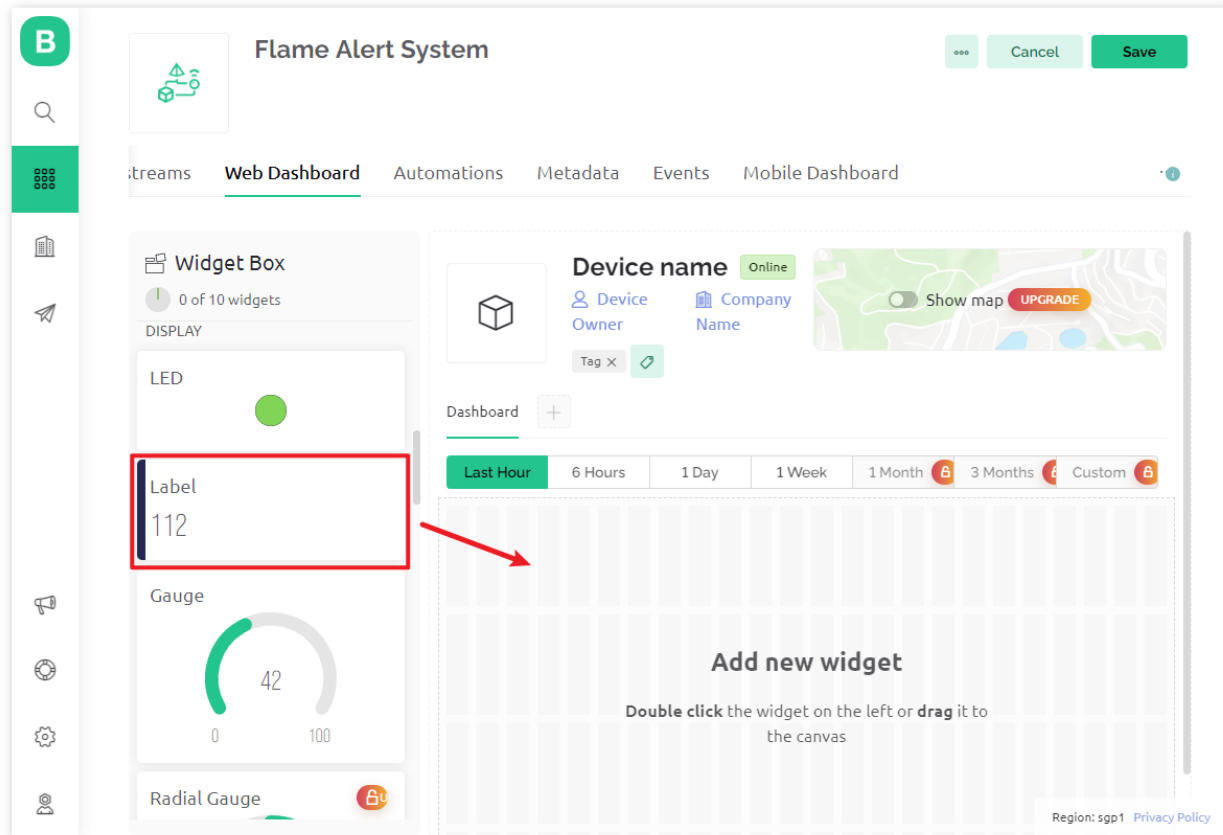
**9.**

Region: sgp1 [Privacy Policy](#)

## 2.4 Web Dashboard

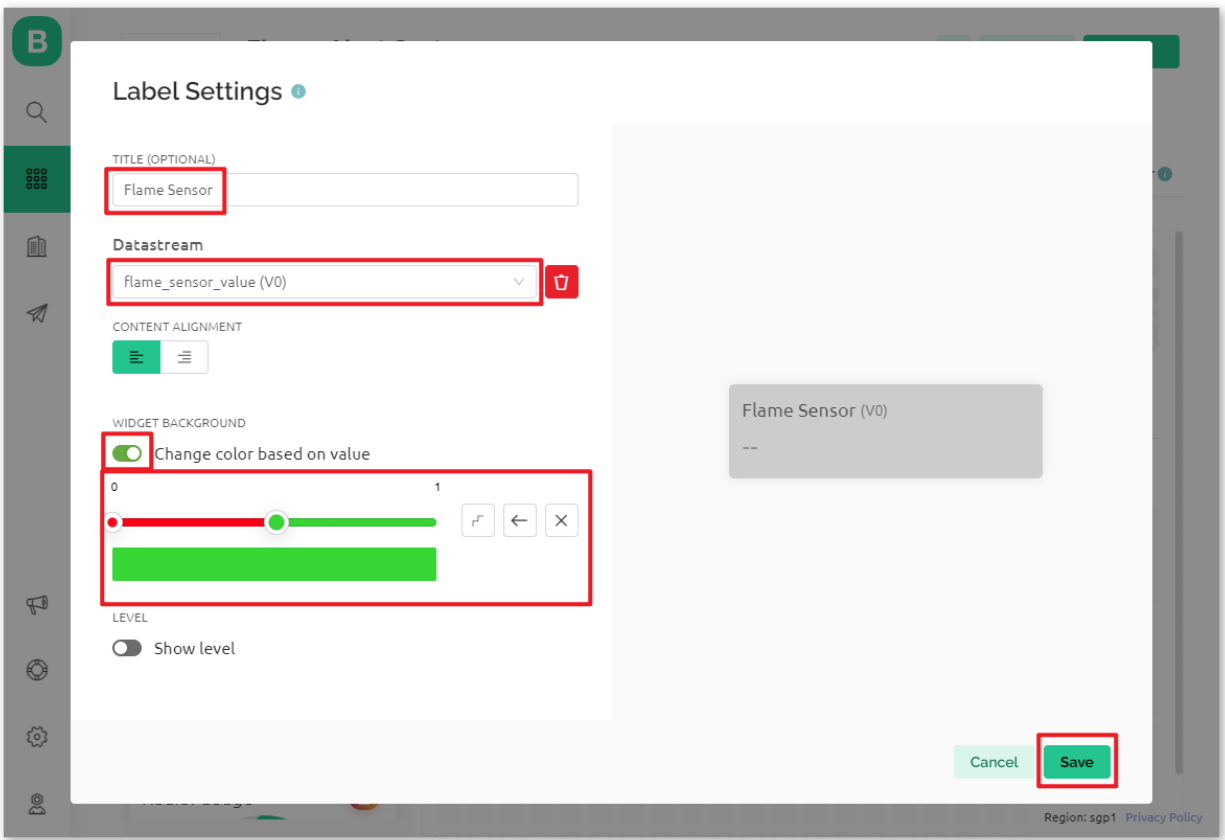
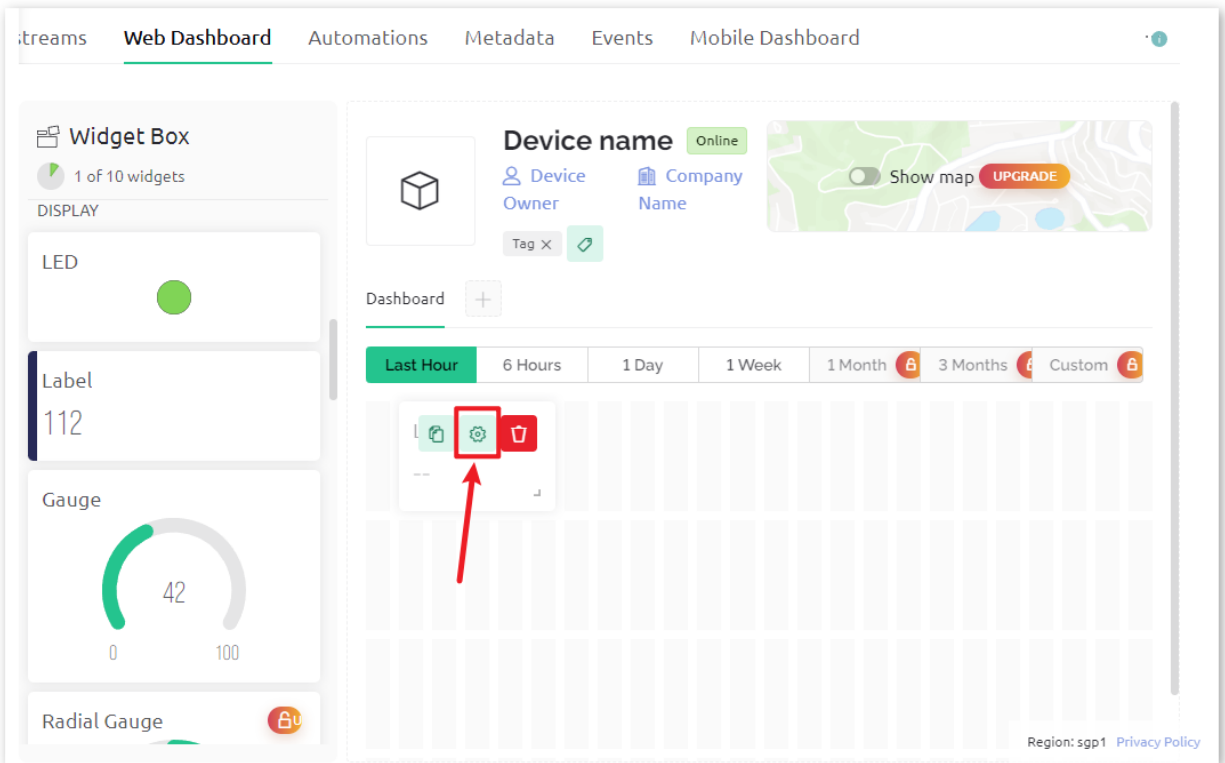
We also need to set up the **Web Dashboard** to display the sensor data sent from the Uno board.

Drag and drop an **Label widget** on the **Web Dashboard** page.



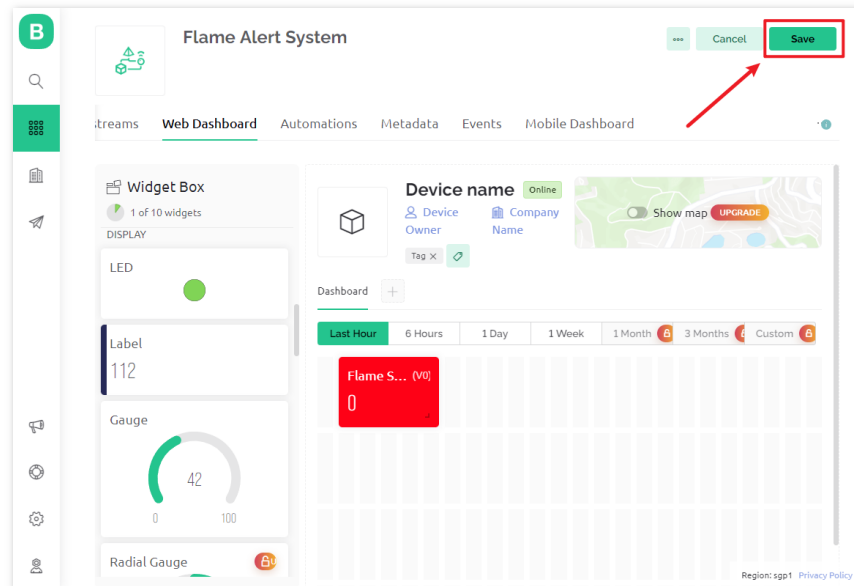
In the settings page of the **Label widget**, select **Datastream** as **flame\_sensor\_value(V0)**. Then set the color of **WIDGET BACKGROUND** to change with the value of data. When the displayed value is 1, it will be shown in green. When the value is 0, it will be shown in red.



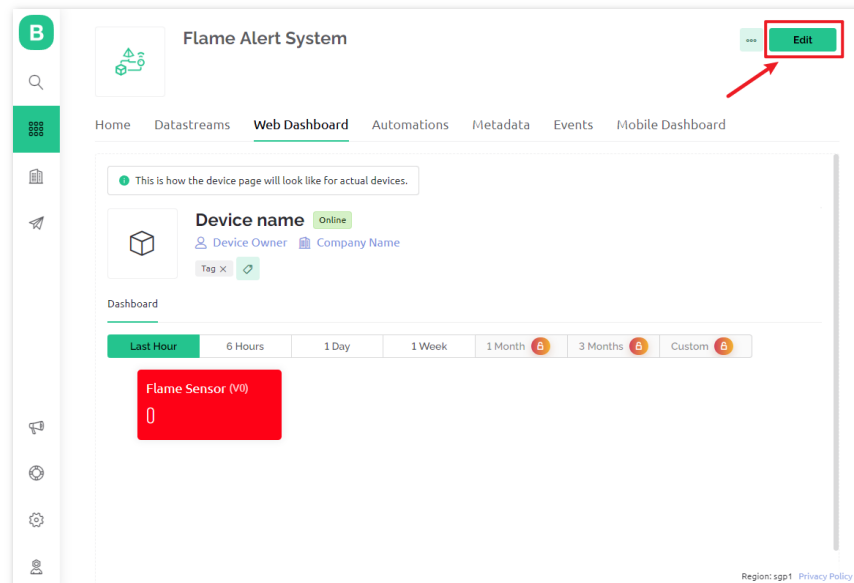


### 2.5 Save template

At last, remember to save the template.



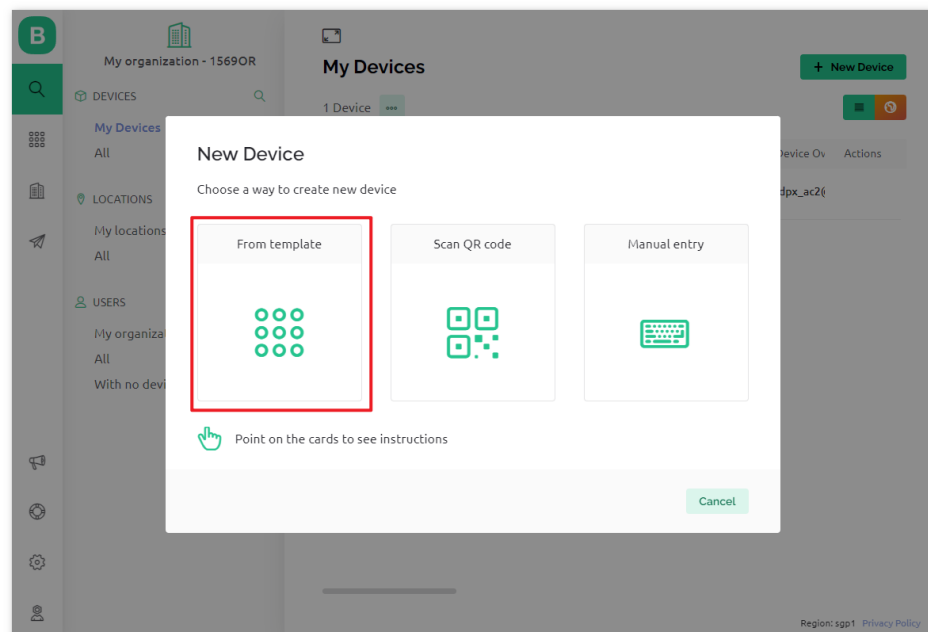
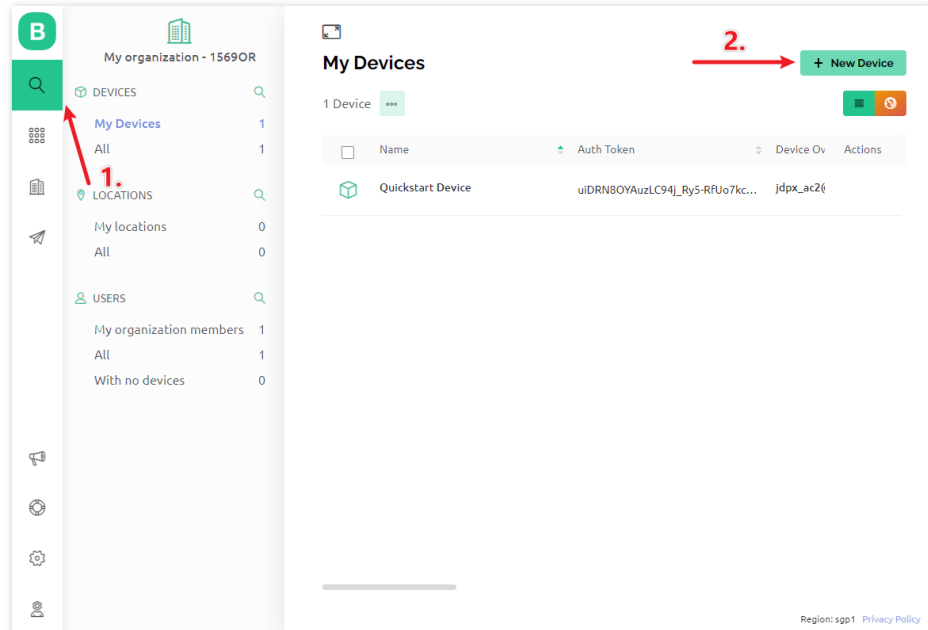
In case you need to edit the template, you can click on the edit button in the upper right corner.

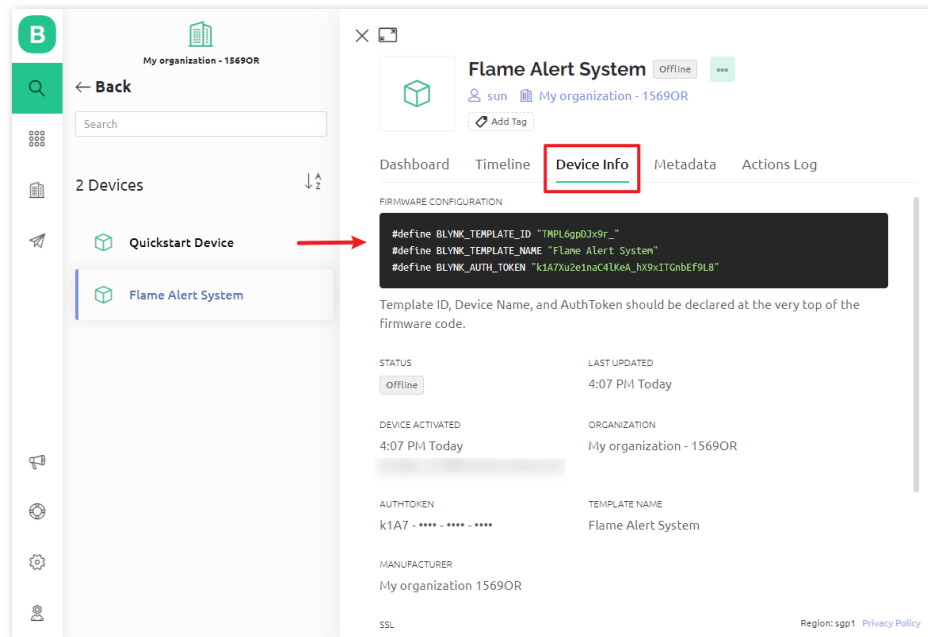
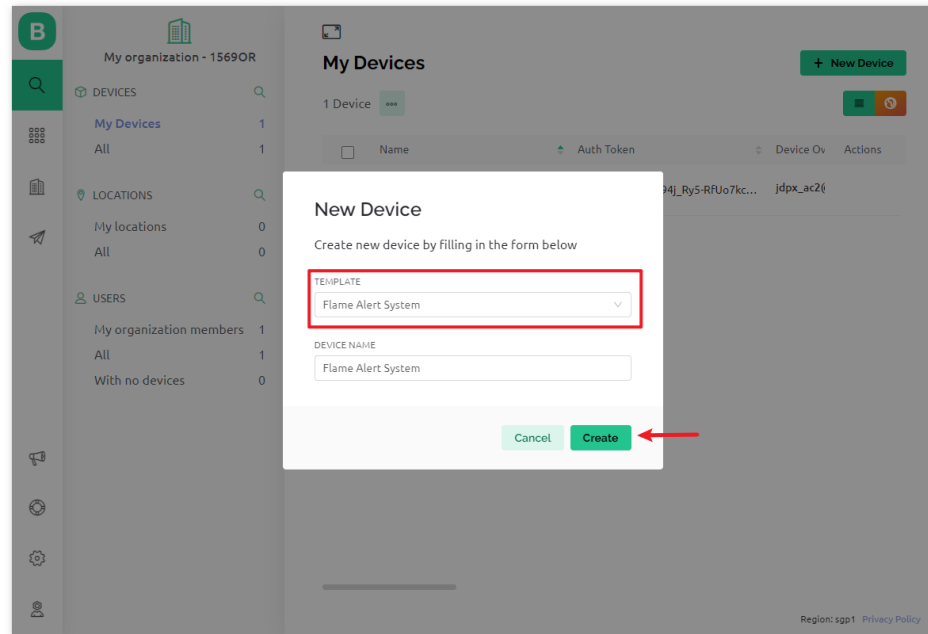


### 3. Run the Code

1. Open the 01-Flame\_alert\_system.ino file under the path of ultimate-sensor-kit\iot\_project\wifi\01-Flame\_alert\_system, or copy this code into **Arduino IDE**.
2. Create a Blynk device using the Flame Detection Alert template. Then, replace the BLYNK\_TEMPLATE\_ID, BLYNK\_TEMPLATE\_NAME, and BLYNK\_AUTH\_TOKEN with your own.

```
#define BLYNK_TEMPLATE_ID "TMPxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Flame Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxxx"
```





3. You also need to enter the ssid and password of the WiFi you are using.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to 115200) and wait for a prompt such as a successful connection to appear.

```

17:04:41.142 -> [4749] AT version:1.7.1.0(Jul 15 2019 16:58:04)
17:04:41.142 -> SDK version:3.0.1(78a3e33)
17:04:41.142 -> compile time:Feb 14 2020 09:19:42
17:04:41.142 -> OK
17:04:42.167 -> [5761] Failed to enable MUX
17:04:50.201 -> [13807] +CIFSR:STAIP,"192.168.1.1"
17:04:50.201 -> +CIFSR:STAMAC,"c8:c9:01:00:00:00"
17:04:50.201 -> [13807] Connected to WiFi
17:04:56.481 -> flame:1
17:04:57.403 -> [21002] Ready (ping: 931ms).

```

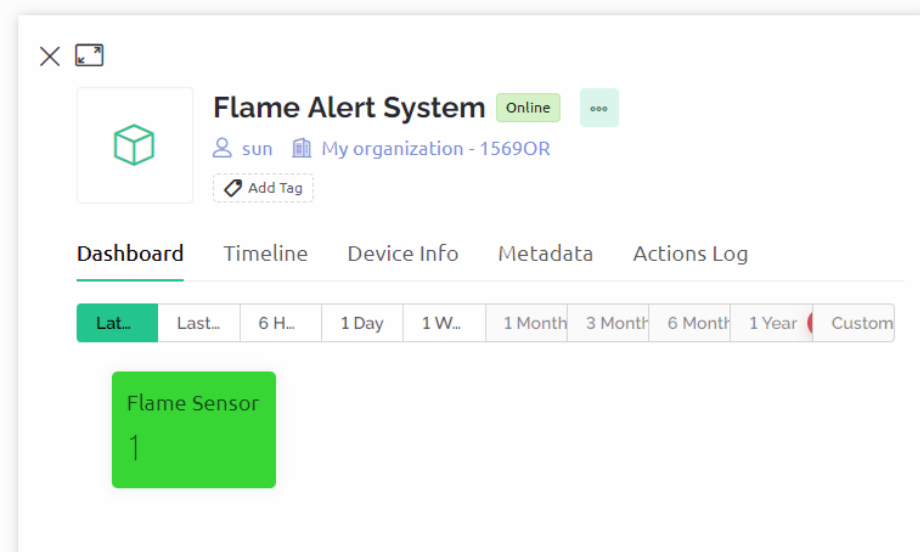
Ln 60, Col 15 Arduino UNO R4 Minima on COM19

**Note:** If the message ESP is not responding appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

- Now, Blynk will show the data read from flame sensor. In the label widget, you can see the value read by the flame sensor. When the displayed value is 1, the background of the label will be shown in green. When the value is 0, the background of the label will be shown in red and Blynk will send you an alert email.



- If you want to use Blynk on mobile devices, please refer to [How to use Blynk on mobile device?](#).

## 4. Code explanation

### 1. Library Initialization

Before we start, it's crucial to set up the necessary libraries and settings for communication between the Arduino, ESP8266 WiFi module, and Blynk app. This code sets up the required libraries and configures a software serial connection between the Arduino and ESP8266 module, with the appropriate baud rate for data transmission.

```
//Set debug prints on Serial Monitor
#define BLYNK_PRINT Serial

#include <ESP8266_Lib.h>           // Library for ESP8266
#include <BlynkSimpleShieldEsp8266.h> // Library for Blynk

// Software Serial on Uno
#include <SoftwareSerial.h>
SoftwareSerial EspSerial(2, 3); // RX, TX
#define ESP8266_BAUD 115200      // Set the ESP8266 baud rate
ESP8266 wifi(&EspSerial);
```

### 2. Blynk and WiFi configuration

For the project to communicate with the Blynk app, it needs to connect to a Wi-Fi network. The credentials need to be specified here.

```
// Template ID, Device Name and Auth Token are provided by the Blynk Cloud
// See the Device Info tab, or Template settings
#define BLYNK_TEMPLATE_ID "TMPxxxxxx"
#define BLYNK_TEMPLATE_NAME "Flame Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxxxxx"

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

### 3. Sensor Pin & Timer Declaration

Define the pin number for the flame. Blynk library provides a built-in timer, and we create a timer object. More about

```
const int sensorPin = 8;
BlynkTimer timer;
```

### 4. setup() Function

Initial configurations such as setting the pin mode for the sensorPin, initiating serial communication, setting the BlynkTimer, and connecting to the Blynk app are done in this function.

- We use `timer.setInterval(1000L, myTimerEvent)` to set the timer interval in `setup()`, here we set to execute the `myTimerEvent()` function every **1000ms**. You can modify the first parameter of `timer.setInterval(1000L, myTimerEvent)` to change the interval between `myTimerEvent` executions.

```
void setup() {
  pinMode(sensorPin, INPUT);
  Serial.begin(115200);
```

(continues on next page)

(continued from previous page)

```

EspSerial.begin(ESP8266_BAUD);
delay(1000);
timer.setInterval(1000L, myTimerEvent);
Blynk.config(wifi,BLYNK_AUTH_TOKEN);
Blynk.connectWiFi(ssid, pass);
}

```

### 5. loop() Function

The main loop runs the Blynk and Timer services continuously.

```

void loop() {
  Blynk.run();
  timer.run();
}

```

### 6. myTimerEvent() & sendData() Function

```

void myTimerEvent() {
  // Please don't send more than 10 values per second.
  sendData(); // Call function to send sensor data to Blynk app
}

```

The sendData() function reads the value from the flame sensor and sends it to Blynk. If it detects a flame (value 0), it sends flame\_detection\_alert event to the Blynk app.

- Use Blynk.virtualWrite(vPin, value) to send data to virtual pin V0 on Blynk. More about .
- Use Blynk.logEvent("event\_code") to log event to Blynk. More about .

```

void sendData() {
  int data = digitalRead(sensorPin);
  Blynk.virtualWrite(V0, data); // send data to virtual pin V0 on Blynk
  Serial.print("flame:");
  Serial.println(data); // Print flame status on Serial Monitor
  if (data == 0) {
    Blynk.logEvent("flame_alert"); // log flame alert event if sensor detects flame
  }
}

```

### Reference

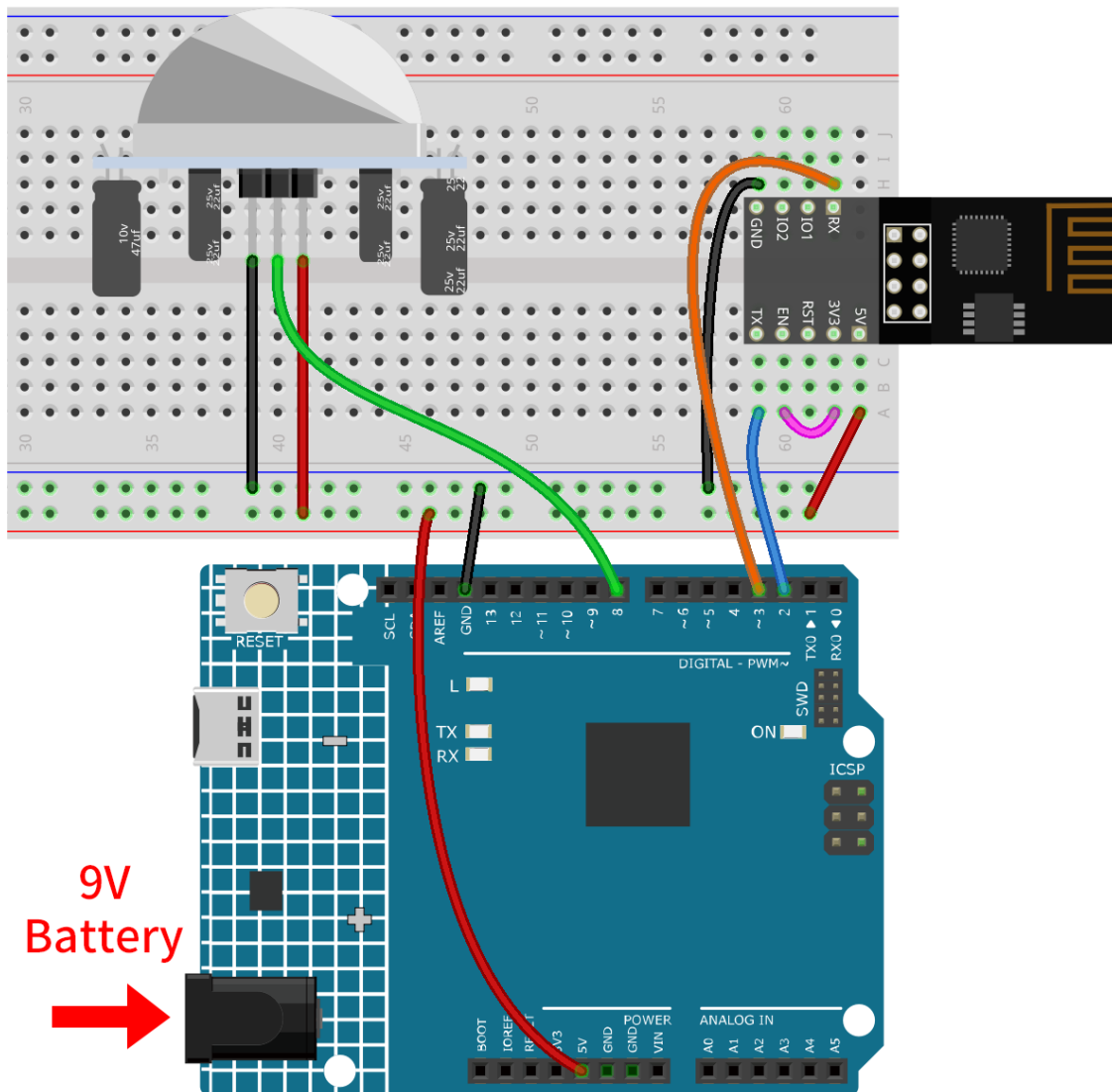
- 
- 
- 
- 
-

### 2.5.3 Intrusion Alert System with Blynk

This project demonstrate a simple home intrusion detection system using a passive infrared (PIR) sensor (HC-SR501). When the system is set to 'Away' mode through the Blynk app, the PIR sensor monitors for motion. Any detected movement triggers a notification on the Blynk app, alerting the user of potential intrusion.

#### 1. Build the Circuit

**Note:** The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.



- *Arduino UNO R4 Minima Board*
- *ESP8266 Module*
- *PIR Motion Module (HC-SR501)*



## 2. Configure Blynk

**Note:** If you are not familiar with Blynk, it is strongly recommended that you read these two tutorials first. [Get Started with Blynk](#) is a beginner's guide for Blynk, which includes how to configure ESP8266 and register with Blynk. And [Flame Alert System with Blynk](#) is a simple example, but the description of the steps will be more detailed.

### 2.1 Create template

Firstly, we need to establish a template on Blynk. Follow the steps below to create a “**Intrusion Alert System**” template.

The screenshot shows the Blynk mobile app interface for creating a new template. The dialog box is titled "Create New Template". It has the following fields:

- NAME:** A text input field containing "Intrusion Alert System".
- HARDWARE:** A dropdown menu showing "ESP8266".
- CONNECTION TYPE:** A dropdown menu showing "WIFI".
- DESCRIPTION:** A text area containing "This is my template".

At the bottom right of the dialog are two buttons: "Cancel" and "Done". Below the description field, the text "19 / 128" indicates the character count. In the background, a "Quickstart Template" button is visible. At the very bottom of the screen, it says "Region: sgp1" and "Privacy Policy".

### 2.2 Datastream

Create **Datastreams** of type **Virtual Pin** in the **Datastream** page receive data from esp8266 and uno r4 board.

- Create Virtual Pin V0 according to the following diagram:

Set the name of the **Virtual Pin V0** to **AwayMode**. Set the **DATA TYPE** to **Integer** and MIN and MAX to **0** and **1**.

The screenshot shows a web interface for configuring a 'Virtual Pin Datastream'. The main title is 'Intrusion Alert System'. Below it, the 'Virtual Pin Datastream' configuration form is displayed. The form includes the following fields:

- NAME:** A text input field containing 'AwayMode'.
- ALIAS:** A text input field containing 'AwayMode'.
- PIN:** A dropdown menu showing 'V0'.
- DATA TYPE:** A dropdown menu showing 'Integer'.
- UNITS:** A dropdown menu showing 'None'.
- MIN:** A text input field containing '0'.
- MAX:** A text input field containing '1'.
- DEFAULT VALUE:** A text input field containing '0'.

At the bottom of the form, there is a checkbox labeled 'ADVANCED SETTINGS' and two buttons: 'Cancel' and 'Create'.

- Create Virtual Pin V1 according to the following diagram:

Set the name of the **Virtual Pin V1** to **Current status**. Set the **DATA TYPE** to **String**.

B

Intrusion Alert System

...

Cancel

Save

### Virtual Pin Datastream

NAME

Current status

ALIAS

Current status

PIN

V1

DATA TYPE

String

DEFAULT VALUE

Default Value

+

 ADVANCED SETTINGS

Cancel

Create

Region: sgp1

Privacy Policy

Make sure that you have set up two Virtual Pins according to the steps above.

B

Intrusion Alert System

...

Cancel

Save

Home

**Datastreams**

Web Dashboard

Automations

Metadata

Events

Mobile Dashboard

Search datastream

+ New Datastream

2 Datastreams

<input type="checkbox"/>	Id	Name	Alias	Color	Pin	Data Type	Units	Is Raw	Min	Max	Actions
<input checked="" type="checkbox"/>	1	AwayMode	AwayMode		V0	Integer		false	0	1	
<input checked="" type="checkbox"/>	2	Current status	Current status		V1	String		false			

Region: sgp1

Privacy Policy

### 2.3 Event

Next, we will create an **event** that logs the detection of intrusion and sends an email notification.

**Note:** It is recommended to keep it consistent with my settings, otherwise you may need to modify the code to run the project. Make sure that the **EVENT CODE** is set as `intrusion_detected`.

**Add New Event**

General Notifications

EVENT NAME: Intrusion Detected

EVENT CODE: intrusion\_detected

TYPE: Info **Warning** Critical Content

DESCRIPTION (OPTIONAL): Intrusion detected, please confirm!

Limit: Every 1 message will trigger the event. Event will be sent to user only once per 5 minutes.

Cancel Create

Go to the **Notifications** page and configure email settings.

**Add New Event**

General **Notifications**

☒ Enable notifications

**Default recipients**

E-MAIL TO  
Device Owner X

PUSH NOTIFICATIONS TO  
Device Owner X

SMS TO  
Select contact

☐ Deliver push notifications as alerts  
When turned on, push notifications will use critical alert sounds. End-users will need to turn this setting on in their app settings. They can also change a sound.

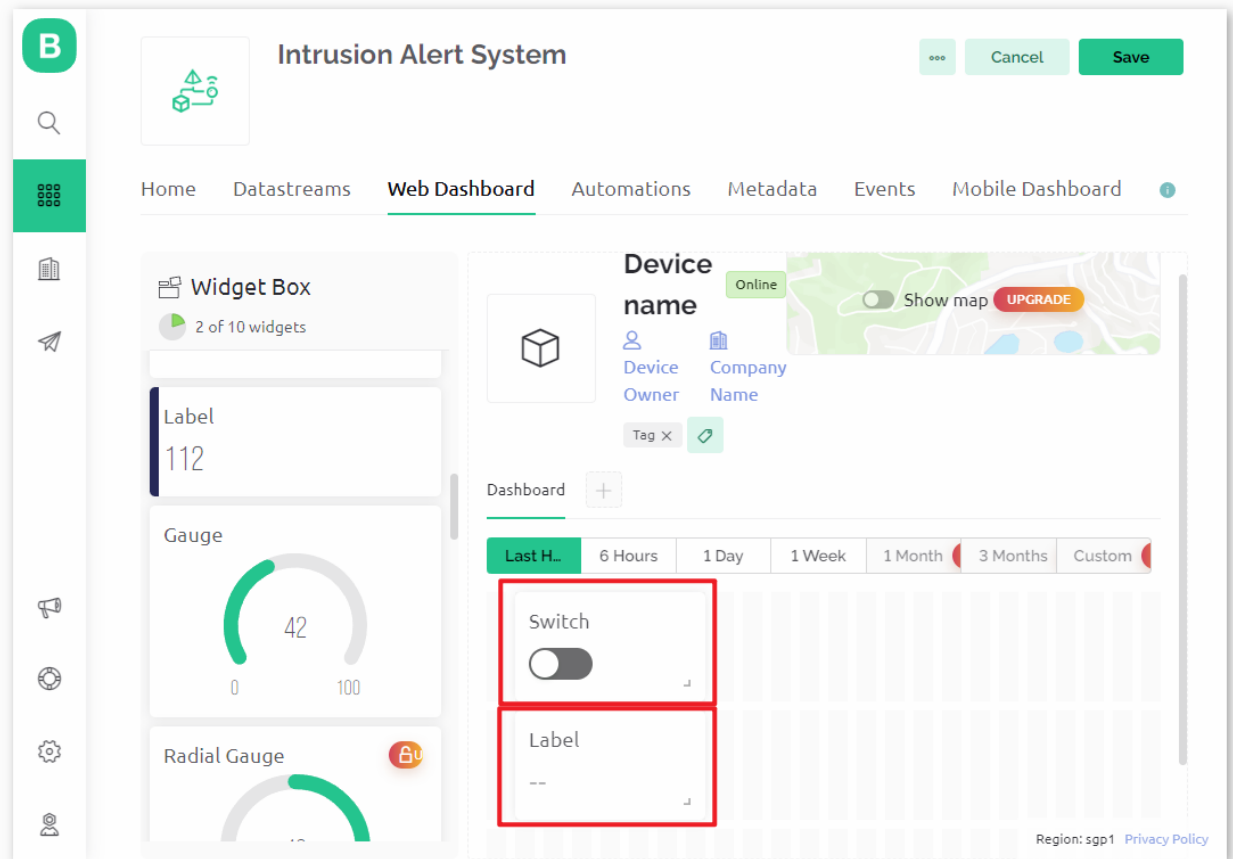
**Notifications Management**

Cancel Create

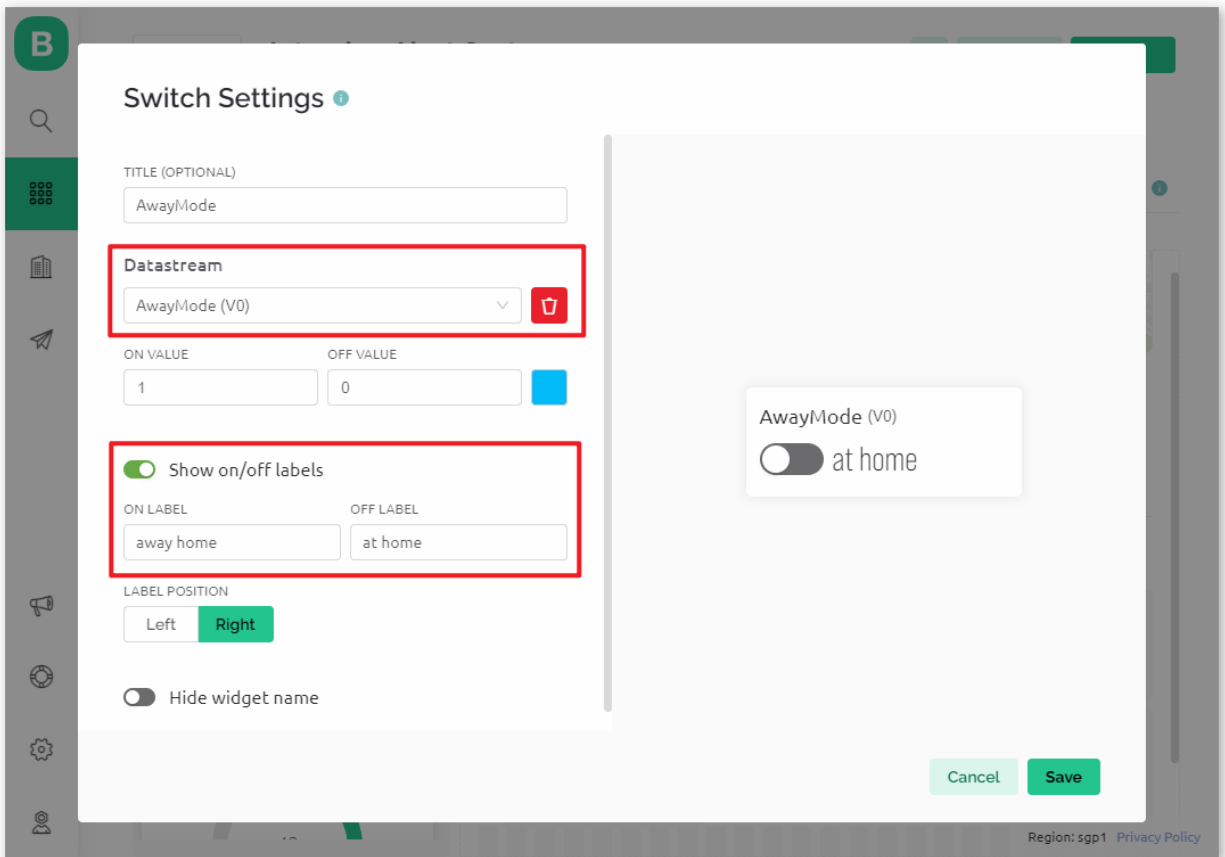
## 2.4 Web Dashboard

We also need to configure the **Web Dashboard** to interact with the Intrusion Alert System.

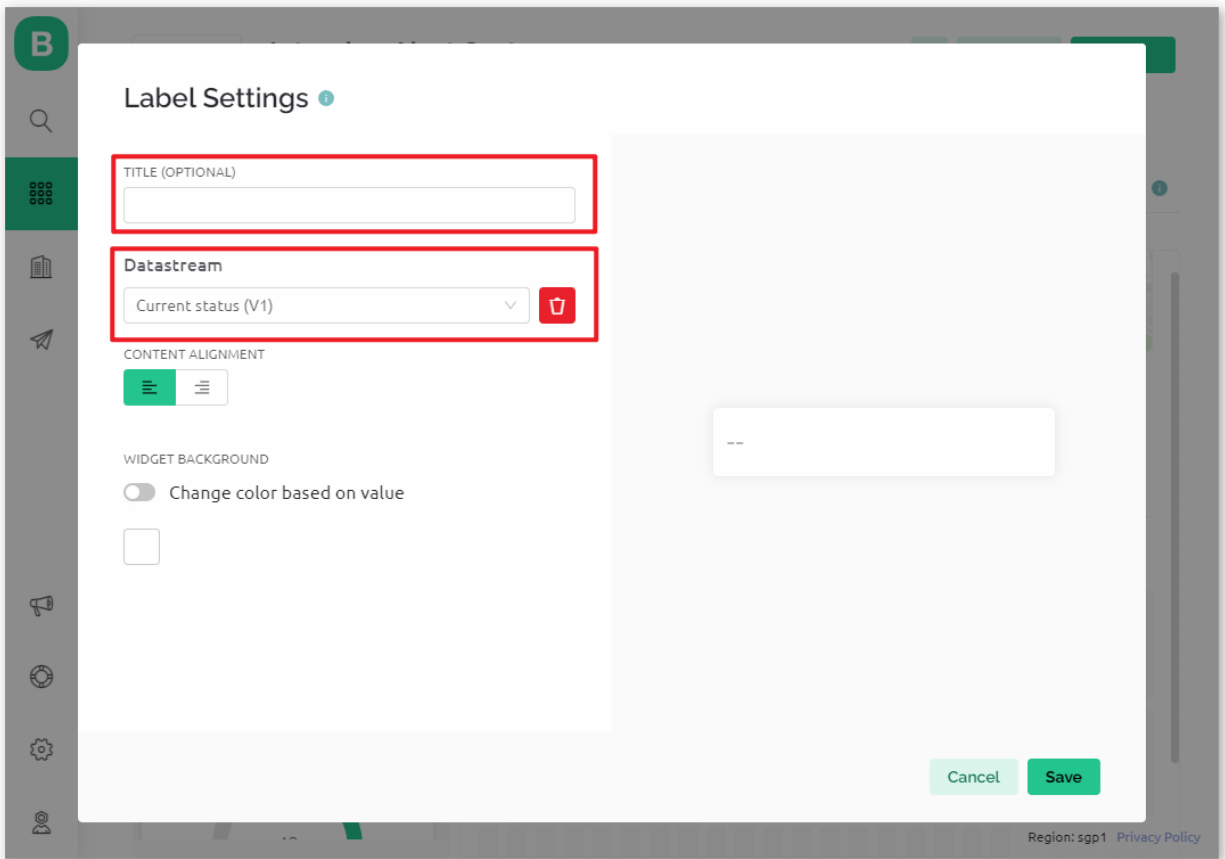
Drag and drop a **Switch widget** and a **Label widget** to the **Web Dashboard** page.



In the settings page of the **Switch widget**, select **Datastream** as **AwayMode(V0)**. Set **ONLABEL** and **OFFLABEL** to display “away home” when the switch is turned on, and “at home” when the switch is turned off.

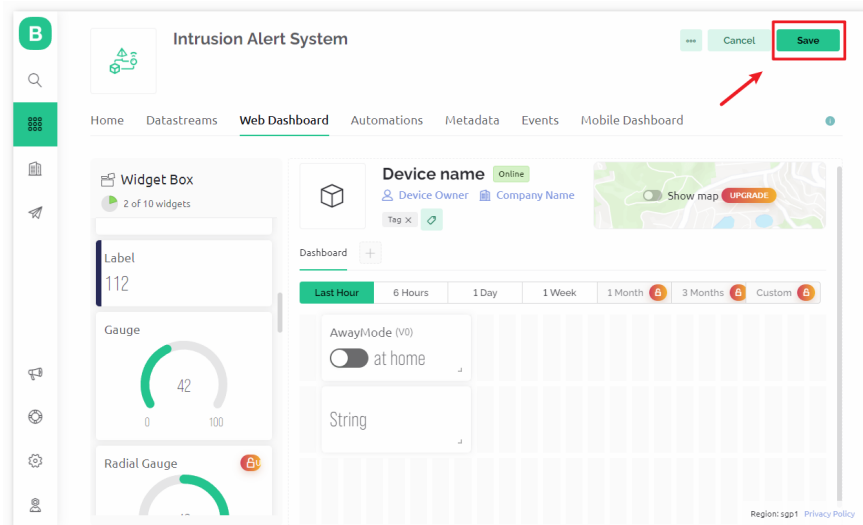


In the settings page of the **Label widget**, select **Datastream** as **Current status(V1)**.



## 2.5 Save template

At last, remember to save the template.





### 3. Run the Code

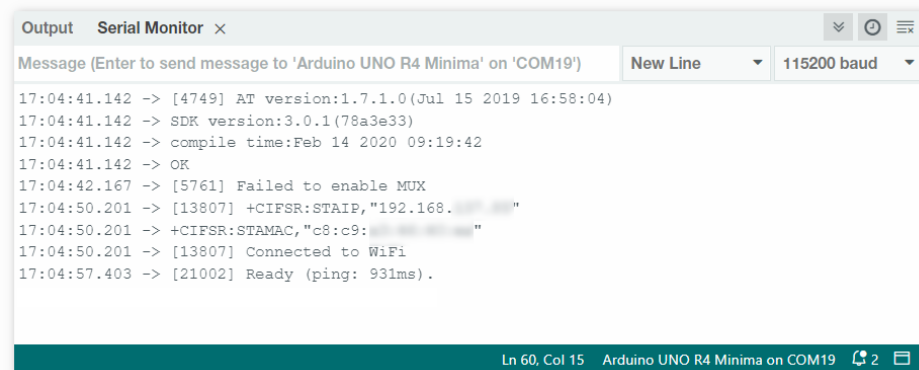
1. Open the 02-Intrusion\_alert\_system.ino file under the path of ultimate-sensor-kit\iot\_project\wifi\02-Intrusion\_alert\_system, or copy this code into **Arduino IDE**.
2. Create a Blynk device using the "Intrusion Alert System" template. Then, replace the BLYNK\_TEMPLATE\_ID, BLYNK\_TEMPLATE\_NAME, and BLYNK\_AUTH\_TOKEN with your own.

```
#define BLYNK_TEMPLATE_ID "TMPxxxxxx"
#define BLYNK_TEMPLATE_NAME "Intrusion Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxxx"
```

3. You also need to enter the ssid and password of the WiFi you are using.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to 115200) and wait for a prompt such as a successful connection to appear.



**Note:** If the message ESP is not responding appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

### 4. Code explanation

#### 1. Configuration & Libraries

Here, constants and credentials for Blynk are set up. Necessary libraries for the ESP8266 WiFi module and Blynk are included.

```
#define BLYNK_TEMPLATE_ID "TMPxxxx"
#define BLYNK_TEMPLATE_NAME "Intrusion Alert System"
#define BLYNK_AUTH_TOKEN "xxxxxx-"
#define BLYNK_PRINT Serial
```

(continues on next page)

(continued from previous page)

```
#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>
```

## 2. WiFi Setup

Configure WiFi credentials and set up software serial communication with the ESP01 module.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";

SoftwareSerial EspSerial(2, 3);
#define ESP8266_BAUD 115200
ESP8266 wifi(&EspSerial);
```

## 3. PIR Sensor Configuration

Define the pin where the PIR sensor is connected and initialize state variables.

```
const int sensorPin = 8;
int state = 0;
int awayHomeMode = 0;
BlynkTimer timer;
```

## 4. setup() Function

This initializes the PIR sensor as an input, sets up serial communication, connects to WiFi, and configures Blynk.

- We use `timer.setInterval(1000L, myTimerEvent)` to set the timer interval in `setup()`, here we set to execute the `myTimerEvent()` function every **1000ms**. You can modify the first parameter of `timer.setInterval(1000L, myTimerEvent)` to change the interval between `myTimerEvent` executions.

```
void setup() {
  pinMode(sensorPin, INPUT);
  Serial.begin(115200);
  EspSerial.begin(ESP8266_BAUD);
  delay(10);
  Blynk.config(wifi, BLYNK_AUTH_TOKEN);
  Blynk.connectWiFi(ssid, pass);
  timer.setInterval(1000L, myTimerEvent);
}
```

## 5. loop() Function

The loop function repeatedly runs Blynk and the Blynk timer functions.

```
void loop() {
  Blynk.run();
  timer.run();
}
```

## 6. Blynk App Interaction

These functions are called when the device connects to Blynk and when there's a change in the state of the virtual pin V0 on the Blynk app.

- Every time the device connects to the Blynk server, or reconnects due to poor network conditions, the BLYNK\_CONNECTED() function is called. The Blynk.syncVirtual() command request a single Virtual Pin value. The specified Virtual Pin will perform BLYNK\_WRITE() call. Please refer to for more details.
- Whenever the value of a virtual pin on the BLYNK server changes, it will trigger BLYNK\_WRITE(). More details at .

```
// This function is called every time the device is connected to the Blynk.Cloud
BLYNK_CONNECTED() {
  Blynk.syncVirtual(V0);
}

// This function is called every time the Virtual Pin 0 state changes
BLYNK_WRITE(V0) {
  awayHomeMode = param.asInt();
  // additional logic
}
```

## 7. Data Handling

Every second, the myTimerEvent() function calls sendData(). If the away mode is enabled on Blynk, it checks the PIR sensor and sends a notification to Blynk if motion is detected.

- We use Blynk.virtualWrite(V1, "Somebody in your house! Please check!"); to change the text of a label.
- Use Blynk.logEvent("intrusion\_detected"); to log event to Blynk.

```
void myTimerEvent() {
  sendData();
}

void sendData() {
  if (awayHomeMode == 1) {
    state = digitalRead(sensorPin); // Read the state of the PIR sensor

    Serial.print("state:");
    Serial.println(state);

    // If the sensor detects movement, send an alert to the Blynk app
    if (state == HIGH) {
      Serial.println("Somebody here!");
      Blynk.virtualWrite(V1, "Somebody in your house! Please check!");
      Blynk.logEvent("intrusion_detected");
    }
  }
}
```

## Reference

- 
- 
- 
- 
-

- 
- 

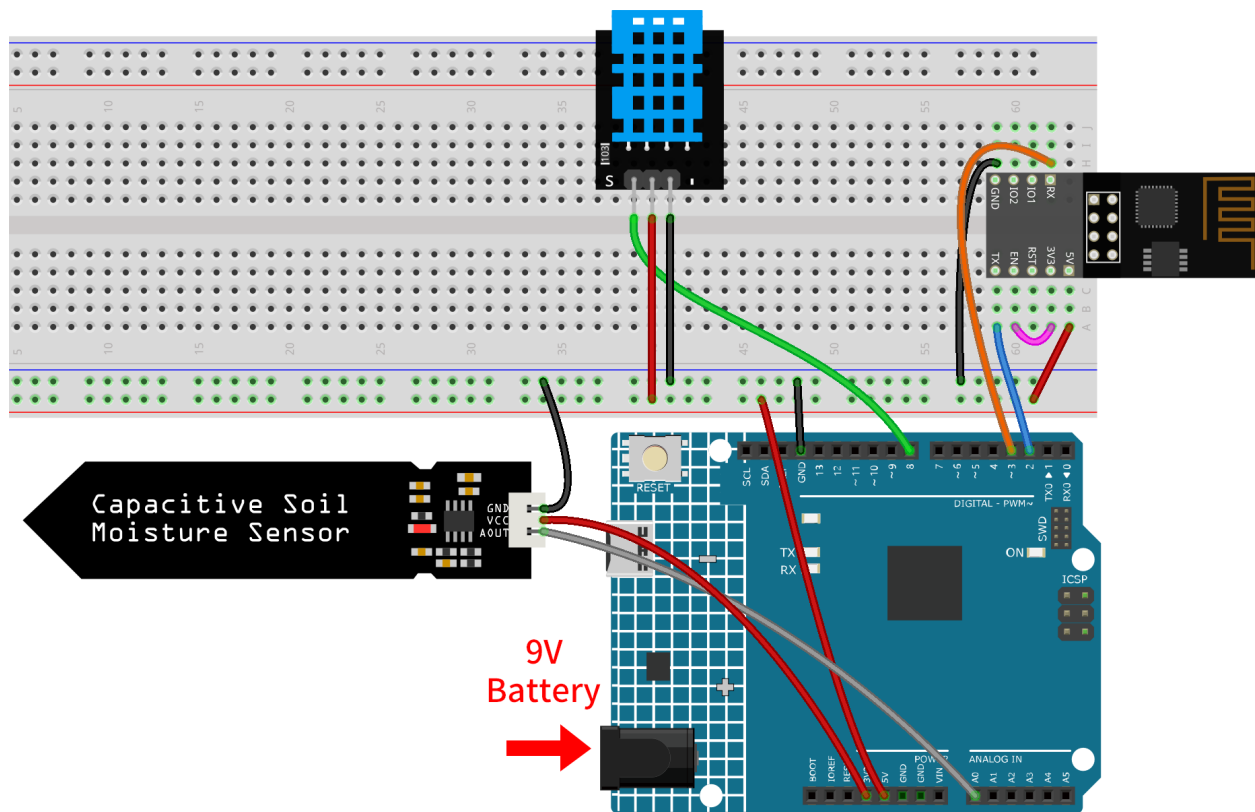
### 2.5.4 Plant Monitor with Blynk

This project creates a plant monitoring demo system that detects the current temperature, humidity, light intensity, and soil moisture. The data is then displayed on Blynk along with suggestions based on the soil moisture levels.

#### 1. Build the Circuit

**Note:** The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.

---



- *Arduino UNO R4 Minima Board*
- *ESP8266 Module*
- *Temperature and Humidity Sensor Module (DHT11)*
- *Capacitive Soil Moisture Module*

## 2. Configure Blynk

**Note:** If you are not familiar with Blynk, it is strongly recommended that you read these two tutorials first. [Get Started with Blynk](#) is a beginner's guide for Blynk, which includes how to configure ESP8266 and register with Blynk. And [Flame Alert System with Blynk](#) is a simple example, but the description of the steps will be more detailed.

### 2.1 Create template

Firstly, we need to establish a template on Blynk. Create a “**Plant Monitor**” template.

### 2.2 Datastream

Create **Datastreams** of type **Virtual Pin** in the **Datastream** page receive data from esp8266 and uno r4 board.

- Create Virtual Pin V0 according to the following diagram:

Set the name of the **Virtual Pin V0** to **temperature**. Set the **DATA TYPE** to **Double** and MIN and MAX to **-100** and **100**. Set the **UNITS** to **Celsius, °C**.

**Virtual Pin Datastream**

NAME: temperature ALIAS: temperature

PIN: V0 DATA TYPE: Double

UNITS: Celsius, °C

MIN: -100 MAX: 100 DECIMALS: ## DEFAULT VALUE: Default Value


+ ADVANCED SETTINGS


Cancel Save



- Create Virtual Pin V1 according to the following diagram:

Set the name of the **Virtual Pin V1** to **humidity**. Set the **DATA TYPE** to **Double** and MIN and MAX to **0** and **100**. Set the **UNITS** to **Percentage, %**.

### Virtual Pin Datastream

NAME	humidity	ALIAS	humidity	
PIN	V1	DATA TYPE	Double	
UNITS	Percentage, %			
MIN	0	MAX	100	DECIMALS
				##
			DEFAULT VALUE	Default Value

 ADVANCED SETTINGS

- Create Virtual Pin V2 according to the following diagram:

Set the name of the **Virtual Pin V2** to **soilMoisture**. Set the **DATA TYPE** to **String**.

### Virtual Pin Datastream


NAME	soilMoisture	ALIAS	soilMoisture
PIN	V2	DATA TYPE	String
DEFAULT VALUE			
hello!			
<input type="checkbox"/> ADVANCED SETTINGS			

Cancel Save

- Create Virtual Pin V3 according to the following diagram:

Set the name of the **Virtual Pin V3** to **LED**. Set the **DATA TYPE** to **Integer** and MIN and MAX to **0** and **255**.

### Virtual Pin Datastream




NAME

LED

ALIAS

LED



PIN

V3

DATA TYPE

Integer

UNITS


None

MIN


0

MAX

255

DEFAULT VALUE 

255

 ADVANCED SETTINGS

Cancel

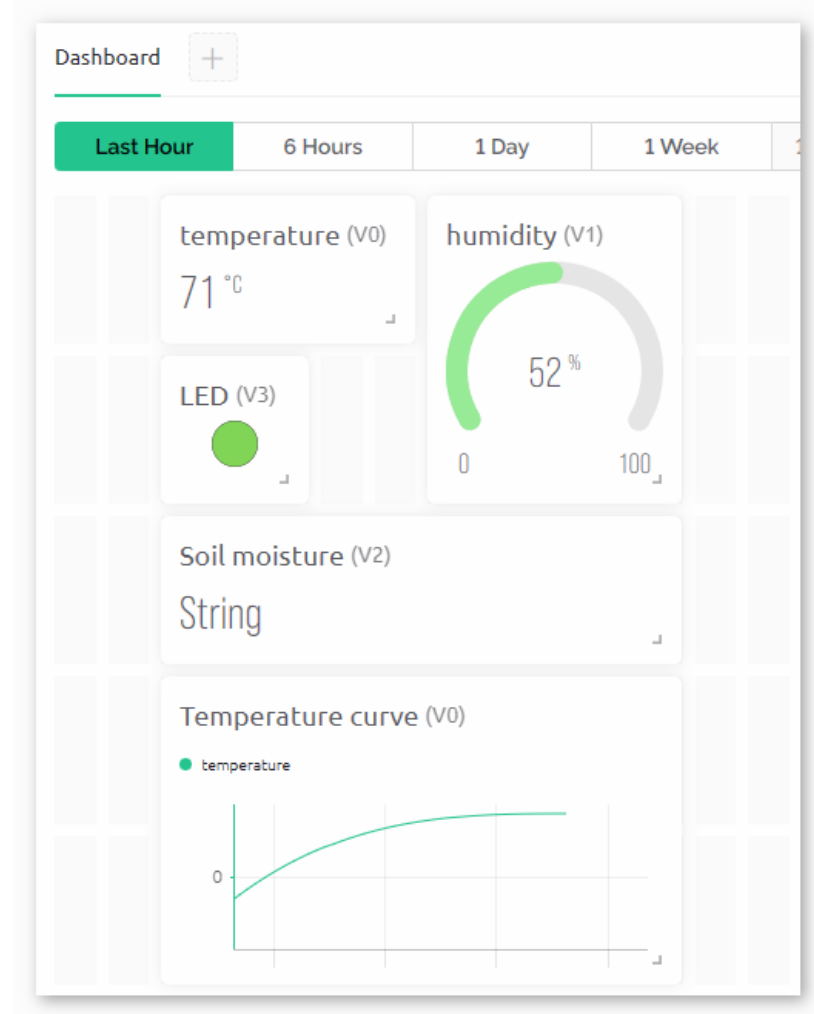
Save

## 2.3 Web Dashboard

We also need to configure the **Web Dashboard** to interact with the Plant monitor.

Configure the Web Dashboard according to the following diagram. We used widgets such as label, gauge, LED, and chart. Be sure to bind each widget to its corresponding virtual pin.





## 2.4 Save template

At last, remember to save the template.

## 3. Run the Code

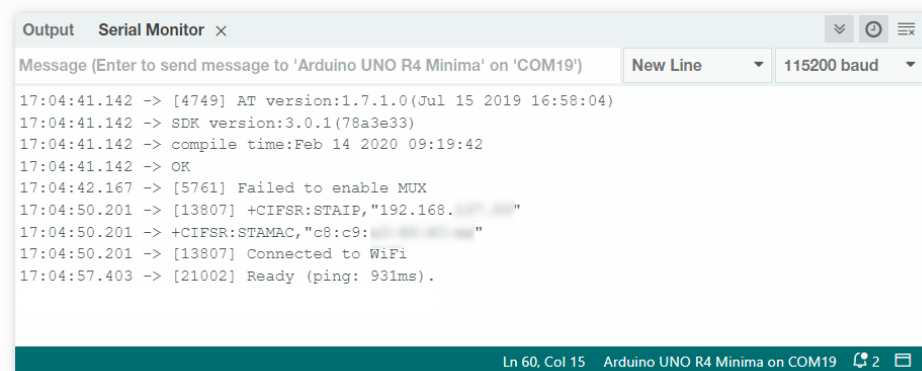
1. Open the 03-Plant\_monitor.ino file under the path of ultimate-sensor-kit\iot\_project\wifi\03-Plant\_monitor, or copy this code into **Arduino IDE**.
2. Create a Blynk device using the “Plant Monitor” template. Then, replace the BLYNK\_TEMPLATE\_ID, BLYNK\_TEMPLATE\_NAME, and BLYNK\_AUTH\_TOKEN with your own.

```
#define BLYNK_TEMPLATE_ID "TMPxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Plant Monitor"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxx"
```

3. You also need to enter the ssid and password of the WiFi you are using.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to 115200) and wait for a prompt such as a successful connection to appear.



**Note:** If the message ESP is not responding appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

## 4. Code explanation

### 1. Initializing Libraries & Defining Constants:

This segment of code includes necessary libraries and defines certain constants like the Blynk template information and WiFi credentials.

```
#define BLYNK_TEMPLATE_ID "TMPLxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Plant Monitor"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxx"
#define BLYNK_PRINT Serial
#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>
char ssid[] = "your_ssid";
char pass[] = "your_password";
#include <SoftwareSerial.h>
SoftwareSerial EspSerial(2, 3);
#define ESP8266_BAUD 115200
ESP8266 wifi(&EspSerial);
```

### 2. Setting up the DHT Sensor:

The DHT sensor is initialized and relevant variables for storing temperature and humidity are defined.

```
#include <DHT.h>
#define DHTPIN 8
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

(continues on next page)

(continued from previous page)

```
float temperature;
float humidity;
```

### 3. Setting up the Soil Moisture Sensor:

Configuration for the soil moisture sensor. Thresholds for wet and dry conditions are defined.

You need to measure your own `wetSoil` and `drySoil` according to your actual situation. Record the reading of Soil Moisture Module as `drySoil` when the soil is dry, and record the reading of Soil Moisture Module within a suitable range that you consider to be the most moist (going beyond this range would be too wet) as `wetSoil`.

```
#define wetSoil 320
#define drySoil 400
const int moistureSensorPin = A0;
int moisture;
String soilStatus;
```

### 4. Setting up the Timer:

A timer is configured which will dictate the frequency of data readings and updates.

```
BlynkTimer timer;
```

### 5. Initialization in Setup Function:

This section sets up the serial communication, configures the ESP8266 for WiFi, and begins the DHT sensor.

- We use `timer.setInterval(5000L, myTimerEvent)` to set the timer interval in `setup()`, here we set to execute the `myTimerEvent()` function every **5000ms**. You can modify the first parameter of `timer.setInterval(1000L, myTimerEvent)` to change the interval between `myTimerEvent` executions.

```
void setup() {
  Serial.begin(115200);
  EspSerial.begin(ESP8266_BAUD);
  delay(10);
  Blynk.config(wifi, BLYNK_AUTH_TOKEN);
  Blynk.connectWifi(ssid, pass);
  timer.setInterval(5000L, myTimerEvent);
  dht.begin();
}
```

### 6. loop() Function:

The main loop runs the Blynk process and timer.

```
void loop() {
  Blynk.run();
  timer.run();
}
```

### 7. sendData() Function:

This function reads values from the DHT sensor and soil moisture sensor, determines the soil's status, and sends data to the Blynk app.

- Use `Blynk.virtualWrite(vPin, value)` to send data to virtual pins on Blynk. Please refer to .
- Use `Blynk.setProperty(V3, "color", color)` to set the color of LED on Blynk. More detail at .

```
void sendData() {  
  // (code for reading and determining values)  
  Blynk.virtualWrite(V0, temperature);  
  Blynk.virtualWrite(V1, humidity);  
  Blynk.virtualWrite(V2, soilStatus);  
  Blynk.virtualWrite(V3, 255);           // set blynk LED brightness  
  Blynk.setProperty(V3, "color", color); // set blynk LED color  
}
```

#### 8. Printing Data to Serial Monitor:

This function is useful for debugging and verifying the readings locally on the Arduino IDE's serial monitor.

