



IOT Training Kit

20210904.V1

Directory

Lesson 1 Installing IDE.....	1
1.1 Introduction.....	1
1.2 Installing Arduino.....	3
1.3 Installing Libraries.....	6
1.4 Arduino.....	8
1.5 Introduction to RGB Nano.....	8
Lesson 2 Light LED.....	11
2.1 Overview.....	11
2.2 working principle.....	11
2.3 Connection description.....	12
2.4 Code explanation.....	12
2.5 Upload code.....	13
Lesson 3 Button control LED.....	15
3.1 Overview.....	15
3.2 Connection description.....	15
3.3 Code explanation.....	15
Lesson 4 active buzzer.....	17
4.1 Overview.....	17
4.2 Connection description.....	17
4.3 Code explanation.....	18
Lesson 5 passive buzzer.....	19
5.1 Overview.....	19
5.2 Connection description.....	19
5.3 Code explanation.....	20
Lesson 6 Traffic Light.....	21
6.1 Overview.....	21
6.2 working principle.....	21
6.3 Wiring schematic.....	22

6.4 Code explanation.....	23
Lesson 7 Running water light.....	25
7.1 Overview.....	25
7.2 working principle.....	25
7.3 Wiring schematic.....	26
7.4 Code explanation.....	26
Lesson 8 WS2812B.....	28
8.1 Overview.....	28
8.2 working principle.....	28
8.3 Characteristics.....	29
8.4 Wiring schematic.....	30
8.5 Code explanation.....	30
Lesson 9 Gradient RGB.....	32
9.1 Overview.....	32
9.2 Working principle.....	32
9.3 Wiring schematic.....	33
9.4 Code explanation.....	34
Lesson 10 DS1307.....	36
10.1 Overview.....	36
10.2 LCD1602 Introduction.....	36
10.3 DS1307 Introduction.....	37
10.4 Wiring schematic.....	38
10.5 Code explanation.....	38
.....	40
Lesson 11 Show temp.....	41
11.1 Overview.....	41
11.2 Analog Temperature Sensor Introduction.....	41
11.3 Wiring schematic.....	42
11.4 Code explanation.....	42

Lesson 12 Show temp and humi.....	44
12.1 Overview.....	44
12.2 Wiring schematic.....	44
12.3 Code explanation.....	45
Lesson 13 Ultrasonic module.....	47
13.1 Overview.....	47
13.2 Ultrasonic sensor Introduction.....	47
13.3 Wiring schematic.....	48
13.4 Code explanation.....	48
Lesson 14 Photosensitive resistance.....	50
14.1 Overview.....	50
14.2 Component Introduction.....	50
14.3 Connection Diagram.....	52
14.4 Wiring schematic.....	53
14.5 Code explanation.....	53
Lesson 15 Rotary encoder control RGB.....	55
15.1 Overview.....	55
15.2 Project wiring diagram.....	55
15.3 Code explanation.....	55
Lesson 16 NRF24L01 launch.....	57
16.1 Overview.....	57
16.2 Project wiring diagram.....	57
16.3 Code explanation.....	57
Lesson 17 Infrared control LED.....	60
17.1 Overview.....	60
17.2 Project wiring diagram.....	60
17.3 Code explanation.....	61
Lesson 18 Infrared control RGB.....	63
18.1 Overview.....	63

18.2 Project wiring diagram.....	63
18.3 Code explanation.....	63
Lesson 19 Bluetooth control RGB.....	66
19.1 Overview.....	66
19.2 Connection description.....	66
19.3 Code explanation.....	67
19.4 Bluetooth remote control.....	69
Lesson 20 ESP8266 Development board.....	71
20.1 introduction:.....	71
The ESP8266 is a Wi-Fi module ideal for Internet of Things and home automation projects. This article is a beginner's guide to the ESP8266 development board.....	71
20.2 ESP8266 specifications.....	71
20.3 ESP8266 version.....	72
20.4 NodeMCU pin arrangement peripherals.....	73
20.5 What pins are used in NODEMCU ESP8266?.....	73
Lesson 21 Installing the ESP8266 development board in the Arduino IDE.....	75
21.1 Install the ESP8266 plug-in in the Arduino IDE.....	75
21.2 Test the installation.....	78
21.3 Wiring Diagram.....	80
Lesson 22 ESP8266 NodeMCU WiFi control traffic light module.....	81
22.1 Asynchronous Network Server.....	81
22.2 Schematic Diagram:.....	82
22.3 Wiring Diagram.....	83
22.4 ESP asynchronous Web server code.....	85
22.5 How the code works.....	90
Lesson 23 ESP8266 Node MCU button control LED.....	101
23.1 ESP8266 NodeMCU controls digital output.....	101
23.2 Project example.....	101

23.3 Wiring diagram:.....	102
23.4 Working Code:.....	102
23.5	103
23.6 Upload code.....	105
23.7 Object diagram:.....	105
Lesson 24 ESP8266 Controlling LED Brightness (PWM).....	106
24.1 ESP8266 NodeMCU PWM.....	106
24.2 Schematic diagram:.....	108
24.3 Upload code:.....	110
24.4 Wiring Diagram:.....	110
Lesson 25 ESP8266 Node MCU Web Server Control LED Brightness (PWM)....	111
25.1 Working Code:.....	112
25.2 Code working principle:.....	116
25.3 Build a web page.....	117
25.4 Web server Demo.....	125
Lesson 26 ESP8266 controls WS2812 lights via Blinker.....	126
26.1 Arduino configuration.....	126
26.2 Working Code:.....	128
26.3 App Connection Configuration.....	134
26.4 The wiring diagram:.....	139
Lesson 27 ESP8266 Nodemcu displays temperature and humidity in combination with Blinker.....	140
27.1 DHT11 Sensor:.....	140
27.2 Install the library.....	141
27.3Working code explanation:.....	141
27.4 Wiring Diagram.....	144
Lesson 28 ESP8266 Nodem combined with HC-SR04 Ultrasonic Ranging.....	148
28.1 Ultrasonic transducer.....	148
28.2 Wiring Diagram.....	149

Lesson 1 Installing IDE

1.1 Introduction

The Arduino Integrated Development Environment (IDE) is the software side of the Arduino platform.

In this lesson, you will learn how to setup your computer to use Arduino and how to set about the lessons that follow.

The Arduino software that you will use to program your Arduino is available for Windows, Mac and Linux. The installation process is different for all three platforms and unfortunately there is a certain amount of manual work to install the software.

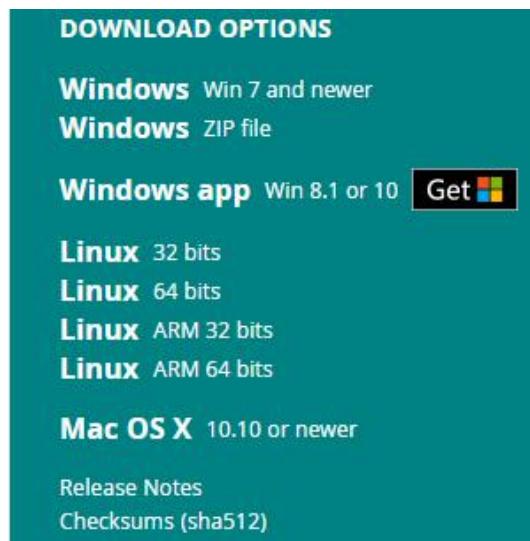
STEP 1: Go to <https://www.arduino.cc/en/software>.



The screenshot shows the Arduino IDE download page. On the left, there's a teal circular icon with a white infinity symbol and a plus sign. To its right, the text "Arduino IDE 1.8.13" is displayed. Below this, a paragraph explains that the open-source Arduino Software (IDE) makes it easy to write code and upload it to the board, compatible with any Arduino board. It links to the "Getting Started" page for installation instructions. Another section, "SOURCE CODE", provides information about the GitHub repository and PGP-signed source code archives. On the right, a teal sidebar titled "DOWNLOAD OPTIONS" lists download links for Windows (Win 7 and newer, ZIP file, and Windows app), Linux (32-bit and 64-bit), and Mac OS X (10.10 or newer). It also includes links for "Release Notes" and "Checksums (sha512)".

The version available at this website is usually the latest version, and the actual version may be newer than the version in the picture.

STEP2 : Download the development software that is compatible with the operating system of your computer. Take Windows as an example here.



Click Windows Installer.

Contribute to the Arduino Software

Consider supporting the Arduino Software by contributing to its development. (US tax payers, please note this contribution is not tax deductible). Learn more on how your contribution will be used.

