RGBDuino Starter Kit Manual

The RGBDuino Starter Kit is one of the best starter kits for users who are new to Arduino. It does not require particularly difficult soldering operations and complicated circuits. Users can concentrate on learning the use of Arduino. This kit is an integrated PCB board composed of an Arduino compatible motherboard and ten common modules. All modules have been connected to Arduino through PCB embossed holes, so no additional data cables are needed for connection. Of course, you can also take out the module and use the data cable to connect the module. You can use this kit to build any Arduino project you like.

Size: 177*164mm

LED: Simple LED module
Buzzer: Piezo buzzer
OLED Display 0.96": 0.96" 128×64 OLED display module
Button: Button switch
Rotary Potentiometer: adjustable potentiometer
Light: Detect the intensity of ambient light
Sound: Detect the intensity of ambient sound
Temperature & Humidity: Detect the surrounding temperature and humidity value
Air Pressure : Detect the surrounding atmospheric pressure
3-Axis Accelerator: Detect object acceleration
RGBDuino Plus: Arduino compatible board with PH2.0-4P port
Note: By default, all modules are connected to RGBDuino plus through PCB embossed holes. This means that if the connection is not disconnected, there is no need to use an additional PH2.0 data cable to connect. The default pins are as follows:

Module	Port	Pin/Address
LED	digital	D4
Buzzer	digital	D5
OLED Display 0.96”	I2C	I2C, 0x78(default)
Button	digital	D6
Rotary Potentiometer	simulation	A0
Light sensor	simulation	A6
Sound sensor	simulation	A2
Temperature & humidity sensor	digital	D3
Air pressure sensor	I2C	I2C,0x77(default)/0x76(optional)
3-axis digital accelerometer	I2C	I2C, 0x19(default)

Split guide

Note: Be careful not to cut your fingers when using the knife

If you prefer to use these modules in other places, you can split these modules out by following the steps below.

first step

Use a knife or sharp object to cut the embossed hole so that it has obvious cut marks.

Second step

Shake the model part up and down, it will be easily separated from the main body.

Product List

Module	Quantity
Sensor	
Temperature & humidity sensor	x1
3-axis digital accelerometer	x1
Air pressure sensor	x1
Light sensor	x1
Sound sensor	x1
Input module	
Rotary Potentiometer	x1
Button	x1
Output module	
LED	x1
Buzzer	x1
Display module	
OLED display	x1
Grove data line	x6
Micro USB data line	x1

The RGBDuino starter kit is configured with preset firmware when it leaves the factory. You only need to plug in the board and use the control buttons and rotary potentiometer to experience all the sensors at once.

Install Arduino IDE

- Arduino IDE is an integrated development environment for Arduino, used for programming, downloading, and testing of MCU software.
- Download and install the corresponding Arduino IDE according to the operating system: https://www.arduino.cc/en/software

Install the USB driver

- The Arduino is connected to the computer via a USB cable. The USB driver depends on the type of USB chip used on the Arduino.
  Download the CP2102 USB driver:

https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

Note: Please install the driver according to the operating system.

After the driver is installed, use the USB cable to connect the Arduino to the USB port of the computer.

Windows users: You can find the COM serial port information under the path of My Computer -> Properties -> Hardware -> Device Manager.

Mac OS users: You can click the symbol in the upper left corner, and then click About This Mac -> System Report... ->USB.

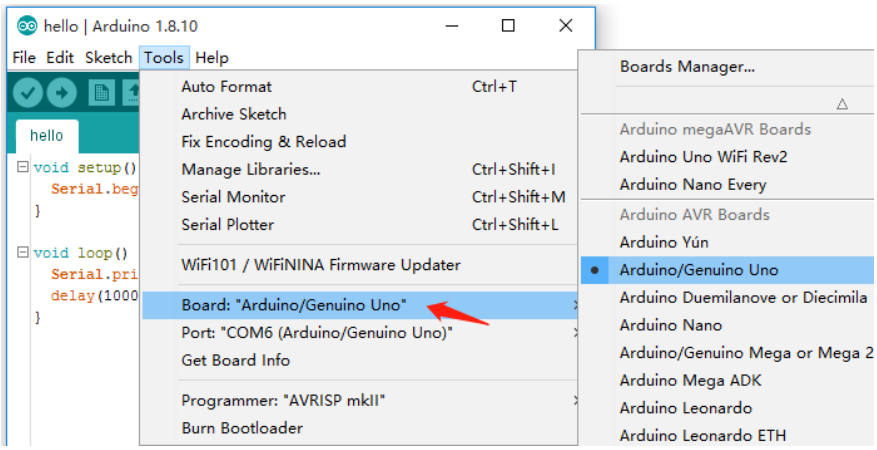
At this time you should be able to find a CP2102 USB driver.

If the driver is not installed, or the driver is not installed correctly (does not match the chip model), it will be displayed as "Unknown Device" in the Device Manager. If this happens, you need to reinstall the driver.

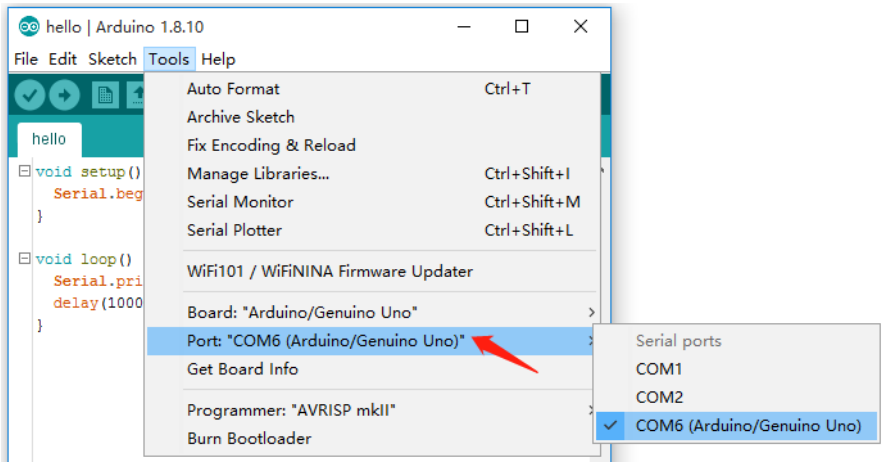
Start the Arduino IDE

1. Open the **Arduino IDE** on the computer.

2. Open the directory menu Tools (Tools) -> Board (development board) and select the corresponding development board, here we should select **Arduino/Genuino Uno**.



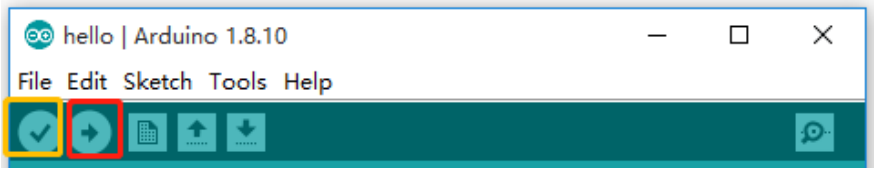
3. Click the Tools->Port menu and select the corresponding port for your development board (this port is the serial port displayed in the device manager in the previous step). In this example we choose COM6. For **Mac OS** users, you should choose /dev/cu.SLAB_USBtoUART.



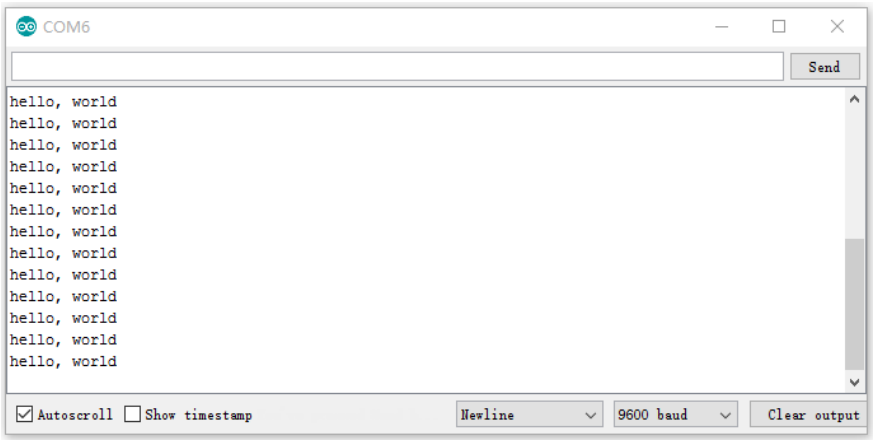
4. Create a new Arduino file and name it Hello.ino, and copy the following code:

```
void setup() {
  Serial.begin(9600); // initializes the serial port with a baud rate of 9600
}
void loop() {
  Serial.println("hello, world"); // prints a string to a serial port
  delay(1000); //delay of 1 second
}
```

5. In the upper left corner of the Arduino IDE, there are two buttons Verify (compile) and Upload (upload). First click the compile button (✓) to compile the program. After successful compilation, click the upload button (→).