```
void printData() {  
  // (code for printing values to serial monitor)  
}
```

#### Reference

- 
- 
- 

### 2.5.5 Remote Relay Controller with Blynk

This project aims to create a remote relay controller that can be operated through a virtual switch in the Blynk app. When the switch is turned on, it sets the digital pin connected to the relay to HIGH, and when it's turned off, it sets the digital pin to LOW. This allows for easy control of the relay from a distance, effectively creating a remote switch.

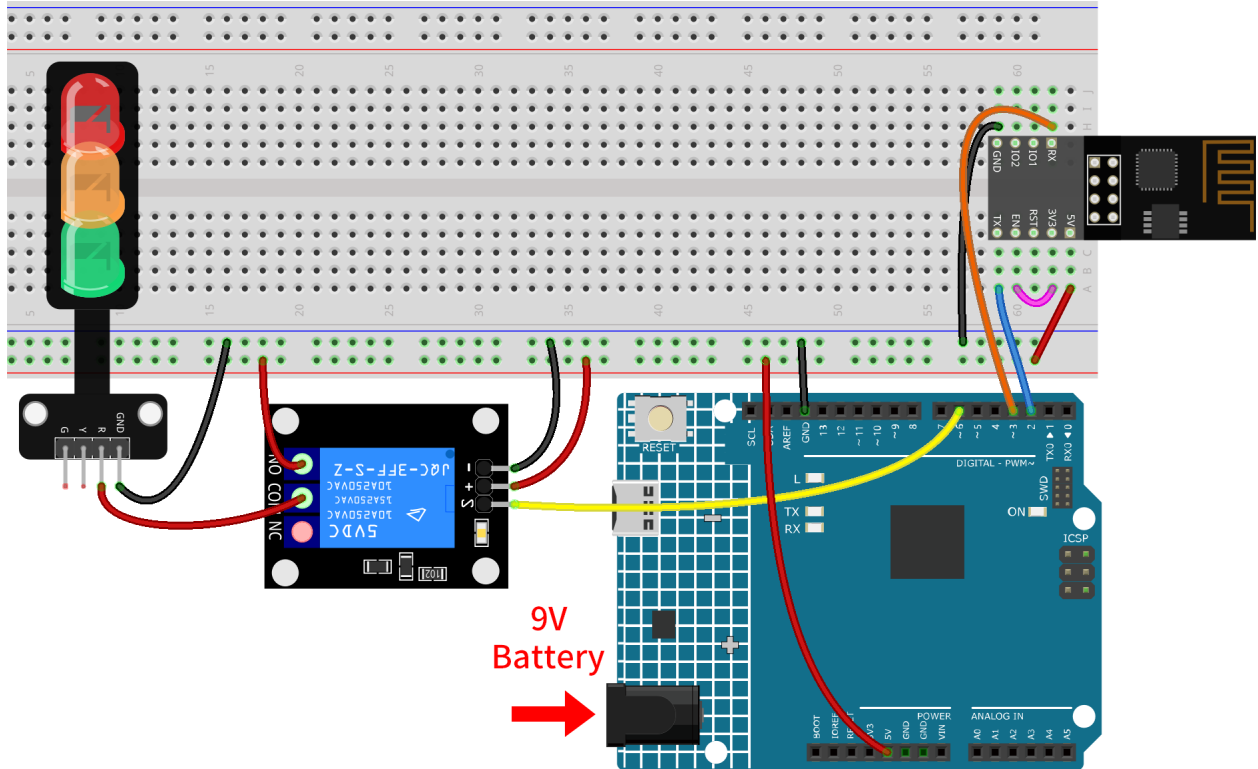
#### 1. Build the Circuit

**Warning:** The following example demonstrates using a relay to control an LED module. **While you can connect the relay to other appliances in actual applications, extreme caution is required when dealing with HIGH AC voltage. Improper or incorrect use can lead to severe injury or even death. Therefore, it is intended for people who are familiar with and knowledgeable about HIGH AC voltage. Always prioritize safety.**

---

**Note:** The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.

---



- *Arduino UNO R4 Minima Board*
- *ESP8266 Module*
- *5V Relay Module*
- *Traffic Light Module*

## 2. Configure Blynk

**Note:** If you are not familiar with Blynk, it is strongly recommended that you read these two tutorials first. [Get Started with Blynk](#) is a beginner's guide for Blynk, which includes how to configure ESP8266 and register with Blynk. And [Flame Alert System with Blynk](#) is a simple example, but the description of the steps will be more detailed.

## 2.1 Create template

Firstly, we need to establish a template on Blynk. Create a “**Remote relay**” template.

## 2.2 Datastream

Create **Datastreams** of type **Virtual Pin** in the **Datastream** page receive data from esp8266 and uno r4 board.

- Create Virtual Pin V0 according to the following diagram:

Set the name of the **Virtual Pin V0** to **Switch status**. Set the **DATA TYPE** to **Integer** and MIN and MAX to **0** and **1**. Set the **UNITS** to **None**.

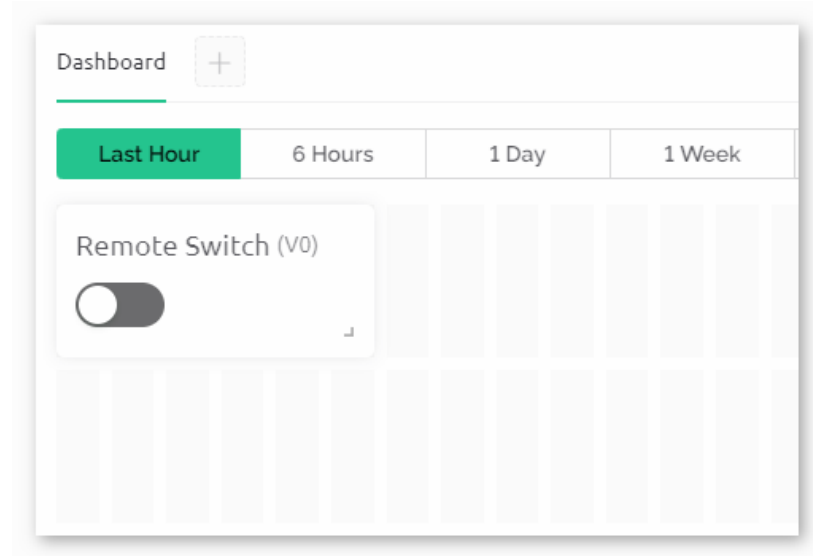
The screenshot shows the 'Virtual Pin Datastream' configuration window. It includes fields for NAME (Switch status), ALIAS (Switch status), PIN (V0), DATA TYPE (Integer), UNITS (None), MIN (0), MAX (1), and DEFAULT VALUE (0). There is also an 'ADVANCED SETTINGS' section and 'Cancel' and 'Save' buttons at the bottom right.

Virtual Pin Datastream			
NAME		ALIAS	
<input type="text" value="Switch status"/>		<input type="text" value="Switch status"/>	
PIN		DATA TYPE	
<input type="text" value="V0"/>		<input type="text" value="Integer"/>	
UNITS			
<input type="text" value="None"/>			
MIN	MAX	DEFAULT VALUE	
<input type="text" value="0"/>	<input type="text" value="1"/>	<input type="text" value="0"/>	
<input type="checkbox"/> ADVANCED SETTINGS			
		<input type="button" value="Cancel"/>	<input type="button" value="Save"/>

## 2.3 Web Dashboard

We also need to configure the **Web Dashboard** to interact with the Remote relay.

Configure the Web Dashboard according to the following diagram. Be sure to bind each widget to its corresponding virtual pin.



### 3. Run the Code

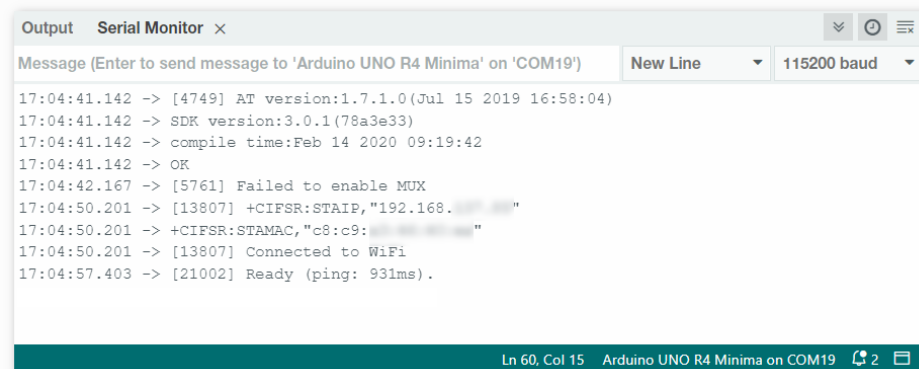
1. Open the 06-Remote\_relay\_controller.ino file under the path of ultimate-sensor-kit\iot\_project\wifi\06-Remote\_relay\_controller, or copy this code into **Arduino IDE**.
2. Create a Blynk device using the “Remote relay” template. Then, replace the BLYNK\_TEMPLATE\_ID, BLYNK\_TEMPLATE\_NAME, and BLYNK\_AUTH\_TOKEN with your own.

```
#define BLYNK_TEMPLATE_ID "TMPxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Remote relay"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxxx"
```

3. You also need to enter the ssid and password of the WiFi you are using.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to 115200) and wait for a prompt such as a successful connection to appear.



**Note:** If the message ESP is not responding appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

---

#### 4. Code explanation

##### 1. Setting up Blynk credentials:

This section contains settings specific to the Blynk app, such as the template ID, device name, and authentication token.

```
#define BLYNK_TEMPLATE_ID "TMPLxxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Remote relay"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxx"
```

##### 2. Include necessary libraries:

We include libraries required for the project, which will allow our Arduino to communicate via WiFi and work with the Blynk app.

```
#define BLYNK_PRINT Serial
#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>
#include <SoftwareSerial.h>
```

##### 3. Configuring WiFi and Serial settings:

The WiFi SSID and password are specified. Additionally, pins for software serial communication with ESP01 are declared. ESP8266\_BAUD defines the baud rate for the ESP8266 module.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
SoftwareSerial EspSerial(2, 3); // RX, TX
#define ESP8266_BAUD 115200
ESP8266 wifi(&EspSerial);
```

##### 4. Relay pin definition:

We define which digital pin of the Arduino will be used to control the relay. We also initialize a variable switchStatus to store the state of our virtual switch in the Blynk app.

```
const int RelayPin = 6;
int switchStatus = 0;
```

##### 5. The setup() function:

In this function, we initialize the relay pin as an output, begin serial communication for debugging, and establish the connection to Blynk using the given WiFi credentials.

```
void setup() {
  pinMode(RelayPin, OUTPUT);
  Serial.begin(115200);
```

(continues on next page)



(continued from previous page)

```

EspSerial.begin(ESP8266_BAUD);
delay(10);
Blynk.config(wifi, BLYNK_AUTH_TOKEN);
Blynk.connectWiFi(ssid, pass);
}

```

#### 6. The loop() function:

It continuously runs two essential functions to keep the connection to Blynk alive and to handle any events (like virtual pin changes).

```

void loop() {
  Blynk.run();
  timer.run();
}

```

#### 7. Handling Blynk's virtual pin:

Here, we read the state of the virtual pin V0 from the Blynk app and control the relay accordingly. If the switch in the app is on (i.e., V0 is 1), the relay pin is set to HIGH, and if it's off, the pin is set to LOW.

- Whenever the value of a virtual pin on the BLYNK server changes, it will trigger BLYNK\_WRITE(). More details at [Blynk API](#).

```

// This function is called every time the Virtual Pin 0 state changes
BLYNK_WRITE(V0) {
  switchStatus = param.asInt(); // Set incoming value from pin V0 to a variable

  if (switchStatus == 1) {
    Serial.println("The switch on Blynk has been turned on.");
    digitalWrite(RelayPin, HIGH);
  } else {
    Serial.println("The switch on Blynk has been turned off.");
    digitalWrite(RelayPin, LOW);
  }
}

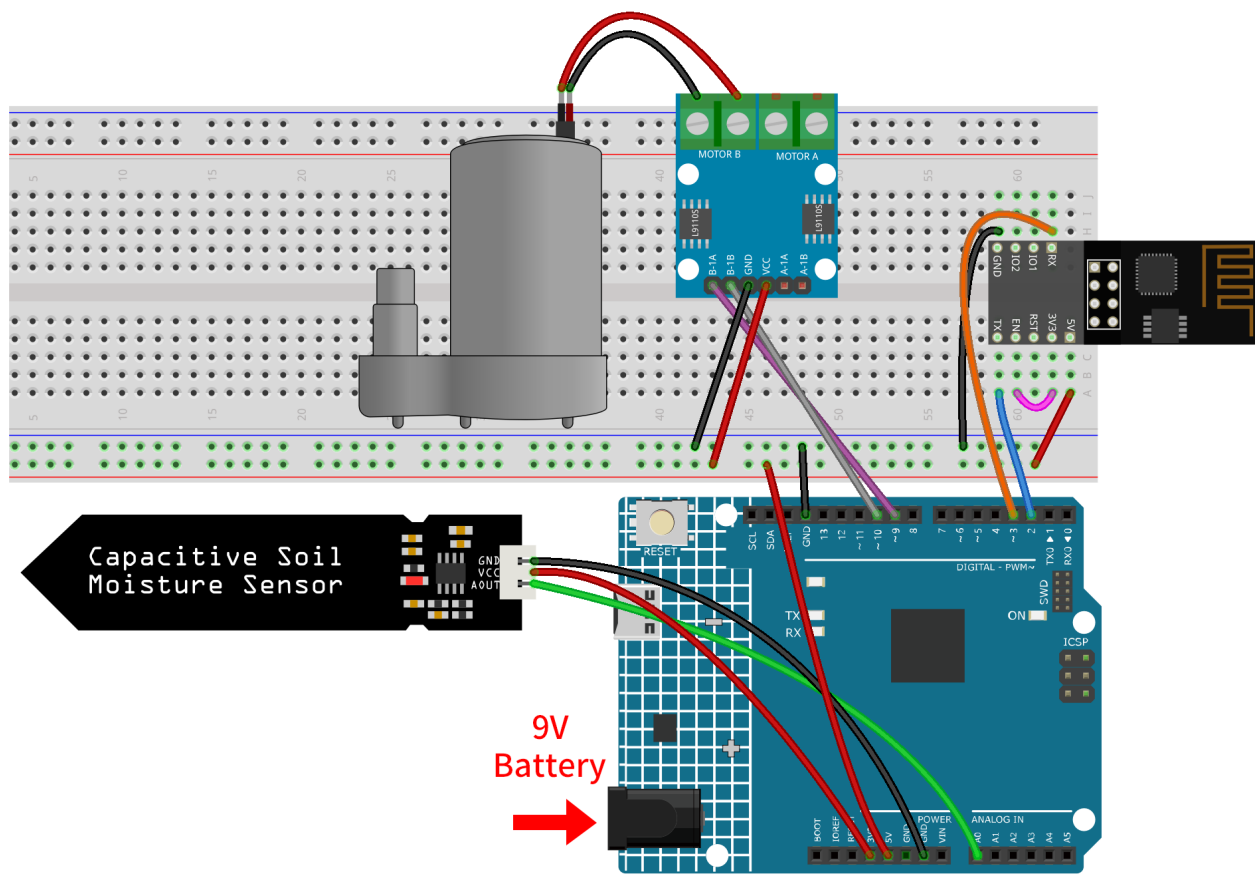
```

## 2.5.6 Auto Watering System with Blynk

This project aims to demonstrate an automated irrigation system that utilizes soil moisture sensors and water pumps. The Blynk app is used for user interaction, where it receives the soil moisture data and sends watering threshold and automatic mode status to the system. When the soil moisture drops below the threshold while in automatic mode, the system activates the water pump.

### 1. Build the Circuit

**Note:** The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.



- *Arduino UNO R4 Minima Board*
- *ESP8266 Module*
- *Capacitive Soil Moisture Module*
- *Centrifugal Pump*

### 2. Configure Blynk

**Note:** If you are not familiar with Blynk, it is strongly recommended that you read these two tutorials first. [Get Started with Blynk](#) is a beginner's guide for Blynk, which includes how to configure ESP8266 and register with Blynk. And [Flame Alert System with Blynk](#) is a simple example, but the description of the steps will be more detailed.

## 2.1 Create template

Firstly, we need to establish a template on Blynk. Create a “**Auto watering system**” template.

## 2.2 Datastream

Create **Datastreams** of type **Virtual Pin** in the **Datastream** page receive data from esp8266 and uno r4 board.

- Create Virtual Pin V0 according to the following diagram:

Set the name of the **Virtual Pin V0** to **Moisture Percentage**. Set the **DATA TYPE** to **Double** and MIN and MAX to **0** and **100**. Set the **UNITS** to **Percentage, %**.

The screenshot shows the 'Virtual Pin Datastream' configuration window. It includes fields for NAME (Moisture Percentage), ALIAS (Moisture Percentage), PIN (V0), DATA TYPE (Double), UNITS (Percentage, %), MIN (0), MAX (100), DECIMALS (###), and a DEFAULT VALUE field (Default Value). There are 'Cancel' and 'Save' buttons at the bottom right, and an 'ADVANCED SETTINGS' toggle at the bottom left.

- Create Virtual Pin V1 according to the following diagram:

Set the name of the **Virtual Pin V1** to **Water Threshold**. Set the **DATA TYPE** to **Double** and MIN and MAX to **0** and **100**. Set the **UNITS** to **Percentage, %**.

### Virtual Pin Datastream

NAME	ALIAS	
<div><div></div>Water Threshold</div>	Water Threshold	<div></div>

PIN	DATA TYPE
V1	Double

UNITS
Percentage, %

MIN	MAX	DECIMALS	DEFAULT VALUE
0	100	###	Default Value

ADVANCED SETTINGS

Cancel

Save


- Create Virtual Pin V2 according to the following diagram:

Set the name of the **Virtual Pin V2** to **Auto Mode**. Set the **DATA TYPE** to **Integer** and MIN and MAX to **0** and **1**.


### Virtual Pin Datastream

NAME

ALIAS

 Auto Mode

Auto Mode



PIN

DATA TYPE

V2


Integer

UNITS

None

MIN


MAX

DEFAULT VALUE 

0

1

0

 ADVANCED SETTINGS

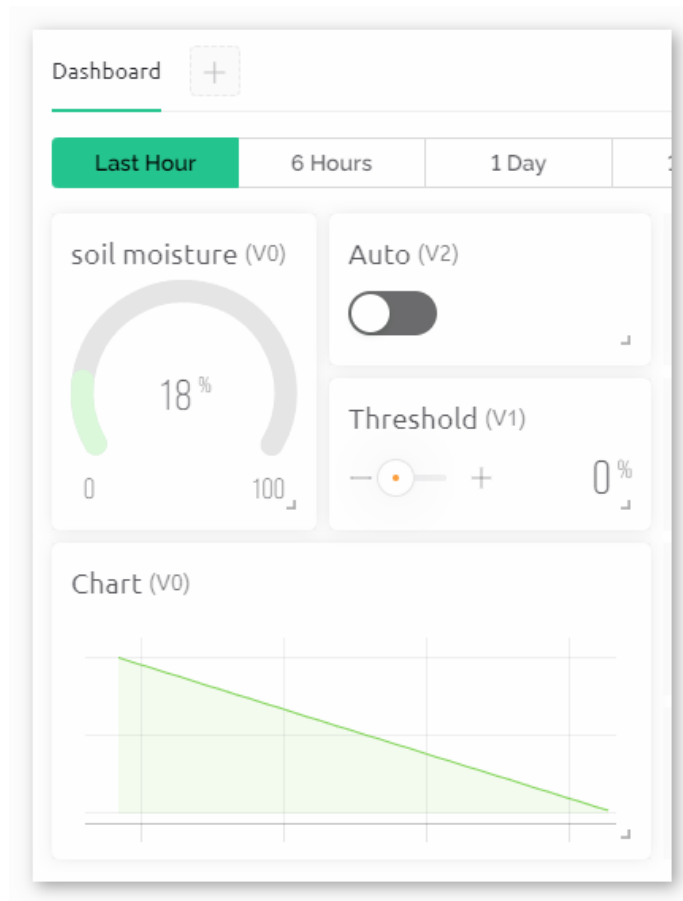
Cancel

Save

## 2.3 Web Dashboard

We also need to configure the **Web Dashboard** to interact with the Auto watering system.

Configure the Web Dashboard according to the following diagram. We used widgets such as label, gauge, switch, slider and chart. Be sure to bind each widget to its corresponding virtual pin.



### 3. Run the Code

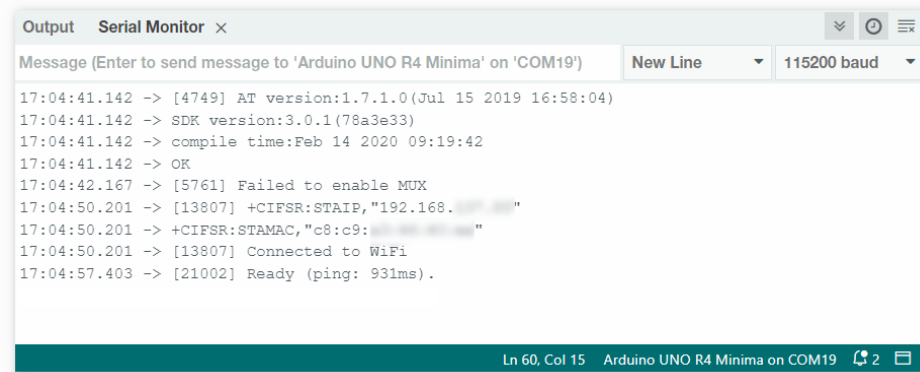
1. Open the 07-Auto\_watering\_system.ino file under the path of ultimate-sensor-kit\iot\_project\wifi\07-Auto\_watering\_system, or copy this code into **Arduino IDE**.
2. Create a Blynk device using the “Auto watering system” template. Then, replace the BLYNK\_TEMPLATE\_ID, BLYNK\_TEMPLATE\_NAME, and BLYNK\_AUTH\_TOKEN with your own.

```
#define BLYNK_TEMPLATE_ID "TMPxxxxxxx"
#define BLYNK_TEMPLATE_NAME "Auto watering system"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxxx"
```

3. You also need to enter the ssid and password of the WiFi you are using.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
```

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to 115200) and wait for a prompt such as a successful connection to appear.



**Note:** If the message ESP is not responding appears when you connect, please follow these steps.

- Make sure the 9V battery is plugged in.
- Reset the ESP8266 module by connecting the pin RST to GND for 1 second, then unplug it.
- Press the reset button on the R4 board.

Sometimes, you may need to repeat the above operation 3-5 times, please be patient.

## 4. Code explanation

### 1. Blynk Cloud Setup and Library Imports

These lines define unique IDs and tokens required to identify and authenticate your Arduino device with the Blynk cloud. Additionally, essential libraries are imported for using the ESP8266 WiFi module, Blynk functionalities, and software serial communication.

```
#define BLYNK_TEMPLATE_ID "TMPLxxxxxx"
#define BLYNK_TEMPLATE_NAME "Auto watering system"
#define BLYNK_AUTH_TOKEN "xxxxxxxxxxxxxxxx"
#define BLYNK_PRINT Serial
#include <ESP8266_Lib.h>
#include <BlynkSimpleShieldEsp8266.h>
#include <SoftwareSerial.h>
```

### 2. WiFi and Hardware Configuration

WiFi credentials (ssid and pass) are defined here. The pins 2 (RX) and 3 (TX) are set for software serial communication between the Arduino and the ESP8266. The baud rate for this communication is defined as 115200.

```
char ssid[] = "your_ssid";
char pass[] = "your_password";
SoftwareSerial EspSerial(2, 3);
#define ESP8266_BAUD 115200
ESP8266 wifi(&EspSerial);
```

### 3. Define pins and global variables

Pins for the water pump control and soil moisture sensor are defined along with global variables to store sensor values, thresholds, modes etc.

You need to measure your own moistureInAir and moistureInWater according to the your actual situation. To avoid the calculated moisturePercentage exceeding the range of 0-100%, adjust the moistureInAir reading upwards to account for sensor fluctuations. Conversely, adjust the moistureInWater reading downwards when recording to compensate for these fluctuations.

```
// Define pin configurations for the water pump
const int pump1A = 9;
const int pump1B = 10;
bool pumpStatus = 0; // 0 indicates OFF, 1 indicates ON

// Define the soil moisture sensor
const float moistureInAir = 535; // Measure by placing the sensor in air
const float moistureInWater = 280; // Measure by immersing the sensor in water
const int sensorPin = A0;
int sensorValue = 0; // Stores the raw sensor value

int autoMode = 0;
int waterThreshold = 0; // The soil moisture percentage threshold to
↳ activate watering
float moisturePercentage = 0; // The calculated soil moisture percentage
```

#### 4. Initial Configurations in the setup() Function

We set two timers:

- We use `timer.setInterval(10000L, updateDataTimer)` to set the timer interval, here we set to execute the `updateDataTimer()` function every **10000ms**. You can modify the first parameter to change the interval between `updateDataTimer()` executions.
- We use `timer.setInterval(35000L, autoWaterTimer)` to set the timer interval, here we set to execute the `autoWaterTimer()` function every **35000ms**. You can modify the first parameter to change the interval between `autoWaterTimer()` executions.

```
void setup() {
  pinMode(pump1A, OUTPUT); // set pump1A as output
  pinMode(pump1B, OUTPUT); // set pump1B as output
  digitalWrite(pump1B, LOW); // Keep pump1B low

  Serial.begin(115200); // Start serial communication at 115200 baud rate
  ↳ for debugging
  EspSerial.begin(ESP8266_BAUD); // Set ESP8266 baud rate
  delay(10);

  // Configure Blynk and connect to WiFi
  Blynk.config(wifi, BLYNK_AUTH_TOKEN);
  Blynk.connectWiFi(ssid, pass);

  // Configure timer events
  timer.setInterval(10000L, updateDataTimer); // Update sensor data every 10
  ↳ seconds
  timer.setInterval(35000L, autoWaterTimer); // Check watering conditions every
  ↳ 35 seconds
}
```

#### 5. loop() Function



This continuously running loop allows the Blynk library to check for updates and handles the defined timer events.

```
void loop() {
  Blynk.run();
  timer.run();
}
```

## 6. Blynk App Interaction

These functions are triggered based on specific interactions with the Blynk app:

- BLYNK\_CONNECTED(): Called when the device connects to Blynk. Syncs the initial states of the virtual pins.
- BLYNK\_WRITE(V1): Triggered when Virtual Pin 1 changes (water threshold).
- BLYNK\_WRITE(V2): Triggered when Virtual Pin 2 changes (auto mode status).

```
// This function is called every time the device is connected to the Blynk.Cloud
BLYNK_CONNECTED() {
  Blynk.syncVirtual(V1); // Sync water threshold
  Blynk.syncVirtual(V2); // Sync auto mode status
}

// This function is called every time the Virtual Pin 1 state changes
BLYNK_WRITE(V1) {
  waterThreshold = param.asInt(); // Update watering threshold
  Serial.print("Received threshold.  waterThreshold:");
  Serial.println(waterThreshold);
}

// This function is called every time the Virtual Pin 2 state changes
BLYNK_WRITE(V2) {
  autoMode = param.asInt(); // Update auto mode status

  if (autoMode == 1) {
    Serial.println("The switch on Blynk has been turned on.");
  } else {
    Serial.println("The switch on Blynk has been turned off.");
  }
}
```

## 7. Timer Callbacks and Automatic Watering Logic

These functions handle the tasks that the timers run:

- updateDataTimer(): Calls sendData() to send current moisture data to Blynk.
- autoWaterTimer(): Calls autoWater() to check if watering is required.
- sendData(): Calculates soil moisture percentage, logs it, and sends it to the Blynk app.
- autoWater(): Checks whether the soil needs watering based on the set threshold and if automatic mode is on.

```
void updateDataTimer() {
  sendData();
}
```

(continues on next page)

(continued from previous page)

```

}

void autoWaterTimer() {
  autoWater();
}

// Function to send sensor data to Blynk app
void sendData() {
  // Calculate soil moisture percentage
  sensorValue = analogRead(sensorPin);
  moisturePercentage = 1 - (sensorValue - moistureInWater) / (moistureInAir -
↵moistureInWater);

  Serial.println("-----");
  Serial.println("Update soil moisture data ...");
  Serial.print("sensorValue:");
  Serial.print(sensorValue);
  Serial.print("  moisturePercentage:");
  Serial.println(moisturePercentage * 100);

  // Send moisture percentage to Blynk app
  Blynk.virtualWrite(V0, moisturePercentage * 100);
}

// Function to control automatic watering based on soil moisture and user settings
void autoWater() {
  if (autoMode == 1 && moisturePercentage * 100 < waterThreshold) {

    if (!pumpStatus) {
      turnOnPump();
      Serial.println("-----");
      Serial.println("Watering...");

      // Turn off pump after 2 seconds
      timer.setTimeout(2000L, turnOffPump);
    }
  }
}

```

## 8. Pump Control Functions

These functions directly control the operation of the water pump:

- `turnOnPump()`: Activates the pump.
- `turnOffPump()`: Deactivates the pump.

```

// Function to turn on the water pump
void turnOnPump() {
  digitalWrite(pump1A, HIGH);
  pumpStatus = 1;
}

// Function to turn off the water pump

```

(continues on next page)

(continued from previous page)

```
void turnOffPump() {  
    digitalWrite(pump1A, LOW);  
    pumpStatus = 0;  
}
```

#### Reference

- 
- 
- 
- 

### 2.5.7 Vibration Alert System with IFTTT

This project sets up a vibration detection system using an Arduino board (Uno R4 or R3) with an ESP8266 module and a vibration sensor (SW-420). When a vibration is detected, the system sends an HTTP request to an IFTTT server, potentially triggering various actions such as sending a notification or an email.

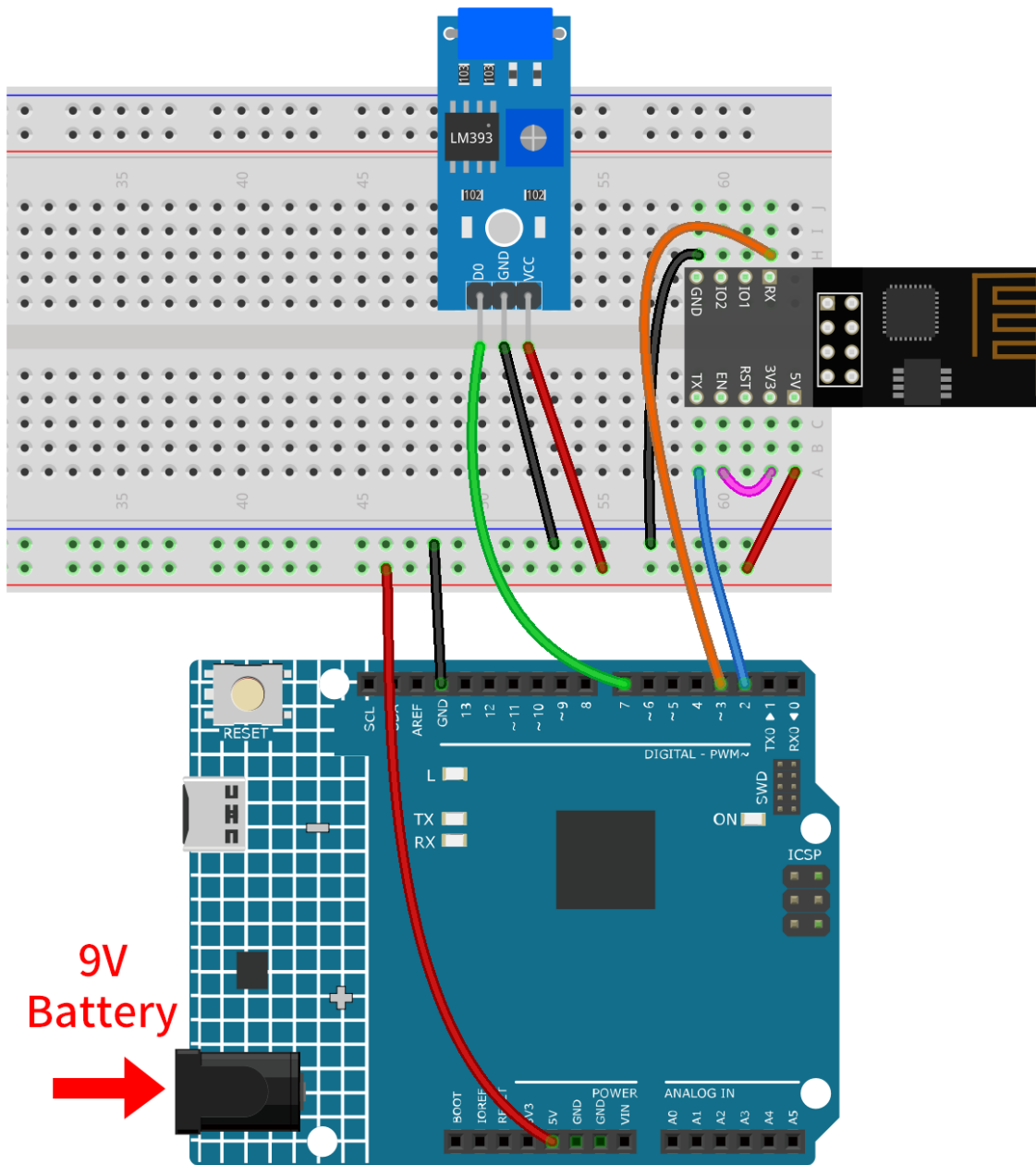
To avoid excessive alerts within a short timeframe, the system has been programmed to send these HTTP requests at a minimum interval of 2 minutes (120000 milliseconds). This interval could be adjusted based on the user's needs.

#### 1. Build the Circuit

---

**Note:** The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.

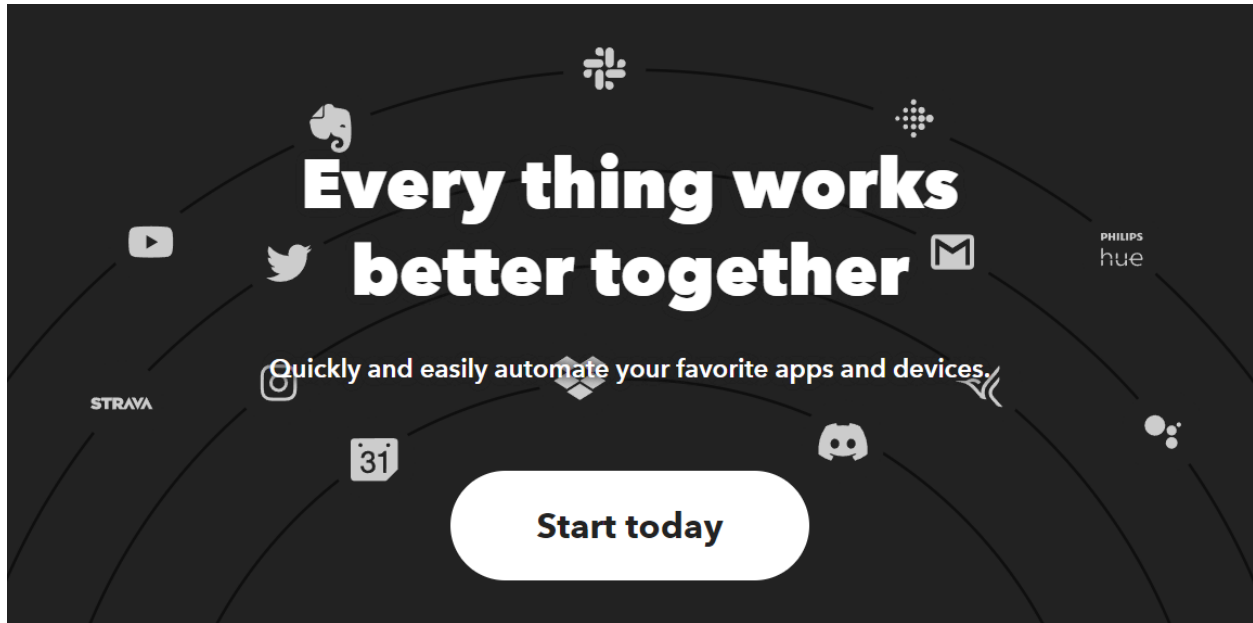
---



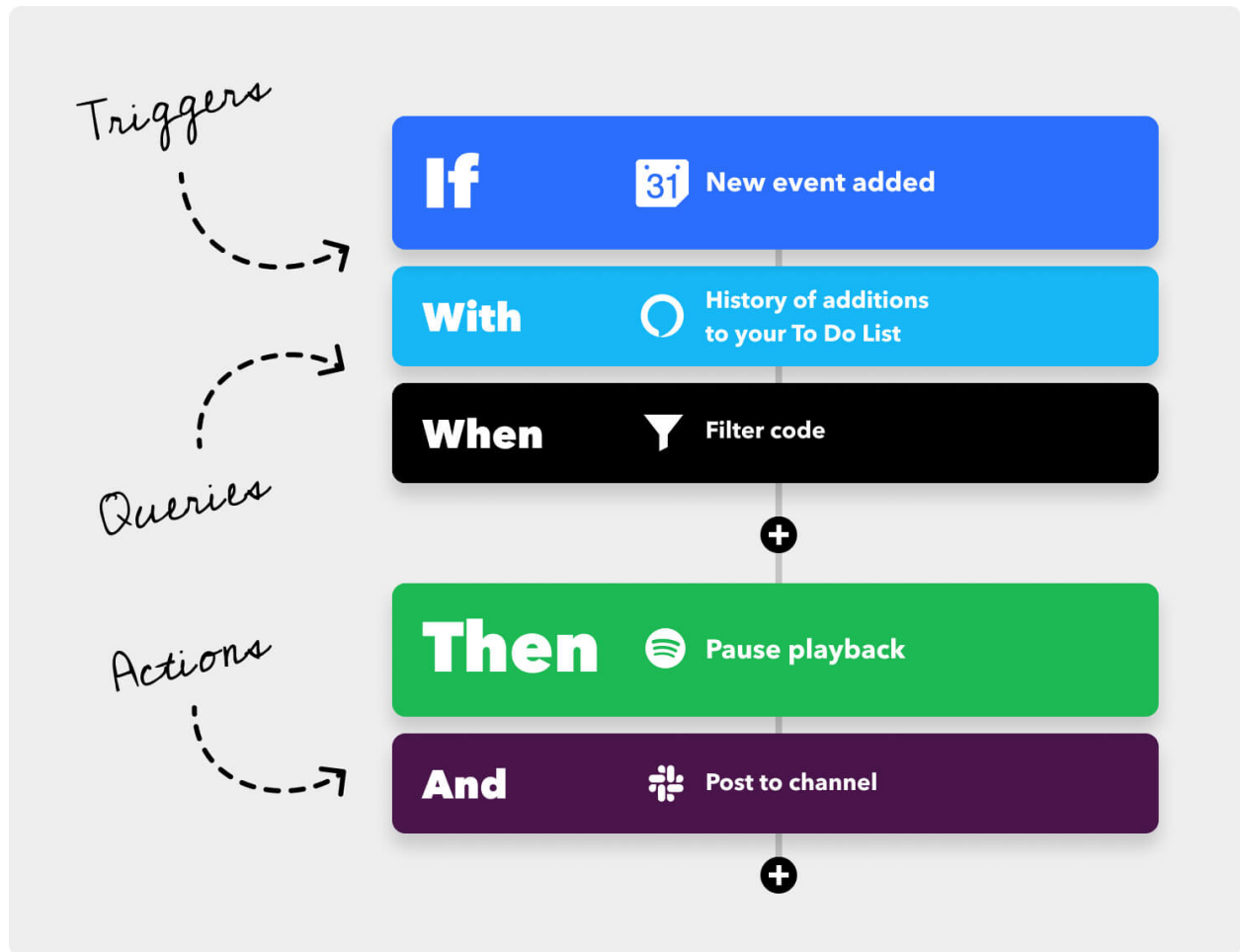
- *Arduino UNO R4 Minima Board*
- *ESP8266 Module*
- *Vibration Sensor Module (SW-420)*

## 2. Configure IFTTT

is a private commercial company founded in 2011 that runs online digital automation platforms which it offers as a service. Their platforms provide a visual interface for making cross-platform if statements to its users, which, as of 2020, numbered 18 million people.



IFTTT stands for “If This Then That.” Basically, if certain conditions are met, then something else will happen. The “if this” part is called a trigger, and the “then that” part is called an action. It joins smart home devices, social media, delivery apps, and more so it can perform automated tasks.



## 2.1 Sign up IFTTT

Type “<https://ifttt.com>” in your browser and click on the “Get started” button located at the center of the page. Fill out the form with your information to create an account.

IFTTT

Explore

Solutions ▼

Plans

Developers

Log in

## Get started



Continue with Apple



Continue with Facebook



Continue with Google

Or use your email to [sign up](#) or [log in](#)

Click “Back” to exit quickstart, return to the IFTTT homepage, refresh the page and log in again.

Back

## Let's start!

What mobile device(s) do you currently use?  
This is important and helps us find the best Applets for you.



Android



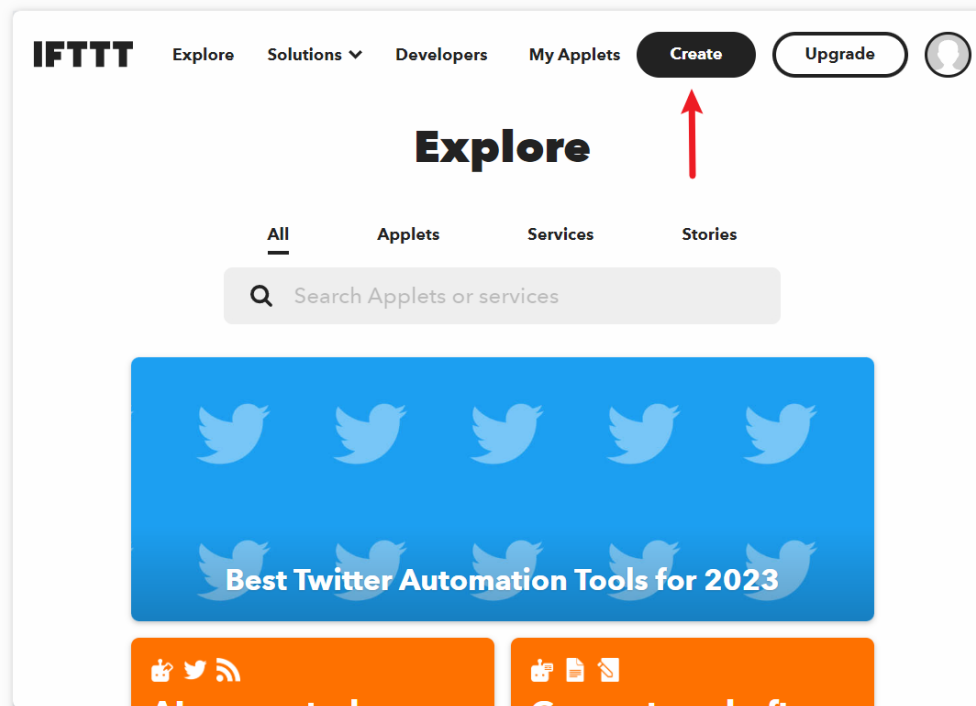
iPhone



Neither

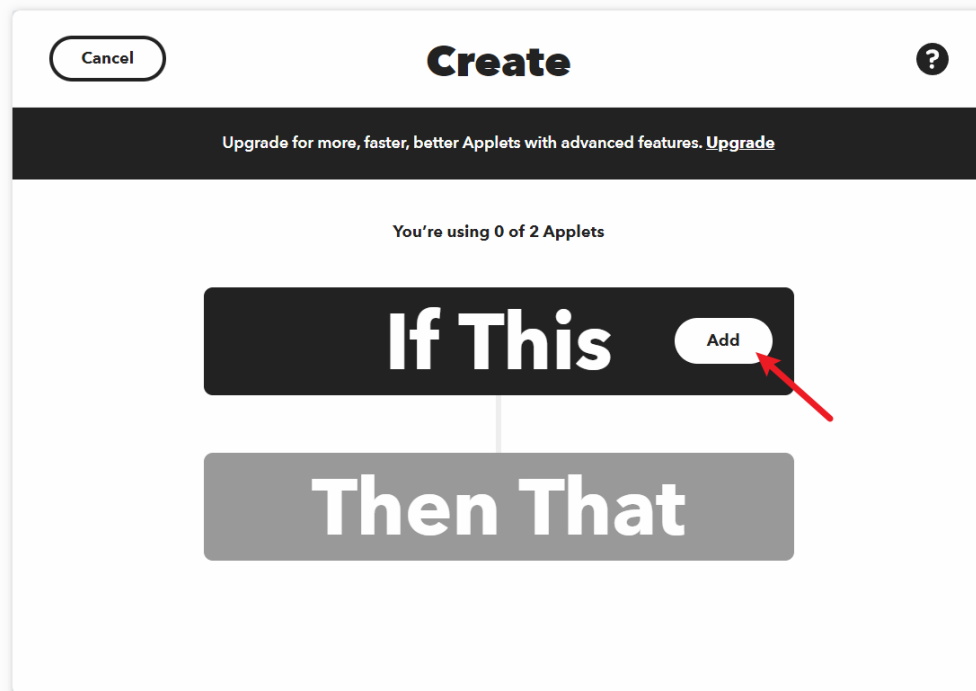
## 2.2 Creating the Applet

Click “Create” to start creating the Applet.



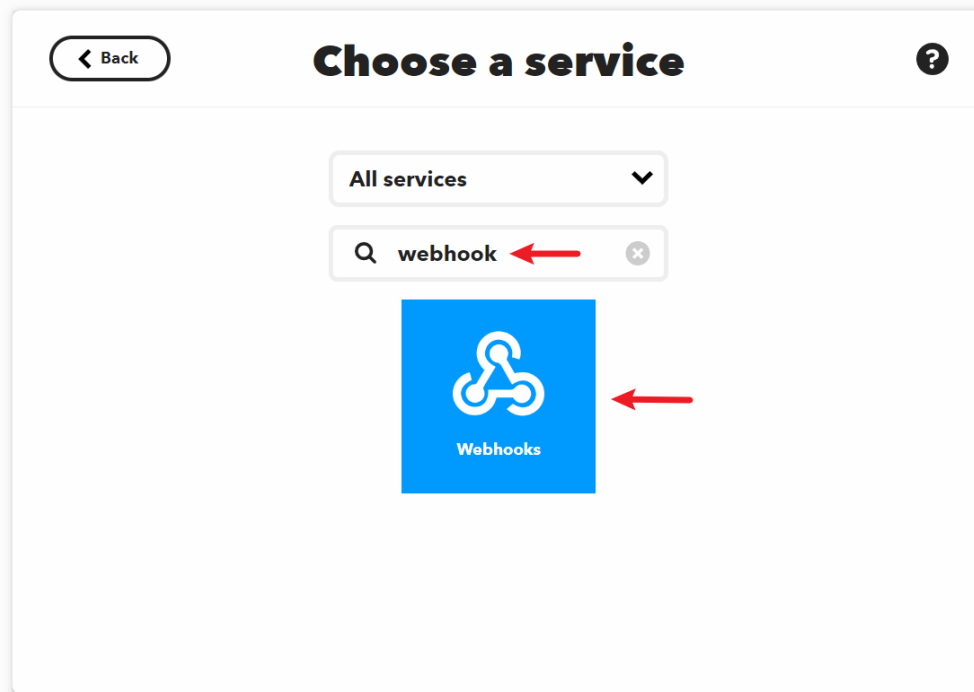
### If This trigger

Click “Add” next to “If This” to add a trigger.

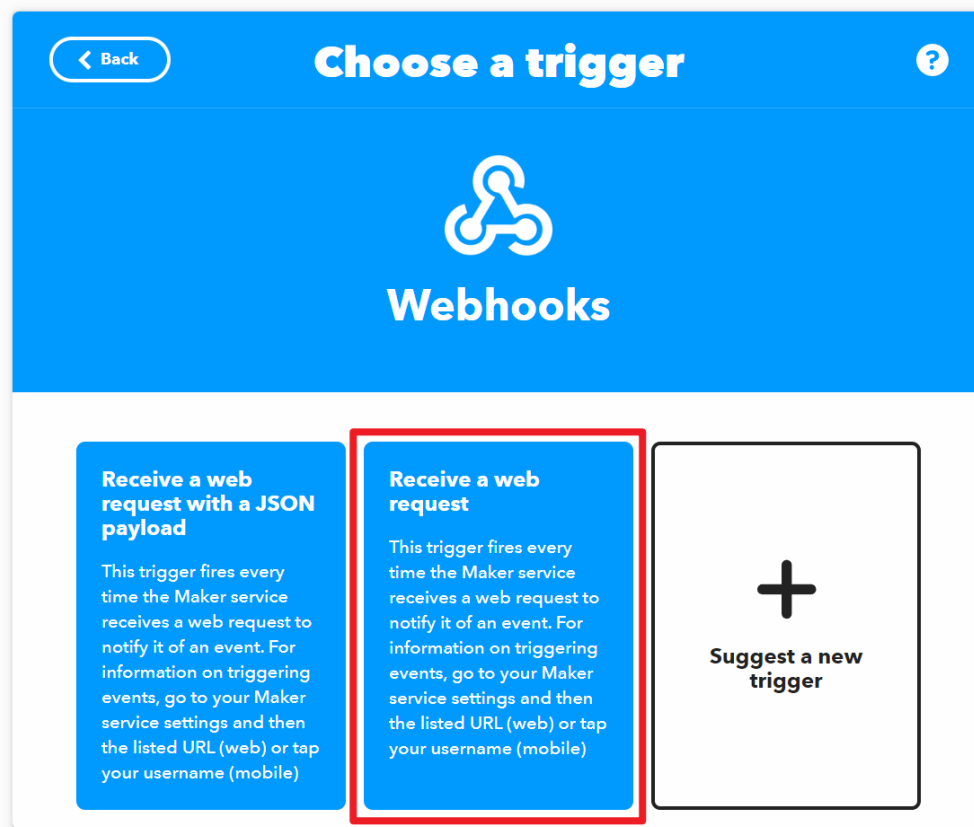




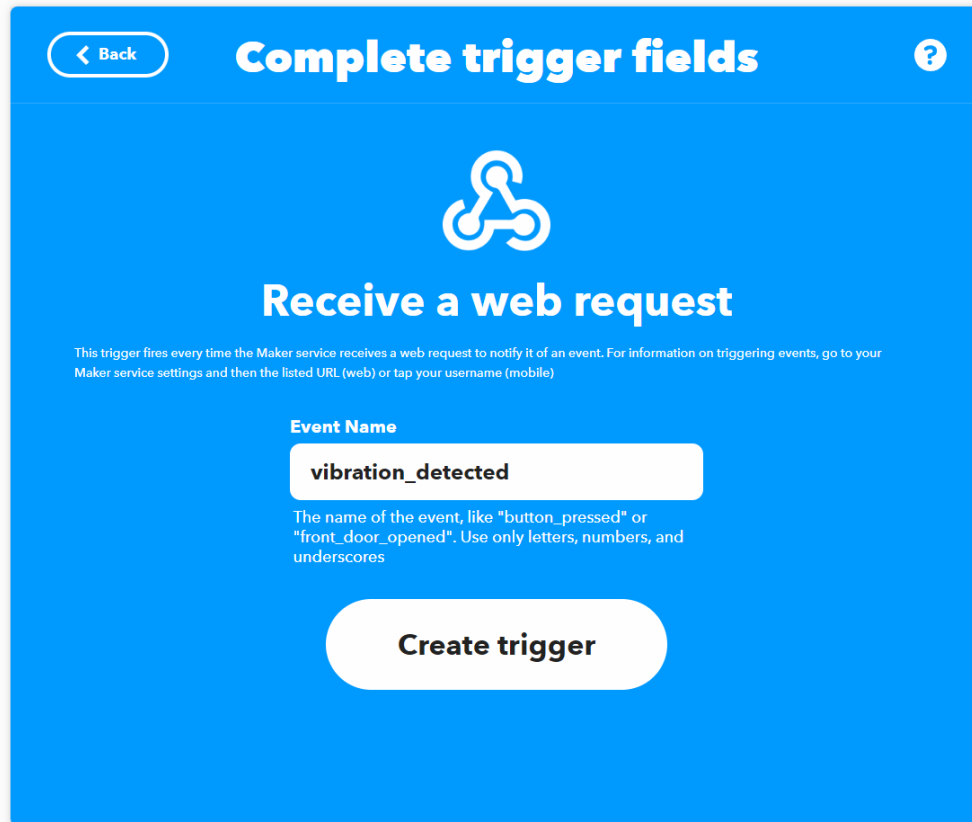
Search for “webhook” and click on “Webhooks”.



Click on “Receive a web request” on the page shown in the following image.

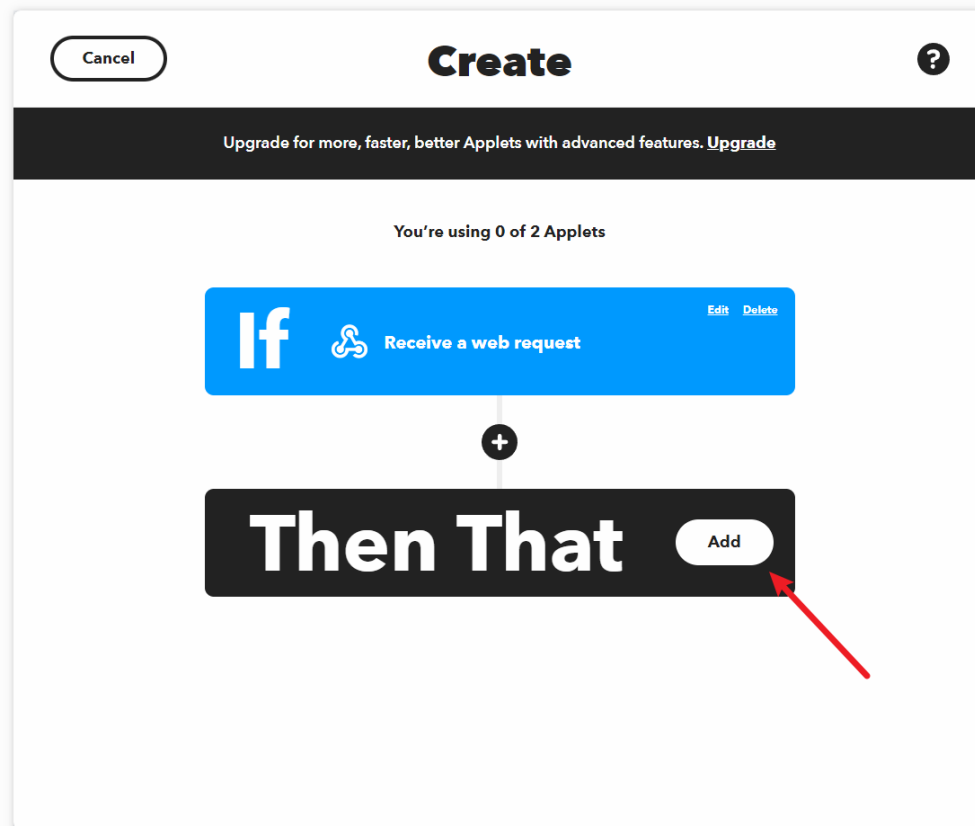


Set the “Event Name” to “vibration\_detected”.

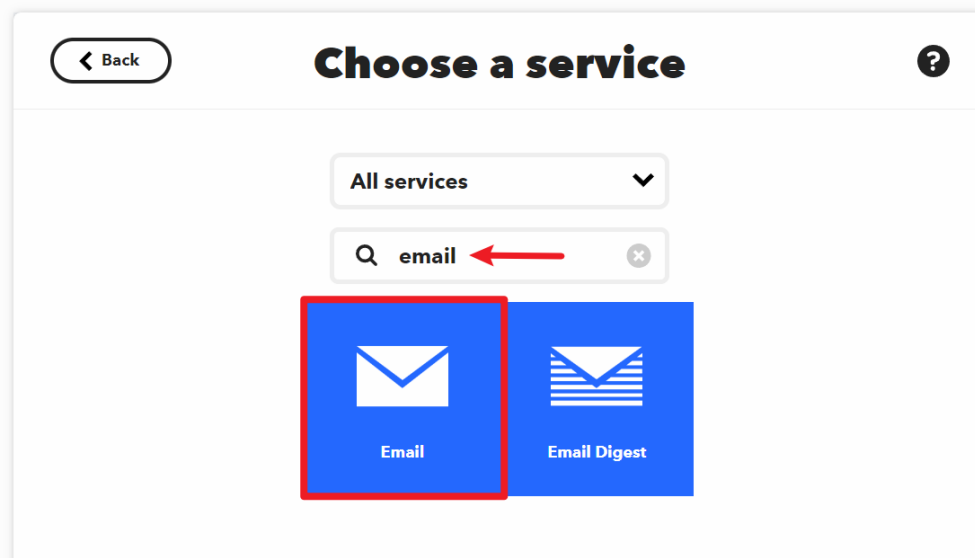


### Then That action

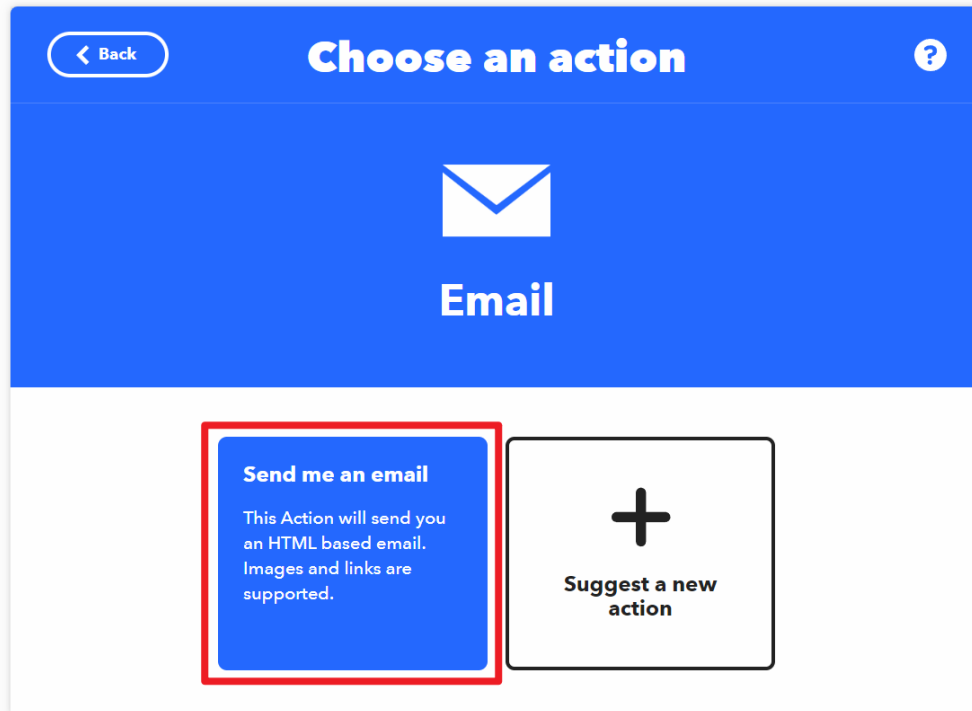
Click on “Add” next to “Then That” to add a action.



Search for “email” and click on “Email”.



Click on “Send me a email” on the page shown in the following image.




Set the subject and content of the email to be sent when vibration is detected.

As a reference, the subject is set to “[ESP-01] Detected vibration!!!”, and the content is set to “Detected vibration, please confirm the situation promptly! {{OccurredAt}}”. When sending an email, {{OccurredAt}} will be automatically replaced with the time when the event occurred.

[< Back](#)

Complete action fields



## Send me an email

This Action will send you an HTML based email. Images and links are supported.

Subject

[ESP-01] Detected vibration!!!

Add ingredient

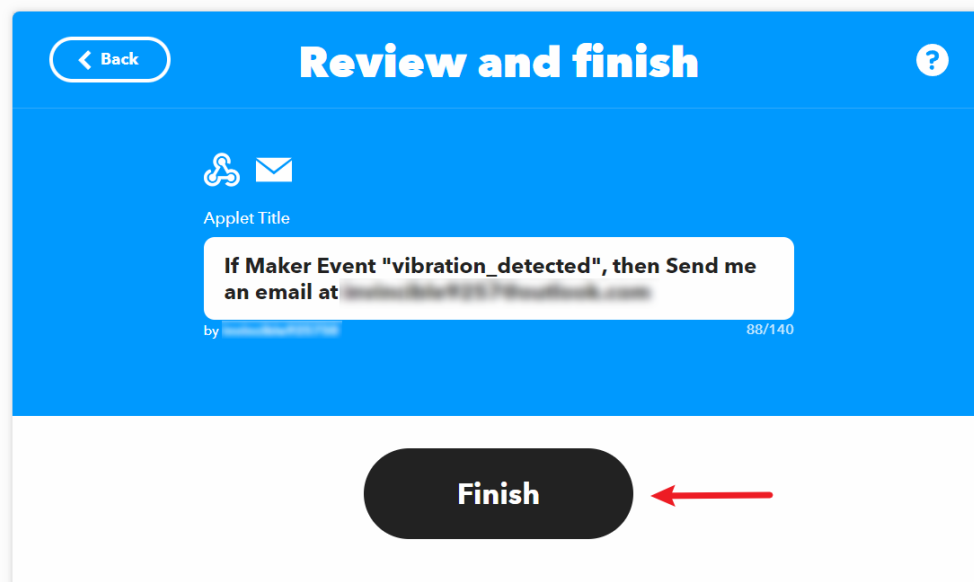
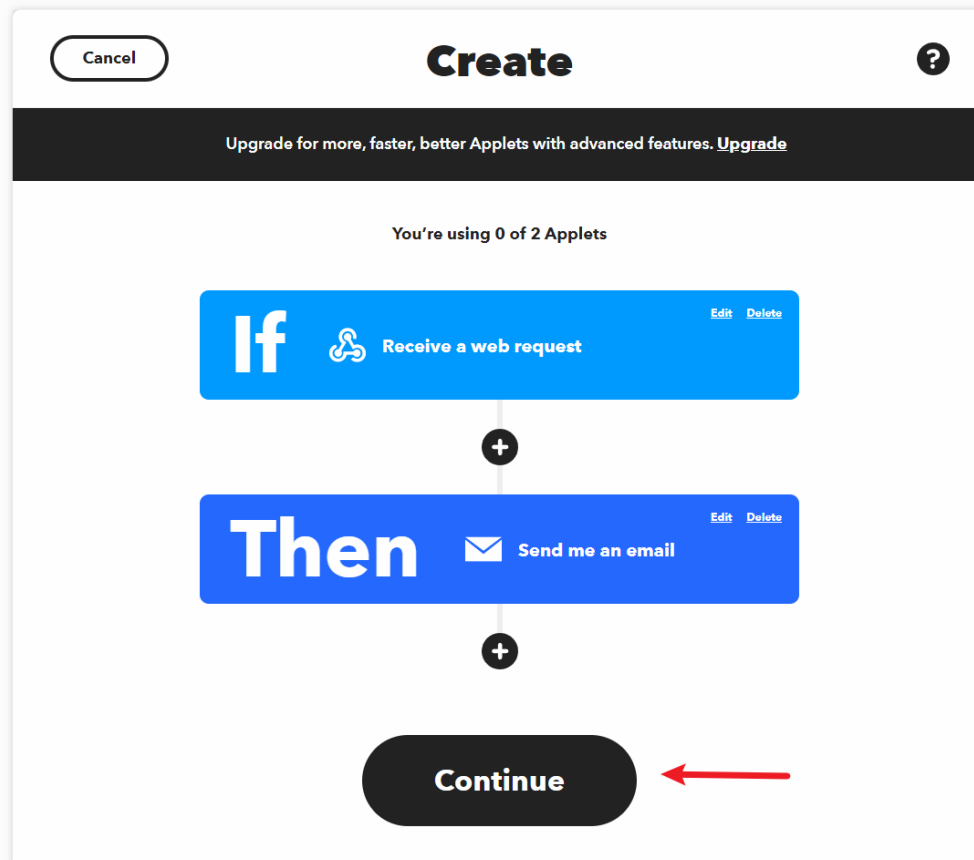
Body

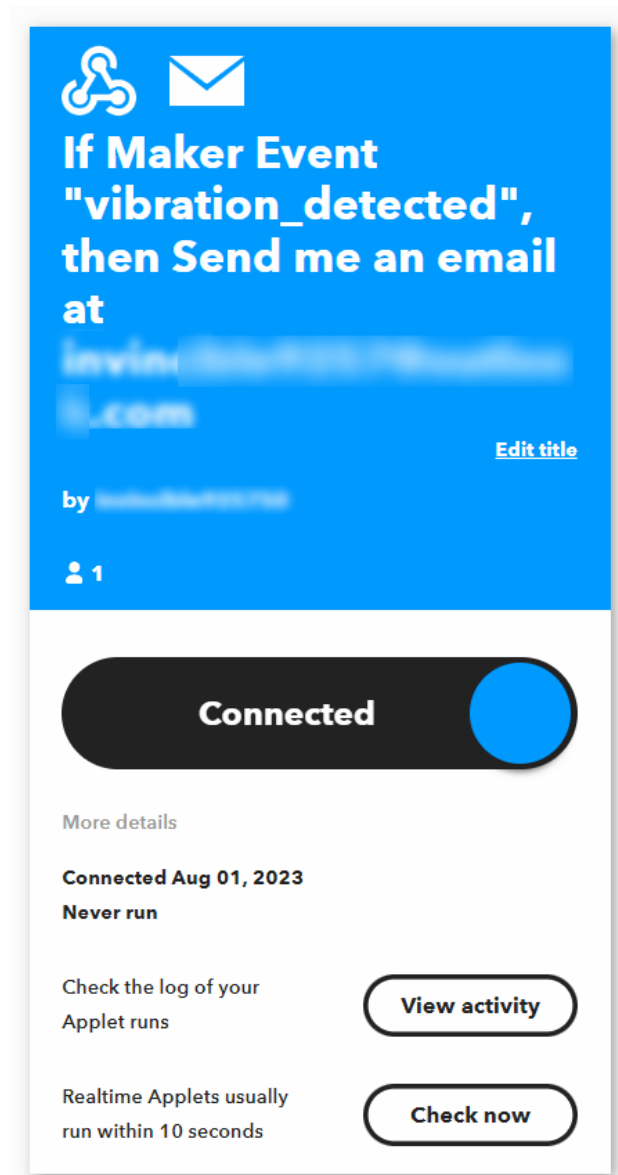
Detected vibration, please confirm the situation promptly!  
{{OccurredAt}}

Add ingredient

Create action

According to the following steps, complete the creation of the Applet.





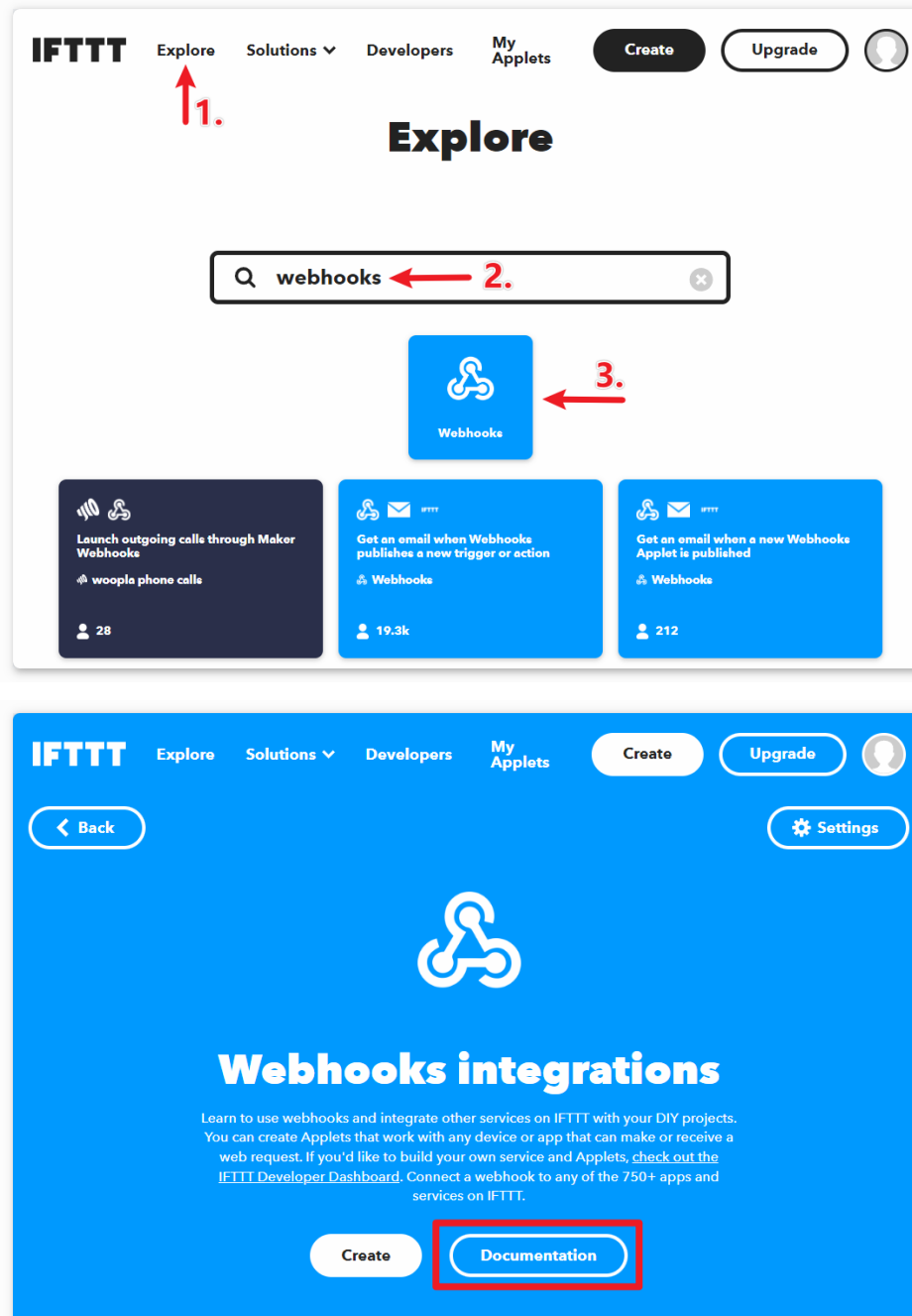
### 3. Run the Code

1. Open the 04-Vibration\_alert\_system.ino file under the path of ultimate-sensor-kit\iot\_project\wifi\04-Vibration\_alert\_system, or copy this code into **Arduino IDE**.
2. You need to enter the mySSID and myPWD of the WiFi you are using.