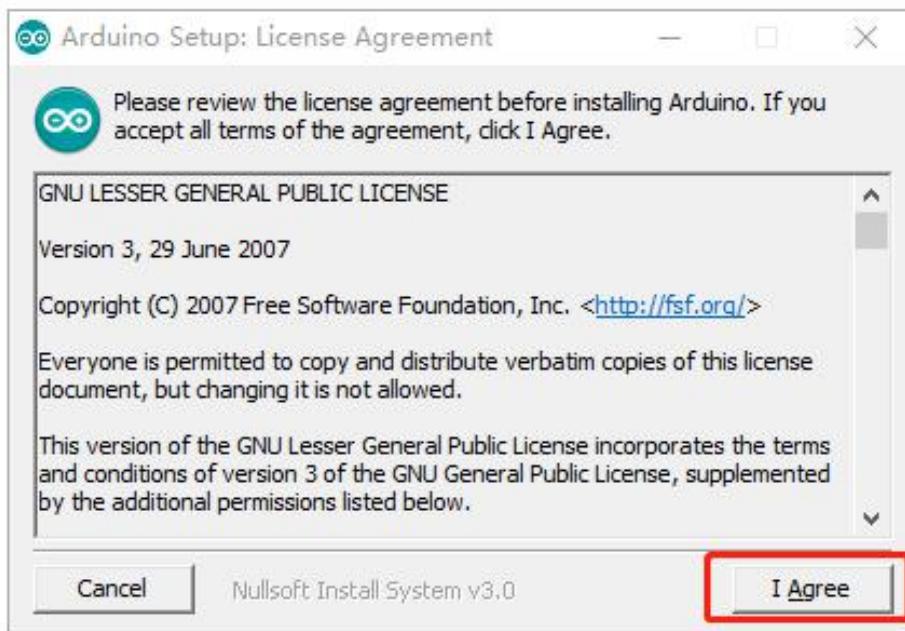


Click JUST DOWNLOAD.

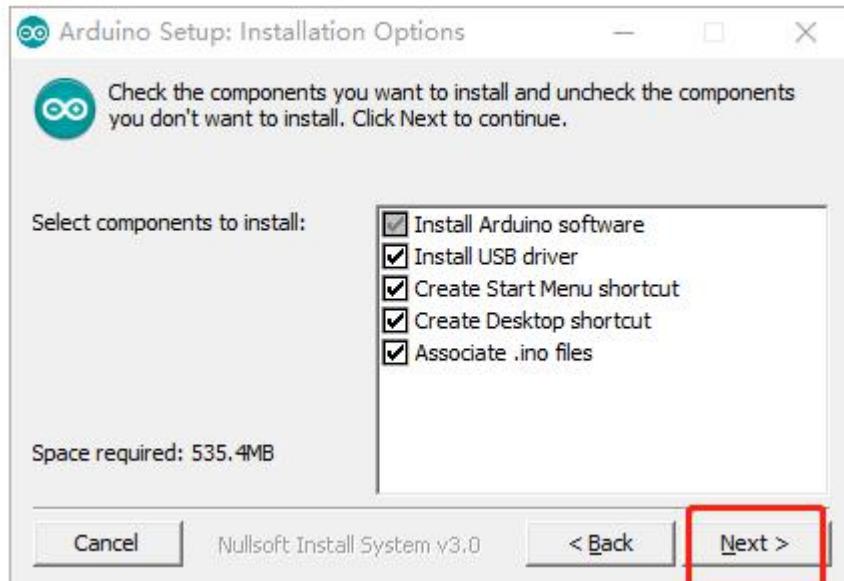
Also version 1.8.9is available in the material we provided, and the versions of our materials are the latest versions when this course was made.

1.2 Installing Arduino

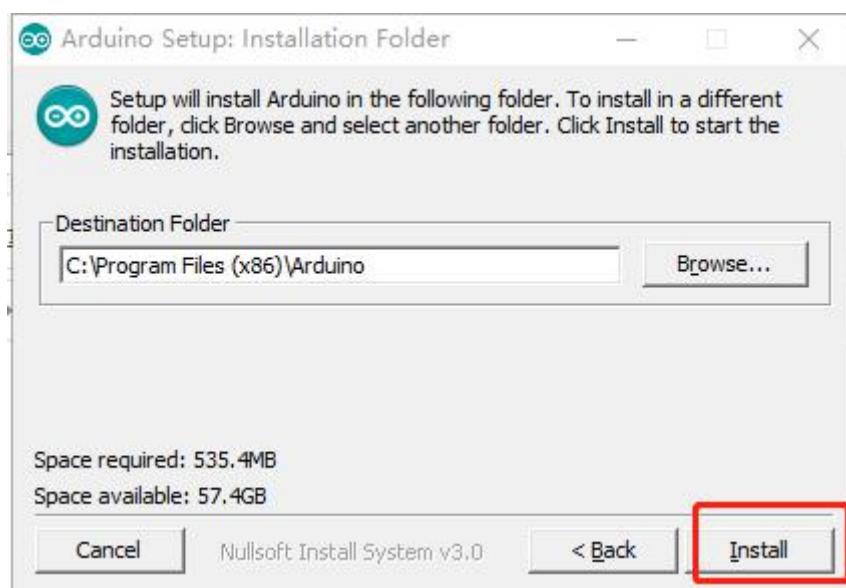
 arduino-1.8.13-windows



Click I Agree to see the following interface.

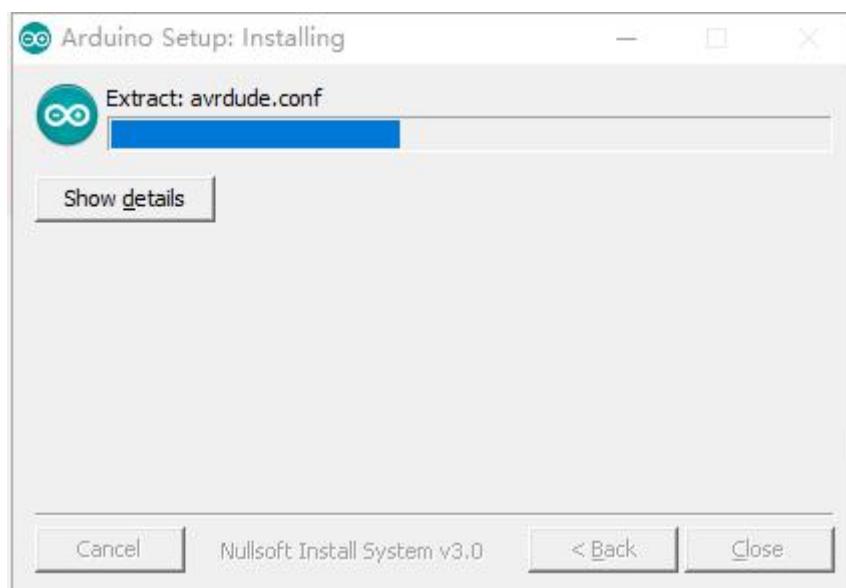


Click Next

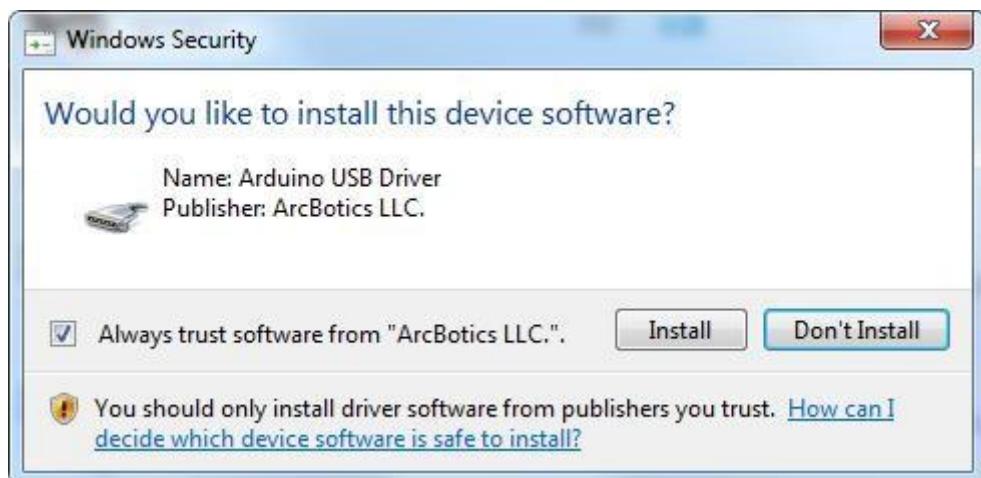


You can press **Browse...** to choose an installation path or directly type in the directory you want.

Click **Install** to initiate installation.