6. Navigate to Tools->Serial Monitor or click the **serial monitor** (magnifying glass icon) in the upper right corner, you can see the running result of the program in the serial monitor:



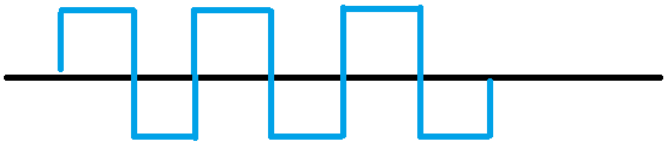
Module example

1. LED

We have learned how to output "Hello world". Now let us learn how to light up the LED module. As we know, the control system consists of three basic parts: input, control and output. Only the output is required to light the LED, not the input. RGBDuino is the control unit, the LED module is the output unit, and the output signal is a digital signal.

background knowledge

Digital signal: Digital signal means that the value of the amplitude is discrete, and the amplitude is limited to a limited number of values. In our controller, the digital signal has two states: **LOW (0V)** is **0**; **HIGH (5V)** is **1**. Therefore, sending a '**HIGH**' signal to the **LED** can make it light up.



- Components involved
- RGBDuino Plus
- LED module (D4)
- PH2.0-4P data cable (only applicable when the module is split)

Hardware connection:

- **Module connection**-The default connection is through the PCB embossed hole.

  -Connect RGBDuino Plus to the computer via a USB cable.

- Program code
- Open the Arduino IDE.
- Copy the following code and click "Compile ()" to check for syntax errors. "Upload (→)" the code after making sure there are no errors:

```
//LED Blink

//The LED will turn on for one second and then turn off for one second

int ledPin = 4;

void setup() {

    pinMode(ledPin, OUTPUT);

}

void loop() {

    digitalWrite(ledPin, HIGH);

    delay(1000);

    digitalWrite(ledPin, LOW);

    delay(1000);

}
```

Code analysis

```
setup()
{
}
```

The setup() function will be called at the beginning of the project to initialize variables, pin patterns and start using the library. The setup() function only runs once every time the **Arduino** development board is powered on or reset.

```
loop()
{
}
```

After creating a setup() function for initializing and setting the initial value, the loop() function will be executed accurately and continuously loop according to the meaning of its name, this allows your program to change and respond. The loop() function is used to actively control the **Arduino** development board.

```
int ledPin = 4;
```

Description:

Convert the value to **int** data type.

Grammar:

int(x) or **(int)x** (C style type conversion)

Parameter:

x: A value. Allowed data types: any type.
Assign an int type **4** to the variable named **ledPin**.

```
pinMode(ledPin, OUTPUT);
```

Description:

Configure the specified pin as input or output. For more information about pin functions, see the "Digital Pins" page.
Starting from **Arduino 1.0.1**, you can use the **INPUT_PULLUP** mode to enable the internal pull-up resistor. In addition, the "**INPUT**" mode explicitly disables the internal pull-up resistor.

Grammar:

pinMode (**pin, mode**)

Parameter:

pin: Set the Arduino board number of the mode.
Mode: INPUT, OUTPUT or INPUT_PULLUP.
Set ledPin to output mode.

```
digitalWrite(ledPin, HIGH);
```

Description:

Write a HIGH or LOW value to the digital pin
If the pin is configured as OUTPUT using **pinMode()**, its voltage should be set to the corresponding value: HIGH is **5V** (or **3.3V**, if it is a **3.3V** board), LOW is **0V**.
If the pin is configured as **INPUT**, the **digitalWrite()** function will enable **(HIGH)** or disable **(LOW)** the internal pull-up on the input pin. It is recommended to set **pinMode()** to INPUT_PULLUP to enable the internal pull-up resistor. For more information, see the "Digital Pins" tutorial.

If you do not set **pinMode()** to **OUTPUT**, but connect an **LED** to a pin, the **LED** may be dimmed when calling **digitalWrite(HIGH)**. If **pinMode()** is not explicitly set, **digitalWrite()** will enable the internal pull-up resistor, which acts as a large current-limiting resistor.

Grammar:

digitalWrite**(pin, value)**

Parameter:

pin: Arduino pin.
value: HIGH or LOW.
When **ledPin** is set to output, **HIGH** means that a high level is sent to this pin, and the **LED** lights up.

```
digitalWrite(ledPin, LOW);
```

When we set the **LED** as an output, **LOW** means sending a low level to the pin and the **LED** is off.

```
delay(1000);
```

Description:

Pause the program for a specified period of time (in milliseconds). (There are 1000 milliseconds per second.)

Grammar:

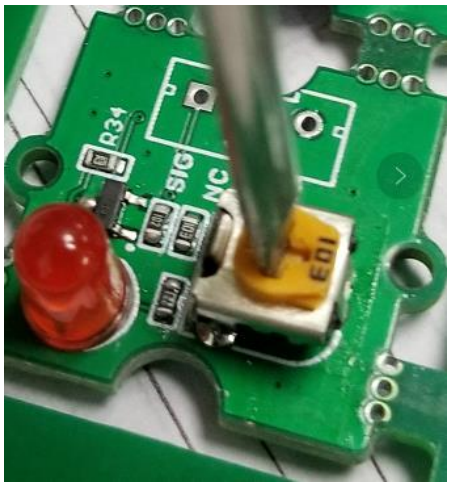
delay**(ms)**

Parameter:

ms: The number of milliseconds to pause. Allowed data types: unsigned long
Delay the program by 1000ms (1s).

Demonstration effects and serial output results:

The LED module will be on for 1 second and off for 1 second.



There is a variable resistor on the LED module, you can use a screwdriver to turn it to make the light brighter.
If the module is separated from the motherboard, you need to use a PH2.0 data cable to connect the **LED module** and the digital interface of RGBDuino Plus **D4**.

2. Button

Regarding the button, we need to understand that the button's input is a digital signal, and it has only two states, 0 or 1, so we can control the output based on these two states.

- **Exercise:** Use buttons to control the on and off of the LED module
- Components involved
  a. RGBDuino Plus
  b. LED module（D4）
  c. Button module (D6)
  d. PH2.0data cable (only applicable when the module is split)
- **Hardware connection:**
- **Module connection-**The default connection is through the PCB embossed hole.
  -Connect RGBDuino Plus to the computer via a USB cable.
- **Hardware analysis:**
- Enter: button
- Control: RGBDuino Plus
- Output: LED module

Both buttons and **LED**s use digital signals, so they should be connected to a digital interface.

- Program code
- Open the **Arduino IDE**.
- Copy the following code and click "Compile (✔)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.

```
//Button to turn ON/OFF LED
//Constants won't change. They're used here to set pin numbers:
const int buttonPin = 6; // the number of the pushbutton pin
const int ledPin = 4; // the number of the LED pin

// variables will change:
int buttonState = 0; // variable for reading the pushbutton status

void setup() {
// initialize the LED pin as an output:
pinMode(ledPin, OUTPUT);
// initialize the pushbutton pin as an input:
pinMode(buttonPin, INPUT);
}
```

```
void loop() {
// read the state of the pushbutton value:
buttonState = digitalRead(buttonPin);

// check if the pushbutton is pressed. If it is, the buttonState is HIGH:
if (buttonState == HIGH) {
// turn LED on:
digitalWrite(ledPin, HIGH);
} else {
// turn LED off:
digitalWrite(ledPin, LOW);
}
}
```

- Code analysis

```
pinMode(ledPin, OUTPUT);
```

Define LED as output unit.

```
pinMode(buttonPin, INPUT);
```

Define the button as the input unit.

```
buttonState = digitalRead(buttonPin);
```

Description:

Read the value from the designated digital pin, HIGH or LOW.

Grammar:

digitalRead**(pin)**

variable:

pin: The pin of the **Arduino** you want to read.

This function is used to read the status of a digital pin, that is, whether it is high (HIGH) or low (LOW). When the button is pressed, the state is high, otherwise it is low.

```
if (buttonState == HIGH) {
digitalWrite(ledPin, HIGH);
} else {
digitalWrite(ledPin, LOW);
}
}
```

description:

Compared with the simplest if statement, the if...else statement allows better control of the code flow by allowing multiple tests to be grouped. If the condition in the if statement is false, the else clause will be executed (if it exists). Another if test can be performed within the else so that multiple mutually exclusive tests can be run at the same time.