```
String mySSID = "your_ssid";    // WiFi SSID
String myPWD = "your_password"; // WiFi Password
```

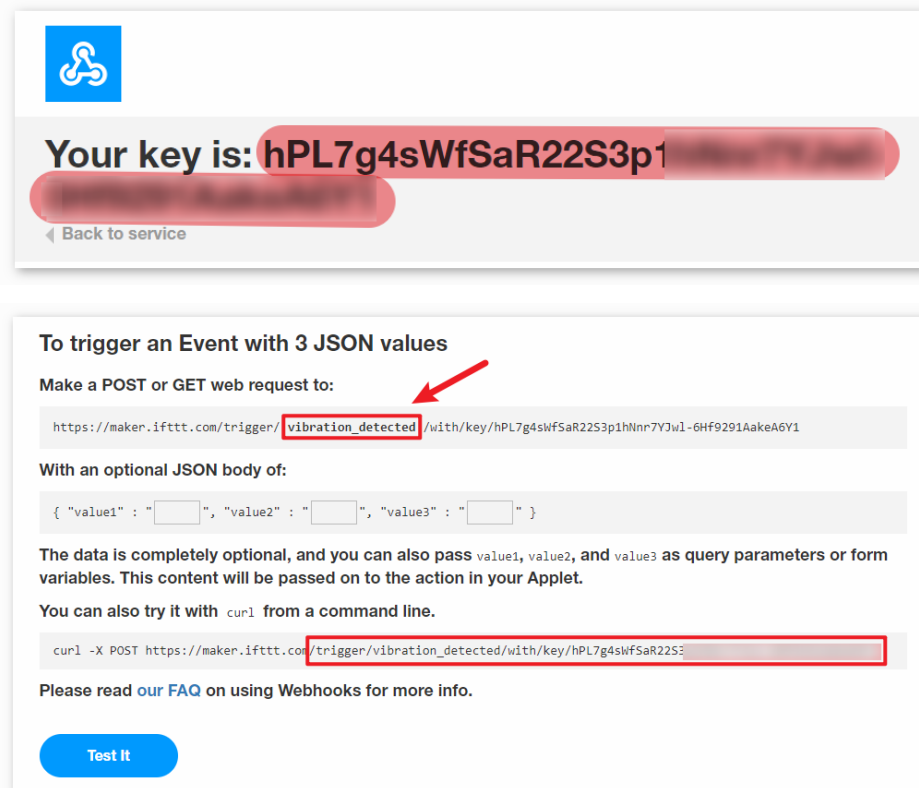
3. You also need to modify the URL with both the event name you set and your API key.


```
String URL = "/trigger/vibration_detected/with/key/xxxxxxxxxxxxxxxxxxxx";
```



Here you can find **your unique API KEY that you must keep private**. Type in the event name as `vibration_detected`. Your final URL will appear at the bottom of the webpage. Copy this URL.







**Your key is:** hPL7g4sWfSaR22S3p1

[Back to service](#)

---

**To trigger an Event with 3 JSON values**

Make a POST or GET web request to:

`https://maker.ifttt.com/trigger/vibration_detected/with/key/hPL7g4sWfSaR22S3p1hNnr7YJwL-6HF9291AakeA6Y1`

With an optional JSON body of:

```
{ "value1" : " ", "value2" : " ", "value3" : " " }
```

The data is completely optional, and you can also pass value1, value2, and value3 as query parameters or form variables. This content will be passed on to the action in your Applet.

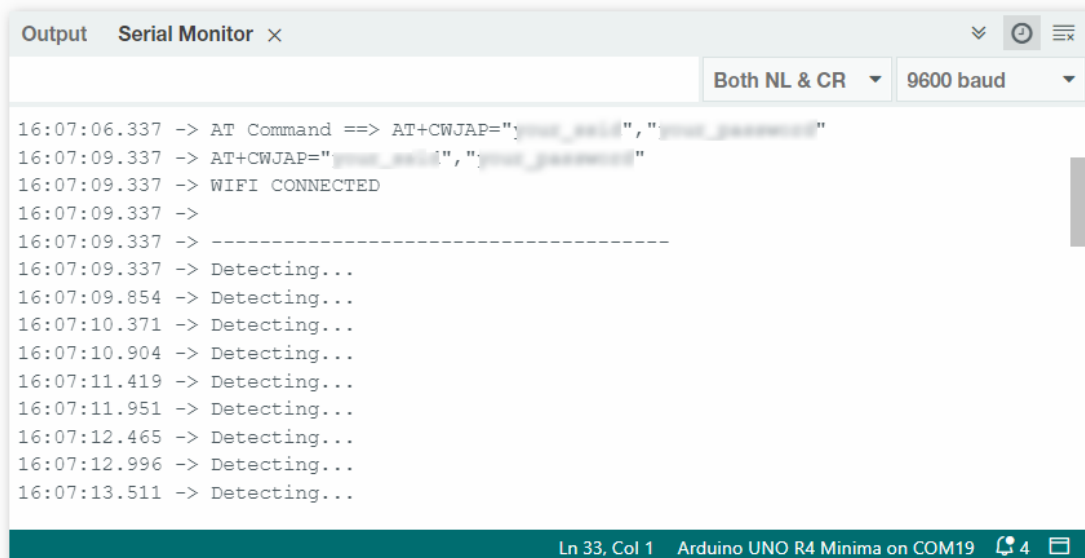
You can also try it with `curl` from a command line.

```
curl -X POST https://maker.ifttt.com/trigger/vibration_detected/with/key/hPL7g4sWfSaR22S3p1hNnr7YJwL-6HF9291AakeA6Y1
```

Please read [our FAQ](#) on using Webhooks for more info.

[Test it](#)

- After selecting the correct board and port, click the **Upload** button.
- Open the Serial monitor(set baudrate to **9600**) and wait for a prompt such as a successful connection to appear.



```

Output  Serial Monitor x
Both NL & CR  9600 baud

16:07:06.337 -> AT Command ==> AT+CWJAP=" ", " "
16:07:09.337 -> AT+CWJAP=" ", " "
16:07:09.337 -> WIFI CONNECTED
16:07:09.337 -> -----
16:07:09.337 -> Detecting...
16:07:09.854 -> Detecting...
16:07:10.371 -> Detecting...
16:07:10.904 -> Detecting...
16:07:11.419 -> Detecting...
16:07:11.951 -> Detecting...
16:07:12.465 -> Detecting...
16:07:12.996 -> Detecting...
16:07:13.511 -> Detecting...

Ln 33, Col 1  Arduino UNO R4 Minima on COM19  4

```

#### 4. Code explanation

The ESP8266 module that comes with the kit is already pre-burned with AT firmware. Therefore, the ESP8266 module can be controlled through AT commands. In this project, we use software serial to enable communication between the Arduino Uno board and the ESP8266 module. The Arduino Uno board sends AT commands to the ESP8266 module for network connection and sending requests. You can refer to .

The Uno board reads sensor values and sends AT commands to the ESP8266 module. The ESP8266 module connects to a network and sends requests to IFTTT servers.

1. Include SoftwareSerial library for serial communication between Arduino and ESP8266

```
#include <SoftwareSerial.h>
SoftwareSerial espSerial(2, 3);
```

2. Configure WiFi credentials and IFTTT server details

```
String mySSID = "your_ssid";
String myPWD = "your_password";
String myHOST = "maker.ifttt.com";
String myPORT = "80";
String URL = "/trigger/xxx/with/key/xxxx";
```

3. Define variables for the vibration sensor and alert frequency control

```
unsigned long lastAlertTime = 0;
const unsigned long postingInterval = 120000L;
const int sensorPin = 7;
```

4. In setup(), initialize serial communication, ESP8266 module and connect to WiFi

```
void setup() {
  Serial.begin(9600);
  espSerial.begin(115200);

  // Initialize the ESP8266 module
  sendATCommand("AT+RST", 1000, DEBUG); //Reset the ESP8266 module
  sendATCommand("AT+CWMODE=1", 1000, DEBUG); //Set the ESP mode as station mode
  sendATCommand("AT+CWJAP=\"" + mySSID + "\",\"" + myPWD + "\"", 3000, DEBUG); //
  ↪ Connect to WiFi network

  while (!espSerial.find("OK")) {
    //Wait for connection
  }
}
```

5. In loop(), detect vibration and send alert if time interval has passed

```
void loop() {

  if (digitalRead(sensorPin)) {
    if (lastAlertTime == 0 || millis() - lastAlertTime > postingInterval) {
      Serial.println("Detected vibration!!!");
      sendAlert(); //Send an HTTP request to IFTTT server
    } else {
```

(continues on next page)

(continued from previous page)

```

        Serial.print("Detected vibration!!! ");
        Serial.println("Since an email has been sent recently, no warning email will_
→be sent this time to avoid bombarding your inbox.");
    }
    } else {
        if (DEBUG) {
            Serial.println("Detecting...");
        }
    }
    delay(500);
}

```

6. sendAlert() constructs HTTP request and sends it via ESP8266

```

void sendAlert() {

    String sendData = "GET " + URL + " HTTP/1.1" + "\r\n";
    sendData += "Host: maker.ifttt.com\r\n";

    sendATCommand("AT+CIPMUX=0", 1000, DEBUG);
    sendATCommand("AT+CIPSTART=...", 3000, DEBUG);
    sendATCommand("AT+CIPSEND=" + String(sendData.length()), 1000, DEBUG);
    espSerial.println(sendData);

}

```

7. Handling AT Commands sendATCommand()

This function sends AT commands to the ESP8266 and collects responses.

```

void sendATCommand(String command, const int timeout, boolean debug) {
    // Print and send command
    Serial.print("AT Command ==> ");
    Serial.print(command);
    Serial.println();
    espSerial.println(command); // Send the AT command

    // Get the response from the ESP8266 module
    String response = "";
    long int time = millis();
    while ((time + timeout) > millis()) { // Wait for the response until the timeout
        while (espSerial.available()) {
            char c = espSerial.read();
            response += c;
        }
    }

    // Print response if debug mode is on
    if (debug) {
        Serial.println(response);
        Serial.println("-----");
    }
}

```

## Reference

- 
- 
- 

### 2.5.8 Weather Monitor with ThingSpeak

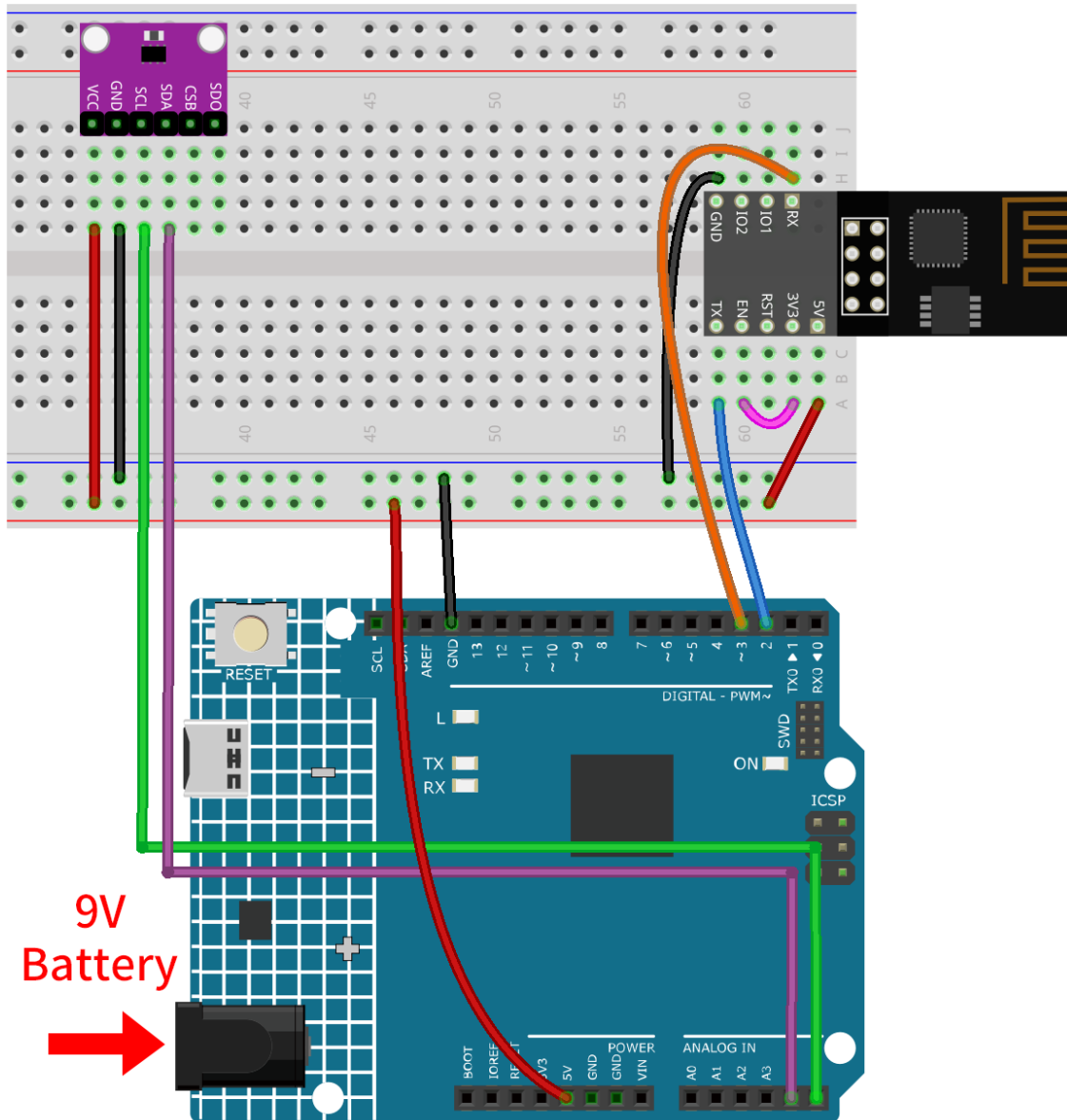
This project collects temperature and pressure data using an Atmospheric Pressure Sensor. The collected data is then transmitted to the ThingSpeak cloud platform via an ESP8266 module and Wi-Fi network at regular time intervals.

#### 1. Build the Circuit

---

**Note:** The ESP8266 module requires a high current to provide a stable operating environment, so make sure the 9V battery is plugged in.

---



- *Arduino UNO R4 Minima Board*
- *ESP8266 Module*
- *Temperature, Humidity & Pressure Sensor (BMP280)*

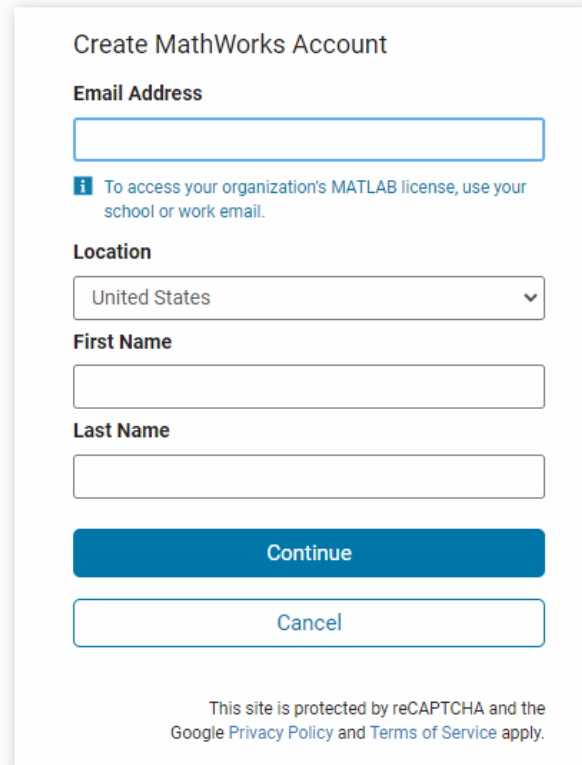
## 2. Configure ThingSpeak

™ is an IoT analytics platform service that allows you to aggregate, visualize and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by your devices to ThingSpeak. With the ability to execute MATLAB® code in ThingSpeak you can perform online analysis and processing of the data as it comes in. ThingSpeak is often used for prototyping and proof of concept IoT systems that require analytics.

## 2.1 Creating ThingSpeak Account

The first thing you need to do is to create an account with ThingSpeak. Since the collaboration with MATLAB, you can use your MathWorks credentials to login to .

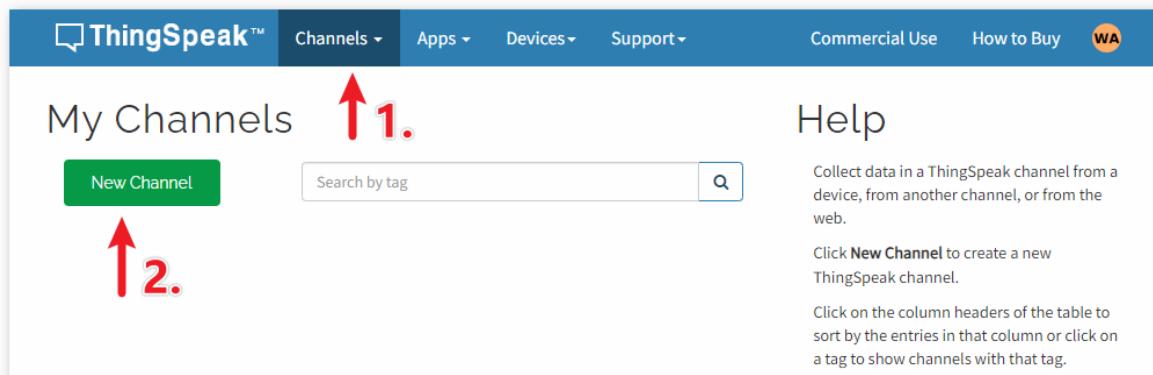
If you do not have one, you need to create an account with MathWorks and login to ThingSpeak Application.



The screenshot shows the 'Create MathWorks Account' form. It includes fields for 'Email Address', 'Location' (a dropdown menu currently showing 'United States'), 'First Name', and 'Last Name'. Below these fields are two buttons: 'Continue' and 'Cancel'. A small information icon (i) is next to the email field with the text: 'To access your organization's MATLAB license, use your school or work email.' At the bottom of the form, it states: 'This site is protected by reCAPTCHA and the Google Privacy Policy and Terms of Service apply.'

## 2.2 Creating the channel

After logging in, create a new channel to store the data by going to “Channels” > “My Channels” and clicking on “New Channel”.



For this project, we need to create a channel called “**Weather Monitor**” with two fields: **Field 1** for “**Temperature**” and **Field 2** for “**Atmospheric Pressure**”.

**ThingSpeak™** Channels Apps Devices Support Commercial Use How to Buy WA

## New Channel

**Name**

**Description**

**Field 1**  ☒

**Field 2**  ☒

**Field 3**  ☐

**Field 4**  ☐

## Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

### Channel Settings

- Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- Channel Name:** Enter a unique name for the ThingSpeak channel.
- Description:** Enter a description of the ThingSpeak channel.
- Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.

### 3. Run the Code

1. Open the 05-Weather\_monitor.ino file under the path of ultimate-sensor-kit\iot\_project\wifi\05-Weather\_monitor, or copy this code into **Arduino IDE**.

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit BMP280**” and install it.

2. You need to enter the mySSID and myPWD of the WiFi you are using.

```
String mySSID = "your_ssid";    // WiFi SSID
String myPWD = "your_password"; // WiFi Password
```

3. You also need to modify the myAPI with your ThingSpeak Channel API key.

```
String myAPI = "xxxxxxxxxxxx"; // API Key
```

## Weather monitor

Channel ID: 22  
Author: mwa0  
Access: Private

Private View Public View Channel Settings Sharing **API Keys** Data Import / Export

### Write API Key

Key

[Generate New Write API Key](#)

### Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

### API Keys Settings

- Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click [Generate New Write API Key](#).

Here you can find **your unique API KEY** that you must keep private.

4. After selecting the correct board and port, click the **Upload** button.
5. Open the Serial monitor(set baudrate to **9600**) and wait for a prompt such as a successful connection to appear.



The screenshot shows the Serial Monitor window in the Arduino IDE. The title bar reads "Output Serial Monitor x". The input field contains the text "Message (Enter to send message to 'Arduino UNO R4 Minima' on". The dropdown menu for line endings is set to "Both NL & CR", and the baud rate is set to "9600 baud". The output area displays the following text:

```
17:57:29.926 -> AT Command ==> AT+CWJAP="ssid","password"
17:57:30.935 -> AT+CWJAP="ssid","password"
17:57:30.935 ->
17:57:30.935 -> -----
17:57:47.919 -> AT Command ==> AT+CWMUX=1
17:57:48.930 -> WIFI CONNECTED
17:57:48.930 -> WIFI GOT IP
17:57:48.930 ->
17:57:48.930 -> OK
17:57:48.930 -> AT+CWMUX=1
17:57:48.930 ->
17:57:48.930 -> OK
17:57:48.930 ->
17:57:48.930 -> -----
17:57:48.930 -> AT Command ==> AT+CIPSTART=0,"TCP","api.thingspeak.com",80
17:57:49.938 -> AT+CIPSTART=0,"TCP","api.thingspeak.com",80
17:57:49.938 -> 0,CONNECT
17:57:49.938 ->
17:57:49.938 -> OK
17:57:49.938 ->
17:57:49.938 -> -----
17:57:49.938 -> AT Command ==> AT+CIPSEND=0,69
17:57:50.946 -> AT+CIPSEND=0,69
17:57:50.946 ->
17:57:50.946 -> OK
17:57:50.946 -> >
```

The status bar at the bottom indicates "Ln 92, Col 9 Arduino UNO R4 Minima on COM17" with 2 notifications.



```

Output Serial Monitor X
Message (Enter to send message to 'Arduino UNO R4 Minima' on Both NL & CR 9600 baud
17:57:50.946 -> -----
17:57:51.954 -> GET /update?api_key=R4PU2F...&field1=32.23&field2=99483.94
17:57:51.954 -> Value to be sent:
17:57:51.954 -> Temperature: 32.23 °C. Pressure: 99483.94 hPa
17:57:51.954 ->
17:57:51.954 -> AT Command ==> AT+CIPCLOSE=0
17:57:52.946 -> +CIPCLOSE=0
17:57:52.946 ->
17:57:52.946 -> busy s...
17:57:52.946 ->
17:57:52.946 -> Recv 69 bytes
17:57:52.946 ->
17:57:52.946 -> SEND OK
17:57:52.946 ->
17:57:52.946 -> +IPD,0,1:4
17:57:52.946 -> -----
Ln 92, Col 9 Arduino UNO R4 Minima on COM17 2

```

#### 4. Code explanation

The ESP8266 module that comes with the kit is already pre-burned with AT firmware. Therefore, the ESP8266 module can be controlled through AT commands. In this project, we use software serial to enable communication between the Arduino Uno board and the ESP8266 module. The Arduino Uno board sends AT commands to the ESP8266 module for network connection and sending requests. You can refer to .

The Uno board reads sensor values and sends AT commands to the ESP8266 module. The ESP8266 module connects to a network and sends requests to ThingSpeak servers.

##### 1. Setting Up & Global Variables:

This section establishes communication with the ESP8266 module and declares necessary global variables.

```

#include <SoftwareSerial.h>
SoftwareSerial espSerial(2, 3);
#define DEBUG true
String mySSID = "your_ssid";
String myPWD = "your_password";
String myAPI = "xxxxxxxxxxxx";
String myHOST = "api.thingspeak.com";
String myPORT = "80";
unsigned long lastConnectionTime = 0;
const unsigned long postingInterval = 20000L;

```

##### 2. BMP280 Sensor Setup:

This code segment sets up the BMP280 sensor for data reading.

```

#include <Wire.h>
#include <Adafruit_BMP280.h>
#define BMP280_ADDRESS 0x76
Adafruit_BMP280 bmp;

```

(continues on next page)

(continued from previous page)

```

unsigned bmpStatus;
float pressure;
float temperature;

```

### 3. Initialization (Setup Function):

The setup() function initializes serial communication, connects the ESP8266 module to Wi-Fi, and initializes the BMP280 sensor.

```

void setup() {
  Serial.begin(9600);
  espSerial.begin(115200);

  // Initialize the ESP8266 module
  sendATCommand("AT+RST", 1000, DEBUG); //
  ↪Reset the ESP8266 module
  sendATCommand("AT+CWMODE=1", 1000, DEBUG); //
  ↪Set the ESP mode as station mode
  sendATCommand("AT+CWJAP=\"" + mySSID + "\",\"" + myPWD + "\"", 1000, DEBUG); //
  ↪Connect to WiFi network

  // Initialize the bmp280 sensor
  bmpStatus = bmp.begin(BMP280_ADDRESS);
  if (!bmpStatus) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring or "
                    "try a different address!"));
    while (1) delay(10); // Stop code execution if the sensor is not found.
  }

  /* Default settings from datasheet. */
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL, /* Operating Mode. */
                 Adafruit_BMP280::SAMPLING_X2, /* Temp. oversampling */
                 Adafruit_BMP280::SAMPLING_X16, /* Pressure oversampling */
                 Adafruit_BMP280::FILTER_X16, /* Filtering. */
                 Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */
}

```

### 4. loop() function:

The main loop checks if 20 seconds have passed since the last data transmission. If so, it sends the data. You can modify the value of the postingInterval variable to adjust the interval at which data is sent.

```

void loop() {
  //Send data according to the time interval you set.
  if (millis() - lastConnectionTime > postingInterval) {
    sendData();
  }
}

```

### 5. Data Transmission:

This function reads the temperature and pressure, constructs the GET request, and sends data to ThingSpeak.

We constructed a GET request in the form of GET /update?api\_key=xxxxxx&field1=xx&field2=xxxxxx and sent three parameters to the ThingSpeak server.

- api\_key: API key for authentication and permission control
- field1: a parameter named “field1” used to record temperature
- field2: a parameter named “field2” used to record atmospheric pressure

```
void sendData() {
    // Read the temperature and pressure from the BMP280 sensor
    pressure = bmp.readPressure();
    temperature = bmp.readTemperature();

    // If the data is invalid, print an error message and stop sending it
    if (isnan(pressure) || isnan(temperature)) {
        Serial.println("Failed to read from BMP sensor!");
        return;
    }

    // Construct the GET request for ThingSpeak
    String sendData = "GET /update?api_key=" + myAPI;
    sendData += "&field1=" + String(temperature);
    sendData += "&field2=" + String(pressure);

    // Send the GET request to ThingSpeak via the ESP8266
    sendATCommand("AT+CIPMUX=1", 1000, DEBUG); //Allow multiple connections
    sendATCommand("AT+CIPSTART=0,\"TCP\", \"" + myHOST + "\",\" + myPORT, 1000, DEBUG);
    → // Start a TCP connection to ThingSpeak
    sendATCommand("AT+CIPSEND=0,\" + String(sendData.length() + 4), 1000, DEBUG);
    → // Send the GET request
    espSerial.find(">"); // Wait for the ">" character from the ESP8266
    espSerial.println(sendData); // Send the GET request
    Serial.println(sendData);

    // Print the values
    Serial.println("Value to be sent: ");
    printBMP(); // Call the printBMP function to print the temperature and pressure

    sendATCommand("AT+CIPCLOSE=0", 1000, DEBUG); // Close the TCP connection
    lastConnectionTime = millis(); // Update the last connection time
}
```

## 6. Helper Functions:

These functions assist in sending AT commands to the ESP8266 and print the BMP280 sensor readings.

```
void sendATCommand(String command, const int timeout, boolean debug) {
    ... // (refer to the provided code for the full sendATCommand function)
}

void printBMP() {
    ... // (refer to the provided code for the full printBMP function)
}
```

## Reference

- 

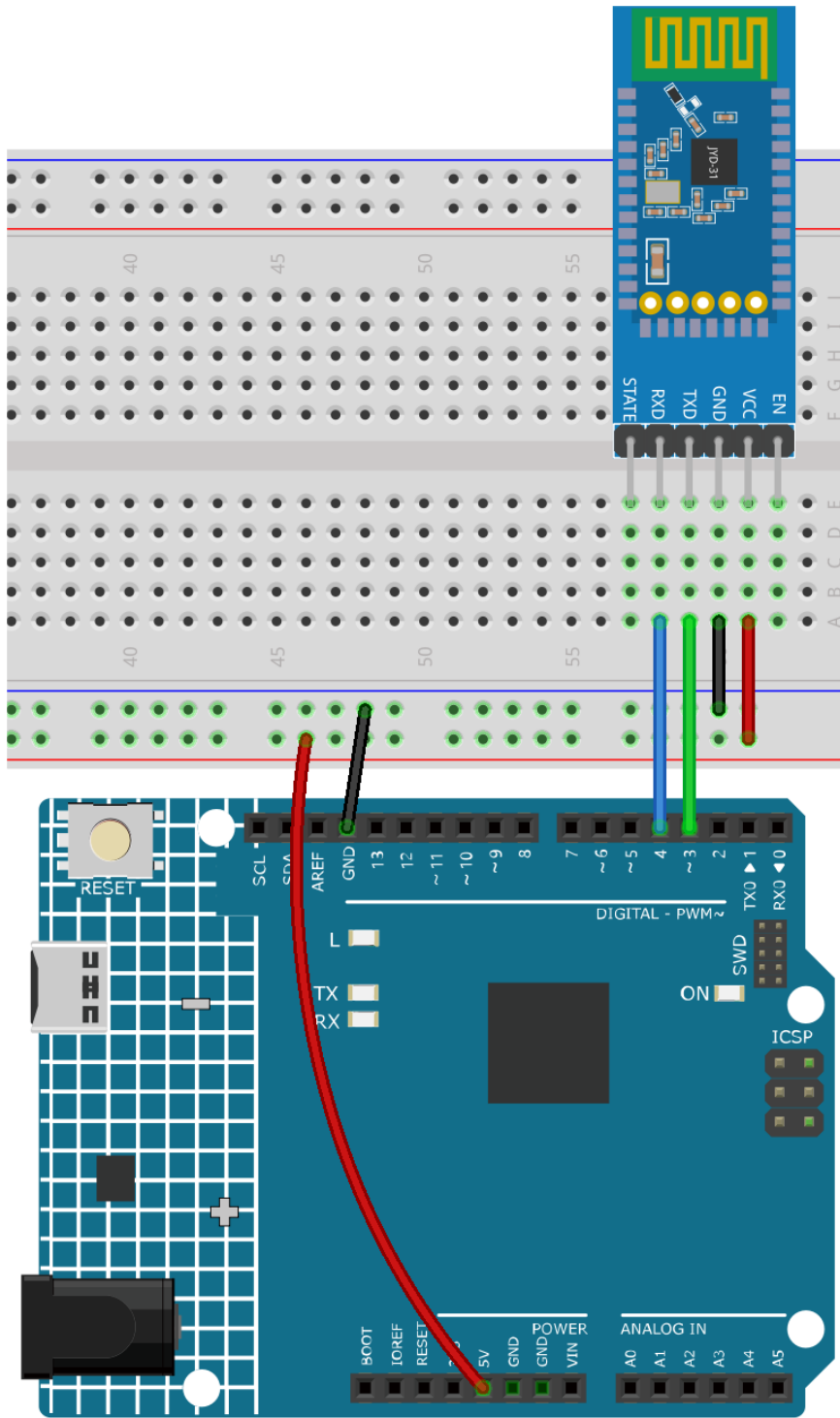
## Bluetooth

### 2.5.9 Get Started with Bluetooth

In this project, we demonstrate how to communicate with a Bluetooth module through Arduino.

Firstly, we need to set up the circuit and use software serial communication. Connect the TX pin of the Bluetooth module to pin 3 of the Uno board, and connect the RX pin of the Bluetooth module to pin 4 of the Uno board.

## 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*

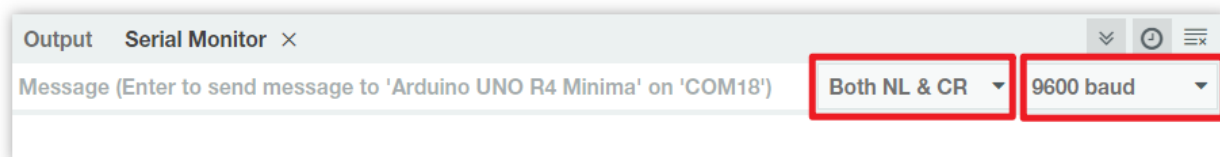
## 2. Upload the code

Open the `00-Bluetooth_start.ino` file under the path of `ultimate-sensor-kit\iot_project\bluetooth\00-Bluetooth_start`, or copy this code into **Arduino IDE**.

The code establishes a software serial communication using Arduino's SoftwareSerial library, allowing the Arduino to communicate with the JDY-31 Bluetooth module through its digital pins 3 and 4 (as Rx and Tx). It checks for data transfer between them, forwarding received messages from one to the other at a baud rate of 9600. **With this code, you can use the Arduino's serial monitor to send AT commands to the JDY-31 Bluetooth module and receive its responses.**

## 3. Configuring the Bluetooth module

Click the magnifying glass icon (Serial Monitor) in the upper right corner and set the baud rate to 9600. Then, select both NL & CR from the drop-down option of the New Line dropdown box.



The following are some examples of using AT commands to configure Bluetooth modules: Enter `AT+NAME` to obtain the name of the Bluetooth device. If you want to modify the Bluetooth name, please add a new name after `AT+NAME`.

- **Query the name of a Bluetooth device:** `AT+NAME`
- **Set Bluetooth device name:** `AT+NAME` (following by the new name). `+OK` means the setting was successful. You can send `AT+NAME` again to verify.

---

**Note:** To ensure consistency in the learning experience, it is recommended not to modify the default baud rate of the Bluetooth module and **keep it at its default value of 4 (i.e. 9600 baud rate)**. In relevant courses, we communicate with Bluetooth using a baud rate of 9600.

---

- **Set Bluetooth baudrate:** `AT+BAUD` (followed by the number indicating the baudrate).
  - 4 == 9600
  - 5 == 19200
  - 6 == 38400
  - 7 == 57600
  - 8 == 115200
  - 9 == 128000

Please refer to the table below for more AT commands.

Command	Function	Default
AT+VERSION	Version Number	JDY-31-V1.2
AT+RESET	Soft reset	
AT+DISC	Disconnect (valid when connected)	
AT+LADDR	Query the MAC address of the module	
AT+PIN	Set or query connection password	1234
AT+BAUD	Set or query baud rate	9600
AT+NAME	Set or query broadcast name	JDY-31-SPP
AT+DEFAULT	Factory reset	
AT+ENLOG	Serial port status output	1

#### 4. Communicating through Bluetooth debugging tools on mobile phones

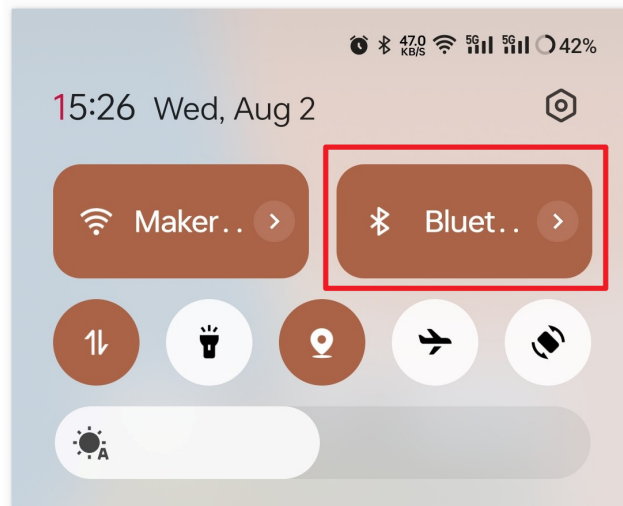
We can use an app called “Serial Bluetooth Terminal” to send messages from the Bluetooth module to Arduino, simulating the process of Bluetooth interaction. The Bluetooth module will send received messages to Arduino through serial port, and similarly, Arduino can also send messages to bluetooth module through serial port.

##### a. Install Serial Bluetooth Terminal

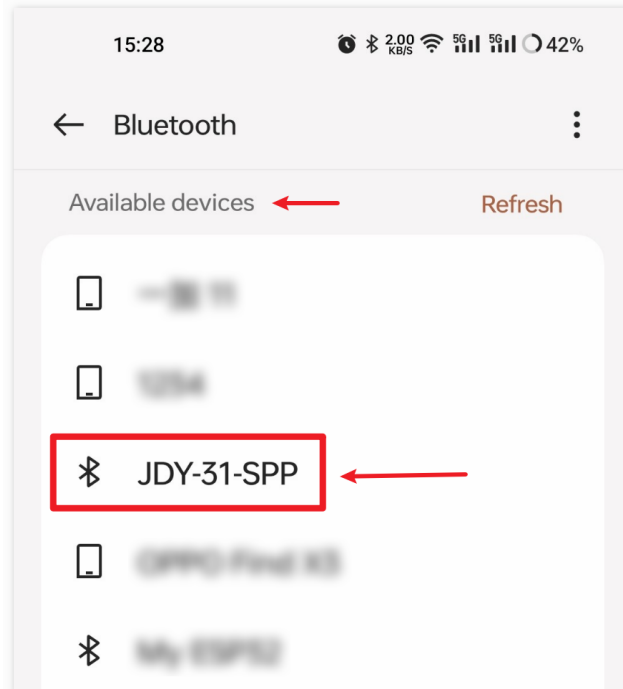
Go to Google Play to download and install .

##### b. Connect Bluetooth

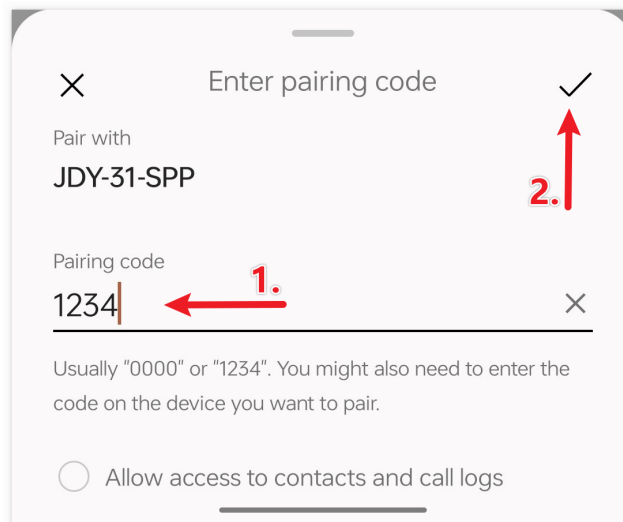
Initially, turn on **Bluetooth** on your smartphone.



Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.



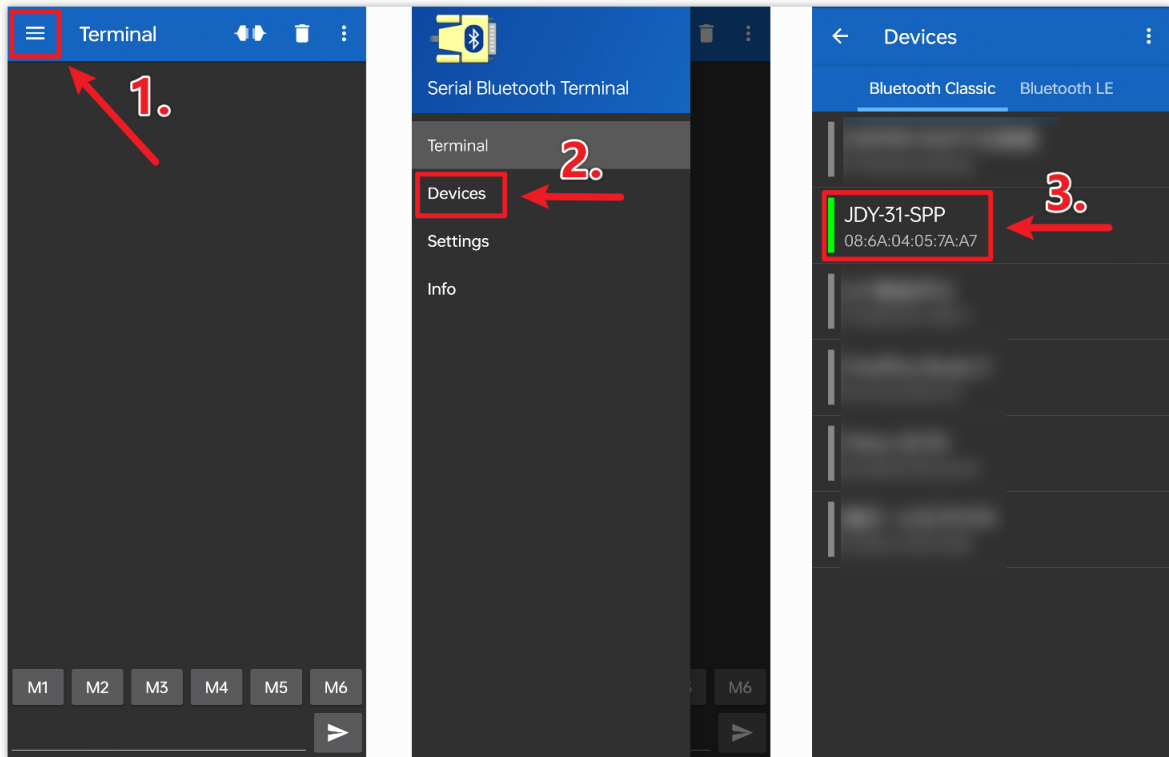
After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



c. **Communicate with Bluetooth module**

Open the Serial Bluetooth Terminal. Connect to “JDY-31-SPP”.

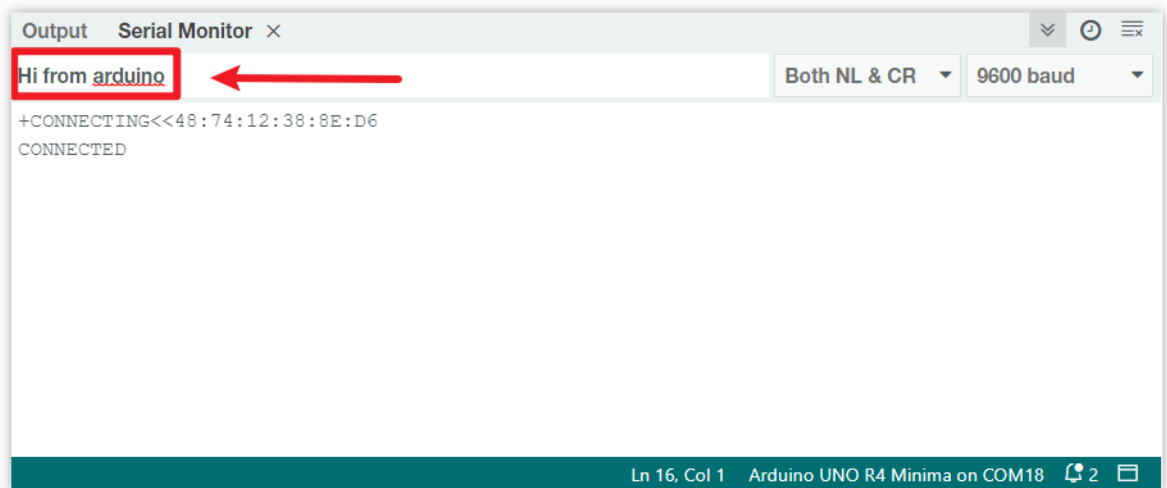




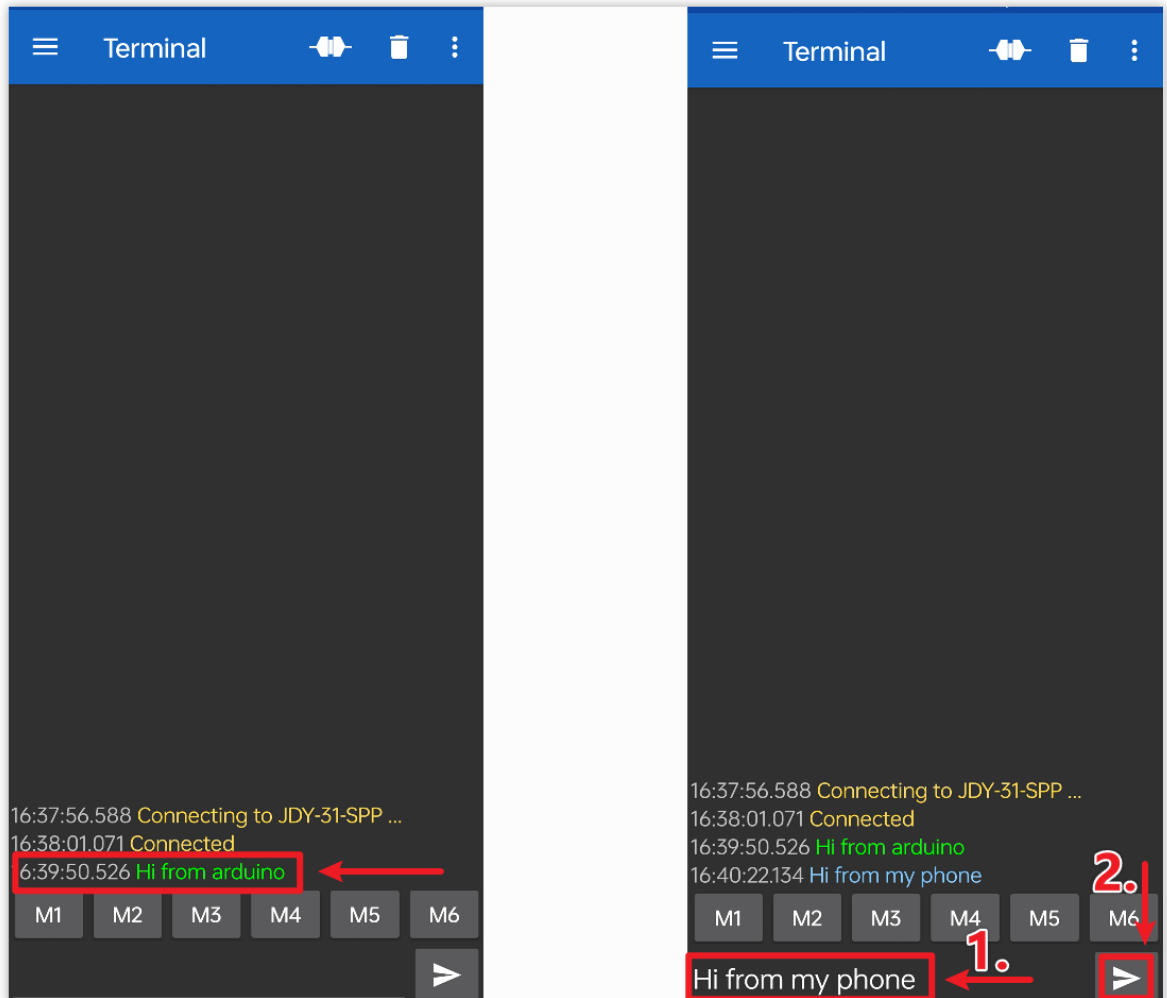
After successful connection, you can see the prompt of successful connection in the serial port monitor.



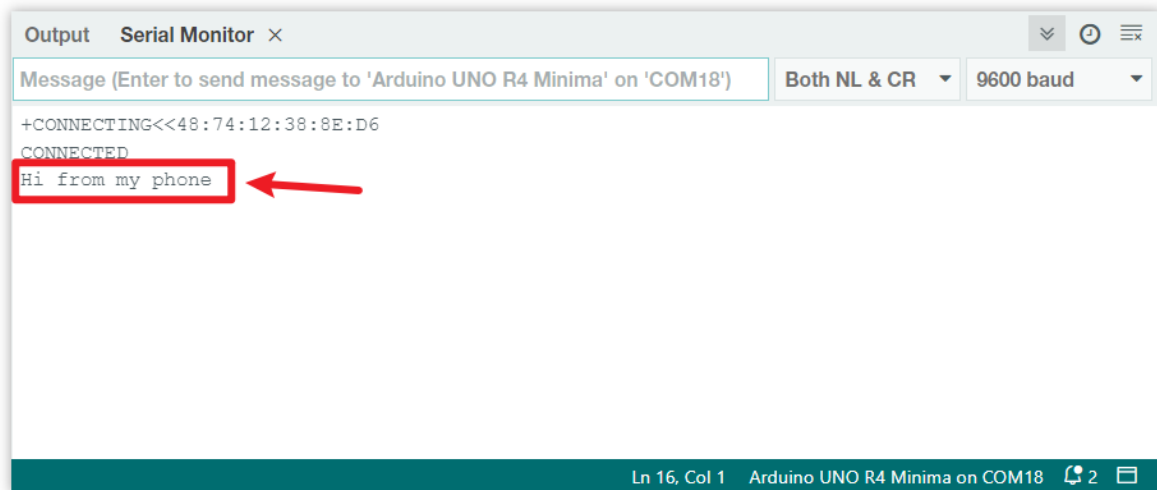
Input the message in the serial monitor and send it to the Bluetooth module.



After sending, you can see this message in the Serial Bluetooth Terminal APP. Similarly, data can be sent to Arduino via Bluetooth in **Serial Bluetooth Terminal APP**.



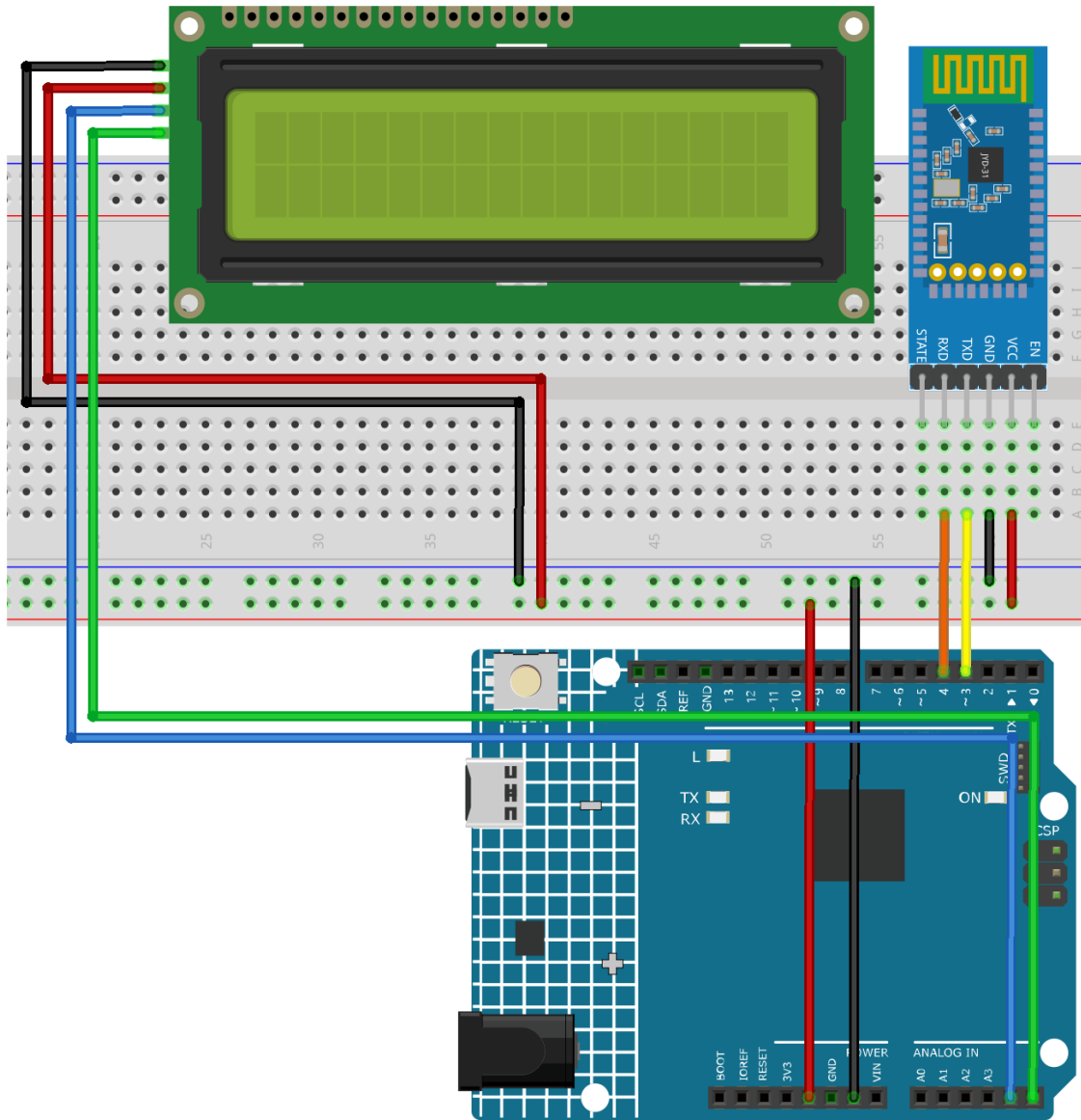
You can see this message from Bluetooth in the serial monitor.



### 2.5.10 Bluetooth LCD

The project receives messages through a Bluetooth module connected to the UNO board and displays the received messages on an LCD screen.

## 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *I2C LCD 1602*

## 2. Upload the Code

1. Open the 01-Bluetooth\_lcd.ino file under the path of ultimate-sensor-kit\iot\_project\bluetooth\01-Bluetooth\_lcd, or copy this code into **Arduino IDE**.

---

**Note:** To install the library, use the Arduino Library Manager and search for “**LiquidCrystal I2C**” and install it.

---

2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

## 3. App and Bluetooth module Connection

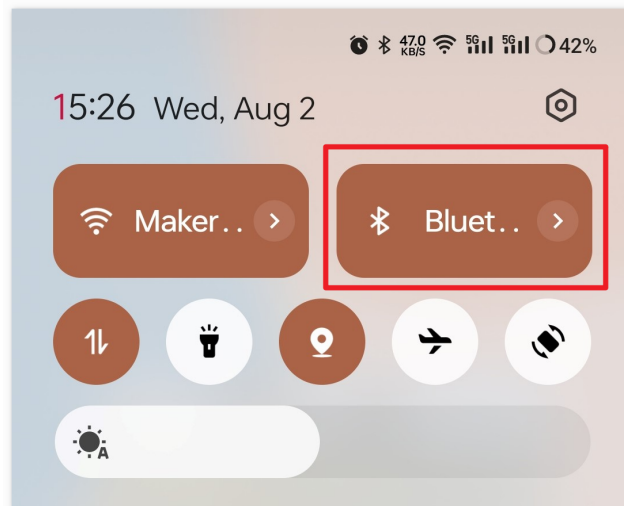
We can use an app called “Serial Bluetooth Terminal” to send messages from the Bluetooth module to Arduino.

### a. Install Serial Bluetooth Terminal

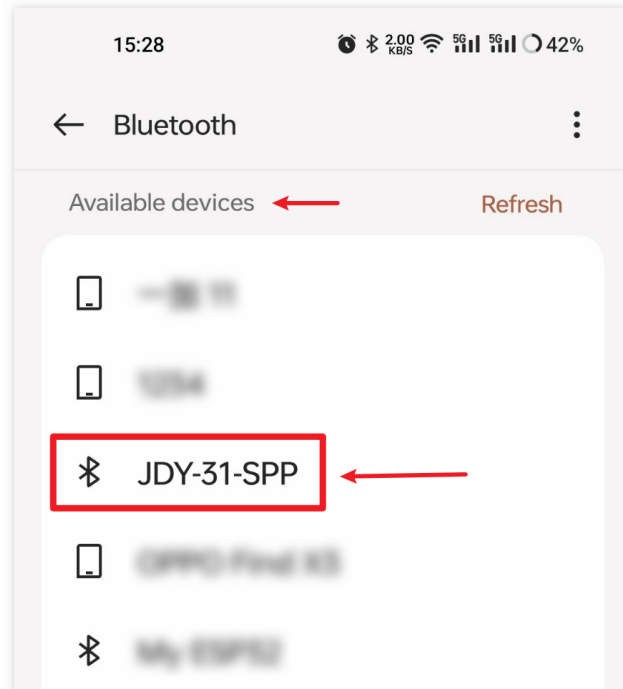
Go to Google Play to download and install .

### b. Connect Bluetooth

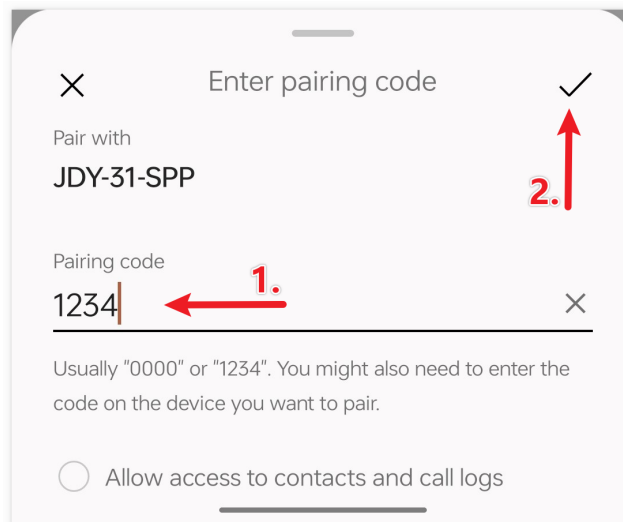
Initially, turn on **Bluetooth** on your smartphone.



Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.

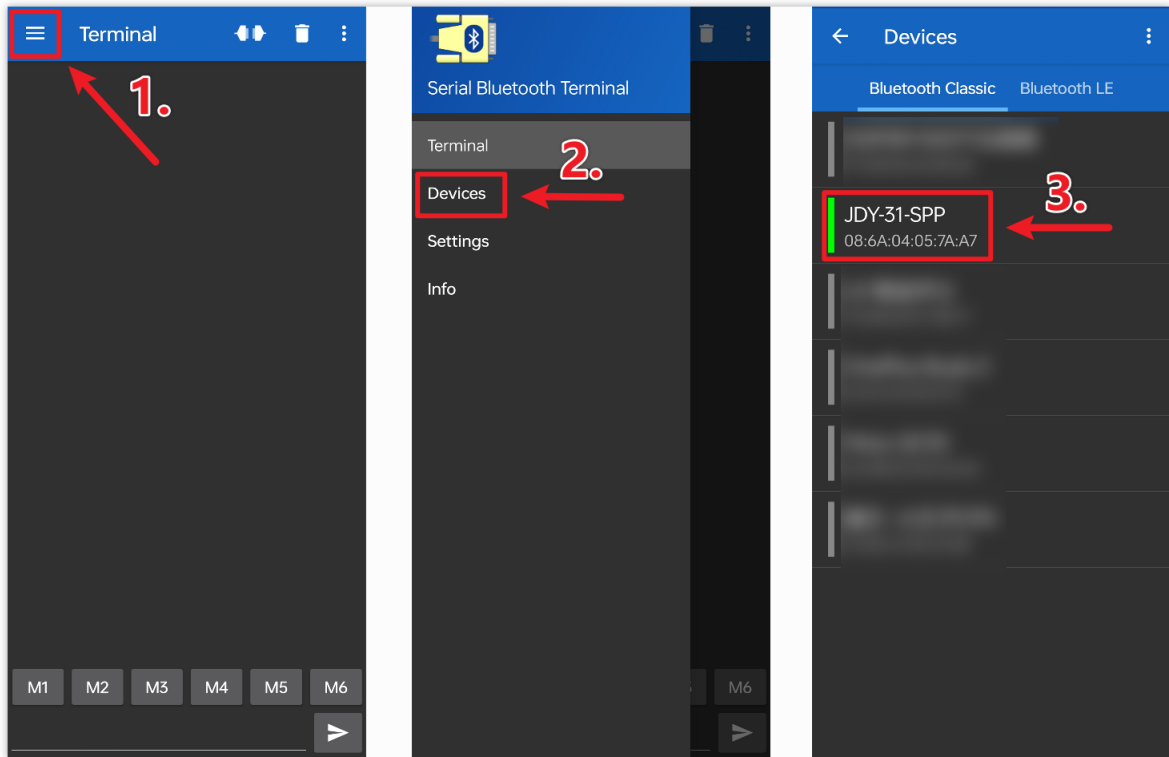


After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



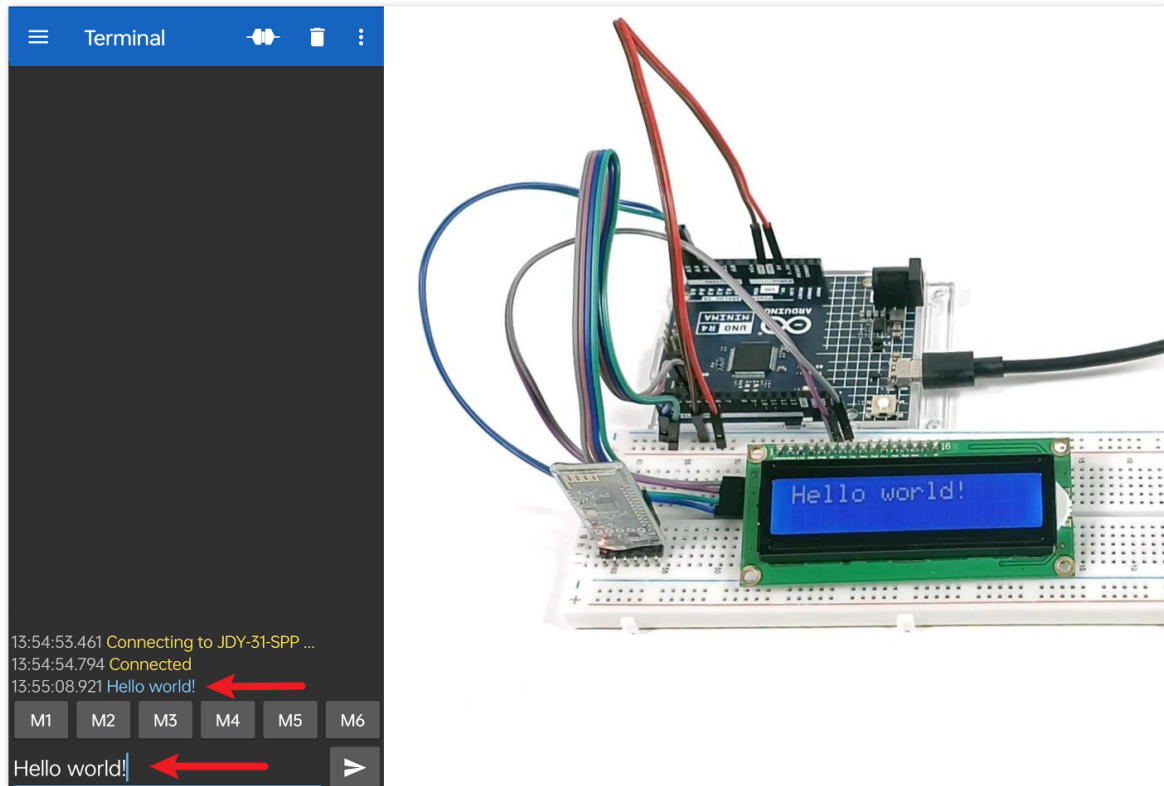
c. **Communicate with Bluetooth module**

Open the Serial Bluetooth Terminal. Connect to “JDY-31-SPP”.



d. **Send command**

Use the Serial Bluetooth Terminal app to send messages to Arduino via Bluetooth. The message sent to Bluetooth will be displayed on the LCD.



#### 4. Code explanation

**Note:** To install library, use the Arduino Library Manager and search for “**LiquidCrystal I2C**” and install the library.

##### 1. Setting up the LCD

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

This segment of code includes the LiquidCrystal\_I2C library and initializes the LCD module with the I2C address as 0x27 and specifies that the LCD has 16 columns and 2 rows.

##### 2. Setting up Bluetooth communication

```
#include <SoftwareSerial.h>
const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);
```

Here, the SoftwareSerial library is included to allow the JDY-31 Bluetooth module to communicate with the Arduino using pins 3 (TX) and 4 (RX).

##### 3. Initialization

```
void setup() {
  lcd.init();
```

(continues on next page)



(continued from previous page)

```
lcd.clear();  
lcd.backlight();  
  
Serial.begin(9600);  
bleSerial.begin(9600);  
}
```

The `setup()` function initializes the LCD and clears any existing content. It also turns on the backlight for the LCD. Communication is started with the serial monitor and the Bluetooth module, both at a baud rate of 9600.

#### 4. Main Loop