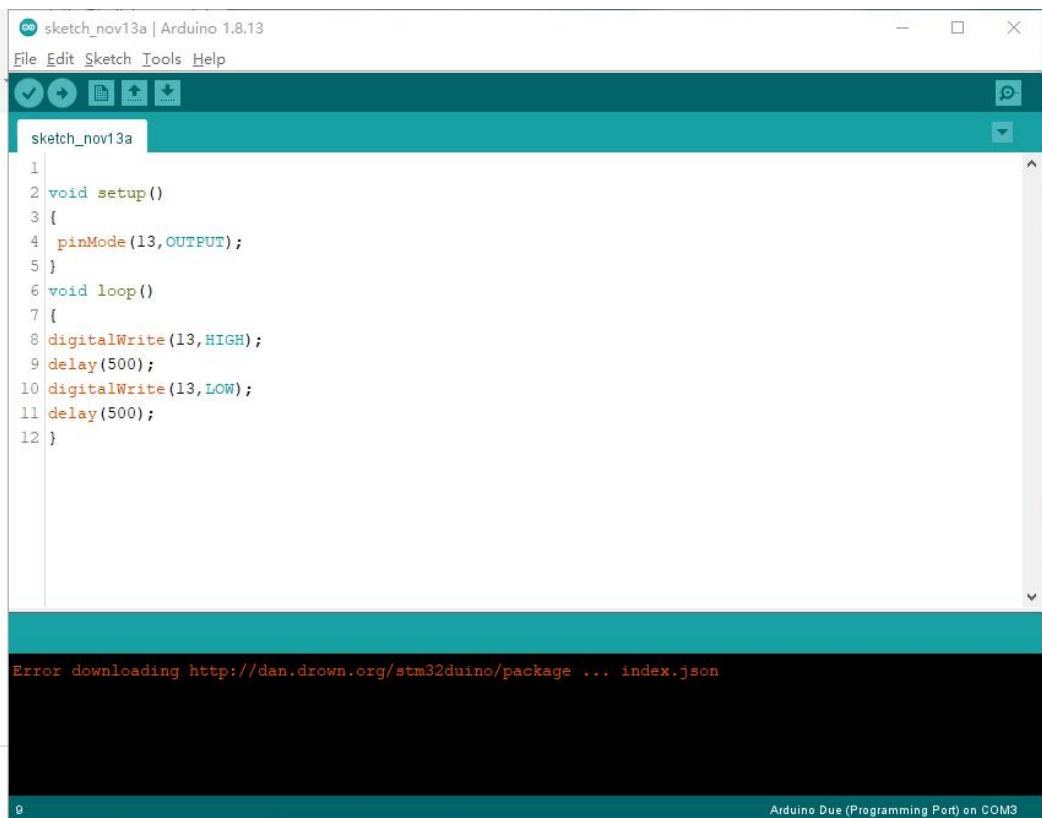
Finally, the following interface appears, click Install to finish the installation.



Next, the following icon appears on the desktop.



Double-click to enter the desired development environment.



Installing Arduino (Mac OS X)

Download and Unzip the zip file, double click the Arduino.app to enter Arduino IDE; the system will ask you to install Java runtime library if you don't have it in your computer. Once the installation is complete you can run the Arduino IDE.

Installing Arduino (Linux)

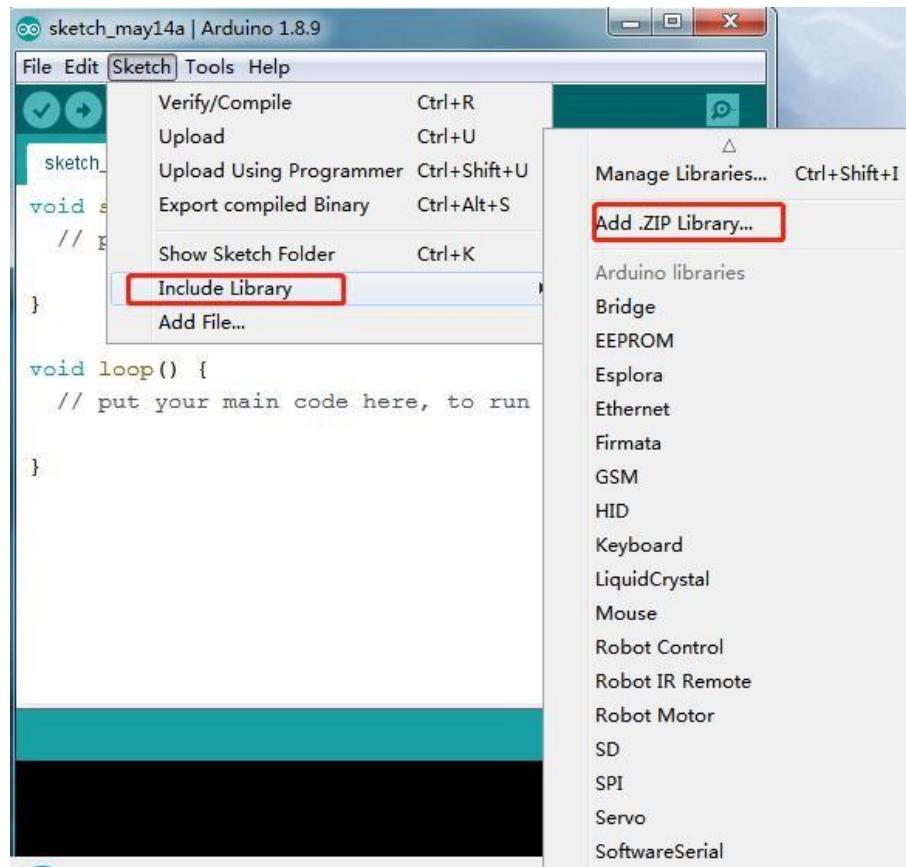
You will have to use the make install command. If you are using the Ubuntu system, it is recommended to install Arduino IDE from the software center of Ubuntu.

Installing Additional Arduino Libraries. Once you are comfortable with the Arduino software and using the built-in functions, you may want to extend the ability of your Arduino with additional libraries.

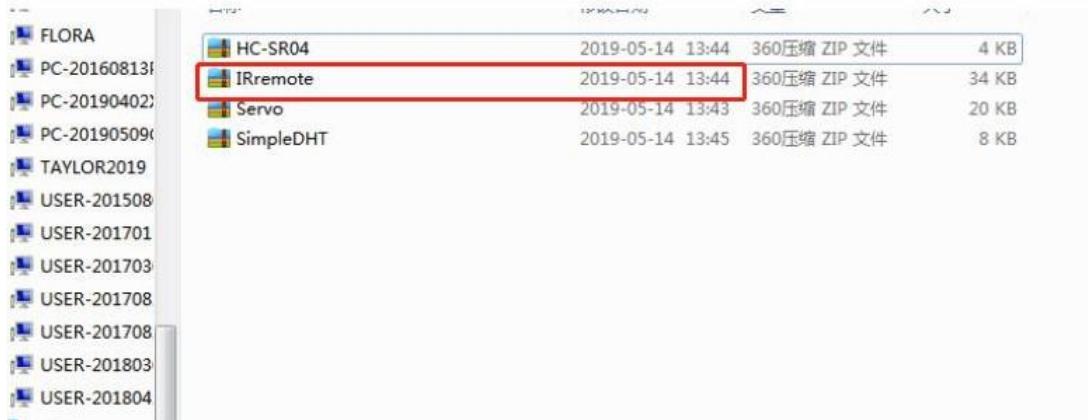
1.3 Installing Libraries

Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. For example, the built-in Liquid Crystal library makes it easy to talk to character LCD displays. There are hundreds of additional libraries available on the Internet for download. The built-in libraries and some of these additional libraries are listed in the reference. To use the additional libraries, you will need to install them.

How to Install a Library? Using the Library Manager. To install a new library into your Arduino IDE you can use the Library Manager (available from IDE version 1.8.9). Open the IDE and click to the "Sketch" menu and then Include Library > Manage Libraries.



and then Include Library > Manage Libraries.



Example: IRremote

Open Arduino software - project - load library - add a .zip library.

Add method two:

Copy the library folder to the Libraries folder in the Arduino installation directory. Restart Arduino and the added library will take effect.

1.4 Arduino

Arduino is an open source electronic platform based on easy-to-use hardware and software. Suitable for anyone working on interactive projects. Usually, an Arduino project consists of circuits and codes.

The Arduino board is a circuit board that integrates a microcontroller, input and output interfaces, etc. The Arduino board can use sensors to sense the environment and receive user operations to control LEDs, and so on. We just need to assemble the circuit and write the code. Currently, there are several models of Arduino development boards, and the codes between different types of development boards are common (due to different hardware, some development boards may not be fully compatible). Popular main control boards include.

1.5 Introduction to RGB Nano

The 14 digital ports of RGB Nano can be used as digital input or output, defined by pinmode() in the program, and controlled by digitalWrite and digitalread() function blocks. They work at 5V. Each port provides output current or receives 40 mA current. There is a pull-up resistor inside with a resistance value of 20-50 kOhms. Other terminals have special definitions.

Serial: 0 (Rx) and 1 (Tx). Used to receive (Rx) and transmit (Tx) TTL serial data.

External interrupt: terminals 2 and 3. These external interfaces can be configured to generate interrupts later, can be triggered when an external low level occurs, or when a rising edge and a falling edge occur. For more information, see the attachinterrupt() function.

PWM: 3, 5, 6, 9, 10, 11, provide 8-bit PWM output, use the analogwrite() function.

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication. Although the hardware supports them, they are not included in the Arduino software.