Each test will continue to the next test until it encounters a test with a true result. When the test result is true, the associated code block will be run, and then the program will jump to the line after the entire if/else structure. If no test proves correct, the default else block (if it exists) is executed and the default behavior is set.

Note that the else if syntax block can be used with or without the terminating else block, and vice versa. An if...else function allows an unlimited number of such else if branches.

grammar:

```
if (condition 1) {
// do Thing A
}
else if (condition 2) {
// do Thing B
}
else {
// do Thing C
}
```

The usage of this statement is: if the logical expression in the brackets is **true**, execute the statement in the braces after the **if**, otherwise, execute the statement in the braces after the **else**. If the state of the button is high, the **LED** pin outputs High level and turn on the **LED**, otherwise it will turn off the **LED**.

Demonstration effects and serial output results:

Press the button to light up the LED module.

- Split guide

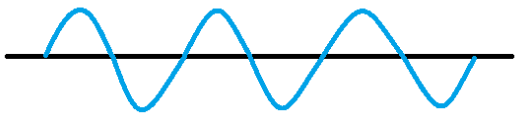
If the module is separated from the main board, you need to use a Grove data cable to connect the Grove LED to the digital interface **D4** of RGBDuino Plus, and connect the Grove button to the digital interface **D6** of RGBDuino Plus.

3. Rotary potentiometer

In the previous section, we learned that the button has only two states, corresponding to the **ON/OFF** states of **0V** and **5V**, but in practice, we often have to meet the needs of many different states, not just **0V** and **5V**. Therefore, you can choose to use analog signals to meet different needs! The rotary potentiometer is a classic example of the use of analog signals.

Background knowledge

Analog signal: The time and value of the signal change continuously, and the amplitude, frequency, or phase of the signal change continuously at any time, such as the current broadcast sound signal or image signal. Common analog signals include sine waves and triangle waves. The analog pin of the microcontroller can map **0V** to **5V** to the range between **0** to **1023**, where **1023** is mapped to **5V**, and **512** is mapped to **2.5v**, etc.



- Components involved
- RGBDuino Plus
- LED module (D4)
- Rotary potentiometer module (A0)
- PH2.0 data cable (applicable only when splitting the module)
- **Hardware connection:**
- **Module connection**-The default connection is through the PCB embossed hole. -Connect RGBDuino Plus to the computer via a USB cable.
- **Hardware analysis:**
- Input: Rotary potentiometer
- Control: RGBDuino Plus
- Output: LED module

The input is an analog signal, so it is connected to the analog signal interface, and the LED module is connected to the digital signal interface.

- Program code
- Open the Arduino IDE.
- Copy the following code and click "Compile (✔)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.

```
//Rotary controls LED
int rotaryPin = A0; // select the input pin for the rotary
int ledPin = 4; // select the pin for the LED
int rotaryValue = 0; // variable to store the value coming from the rotary

void setup() {
// declare the ledPin as an OUTPUT:
pinMode(ledPin, OUTPUT);
pinMode(rotaryPin, INPUT);
}

void loop() {
// read the value from the sensor:
rotaryValue = analogRead(rotaryPin);
// turn the ledPin on
digitalWrite(ledPin, HIGH);
// stop the program for <sensorValue> milliseconds:
delay(rotaryValue);
// turn the ledPin off:
digitalWrite(ledPin, LOW);
// stop the program for for <sensorValue> milliseconds:
delay(rotaryValue);
}
```

- Code analysis

```
int rotaryPin = A0; // select the input pin for the rotary
int ledPin = 4; // select the pin for the LED
```

Description:

You may find that the way we define the rotating pin and the led pin is different, this is because the rotating potentiometer generates an analog signal, and the LED module is controlled by a digital signal

To define the analog interface, use A + digital pin number (here A0).

To define the digital interface, use the digital pin number (here 4).

```
rotaryValue = analogRead(rotaryPin);
```

description:

Read the value from the designated analog pin. The Arduino development board contains a multi-channel 10-bit analog-to-digital converter. This means that it will map the input voltage between 0 and the operating voltage (5V or 3.3V) to an integer value between 0 and 1023. For example, on the Arduino UNO, the resolution between the readings is: 5 volts/1024 units, or 0.0049 volts per unit (4.9 mV).

grammar:

analogRead(**pin**)

parameter:

pin: The name of the analog input pin (A0 to A5 on most boards).

Returns: the analog reading on the pin. Although it is limited by the resolution of the analog-to-digital converter (10-bit is 0-1023 or 12-bit is 0-4095). Data type: int.

This function is used to read the value of the analog pin (rotation sensor position), and the value range is: 0~1023.

```
delay(rotaryValue);
```

Delay function, the number of milliseconds delayed is the value in parentheses. Because this value is the value of the analog signal of the knob pin being read, the delay time is controlled by the knob.

Demonstration effects and serial output results:

Turning the potentiometer will change the frequency of the LED blinking.

- **Split guide**

If the module is separated from the motherboard, you need to use a PH2.0 data cable to connect the LED module and the digital interface D4 of RGBDuino Plus, and connect the rotary potentiometer to the analog interface A0 of RGBDuino Plus.

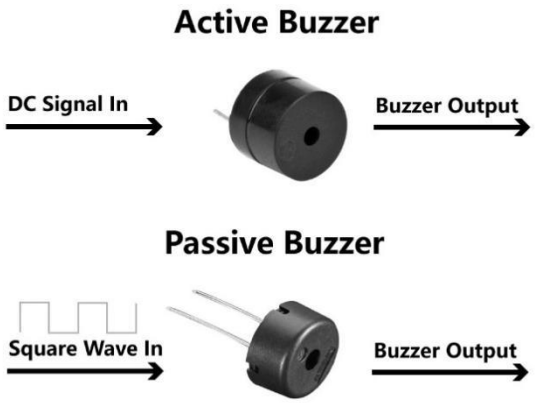
4. Buzzer

background knowledge

There are two types of buzzers, one is active and the other is passive. Both active buzzer and passive buzzer are used to sound electronic products.

The active buzzer has an internal oscillation source, as long as the power is turned on, the buzzer will emit a sound. Active buzzers are widely used in sound devices of electronic products such as computers, printers, copiers, alarms, electronic toys, automotive electronics, telephones, timers, etc.

The passive buzzer has no internal oscillation source and needs to be driven by a square wave and different frequencies. It acts like an electromagnetic speaker, changing the input signal will automatically produce sound, not tone.



The buzzer module used in this kit is a **passive buzzer**, so an alternating current signal (AC) is needed to control it. This leads to the next knowledge point, how to generate square wave (AC signal) with Arduino! A simple method is to use PWM.

There are six digital pins on RGBDuino Plus, which are marked with a "~" symbol, indicating that they can send PWM signals: 3, 5, 6, 9, 10, 11. They are called PWM pins.

- What is **PWM**?

Pulse Width Modulation (PWM) is a technology that obtains analog results digitally. Digital control is used to create a square wave, a signal that switches between on (ON) and off (OFF). By changing the ratio of signal "ON" and signal "OFF" time length, this switch mode can simulate the voltage between full open (5 volts) and off (0 volts). The duration of the "on time" is called the pulse width. To obtain a varying analog value, you can change or modulate the pulse width. For example, if you repeat this switching pattern fast enough with an LED, the result is as if the signal is a stable voltage between 0 and 5v that controls the brightness of the LED.

To generate a PWM signal in Arduino, you can use `analogWrite()`, which is different from using `digitalWrite()` to generate a DC signal.

There are six digital pins on RGBDuino Plus, which are marked with a "~" symbol, indicating that they can send **PWM** signals: 3, 5, 6, 9, 10, 11. They are called **PWM** pins.

 Components involved

RGBDuino Plus

b. Buzzer module (D5)

C. PH2.0 data cable (applicable only when splitting the module)

- Hardware connection:
- Module connection-the default connection is through the PCB embossed holes. -Connect RGBDuino Plus to the computer via a USB cable.
- Program code
- Open the Arduino IDE.

- Copy the following code and click "Compile (✓)" to check for syntax errors. If there are no errors in the compilation, you can "upload (→)" the code.

```
int BuzzerPin = 5;

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

void loop() {
analogWrite(BuzzerPin, 128);
delay(1000);
analogWrite(BuzzerPin, 0);
delay(0);
}
```

- Code analysis

```
analogWrite(BuzzerPin, Value);
```

Description:

Write the analog value (PWM wave) to the pin. It can be used to light up LEDs with different brightness or drive motors at various speeds. After calling the analogWrite() function, this pin will generate a stable rectangular wave with a specified duty cycle until the next call to analogWrite() (or digitalRead() or digitalWrite()) on the same pin.