```
void loop() {  
  String data;  
  
  if (bleSerial.available()) {  
    data += bleSerial.readString();  
    data = data.substring(0, data.length() - 2);  
    Serial.print(data);  
  
    lcd.clear();  
    lcd.setCursor(0, 0);  
    lcd.print(data);  
  }  
  
  if (Serial.available()) {  
    bleSerial.write(Serial.read());  
  }  
}
```

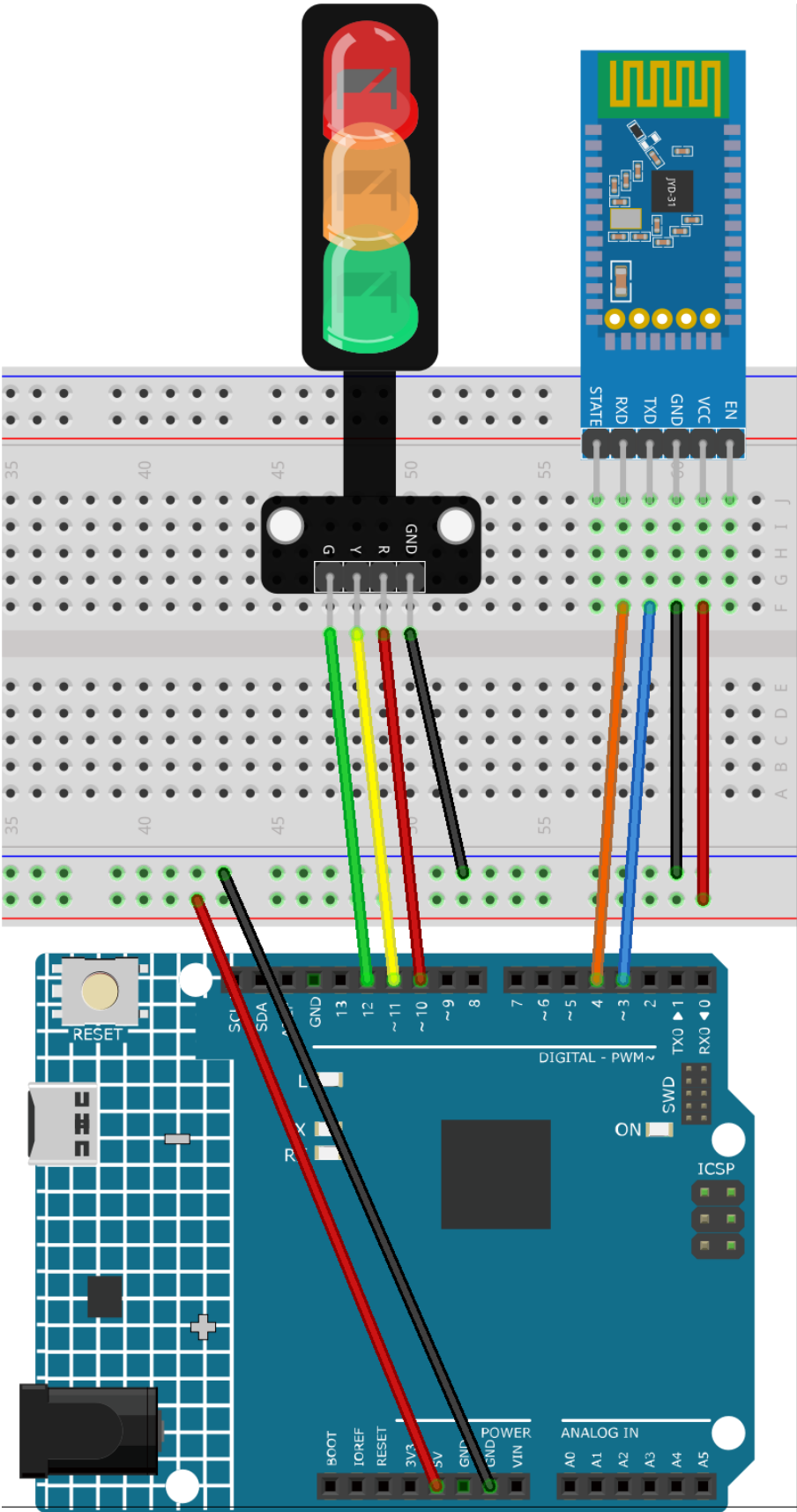
This is the main operational loop of the Arduino program. It continually checks for incoming data from both the Bluetooth module and the serial monitor. When data is received from the Bluetooth device, it's processed, displayed on the serial monitor, and shown on the LCD. If data is entered into the serial monitor, this data is sent to the Bluetooth module.

### 2.5.11 Bluetooth Traffic Light

This project is designed to control a traffic light (Red, Yellow, Green LEDs) using Bluetooth communication. The user can send a character ('R', 'Y', or 'G') from a Bluetooth device. When the Arduino receives one of these characters, it lights up the corresponding LED, while ensuring the other two LEDs are turned off.



1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *Traffic Light Module*

## 2. Upload the Code

1. Open the 02-Bluetooth\_traffic\_light.ino file under the path of ultimate-sensor-kit\iot\_project\bluetooth\02-Bluetooth\_traffic\_light, or copy this code into **Arduino IDE**.
2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

## 3. App and Bluetooth module Connection

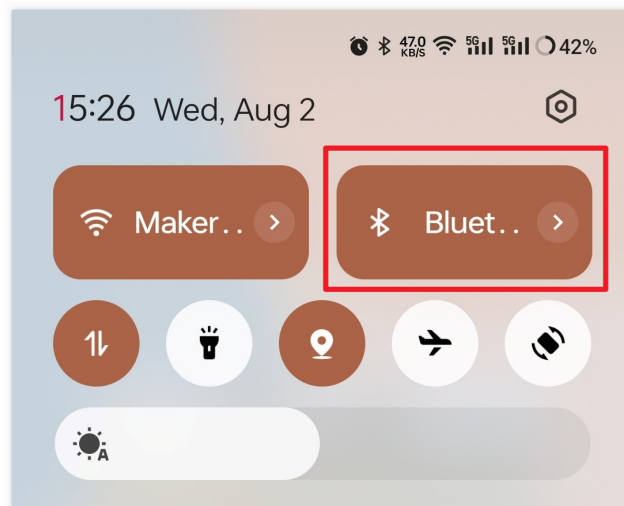
We can use an app called “Serial Bluetooth Terminal” to send messages from the Bluetooth module to Arduino.

### a. Install Serial Bluetooth Terminal

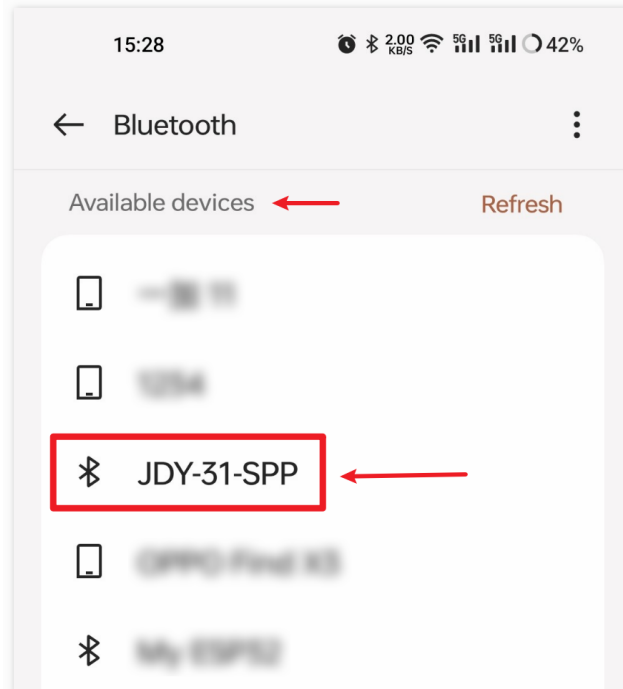
Go to Google Play to download and install .

### b. Connect Bluetooth

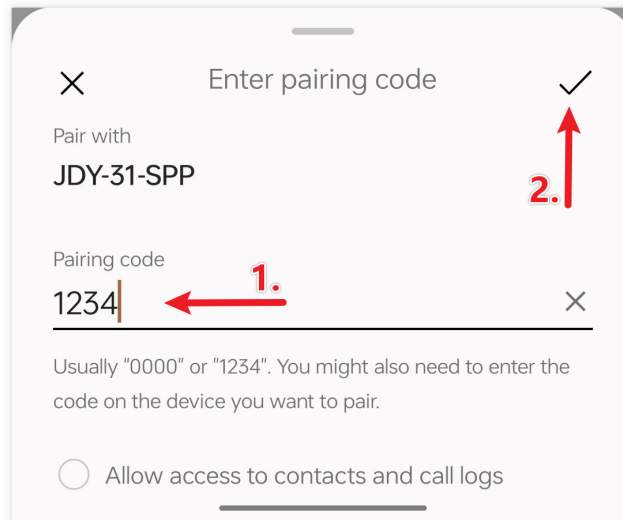
Initially, turn on **Bluetooth** on your smartphone.



Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.

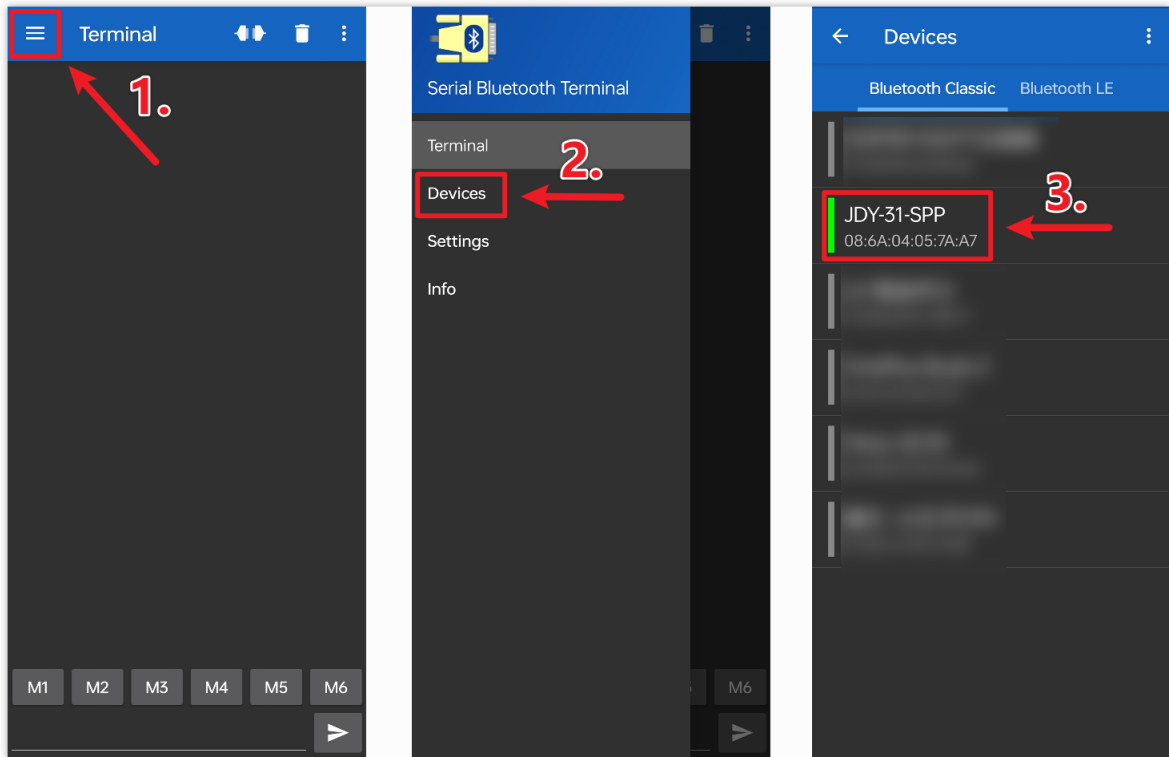


After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



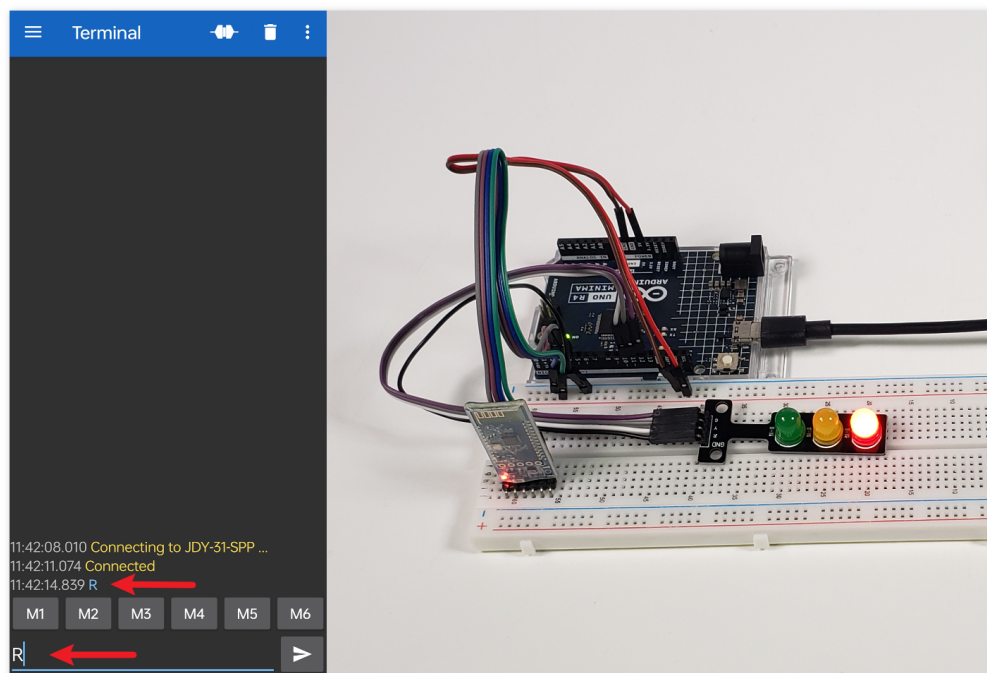
c. **Communicate with Bluetooth module**

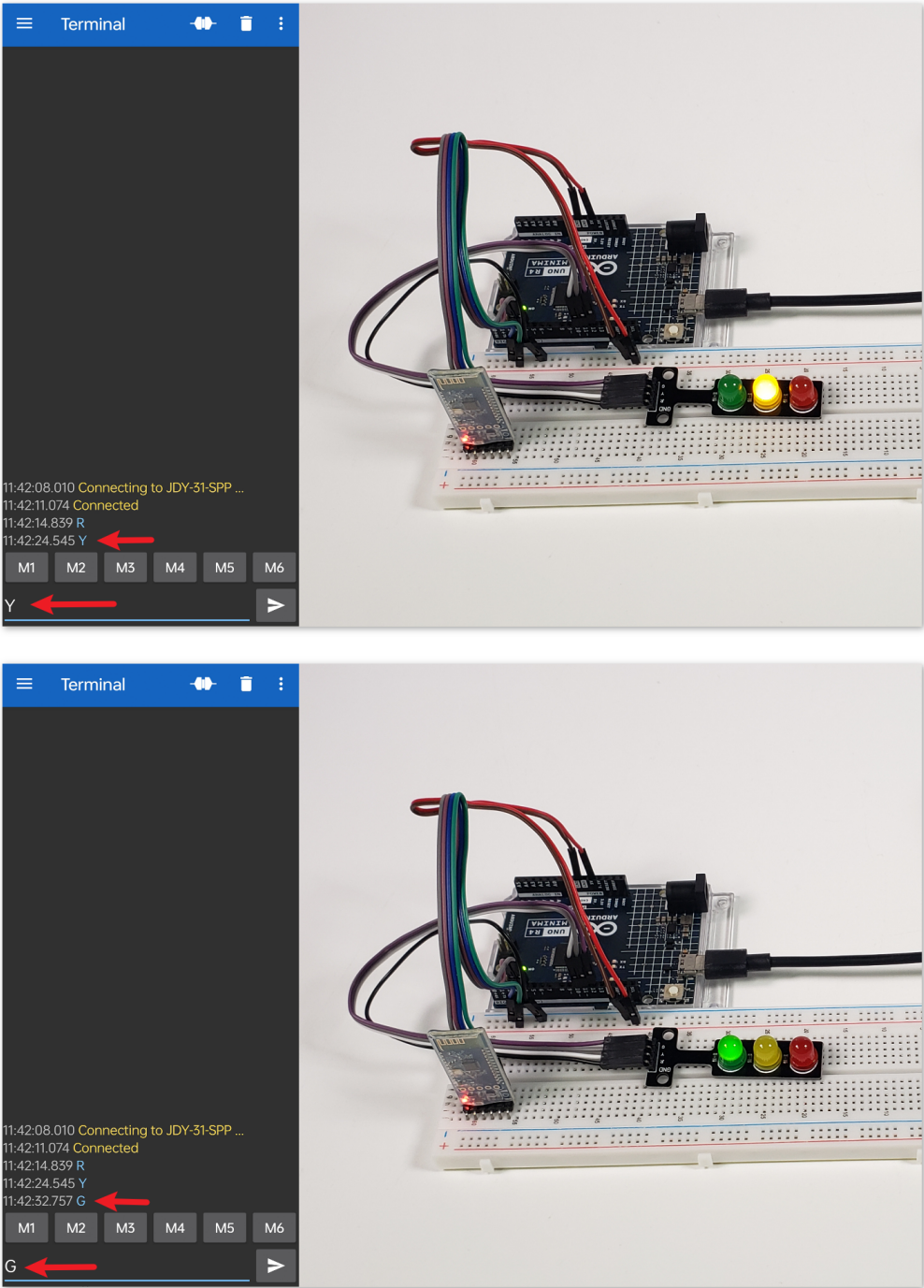
Open the Serial Bluetooth Terminal. Connect to “JDY-31-SPP”.



d. **Send command**

Use the Serial Bluetooth Terminal app to send commands to Arduino via Bluetooth. Send R to turn on the red light, Y for yellow, and G for green.





## 4. Code explanation

### 1. Initialization and Bluetooth setup

```
// Set up Bluetooth module communication
#include <SoftwareSerial.h>
const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);
```

We begin by including the SoftwareSerial library to help us with Bluetooth communication. The Bluetooth module's TX and RX pins are then defined and associated with pins 3 and 4 on the Arduino. Finally, we initialize the bleSerial object for Bluetooth communication.

### 2. LED Pin Definitions

```
// Pin numbers for each LED
const int rledPin = 10; //red
const int yledPin = 11; //yellow
const int gledPin = 12; //green
```

Here, we're defining which Arduino pins our LEDs are connected to. The red LED is on pin 10, yellow on 11, and green on 12.

### 3. setup() Function

```
void setup() {
  pinMode(rledPin, OUTPUT);
  pinMode(yledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);

  Serial.begin(9600);
  bleSerial.begin(9600);
}
```

In the setup() function, we set the LED pins as OUTPUT. We also start serial communication for both the Bluetooth module and the default serial (connected to the computer) at a baud rate of 9600.

### 4. Main loop() for Bluetooth Communication

```
void loop() {
  while (bleSerial.available() > 0) {
    character = bleSerial.read();
    Serial.println(character);

    if (character == 'R') {
      toggleLights(rledPin);
    } else if (character == 'Y') {
      toggleLights(yledPin);
    } else if (character == 'G') {
      toggleLights(gledPin);
    }
  }
}
```

Inside our main loop(), we continuously check if data is available from the Bluetooth module. If we receive



data, we read the character and display it in the serial monitor. Depending on the character received (R, Y, or G), we toggle the respective LED using the `toggleLights()` function.

#### 5. Toggle Lights Function

```
void toggleLights(int targetLight) {  
    digitalWrite(rledPin, LOW);  
    digitalWrite(yledPin, LOW);  
    digitalWrite(gledPin, LOW);  
  
    digitalWrite(targetLight, HIGH);  
}
```

This function, `toggleLights()`, turns off all the LEDs first. After ensuring they are all off, it turns on the specified target LED. This ensures that only one LED is on at a time.

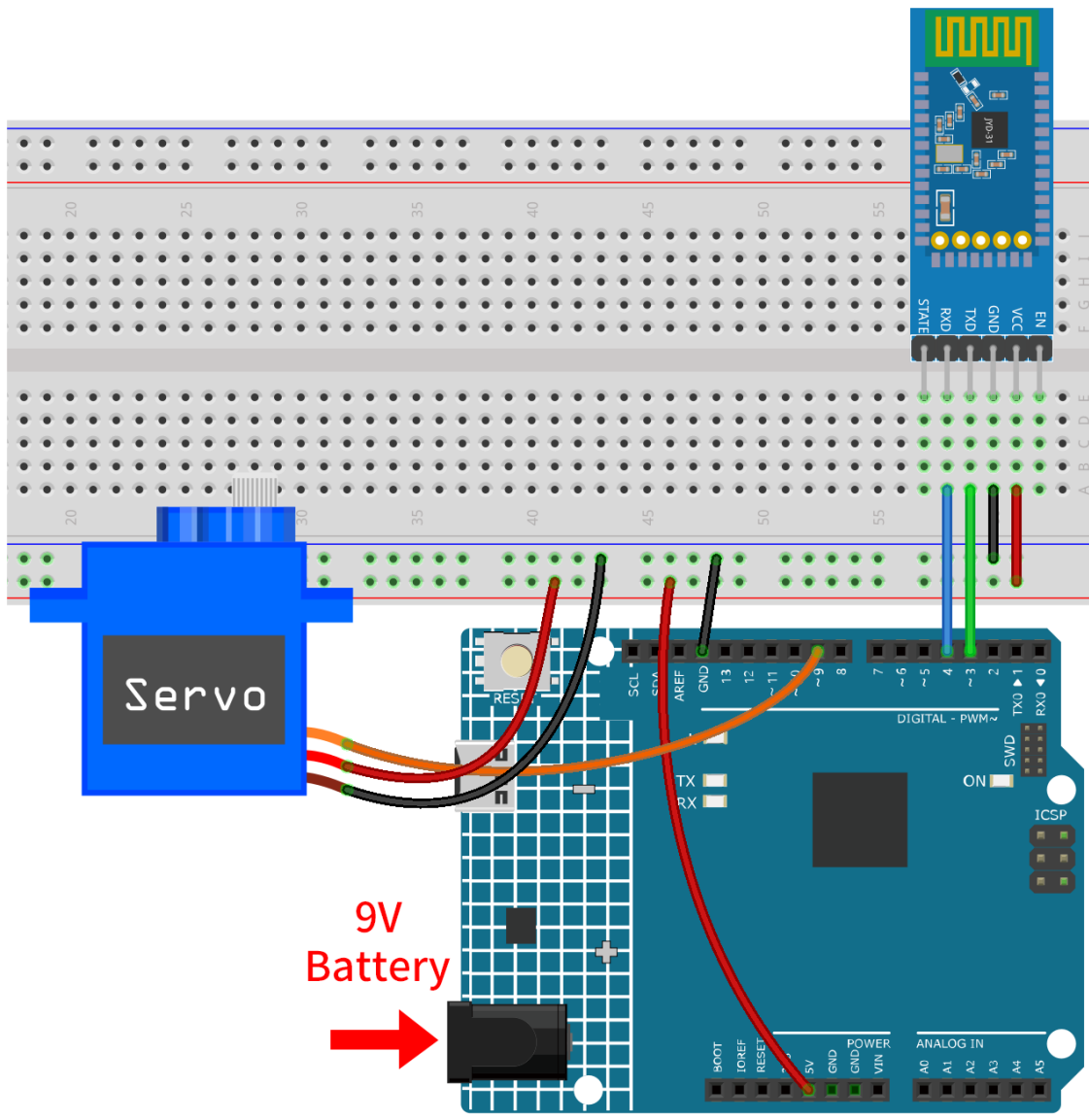
### 2.5.12 Bluetooth Lock Controller

This project uses an Android app created with MIT App Inventor to remotely control a servo motor via Bluetooth, simulating a locking mechanism. Users can command the servo to move to either the “locked” or “unlocked” position by sending specific messages through the app.

The system uses a JDY-31 Bluetooth module to receive these messages and instructs an Arduino Uno board to adjust the servo motor’s angle accordingly. The servo transitions to a “locked” position upon receiving the ‘1’ message and to an “unlocked” position upon receipt of the ‘0’ message.

This Android application will be constructed utilizing a complimentary web-based platform known as . The project presents an excellent opportunity to gain familiarity with the interfacing of an Arduino with a smartphone.

## 1. Build the Circuit



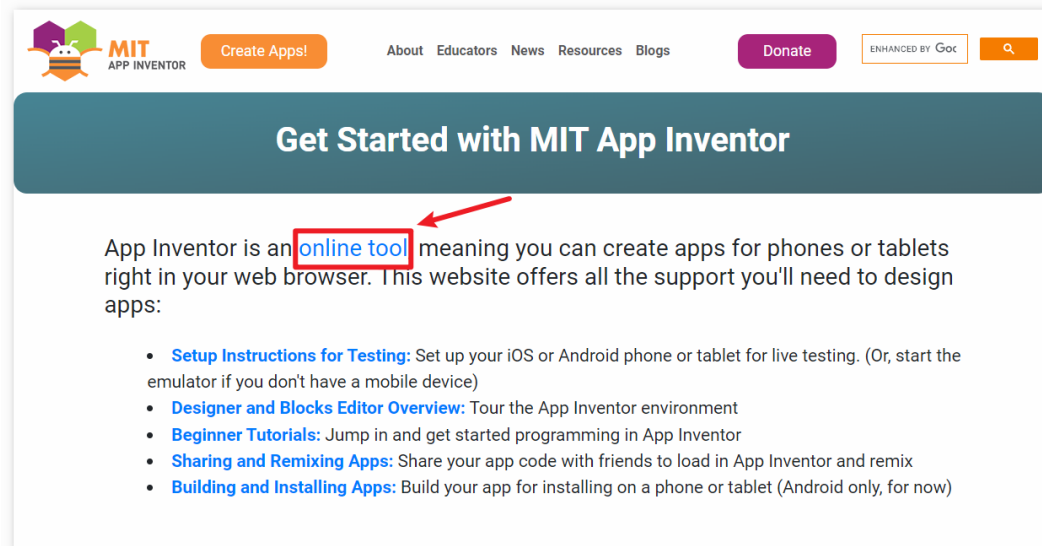
- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *Servo Motor (SG90)*

## 2. Create the Android App

The Android application will be developed using a free web application known as . MIT App Inventor serves as an excellent starting point for Android development, owing to its intuitive drag-and-drop features allowing for the creation of simplistic applications.

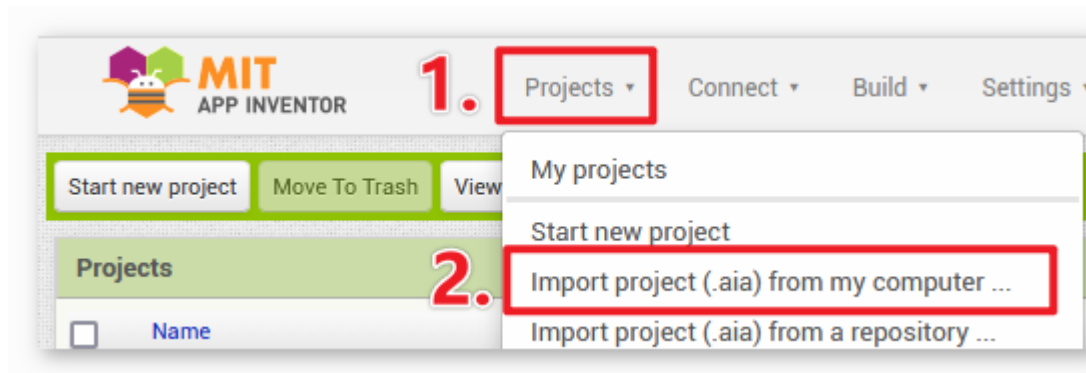
Now, let's begin.

1. Go to , and click “online tool” to login. You will require a Google account to register with MIT App Inventor.

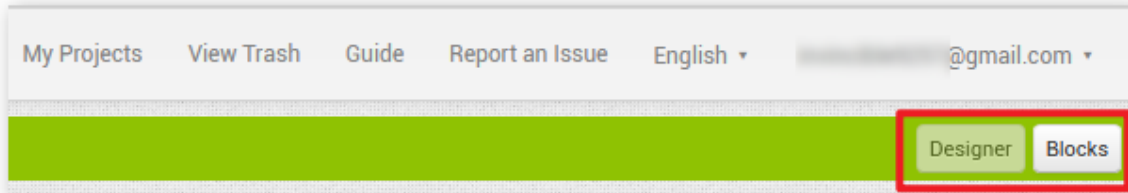


2. After logging in, navigate to **Projects -> Import project (.aia) from my computer**. Subsequently, upload the Bluetooth\_controlled\_lock.aia file located in the path ultimate-sensor-kit\iot\_project\bluetooth\03-Bluetooth\_lock\_controller.

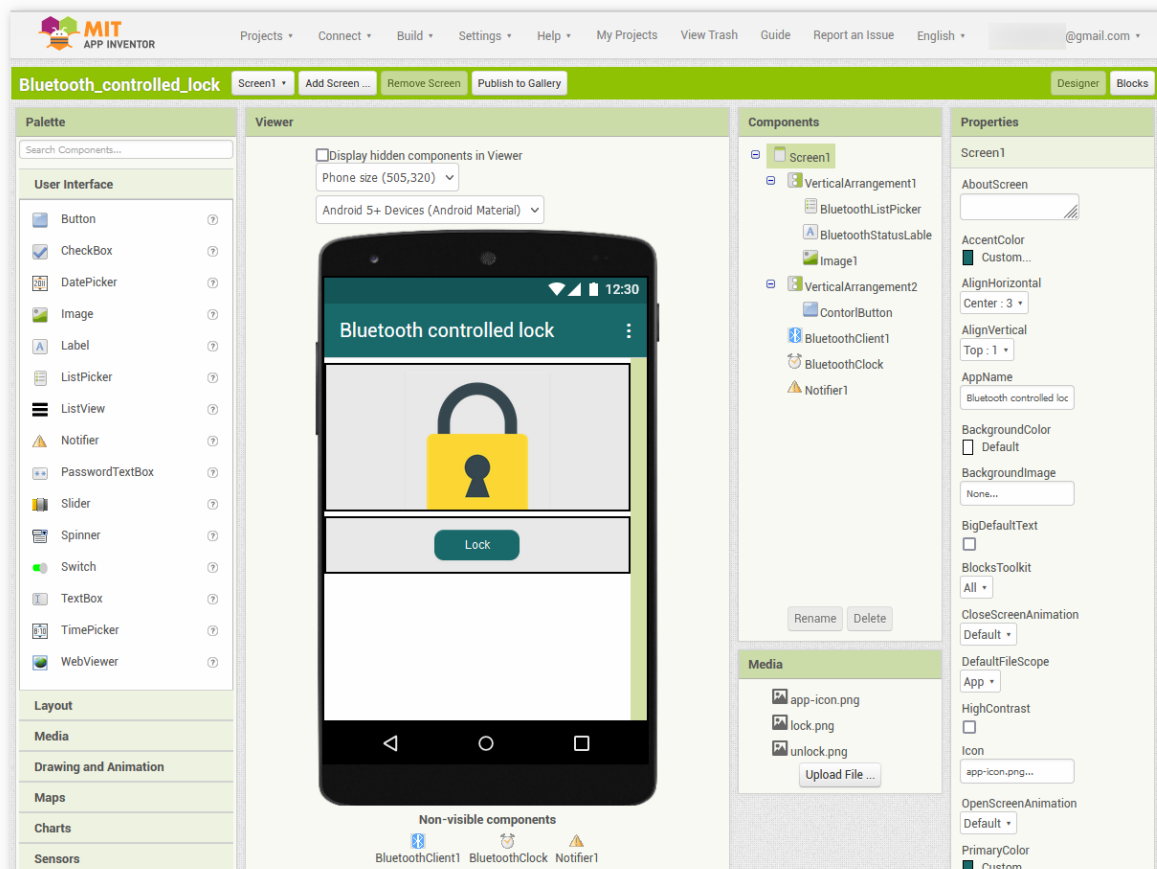
You can also directly download here: Bluetooth\_controlled\_lock.aia



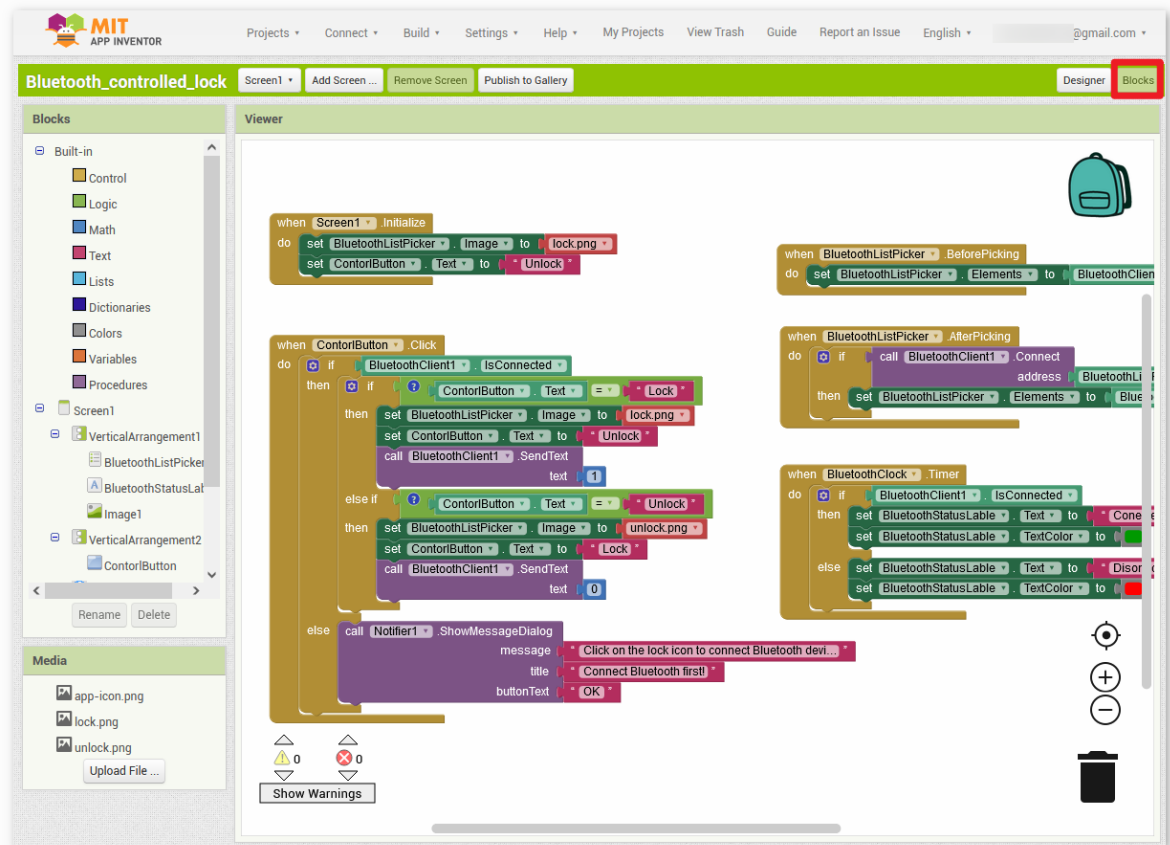
3. Upon uploading the .aia file, you will see the application on the MIT App Inventor software. This is a pre-configured template. You can modify this template after you have familiarized yourself with MIT App Inventor through the following steps.
4. In MIT App Inventor, you have 2 primary sections: the **Designer** and the **Blocks**. You can switch between these two sections in the upper right corner of the page.



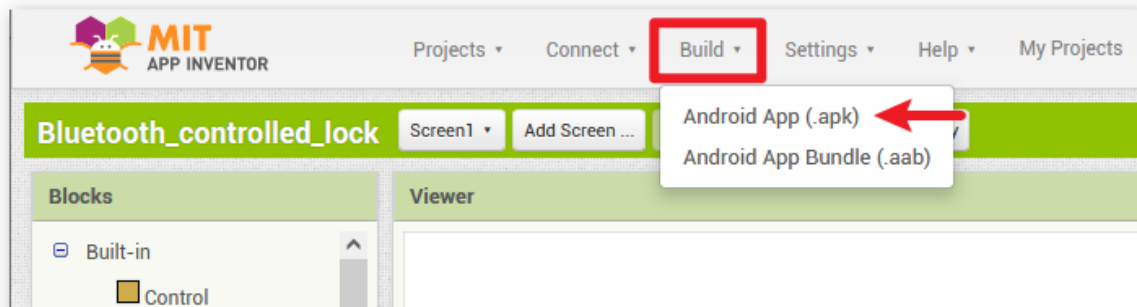
5. The **Designer** allows you to add buttons, text, screens, and modify the overall aesthetic of your application.



6. Next, there's the **Blocks** section. This section lets you craft custom functionalities for your app, allowing you to program each component on the app's GUI to achieve desired features.



7. To install the application on a smartphone, navigate to the **Build** tab.



- You can generate a .apk file. After selecting this option, a page will appear allowing you to choose between downloading a .apk file or scanning a QR code for installation. Follow the installation guide to complete the application installation.

You also download our pre-compiled APK here: [Bluetooth\\_controlled\\_lock.apk](#)

- If you wish to upload this app to Google Play or another app marketplace, you can generate a .aab file.

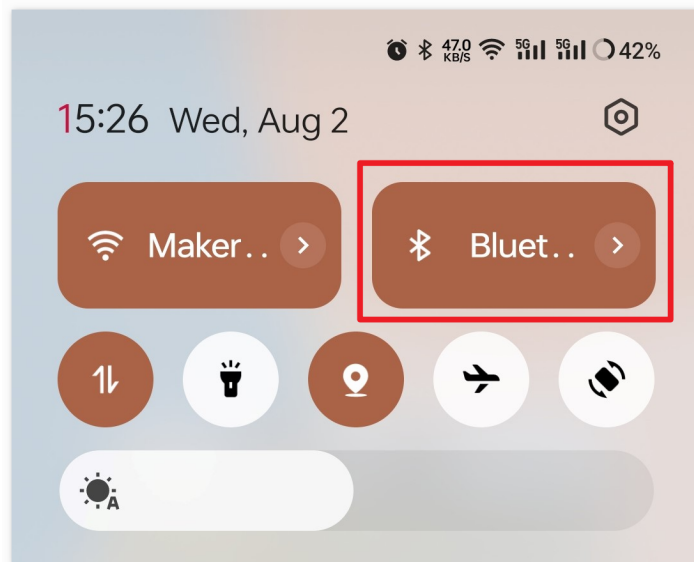
### 3. Upload the Code

1. Open the 03-Bluetooth\_lock\_controller.ino file under the path of ultimate-sensor-kit\iot\_project\bluetooth\03-Bluetooth\_lock\_controller, or copy this code into **Arduino IDE**.
2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

### 4. App and Bluetooth module Connection

Ensure that the application created earlier is installed on your smartphone.

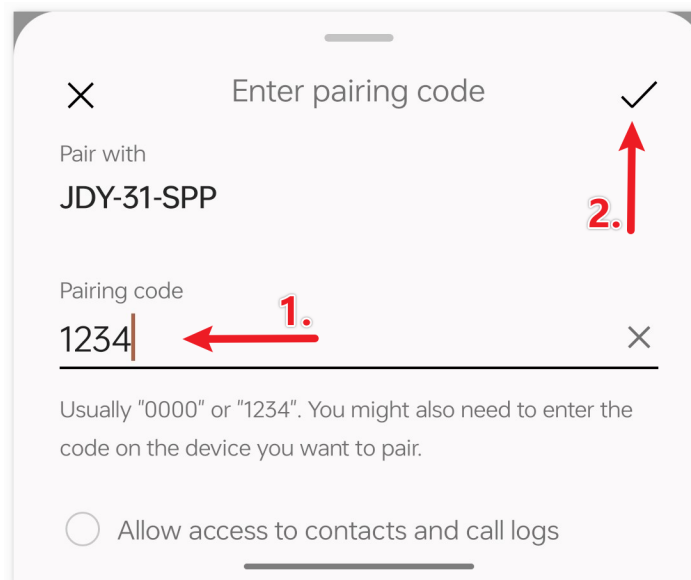
1. Initially, turn on **Bluetooth** on your smartphone.



2. Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.



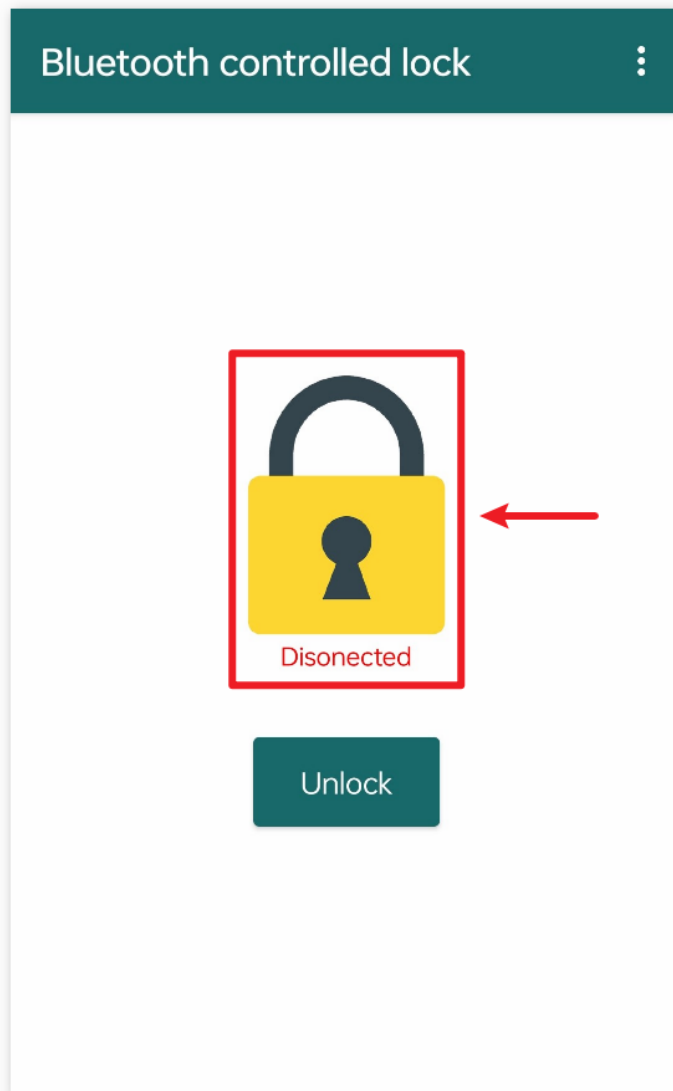
3. After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



4. Now open the newly installed **Control\_RGB\_LED** APP.

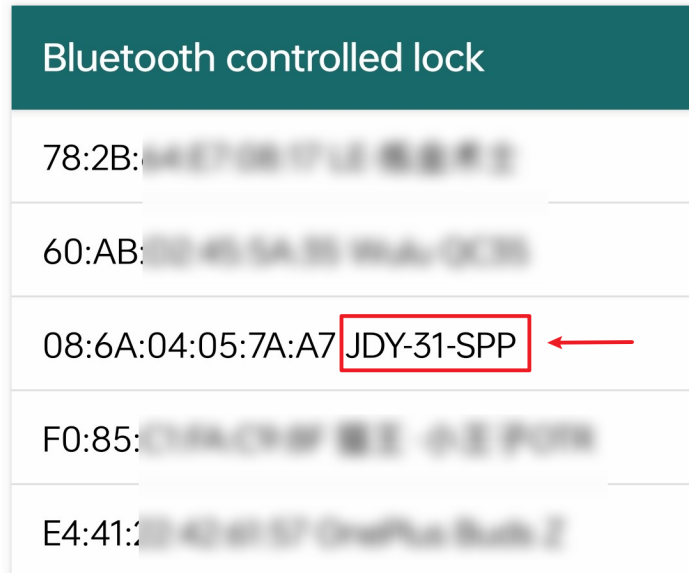


5. In the APP, click on the **lock icon** to establish a connection between the APP and Bluetooth module.

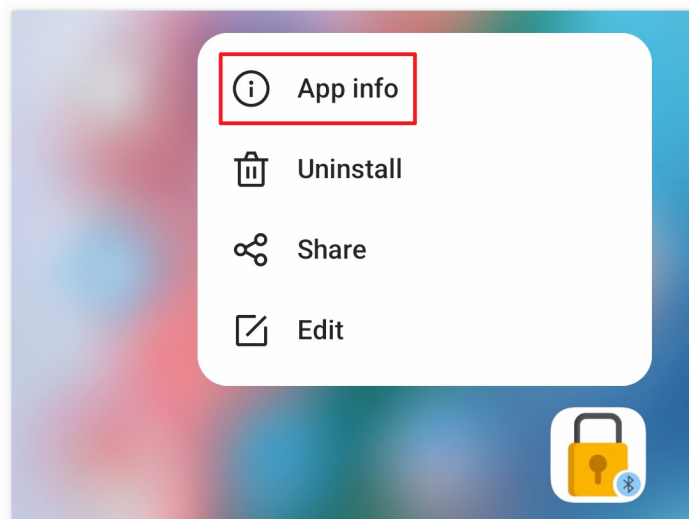


6. This page displays a list of all paired Bluetooth devices. Choose the `xx.xx.xx.xx.xx.xx JDY-31-SPP` option from the list. The name of each device is listed next to its MAC address.

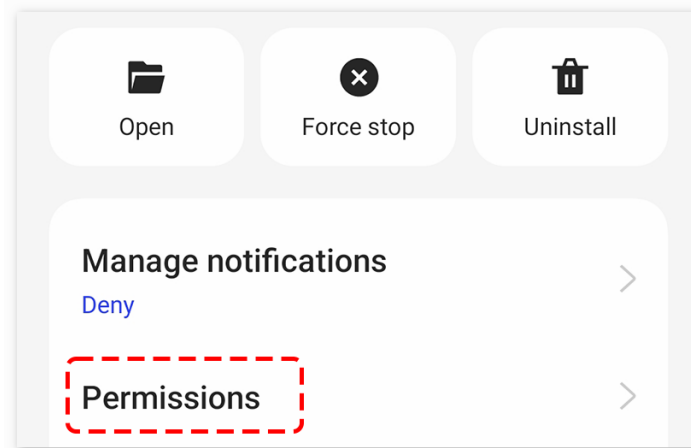




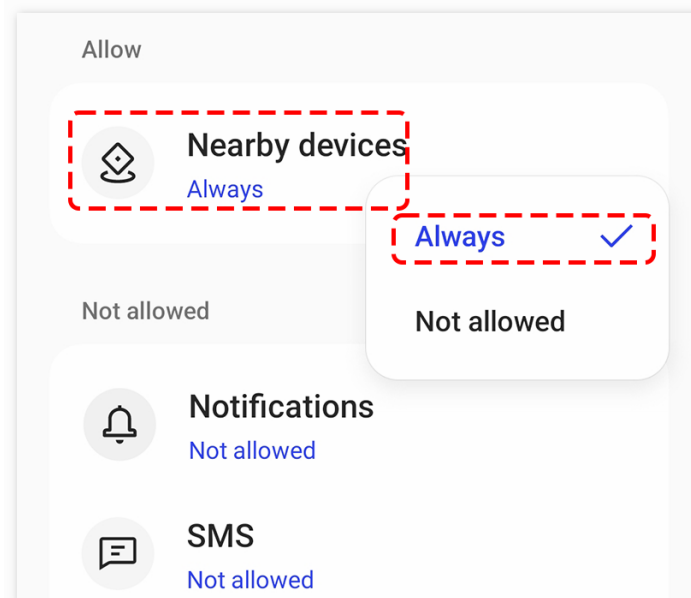
7. If you don't see any devices on the page shown above, it could be because this app is not authorized to scan for nearby devices. In such a case, you will need to adjust the settings manually.
- To access the **APP Info** page, long-press the app icon and select it. Alternatively, if you have another method to reach this page, use that instead.



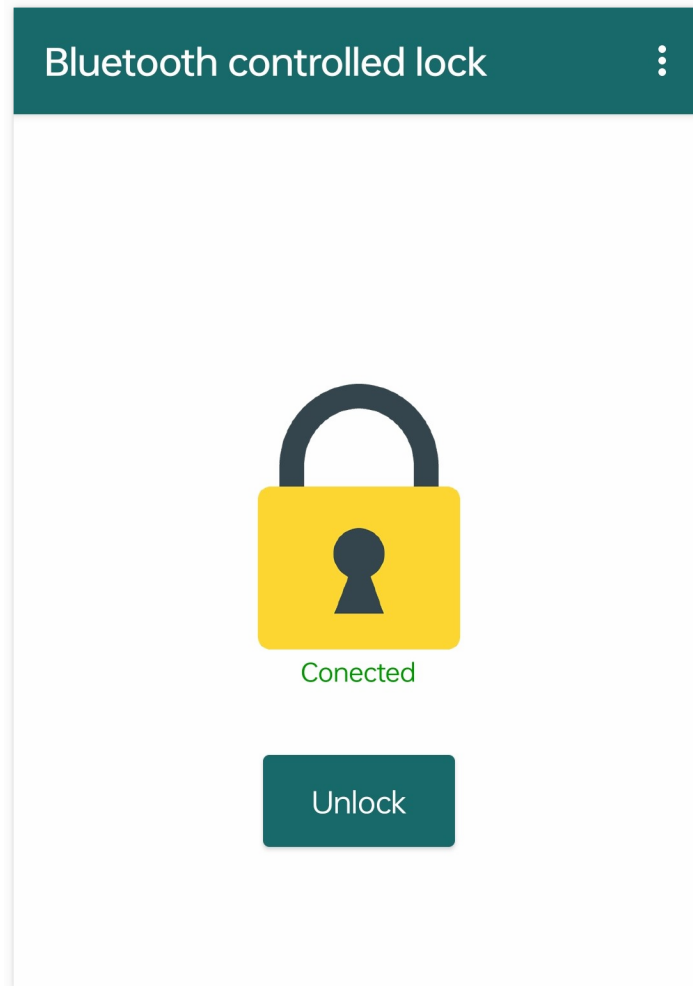
- Navigate to the **Permissions** page.



- To enable the APP to scan for nearby devices, go to **Nearby devices** and select **Always**.



- Now, restart the APP and repeat steps 5 and 6 to successfully connect to Bluetooth.
8. After a successful connection, you will be redirected to the main page where it will display “connected”. Then, you can click on either “Unlock” or “Lock” to control the locking mechanism.



## 5. Code explanation

1. Define the communication pins and initialize the SoftwareSerial library

```
const int bluetoothTx = 3;  
const int bluetoothRx = 4;  
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);
```

The above code defines the transmit (Tx) and receive (Rx) pins used by the JDY-31 Bluetooth module for communication. It then initializes the SoftwareSerial library, which allows the Bluetooth module to communicate with the Arduino board.

2. Define servo-related constants and create a servo object

```
const int servoPin = 9;  
const int lockAngle = 180;  
const int unlockAngle = 90;  
Servo myservo;
```

Here, the pin attached to the servo is defined, along with the angles for “lock” and “unlock” positions. A Servo object myservo is also created for controlling the servo motor.

## 3. Initialize the servo and serial communications

```
void setup() {  
  myservo.attach(servoPin);  
  Serial.begin(9600);  
  bleSerial.begin(9600);  
}
```

## 4. Control servo based on Bluetooth module's input

```
void loop() {  
  if (bleSerial.available() > 0) {  
    char message = bleSerial.read();  
    if (message == '1') {  
      myservo.write(lockAngle);  
      Serial.println("Locked");  
    }  
    else if (message == '0') {  
      myservo.write(unlockAngle);  
      Serial.println("Unlocked");  
    }  
  }  
}
```

The `loop()` function runs repeatedly. It reads incoming messages from the Bluetooth module. If the message is '1', the servo is moved to the “locked” position, and if the message is '0', the servo is moved to the “unlocked” position. The current status (“Locked” or “Unlocked”) is printed to the Serial Monitor.

### 2.5.13 Bluetooth RGB Controller

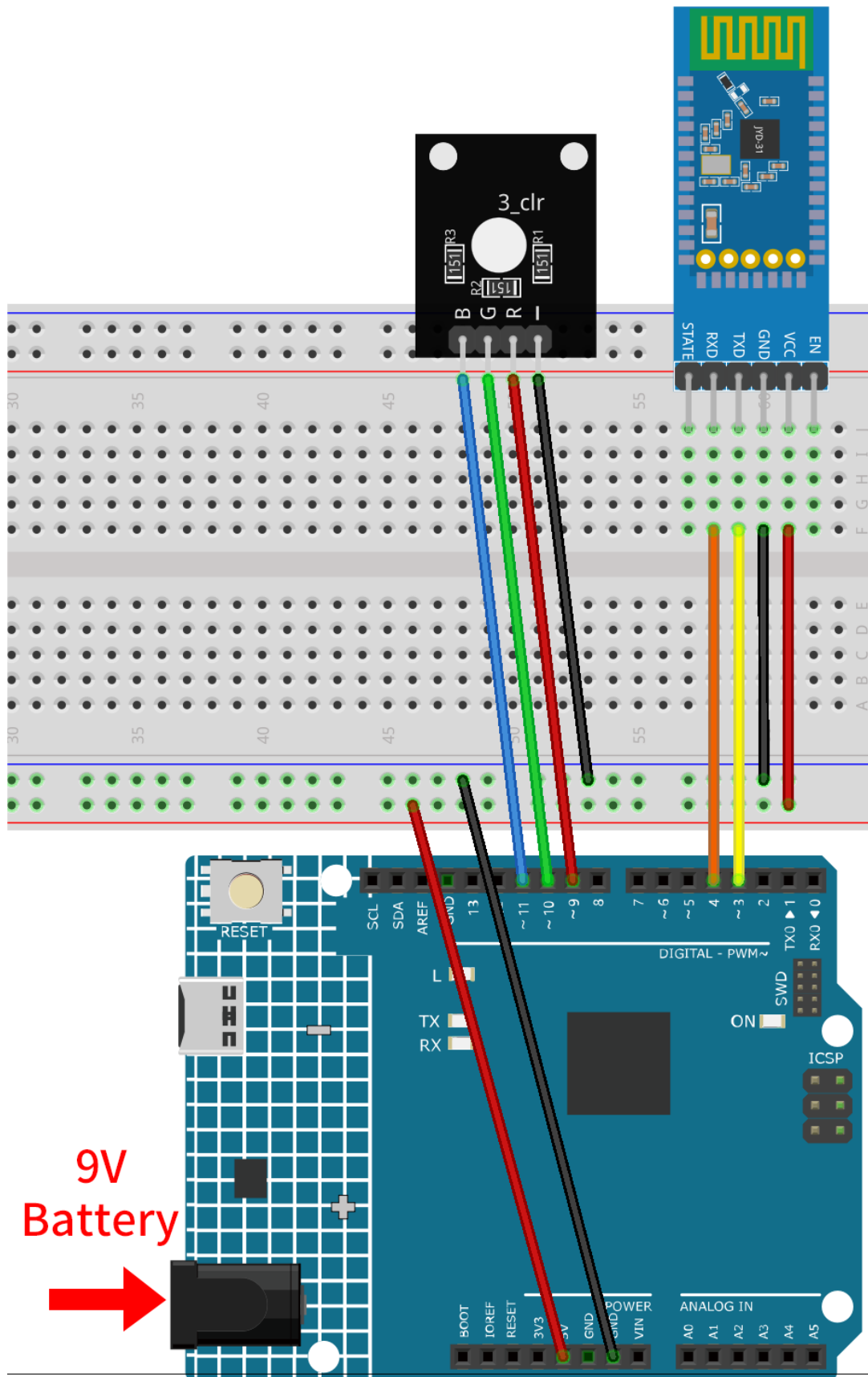
This project uses an Android app to control the color of an RGB LED through Bluetooth technology using a smartphone.

This Android application will be constructed utilizing a complimentary web-based platform known as [Blyn](#). The project presents an excellent opportunity to gain familiarity with the interfacing of an Arduino with a smartphone.

This project control an RGB LED connected to an Arduino Uno via a JDY-31 Bluetooth module. The Android application is used to send various color values to the Arduino Uno board via Bluetooth, based on user operations on the GUI. The program on Uno board receives RGB color values as characters from a serial port over Bluetooth and adjusts the LED's color accordingly.



## 1. Build the Circuit



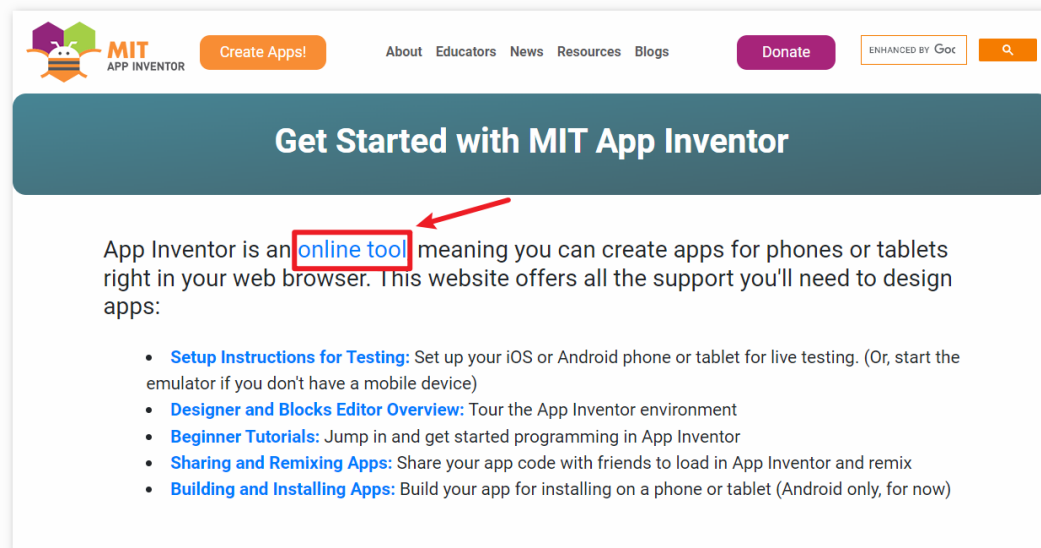
- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *RGB Module*

## 2. Create the Android App

The Android application will be developed using a free web application known as . MIT App Inventor serves as an excellent starting point for Android development, owing to its intuitive drag-and-drop features allowing for the creation of simplistic applications.

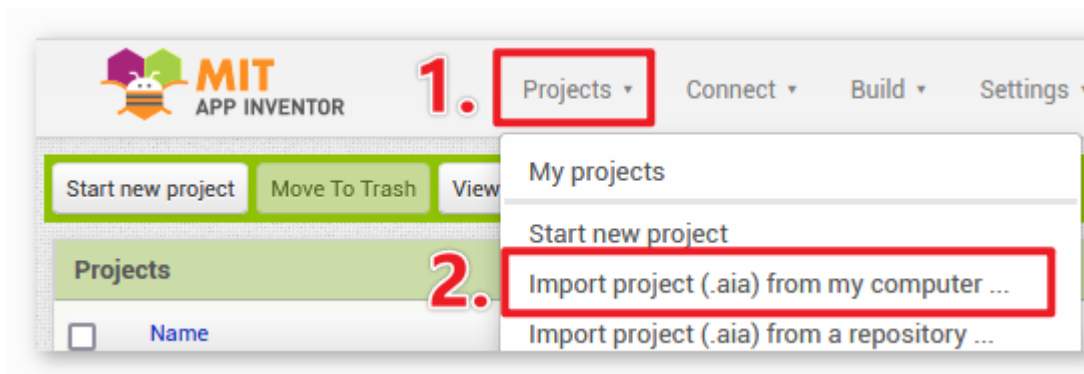
Now, let's begin.

1. Go to , and click “online tool” to login. You will require a Google account to register with MIT App Inventor.

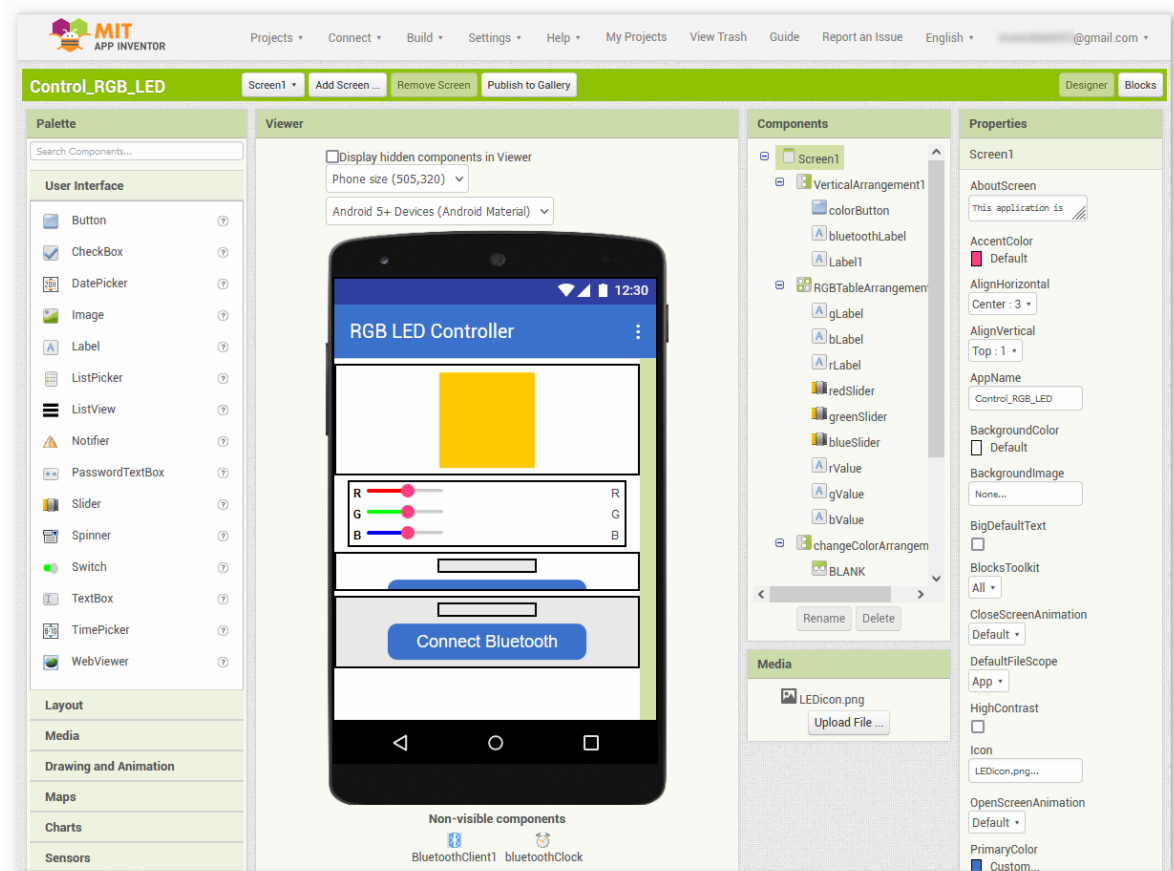


2. After logging in, navigate to **Projects -> Import project (.aia) from my computer**. Subsequently, upload the Control\_RGB\_LED.aia file located in the path ultimate-sensor-kit\iot\_project\bluetooth\04-Bluetooth\_RGB\_controller.

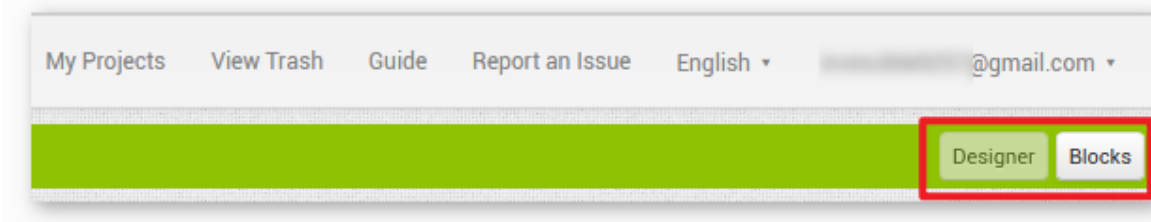
You can also directly download here: Control\_RGB\_LED.aia



3. Upon uploading the .aia file, you will see the application on the MIT App Inventor software. This is a pre-configured template. You can modify this template after you have familiarized yourself with MIT App Inventor through the following steps.

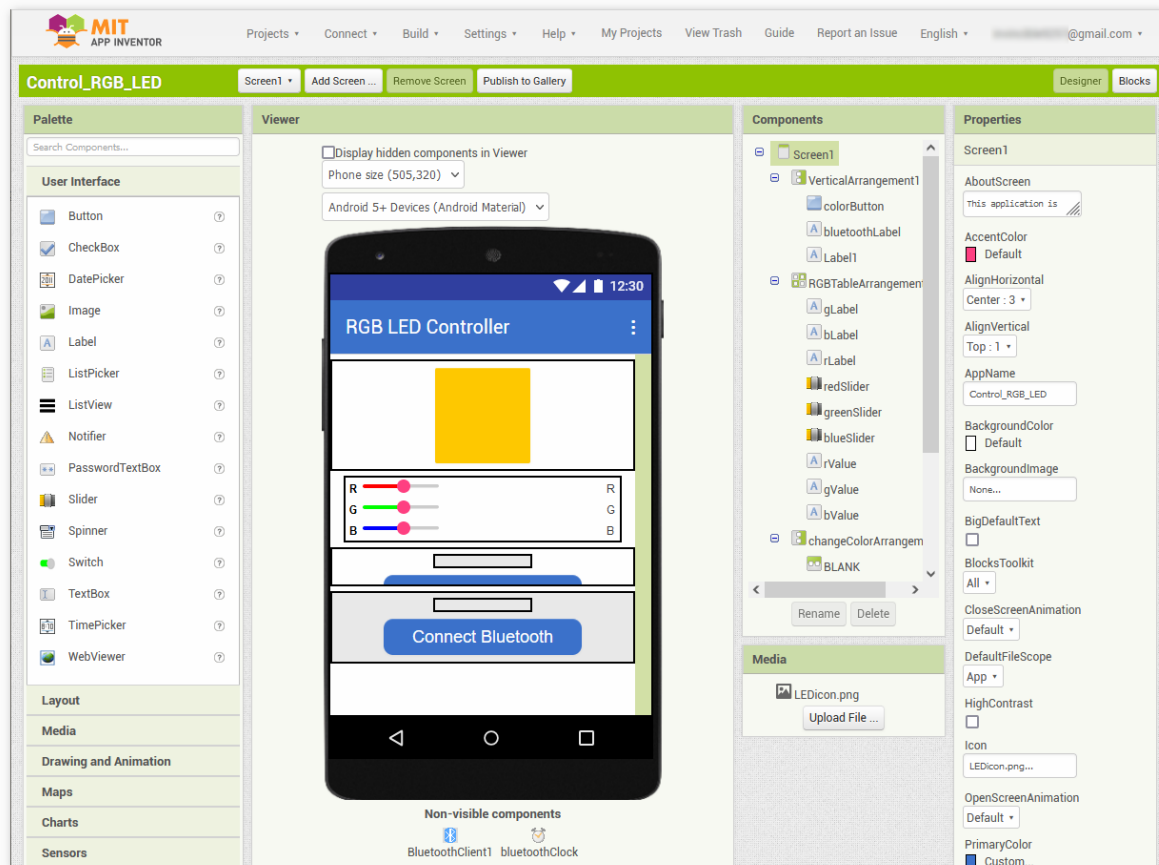


4. In MIT App Inventor, you have 2 primary sections: the **Designer** and the **Blocks**. You can switch between these two sections in the upper right corner of the page.

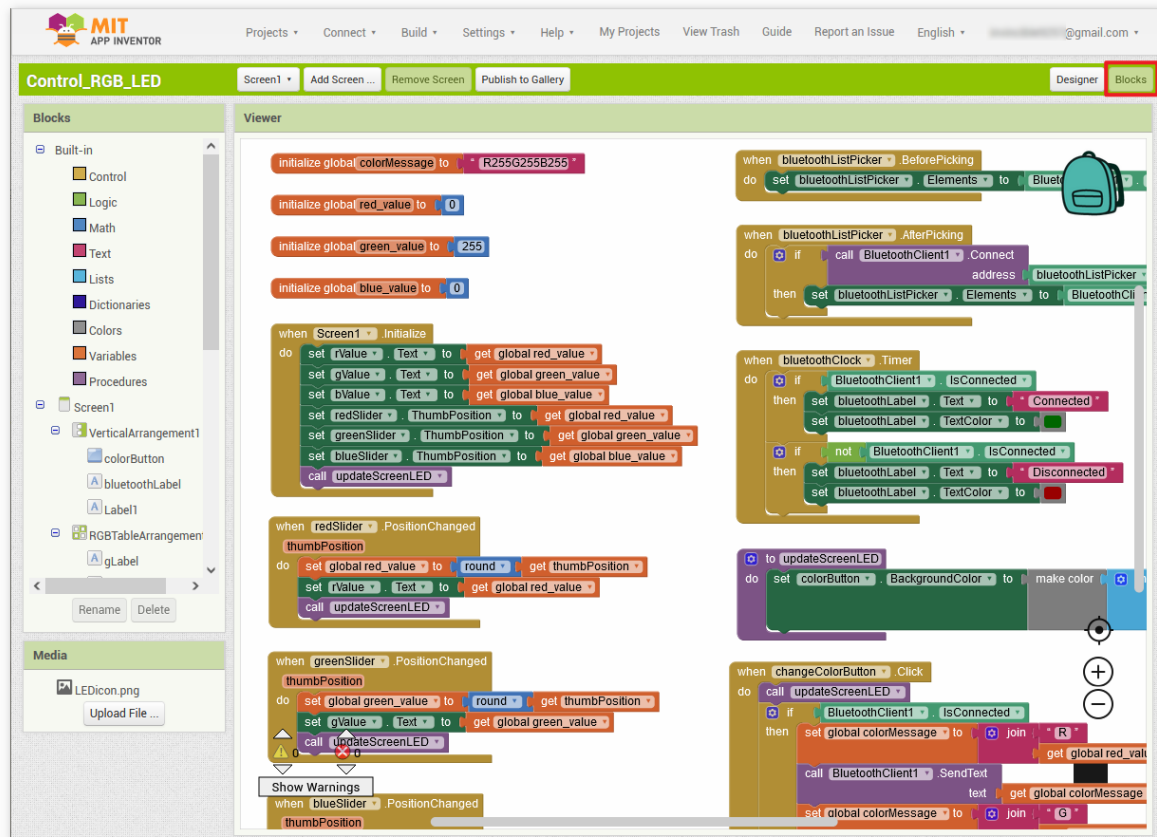


5. The **Designer** allows you to add buttons, text, screens, and modify the overall aesthetic of your application.

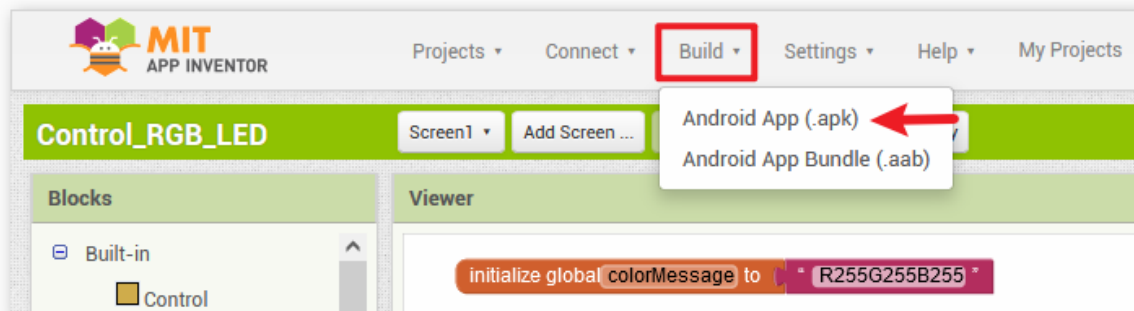




6. Next, there's the **Blocks** section. This section lets you craft custom functionalities for your app, allowing you to program each component on the app's GUI to achieve desired features.



7. To install the application on a smartphone, navigate to the **Build** tab.



- You can generate a .apk file. After selecting this option, a page will appear allowing you to choose between downloading a .apk file or scanning a QR code for installation. Follow the installation guide to complete the application installation.

You also download our pre-compiled APK here: [Control\\_RGB\\_LED.apk](#)

- If you wish to upload this app to Google Play or another app marketplace, you can generate a .aab file.

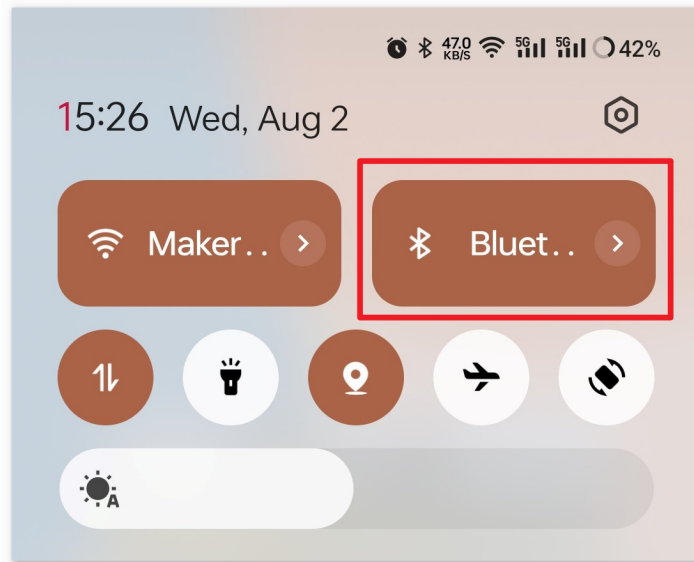
### 3. Upload the Code

1. Open the 04-Bluetooth\_RGB\_controller.ino file under the path of ultimate-sensor-kit\iot\_project\bluetooth\04-Bluetooth\_RGB\_controller, or copy this code into **Arduino IDE**.
2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

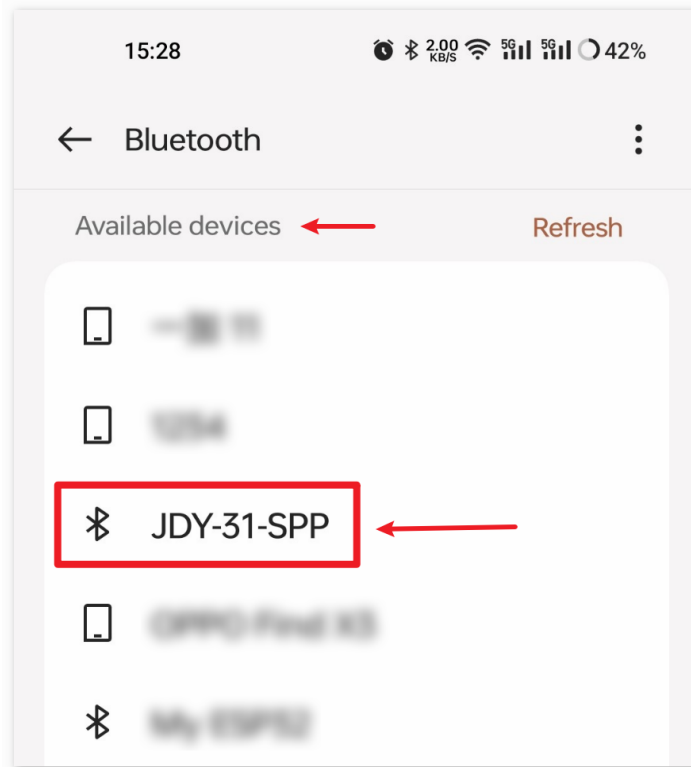
### 4. App and Bluetooth module Connection

Ensure that the application created earlier is installed on your smartphone.

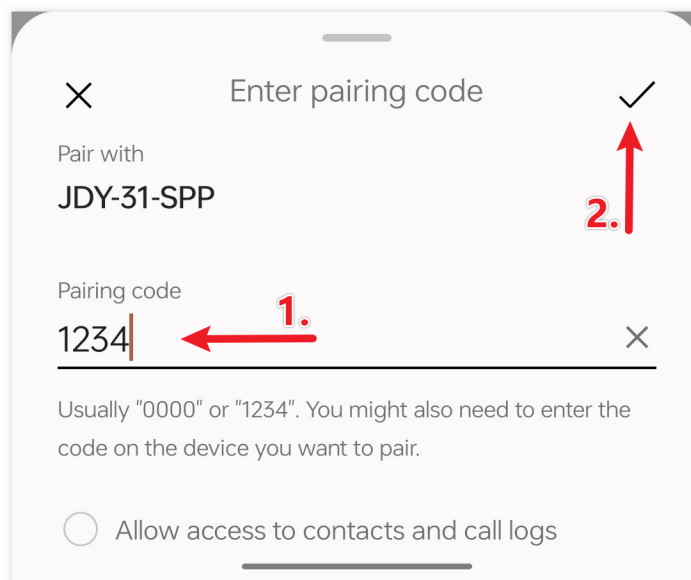
1. Initially, turn on **Bluetooth** on your smartphone.



2. Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.



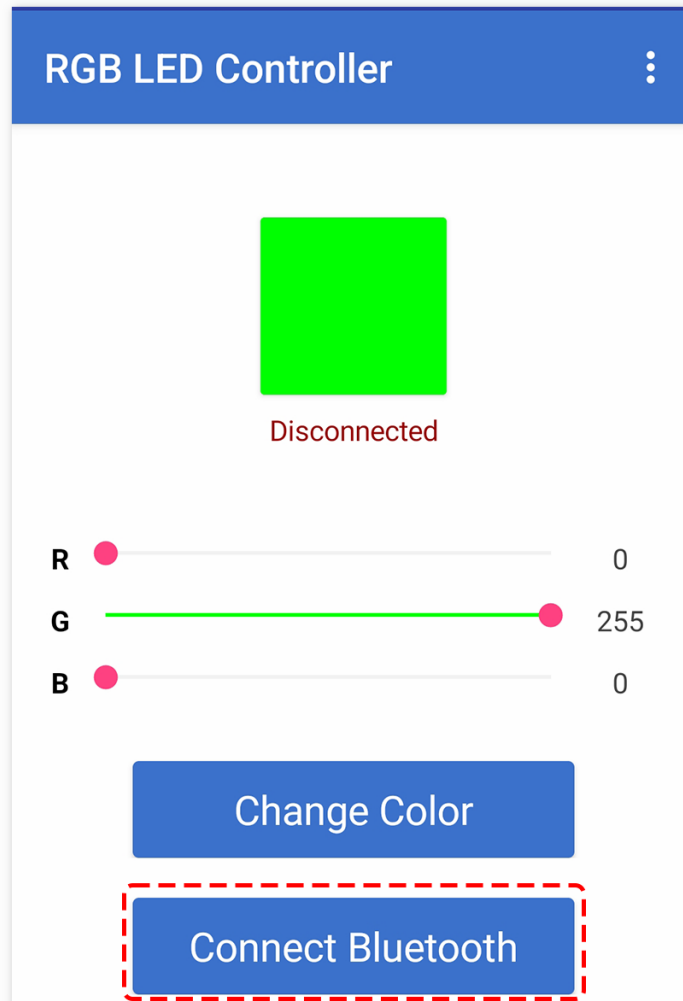
3. After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



4. Now open the newly installed **Control\_RGB\_LED** APP.



5. In the APP, click on **Connect Bluetooth** to establish a connection between the APP and Bluetooth module.

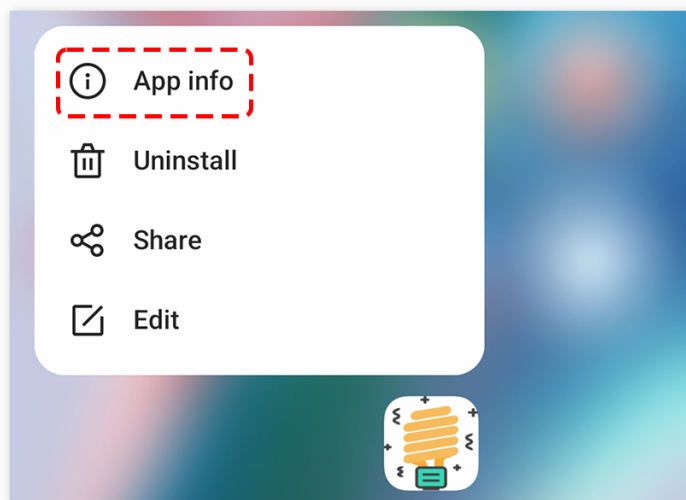


6. This page displays a list of all paired Bluetooth devices. Choose the `xx.xx.xx.xx.xx.xx JDY-31-SPP` option from the list. The name of each device is listed next to its MAC address.

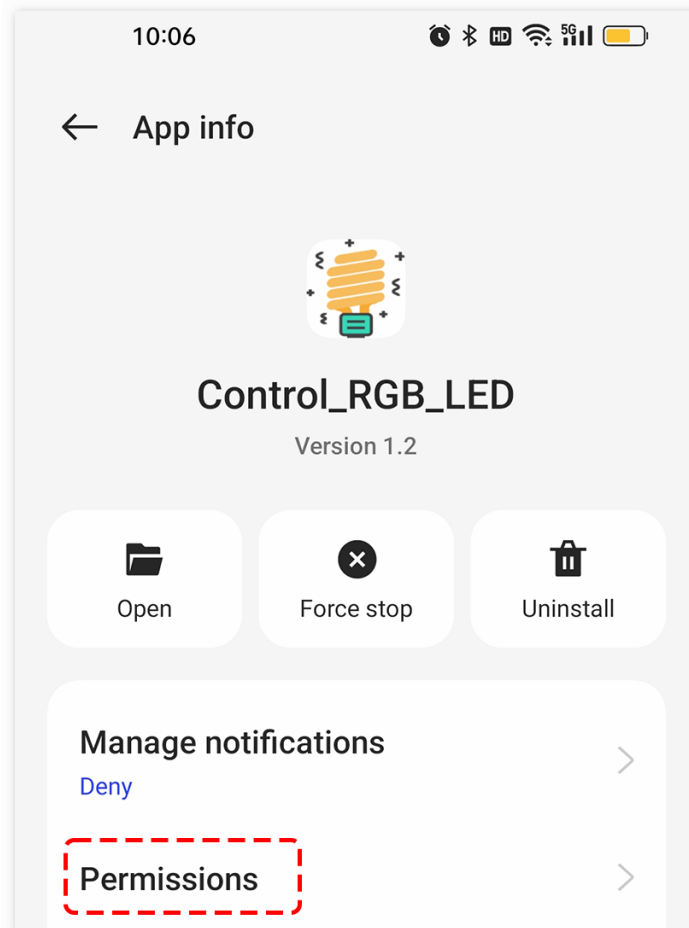


7. If you don't see any devices on the page shown above, it could be because this app is not authorized to scan for nearby devices. In such a case, you will need to adjust the settings manually.

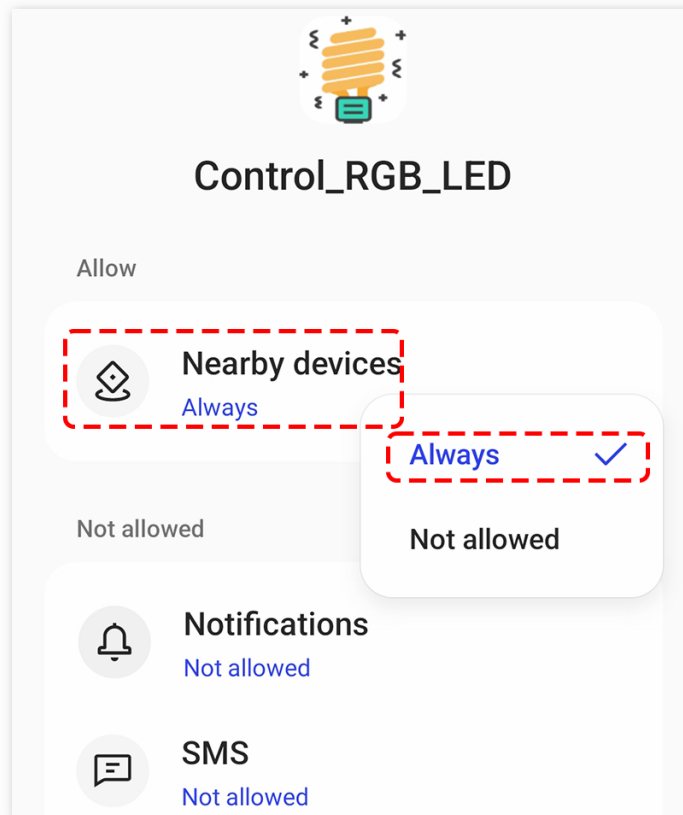
- To access the **APP Info** page, long-press the app icon and select it. Alternatively, if you have another method to reach this page, use that instead.



- Navigate to the **Permissions** page.

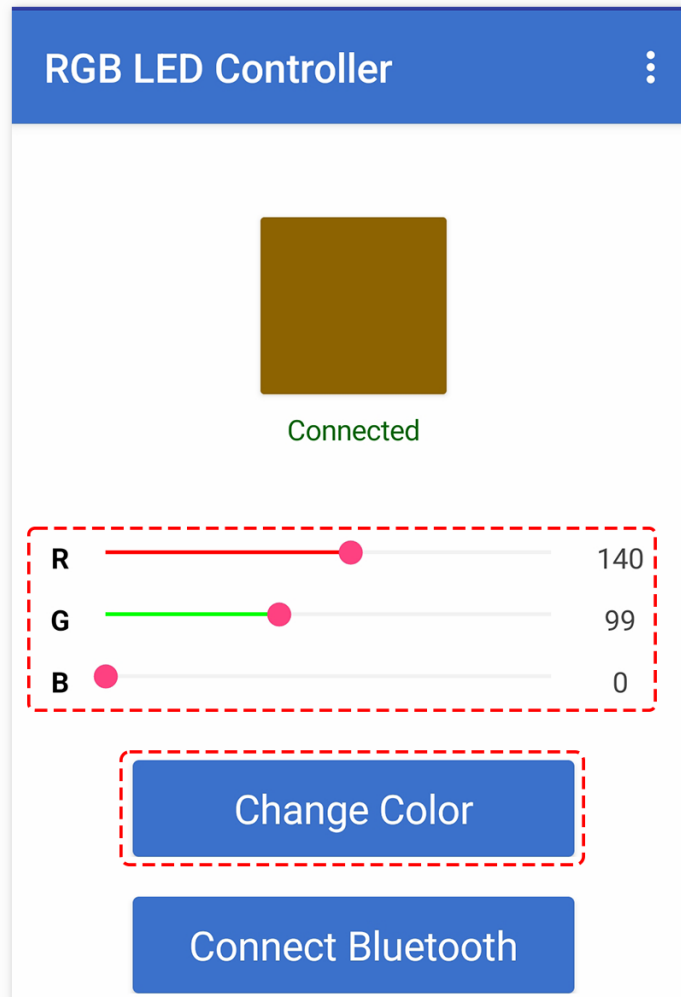


- To enable the APP to scan for nearby devices, go to **Nearby devices** and select **Always**.



- Now, restart the APP and repeat steps 5 and 6 to successfully connect to Bluetooth.
8. After successfully connecting, you will be redirected to the main page where it will show “connected”. From there, you can easily modify the RGB values and alter the color of the display by clicking on the **Change Color** button.





## 5. Code explanation

1. Setting up the Bluetooth module and initializing variables:

The code begins by including the SoftwareSerial library and initializing the necessary variables.