Led: 2-13, built-in LED, connected to pin 2-13, when this pin outputs high

voltage, the LED is on, when outputting low voltage, the LED is off.

button: 2, is a built-in button, connected to pin 2, this pin can be used as a pull-up input, when the software is configured, it can detect whether the button is pressed.

buzzer: 8 is the built-in pin, connected to pin 8. A passive buzzer module is connected to this pin. When this pin outputs a frequency level, the buzzer can make a different sound.

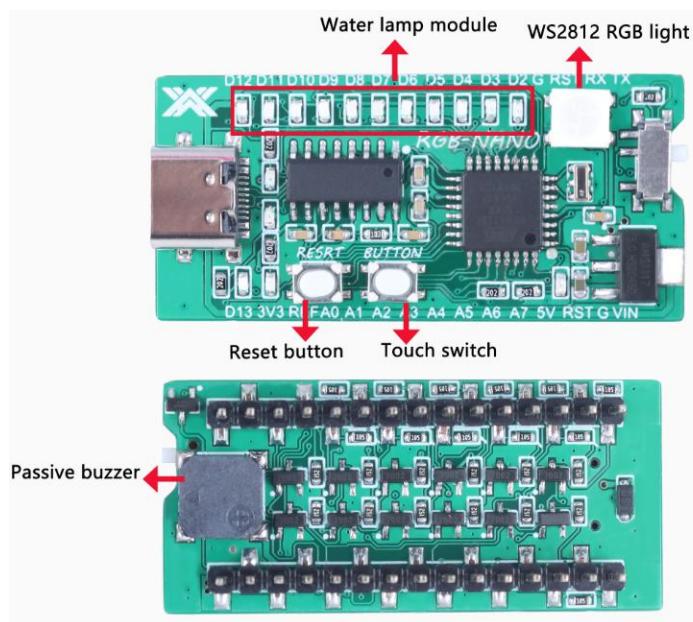
RGB: 13 is a built-in pin, connect to pin 13, this pin integrates WS2812RGB light, you can use software configuration to make it emit different colors

RGB Nano has eight analog inputs, each with a resolution of 10 bits (ie 1024 different possibilities). By default, the measured voltage to ground is 5V. Of course, its upper limit can also be modified by the analogreference() function. Analog pins 6 and 7 cannot be used as digital ports. In addition, some ports have many special functions.

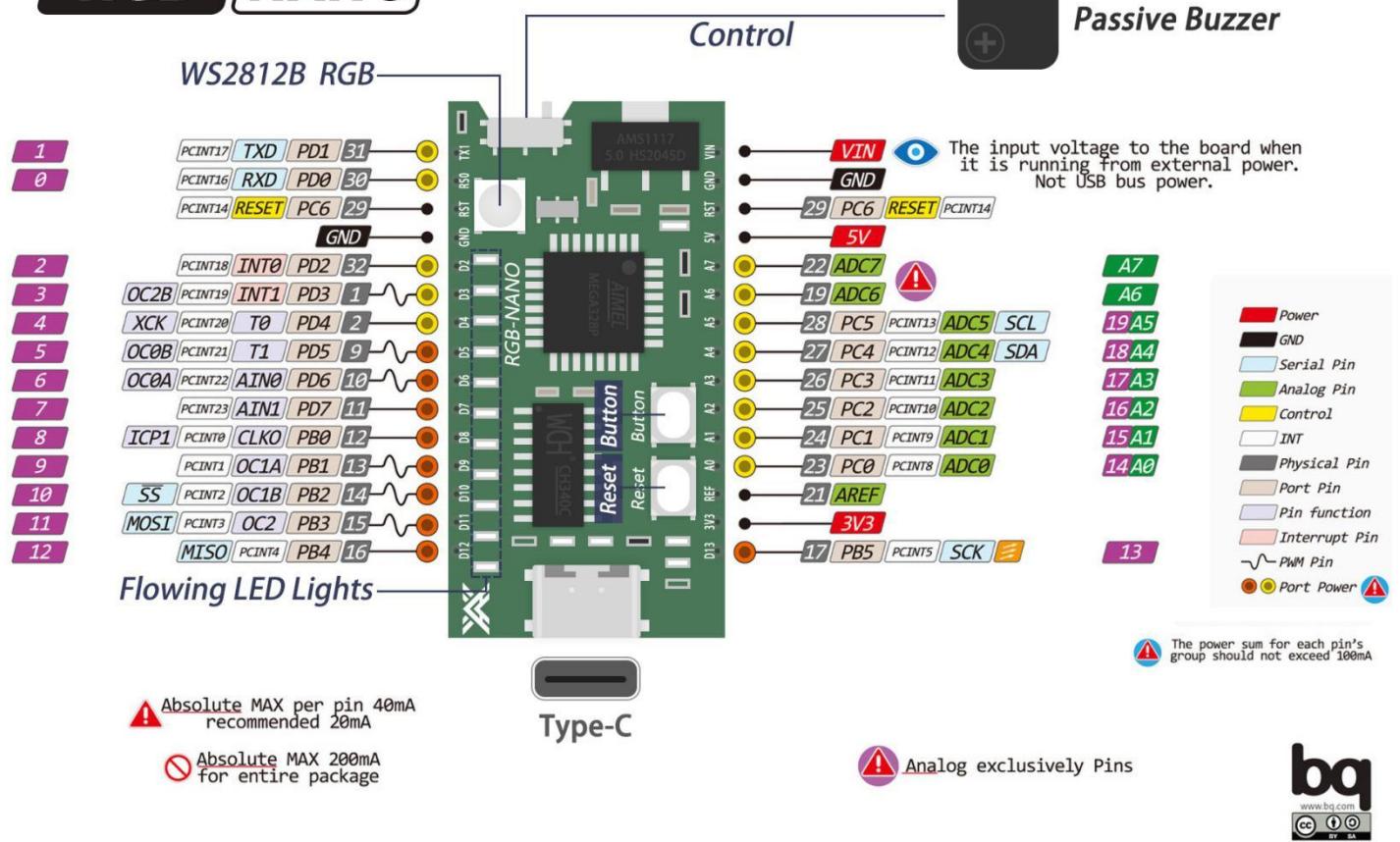
I2C: A4 (SDA) and A5 (SCL). There are other ports on the board.

Aref: Reference voltage of analog input, used with analogreference().

Reset: Pull down the potential and reset the microprocessor. After pressing the button, the whole system can be reset.



RGB NANO



Lesson 2 Light LED

2.1 Overview

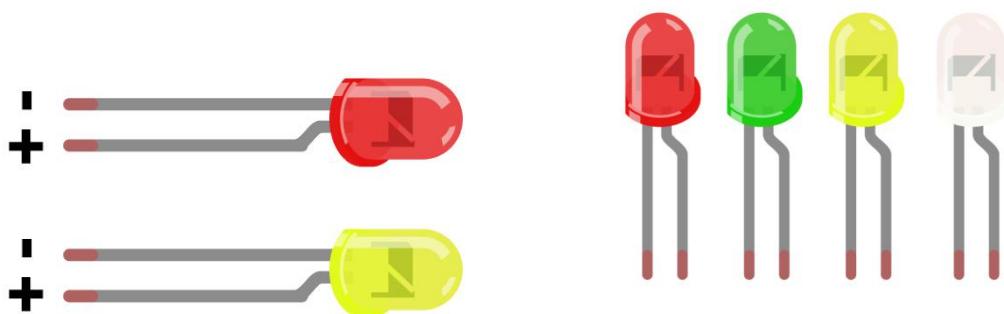
Through this project, you can learn how to use RGB Nano to light up a 10mm LED module. After downloading the program and connecting the line, you will see that the LED light is successfully lit. If it is not lit, you need to check whether the line is correctly connected and check whether the pin number of the connected micro controller corresponds to it.

2.2 working principle

LED (Light Emitting Diode), which converts electrical energy into light energy, also has unidirectional conductivity and a reverse breakdown voltage of about 5V. Its forward volt-ampere characteristic curve is very steep, and the current-limiting resistor must be connected in series. In a 5V circuit, a resistor of about 400 ohms is generally used. The longer of the two pins of the LED is the positive pole. There are two ways to connect, when the positive pole of the led through the current limiting resistor and Arduino.The I/O port is connected and the other end is grounded. At this time, when the Arduino output is high, the led is lit, and when the output is low, the led is off.

When the negative pole of the led is connected to the I/O port of the Arduino, the other end is connected to the 5V voltage through the current limiting+ resistor. At this time.

When the output is low, the led is lit, and when the output is high, the led is off.



If you do not use a resistor with an LED, then it may well be destroyed almost immediately, as too much current will flow through, heating it and destroying the 'junction' where the light is produced.

There are two ways to tell which is the positive lead of the LED and which the negative.