Grammar:

analogWrite(**pin, value**)

Parameter:

pin: the Arduino pin to be written. Allowed data types: int.

Value: Duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int.

Write the analog value (PWM wave) to the buzzer.

Demonstration effects and serial output results:

The buzzer will sound.

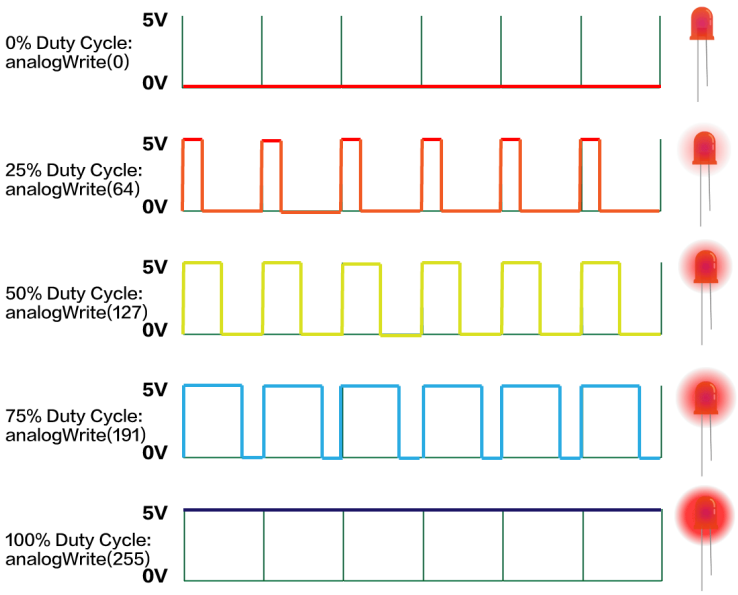
- Split guide

If the module is separated from the main board, you need to use the PH2.0 data cable to connect the buzzer and the digital interface **D5** of RGBDuino Plus.

- PWM usage

Now we have learned how to use PWM. In addition to using PWM to control passive buzzers, we can also use PWM to control the speed of the motor and the brightness of LED lights.

As shown in the figure below, using analogWrite() to generate PWM wave, the higher the duty cycle, the brighter the LED light.



However, the LED lights in this kit cannot be directly controlled by PWM, because the LED is connected to D4, and as mentioned above, the PWM pins are 3,5,6,9,10,11, and pin 4 is not a PWM pin. If you want to use PWM to control the LED, you need to break it off and use the PH2.0 line to connect to the RGBDuino Plus pin with PWM function, such as D3.

```
int LED = 3; // Cable connection from LED to D3

int Potentiometer = A0;


void setup() {

   pinMode(LED, OUTPUT);

   pinMode(Potentiometer, INPUT);

}


void loop() {

   int potentioValue, Value;

   potentioValue = analogRead(Potentiometer);
```

```
  Value = map(potentioValue, 0, 1023, 0, 255); //Mapping potentiometer value to PWM signal value

  analogWrite(LED, Value);

}
```

Compile and upload the code, you should be able to use the **PWM** signal to reverse and adjust the brightness of the LED!

Code analysis:

```
outputValue = map(sensorValue, 0, 1023, 0, 255);
```

Description:

Remap numbers from one range to another. That is to say, the value of fromLow will be mapped to toLow, the value of fromHigh will be mapped to toHigh, the middle value will be mapped to the middle value, etc.

Do not limit the value to this range, because sometimes values outside the range are intentionally used. If you need to limit the range, you can use the "constrain()" function before or after the function.

Please note that the "lower limit" of any range may be greater or less than the "upper limit", so you can use the map() function to reverse the number range, for example:

```
y = map(x, 1, 50, 50, 1);
```

This function can also handle negative numbers very well, so it is also effective for the following examples and works well.

```
y = map(x, 1, 50, 50, -100);
```

The map() function uses integer calculations. Even if the result of the mathematical operation is a score, it will not generate a score. The part after the decimal point will be truncated, and will not be rounded or averaged.

Grammar:

```
map(value, fromLow, fromHigh, toLow, toHigh)
```

Parameter:

value: The value to be mapped.

fromLow: The lower limit of the current range of the value.

fromHigh: The upper limit of the current range of the value.

toLow: The lower limit of the target range of this value.

toHigh: The upper limit of the target range of the value.

Map the light sensor analog signal (0 to 1023) to the brightness value of the LED (0 to 255).

After mapping the time value, keep the potentiometers equal. Map has five parameters, in order: the original value of the mapping, the minimum value of the original value, the maximum value of the original value, the minimum value after the mapping, and the maximum value of the mapping. In this way, the data returned by the sensor can be mapped from its original value 0-1023 to 0-255.

Demonstration effects and serial output results:

Adjust the potentiometer to adjust the brightness of the LED.

All in all, when you want to use the PWM function, you need to select those leads with a "~" symbol in front of their names.

5. Light sensor

The light sensor contains a photosensitive resistor for measuring light intensity. The resistance value of this photosensitive resistor translates to an increase in light intensity and vice versa. The output signal is an analog value, the brighter the light source, the analog value replaces. Based on this property, you can use it to control the light switch.

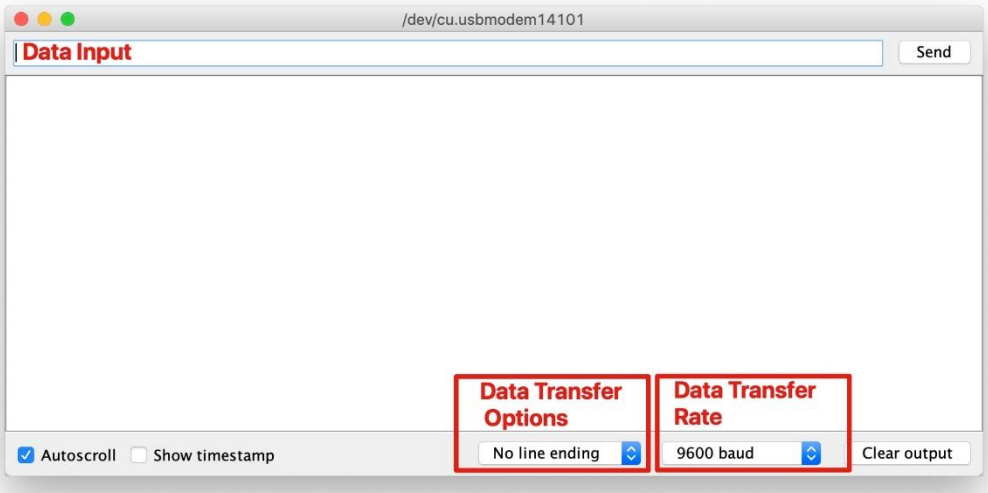
In the following sections, we will use the serial monitor to observe the results of the sensor, so a brief introduction will be given in this section!

background knowledge:

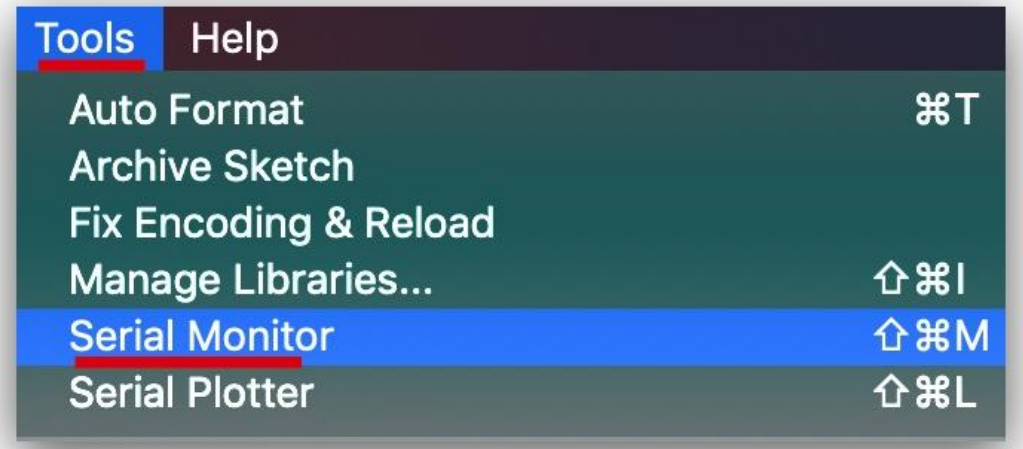
What is a serial monitor (Serial Monitor)?

The serial monitor is a useful tool for observing the results on the Arduino. It is often used to print or debug the results from the sensor. You can also send data back to the controller through

the serial monitor to perform certain tasks! Note: Please ensure that the serial port data transmission matches the code.



You can open the serial monitor by clicking: **Tools** -> **Serial Monitor**.