```
#include <SoftwareSerial.h>
SoftwareSerial bleSerial(3, 4); //Rx,Tx

#define max_char 12
char message[max_char];
char r_char;
byte currentIndex = 0;

const int redPin = 9;
const int greenPin = 10;
const int bluePin = 11;

int redValue = 0;
int greenValue = 255;
```

(continues on next page)

(continued from previous page)

```
int blueValue = 0;

String redTempValue;
String greenTempValue;
String blueTempValue;

int flag = 0;
char currentColor;
```

## 2. setup() function:

Here, the RGB LED pins are set as output pins, and the serial communication is initialized with a baud rate of 9600 for both the Arduino's main serial and the Bluetooth module.

```
void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  Serial.begin(9600);
  bleSerial.begin(9600);
}
```

## 3. Reading and processing the data:

In the main loop, the code continuously checks for incoming data from the Bluetooth module. Upon receiving any data, it processes the characters to identify RGB values and sets the color of the RGB LED accordingly.

```
void loop() {
  while (bleSerial.available() > 0) {
    ... [data reading and processing]
  }

  if (flag == 0) {
    Serial.println(message);
    analogWrite(redPin, redTempValue.toInt());
    analogWrite(greenPin, greenTempValue.toInt());
    analogWrite(bluePin, blueTempValue.toInt());

    flag = 1;

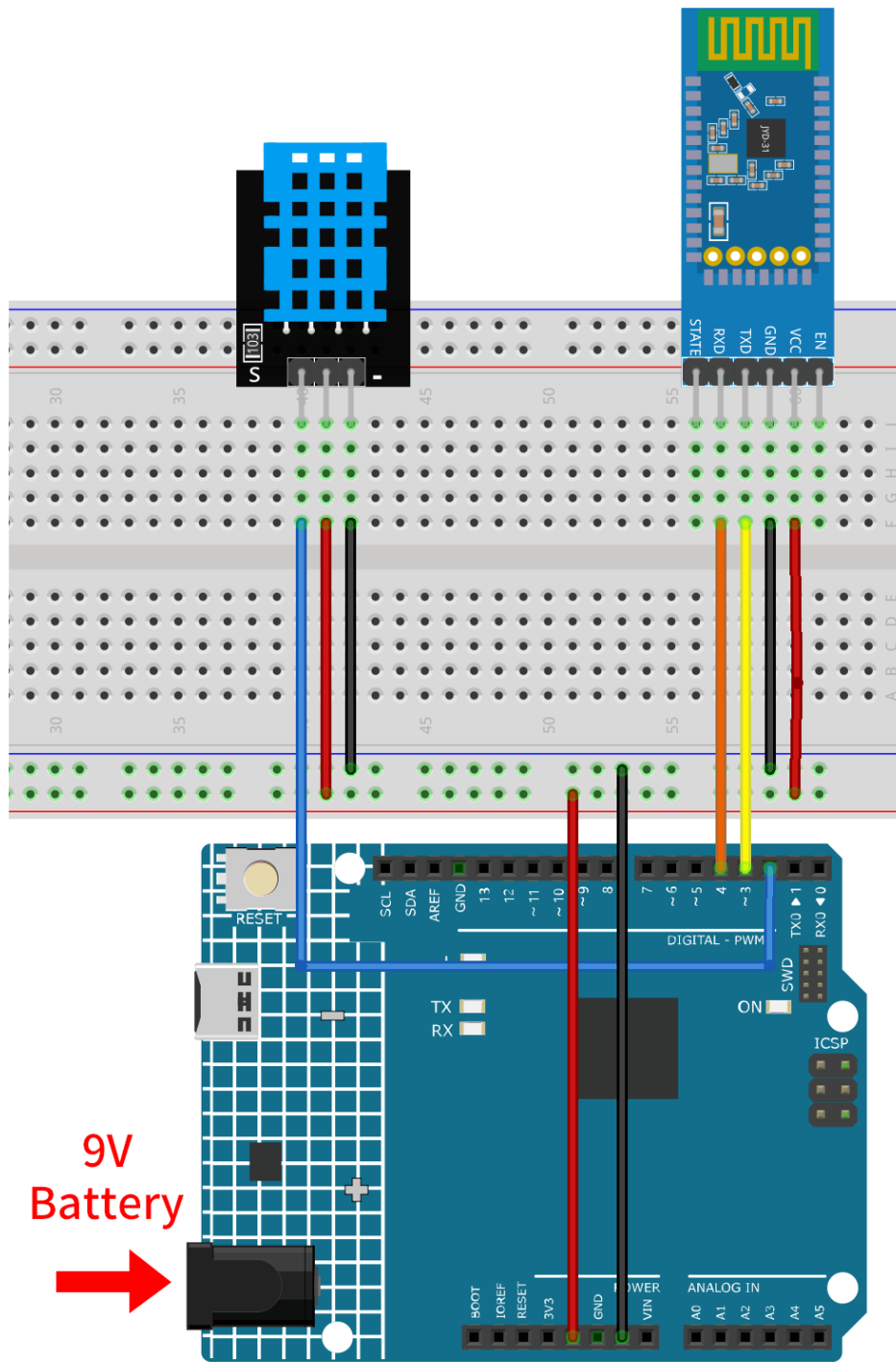
    for (int i = 0; i < 12; i++) {
      message[i] = '\0';
    }
    currentIndex = 0;
  }
}
```

### **2.5.14 Bluetooth Environmental Monitor**

This project uses an Android app created with MIT App Inventor to receive and display environmental data from an Arduino board. The Arduino board fetches data from a DHT11 sensor to measure temperature and humidity. Once the data is collected, it's transmitted over Bluetooth using the JDY-31 module. APP will display the data on the screen once it receives it.

The Android application will be constructed utilizing a complimentary web-based platform known as . The project presents an excellent opportunity to gain familiarity with the interfacing of an Arduino with a smartphone.

## 1. Build the Circuit



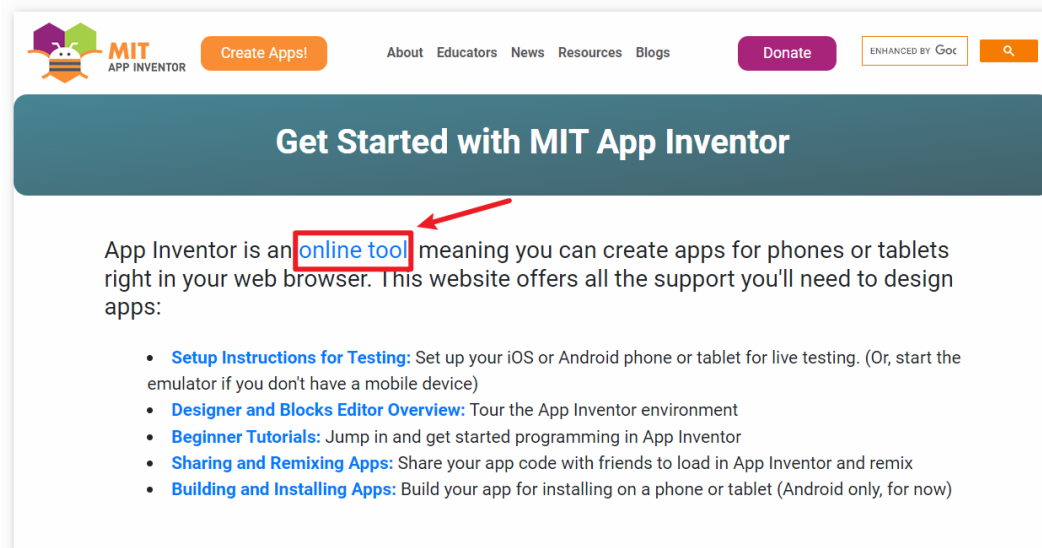
- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *Temperature and Humidity Sensor Module (DHT11)*

## 2. Create the Android App

The Android application will be developed using a free web application known as . MIT App Inventor serves as an excellent starting point for Android development, owing to its intuitive drag-and-drop features allowing for the creation of simplistic applications.

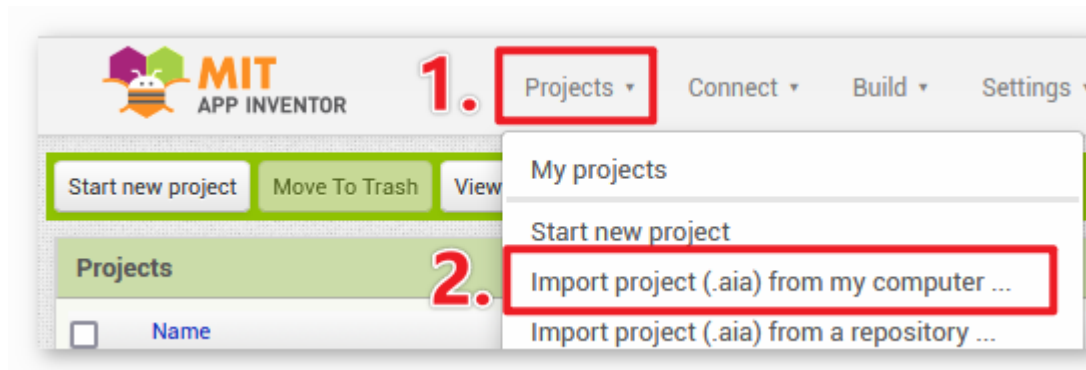
Now, let's begin.

1. Go to , and click “online tool” to login. You will require a Google account to register with MIT App Inventor.

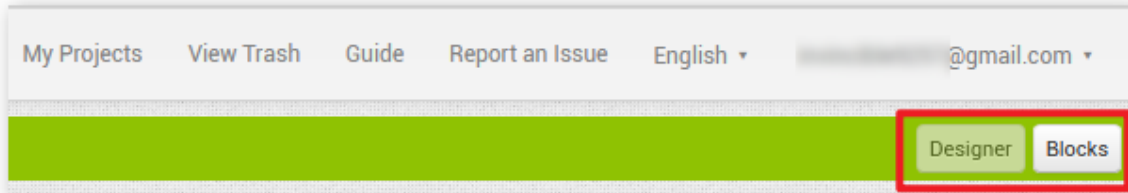


2. After logging in, navigate to **Projects -> Import project (.aia) from my computer**. Subsequently, upload the Bluetooth\_controlled\_lock.aia file located in the path ultimate-sensor-kit\iot\_project\bluetooth\05-Bluetooth\_environmental\_monitor.

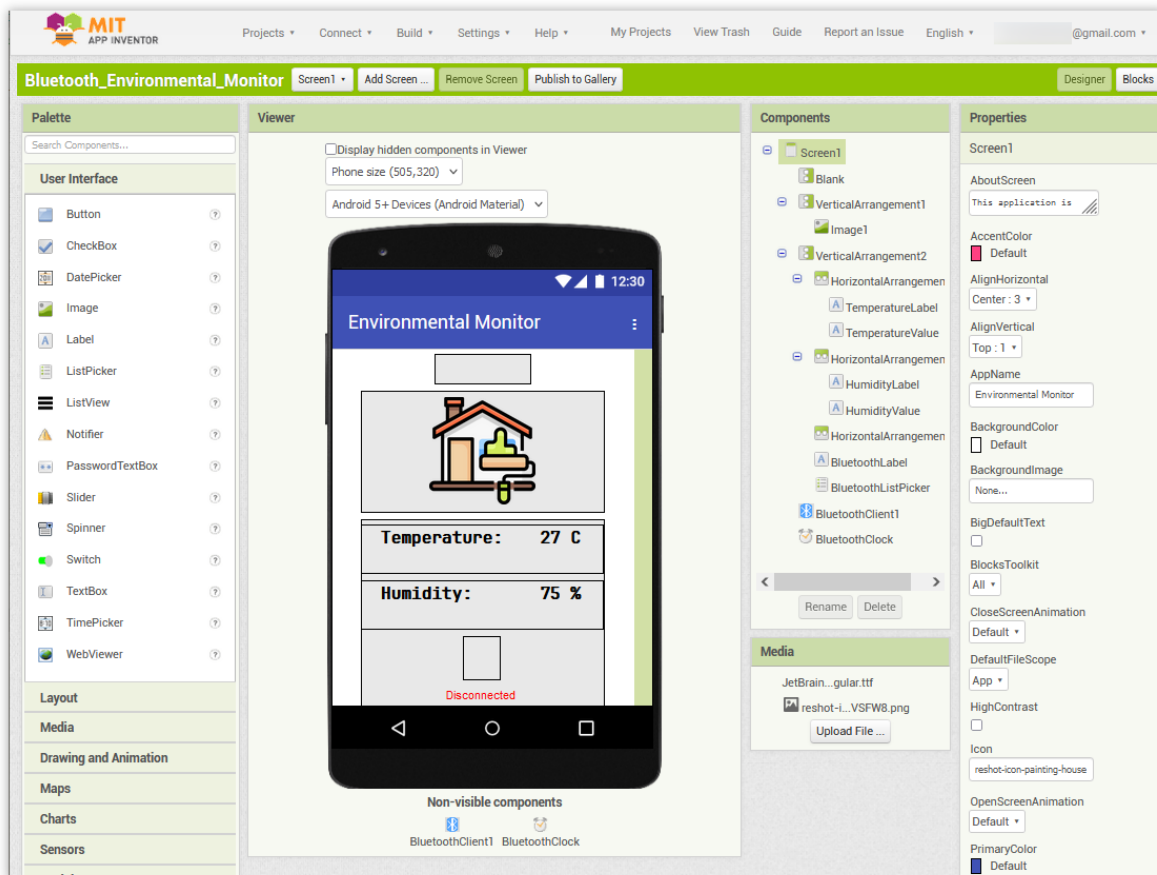
You can also directly download here: Bluetooth\_Environmental\_Monitor.aia



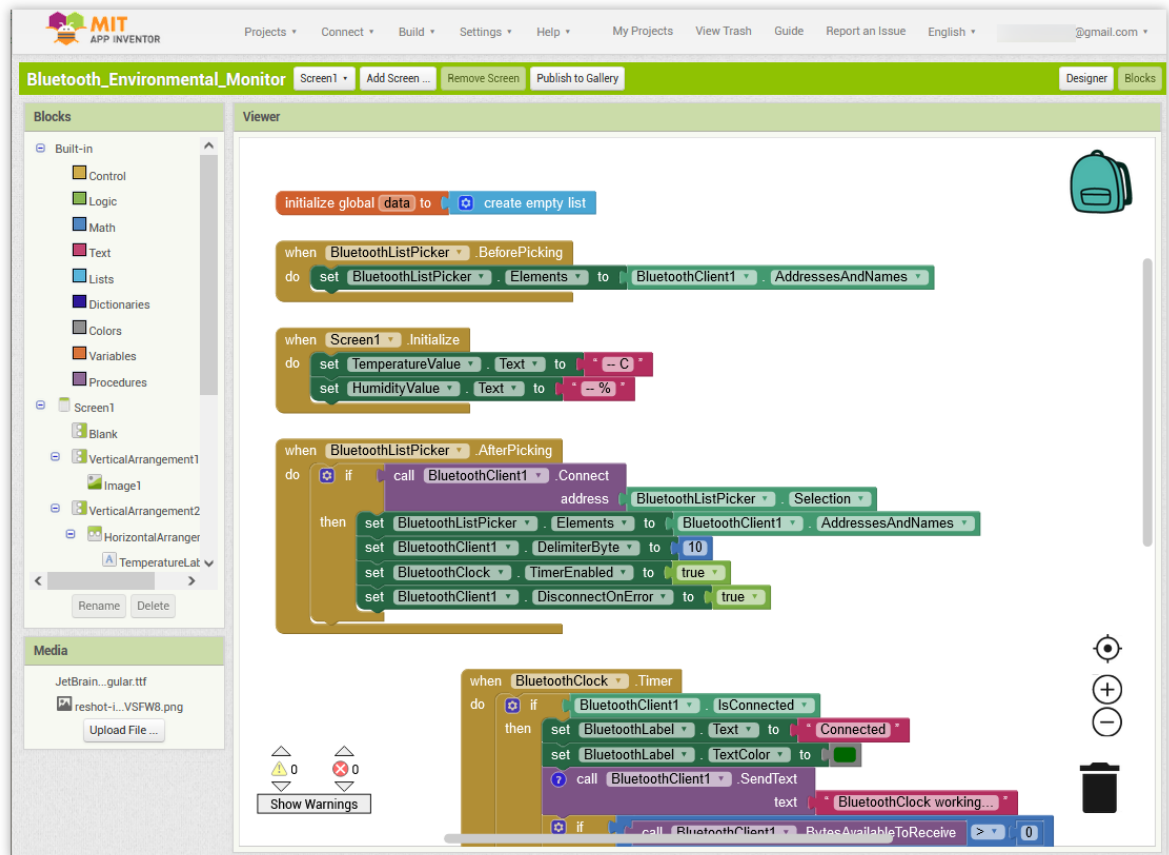
3. Upon uploading the .aia file, you will see the application on the MIT App Inventor software. This is a pre-configured template. You can modify this template after you have familiarized yourself with MIT App Inventor through the following steps.
4. In MIT App Inventor, you have 2 primary sections: the **Designer** and the **Blocks**. You can switch between these two sections in the upper right corner of the page.



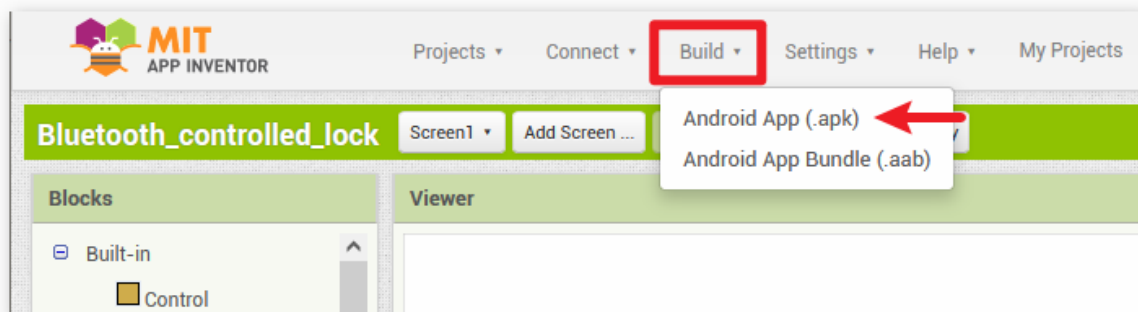
5. The **Designer** allows you to add buttons, text, screens, and modify the overall aesthetic of your application.



6. Next, there's the **Blocks** section. This section lets you craft custom functionalities for your app, allowing you to program each component on the app's GUI to achieve desired features.



7. To install the application on a smartphone, navigate to the **Build** tab.



- You can generate a .apk file. After selecting this option, a page will appear allowing you to choose between downloading a .apk file or scanning a QR code for installation. Follow the installation guide to complete the application installation.

You also download our pre-compiled APK here: [Bluetooth\\_Environmental\\_Monitor.apk](#)

- If you wish to upload this app to Google Play or another app marketplace, you can generate a .aab file.

### 3. Upload the Code

1. Open the 05-Bluetooth\_environmental\_monitor.ino file under the path of ultimate-sensor-kit\iot\_project\bluetooth\05-Bluetooth\_environmental\_monitor, or copy this code into **Arduino IDE**.

---

**Note:** To install the library, use the Arduino Library Manager and search for “**DHT sensor library**” and install it.

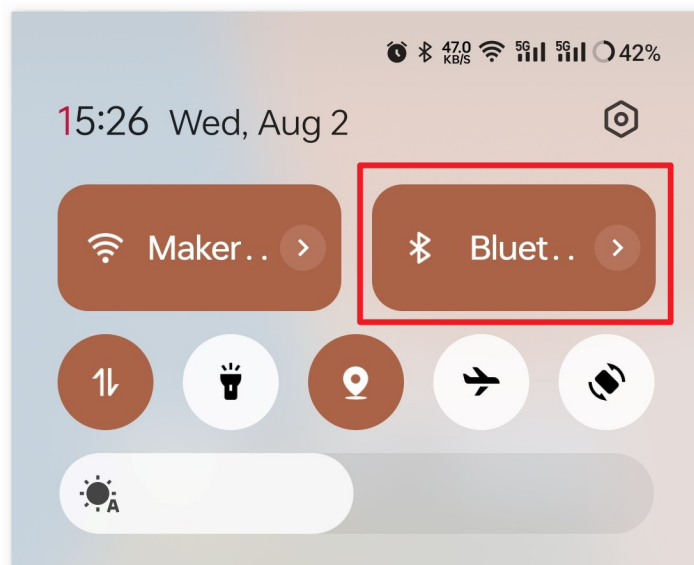
---

2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

### 4. App and Bluetooth module Connection

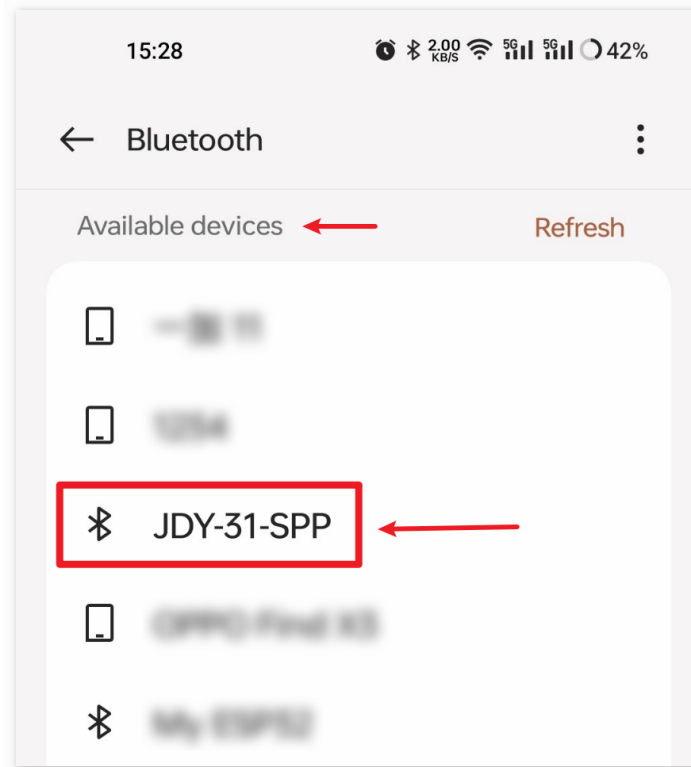
Ensure that the application created earlier is installed on your smartphone.

1. Initially, turn on **Bluetooth** on your smartphone.

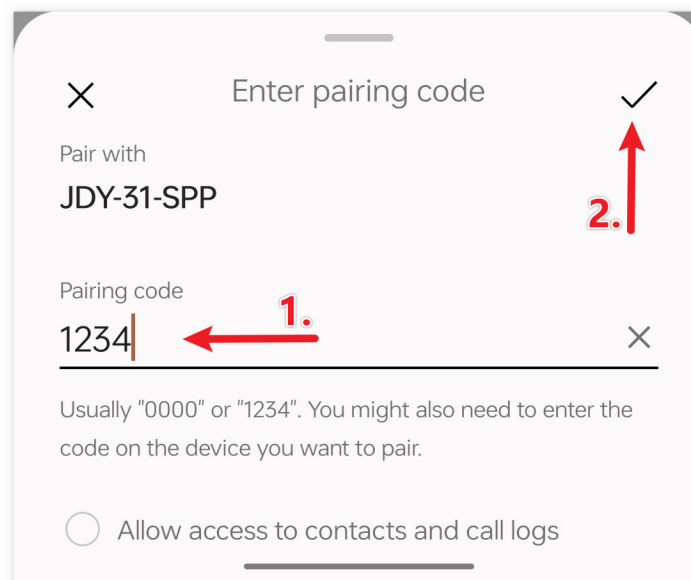


2. Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.

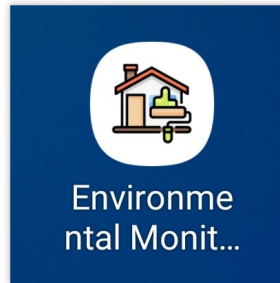




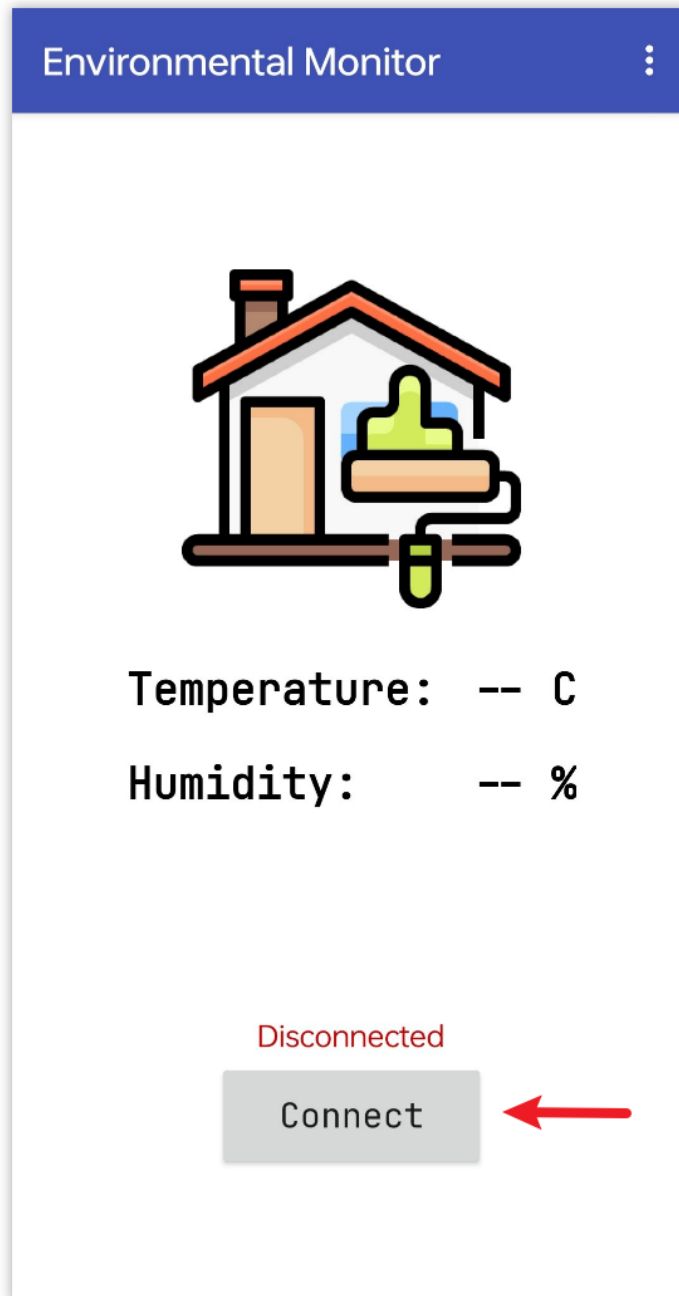
3. After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



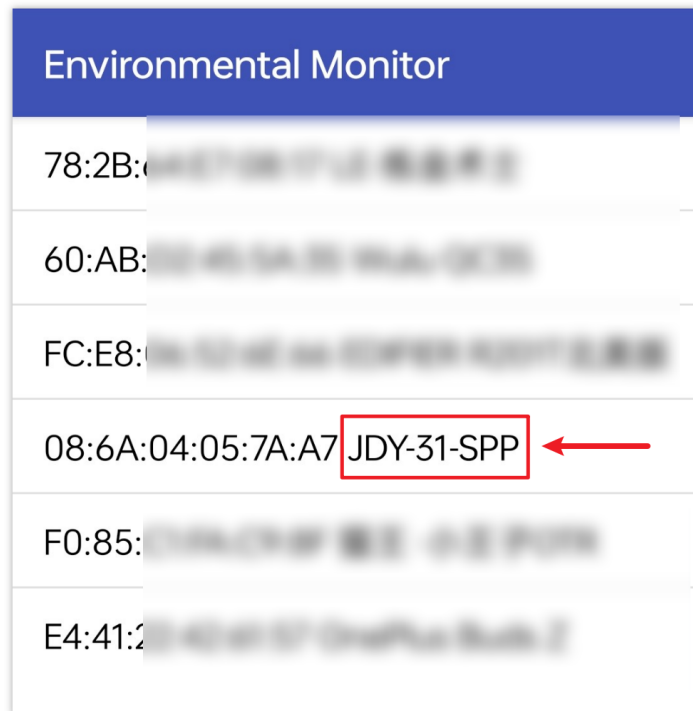
4. Now open the newly installed **Environmental Monitor** APP.



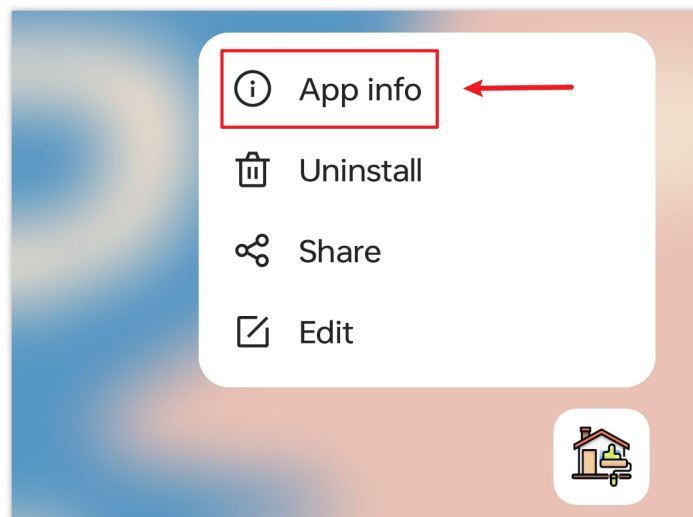
5. In the APP, click on **Connect** button to establish a connection between the APP and Bluetooth module.



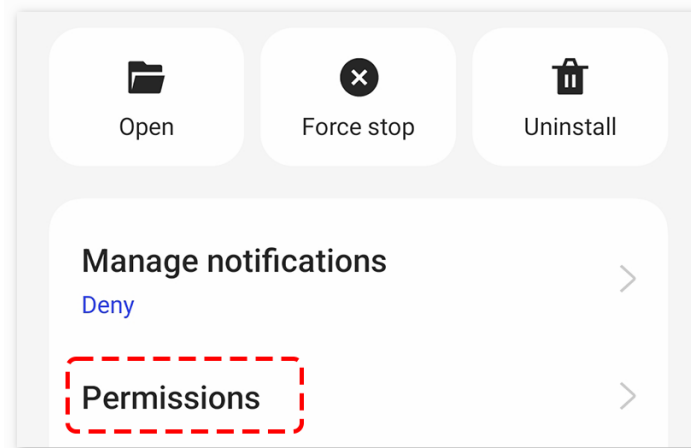
6. This page displays a list of all paired Bluetooth devices. Choose the `xx.xx.xx.xx.xx.xx JDY-31-SPP` option from the list. The name of each device is listed next to its MAC address.



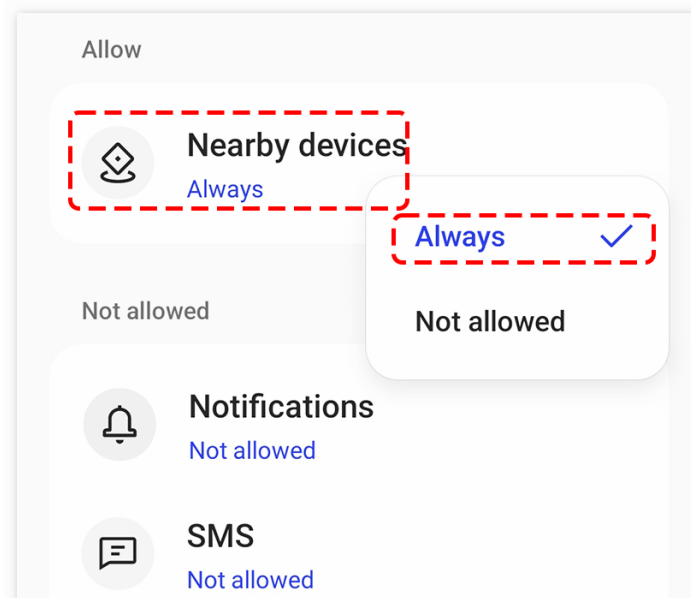
7. If you don't see any devices on the page shown above, it could be because this app is not authorized to scan for nearby devices. In such a case, you will need to adjust the settings manually.
- To access the **APP Info** page, long-press the app icon and select it. Alternatively, if you have another method to reach this page, use that instead.



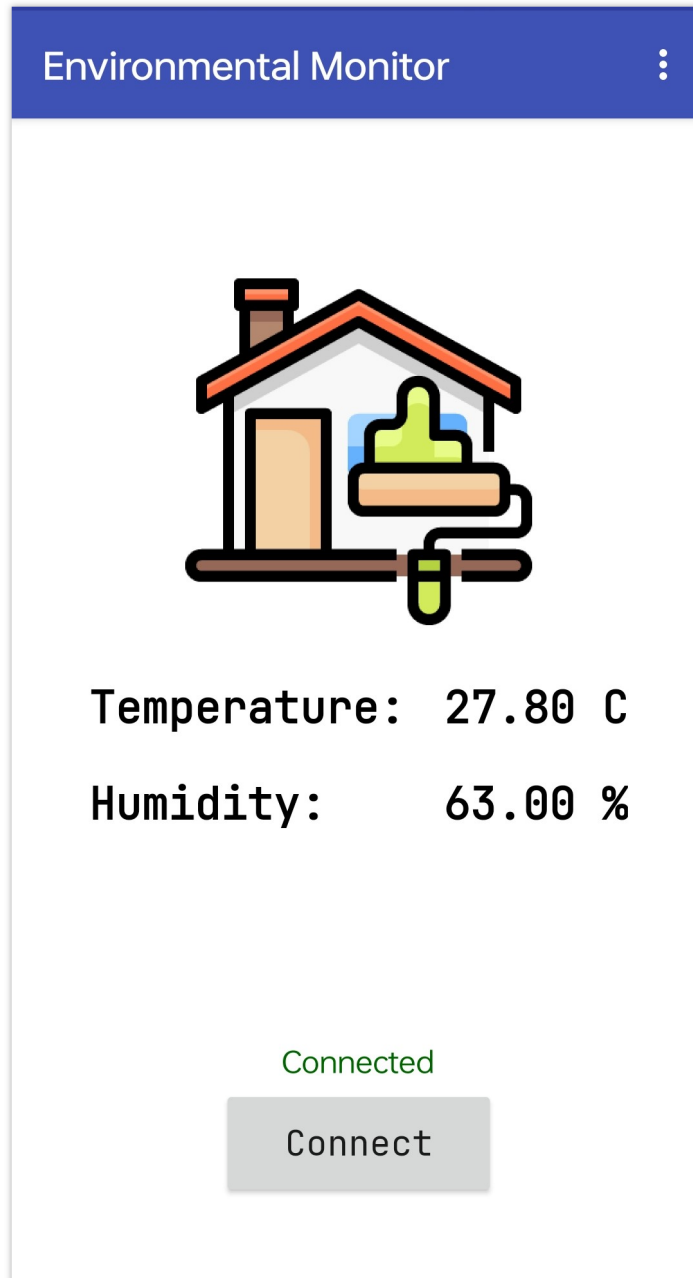
- Navigate to the **Permissions** page.



- To enable the APP to scan for nearby devices, go to **Nearby devices** and select **Always**.



- Now, restart the APP and repeat steps 5 and 6 to successfully connect to Bluetooth.
8. After a successful connection, you will be redirected to the main page where it will display temperature and humidity.



## 5. Code explanation

1. Setting up Bluetooth communication and DHT11 sensor.

```
#include <SoftwareSerial.h>
const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);

#include <DHT.h>
#define DHTPIN 2
```

(continues on next page)

(continued from previous page)

```
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

The code includes necessary libraries and defines pins for the Bluetooth module and DHT11 sensor. It also declares objects for Bluetooth communication and DHT11.

## 2. Initialization in setup function.

```
void setup() {
  Serial.begin(9600);
  bleSerial.begin(9600);
  dht.begin();
}
```

This segment initializes serial communication for debugging, Bluetooth module, and the DHT sensor.

## 3. Reading data and sending via Bluetooth.

```
void loop() {

  delay(2000);
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

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

  // For debug
  // Print the humidity and temperature to the serial monitor
  Serial.print(F("Humidity: "));
  Serial.print(humidity);
  Serial.print(F("% Temperature: "));
  Serial.print(temperature);
  Serial.println(F("°C "));

  sensorData = String(temperature) + "," + String(humidity); // Concatenate
  → temperature and humidity values
  Serial.print("Data to send: ");
  Serial.println(sensorData);

  bleSerial.println(sensorData); // Send temperature and humidity values to the
  → Bluetooth module
}
```

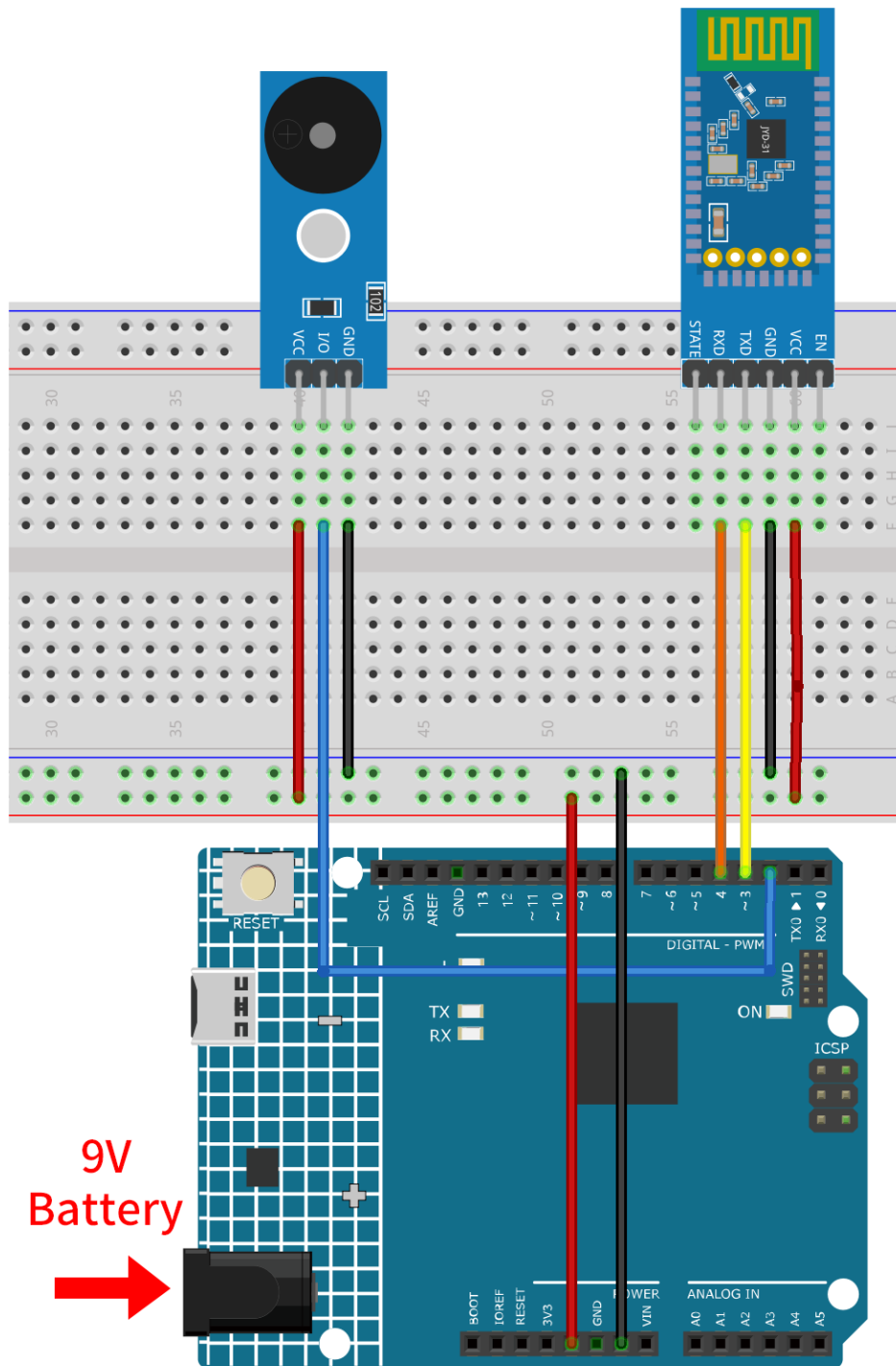
This segment reads temperature and humidity from the DHT11 sensor every 2 seconds. If the reading fails, it prints an error message. Otherwise, it prints the readings to the Serial Monitor and sends them via the Bluetooth module in a comma-separated format. When the app receives data in the format of “temperature,humidity”, it will parse the information and present it on the user interface.

### 2.5.15 Bluetooth Piano

This project uses an Android app created with MIT App Inventor to enable a straightforward “piano” feature by utilizing a JDY-31 Bluetooth module and a Passive Buzzer Module. The Bluetooth piano project allows users to play different musical notes on a passive buzzer module using a JDY-31 Bluetooth module. By sending specific note instructions via Bluetooth to the Arduino, users can generate corresponding tones on the buzzer.

The Android application will be constructed utilizing a complimentary web-based platform known as . The project presents an excellent opportunity to gain familiarity with the interfacing of an Arduino with a smartphone.

## 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *Passive Buzzer Module*

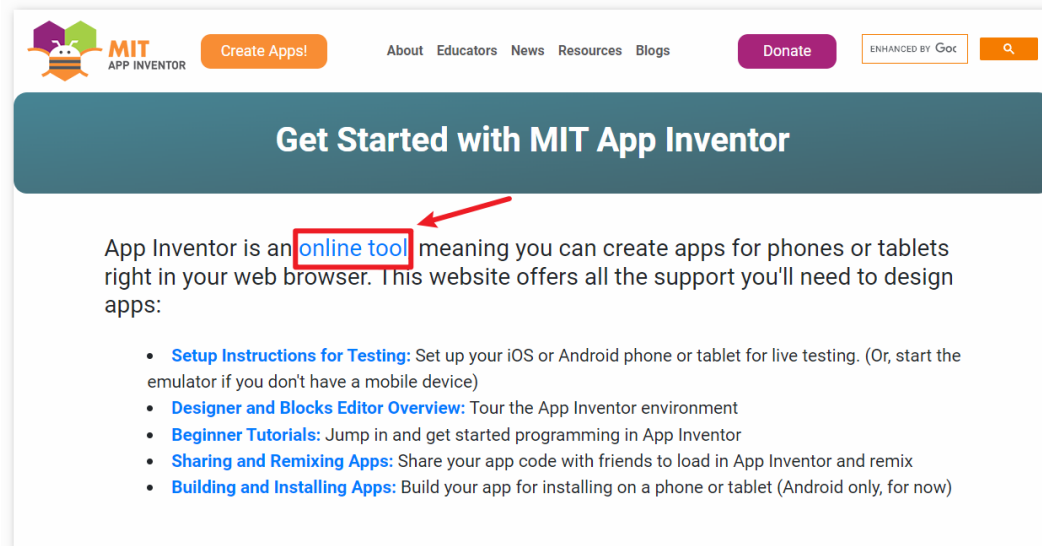


## 2. Create the Android App

The Android application will be developed using a free web application known as . MIT App Inventor serves as an excellent starting point for Android development, owing to its intuitive drag-and-drop features allowing for the creation of simplistic applications.

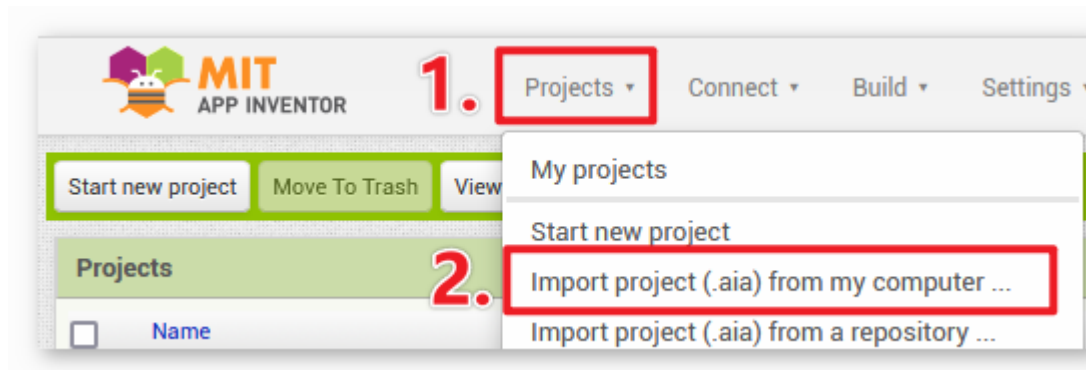
Now, let's begin.

1. Go to [MITAppInventor.org](https://MITAppInventor.org), and click “online tool” to login. You will require a Google account to register with MIT App Inventor.

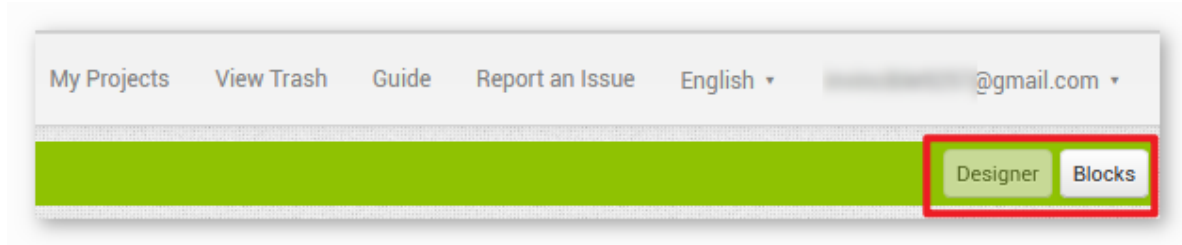


2. After logging in, navigate to **Projects** -> **Import project (.aia) from my computer**. Subsequently, upload the piano.aia file located in the path `ultimate-sensor-kit\iot_project\bluetooth\06-Bluetooth_piano`.

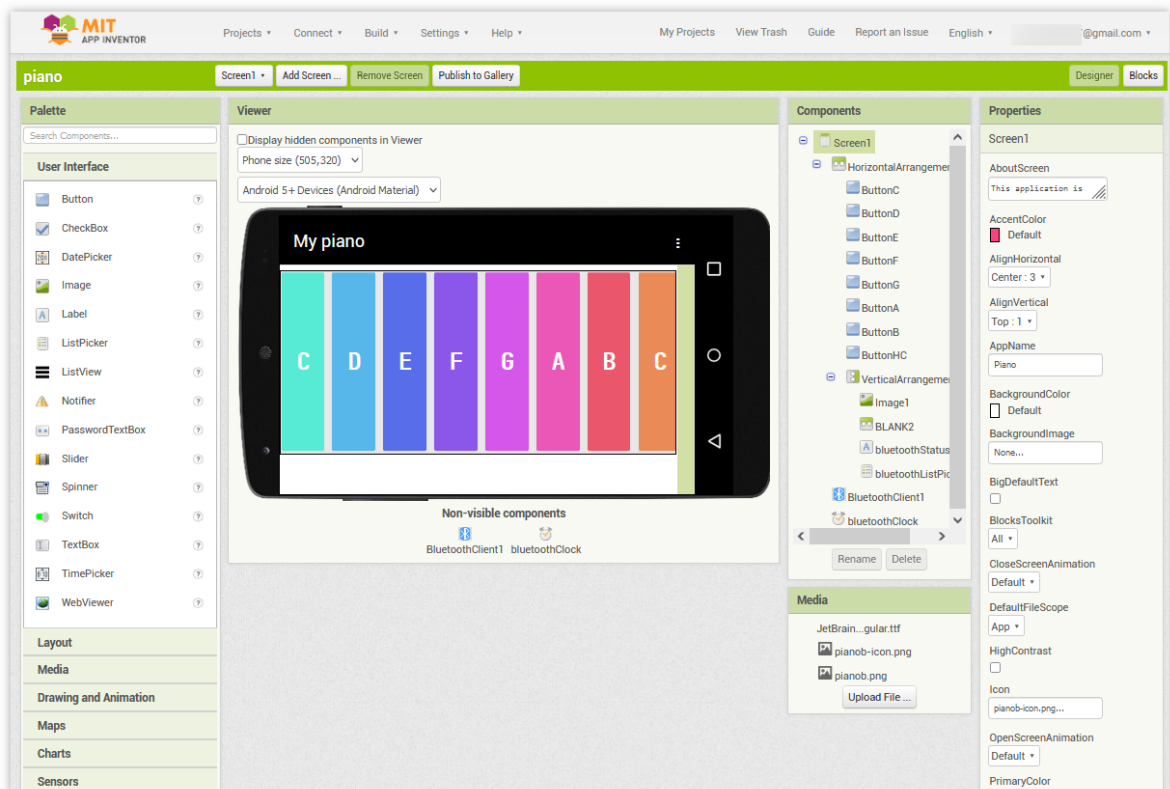
You can also directly download here: [piano.aia](#)



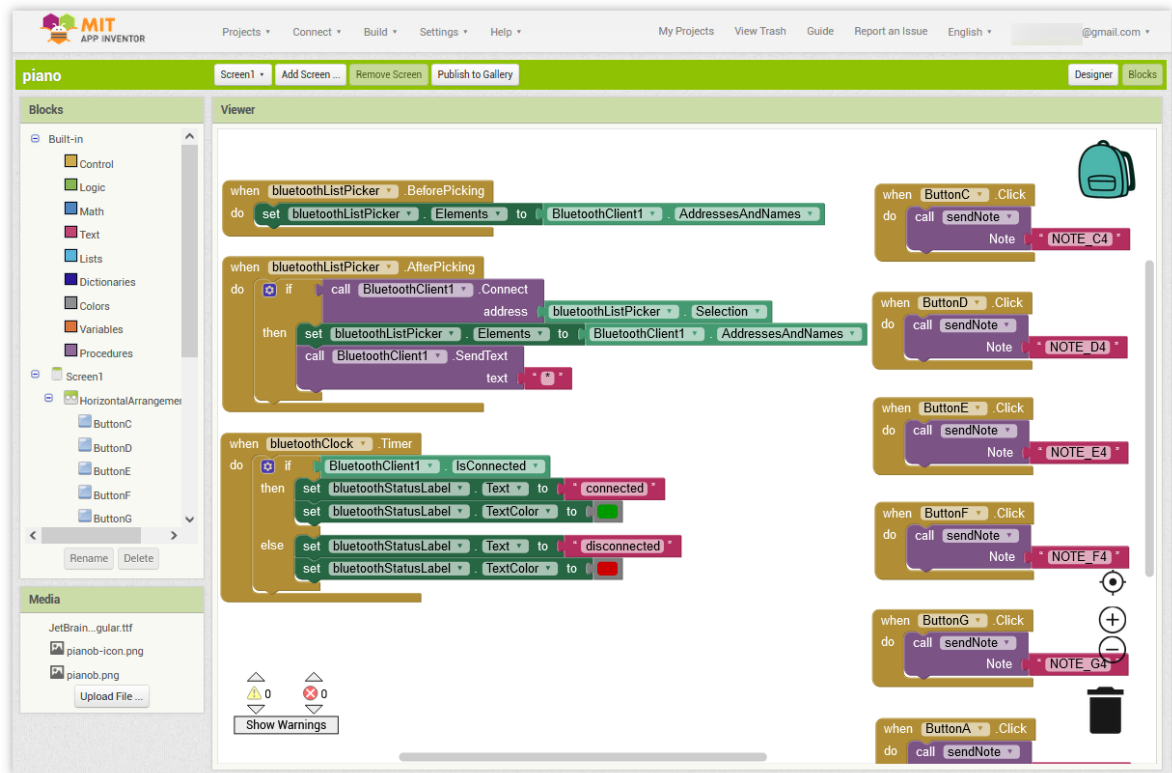
3. Upon uploading the .aia file, you will see the application on the MIT App Inventor software. This is a pre-configured template. You can modify this template after you have familiarized yourself with MIT App Inventor through the following steps.
4. In MIT App Inventor, you have 2 primary sections: the **Designer** and the **Blocks**. You can switch between these two sections in the upper right corner of the page.



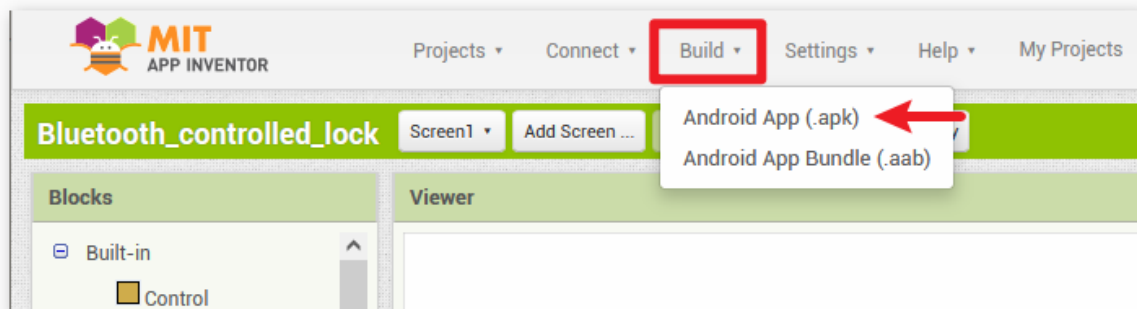
5. The **Designer** allows you to add buttons, text, screens, and modify the overall aesthetic of your application.



6. Next, there's the **Blocks** section. This section lets you craft custom functionalities for your app, allowing you to program each component on the app's GUI to achieve desired features.



7. To install the application on a smartphone, navigate to the **Build** tab.



- You can generate a .apk file. After selecting this option, a page will appear allowing you to choose between downloading a .apk file or scanning a QR code for installation. Follow the installation guide to complete the application installation.

You also download our pre-compiled APK here: `piano.apk`

- If you wish to upload this app to Google Play or another app marketplace, you can generate a .aab file.

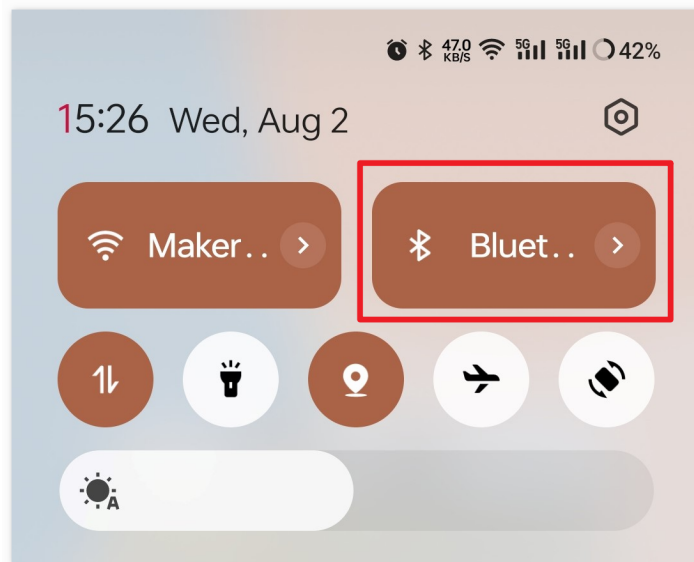
### 3. Upload the Code

1. Open the 06-Bluetooth\_piano.ino file under the path of ultimate-sensor-kit\iot\_project\bluetooth\06-Bluetooth\_piano, or copy this code into **Arduino IDE**.
2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

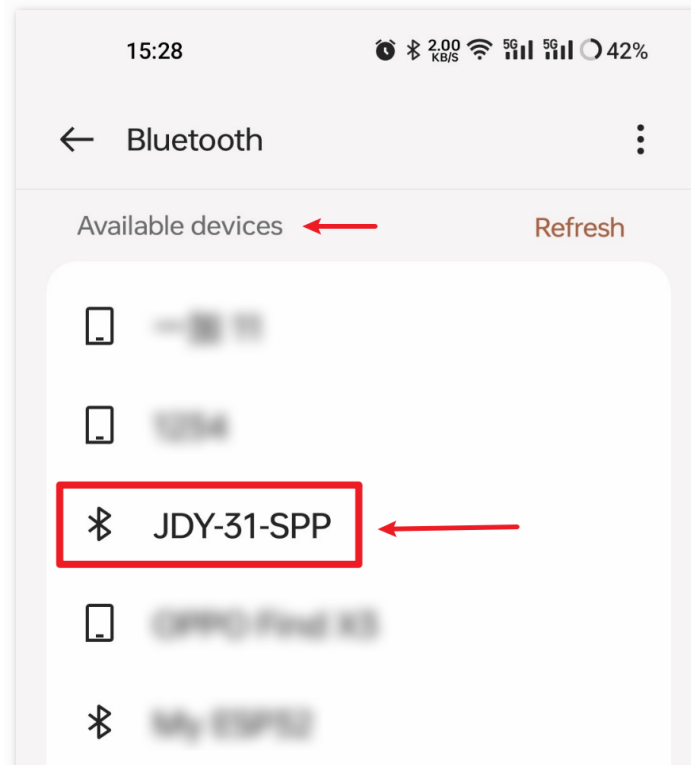
### 4. App and Bluetooth module Connection

Ensure that the application created earlier is installed on your smartphone.