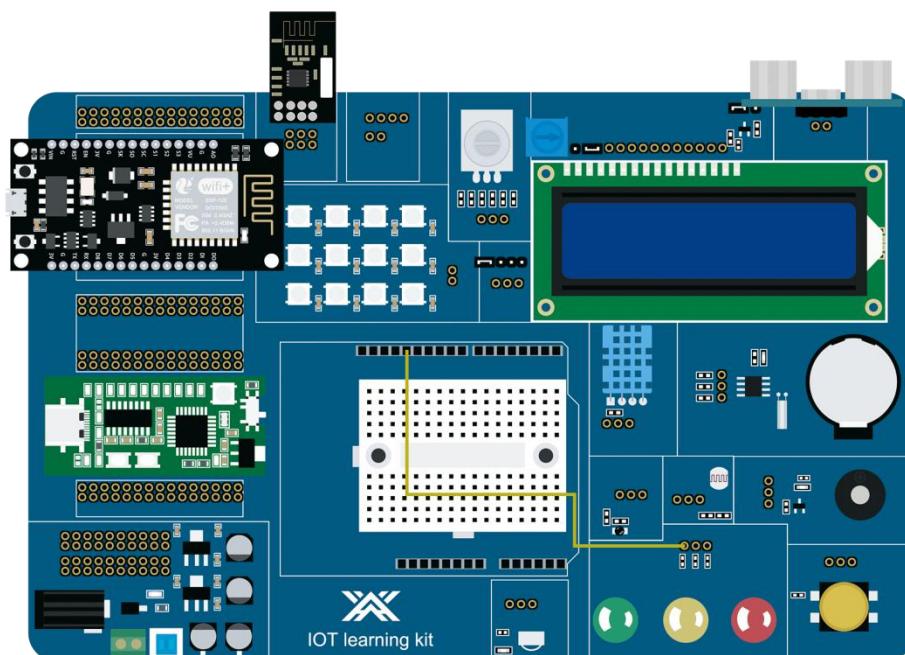
Firstly, the positive lead is longer.

Secondly, where the negative lead enters the body of the LED, there is a flat edge to the case of the LED.

If you happen to have an LED that has a flat side next to the longer lead, you should assume that the longer lead is positive.

2.3 Connection description

Use the dupont line to lead the D13 pin of the micro controller to any interface of the JP12 header, plug it in, and the line is successfully connected. The wiring diagram is as follows.



2.4 Code explanation

Define LED signal enable pin.

```
#define LED 13 //Define output pin
```

Function initialization, define 13 pin as output.

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

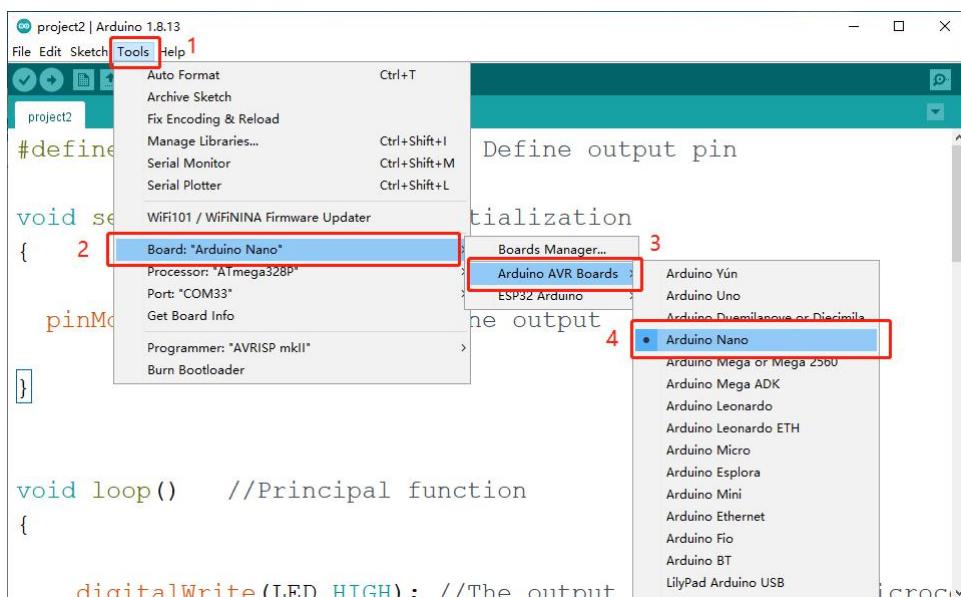
}
```

Main function, let 13 pin output high level.

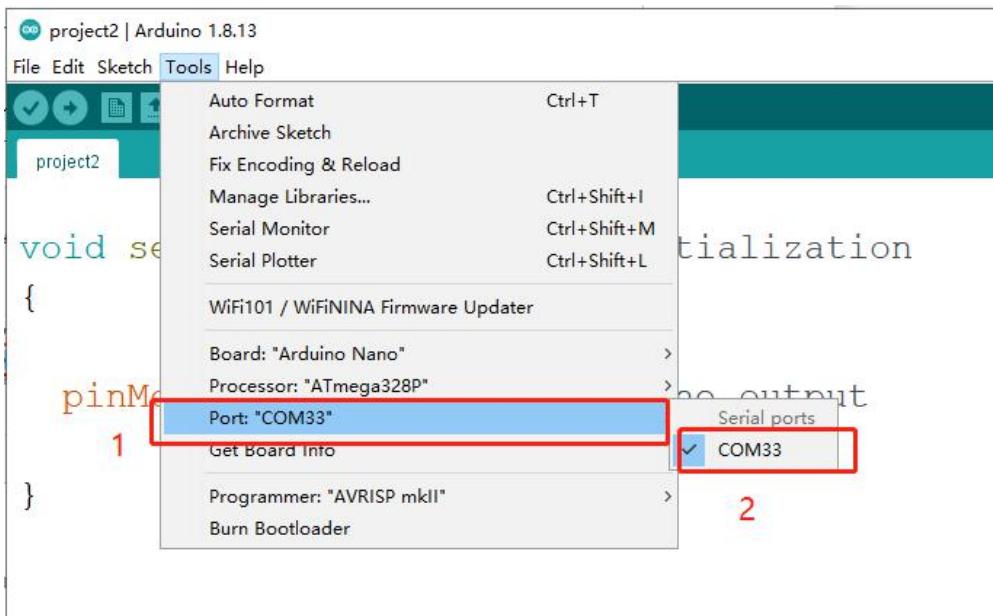
```
void loop() {
    // put your main code here, to run repeatedly:
    /*Since the negative pole of our LED pin has been
    connected, we only need to use the control pin of
    Arduino to output the high level, which is the positive
    pole, and the LED will be lit */
    digitalWrite(LED ,HIGH);
}
```

2.5 Upload code

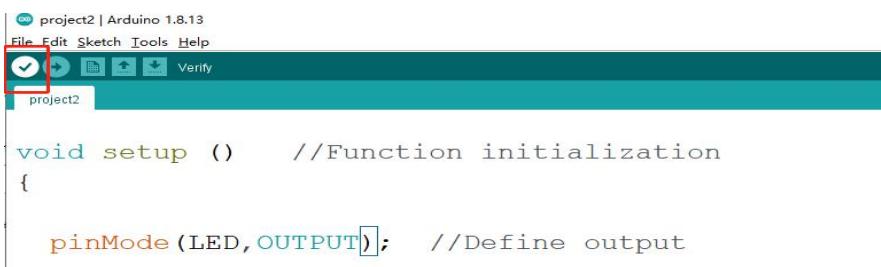
Choose NANO development board.



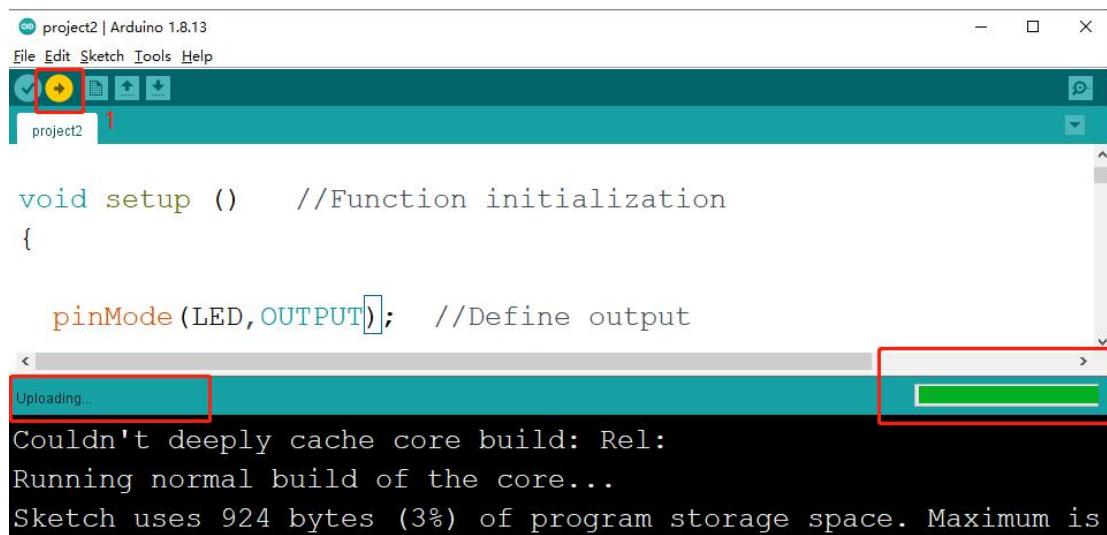
Select port.



Click compile.



Click upload.



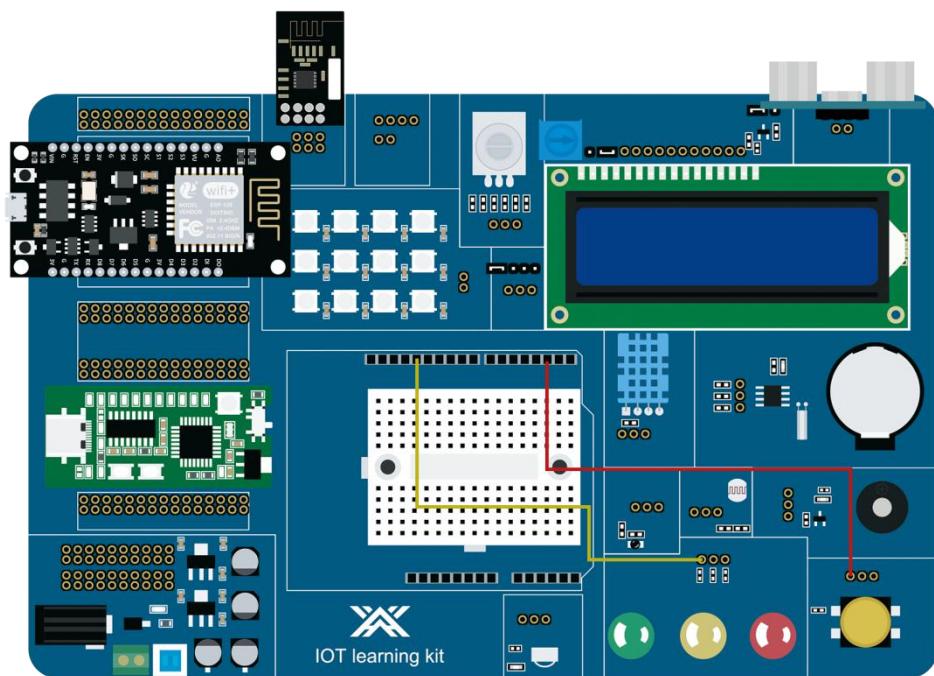
Lesson 3 Button control LED

3.1 Overview

Through this project, you can learn how to use the input of the RGB Nano control button to light up a 10mm LED light. After downloading the program and connecting the line, you need to press the WS1 button. After pressing it, you will see the LED light is successfully lit. After pressing it again, the LED light will go out. If it is not lit and extinguished, you need to check whether the circuit is correctly connected, and check whether the pin number of the connected microcontroller corresponds to it.

3.2 Connection description

Use DuPont wire to lead the D13 pin of the single-chip microcomputer to any interface of the JP12 header, plug it in and the connection is successful, connect the D2 pin of the single-chip microcomputer to the S port on the header JP1, and the button wiring is also completed. The wiring diagram is as follows.



3.3 Code explanation

Define LED signal enable pin.