- Components involved
- RGBDuino Plus
- LED module (D4)
- Light sensor module (A6)
- PH2.0 data cable (only applicable when the module is split)
- **Hardware connection:**
- **Module connection**-The default connection is through the PCB embossed hole. -Connect RGBDuino Plus to the computer via a USB cable.

- **Hardware analysis:**
- Input: light sensor
- Control: RGBDuino Plus
- Output: LED module
- Program code
- Open the Arduino IDE.
- Copy the following code and click "Compile (✔)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.

```
// Light Switch
int sensorpin = A6; // Analog input pin that the sensor is attached to
int ledPin = 4; // LED port
int sensorValue = 0; // value read from the port
int outputValue = 0; // value output to the PWM (analog out)

void setup() {
pinMode(ledPin,OUTPUT);
pinMode(sensorpin, INPUT);
Serial.begin(9600);
}

void loop() {
// read the analog in value:
sensorValue = analogRead(sensorpin);

Serial.println(sensorValue);

if (sensorValue < 200) {
digitalWrite(ledPin, HIGH);
}
else {
digitalWrite(ledPin, LOW);
}

delay(200);
}
```

You can also view the light intensity reading from the **serial monitor**, navigate to **Tools** -> **Serial Monitor.**

- Code analysis

```
Serial.begin(9600);
```

Description:

Set the serial data transmission rate in bits per second (baud). To communicate with the serial monitor, make sure to use one of the baud rates listed in the menu at the bottom right corner of its screen. However, you can specify other rates. For example, communicate with components that require a specific baud rate through pins 0 and 1.

The optional second parameter is used for configuration data, parity and stop bits. The default value is 8 data bits, no parity, and one stop bit.

The software running on the computer communicates with the development board at a baud rate of 9600.

Grammar:

Serial.begin(**speed**)

Parameter:

speed: serial communication speed, for example: 9600, 115200, etc.

Set the serial baud rate to 9600.

```
Serial.println(sensorValue);
```

Description:

Print the data to the serial port as human-readable ASCII text, followed by a carriage return (ASCII 13, or'\r') and line feed (ASCII 10, or'\n'). The format of this command is the same as Serial.print().

Grammar:

Serial.println(**val**) or Serial.println(**val**, **format**)

Parameter:

val: The value to be printed. Allowed data types: any data type.

format: Specify the base (for integer data types) or the number of decimal places (for floating point types).

Print the value of the light sensor through the serial port, and open the **serial monitor** in the IDE interface, and you can see the value of the output sensor.

Demonstration effects and serial output results:

If the light detection environment is dark, the LED module will light up.

- Split guide

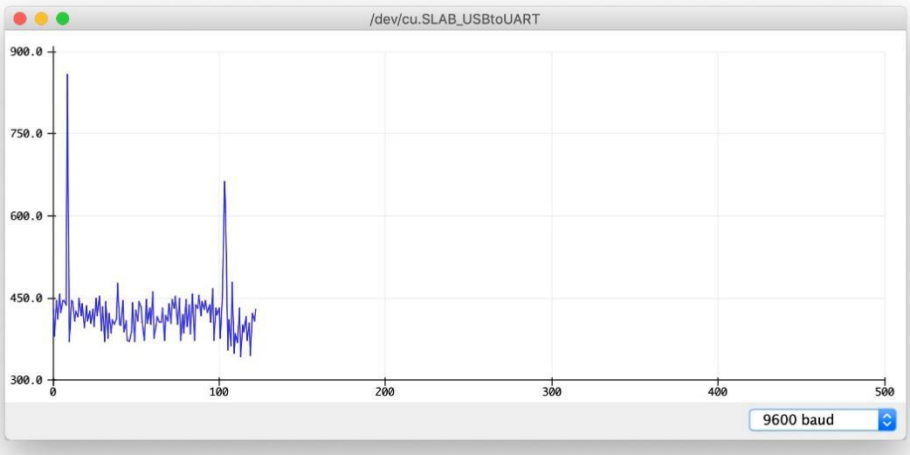
Use the PH2.0 data cable to connect the LED module to the digital signal interface **D4** of RGBDuino Plus, and connect the light sensor module to the analog signal interface **A6** of RGBDuino Plus.

6. Sound sensor

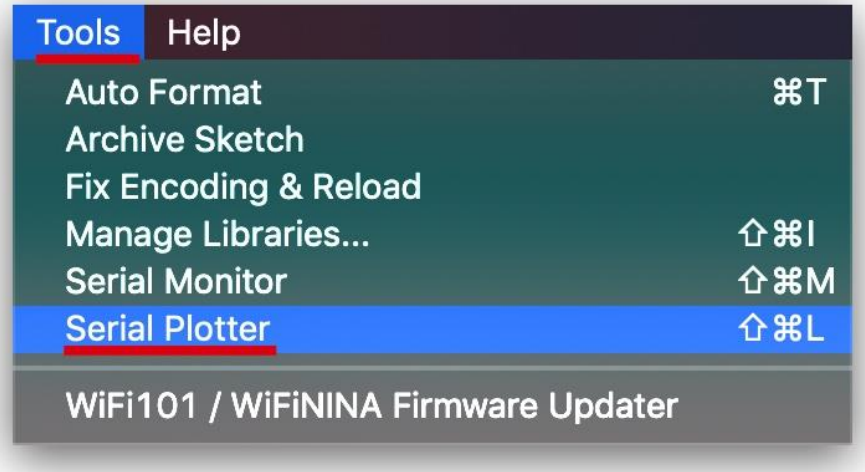
The sound sensor can detect the sound intensity of the environment and simulate its output signal. I am sure that all of you have already been exposed to voice-activated lights. After studying the first few chapters, we already have the basic knowledge. Now we can use the knowledge we have learned to make our own sound-activated lights. In this experiment we will use a serial plotter to visualize the results.

Background information

Similar to the serial monitor, the serial plotter allows you to transfer the Arduino real-time data to the computer and display it in graphical form. When you need to visualize data, a serial plotter is very useful.



We can open the serial plotter by **clicking Tools->Serial Plotter**.



- **Practice**: When a sound is made, the LED light is on. When there is no sound and it is very quiet, the LED light will go out.
- Components involved
- RGBDuino Plus
- LED Module (D4)
- Sound sensor module (A2)
- PH2.0 data cable (only applicable when the module is split)
- **Hardware connection:**
- **Module connection**-The default connection is through the PCB embossed hole.
  -Connect RGBDuino Plus to the computer via a USB cable.
- Program code
- Open the Arduino IDE.
- Copy the following code and click "Compile (✔)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.

```
//Sound Control Light
int soundPin = A2; // Analog sound sensor is to be attached to analog
int ledPin = 4; // Digital LED is to be attached to digital
void setup() {
pinMode(ledPin, OUTPUT);
pinMode(soundPin, INPUT);
Serial.begin(9600);
}
void loop(){
int soundState = analogRead(soundPin); // Read sound sensor's value
Serial.println(soundState);
// if the sound sensor's value is greater than 200, the light will be on for 5 seconds.
//Otherwise, the light will be turned off
```

```
if (soundState > 400) {
digitalWrite(ledPin, HIGH);
delay(100);
}else{
digitalWrite(ledPin, LOW);
}
}
```

You can also see the light intensity reading from the **serial plotter** and navigate to **Tools->Serial Plotter.**

Note: You can also adjust the value according to the surrounding light intensity.

- Code analysis

```
Serial.begin(9600);
```

The software running on the computer communicates with the development board at a baud rate of 9600.

```
Serial.print(" ");
```

This function is used to output data from the serial port, and the output is the content enclosed in double quotes.

```
Serial.println( );
```

This statement is similar to the above statement, except that **serial.println** has a newline returned.

```
Serial.println(soundState);
```

The serial port prints the value of the sound sensor. Open the **serial monitor** in the IDE interface, and you will see the value of the output sensor.

Demonstration effects and serial output results:

If the surrounding sound is loud enough, the LED module will light up.

- Split guide

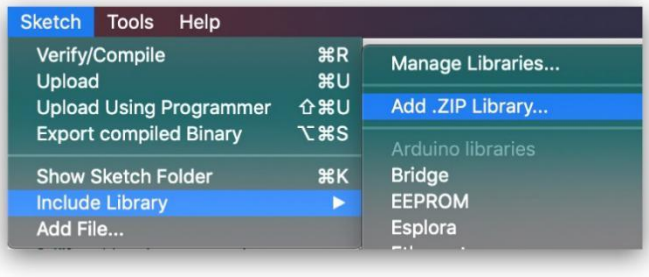
Use the PH2.0 data cable to connect the LED module to the digital signal interface **D4** of RGBDuino Plus, and connect the sound sensor module to the analog signal interface **A2** of RGBDuino Plus.