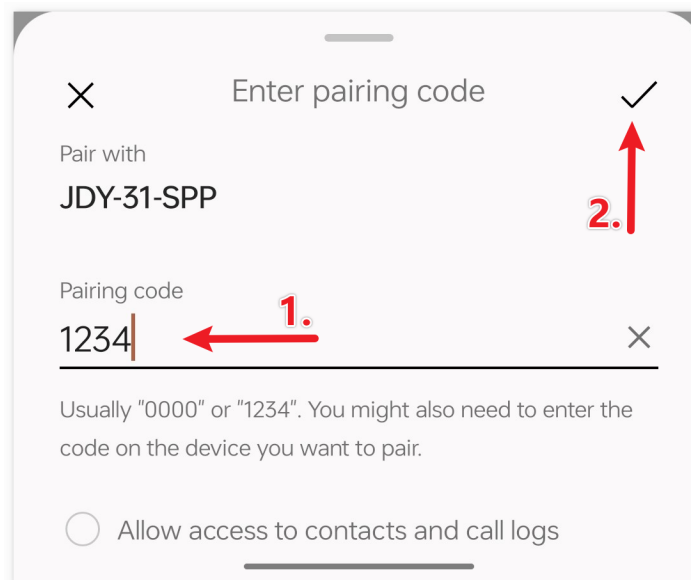
1. Initially, turn on **Bluetooth** on your smartphone.



2. Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.



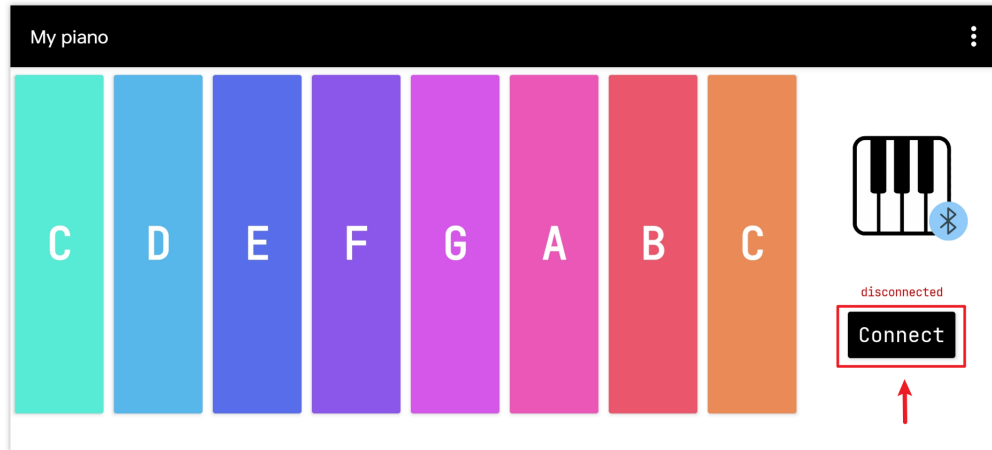
3. After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter "1234".



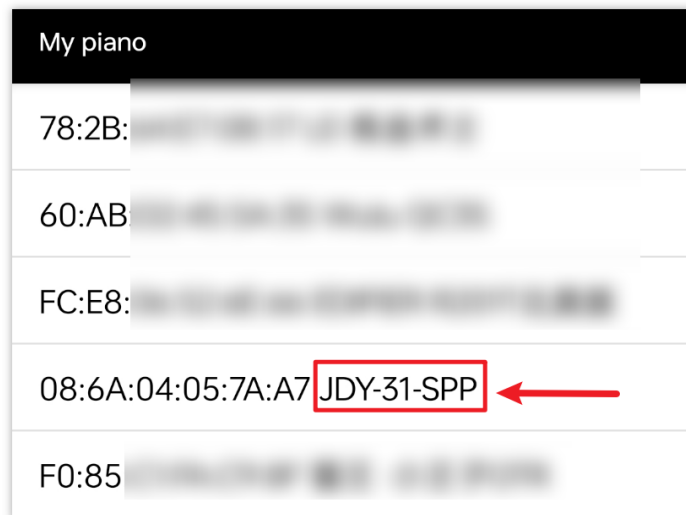
4. Now open the newly installed **Piano APP**.



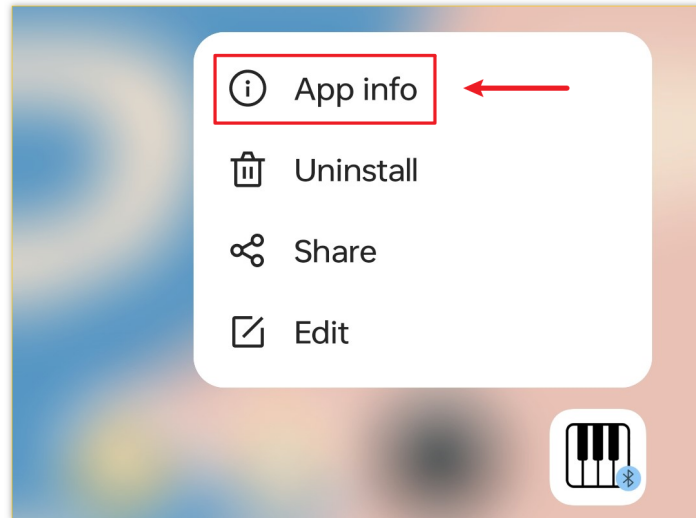
5. In the APP, click on **Connect** button to establish a connection between the APP and Bluetooth module.



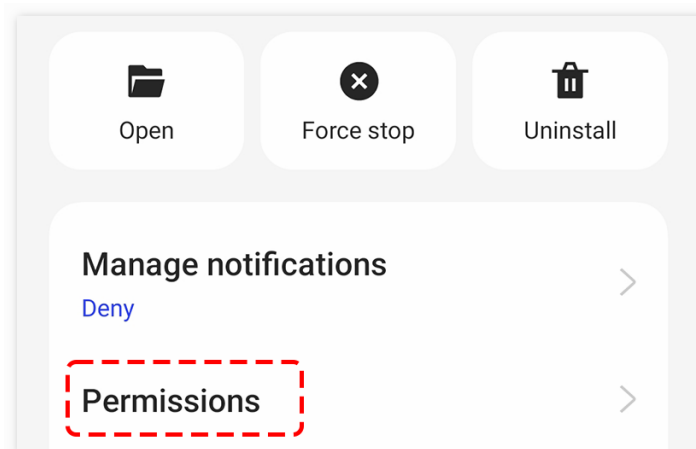
6. This page displays a list of all paired Bluetooth devices. Choose the `xx.xx.xx.xx.xx.xx JDY-31-SPP` option from the list. The name of each device is listed next to its MAC address.



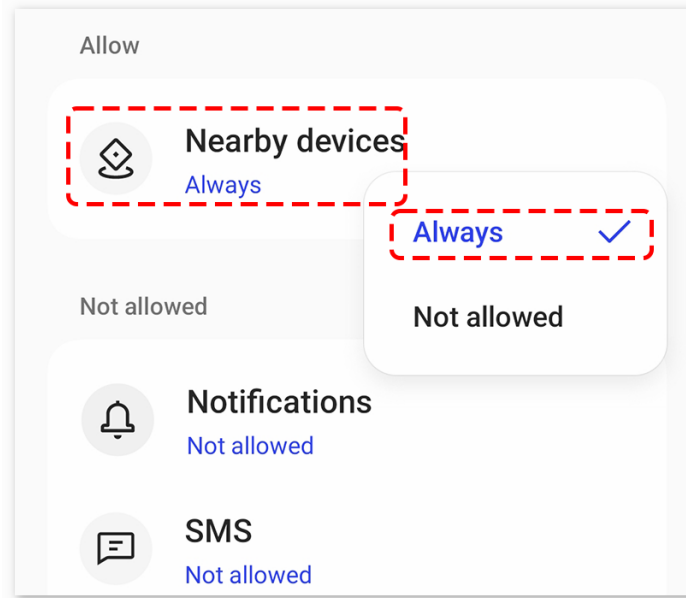
7. If you don't see any devices on the page shown above, it could be because this app is not authorized to scan for nearby devices. In such a case, you will need to adjust the settings manually.
- To access the **APP Info** page, long-press the app icon and select it. Alternatively, if you have another method to reach this page, use that instead.



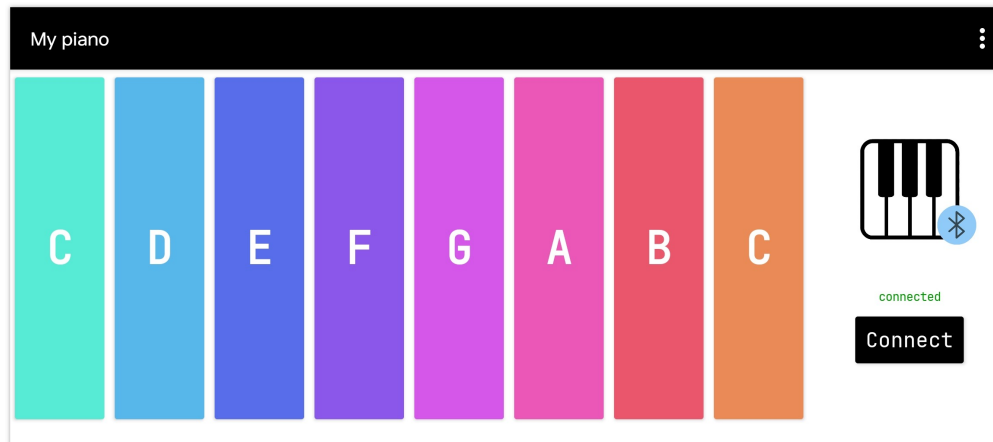
- Navigate to the **Permissions** page.



- To enable the APP to scan for nearby devices, go to **Nearby devices** and select **Always**.



- Now, restart the APP and repeat steps 5 and 6 to successfully connect to Bluetooth.
8. After a successful connection, you can click on the buttons in the app to play different notes, and even perform some simple songs.



## 5. Code explanation

### 1. Setting Up Libraries and Pins

```
#include "pitches.h"
#include <SoftwareSerial.h>
const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);
const int buzzerPin = 2;
```

- `pitches.h`: This file contains the frequency values for musical notes.

### 2. Variable Declarations for Storing Bluetooth Data



```
char character;
String noteType;
```

- `character`: Stores individual characters received from Bluetooth.
- `noteType`: Aggregates the characters to form the full note instruction.

### 3. Setup Function - Initializing Serial Communications

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

- Initializes serial communication at a baud rate of 9600.
- The standard `Serial` is for debugging while `bleSerial` is specifically for Bluetooth communication.

### 4. Main Loop - Reading Bluetooth Data and Playing Corresponding Notes

```
void loop() {
  while (bleSerial.available() > 0) {
    character = bleSerial.read();
    noteType = noteType + character;
    if (character == '*') {
      noteType = noteType.substring(0, noteType.length() - 1);
      Serial.println(noteType);
      if (noteType == "NOTE_C4") {
        tone(buzzerPin, NOTE_C4);
      } // ... other notes check similarly ...
      noteType = "";
      delay(200);
      noTone(buzzerPin);
    }
  }
}
```

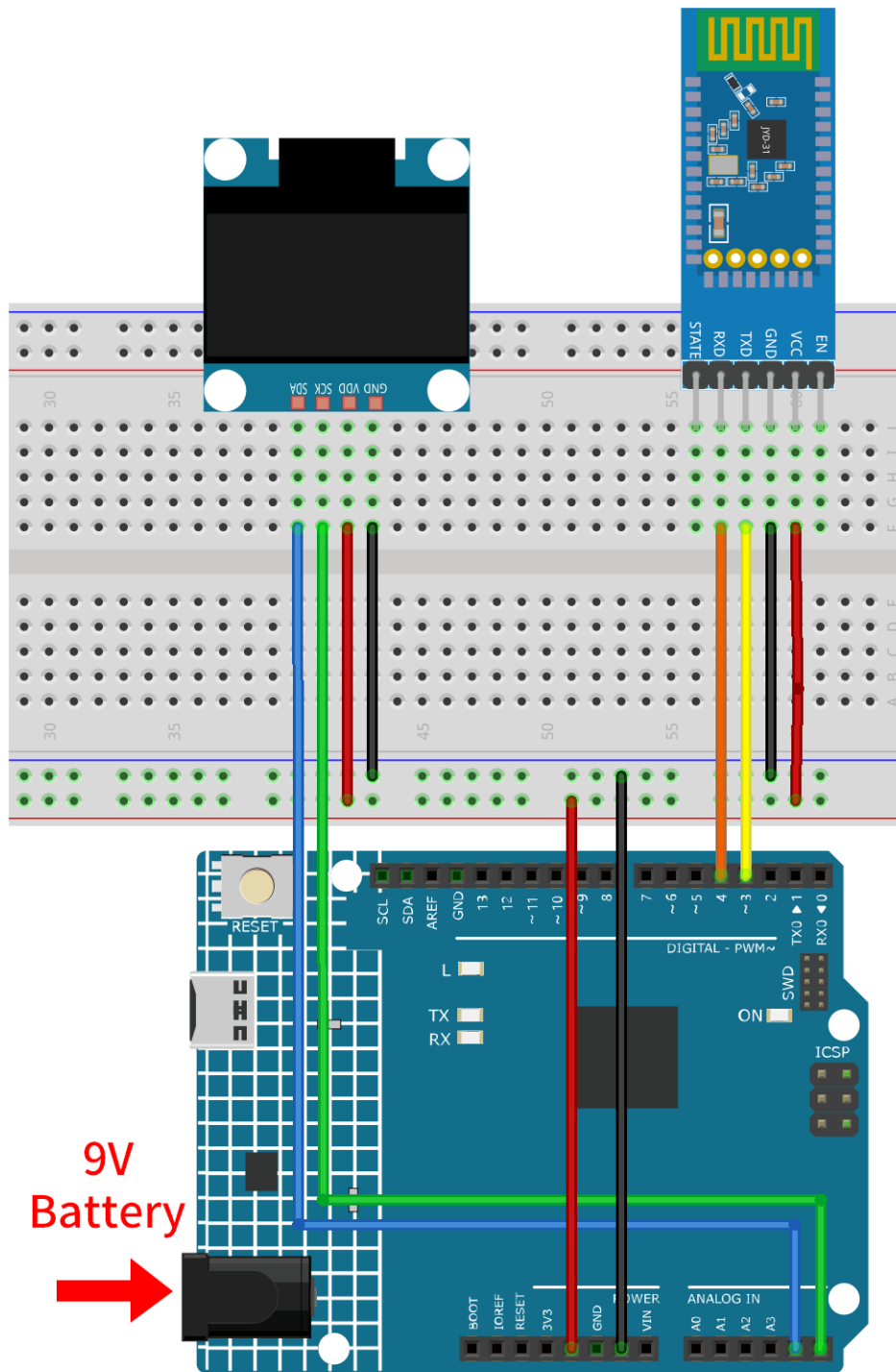
- Reads characters from Bluetooth and assembles the `noteType`.
- If an asterisk (\*) is detected, it indicates the end of the note instruction. The note is then played, followed by a short delay and then stopped.

## 2.5.16 Bluetooth OLED

This project uses an Android app created with MIT App Inventor to send messages over Bluetooth to an Arduino device. The Arduino, upon receiving the messages, displays them on an OLED screen. The Android app, designed with a user-friendly interface, allows users to input messages and send them with the press of a button.

The Android application will be constructed utilizing a complimentary web-based platform known as [MIT App Inventor](#). The project presents an excellent opportunity to gain familiarity with the interfacing of an Arduino with a smartphone.

## 1. Build the Circuit



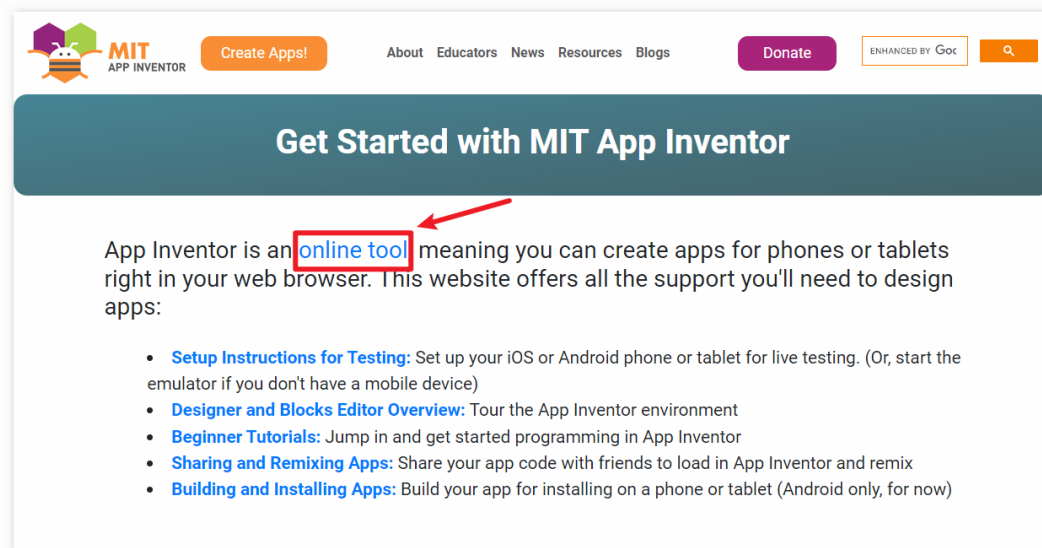
- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *OLED Display Module*

## 2. Create the Android App

The Android application will be developed using a free web application known as . MIT App Inventor serves as an excellent starting point for Android development, owing to its intuitive drag-and-drop features allowing for the creation of simplistic applications.

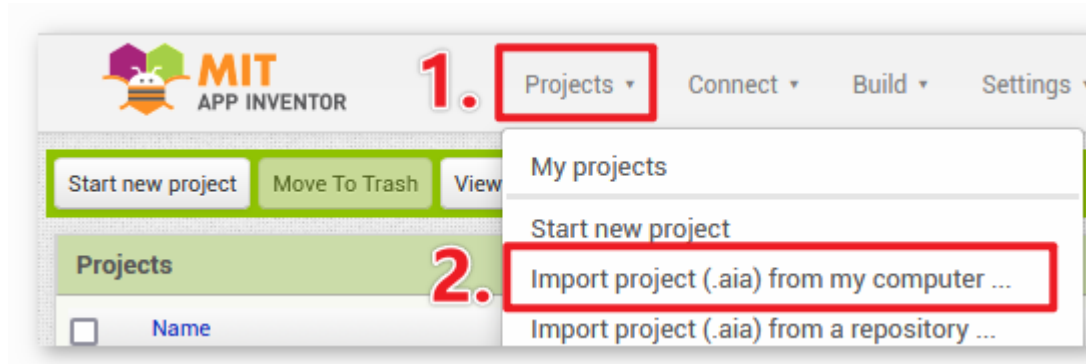
Now, let's begin.

1. Go to [MITAppInventor.org](https://MITAppInventor.org), and click “online tool” to login. You will require a Google account to register with MIT App Inventor.

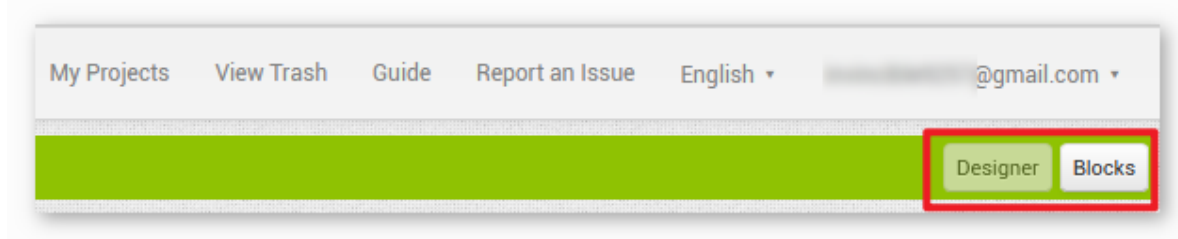


2. After logging in, navigate to **Projects** -> **Import project (.aia) from my computer**. Subsequently, upload the oled.aia file located in the path `ultimate-sensor-kit\iot_project\bluetooth\07-Bluetooth_oled`.

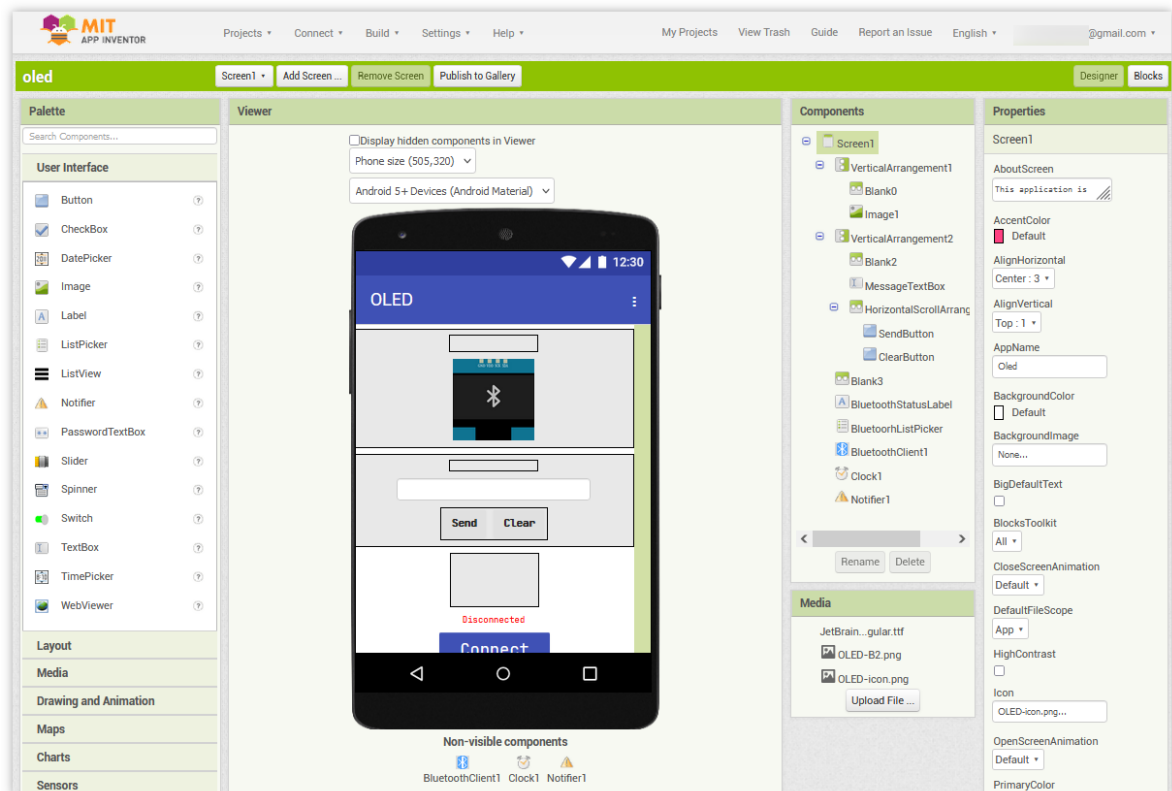
You can also directly download here: `oled.aia`



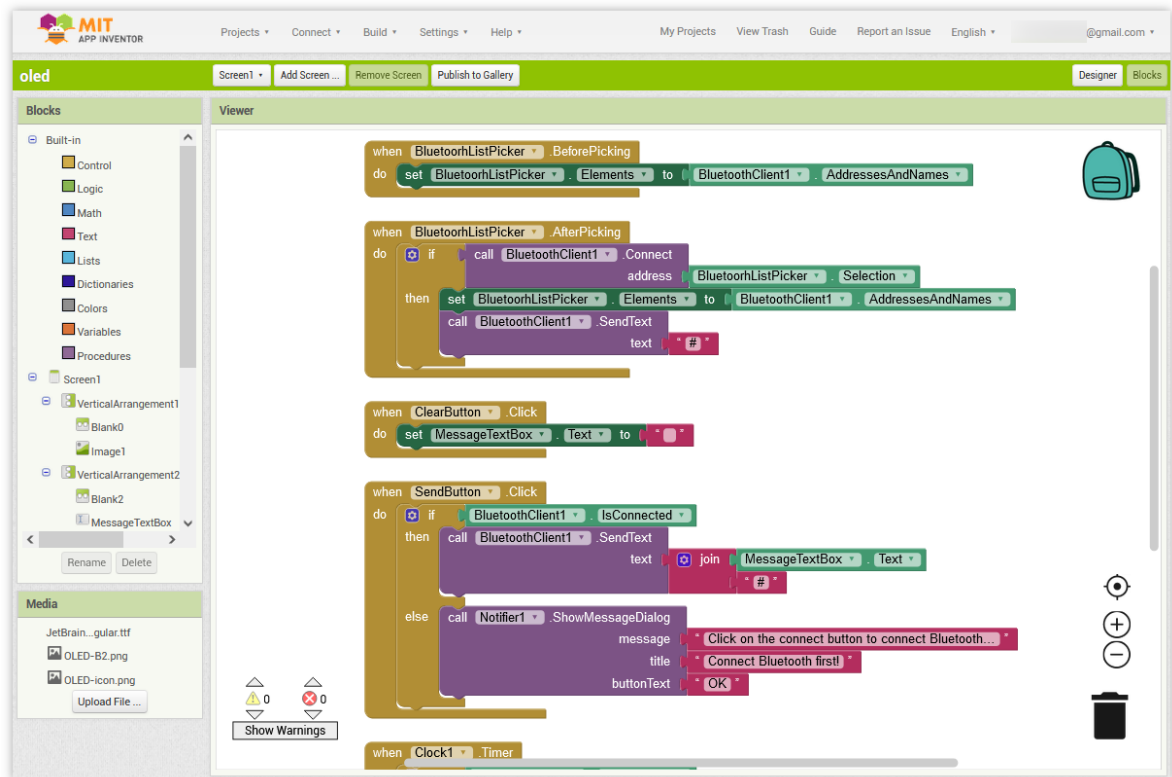
3. Upon uploading the .aia file, you will see the application on the MIT App Inventor software. This is a pre-configured template. You can modify this template after you have familiarized yourself with MIT App Inventor through the following steps.
4. In MIT App Inventor, you have 2 primary sections: the **Designer** and the **Blocks**. You can switch between these two sections in the upper right corner of the page.



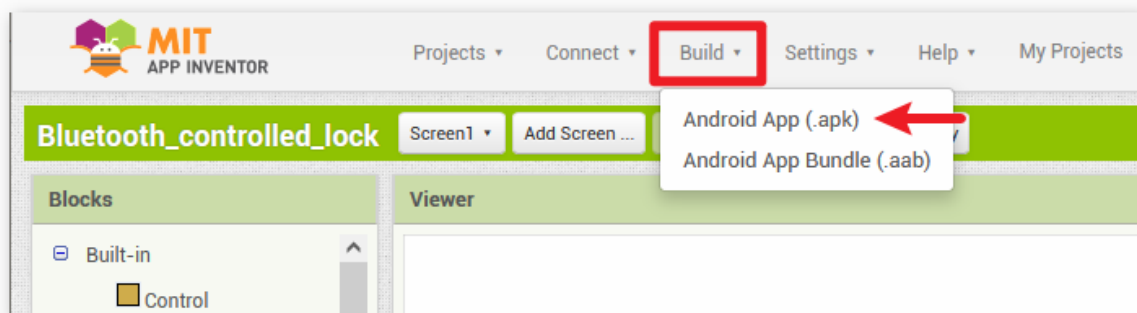
5. The **Designer** allows you to add buttons, text, screens, and modify the overall aesthetic of your application.



6. Next, there's the **Blocks** section. This section lets you craft custom functionalities for your app, allowing you to program each component on the app's GUI to achieve desired features.



7. To install the application on a smartphone, navigate to the **Build** tab.



- You can generate a .apk file. After selecting this option, a page will appear allowing you to choose between downloading a .apk file or scanning a QR code for installation. Follow the installation guide to complete the application installation.

You also download our pre-compiled APK here: [piano.apk](#)

- If you wish to upload this app to Google Play or another app marketplace, you can generate a .aab file.

### 3. Upload the Code

1. Open the 07-Bluetooth\_oled.ino file under the path of ultimate-sensor-kit\iot\_project\bluetooth\07-Bluetooth\_oled, or copy this code into **Arduino IDE**.

---

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit SSD1306**” and “**Adafruit GFX**” and install it.

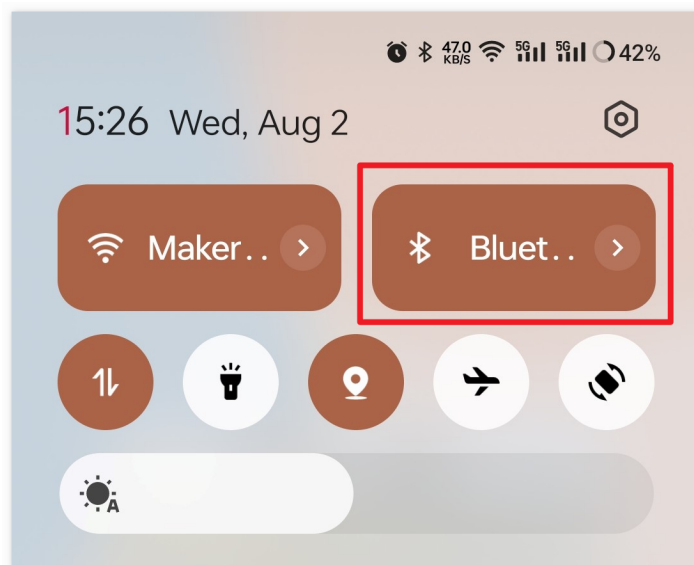
---

2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

### 4. App and Bluetooth module Connection

Ensure that the application created earlier is installed on your smartphone.

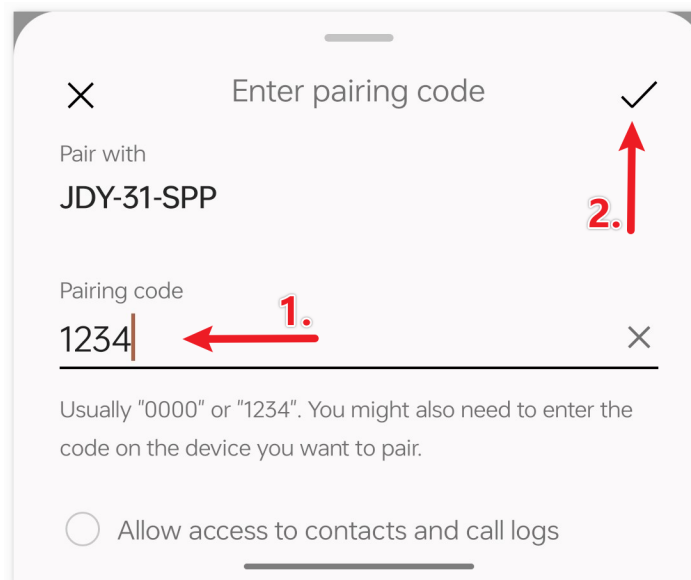
1. Initially, turn on **Bluetooth** on your smartphone.



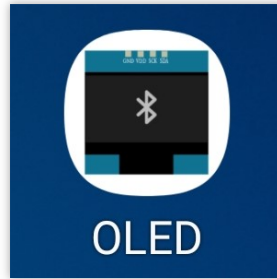
2. Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.



3. After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



4. Now open the newly installed **OLED APP**.



5. In the APP, click on **Connect** button to establish a connection between the APP and Bluetooth module.

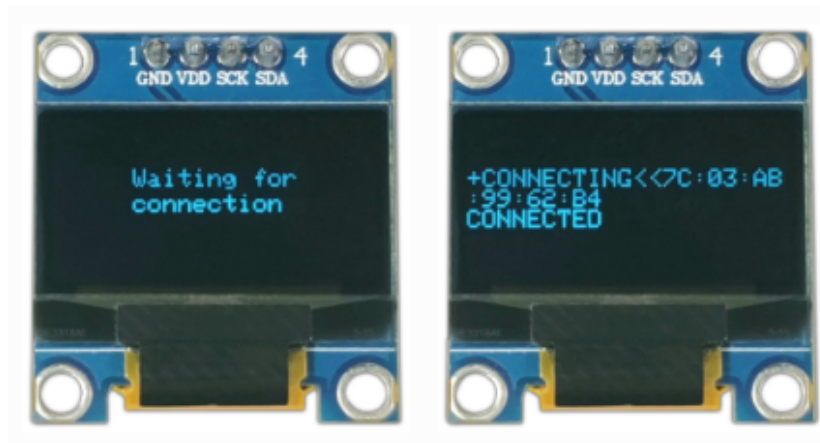


6. This page displays a list of all paired Bluetooth devices. Choose the xx.xx.xx.xx.xx.xx JDY-31-SPP option from the list. The name of each device is listed next to its MAC address.

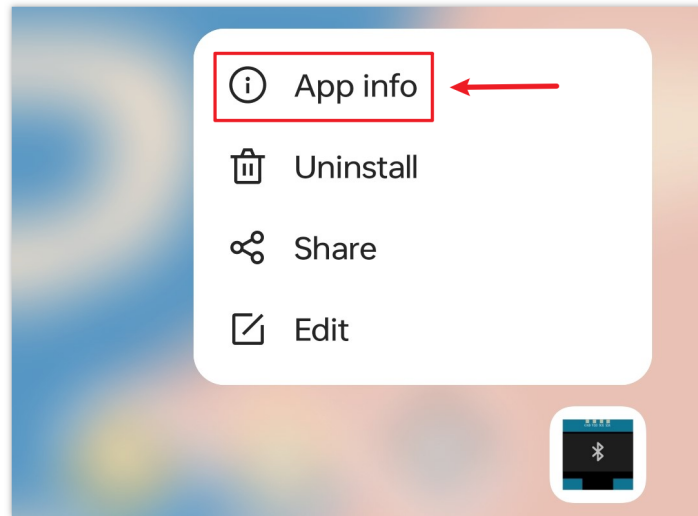


OLED	
78:2B	
60:AE	
FC:E8	
08:6A:04:05:7A:A7	JDY-31-SPP ←
F0:85	
E4:41:	

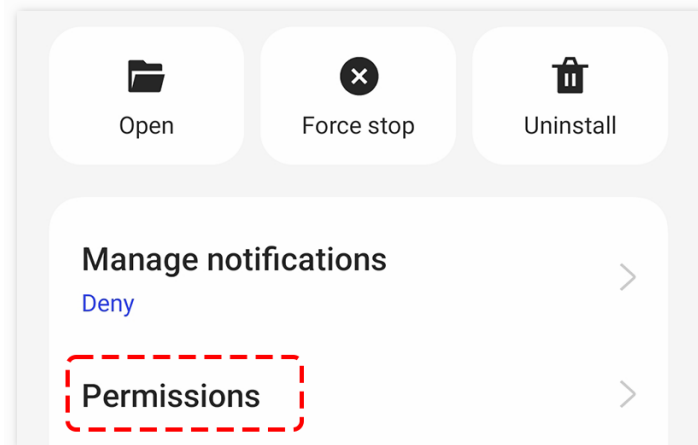
After a successful connection, the OLED display will switch from showing “Waiting for connection” to displaying the MAC address of the connected device.



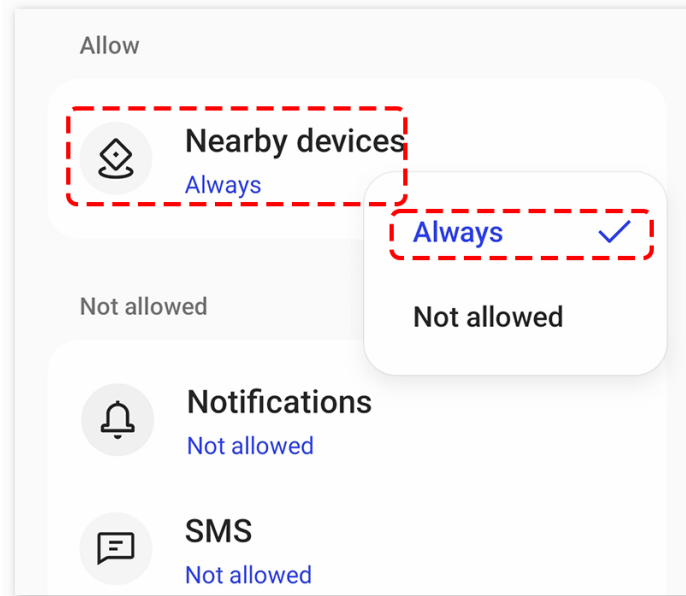
7. If you don't see any devices on the page shown above, it could be because this app is not authorized to scan for nearby devices. In such a case, you will need to adjust the settings manually.
  - To access the **APP Info** page, long-press the app icon and select it. Alternatively, if you have another method to reach this page, use that instead.



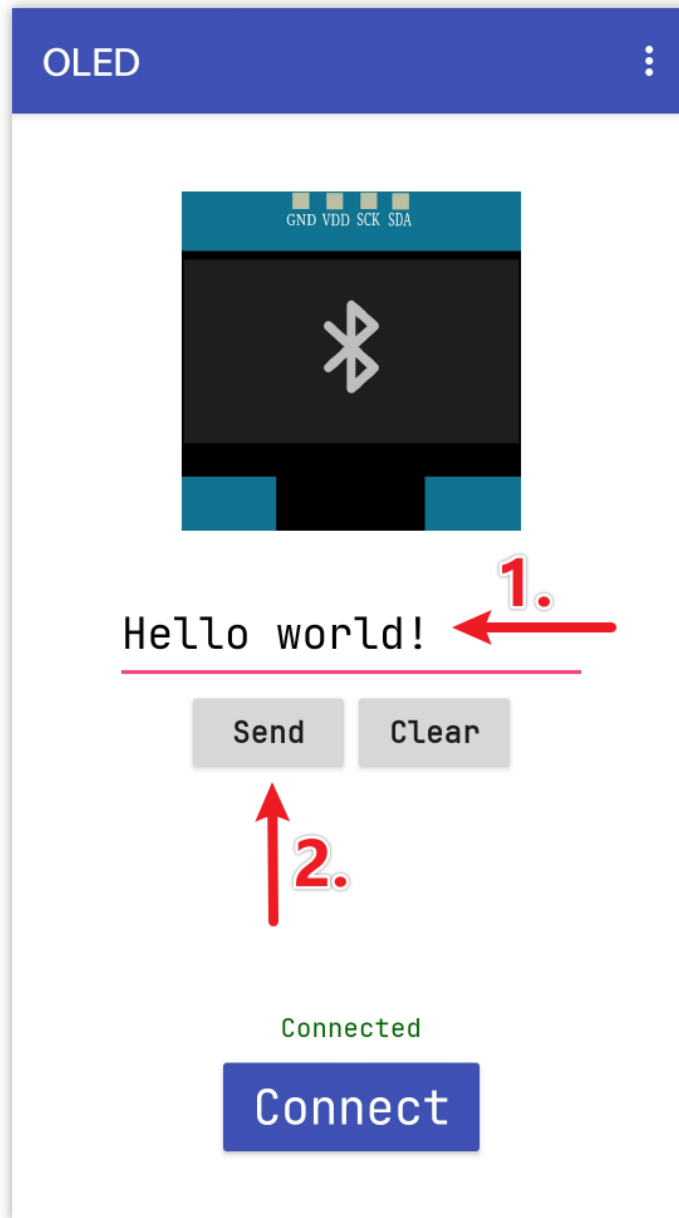
- Navigate to the **Permissions** page.



- To enable the APP to scan for nearby devices, go to **Nearby devices** and select **Always**.



- Now, restart the APP and repeat steps 5 and 6 to successfully connect to Bluetooth.
8. After a successful connection, you will be redirected to the main page. Enter your desired message in the text box provided and click on the send button to display it on the OLED screen.



## 5. Code explanation

### 1. Setting Up Bluetooth Communication:

This section includes the `SoftwareSerial` library and sets up the digital pins for Bluetooth communication. The standard `Serial` is for debugging while `bleSerial` is specifically for Bluetooth communication.

```
#include <SoftwareSerial.h>
const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);
```

### 2. Setting Up OLED Display:

The libraries and constants required to initialize and manage the OLED display are declared here.

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

### 3. Initialization:

In the `setup()` function, serial communications are initialized. The OLED display is started, and an initial message "Waiting for connection" is displayed.

```
void setup() {
  Serial.begin(9600);
  bleSerial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ;
  }
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(1);
  display.setCursor(32, 20);
  display.println("Waiting for");
  display.setCursor(32, 30);
  display.println("connection");
  display.display();
}
```

### 4. Main Loop:

Within the `loop()`, the code constantly checks for incoming data from the Bluetooth module. Once a full message (ending with a '#', the APP will automatically add a '#' at the end of the message that the user send) is received, it's displayed on the OLED. Also, the received message is printed to the serial monitor for debugging purposes.

```
void loop() {
  while (bleSerial.available() > 0) {
    character = bleSerial.read();
    message = message + character;
    if (character == '#') {
      message = message.substring(0, message.length() - 1);
      Serial.println();
      Serial.print("DEBUG:");
      Serial.println(message);
      display.clearDisplay();
      display.setTextColor(WHITE);
      display.setTextSize(1);
      display.setCursor(0, 20);
```

(continues on next page)

(continued from previous page)

```
        display.println(message);
        display.display();
        message = "";
        delay(200);
    }
}
```

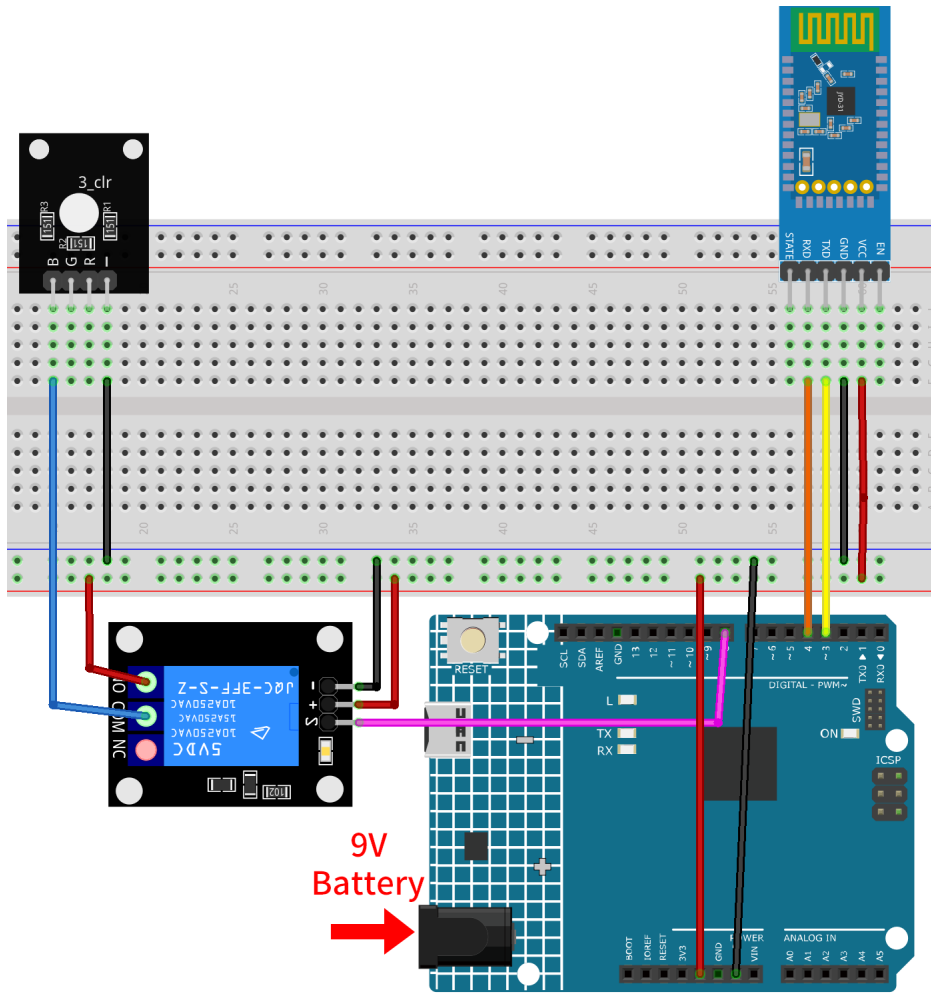
## 2.5.17 Bluetooth Remote Relay

This project uses an Android app created with MIT App Inventor to remotely control a relay module through the JDY-31 Bluetooth module connected to an Arduino Uno. When the app's buttons are pressed, it sends a simple command ('1' or '0') to the Arduino. When the Arduino receives the command '1' via Bluetooth, it activates the relay, and when it receives '0', it deactivates the relay. This provides a user-friendly interface on a smartphone to control devices connected to the relay.

The Android application will be constructed utilizing a complimentary web-based platform known as . The project presents an excellent opportunity to gain familiarity with the interfacing of an Arduino with a smartphone.

### 1. Build the Circuit

**Warning:** The following example demonstrates using a relay to control an LED module. **While you can connect the relay to other appliances in actual applications, extreme caution is required when dealing with HIGH AC voltage. Improper or incorrect use can lead to severe injury or even death. Therefore, it is intended for people who are familiar with and knowledgeable about HIGH AC voltage. Always prioritize safety.**



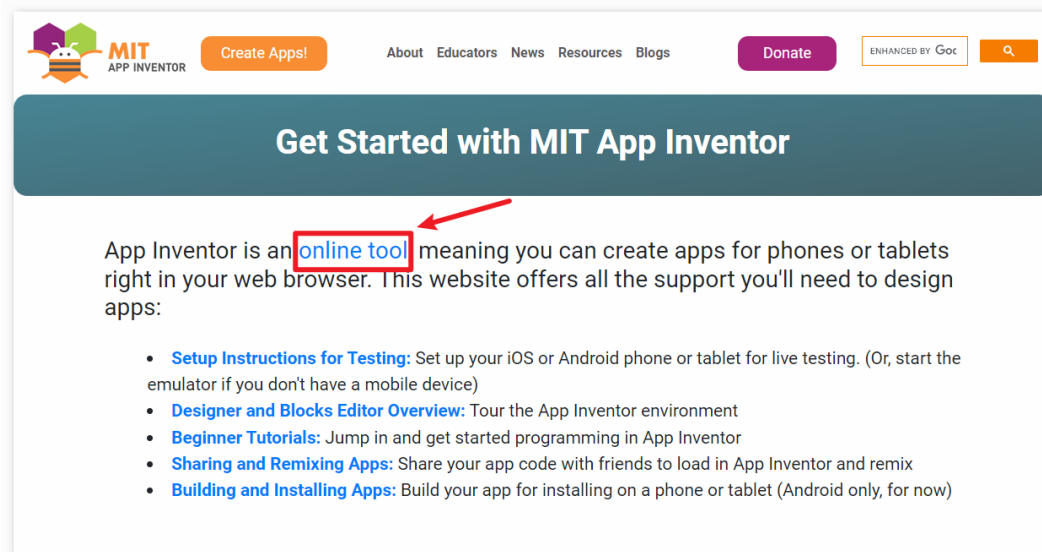
- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *5V Relay Module*
- *RGB Module*

## 2. Create the Android App

The Android application will be developed using a free web application known as . MIT App Inventor serves as an excellent starting point for Android development, owing to its intuitive drag-and-drop features allowing for the creation of simplistic applications.

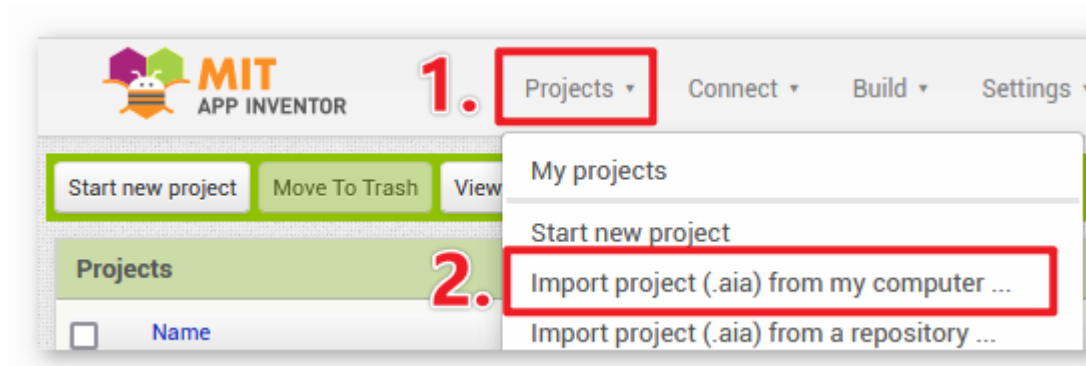
Now, let's begin.

1. Go to [MITAppInventor.com](https://MITAppInventor.com), and click “online tool” to login. You will require a Google account to register with MIT App Inventor.

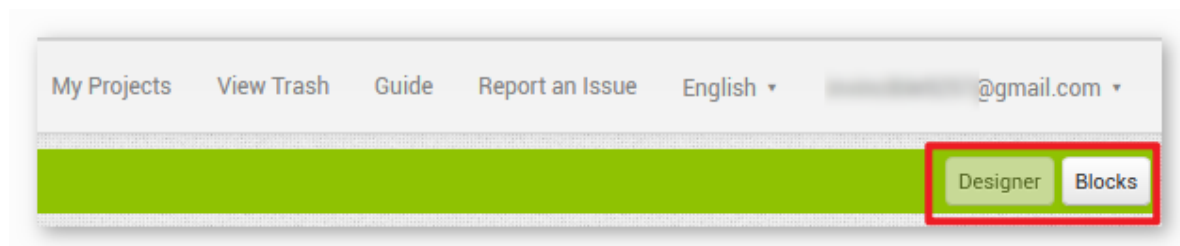


- After logging in, navigate to **Projects -> Import project (.aia) from my computer**. Subsequently, upload the RemoteRelay.aia file located in the path `ultimate-sensor-kit\iot_project\bluetooth\08-Bluetooth_remote_relay`.

You can also directly download here: RemoteRelay.aia

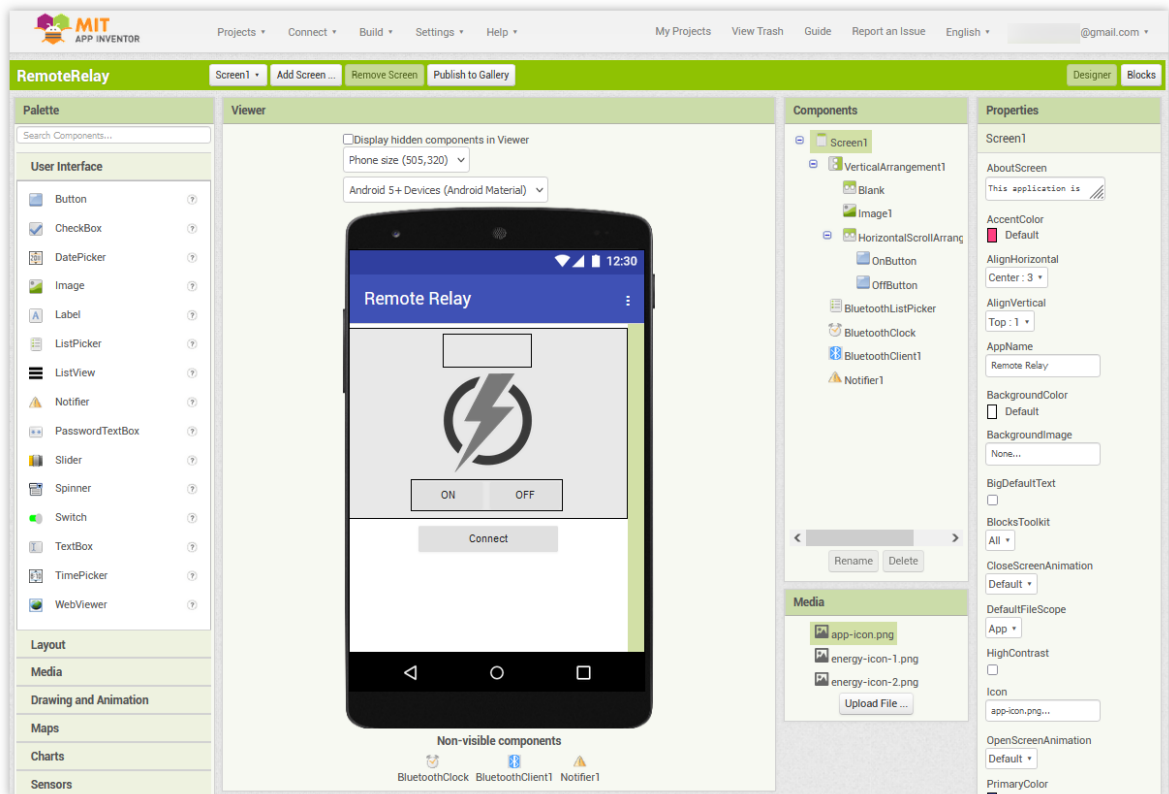


- Upon uploading the .aia file, you will see the application on the MIT App Inventor software. This is a pre-configured template. You can modify this template after you have familiarized yourself with MIT App Inventor through the following steps.
- In MIT App Inventor, you have 2 primary sections: the **Designer** and the **Blocks**. You can switch between these two sections in the upper right corner of the page.



- The **Designer** allows you to add buttons, text, screens, and modify the overall aesthetic of your application.

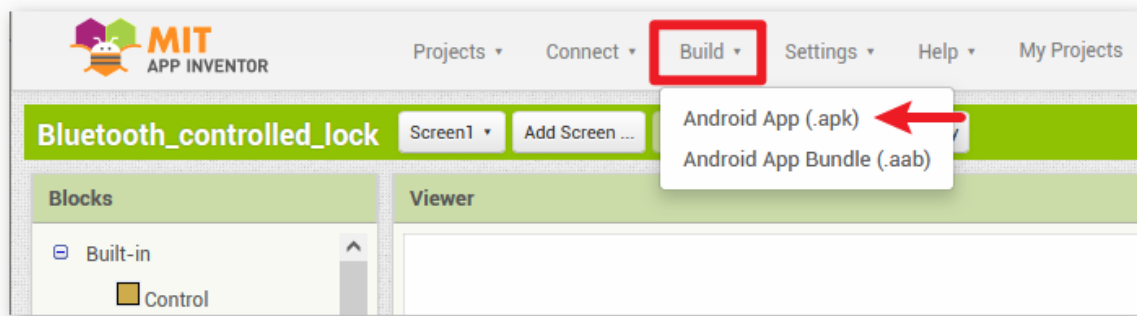




6. Next, there's the **Blocks** section. This section lets you craft custom functionalities for your app, allowing you to program each component on the app's GUI to achieve desired features.



7. To install the application on a smartphone, navigate to the **Build** tab.



- You can generate a .apk file. After selecting this option, a page will appear allowing you to choose between downloading a .apk file or scanning a QR code for installation. Follow the installation guide to complete the application installation.

You also download our pre-compiled APK here: RemoteRelay.apk

- If you wish to upload this app to Google Play or another app marketplace, you can generate a .aab file.

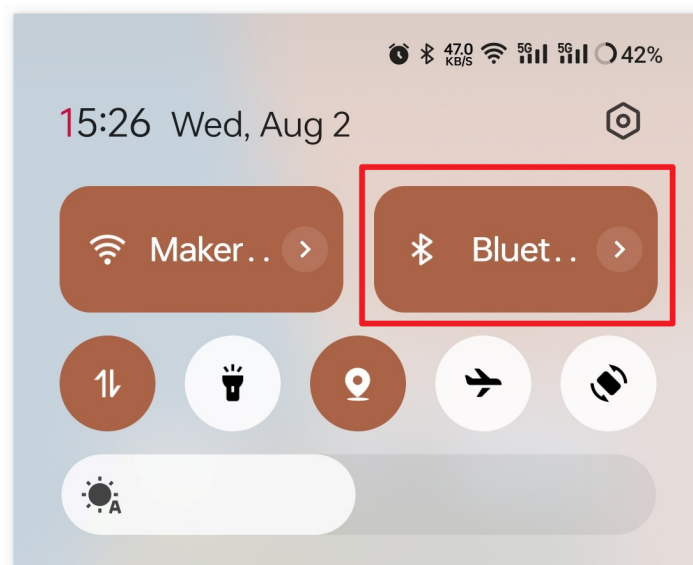
### 3. Upload the Code

1. Open the 08-Bluetooth\_remote\_relay.ino file under the path of ultimate-sensor-kit\iot\_project\bluetooth\08-Bluetooth\_remote\_relay, or copy this code into **Arduino IDE**.
2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

### 4. App and Bluetooth module Connection

Ensure that the application created earlier is installed on your smartphone.

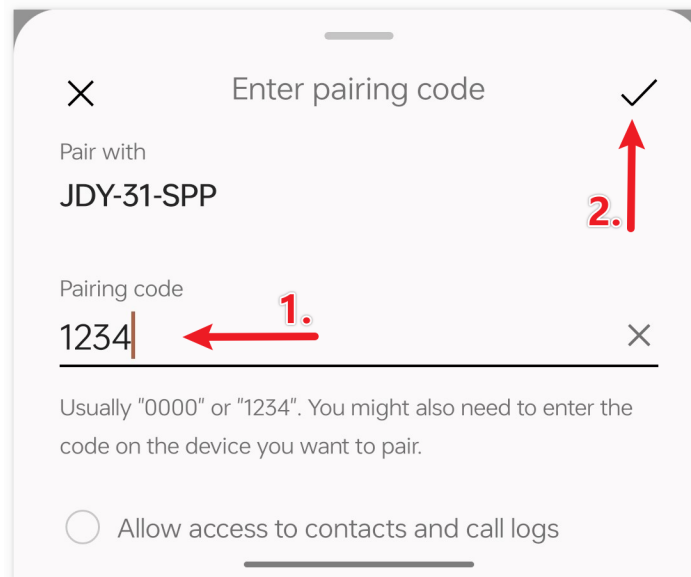
1. Initially, turn on **Bluetooth** on your smartphone.



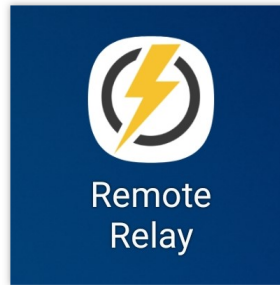
2. Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.



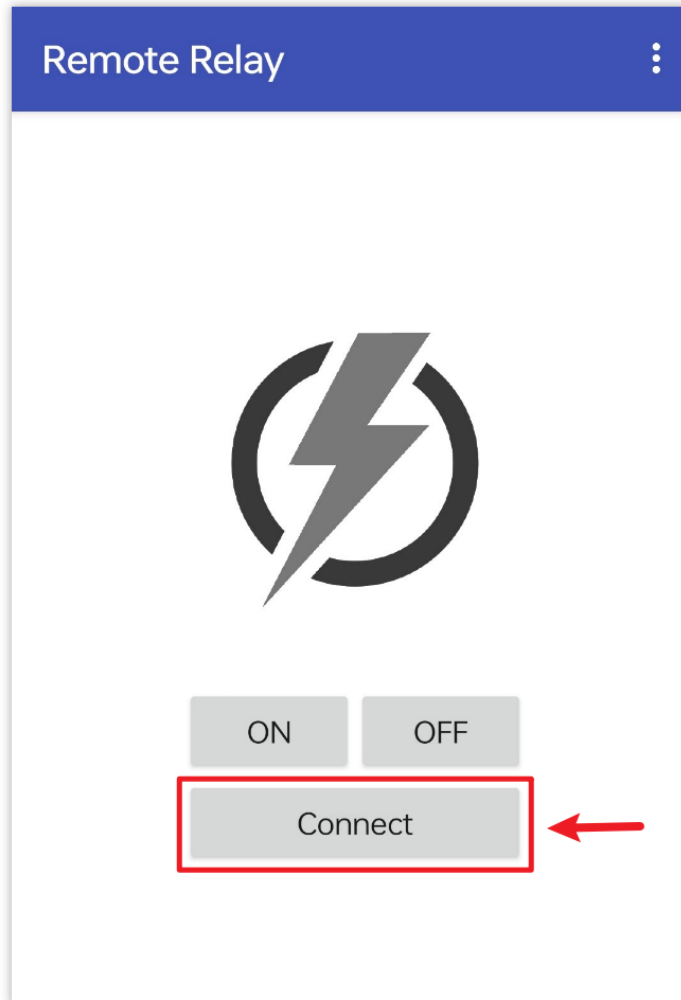
3. After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



4. Now open the newly installed **Remote Relay APP**.



5. In the APP, click on **Connect** button to establish a connection between the APP and Bluetooth module.

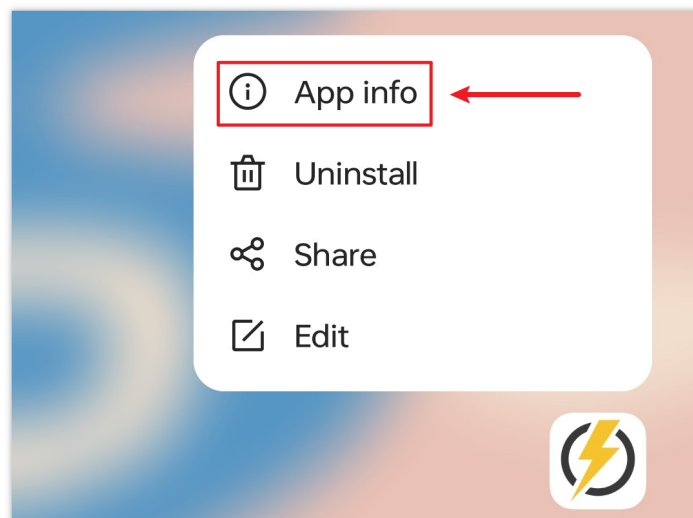


6. This page displays a list of all paired Bluetooth devices. Choose the `xx.xx.xx.xx.xx.xx JDY-31-SPP` option from the list. The name of each device is listed next to its MAC address.

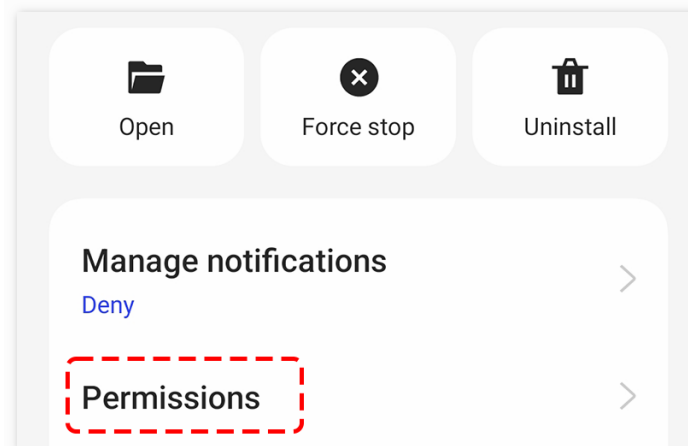


7. If you don't see any devices on the page shown above, it could be because this app is not authorized to scan for nearby devices. In such a case, you will need to adjust the settings manually.

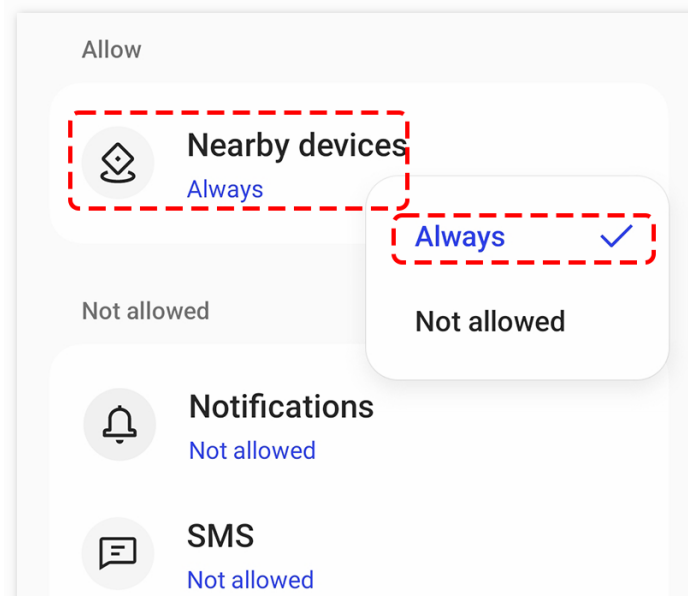
- To access the **APP Info** page, long-press the app icon and select it. Alternatively, if you have another method to reach this page, use that instead.



- Navigate to the **Permissions** page.



- To enable the APP to scan for nearby devices, go to **Nearby devices** and select **Always**.

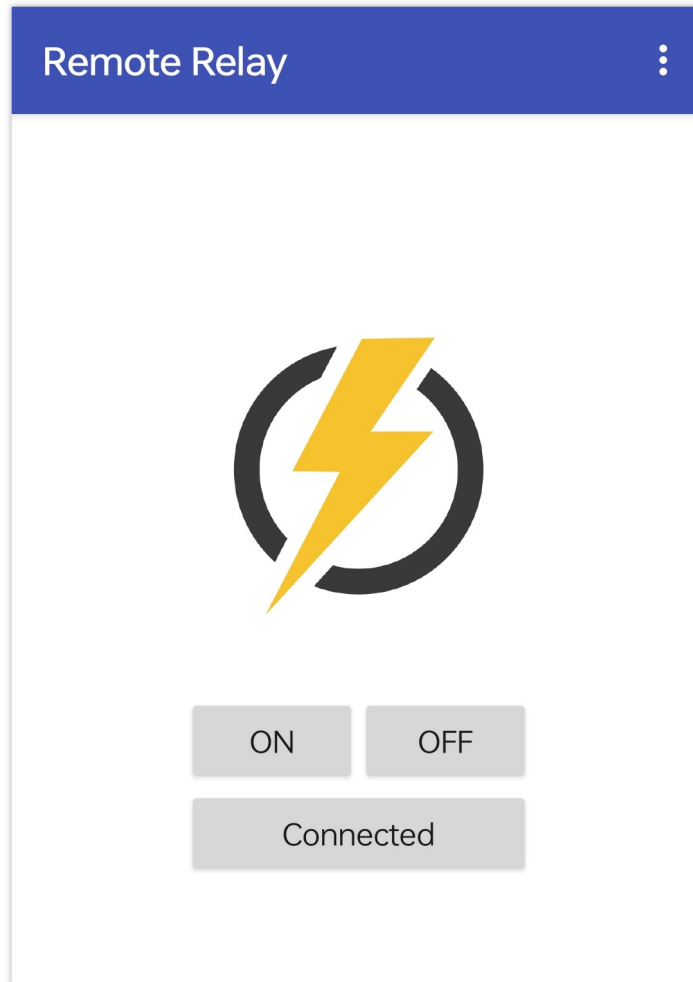


- Now, restart the APP and repeat steps 5 and 6 to successfully connect to Bluetooth.
8. After a successful connection, you will be redirected to the main page. Click the “ON” or “OFF” button to turn on or off the relay.

---

**Note:** When the MAC address of Bluetooth contains “1”, the relay will be turned on and then quickly turned off after the first successful Bluetooth connection. Because when the Bluetooth is connected, the MAC address will be sent to Arduino. Arduino detects “1” and then opens the relay. After Bluetooth initialization, the app sends 0 to Arduino via Bluetooth to ensure that the initial state of the relay is closed after connection.

---



## 5. Code explanation

### 1. Library and Global Variable Initialization

```
#include <SoftwareSerial.h>

const int bluetoothTx = 3;
const int bluetoothRx = 4;
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx);

const int relayPin = 8;
```

This segment includes the SoftwareSerial library and sets up the global variables. Pins 3 and 4 are defined for transmitting and receiving data with the Bluetooth module, respectively. Additionally, the relay module is connected to pin 8.

### 2. setup() Function

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

(continues on next page)

(continued from previous page)

```
pinMode(relayPin, OUTPUT);
}
```

It initiates the Serial monitor and Bluetooth module communication at a baud rate of 9600. It also sets the relayPin as an OUTPUT pin.

### 3. loop() Function

```
void loop() {
  if (bleSerial.available() > 0) {
    char message = bleSerial.read();
    // Serial.println(message); //for debug

    if (message == '1') {
      digitalWrite(relayPin, HIGH);
      Serial.println("On");
    } else if (message == '0') {
      digitalWrite(relayPin, LOW);
      Serial.println("Off");
    }
  }
}
```

The loop() function runs continuously. It checks if there's a message received from the Bluetooth module. If a message is received, it reads the character. Depending on the character ('1' or '0'), it either turns the relay on or off and sends a confirmation message ("On" or "Off") to the Serial monitor.