```
#define LED 13
#define button 2
```

Define global variables, which are used as flag bits for buttons and LEDs, function initialization, define 13 pin as output, and define 2 pin as input State.

```
#define LED 13
#define button 2

int key_ok=0; //Define the data variables required by the project
int LED_en=0;
void setup() {
    // put your setup code here, to run once:
    pinMode(button,INPUT); //Define pin port work type
    pinMode(LED,OUTPUT);

}
```

The main function is to judge whether the button is pressed, after pressing it, judge whether it has been pressed according to the flag bit, and determine whether the LED is lit or extinguished according to the flag bit.

```
void loop() {
    // put your main code here, to run repeatedly:
    //Determine whether there is a button pressed, read the button level
    if(digitalRead(button))
    {
        if(key_ok) //Determine whether the button is pressed
        {
            key_ok = 0;
            if(LED_en)LED_en=0; //Determine whether the last flag bit is established
            else LED_en = 1;
        }
    }
    else
    {
        delay(20); //Delayed debounce
        if(!digitalRead(button)) key_ok = 1;
    }

    //When a button is pressed, the pin port status is reversed
    if(LED_en) digitalWrite(LED,HIGH);
    else digitalWrite (LED,LOW);

}
```

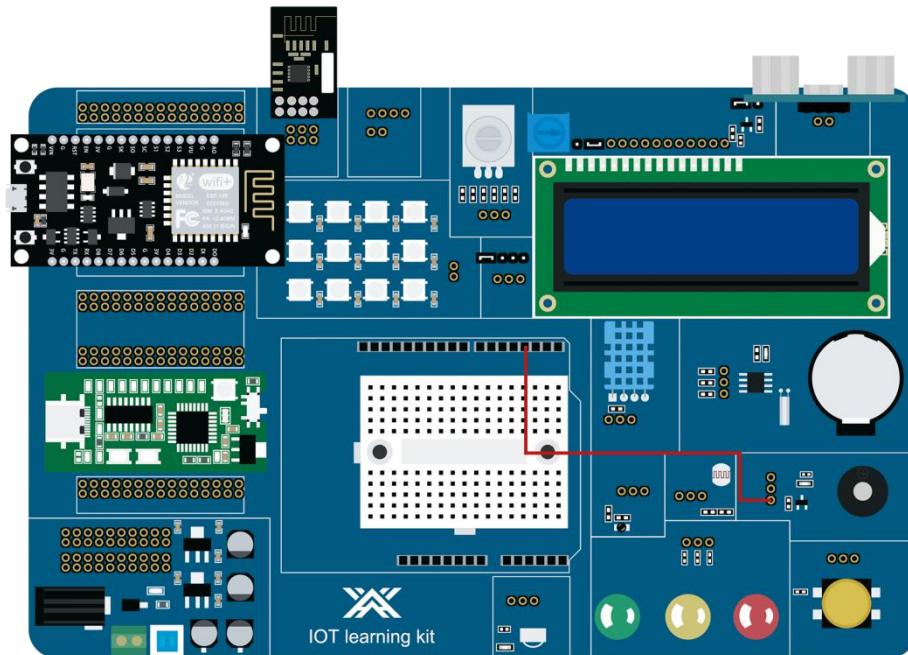
Lesson 4 active buzzer

4.1 Overview

Through this project, you can learn how to use RGB Nano to make the active buzzer sound an alarm. After downloading the program and connecting the line, you will hear an alarm sound from the buzzer. If the buzzer does not sound, you need to check whether the circuit is correctly connected, and check whether the pin number of the connected micro controller corresponds to it.

4.2 Connection description

Connect the D3 pin of the single-chip microcomputer to the S interface of the JP7 row seat with a Dupont cable, and the connection is successful when it is plugged in. The wiring diagram is as follows



4.3 Code explanation

Definition Active buzzer signal enable pin is the third pin.

```
#define Buzzer 3
```

The function is initialized, and pin 3 is defined as output.

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

}
```

The main function allows the micro controller to output a high level of 500MS and then a low level of 500MS, so that the active buzzer can sound an alarm.

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

}
```

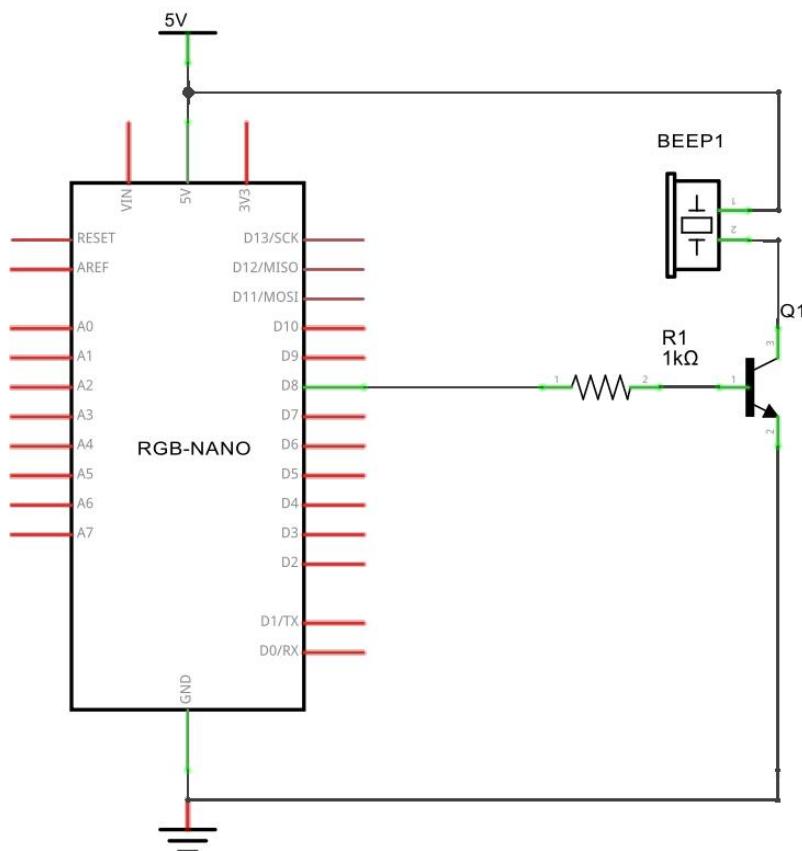
Lesson 5 passive buzzer

5.1 Overview

Through this project, you can learn how to use RGB Nano to make a passive buzzer sound an alarm. You can hear the sound after downloading the program. This is emitted by the passive buzzer, which is emitted once every 0.5S.

5.2 Connection description

No wiring is required in this lesson, because the passive buzzer has been integrated on the RGB Nano, and the corresponding pin of the passive buzzer is pin D8. If there is no sound when downloading the program, check whether there is a swing switch above the buzzer Turn on, this swing switch is the connection switch between the D8 pin and the passive buzzer, here you need to pay attention!The wiring diagram is as follows.



5.3 Code explanation

Define the passive buzzer signal enable pin as the D8 pin.