7. OLED display

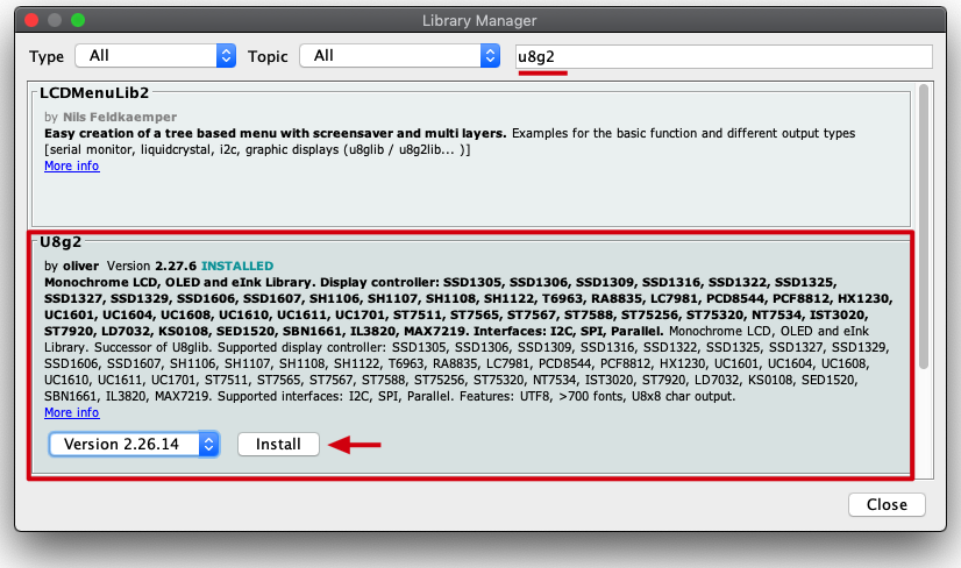
The OLED display can be used in many situations, and you can use it to visualize sensor readings!

Background information

Just like most other programming platforms, the Arduino environment can be extended by using libraries. The library provides additional functions for the project, namely using specific hardware or processing data. To use the library in the project, please **select Sketch->Include Library**.



- Components involved
- RGBDuino Plus
- OLED module
- PH2.0 data cable (only applicable when the module is split)
- **Hardware connection:**
- **Module connection**-The default connection is through the PCB embossed hole. -Connect RGBDuino Plus to the computer via a USB cable.
- Program code
- Open the Arduino IDE.
- Install **U8g2 library**: Navigate to **Sketch->Include Library->Manage Libraries**... Search for the keyword **"U8g2"** in **Library Manager**, confirm it is **U8g**2 by oliver, and install it.



- Copy the following code and click "Compile (✔)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.

```
#include <U8x8lib.h>

U8X8_SSD1306_128X64_ALT0_HW_I2C u8x8(/* reset=*/ U8X8_PIN_NONE);

void setup(void) {
u8x8.begin();
u8x8.setFlipMode(1);
}

void loop(void) {
u8x8.setFont(u8x8_font_chroma48medium8_r);
u8x8.setCursor(0, 0);
u8x8.print("Hello World!");
}
```

- Code analysis

```
#include <>
```

Description:

```
#include <U8x8lib.h>
```

#include Used to include external libraries in the project. This allows programmers to access a large number of standard C language libraries (a set of pre-made functions), as well as libraries specifically written for Arduino.

Note that #include and #define are similar, there is no semicolon terminator. If you add a semicolon, the compiler will generate an error message.

```
U8X8_SSD1306_128X64_ALT0_HW_I2C u8x8(/* reset=*/ U8X8_PIN_NONE);
```

After the object is declared, the functions in the library can be used.

```
u8x8.begin();
```

Description:

Simplified setup process of display in Arduino environment. For information on choosing the appropriate U8g2 constructor, see the installation guide.

Grammar:

u8x8.begin()

Initialize the u8g2 library.

```
u8x8.setFlipMode(1);
```

Description:

Some monitors support 180 degree rotation of the internal frame buffer. You can use this process to control this hardware function. Important: After changing the flip mode, please redraw the complete display. It is best to clear the display first, then change the flip mode, and finally redraw the content. The result of any existing content on the screen will be uncertain.

Grammar:

u8x8.setFlipMode(**mode**)

Parameter:

mode: 0 or 1

Flip the display 180 degrees.

```
u8x8.setCursor();
```

Description:

Define the cursor of the print function. Any output from the print function will start from this position.

Grammar:

u8x8.setCursor(**x, y**)

Parameter:

x, y: Print the column/row position of the function cursor.

Set the drawing cursor position.

```
u8x8.setFont()
```

Description:

Define u8x8 fonts for glyph and string drawing functions.

Grammar:

u8x8.setFont(**font_8x8**)

Set the display font.

```
u8x8.print();
```

Draw content on the OLED.

Demonstration effects and serial output results:

Print Hello World to the OLED display.

U8g2 library usage reference

If you want to find more information about U8g2 library, please refer to here.

- Split guide

Use the Grove data cable to connect the OLED to the I2C interface of RGBDuino Plus (Note: The default I2C address is 0x78).

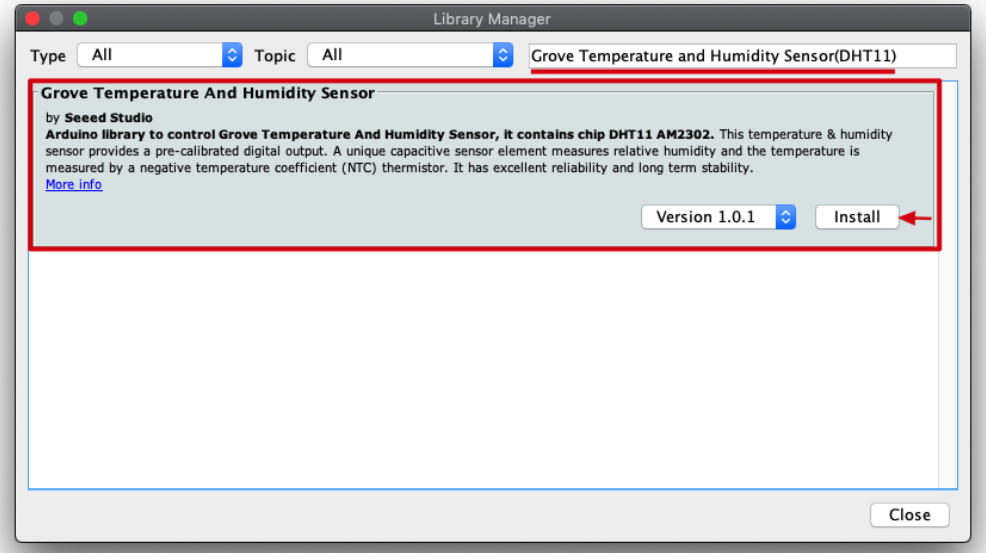
7. Temperature and humidity sensor

Background information

Protocol signal: The protocol signal we use is I2C, here is a brief introduction of I2C. The I2C bus only needs two wires to transfer information between devices: SDA (serial data line) and SCL (serial clock line). These two lines are bidirectional I/O lines, which are mainly used to start the bus to transmit data and generate a clock to turn on the transmission device. At this time, any device being addressed will be considered from the device. The relationship between the master-slave, sender and receiver on the bus is not constant, but depends on the direction of data transmission. If the host wants to send data to the slave device, the host first addresses the slave device, then actively sends data to the slave device, and finally the host terminates the data transmission. If the host wants to receive data from the slave device, the slave device is first addressed by the host. Then the host receives the data from the slave, and the host terminates the receiving process. under these circumstances. The host is responsible for generating the timing clock and terminating data transmission.

- **Practice:** Let your OLED display display the current ambient temperature and humidity.
- Components involved
- RGBDuino Plus
- OLED module
- Temperature and Humidity Sensor Module (D3)
- PH2.0 data cable (only applicable when the module is split)
- **Hardware connection:**
- **Module connection**-The default connection is through the PCB embossed hole. -Connect RGBDuino Plus to the computer via a USB cable.
- Program code
- Open the Arduino IDE.

- Install the **temperature and humidity sensor (DHT11) library file**: Navigate to **Sketch->Include Library->Manage Libraries**... Search for the keyword "Grove Temperature and Humidity Sensor(DHT11)" in **Library Manager**, and then install it.