## 2.5.18 Bluetooth Voice-control Relay

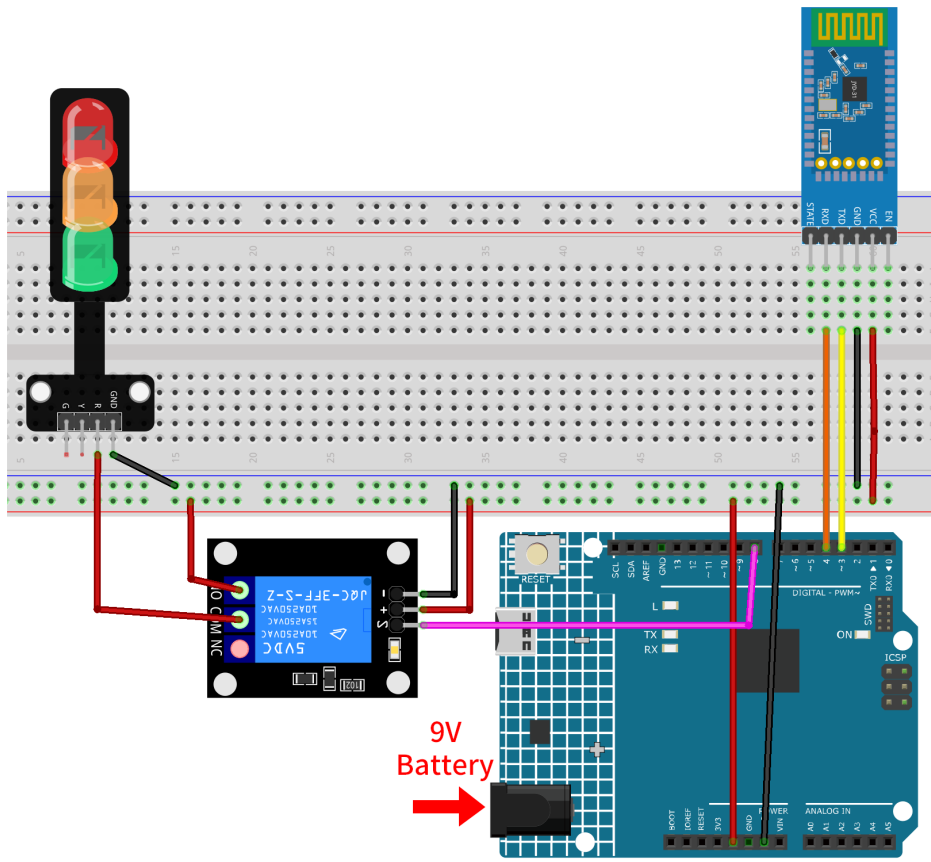
This project integrates an Android app, developed using MIT App Inventor, with an Arduino Uno board. The app offers a voice input feature. When the user's voice input contains the word "on", the app sends a "1" message via Bluetooth to the Arduino, instructing it to turn the relay on. Similarly, if the voice input contains the word "off", the app sends a "0" message, signaling the Arduino to turn the relay off. Once the Arduino receives these messages, it processes them and turns the relay on or off accordingly.

The Android application will be constructed utilizing a complimentary web-based platform known as . The project presents an excellent opportunity to gain familiarity with the interfacing of an Arduino with a smartphone.

### 1. Build the Circuit

**Warning:** The following example demonstrates using a relay to control an traffic light module. **While you can connect the relay to other appliances in actual applications, extreme caution is required when dealing with HIGH AC voltage. Improper or incorrect use can lead to severe injury or even death. Therefore, it is intended for people who are familiar with and knowledgeable about HIGH AC voltage. Always prioritize safety.**





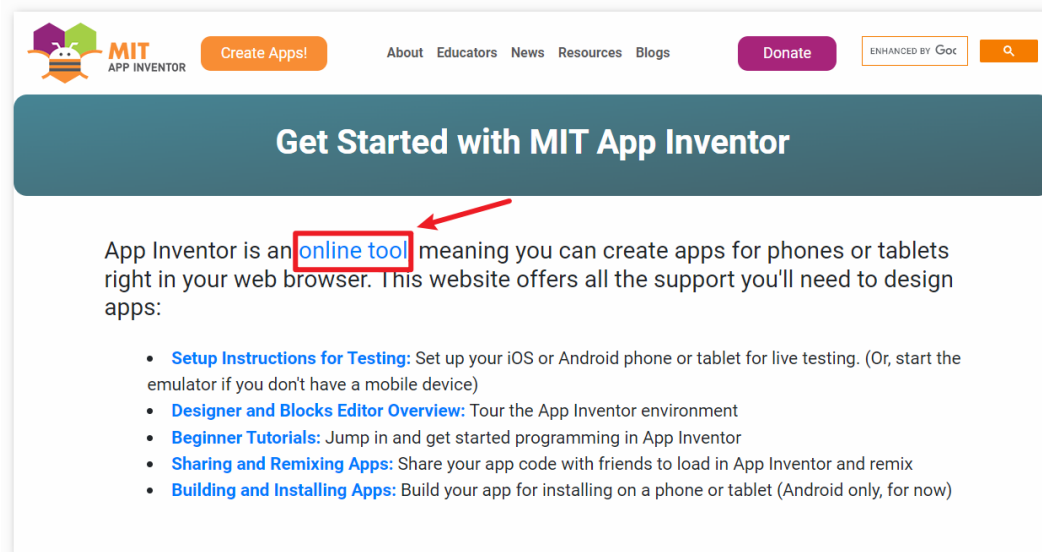
- *Arduino UNO R4 Minima Board*
- *JDY-31 Bluetooth Module*
- *5V Relay Module*
- *Traffic Light Module*

## 2. Create the Android App

The Android application will be developed using a free web application known as . MIT App Inventor serves as an excellent starting point for Android development, owing to its intuitive drag-and-drop features allowing for the creation of simplistic applications.

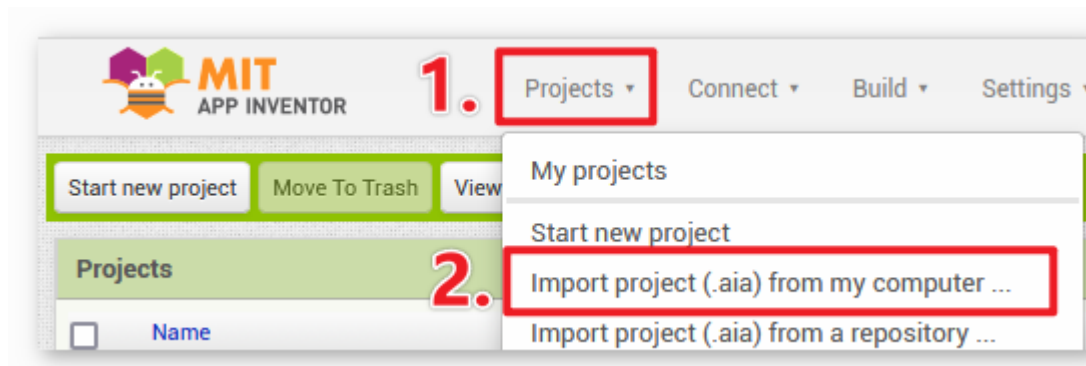
Now, let's begin.

1. Go to [MITAppInventor.org](https://MITAppInventor.org), and click “online tool” to login. You will require a Google account to register with MIT App Inventor.

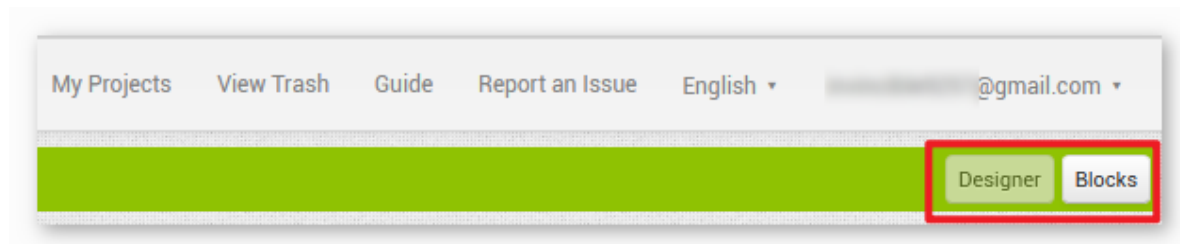


2. After logging in, navigate to **Projects -> Import project (.aia) from my computer**. Subsequently, upload the VoiceControl.aia file located in the path `ultimate-sensor-kit\iot_project\bluetooth\09-Bluetooth_voice_control_relay`.

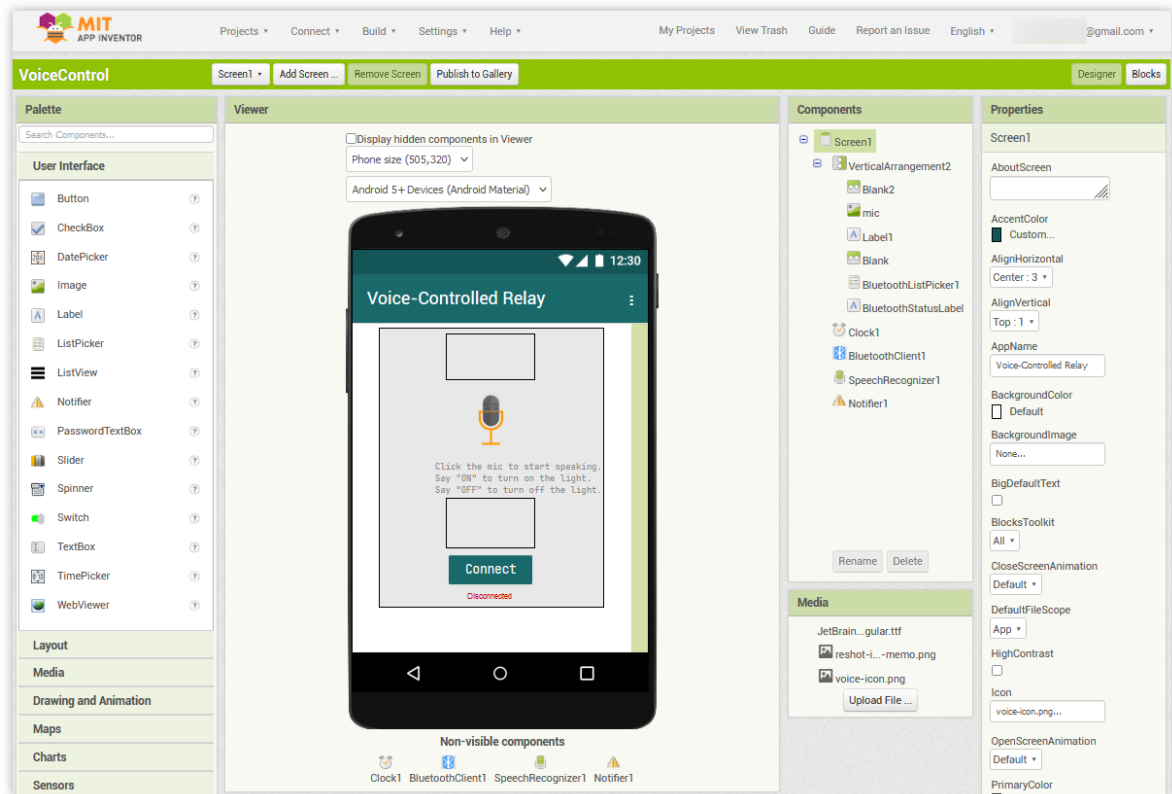
You can also directly download here: `VoiceControl.aia`



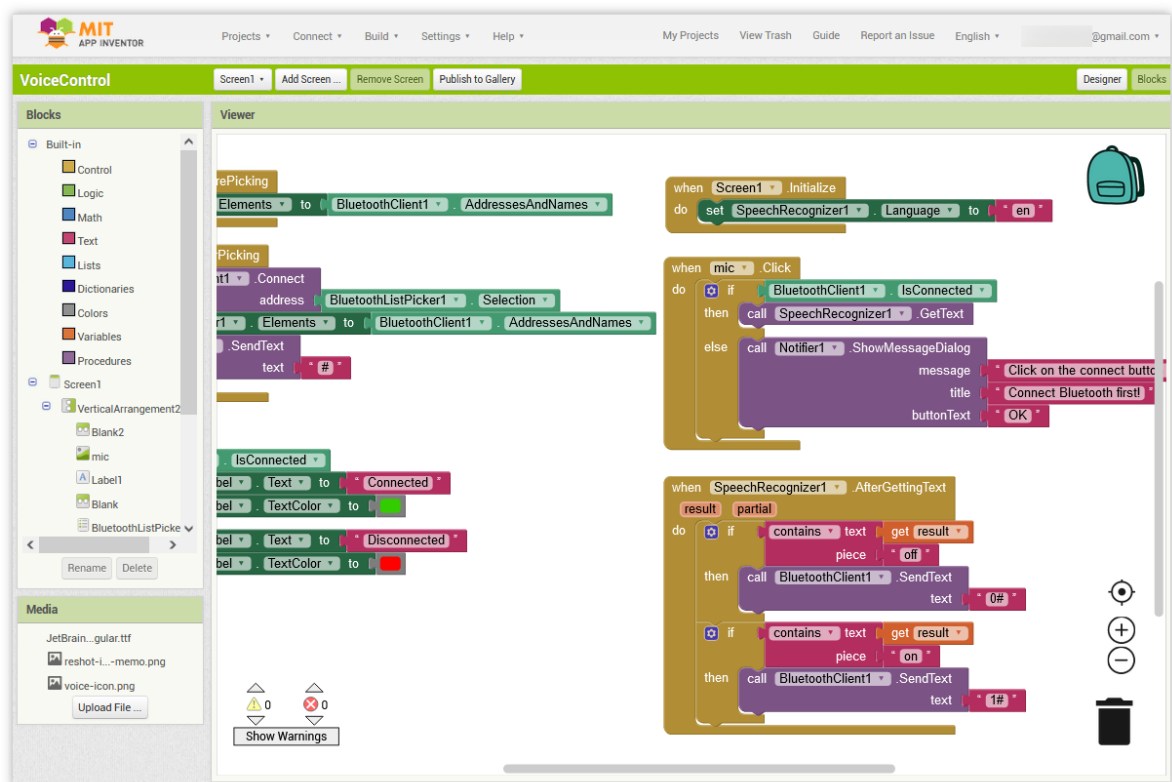
3. Upon uploading the .aia file, you will see the application on the MIT App Inventor software. This is a pre-configured template. You can modify this template after you have familiarized yourself with MIT App Inventor through the following steps.
4. In MIT App Inventor, you have 2 primary sections: the **Designer** and the **Blocks**. You can switch between these two sections in the upper right corner of the page.



5. The **Designer** allows you to add buttons, text, screens, and modify the overall aesthetic of your application.



6. Next, there's the **Blocks** section. This section lets you craft custom functionalities for your app, allowing you to program each component on the app's GUI to achieve desired features.

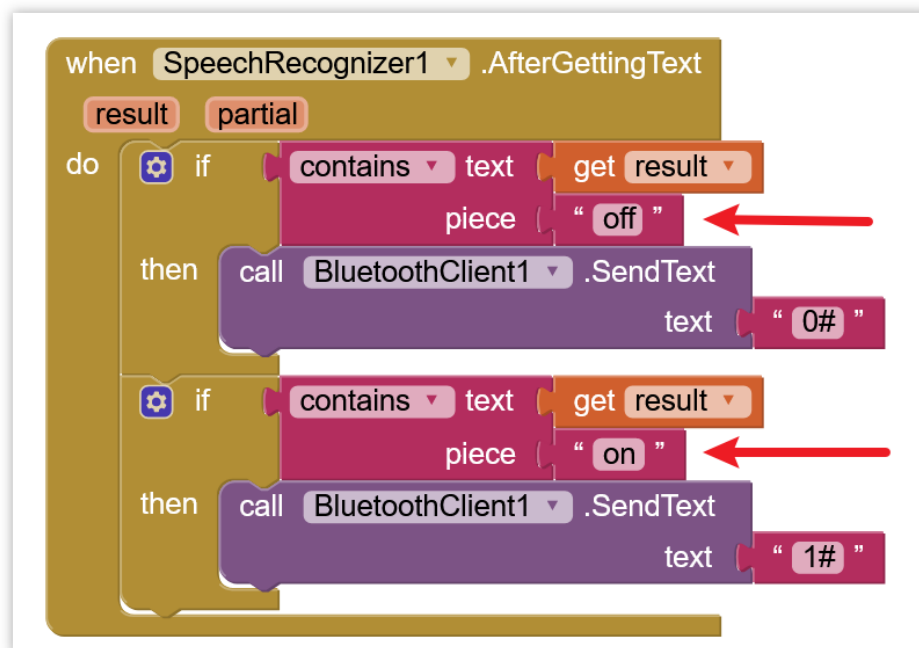


In this project, we take English recognition as an example. If you want to apply recognition of other languages, you need to modify the code block below and then compile the APK by yourself.

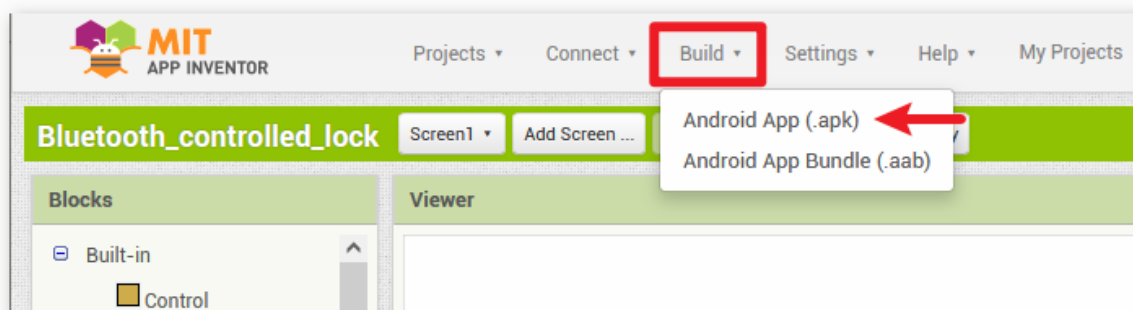
Firstly, you need to set `SpeechRecognizer1.Language` to the **language tag** of the language you want to recognize. Language is specified using a language tag with an optional region suffix, such as `en`, `de` or `ja`. The language tag can be found at .



Then, you need to modify the corresponding judgment condition. The part indicated by the arrow in the following figure.



7. To install the application on a smartphone, navigate to the **Build** tab.



- You can generate a .apk file. After selecting this option, a page will appear allowing you to choose between downloading a .apk file or scanning a QR code for installation. Follow the installation guide to complete the application installation.

You also download our pre-compiled APK here: `VoiceControl.apk`

- If you wish to upload this app to Google Play or another app marketplace, you can generate a `.aab` file.

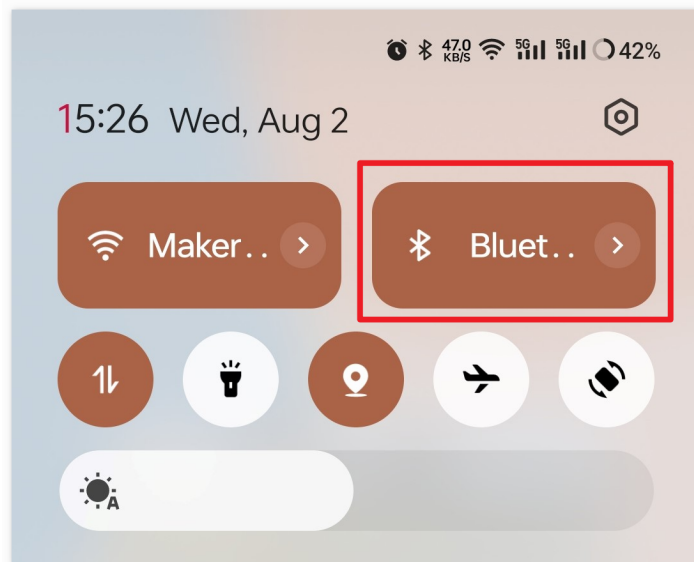
### 3. Upload the Code

1. Open the `09-Bluetooth_voice_control_relay.ino` file under the path of `ultimate-sensor-kit\iot_project\bluetooth\09-Bluetooth_voice_control_relay`, or copy this code into **Arduino IDE**.
2. After selecting the correct board and port, click the **Upload** button.
3. Open the Serial monitor(set baudrate to **9600**) to view debug messages.

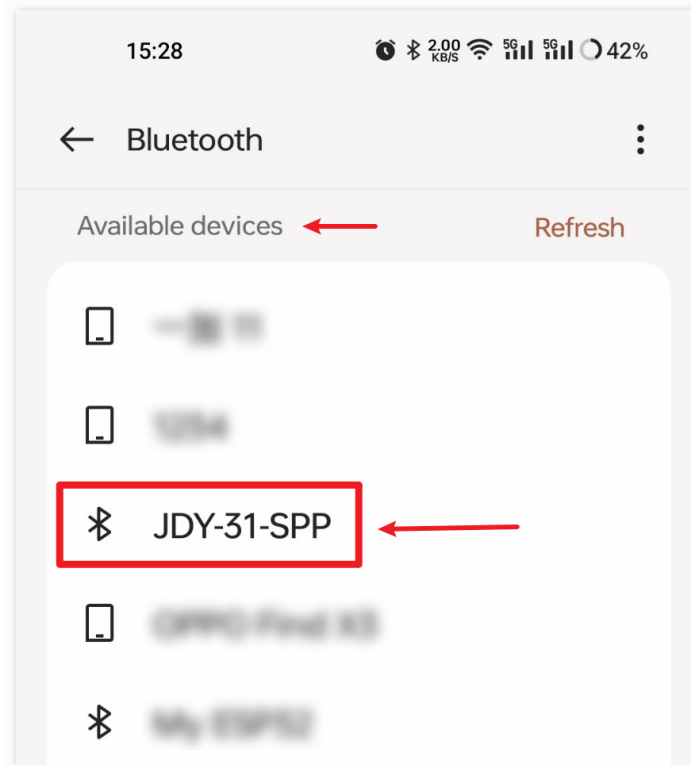
### 4. App and Bluetooth module Connection

Ensure that the application created earlier is installed on your smartphone.

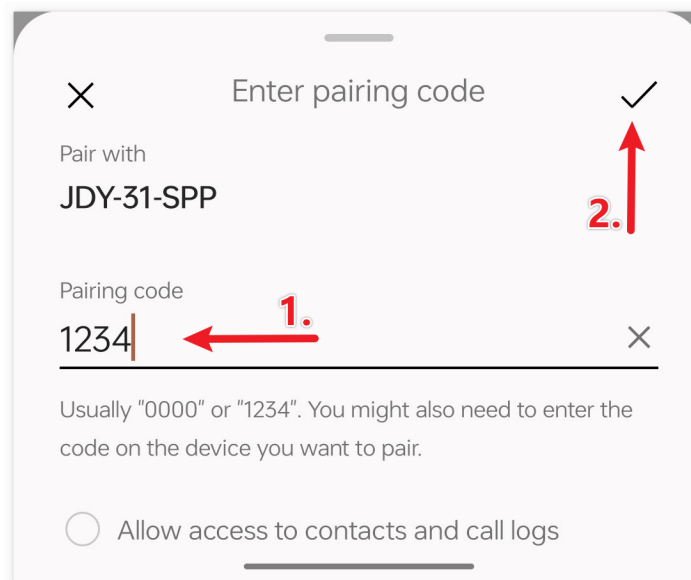
1. Initially, turn on **Bluetooth** on your smartphone.



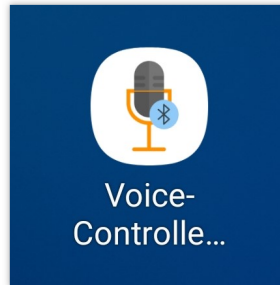
2. Navigate to the **Bluetooth settings** on your smartphone and look for names like **JDY-31-SPP**.



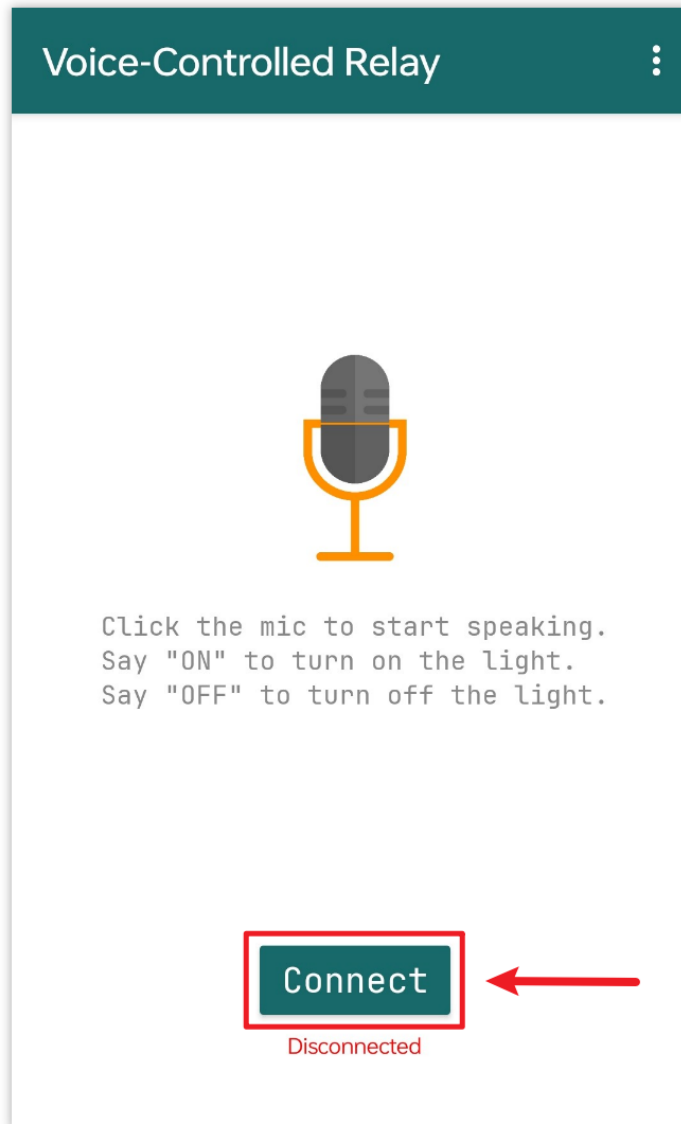
3. After clicking it, agree to the **Pair** request in the pop-up window. If prompted for a pairing code, please enter “1234”.



4. Now open the newly installed **Voice-Controlled Relay** APP.



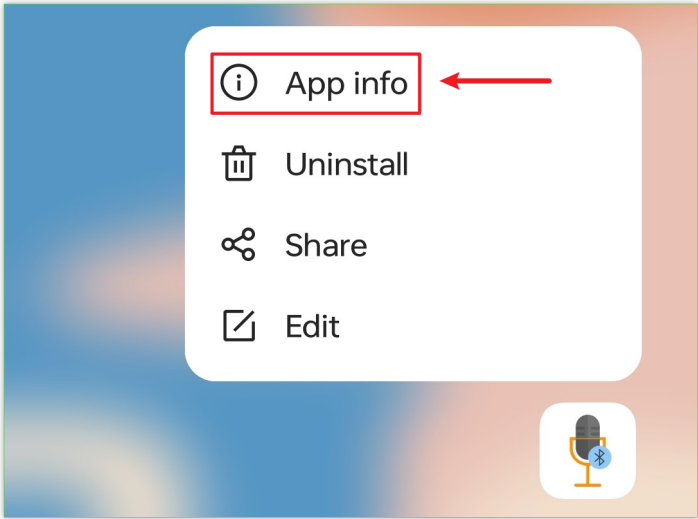
5. In the APP, click on **Connect** button to establish a connection between the APP and Bluetooth module.



6. This page displays a list of all paired Bluetooth devices. Choose the `xx.xx.xx.xx.xx.xx JDY-31-SPP` option from the list. The name of each device is listed next to its MAC address.

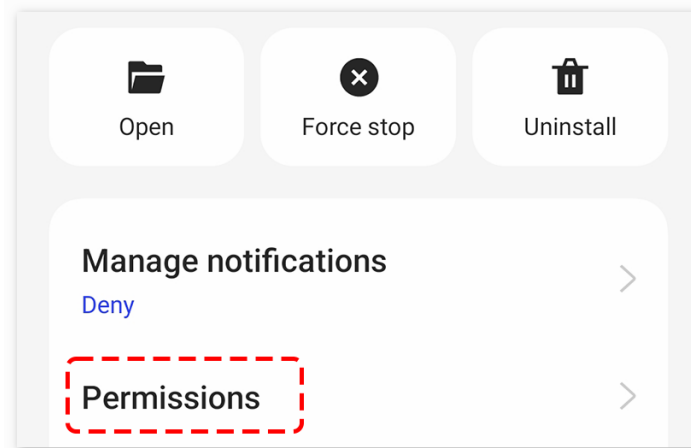


7. If you don't see any devices on the page shown above, it could be because this app is not authorized to scan for nearby devices. In such a case, you will need to adjust the settings manually.
- To access the **APP Info** page, long-press the app icon and select it. Alternatively, if you have another method to reach this page, use that instead.

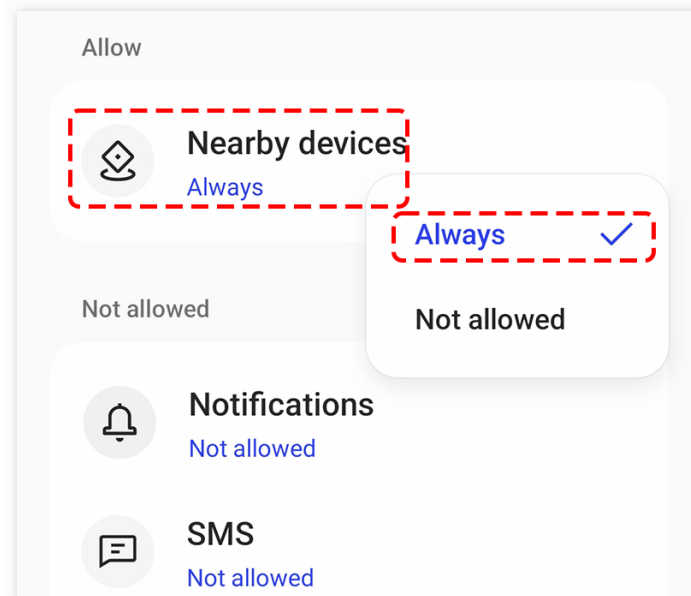


- Navigate to the **Permissions** page.

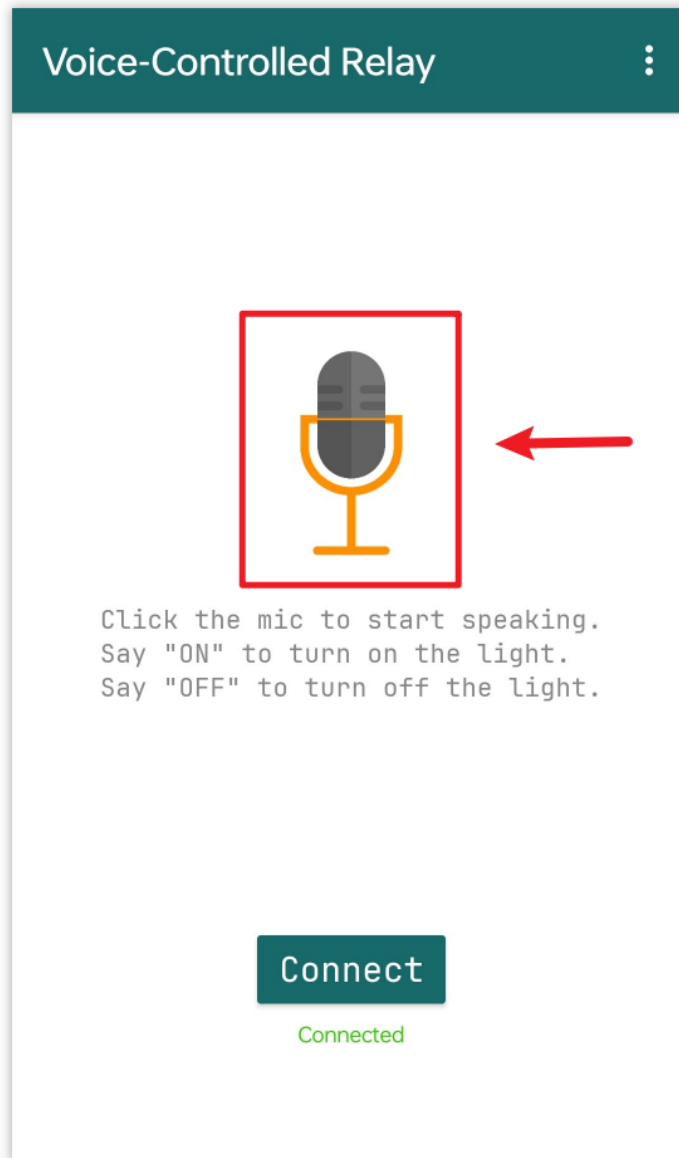




- To enable the APP to scan for nearby devices, go to **Nearby devices** and select **Always**.

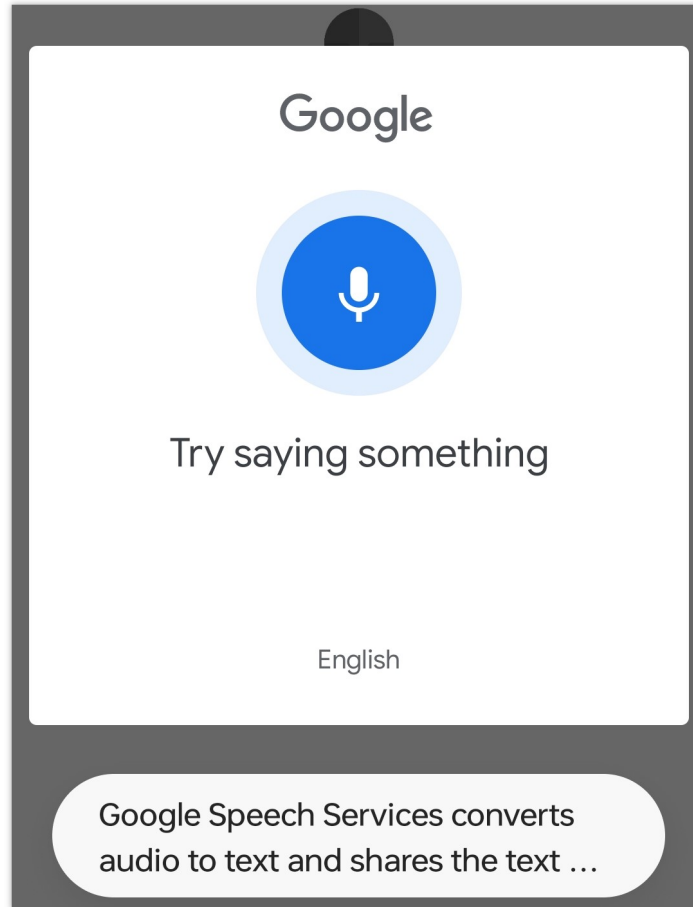


- Now, restart the APP and repeat steps 5 and 6 to successfully connect to Bluetooth.
8. After a successful connection, you will be redirected to the main page. Click the “ON” or “OFF” button to turn on or off the relay.



Although the relay can be controlled by voice input commands containing “on” or “off”, it is recommended to use longer or complete sentences such as “turn on” or “turn on the light” to avoid recognition errors caused by short voice inputs.

Speech recognition function relies on Google’s speech recognition engine, so you may need to install in advance (most Android phones come with this feature pre-installed).



## 5. Code explanation

1. Set up Bluetooth module communication

```
#include <SoftwareSerial.h>
const int bluetoothTx = 3;           // bluetooth tx to 3 pin
const int bluetoothRx = 4;           // bluetooth rx to 4 pin
SoftwareSerial bleSerial(bluetoothTx, bluetoothRx); // Declare SoftwareSerial_
// object for Bluetooth communication
```

This section initializes the Bluetooth communication using the SoftwareSerial library. This library allows the Arduino to have an additional serial port. The Bluetooth module's "TX" pin is connected to the Arduino's pin 3 and the "RX" pin is connected to pin 4.

2. Define variables and relay control pin

```
char character; // Character received from Bluetooth serial
String message; // Stores the complete message from Bluetooth
const int relayPin = 8;
```

Here, we declare variables to store individual characters received from Bluetooth (`character`) and the complete message (`message`). The `relayPin` is initialized to pin 8, which will be used to control the relay.

3. Initialize serial communication and set relay pin mode

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

In the `setup()` function, we initialize the standard serial port and the Bluetooth serial port with a baud rate of 9600. We also set the `relayPin` as an output pin.

#### 4. Read Bluetooth messages and control the relay

```
void loop() {  
  while (bleSerial.available() > 0) {  
    character = bleSerial.read();  
    message = message + character;  
    if (character == '#') {  
      message = message.substring(0, message.length() - 1);  
      Serial.println();  
      Serial.print("DEBUG:");  
      Serial.println(message);  
      if (message == "1") {  
        digitalWrite(relayPin, HIGH);  
        Serial.println("On");  
      } else if (message == "0") {  
        digitalWrite(relayPin, LOW);  
        Serial.println("Off");  
      }  
      message = "";  
      delay(200);  
    }  
  }  
}
```

The `loop()` function continuously checks for incoming messages from Bluetooth. When a message is received, each character is appended to the `message` string. Once the `#` character is detected, the message is considered complete. We then remove the `#`, print a debug message, and check the content. If it's "1", the relay is turned on; if "0", it's turned off. The `message` string is then cleared, and we wait briefly before checking for the next message.

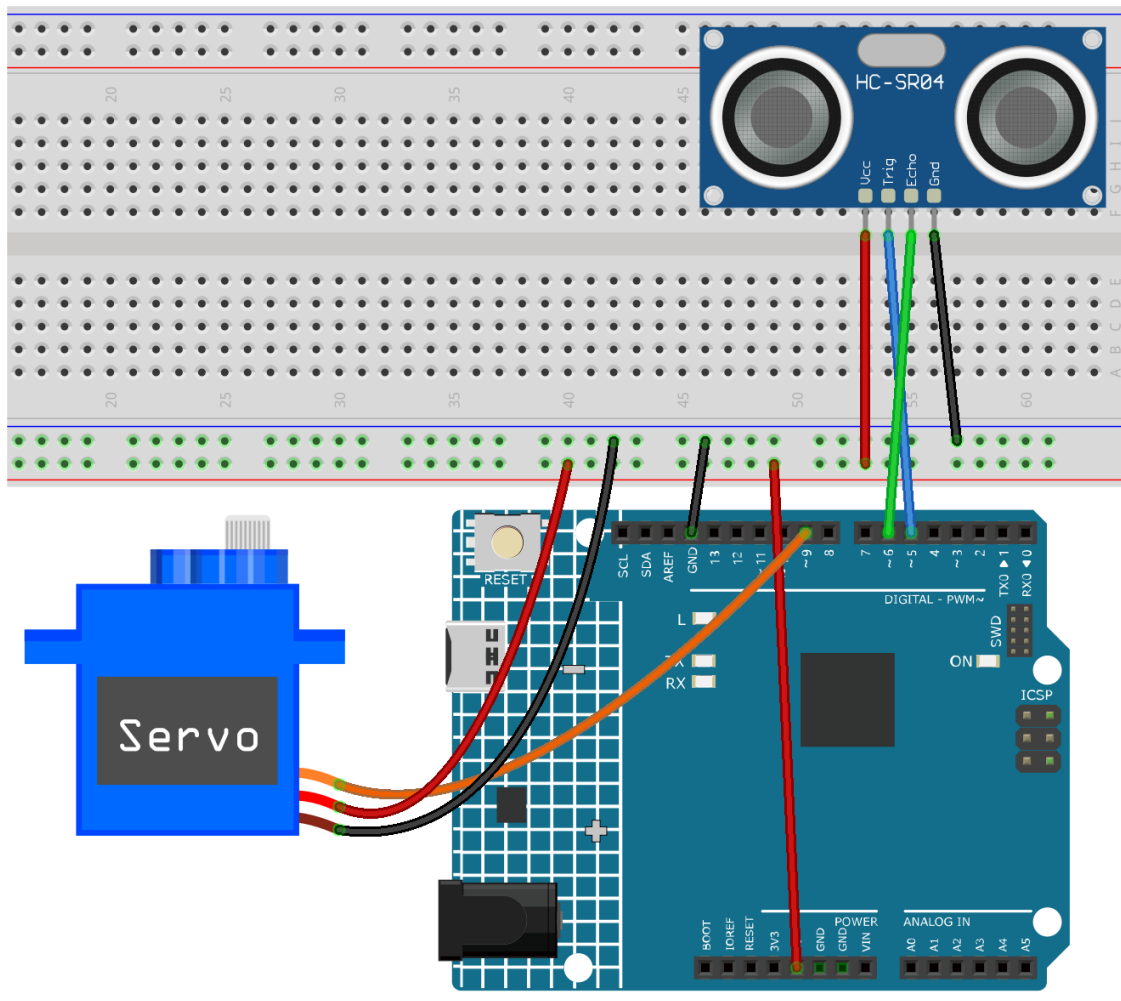
## 2.6 Fun Projects

In this chapter, you will find some fun projects. These projects involve the use of multiple electronic components and illustrate the fundamental logic behind how most programs interact with reality.

### 2.6.1 Smart trashcan

This project revolves around the concept of a smart trash can. The primary aim is to have the trash can's lid automatically open when an object approaches within a set distance (20cm in this case). The functionality is achieved by using an ultrasonic distance sensor paired with a servo motor. The distance between the object and the sensor is continually measured. If the object is close enough, the servo motor is triggered to open the lid.

#### 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *Ultrasonic Sensor Module (HC-SR04)*
- *Servo Motor (SG90)*

## 2. Code

1. Open the 01-Smart\_trashcan.ino file under the path of ultimate-sensor-kit\fun\_project\01-Smart\_trashcan, or copy this code into **Arduino IDE**.

## 3. Code explanation

The project is based on real-time monitoring of the distance between an object and a trash can. An ultrasonic sensor continuously measures this distance, and if an object approaches within 20cm, the trash can interprets it as an intention to dispose of waste and automatically opens its lid. This automation adds smartness and convenience to a regular trash can.

### 1. Initial Setup and Variable Declaration

Here, we're including the Servo library and defining the constants and variables we'll use. The pins for the servo and the ultrasonic sensor are declared. We also have an array `averDist` to hold the three distance measurements.

```
#include <Servo.h>
Servo servo;
const int servoPin = 9;
const int openAngle = 0;
const int closeAngle = 90;
const int trigPin = 5;
const int echoPin = 6;
long distance, averageDistance;
long averDist[3];
const int distanceThreshold = 20;
```

### 2. setup() Function

The `setup()` function initializes serial communication, configures the ultrasonic sensor's pins, and sets the initial position of the servo to the closed position.

```
void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  servo.attach(servoPin);
  servo.write(closeAngle);
  delay(100);
}
```

### 3. loop() Function

The `loop()` function is responsible for continuously measuring the distance, computing its average, and then making a decision whether to open or close the trash can's lid based on this averaged distance.

```
void loop() {
  for (int i = 0; i <= 2; i++) {
    distance = readDistance();
    averDist[i] = distance;
    delay(10);
  }
  averageDistance = (averDist[0] + averDist[1] + averDist[2]) / 3;
  Serial.println(averageDistance);
}
```

(continues on next page)

(continued from previous page)

```
if (averageDistance <= distanceThreshold) {
    servo.write(openAngle);
    delay(3500);
} else {
    servo.write(closeAngle);
    delay(1000);
}
```

#### 4. Distance Reading Function

This function, `readDistance()`, is what actually interacts with the ultrasonic sensor. It sends a pulse and waits for an echo. The time taken for the echo is then used to calculate the distance between the sensor and any object in front of it.

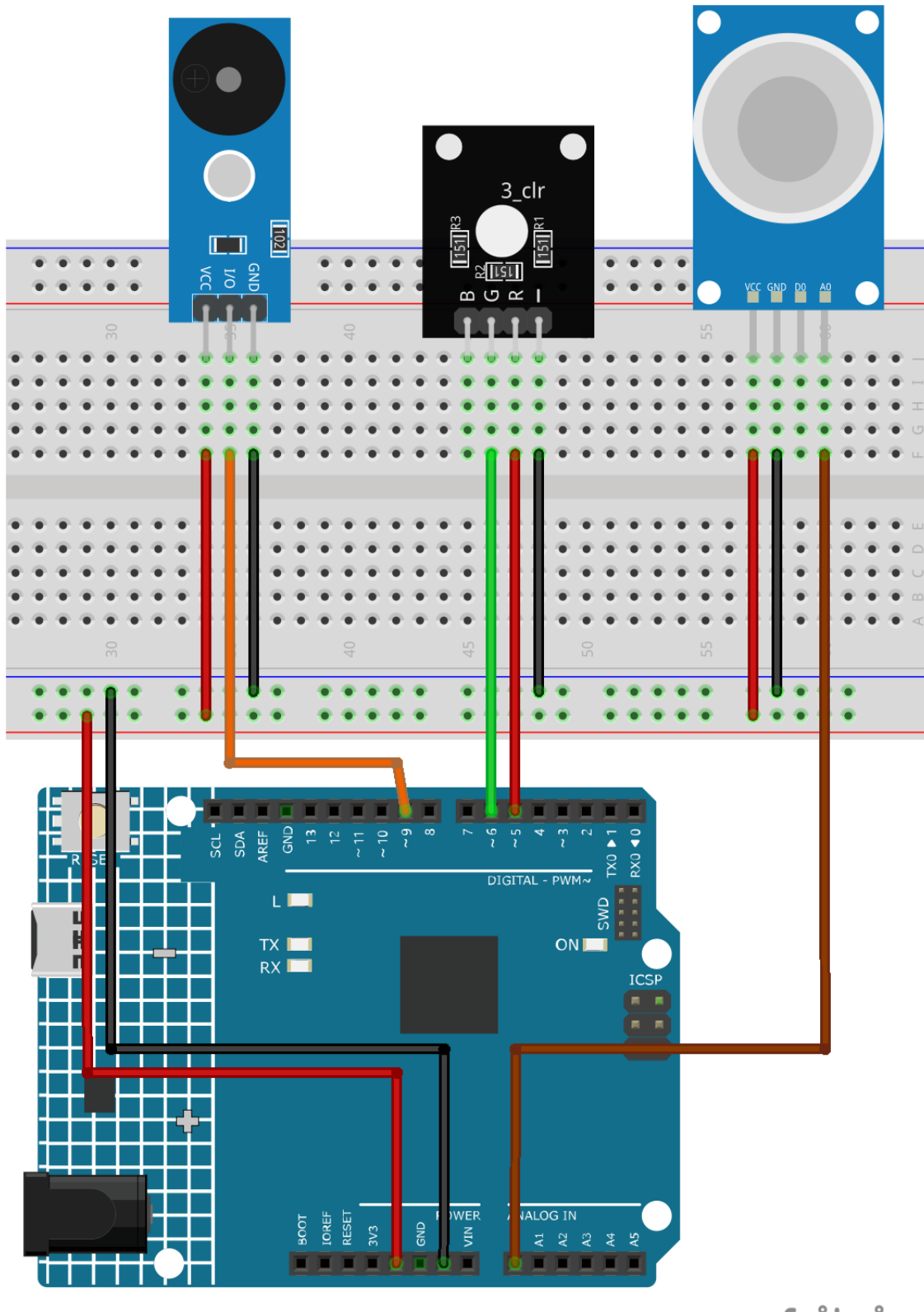
You can refer to the ultrasonic sensor principle in *Principle*.

```
float readDistance() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    float distance = pulseIn(echoPin, HIGH) / 58.00;
    return distance;
}
```

### 2.6.2 Gas leak alarm

This project revolves around simulating a gas leak detection scenario using an Arduino Uno board. By incorporating an MQ-2 gas sensor and an RGB LED, this demonstration continuously reads the gas concentration. If this concentration surpasses a predefined threshold, it activates an alarm (buzzer) and illuminates the RGB LED in red. Conversely, if the concentration remains below this threshold, the alarm remains inactive and the LED shines green. It's crucial to note that this demo is purely illustrative and shouldn't replace real gas leak detection systems.

## 1. Build the Circuit



- *Arduino UNO R4 Minima Board*



- *Gas/Smoke Sensor Module (MQ2)*
- *RGB Module*
- *Passive Buzzer Module*

## 2. Code

1. Open the 02-Gas\_leak\_alarm.ino file under the path of ultimate-sensor-kit\fun\_project\02-Gas\_leak\_alarm, or copy this code into **Arduino IDE**.

## 3. Code explanation

The core principle of the project revolves around continuously monitoring the gas concentration. When the detected gas concentration surpasses a certain threshold, it sets off an alarm and changes the LED's color to red. This serves as a simulated warning mechanism, indicative of potentially hazardous conditions. If the concentration drops below the threshold, the alarm is deactivated and the LED switches to green, indicating a safe environment.

### 1. Defining Constants and Variables

These lines declare and initialize the pin numbers for various components. The `sensorPin` denotes the analog pin where the MQ-2 gas sensor is connected. `sensorValue` is an integer variable storing the sensor's analog output. The `buzzerPin` indicates the digital pin to which the buzzer is connected. Finally, the `RPin` and `GPin` are the pins for the red and green channels of the RGB LED, respectively.

```
// Define the pin numbers for the Gas Sensor
const int sensorPin = A0;
int sensorValue;

// Define the pin number for the buzzer
const int buzzerPin = 9;

// Define pin numbers for the RGB LED
const int RPin = 5; // R channel of RGB LED
const int GPin = 6; // G channel of RGB LED
```

### 2. Initialization in setup()

The `setup()` function initializes the required settings. Serial communication begins at a baud rate of 9600, allowing us to view sensor readings on the Serial Monitor. Pins for the buzzer and RGB LED are set as `OUTPUT`, meaning they'll send signals out to external components.

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

  // Initialize the buzzer and RGB LED pins as output
  pinMode(buzzerPin, OUTPUT);
  pinMode(RPin, OUTPUT);
  pinMode(GPin, OUTPUT);
}
```

### 3. Main Loop: Reading Sensor and Triggering Alarm

The `loop()` function continually reads the gas sensor's output. The reading is then displayed on the Serial Monitor for observation. Depending on the sensor value, two scenarios can occur:

- If the value exceeds 300, the buzzer is activated using `tone()`, and the RGB LED turns red.
- If the value is below 300, the buzzer is silenced using `noTone()`, and the LED turns green.

Lastly, a delay of 50 milliseconds is introduced before the next loop iteration to manage the read frequency and reduce the CPU load.

```
void loop() {  
  // Read the analog value of the gas sensor  
  sensorValue = analogRead(sensorPin);  
  
  // Print the sensor value to the serial monitor  
  Serial.print("Analog output: ");  
  Serial.println(sensorValue);  
  
  // If the sensor value exceeds the threshold, trigger the alarm and make the RGB_  
  ↪ LED red  
  if (sensorValue > 300) {  
    tone(buzzerPin, 500, 300);  
    digitalWrite(GPin, LOW);  
    digitalWrite(RPin, HIGH);  
  } else {  
    // If the sensor value is below the threshold, turn off the alarm and make the_  
    ↪ RGB LED green  
    noTone(buzzerPin);  
    digitalWrite(RPin, LOW);  
    digitalWrite(GPin, HIGH);  
  }  
  
  // Wait for 50 milliseconds before the next loop iteration  
  delay(50);  
}
```

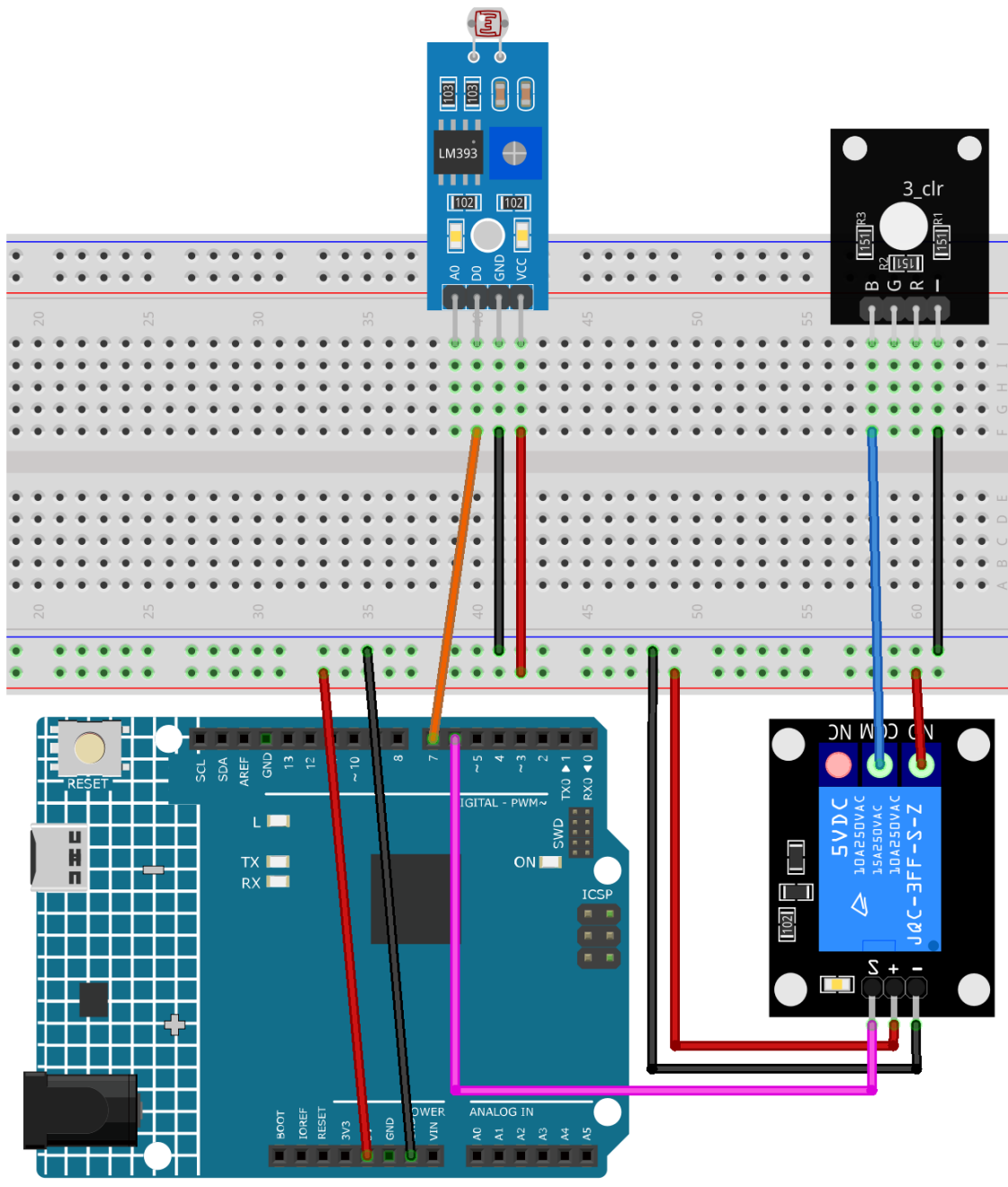
### 2.6.3 Light control switch

This project is a light control switch system. The main idea is to use a photoresistance sensor module to detect the ambient light level and, based on this detection, control a relay module. If the ambient light is below a certain threshold, the relay is switched on. Conversely, if the ambient light is above the threshold, the relay is switched off.

**Warning:** As a demonstration, we are using a relay to control an RGB LED module. However, in real-life scenarios, this may not be the most practical approach.

**While you can connect the relay to other appliances in actual applications, extreme caution is required when dealing with HIGH AC voltage. Improper or incorrect use can lead to severe injury or even death. Therefore, it is intended for people who are familiar with and knowledgeable about HIGH AC voltage. Always prioritize safety.**

## 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *Photoresistor Module*
- *5V Relay Module*
- *RGB Module*

## 2. Code

1. Open the 03-fun\_Light\_control\_switch.ino file under the path of ultimate-sensor-kit\fun\_project\03-fun\_Light\_control\_switch, or copy this code into **Arduino IDE**.

## 3. Code explanation

The primary principle behind this project is the use of a photoresistance sensor to detect ambient light levels. Photoresistors change their resistance based on the light falling on them. This property is utilized in the sensor module to give a digital output. When the light is below the set threshold, the sensor sends a HIGH signal to the Arduino. This signal is then used to activate a relay, which can control other devices.

---

**Note:** The photoresistance sensor has a potentiometer (a small adjustable knob) that sets the threshold for when it outputs HIGH vs. LOW. This threshold might need to be adjusted based on the desired light levels for switching.

---

### 1. Setting up constants and defining pins

Here, we are defining the pins we will use for the relay and the sensor. We use the `const` keyword because these pin numbers won't change throughout the program.

```
const int RelayPin = 6;
const int sensorPin = 7;
```

### 2. Initialization in the setup() function

The `setup()` function is executed once when the program starts. Here, we declare the `RelayPin` as an output since we will be sending signals to control the relay. We also start Serial communication at 9600 baud for debugging purposes.

```
void setup() {
  // Set RelayPin as an output pin
  pinMode(RelayPin, OUTPUT);
  // Start the Serial communication for debugging
  Serial.begin(9600);
}
```

### 3. Reading Sensor and Controlling Relay

The `loop()` function is where the main logic resides. It repeatedly checks the value from the photoresistance sensor. If the sensor reads a value of 1 (indicative of light level below the threshold), the relay is turned on by setting `RelayPin` to HIGH. Otherwise, the relay is turned off by setting `RelayPin` to LOW.

```
void loop() {
  // Read the value from the photoresistance sensor module
  const int sensorValue = digitalRead(sensorPin);
  // If the light level is lower than the threshold (sensor value equals 1),
  // switch the relay module ON.
  if (sensorValue == 1) {
    digitalWrite(RelayPin, HIGH);
  } else
  // If the light level is higher than the threshold (sensor value equal 0),
  // switch the relay module OFF.
  {

```

(continues on next page)

(continued from previous page)

```

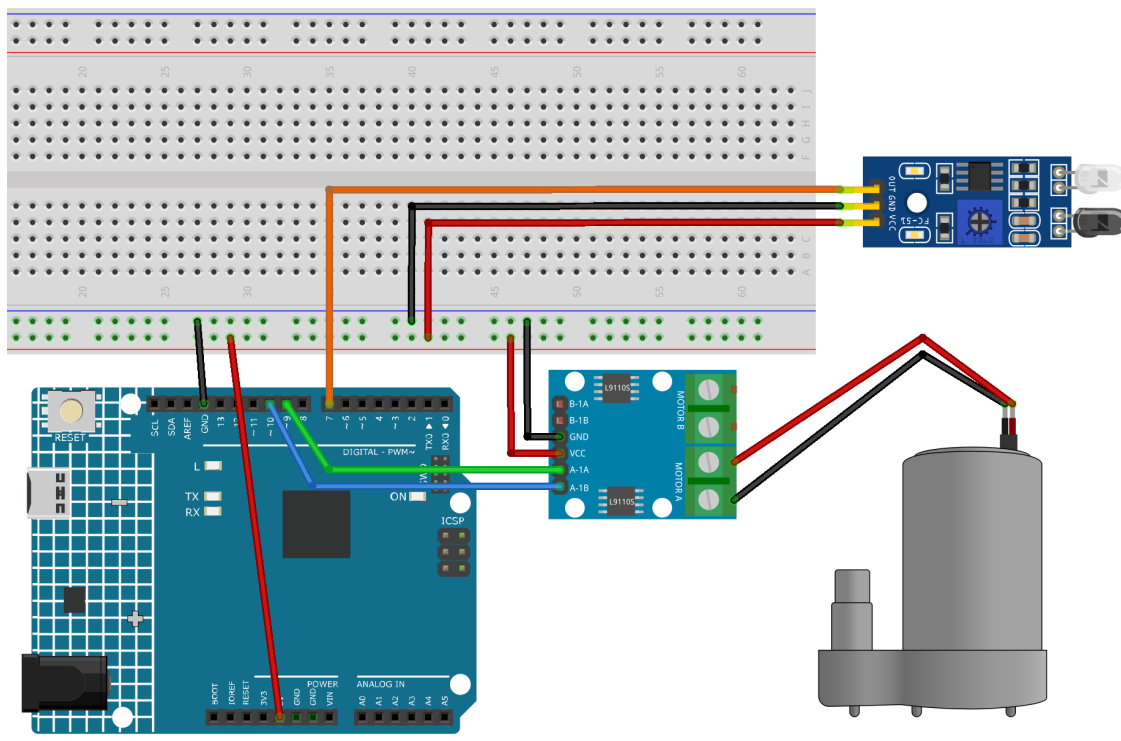
    digitalWrite(RelayPin, LOW);
  }
}

```

## 2.6.4 Automatic soap dispenser

The Automatic Soap Dispenser project uses an Arduino Uno board along with an infrared obstacle avoidance sensor and a water pump. The sensor detects the presence of an object such as a hand, which activates the water pump to dispense soap.

### 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *IR Obstacle Avoidance Sensor Module*
- *Centrifugal Pump*

## 2. Code

1. Open the 04-Automatic\_soap\_dispenser.ino file under the path of ultimate-sensor-kit\fun\_project\04-Automatic\_soap\_dispenser, or copy this code into **Arduino IDE**.

## 3. Code explanation

The main idea behind this project is to create a hands-free soap dispensing system. The infrared obstacle avoidance sensor detects when an object (like a hand) is close. Upon detecting an object, the sensor sends a signal to the Arduino, which in turn triggers the water pump to dispense soap. The pump stays active for a brief period, dispensing soap, then turns off.

### 1. Defining the pins for the sensor and the pump

In this code snippet, we define the Arduino pins that connect to the sensor and pump. We define pin 7 as the sensor pin and we will use the variable `sensorValue` to store the data read from this sensor. For the water pump, we use two pins, 9 and 10.

```
const int sensorPin = 7;
int sensorValue;
const int pump1A = 9;
const int pump1B = 10;
```

### 2. Setting up the sensor and pump

In the `setup()` function, we define the modes for the pins we're using. The sensor pin is set to `INPUT` as it will be used to receive data from the sensor. The pump pins are set to `OUTPUT` as they will send commands to the pump. We ensure that the pin `pump1B` starts in a `LOW` state (off), and we start the serial communication with a baud rate of 9600.

```
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(pump1A, OUTPUT);
  pinMode(pump1B, OUTPUT);
  digitalWrite(pump1B, LOW);
  Serial.begin(9600);
}
```

### 3. Continuously checking the sensor and controlling the pump

In the `loop()` function, the Arduino constantly reads the value from the sensor using `digitalRead()` and assigns it to `sensorValue()`. It then prints this value to the serial monitor for debugging purposes. If the sensor detects an object, `sensorValue()` will be 0. When this happens, `pump1A` is set to `HIGH`, activating the pump, and a delay of 700 milliseconds allows the pump to dispense soap. The pump is then deactivated by setting `pump1A` to `LOW`, and a 1-second delay gives the user time to move their hand away before the cycle repeats.

```
void loop() {
  sensorValue = digitalRead(sensorPin);
  Serial.println(sensorValue);
  if (sensorValue == 0) {
    digitalWrite(pump1A, HIGH);
    delay(700);
    digitalWrite(pump1A, LOW);
    delay(1000);
  }
}
```

(continues on next page)

(continued from previous page)

```
}  
}
```

### 2.6.5 Motion triggered relay

This Arduino project aims to control a relay-operated light using a passive infrared (PIR) sensor. When the PIR sensor detects motion, the relay is activated, turning the light on. The light remains on for 5 seconds after the last detected motion.

**Warning:** As a demonstration, we are using a relay to control an RGB LED module. However, in real-life scenarios, this may not be the most practical approach.

**While you can connect the relay to other appliances in actual applications, extreme caution is required when dealing with HIGH AC voltage. Improper or incorrect use can lead to severe injury or even death. Therefore, it is intended for people who are familiar with and knowledgeable about HIGH AC voltage. Always prioritize safety.**





## 2. Code

1. Open the 05-Motion\_triggered\_relay.ino file under the path of ultimate-sensor-kit\fun\_project\05-Motion\_triggered\_relay, or copy this code into **Arduino IDE**.

## 3. Code explanation

The project revolves around the PIR motion sensor's capability to detect motion. When motion is detected, a signal is sent to the Arduino, triggering the relay module, which in turn activates a light. The light stays on for a specified duration (in this case, 5 seconds) after the last detected motion, ensuring the area remains illuminated for a short period even if motion ceases.

### 1. Initial setup and variable declarations

This segment defines constants and variables that will be used throughout the code. We set up the relay and PIR pins and a delay constant for motion. We also have a variable to keep track of the last detected motion time and a flag to monitor if motion is detected.

```
// Define the pin number for the relay
const int relayPin = 9;

// Define the pin number for the PIR sensor
const int pirPin = 8;

// Motion delay threshold in milliseconds
const unsigned long MOTION_DELAY = 5000;

unsigned long lastMotionTime = 0; // Timestamp of the last motion detection
bool motionDetected = false;    // Flag to track if motion is detected
```

### 2. Configuration of pins in setup() function

In the setup() function, we configure the pin modes for both the relay and PIR sensor. We also initialize the relay to be off at the start.

```
void setup() {
  pinMode(relayPin, OUTPUT); // Set relayPin as an output pin
  pinMode(pirPin, INPUT);   // Set the PIR pin as an input
  digitalWrite(relayPin, LOW); // Turn off the relay initially
}
```

### 3. Main logic in loop() function

The loop() function contains the primary logic. When the PIR sensor detects motion, it sends a HIGH signal, turning on the relay and updating the lastMotionTime. If there's no motion for the specified delay (5 seconds in this case), the relay is turned off.

This approach ensures that even if motion is sporadic or brief, the light remains on for at least 5 seconds after the last detected motion, providing a consistent illumination duration.

```
void loop() {
  if (digitalRead(pirPin) == HIGH) {
    lastMotionTime = millis(); // Update the last motion time
    digitalWrite(relayPin, HIGH); // Turn on the relay (and hence the light)
    motionDetected = true;
  }
}
```

(continues on next page)

(continued from previous page)

```
}  
  
// If motion was detected earlier and 5 seconds have elapsed, turn off the relay  
if (motionDetected && (millis() - lastMotionTime >= MOTION_DELAY)) {  
    digitalWrite(relayPin, LOW); // Turn off the relay  
    motionDetected = false;  
}  
}
```

### 2.6.6 Heart rate monitor

This Arduino project aims to build a simple Heart Rate Monitor using a MAX30102 pulse oximeter sensor and an SSD1306 OLED Display. The code takes measurements of the heart rate by determining the time between heartbeats. By taking four measurements, it computes their average and presents the resultant average heart rate on an OLED screen. If the sensor doesn't detect a finger, it sends a prompt to the user to position their finger correctly on the sensor.



## 2. Code

1. Open the 06-Heart\_rate\_monitor.ino file under the path of ultimate-sensor-kit\fun\_project\06-Heart\_rate\_monitor, or copy this code into **Arduino IDE**.

---

**Note:** To install the library, use the Arduino Library Manager and search for “**SparkFun MAX3010x**” and install it.

---

## 3. Code explanation

The main principle behind this project is to capture the pulsation of blood flow through a finger using the MAX30102 sensor. As blood pumps through the body, it causes tiny changes in the volume of blood in the vessels of the fingertip. By shining light through the finger and measuring the amount of light that gets absorbed or reflected back, the sensor detects these minute volume changes. The time interval between subsequent pulses is then used to calculate the heart rate in beats per minute (BPM). This value is then averaged over four measurements and displayed on the OLED screen.

### 1. Library Inclusions and Initial Declarations:

The code begins by including necessary libraries for the OLED display, MAX30102 sensor, and heart rate calculation. Additionally, the configuration for the OLED display and the variables for heart rate calculation are declared.

---

**Note:** To install the library, use the Arduino Library Manager and search for “**SparkFun MAX3010x**” and install it.

---

```
#include <Adafruit_GFX.h> // OLED libraries
#include <Adafruit_SSD1306.h>
#include <Wire.h>
#include "MAX30105.h" // MAX3010x library
#include "heartRate.h" // Heart rate calculating algorithm

// ... Variables and OLED configuration
```

In this project, we’ve also whipped up a couple of bitmaps. The PROGMEM keyword denotes that the array is stored in the program memory of the Arduino microcontroller. Storing data in program memory (PROGMEM) instead of RAM can be helpful for large amounts of data, which would otherwise take up too much space in RAM.

```
static const unsigned char PROGMEM beat1_bmp[] = {...}

static const unsigned char PROGMEM beat2_bmp[] = {...}
```

### 2. Setup Function:

Initializes I2C communication, starts serial communication, initializes the OLED display, and sets up the MAX30102 sensor.

```
void setup() {
  Wire.setClock(400000);
  Serial.begin(9600);
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  // ... Rest of the setup code
```

### 3. Main Loop:

The core functionality resides here. The IR value is read from the sensor. If a finger is detected (IR value greater than 50,000), the program checks if a heartbeat is sensed. When a heartbeat is detected, the OLED screen displays the BPM and the time between heartbeats is used to calculate BPM. Otherwise, it prompts the user to place their finger on the sensor.

We have also prepared two bitmaps with heartbeats, and by switching between these two bitmaps, we can achieve a dynamic visual effect.