```
#define Buzzer 8
```

A global flag is defined, and the function is initialized at the same time, and pin 8 is defined as an output.

```
#define Buzzer 8
int beep_bit=0;
void setup() {
    // put your setup code here, to run once:
    pinMode(Buzzer,OUTPUT);

}
```

The main function is to make the single-chip microcomputer output a level frequency, which can make the passive buzzer sound. After a period of time, the passive buzzer can make an alarm sound. If you want to make the passive buzzer sound all the time , You can make the "beep_bit" variable always wait for 0.

```
void loop() {
    // put your main code here, to run repeatedly:
    if(beep_bit == 0)
    {
        for(int i=0;i<=3000;i++) //Let the pin send high and low levels at a frequency
        {
            digitalWrite(Buzzer,HIGH);
            delayMicroseconds(100);
            digitalWrite(Buzzer,LOW);
            delayMicroseconds(100);
        }
    }
    delay(500); //Let the sound last for a while, then turn it off
    if(beep_bit)beep_bit=0;
    else beep_bit=1;
}
```

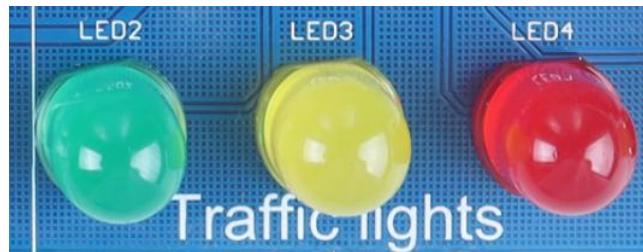
Lesson 6 Traffic Light

6.1 Overview

Above, we have completed the control experiment of a single small light. Next, let's do a slightly more complex traffic light experiment. In fact, smart friends can see that this experiment is to expand the experiment of a single small light into 3 colors, which can achieve our simulation of the traffic light experiment.

6.2 working principle

Signal light is an important part of traffic signal, and it is the basic language of road traffic.



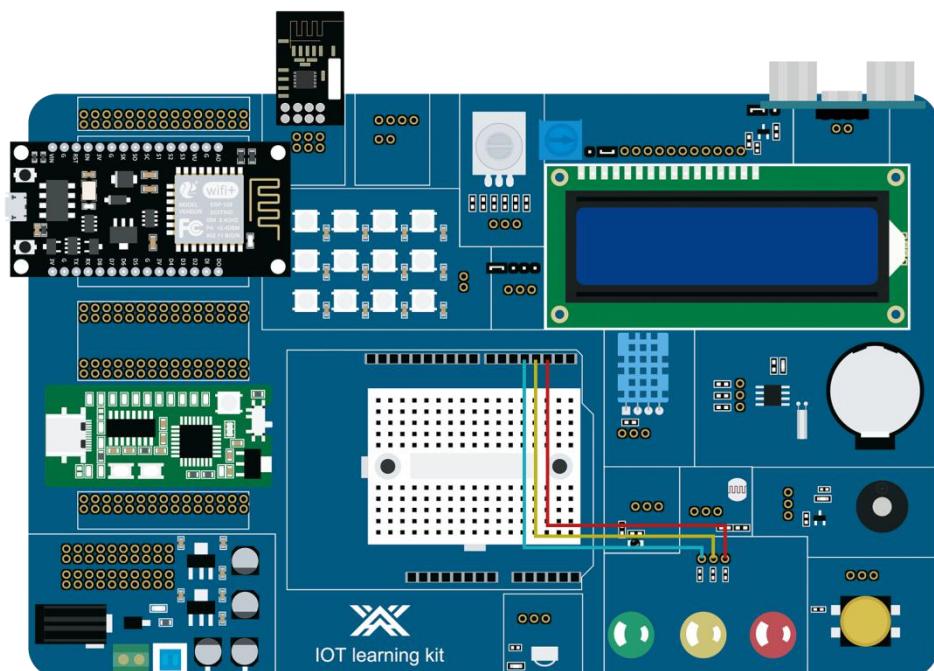
The traffic signal light consists of a red light (indicating no passage), a green light (indicating permission to pass) and a yellow light (indicating a warning).

It is divided into: motor vehicle signal light, non-motor vehicle signal light, pedestrian crossing signal light, lane signal light, direction indicator signal light, flashing warning signal light, road and railway plane crossing signal light.

Road traffic signal lamp is a category of traffic safety products, is to strengthen the road traffic management, reduce the occurrence of traffic accidents, improve the efficiency of road use, improve traffic conditions of an important tool.

It is suitable for crossing, T-word and other intersections. It is controlled by road traffic signal control machine to guide the safe and orderly passage of vehicles and pedestrians.

6.3 Wiring schematic



6.4 Code explanation

Set the number 2,3,4 ports to output mode as they are connected to the LED positive terminal.

```
void setup() {  
    // put your setup code here, to run once:  
    pinMode(2,OUTPUT); // green light  
    pinMode(3,OUTPUT); // Yellow light  
    pinMode(4,OUTPUT); // red light  
}
```

Set digital pin 2 to high and the rest to low with a delay of 5 seconds, leaving the green light on for 5 seconds.

```
void loop() {  
    // put your main code here, to run repeatedly:  
    digitalWrite(2,HIGH); // Open the green light  
    digitalWrite(3,LOW); // Close the yellow light  
    digitalWrite(4,LOW); // Close the red light  
    delay(5000); // Let the green light go on for five seconds
```

The green light flashes every 500 milliseconds for a total of three times

```
//The green light flashes every 500 milliseconds  
digitalWrite(2,HIGH);  
delay(500);  
digitalWrite(2,LOW);  
delay(500);  
digitalWrite(2,HIGH);  
delay(500);  
digitalWrite(2,LOW);  
delay(500);  
digitalWrite(2,HIGH);  
delay(500);
```

Turn on the yellow light and turn off the red and green light for 1 second. Then the red light goes on for five seconds.

```
//Yellow light for one second  
digitalWrite(2,LOW);  
digitalWrite(3,HIGH);  
digitalWrite(4,LOW);  
delay(1000);  
  
//The red light is on for five seconds  
digitalWrite(2,LOW);  
digitalWrite(3,LOW);  
digitalWrite(4,HIGH);  
delay(5000);
```

The red light flashes three times every 500 milliseconds.