- Copy the following code and click "Compile (✓)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.

```
//Temperature and Humidity Sensor
#include "DHT.h"
#include <U8x8lib.h>

#define DHTPIN 3 // what pin we're connected to
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);

U8X8_SSD1306_128X64_ALT0_HW_I2C u8x8(/* reset=*/ U8X8_PIN_NONE);

void setup(void) {
Serial.begin(9600);
Serial.println("DHTxx test!");
dht.begin();
u8x8.begin();
u8x8.setPowerSave(0);
u8x8.setFlipMode(1);
}

void loop(void) {
```

```
float temp, humi;
temp = dht.readTemperature();
humi = dht.readHumidity();

u8x8.setFont(u8x8_font_chroma48medium8_r);
u8x8.setCursor(0, 33);
u8x8.print("Temp:");
u8x8.print(temp);
u8x8.print("C");
u8x8.setCursor(0,50);
u8x8.print("Humidity:");
u8x8.print(humi);
u8x8.print("%");
u8x8.refreshDisplay();
delay(200);
}
```

- Code analysis

float temp, humi;

Define variables to store readings.

```
temp = dht.readTemperature();
humi = dht.readHumidity();
```

Description:

A function used to read temperature and humidity from the sensor.

Grammar:

dht.readTemperature() and **dht.readHumidity()**. Return type: **float.**

Call these functions to read the temperature and humidity and store them in the defined variables.

Demonstration effects and serial output results:

The temperature and humidity of the surrounding environment appear on the OLED screen.

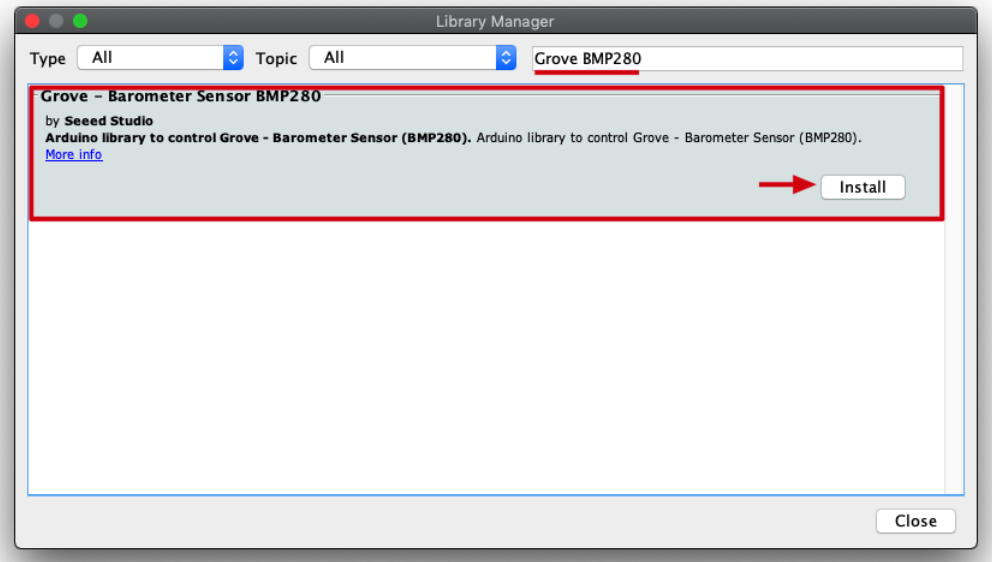
- Split guide

Use the PH2.0 data cable to connect the OLED to the **I2C** interface of RGBDuino Plus (Note: The default address of I2C is 0x78). Connect the temperature and humidity sensor module to the digital signal interface **D3** of RGBDuino Plus.

9. Air pressure sensor

The air pressure sensor module (BMP280) is a breakout board for Bosch BMP280's high-precision low-power digital barometer. This module can be used to accurately measure temperature and atmospheric pressure. When atmospheric pressure changes with altitude, it can also measure the approximate altitude of a place.

- Components involved
- RGBDuino Plu
- Air pressure sensor module
- PH2.0 data cable (only applicable when the module is split)
- **Hardware connection:**
- **Module connection**-The default connection is through the PCB embossed hole. -Connect RGBDuino Plus to the computer via a USB cable.
- Program code
- Open the Arduino IDE.
- Install **Grove barometer sensor library file**: Navigate to **Sketch->Include Library->Manage Libraries**... Search for the keyword "**Grove BMP280**" in **Library Manager**, and then install it.
- Copy the following code and click "Compile (✓)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.



- In this program, the acceleration information is sent to **RGBDuino Plus** via I2C bus, and then **RGBDuino Plus** prints them to the serial monitor. Open the **serial monitor** to view the results.

```
//Air pressure detection
#include "Seeed_BMP280.h"
```

```
#include <Wire.h>

BMP280 bmp280;

void setup() {
Serial.begin(9600);
if (!bmp280.init()) {
Serial.println("Device not connected or broken!");
}
}

void loop() {

float pressure;

//get and print temperatures
Serial.print("Temp: ");
Serial.print(bmp280.getTemperature());
Serial.println("C"); // The unit for Celsius because original arduino don't support speical symbols

//get and print atmospheric pressure data
Serial.print("Pressure: ");
Serial.print(pressure = bmp280.getPressure());
Serial.println("Pa");

//get and print altitude data
Serial.print("Altitude: ");
Serial.print(bmp280.calcAltitude(pressure));
Serial.println("m");

Serial.println("\n");//add a line between output of different times.

delay(1000);
}
```

- Code analysis

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

#include is an instruction to include header files. Here we use the library, which is included in the **Arduino IDE**.

```
#include "Seeed_BMP280.h"
```

Indicates that the **Seeed_BMP280.h** header file of the current path is imported.

```
if (!bmp280.init()) {
Serial.println("Device not connected or broken!");
```

```
}
```

Description:

Initialize the air pressure sensor. If it cannot be initialized, an error will be printed.

Grammar:

bmp280.init()

If the air pressure sensor cannot start normally, an error is output to the serial monitor.

```
Serial.print(bmp280.getTemperature());
```

Description:

The function used to read pressure readings from the sensor.

Grammar:

bmp280.getTemperature(), return type: float

Print the temperature data to the serial monitor.

```
Serial.print(pressure = bmp280.getPressure());
```

Description:

The function used to read the barometric pressure value from the sensor.

Grammar:

bmp280.getPressure(), return type: float

Print the current pressure value.

```
Serial.print(bmp280.calcAltitude(pressure));
```

Description:

Take the pressure value and convert it to height.

Grammar:

bmp280.calcAltitude(float), return type: float

Parameter:

float: The pressure value.

Print amplitude.

Demonstration effects and serial output results:

The barometric pressure reading is displayed on the serial monitor.

- Split guide

Use the PH2.0 data cable to connect the air pressure sensor to the **I2C** interface of RGBDuino Plus (note: the default address of I2C is 0x77 or 0x76).

10. 3-axis accelerometer

This is the last sensor, the three-axis accelerometer. Using this module, you can easily add motion monitoring functions to your design. Therefore, we can carry out many interesting little experiments on the basis of sports.

- **Practice**: When motion is detected, the buzzer will sound an alarm to indicate that the object is moving.
- Components involved
- RGBDuino Plus
- 3-axis accelerometer
- PH2.0 data cable (only applicable when the module is split)
- **Hardware connection:**
- **Module connection**-The default connection is through the PCB embossed hole. -Connect RGBDuino Plus to the computer via a USB cable.
- Program code
- Open the Arduino IDE.
- Download the 3-axis digital accelerometer (±2g to 16g) from Github. Click **Sketch> Include library> Add .ZIP library** to import the library into the IDE.
- Copy the following code and click "Compile (✓)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.
- In this program, acceleration information is sent from the sensor to RGBDuino Plus via I2C bus, and then RGBDuino Plus prints them to the serial monitor. Open the serial monitor to check the result.

```
//Gravity Acceleration
#include "LIS3DHTR.h"
#ifdef SOFTWAREWIRE
#include <SoftwareWire.h>
SoftwareWire myWire(3, 2);
LIS3DHTR<SoftwareWire> LIS; //Software I2C
```

```
#define WIRE myWire
#else
#include <Wire.h>
LIS3DHTR<TwoWire> LIS; //Hardware I2C
#define WIRE Wire
#endif

void setup() {
Serial.begin(9600);
while (!Serial) {};
LIS.begin(WIRE, 0x19); //IIC init
delay(100);
LIS.setOutputDataRate(LIS3DHTR_DATARATE_50HZ);
}
void loop() {
if (!LIS) {
Serial.println("LIS3DHTR didn't connect.");
while (1);
return;
}
//3 axis
Serial.print("x:"); Serial.print(LIS.getAccelerationX()); Serial.print(" ");
Serial.print("y:"); Serial.print(LIS.getAccelerationY()); Serial.print(" ");
Serial.print("z:"); Serial.println(LIS.getAccelerationZ());

delay(500);
}
```

- Code analysis

```
#include "LIS3DHTR.h"
#ifdef SOFTWAREWIRE
#include <SoftwareWire.h>
SoftwareWire myWire(3, 2);
LIS3DHTR<SoftwareWire> LIS; //Software I2C
#define WIRE myWire
#else
#include <Wire.h>
LIS3DHTR<TwoWire> LIS; //Hardware I2C
#define WIRE Wire
#endif
```

Use software I2C or hardware I2C to initialize the module.

```
while (!Serial) {};
```

If you don't open the serial monitor, the code will stop here, so open the serial monitor.

```
LIS.begin(WIRE, 0x19);
LIS.setOutputDataRate(LIS3DHTR_DATARATE_50HZ);
```

Description: Initialize the accelerometer.