```
void loop() {
  // Get IR value from sensor
  long irValue = particleSensor.getIR();

  //If a finger is detected
  if (irValue > 50000) {

    // Check if a beat is detected
    if (checkForBeat(irValue) == true) {

      // Update OLED display
      // Calculate the BPM

      // Calculate the average BPM
      //Print the IR value, current BPM value, and average BPM value to the serial_
      ↪monitor

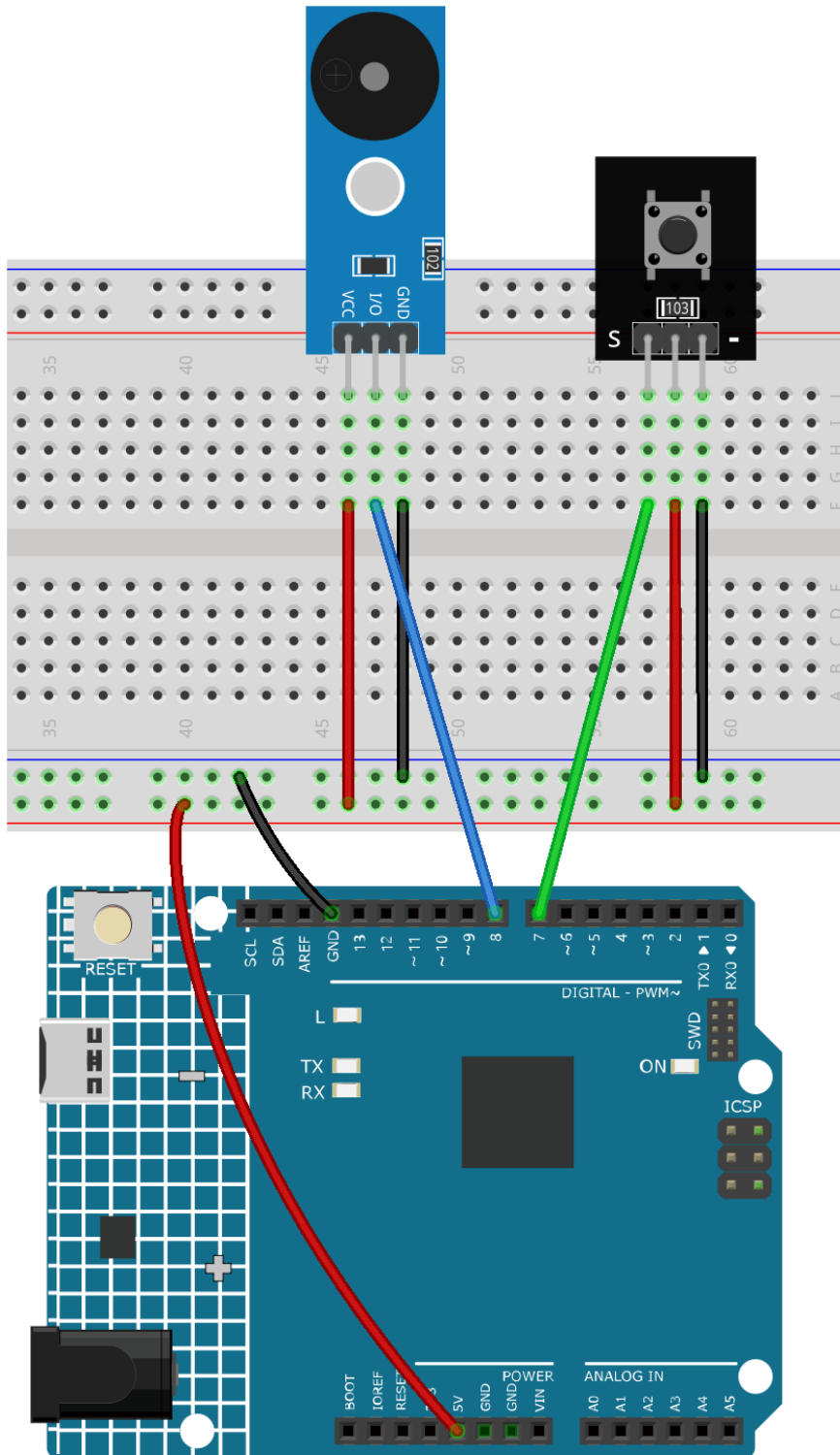
      // Update OLED display

    }
  }
  else {
    // ... Prompt to place the finger on the sensor
  }
}
```

#### 2.6.7 Doorbell

The “doorbell” project aims to simulate the function of a doorbell. When a button is pressed, the Arduino plays a predefined melody using a passive buzzer module.

## 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *Button Module*

- *Passive Buzzer Module*

## 2. Code

1. Open the 07-Doorbell.ino file under the path of ultimate-sensor-kit\fun\_project\07-Doorbell, or copy this code into **Arduino IDE**.

## 3. Code explanation

The main idea behind this project is to use the Arduino Uno board to detect a button press and, in response, play a melody on the passive buzzer. The melody consists of a sequence of notes (defined by their pitches) and their durations.

1. Setting up necessary libraries and global variables

```
#include "pitches.h" // This library provides the frequency values for musical
notes.

const int buttonPin = 7; // Button connected to digital pin 7
const int buzzerPin = 8; // Buzzer connected to digital pin 8

// Arrays to define the melody and the corresponding note durations
int melody[] = {...};
int noteDurations[] = {...};
```

This segment includes the necessary library for musical notes and sets up the pins for our components. Additionally, the melody and its durations are defined in arrays.

2. Initializing the button and starting serial communication

```
void setup() {
  Serial.begin(9600); // Start serial communication at 9600 baud rate
  pinMode(buttonPin, INPUT); // Set the button pin as an input
}
```

In the setup() function, we start serial communication and configure the buttonPin to act as an input.

3. Monitoring the button press to play the melody

```
void loop() {
  int buttonState = digitalRead(buttonPin); // Read the state of the button

  if (buttonState == LOW) { // Check if the button is pressed
    Serial.println("Button pressed"); // Send a message to serial monitor
    buzzer(); // Play the buzzer melody
  }
}
```

Here, we continuously check the state of the button in the loop. If pressed, a message is sent to the serial monitor, and the buzzer() function (which plays the melody) is called.

4. The buzzer() function to play the melody

```
void buzzer() {
  int size = sizeof(noteDurations) / sizeof(int); // Calculate the number of notes
```

(continues on next page)

(continued from previous page)

```
for (int thisNote = 0; thisNote < size; thisNote++) {  
  int noteDuration = 1000 / noteDurations[thisNote];    // Calculate note's play_  
  ↳duration  
  tone(buzzerPin, melody[thisNote], noteDuration);      // Play the note on the_  
  ↳buzzer  
  
  int pauseBetweenNotes = noteDuration * 1.30;          // Calculate pause between_  
  ↳notes  
  delay(pauseBetweenNotes);                             // Introduce the pause  
  noTone(buzzerPin);                                     // Stop playing the note  
}  
}
```

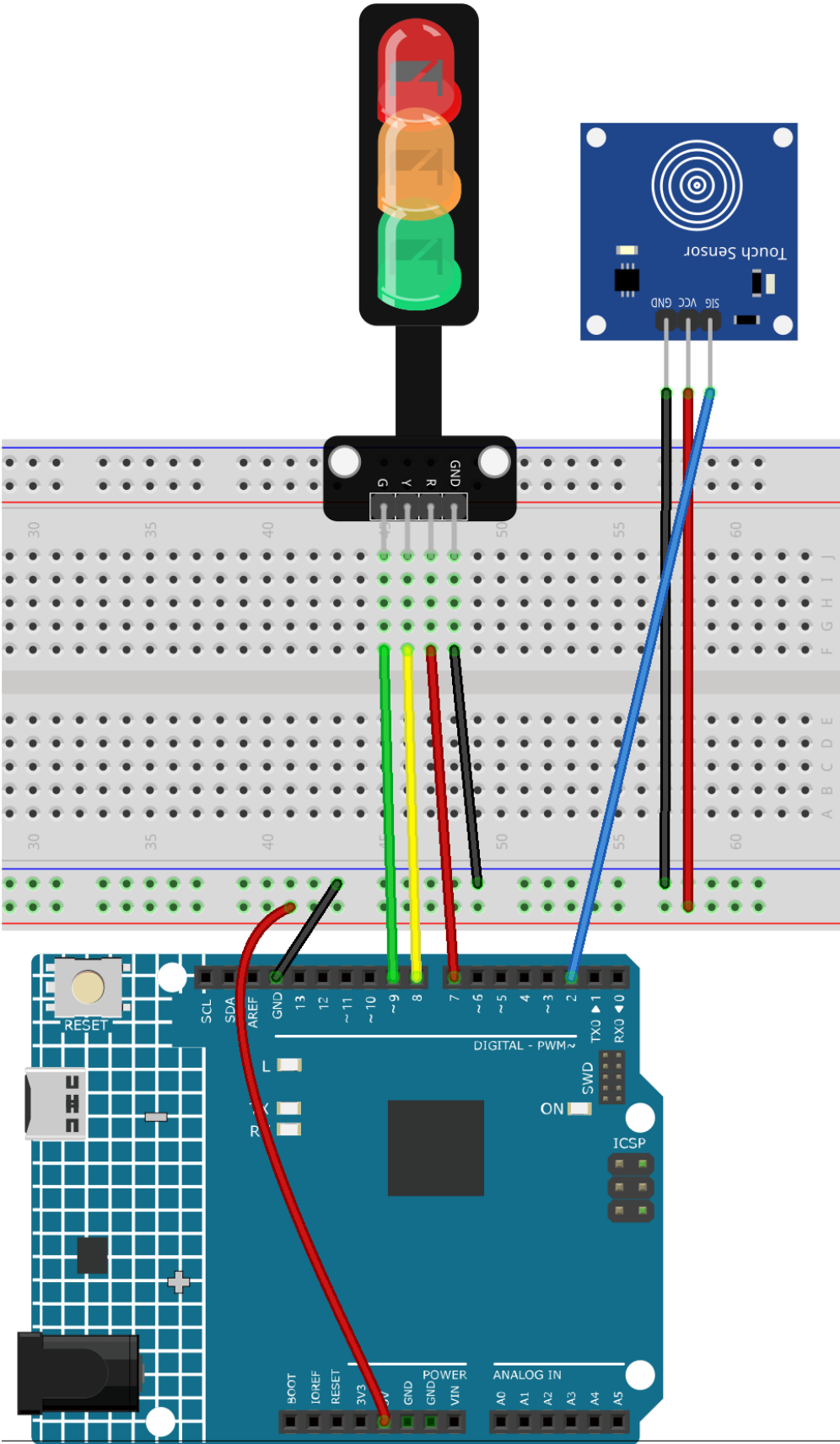
In the `buzzer()` function, the melody's notes are played sequentially. The `tone()` function generates a tone on the buzzer for a specified duration. After playing each note, there's a brief pause before playing the next note.

### 2.6.8 Touch toggle light

The project involves creating a simple traffic light control mechanism using a touch sensor and a traffic light LED module. When the touch sensor is activated, the LEDs will cycle through the following sequence: Red -> Yellow -> Green.



1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *Touch Sensor Module*
- *Traffic Light Module*

## 2. Code

1. Open the 08-Touch\_toggle\_light.ino file under the path of ultimate-sensor-kit\fun\_project\08-Touch\_toggle\_light, or copy this code into **Arduino IDE**.

## 3. Code explanation

This project operates on a simple principle: when a touch is detected on the touch sensor, the next LED in the sequence (Red -> Yellow -> Green) will light up. The state of which LED is currently active is managed by the variable `currentLED`.

1. Define pins and initial values

```
const int touchSensorPin = 2; // touch sensor pin
const int rledPin = 9;        // red LED pin
const int yledPin = 8;        // yellow LED pin
const int gledPin = 7;        // green LED pin
int lastTouchState;           // the previous state of touch sensor
int currentTouchState;        // the current state of touch sensor
int currentLED = 0;           // current LED 0->Red, 1->Yellow, 2->Green
```

These lines define the pins that we connect the components to on the Arduino board and initialize the states for touch and LEDs.

2. `setup()` function

```
void setup() {
  Serial.begin(9600);           // initialize serial
  pinMode(touchSensorPin, INPUT); // configure touch sensor pin as input
  // set LED pins as outputs
  pinMode(rledPin, OUTPUT);
  pinMode(yledPin, OUTPUT);
  pinMode(gledPin, OUTPUT);
  currentTouchState = digitalRead(touchSensorPin);
}
```

This function runs once when the Arduino is powered on or reset. Here, the touch sensor is set as an input (it will read values), while the LEDs are set as outputs (we will set their values). Also, the serial communication is started to allow debugging, and the initial touch state is read.

3. `loop()` function

```
void loop() {
  lastTouchState = currentTouchState; // save the last state
  currentTouchState = digitalRead(touchSensorPin); // read new state
  if (lastTouchState == LOW && currentTouchState == HIGH) {
    Serial.println("The sensor is touched");
    turnAllLEDsOff(); // Turn off all LEDs
    // switch on the next LED in sequence
  }
```

(continues on next page)

(continued from previous page)

```
switch (currentLED) {  
  case 0:  
    digitalWrite(rledPin, HIGH);  
    currentLED = 1;  
    break;  
  case 1:  
    digitalWrite(yledPin, HIGH);  
    currentLED = 2;  
    break;  
  case 2:  
    digitalWrite(gledPin, HIGH);  
    currentLED = 0;  
    break;  
}  
}
```

In the main loop, the current touch state is read and compared with the previous one. If a touch is detected (transition from LOW to HIGH), all LEDs are turned off, and the next one in the sequence is turned on.

#### 4. Turn off LEDs function

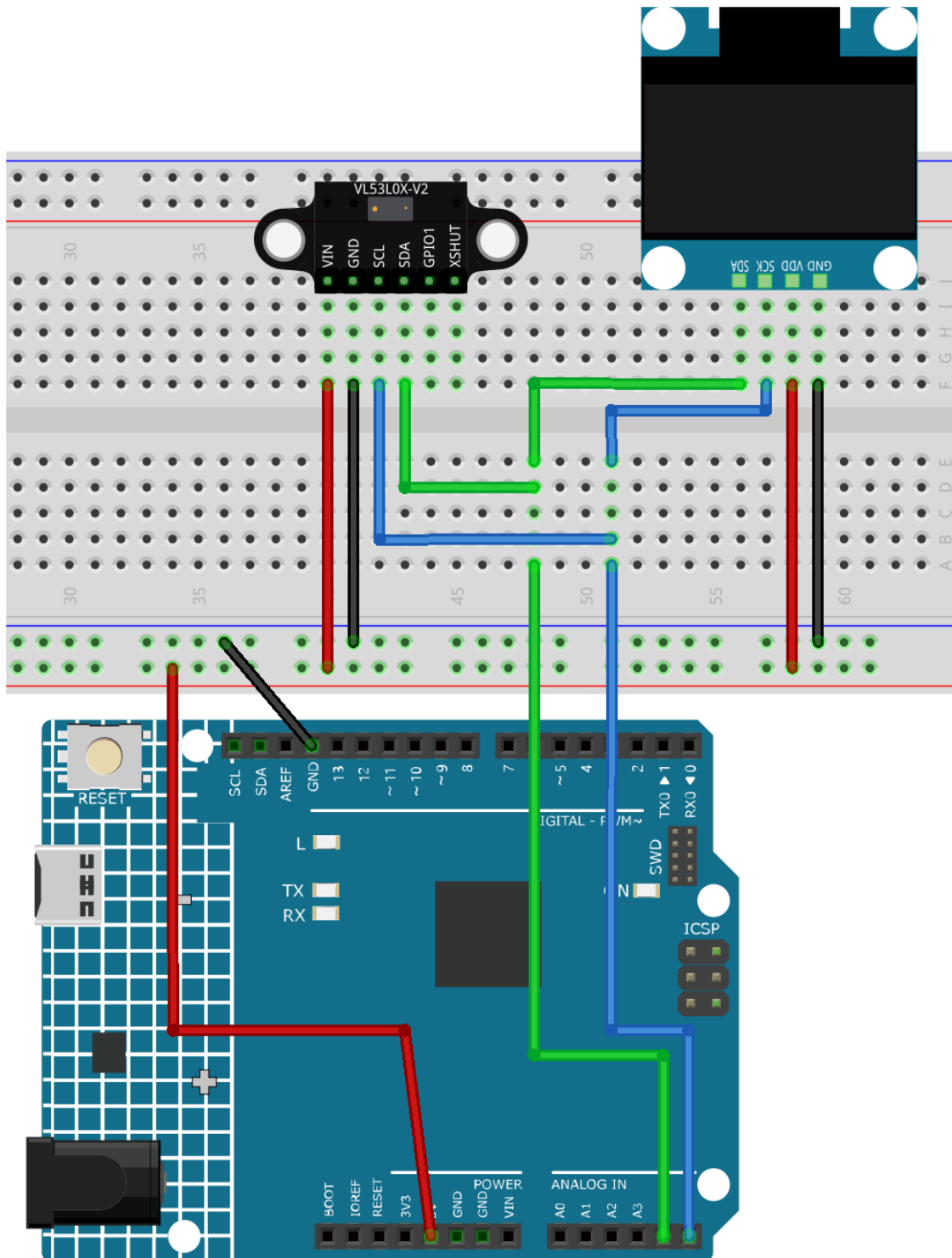
```
void turnAllLEDsOff() {  
  digitalWrite(rledPin, LOW);  
  digitalWrite(yledPin, LOW);  
  digitalWrite(gledPin, LOW);  
}
```

This function, when called, will turn off all the LEDs by setting their pins to LOW.

### 2.6.9 ToF distance monitor

This project is designed to measure and display the distance to an object using the VL53L0X Time of Flight (ToF) Micro-LIDAR Distance Sensor. The measured distance in millimeters is displayed on an OLED screen, and the values are also output to the serial monitor. The VL53L0X can measure a range of approximately 50mm to 1200mm.

## 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *Time of Flight Micro-LIDAR Distance Sensor (VL53L0X)*
- *OLED Display Module*

## 2. Code

1. Open the 09-ToF\_distance\_monitor.ino file under the path of ultimate-sensor-kit\fun\_project\09-ToF\_distance\_monitor, or copy this code into **Arduino IDE**.

---

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit\_VL53L0X**” and install it.

---

## 3. Code explanation

This project uses the VL53L0X Time of Flight sensor to measure distances by measuring the time it takes for light to travel to an object and return to the sensor. The OLED display then shows the distance measurement in millimeters. Serial communication is also used to print the measurement values for monitoring and debugging. Both the OLED display and the VL53L0X sensor communicate with the Arduino using the I2C protocol.

1. Include necessary libraries and initialize components

---

**Note:** To install the library, use the Arduino Library Manager and search for “**Adafruit\_VL53L0X**” and install it.

---

```
#include <Wire.h>
#include "Adafruit_VL53L0X.h"
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Initialize the OLED display module with a resolution of 128x64
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 64, &Wire, -1);

// Initialize the VL53L0X distance sensor
Adafruit_VL53L0X lox = Adafruit_VL53L0X();
```

- Necessary libraries for handling I2C communication, the distance sensor, SPI protocol, and the OLED display are included.
- The OLED display and the VL53L0X distance sensor are initialized.

2. Initialize the serial communication and prepare the display as well as the VL53L0X distance sensor.

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

  // Start the OLED display with I2C address 0x3C
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.display();
  delay(1000);

  // Begin I2C communication
  Wire.begin();

  // Start the VL53L0X distance sensor, halt if initialization fails
```

(continues on next page)

(continued from previous page)

```

if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1)
        ;
}

// Set OLED display text size and color
display.setTextSize(3);
display.setTextColor(WHITE);
}

```

- Start serial communication at 9600 baud.
- Initialize the OLED display with its I2C address.
- Begin I2C communication.
- Check if the VL53L0X distance sensor is initialized properly. If not, an error message is displayed, and the Arduino enters an infinite loop.
- Set text size and color for the OLED display.

### 3. Main loop() to measure the distance and display the result.

```

void loop() {
    VL53L0X_RangingMeasurementData_t measure;

    lox.rangingTest(&measure, false); // pass in 'true' to get debug data printout

    // If there are no phase failures, display the measured distance
    if (measure.RangeStatus != 4) {
        display.clearDisplay();
        display.setCursor(12, 22);
        display.print(measure.RangeMilliMeter);
        display.print("mm");
        display.display();
        Serial.println();
        delay(50);
    } else {
        display.display();
        display.clearDisplay();
        return;
    }
}

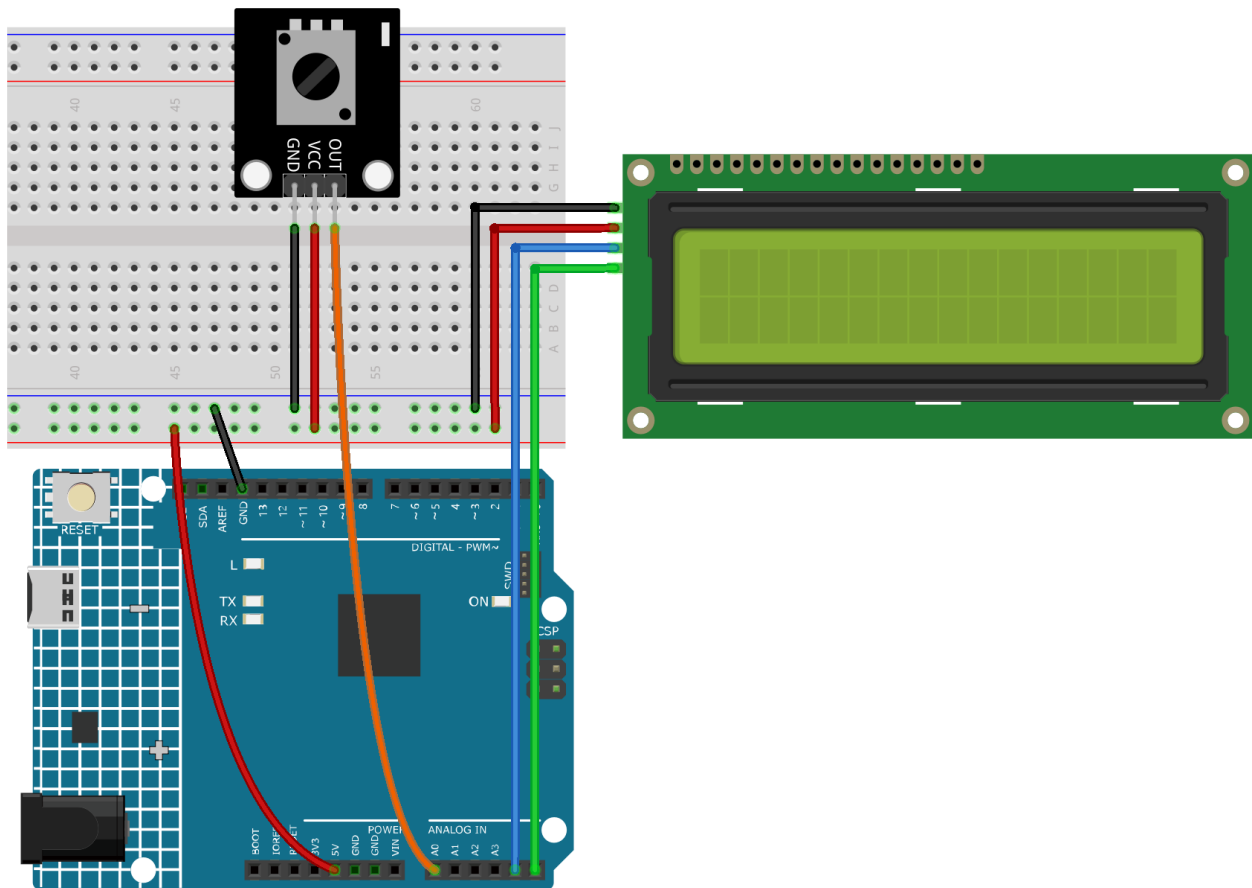
```

- Create a variable to store the measurement data.
- Take a measurement using the VL53L0X sensor.
- Check if the measurement is valid (i.e., no phase failures).
- If the measurement is valid, clear the OLED display, set the cursor position, and display the measured distance.
- Else, refresh the display and clear it for the next reading.

### 2.6.10 Potentiometer scale value

This project is designed to read the value from a potentiometer and display the read value on an LCD 1620 with an I2C interface. The value is also sent to the serial monitor for real-time viewing. A unique feature of this project is the visual representation of the potentiometer's value on the LCD, displaying a bar that varies in length corresponding to the value.

#### 1. Build the Circuit



- *Arduino UNO R4 Minima Board*
- *Potentiometer Module*
- *I2C LCD 1602*

## 2. Code

1. Open the 10-Potentiometer\_scale\_value.ino file under the path of ultimate-sensor-kit\fun\_project\10-Potentiometer\_scale\_value, or copy this code into **Arduino IDE**.

## 3. Code explanation

The project functions by continually reading the value from a connected potentiometer. This value is then mapped to a smaller scale (0-16) and represented both numerically and visually on the LCD. By checking the difference between consecutive readings, the code ensures that only significant changes are reflected on the display, thereby reducing jitter. Reducing jitter helps avoid undesirable visual effects resulting from frequent refreshing of the LCD screen.

### 1. Library Inclusion and Initialization:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

Here, the required libraries (Wire for I2C communication and LiquidCrystal\_I2C for the LCD) are included. An LCD object is created with the I2C address 0x27 and is defined to have 16 columns and 2 rows.

### 2. Variable Declaration:

```
int lastRead = 0;    // Previous potentiometer value
int currentRead = 0; // Current potentiometer value
```

lastRead stores the previously read potentiometer value. currentRead will store the current reading from the potentiometer.

### 3. setup() Function:

```
void setup() {
  lcd.init();           // Initialize the LCD
  lcd.backlight();      // Turn on the LCD backlight
  Serial.begin(9600);   // Start serial communication at 9600 baud rate
}
```

The LCD is initialized, its backlight is turned on, and serial communication is started at a baud rate of 9600.

### 4. Main Loop:

```
void loop() {
  int currentRead = analogRead(A0);
  int barLength = map(currentRead, 0, 1023, 0, 16);
  if (abs(lastRead - currentRead) > 2) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Value:");
    lcd.setCursor(7, 0);
    lcd.print(currentRead);
    Serial.println(currentRead);
    for (int i = 0; i < barLength; i++) {
      lcd.setCursor(i, 1);
      lcd.print(char(255));
    }
  }
}
```

(continues on next page)



(continued from previous page)

```
}  
lastRead = currentRead;  
delay(200);  
}
```

- The potentiometer value is read and mapped to a bar length (0-16).
- If the difference between the last and current reading is more than 2, the LCD is updated.
- The value is printed on the first row and a bar (based on the mapped value) on the second row.
- The value is also sent to the serial monitor.
- Before the next iteration, lastRead is updated, and a delay of 200ms is introduced for stability.

## 2.7 FAQ

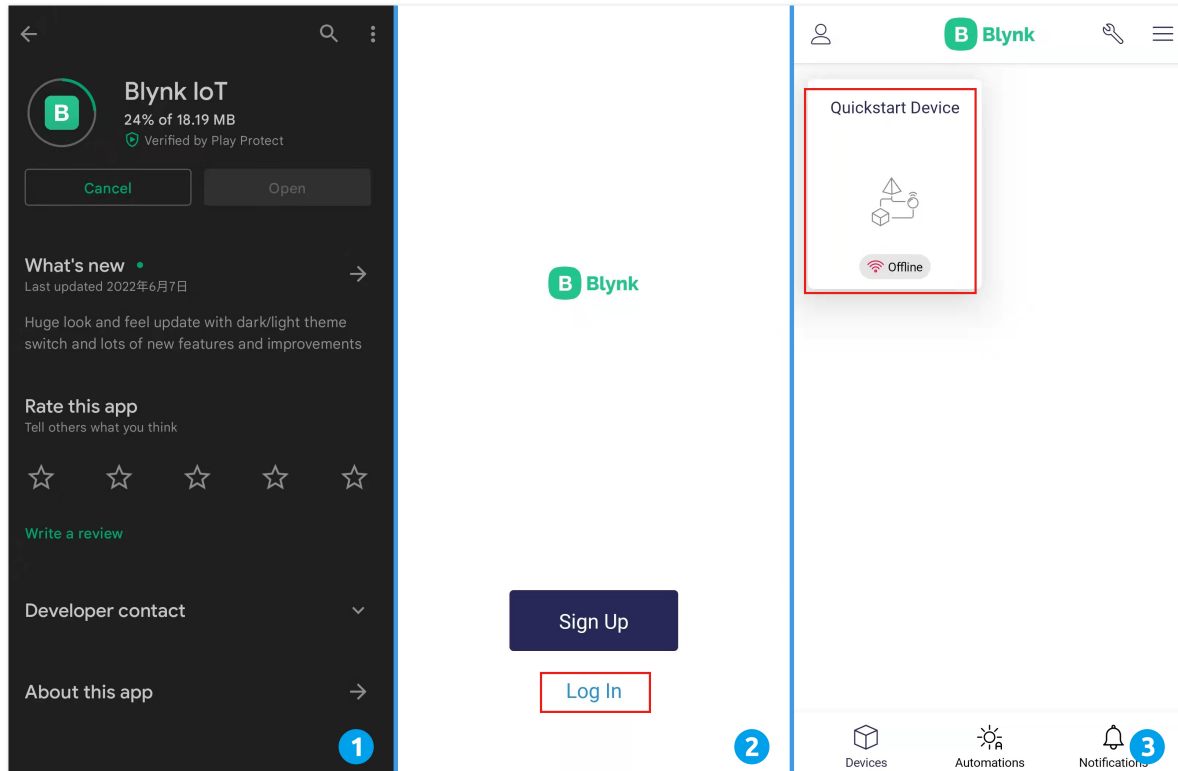
### 2.7.1 How to use Blynk on mobile device?

---

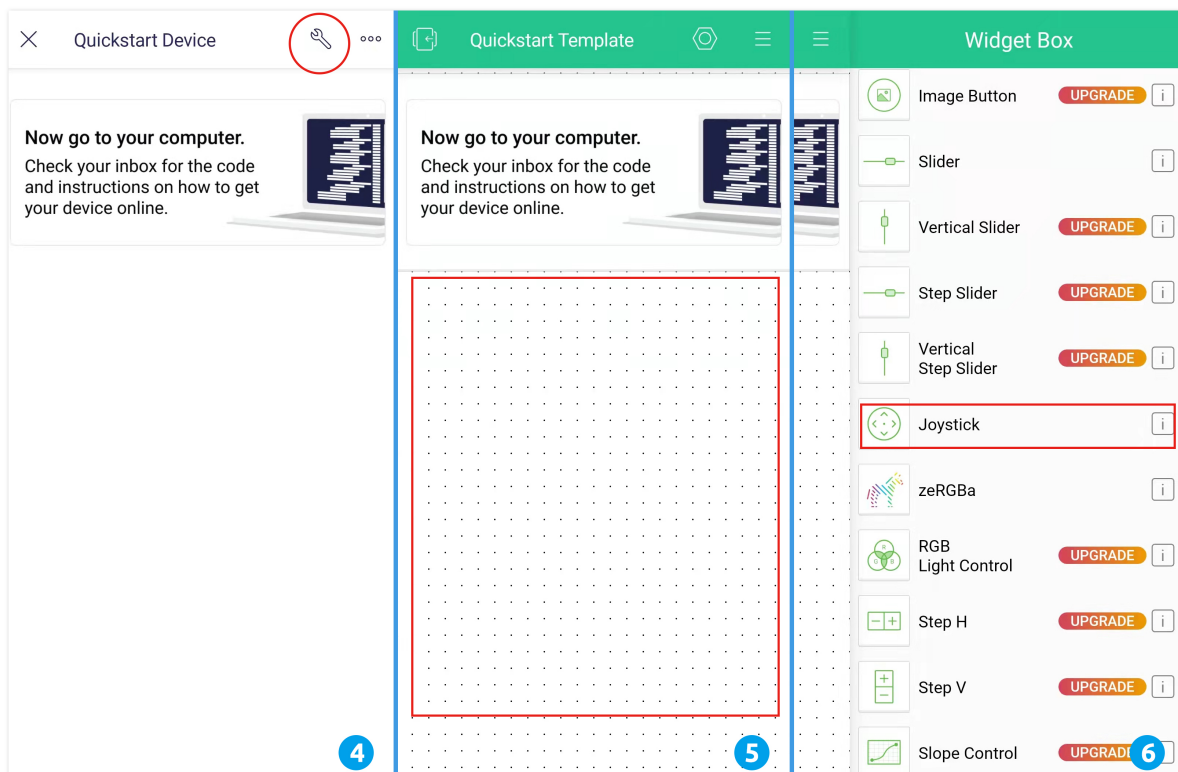
**Note:** As datastreams can only be created in Blynk on the web, you will need to reference different projects to create datastreams on the web, then follow the tutorial below to create widgets in Blynk on your mobile device.

---

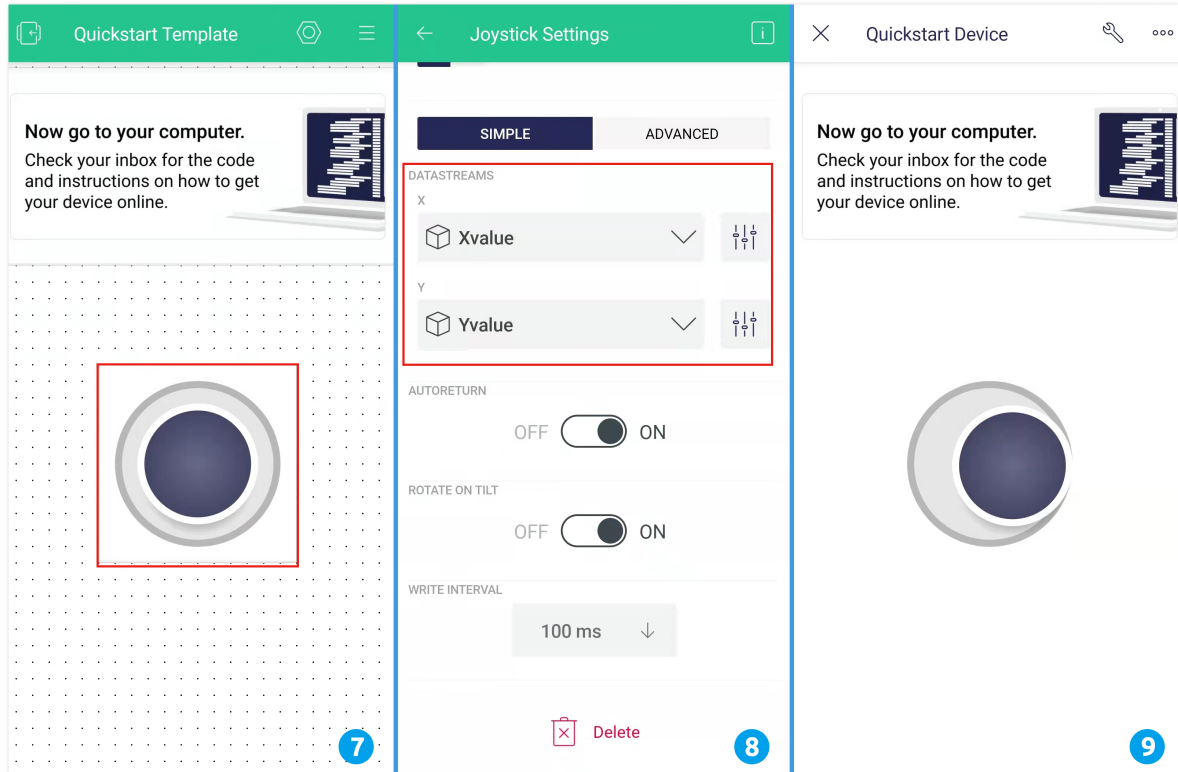
1. Open Google Play or APP Store on your mobile device and search for “Blynk IoT” (not Blynk(legacy)) to download.
2. After opening the APP, login in, this account should be the same as the account used on the web client.
3. Then go to **Dashboard** (if you don’t have one, create one) and you will see that the **Dashboard** for mobile and web are independent of each other.



4. Click **Edit** Icon.
5. Click on the blank area.
6. Choose the same widget as on the web page, such as select a **Joystick** widget.



7. Now you will see a **Joystick** widget appear in the blank area, click on it.
8. **Joystick** Settings will appear, select the **Xvalue** and **Yvalue** datastreams you just set in the web page. Note that each widget corresponds to a different datastream in each project.
9. Go back to the **Dashboard** page and you can operate the **Joystick** when you want.

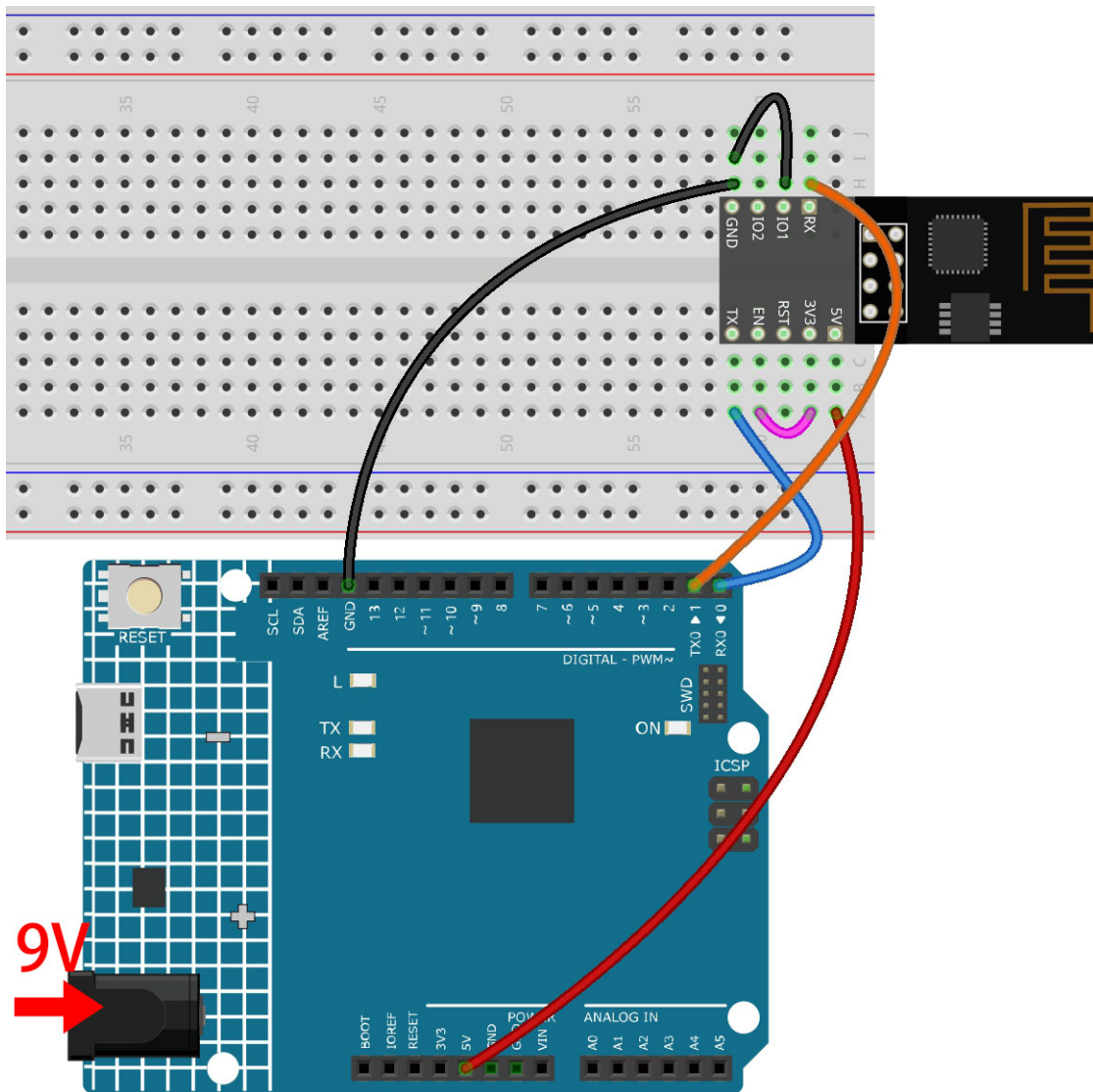


## 2.7.2 How to re-burn the firmware for ESP8266 module?

### Re-burn the Firmware with R4

#### 1. Build the circuit

Connect ESP8266 and Arduino UNO R4 board.



## 2. Upload the Following Code to R4

```
void setup() {
  Serial.begin(115200);
  Serial1.begin(115200);
}

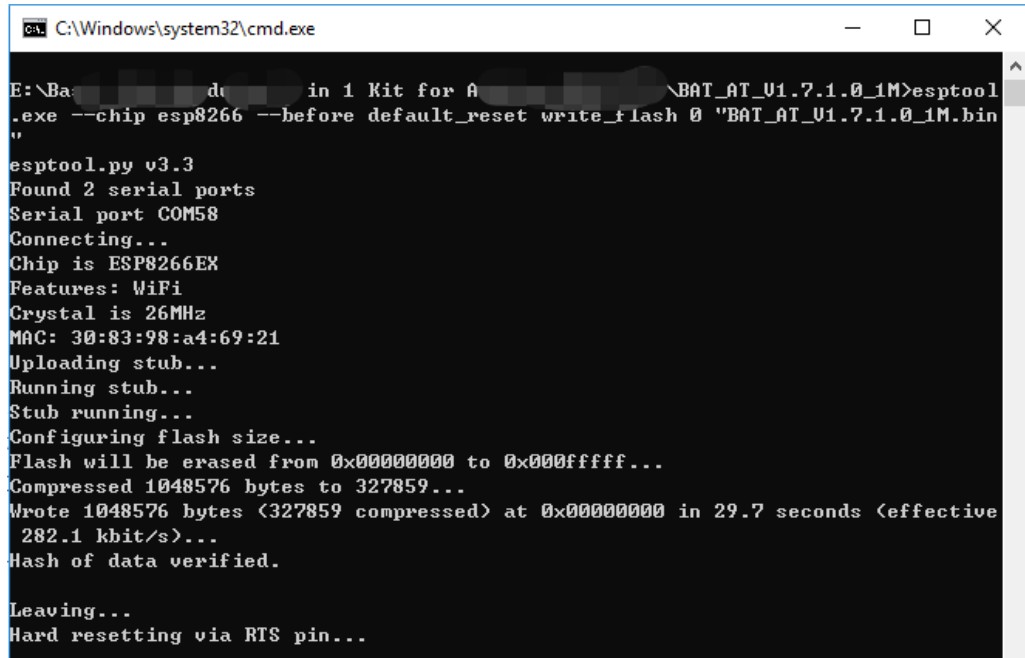
void loop() {
  if (Serial.available()) {      // If anything comes in Serial (USB),
    Serial1.write(Serial.read()); // read it and send it out Serial1 (pins 0 & 1)
  }

  if (Serial1.available()) {    // If anything comes in Serial1 (pins 0 & 1)
    Serial.write(Serial1.read()); // read it and send it out Serial (USB)
  }
}
```

## 3. Burning the firmware

- Follow the steps below to burn the firmware if you are using **Windows**.

1. Download firmware and burn-in tool.
  - ESP8266 Firmware
2. After unzipping, you will see 4 files.
  - BAT\_AT\_V1.7.1.0\_1M.bin: The firmware to burn to the ESP8266 module.
  - esptool.exe: This is a command-line utility for Windows.
  - install\_r3.bat: This is the command package for Windows system, double click this file will run all the commands inside the file.
  - install\_r4.bat: Same as install\_r3.bat, but dedicated to UNO R4 board.
3. Double click install\_r4.bat to start the firmware burning. If you see the following prompt, the firmware has been installed successfully.



```

C:\Windows\system32\cmd.exe

E:\Ba... du... in 1 Kit for A... \BAT_AT_V1.7.1.0_1M>esptool
.exe --chip esp8266 --before default_reset write_flash 0 "BAT_AT_V1.7.1.0_1M.bin
"
esptool.py v3.3
Found 2 serial ports
Serial port COM58
Connecting...
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 30:83:98:a4:69:21
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Flash will be erased from 0x00000000 to 0x000fffff...
Compressed 1048576 bytes to 327859...
Wrote 1048576 bytes (327859 compressed) at 0x00000000 in 29.7 seconds (effective
282.1 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
  
```

**Note:** If the burn-in fails, please check the following points.

- Reset the ESP8266 module by inserting the RST on the ESP8266 Adapter to GND and then unplugging it.
- Check if the wiring is correct.
- Whether the computer has recognized your board properly, and make sure the port is not occupied.
- Reopen the install.bat file.

- To burn the firmware, follow these steps if you are using a **Mac OS** system.

1. Use the following commands to install Esptool. Esptool is a Python-based, open-source, platform-independent utility to communicate with the ROM bootloader in Espressif chips.

```
python3 -m pip install --upgrade pip
python3 -m pip install esptool
```

2. If esptool is properly installed, it will output a message such as [usage: esptool] if you run `python3 -m esptool`.
3. Download firmware.
  - ESP8266 Firmware
4. After unzipping, you will see 4 files.
  - BAT\_AT\_V1.7.1.0\_1M.bin: The firmware to burn to the ESP8266 module.
  - esptool.exe: This is a command-line utility for Windows.
  - install\_r3.bat: This is the command package for Windows system.
  - install\_r4.bat: Same as install\_r3.bat, but dedicated to UNO R4 board.
5. Open a terminal and use the `cd` command to go into the firmware folder you just downloaded, then run the following command to erase the existing firmware and re-burn the new firmware.

```
python3 -m esptool --chip esp8266 --before no_reset_no_sync erase_flash
python3 -m esptool --chip esp8266 --before no_reset_no_sync write_flash
↪ 0 "BAT_AT_V1.7.1.0_1M.bin"
```

6. If you see the following prompt, the firmware has been installed successfully.



```
BAT_AT_V1.7.1.0_1M -- -bash -- 99x22
MacdeMacBook-Pro:BAT_AT_V1.7.1.0_1M mac$ python3 -m esptool --chip esp8266 --before default_reset w
rite_flash 0 "BAT_AT_V1.7.1.0_1M.bin"
esptool.py v4.3
[Found 3 serial ports
Serial port /dev/cu.usbmodem143401
Connecting...
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 30:83:98:a4:69:21
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Flash will be erased from 0x00000000 to 0x000fffff...
Compressed 1048576 bytes to 327859...
Wrote 1048576 bytes (327859 compressed) at 0x00000000 in 29.6 seconds (effective 283.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
MacdeMacBook-Pro:BAT_AT_V1.7.1.0_1M mac$
```

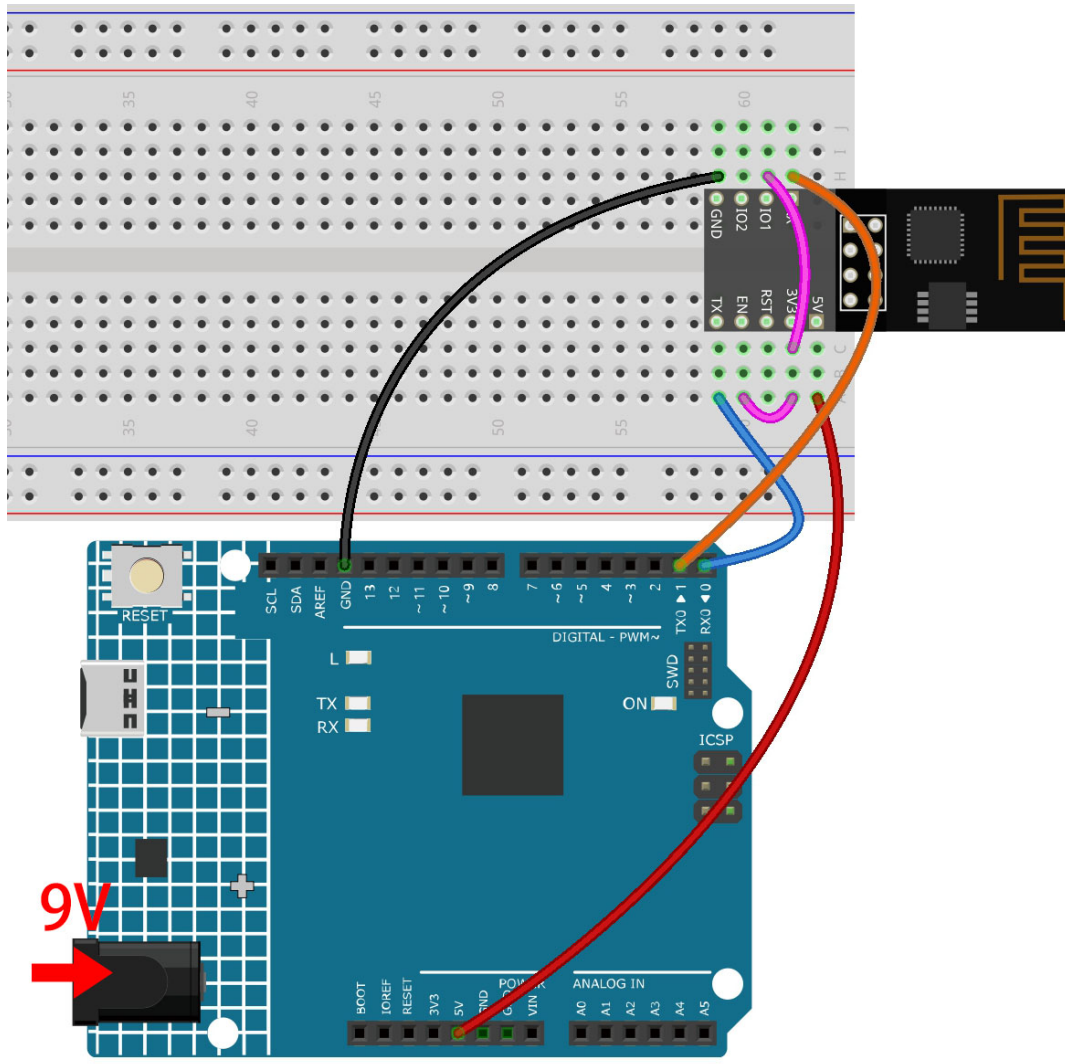
---

**Note:** If the burn-in fails, please check the following points.

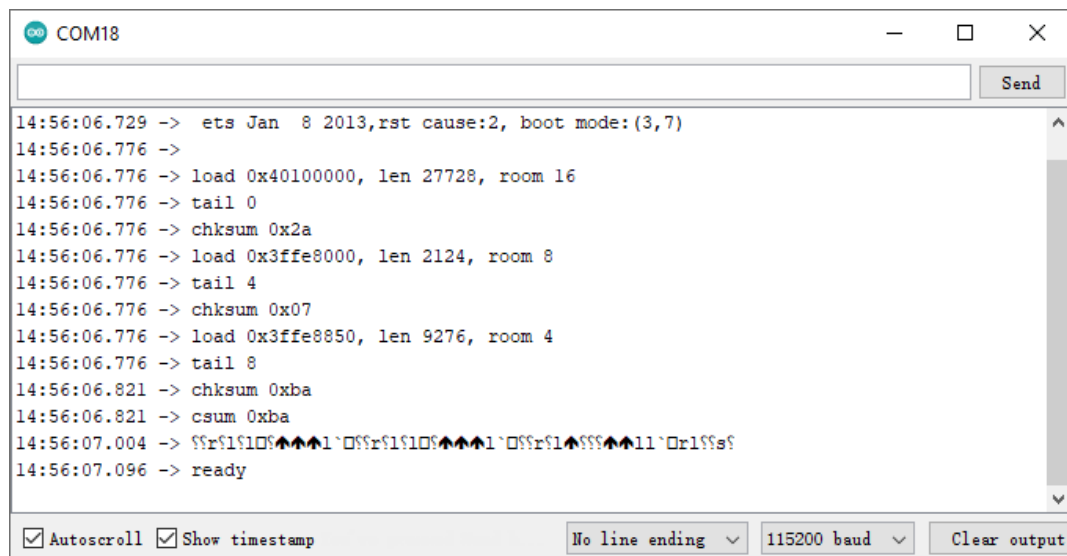
- Reset the ESP8266 module by inserting the RST on the ESP8266 Adapter to GND and then unplugging it.
  - Check if the wiring is correct.
  - Whether the computer has recognized your board properly, and make sure the port is not occupied.
  - Reopen the install.bat file.
- 

## 4. Test

1. On the basis of the original wiring, connect IO1 to 3V3.



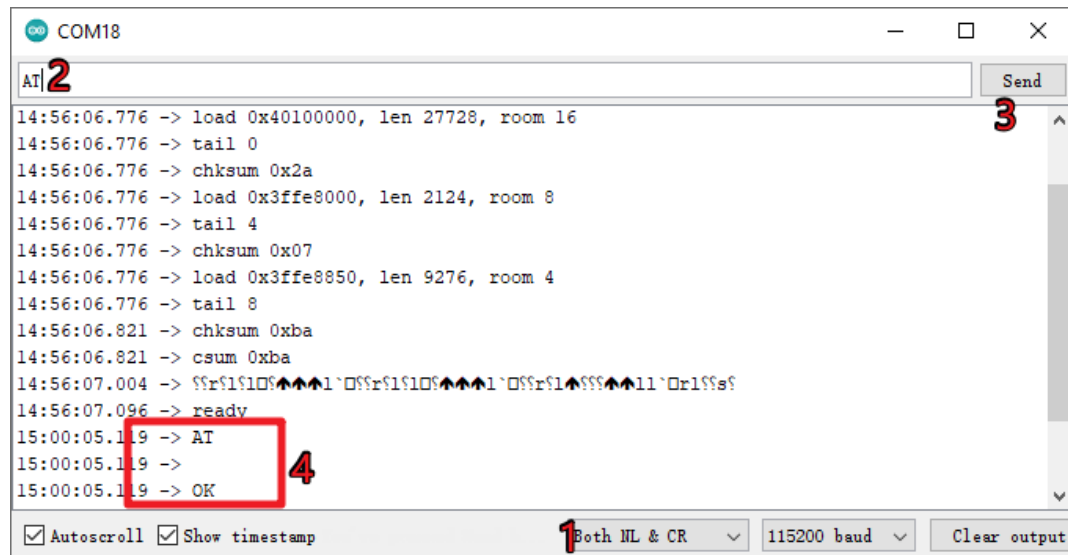
2. You will be able to see information about the ESP8266 module if you click the magnifying glass icon(Serial Monitor) in the upper right corner and set the baud rate to **115200**.



### Note:

- If ready doesn't appear, you can try to reset the ESP8266 module(connect RST to GND) and re-open the Serial Monitor.

3. Click on **NEWLINE DROPDOWN BOX**, select both NL & CR in the drop down option, enter AT, if it returns OK, it means ESP8266 has successfully established connection with R3 board.



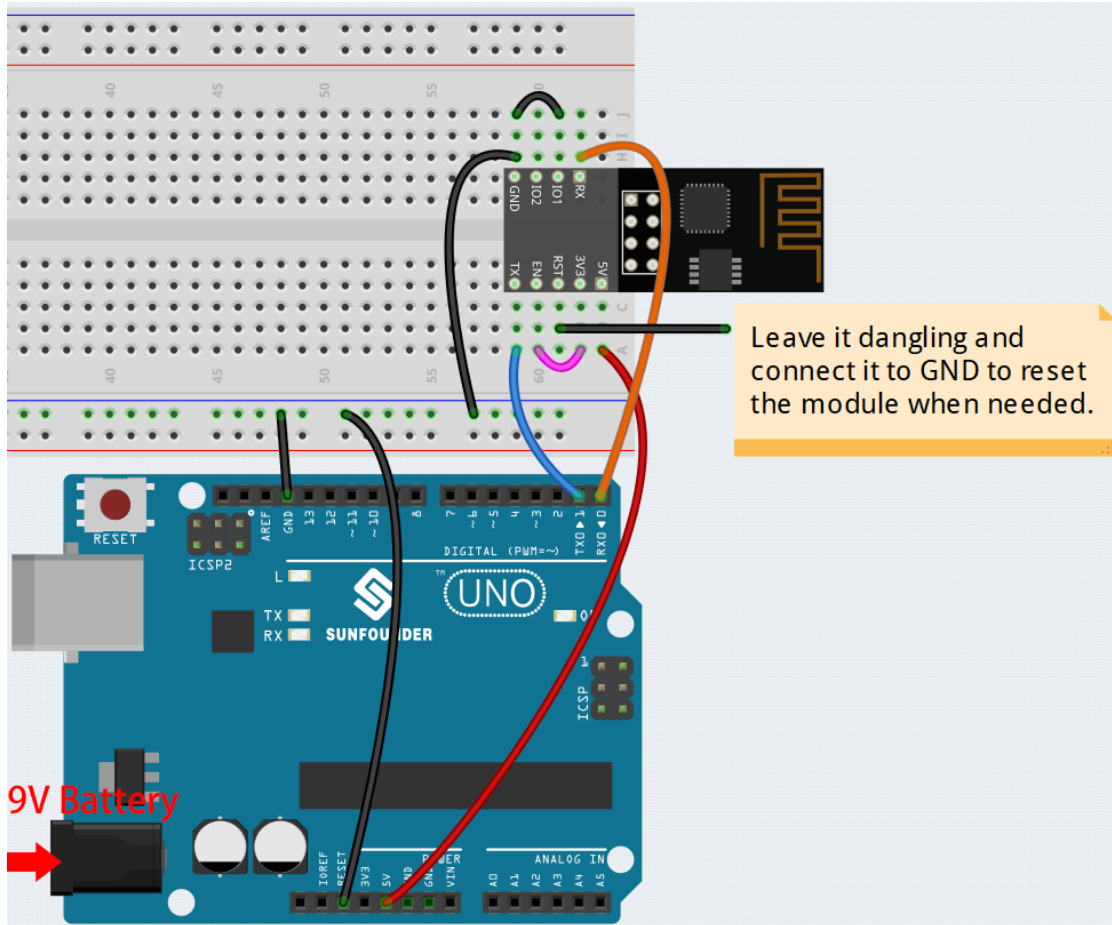
Now you can continue to follow *1.1 Configuring the ESP8266* to set the working mode and baud rate of the ESP8266 module.

## Re-burn the Firmware with R3

### 1. Build the circuit

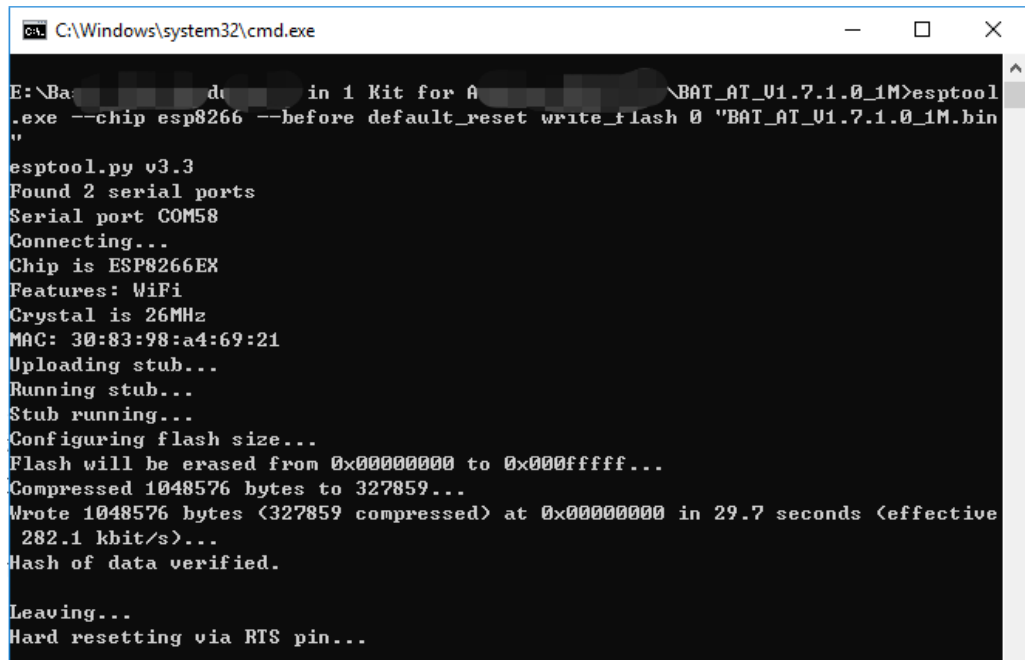
Connect ESP8266 and SunFounder R3 board.





## 2. Burning the firmware

- Follow the steps below to burn the firmware if you are using **Windows**.
  1. Download firmware and burn-in tool.
    - ESP8266 Firmware
  2. After unzipping, you will see 4 files.
    - BAT\_AT\_V1.7.1.0\_1M.bin: The firmware to burn to the ESP8266 module.
    - esptool.exe: This is a command-line utility for Windows.
    - install\_r3.bat: This is the command package for Windows system, double click this file will run all the commands inside the file.
    - install\_r4.bat: Same as install\_r3.bat, but dedicated to UNO R4 board.
  3. Double click install\_r3.bat to start the firmware burning. If you see the following prompt, the firmware has been installed successfully.



```

C:\Windows\system32\cmd.exe

E:\Ba... d... in 1 Kit for A... \BAT_AT_V1.7.1.0_1M>esptool
.exe --chip esp8266 --before default_reset write_flash 0 "BAT_AT_V1.7.1.0_1M.bin
"
esptool.py v3.3
Found 2 serial ports
Serial port COM58
Connecting...
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 30:83:98:a4:69:21
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Flash will be erased from 0x00000000 to 0x000fffff...
Compressed 1048576 bytes to 327859...
Wrote 1048576 bytes (327859 compressed) at 0x00000000 in 29.7 seconds (effective
282.1 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

```

**Note:** If the burn-in fails, please check the following points.

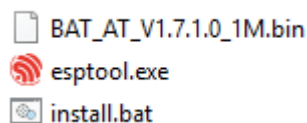
- Reset the ESP8266 module by inserting the RST on the ESP8266 Adapter to GND and then unplugging it.
- Check if the wiring is correct.
- Whether the computer has recognized your board properly, and make sure the port is not occupied.
- Reopen the install.bat file.

- To burn the firmware, follow these steps if you are using a **Mac OS** system.

1. Use the following commands to install Esptool. Esptool is a Python-based, open-source, platform-independent utility to communicate with the ROM bootloader in Espressif chips.

```
python3 -m pip install --upgrade pip
python3 -m pip install esptool
```

2. If esptool is properly installed, it will output a message such as [usage: esptool] if you run `python3 -m esptool`.
3. Download firmware.
  - ESP8266 Firmware
4. After unzipping, you will see 3 files.



BAT\_AT\_V1.7.1.0\_1M.bin  
esptool.exe  
install.bat

- BAT\_AT\_V1.7.1.0\_1M.bin: The firmware to burn to the ESP8266 module.

- esptool.exe: This is a command-line utility for Windows.
  - install\_r3.bat: This is the command package for Windows system.
  - install\_r4.bat: Same as install\_r3.bat, but dedicated to UNO R4 board.
5. Open a terminal and use the cd command to go into the firmware folder you just downloaded, then run the following command to erase the existing firmware and re-burn the new firmware.

```
python3 -m esptool --chip esp8266 --before default_reset erase_flash
python3 -m esptool --chip esp8266 --before default_reset write_flash 0
↪ "BAT_AT_V1.7.1.0_1M.bin"
```

6. If you see the following prompt, the firmware has been installed successfully.

```
MacdeMacBook-Pro:BAT_AT_V1.7.1.0_1M mac$ python3 -m esptool --chip esp8266 --before default_reset w
rite_flash 0 "BAT_AT_V1.7.1.0_1M.bin"
esptool.py v4.3
[Found 3 serial ports
Serial port /dev/cu.usbmodem143401
Connecting...
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 30:83:98:a4:69:21
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Flash will be erased from 0x00000000 to 0x000fffff...
Compressed 1048576 bytes to 327859...
Wrote 1048576 bytes (327859 compressed) at 0x00000000 in 29.6 seconds (effective 283.0 kbit/s)...
Hash of data verified.

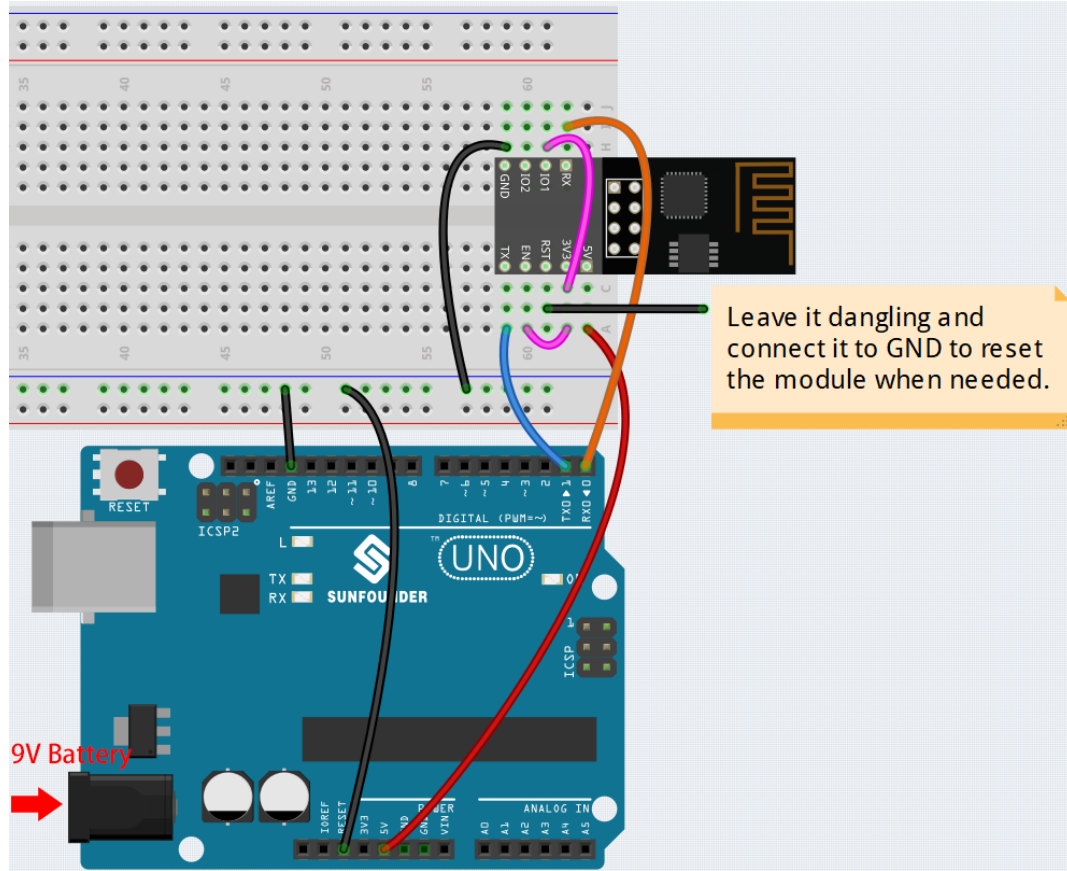
Leaving...
Hard resetting via RTS pin...
MacdeMacBook-Pro:BAT_AT_V1.7.1.0_1M mac$
```

**Note:** If the burn-in fails, please check the following points.

- Reset the ESP8266 module by inserting the RST on the ESP8266 Adapter to GND and then unplugging it.
- Check if the wiring is correct.
- Whether the computer has recognized your board properly, and make sure the port is not occupied.
- Reopen the install.bat file.

### 3. Test

1. On the basis of the original wiring, connect IO1 to 3V3.



2. You will be able to see information about the ESP8266 module if you click the magnifying glass icon(Serial Monitor) in the upper right corner and set the baud rate to **115200**.

```

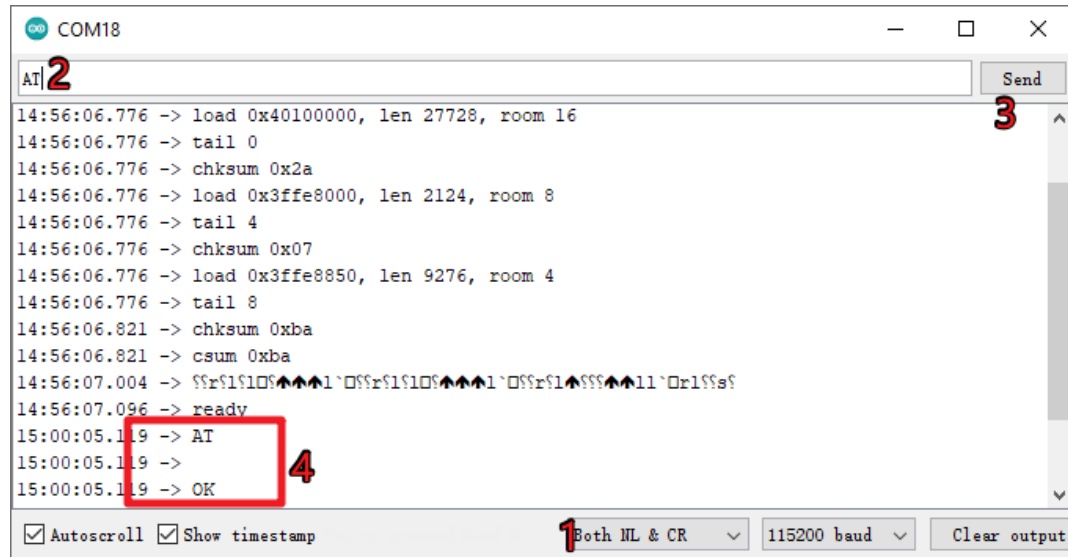
COM18
14:56:06.729 -> ets Jan 8 2013,rst cause:2, boot mode:(3,7)
14:56:06.776 ->
14:56:06.776 -> load 0x40100000, len 27728, room 16
14:56:06.776 -> tail 0
14:56:06.776 -> checksum 0x2a
14:56:06.776 -> load 0x3ffe8000, len 2124, room 8
14:56:06.776 -> tail 4
14:56:06.776 -> checksum 0x07
14:56:06.776 -> load 0x3ffe8850, len 9276, room 4
14:56:06.776 -> tail 8
14:56:06.821 -> checksum 0xba
14:56:06.821 -> csum 0xba
14:56:07.004 -> $$$r$1$10$1111`0$$r$1$10$1111`0$$r$1$10$1111`0r1$$$s$
14:56:07.096 -> ready
  
```

☒ Autoscroll 
 ☒ Show timestamp 
 No line ending 
 115200 baud 
 Clear output

**Note:**

- If ready doesn't appear, you can try to reset the ESP8266 module(connect RST to GND) and re-open the Serial Monitor.

3. Click on **NEWLINE DROPDOWN BOX**, select both NL & CR in the drop down option, enter AT, if it returns OK, it means ESP8266 has successfully established connection with R3 board.



Now you can continue to follow *1.1 Configuring the ESP8266* to set the working mode and baud rate of the ESP8266 module.

## 2.8 Thank You

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

If you have any questions or other interesting ideas, feel free to send an email to [service@sunfounder.com](mailto:service@sunfounder.com).



## **COPYRIGHT NOTICE**

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.