Syntax: `LIS.begin(Wire, address)`.

Description: Set the frequency of accelerometer output data.

Syntax: `LIS.setOutputDataRate(odr_type_t odr)`.

Initialize the accelerometer and set the output frequency to 50Hz.

```
Serial.print("x:"); Serial.print(LIS.getAccelerationX()); Serial.print(" ");
Serial.print("y:"); Serial.print(LIS.getAccelerationY()); Serial.print(" ");
Serial.print("z:"); Serial.println(LIS.getAccelerationZ());
```

Description:

The function used to read x-axis data from the sensor.

Grammar:

LIS.getAccelerationX(). Return type: float.

Description:

The function used to read the y-axis data from the sensor.

Grammar:

LIS.getAccelerationY(). Return type: float.

Description:

The function used to read z-axis data from the sensor.

Grammar:

LIS.getAccelerationZ(). Return type: float.

Print the 3-axis data to the serial monitor.

- **Demonstration effects and serial output results:**

The 3-axis accelerator reading is displayed on the serial monitor.

- Split guide

Use the PH2.0 data cable to connect the 3-axis accelerometer module to the I2C interface of RGBDuino Plus (Note: The default I2C address is 0x19).

Project

1. Music dynamic rhythm light

- **Project introduction:** In this experiment, we will make the buzzer play sweet music, and the led lights will flash according to the frequency and beat of the music.
- Components involved
- RGBDuino Plus
- LED module (D4)
- Buzzer module (D5)
- PH2.0 data cable (only applicable when the module is divided)
- **Hardware connection:**
- **Module connection**-the default connection is through the PCB embossed hole. -Connect the RGBDuino Plus to the computer via the USB cable.
- Program code
- Open the Arduino IDE.
- Copy the following code and initialize "Compile (✓)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.

```
//Music Dynamic Rhythm Lamp
#define NTD0 -1
#define NTD1 294
#define NTD2 330
#define NTD3 350
#define NTD4 393
#define NTD5 441
#define NTD6 495
#define NTD7 556

#define NTDL1 147
#define NTDL2 165
#define NTDL3 175
#define NTDL4 196
#define NTDL5 221
#define NTDL6 248
#define NTDL7 278

#define NTDH1 589
```

```
#define NTDH2 661
#define NTDH3 700
#define NTDH4 786
#define NTDH5 882
#define NTDH6 990
#define NTDH7 112

#define WHOLE 1
#define HALF 0.5
#define QUARTER 0.25
#define EIGHTH 0.25
#define SIXTEENTH 0.625

int tune[]=
{
NTD3,NTD3,NTD4,NTD5,
NTD5,NTD4,NTD3,NTD2,
NTD1,NTD1,NTD2,NTD3,
NTD3,NTD2,NTD2,
NTD3,NTD3,NTD4,NTD5,
NTD5,NTD4,NTD3,NTD2,
NTD1,NTD1,NTD2,NTD3,
NTD2,NTD1,NTD1,
NTD2,NTD2,NTD3,NTD1,
NTD2,NTD3,NTD4,NTD3,NTD1,
NTD2,NTD3,NTD4,NTD3,NTD2,
NTD1,NTD2,NTDL5,NTD0,
NTD3,NTD3,NTD4,NTD5,
NTD5,NTD4,NTD3,NTD4,NTD2,
NTD1,NTD1,NTD2,NTD3,
NTD2,NTD1,NTD1
};

float durt[]=
{
1,1,1,1,
1,1,1,1,
1,1,1,1,
1+0.5,0.5,1+1,
1,1,1,1,
1,1,1,1,
1,1,1,1,
1+0.5,0.5,1+1,
1,1,1,1,
```

```
1,0.5,0.5,1,1,
1,0.5,0.5,1,1,
1,1,1,1,
1,1,1,1,
1,1,1,0.5,0.5,
1,1,1,1,
1+0.5,0.5,1+1,
};

int length;
int tonepin=5;
int ledp=4;

void setup()
{
pinMode(tonepin,OUTPUT);
pinMode(ledp,OUTPUT);
length=sizeof(tune)/sizeof(tune[0]);
}

void loop()
{
for(int x=0;x<length;x++)
{
tone(tonepin,tune[x]);
digitalWrite(ledp, HIGH);
delay(400*durt[x]);
digitalWrite(ledp, LOW);
delay(100*durt[x]);
noTone(tonepin);

}
delay(4000);
}
```

- Code analysis

```
#define NTD
```

The definition of the D key frequency, the frequency is divided into bass, midrange and treble.

```
#define WHOLE 1
#define HALF 0.5
#define QUARTER 0.25
#define EIGHTH 0.25
#define SIXTEENTH 0.625
```

Note: The rhythm is divided into one beat, half beat, ¼ beat, ⅛ beat, we specify the note time of one beat as 1; half beat as 0.5; ¼ beat as 0.25; ⅛ beat as 0.125.

```
int tune[]=...
```

List the frequencies according to the spectrum.

```
float durt[]=...
```

List the beats according to the frequency spectrum.

```
delay(100*durt[x]);
```

Control the LED to turn on and off respectively.

- **Demonstration effect and walking output result:**

The buzzer will beep and the LED module will flash at the same frequency.

- Split guide

Connect the LED module to the digital signal interface **D4** of RGBDuino Plus, and connect the buzzer to the digital signal interface **D5** of RGBDuino Plus.

2. Intelligent sound and light sensor table lamp

- **Project introduction:** As the name says, this project is to make a smart desk lamp controlled by sound and light. We need to use LED modules. Of course, sound sensors and light sensors are also indispensable. In this way, we can realize the function of a smart desk lamp: when a sound is made, the light will light up; if the environment becomes dark, the light bulb will automatically light up.
- Components involved
- RGBDuino Plus
- LED module (D4)
- Light sensor module (A6)
- Sound sensor module (D5)
- PH2.0 data cable (only applicable when the module is split)
- **Hardware connection**
- **Module connection**-The default connection is through the PCB embossed hole. -Connect RGBDuino Plus to the computer via a USB cable.
- Program code
- Open the Arduino IDE.

- Copy the following code and click "Compile (✓)" to check for syntax errors. If there is no error in the compilation, you can "upload (→)" the code.

```
//light Induction Desk Lamp
int soundPin = A2; // Analog sound sensor is to be attached to analog
int lightPin = A6; //Analog light sensor is to be attached to analog
int ledPin = 4; // Digital LED is to be attached to digital

void setup() {
pinMode(ledPin, OUTPUT);
pinMode(lightPin, INPUT);
pinMode(soundPin, INPUT);
}

void loop(){
int soundState = analogRead(soundPin); // Read sound sensor's value
int lightState = analogRead(lightPin); // Read light sensor's value
// if the sound sensor's value is greater than 500 or the sound sensor's
is less than 200, the light will be on.
//Otherwise, the light will be turned off
if (soundState > 500 || lightState < 200) {
digitalWrite(ledPin, HIGH);
delay(500); //You can add the "//" to remove the delay
}else{
digitalWrite(ledPin, LOW);
}
}
```

The parenthesis is a logical expression. **&&** and **||** are widely used in logical expressions. The usual usage is **if (expression 1 || expression 2)** and **if (expression 1 && expression 2)**.

|| stands for "**or**". If one of the conditions is satisfied and the condition of if judgment is satisfied, the entire expression is true.

&& means "**and**", and the statement in if{} is executed only when all expressions in parentheses are true.

- **Demonstration effect and serial printing result:**

If the surrounding sound is loud enough or the light intensity is low, the LED module will be brighter.

If the module is separated from the main board, you need to use a Grove data cable to connect the LED module and the digital interface **D4** of RGBDuino Plus, and connect the light sensor to the analog signal interface **A1** of RGBDuino Plus. Finally, connect the sound sensor to the analog signal interface **A2** of RGBDuino Plus.