```
//The red light blinks every 500 milliseconds  
digitalWrite(4,HIGH);  
delay(500);  
digitalWrite(4,LOW);  
delay(500);  
digitalWrite(4,HIGH);  
delay(500);  
digitalWrite(4,LOW);  
delay(500);  
digitalWrite(4,HIGH);  
delay(500);
```

Lesson 7 Running water light

7.1 Overview

In this course, you will learn how to control the led on and off to achieve the flow light function.

7.2 working principle

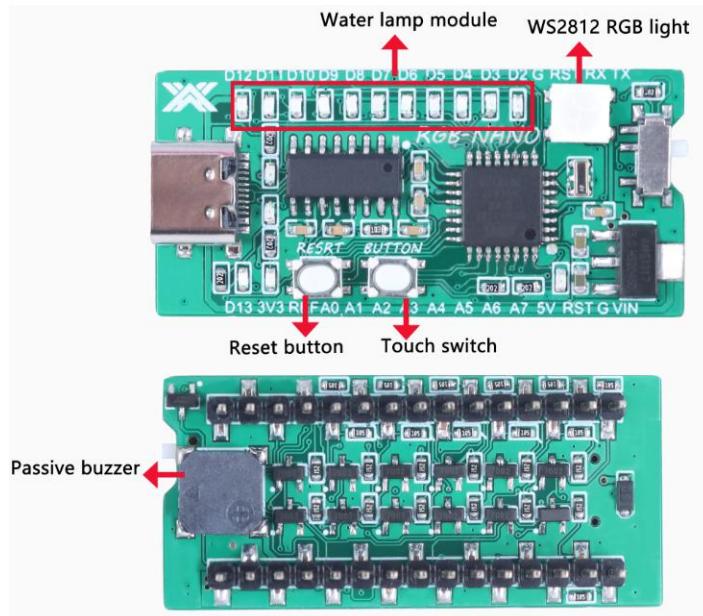
mall light on RGB-Nano, and can realize the function of light on, off and flashing, and the LED flashing time is set by itself.

When the pin output is low, the LED light will not be lit, when the pin output is high, the LED light will be lit.

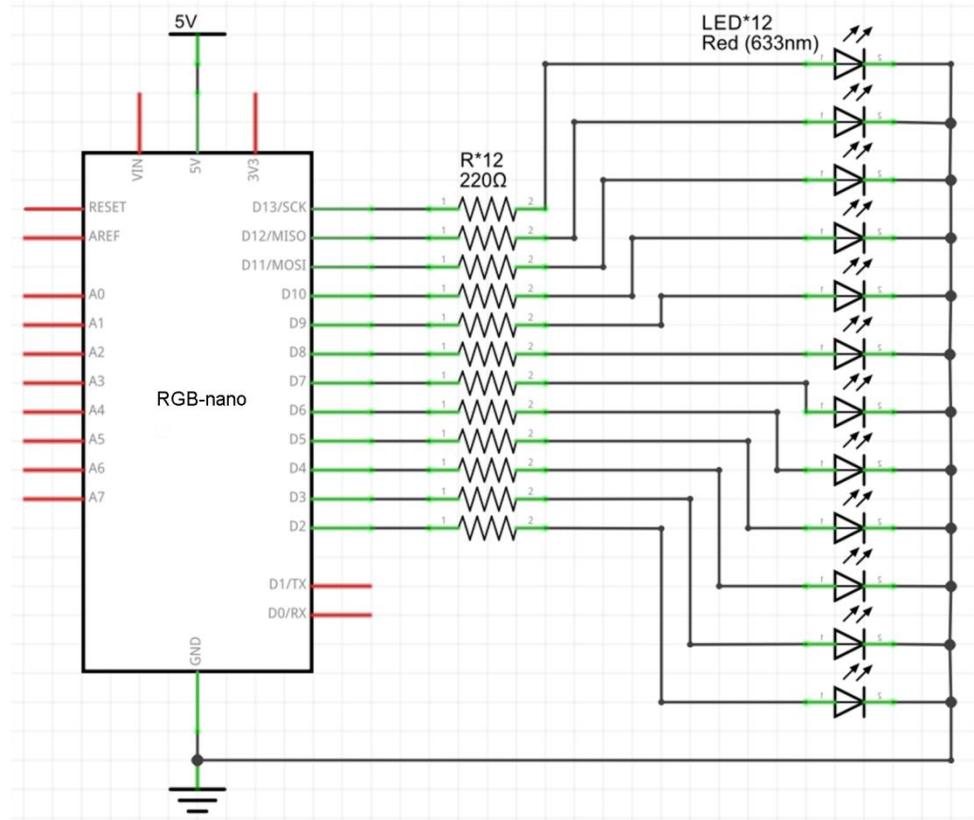
Multiple pin controls are required, so digital pins 2 through 13 are used here.

Control the light on and off time respectively, you can achieve the effect of water lamp.

Running water light effect: all the lights go out, and then lit one by one to the left, to carry out one by one to the right and faster at the beginning, as the change of time, more and more slow, to reach the slowest time flow will become soon, this process is circular, of course can also be other effects, can be set for yourself.



7.3 Wiring schematic



7.4 Code explanation

Define two time variables, and then set pins 2 through 13 to output mode.

```
int time1=0,time2=0; //Define two time summations

void setup()
{
    // Initialize the LED lamp pins, pin 2 to pin 13
    for (int i = 2; i <= 13; i++)
    {
        pinMode(i,OUTPUT);// Set to output mode
        digitalWrite(i, LOW); // Set the default level to low
    }
}
```

From the second light on for a period of time, then off for a period of time.

Turn on the third light, wait a while, turn off the third light, then turn on the fourth light, and so on.

```
void loop()
{
    for (int i = 2; i <= 13; i++)
    {
        time1++; //It keeps adding up
        digitalWrite(12, HIGH);
        digitalWrite(i, HIGH);
        delay(time1); //The duration of a high level

        digitalWrite(i, LOW);
        delay(time1); //The duration of a low level

        if(time1 == 100) time1 = 0; //Clear when the count reaches 100
        Serial.print("LED");
        Serial.println(i);
    }
}
```

From the 13th light on, wait for a period of time to turn off the 13th light, let the twelfth light on, wait for a period of time, turn off the second light, continue to light and turn off the next light.

```
for (int i = 13; i >= 2; i--)
{
    time2++;
    digitalWrite(2, HIGH);
    digitalWrite(i, HIGH);
    delay(time1); //The duration of a high level

    digitalWrite(i, LOW);
    delay(time1); //The duration of a low level

    if(time2 == 100) time2 = 0;
    Serial.print("LED");
    Serial.println(i);
}
}
```

Lesson 8 WS2812B

8.1 Overview

WS2812B is a chip with a built-in LED driver. One IO port can control multiple LEDs, brightness adjustment, color adjustment and other functions.

8.2 working principle

WS2812B is an intelligent external controlled LED light source integrating control circuit and luminous circuit.

The shape is the same as a 5050 LED lamp bead, and each element is a pixel point.

The pixel contains an intelligent digital interface data latch signal shaping amplifier drive circuit, also contains a high-precision internal oscillator and 5V voltage programmable fixed current control part, effectively ensure the pixel light color highly consistent.

The data protocol adopts the communication mode of single-line return to zero code. After the pixel is powered on and reset, the DIN end accepts the data transmitted from the controller. The first sent 24bit data is extracted by the first pixel and sent to the data latch inside the pixel.

After internal shaping and circuit shaping, the remaining data is amplified and forwarded to the next cascaded pixel through the DO port. Every transmission of a pixel, the signal is reduced by 24bit.

Automatic shaping and forwarding technology is adopted for pixels, so that the number of cascaded pixels is not limited by signal transmission, but only limited by signal transmission speed.

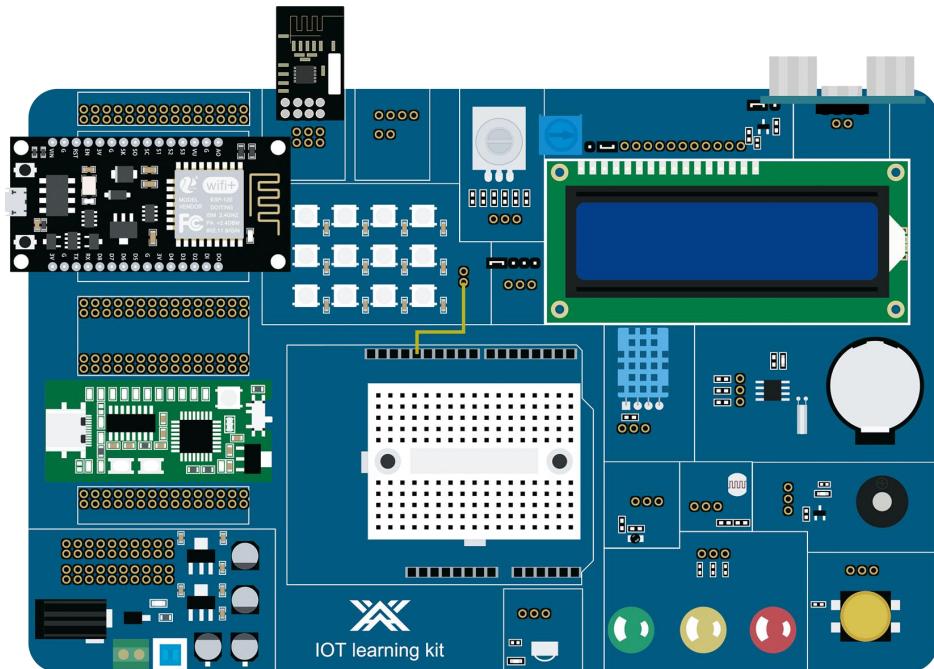
RGB is integrated on the development board, and the control pin of the RGB light is connected to pin 13 of the RGB-Nano board.



8.3 Characteristics

1. The control circuit and RGB chip are integrated in a 5050 package component, forming a complete external control pixel point.
2. Built-in signal shaping circuit, any pixel after receiving the signal through the waveform .
3. Type output, ensure that the line waveform distortion will not accumulate.
4. Built-in power on reset and power off reset circuit.
5. The three primary colors of each pixel can achieve 256-level brightness display, complete the full true color display of 16777216 colors, and the scanning frequency is not less than 400Hz/s.
6. Serial level connection port, through a signal line to complete the data reception and decoding.
7. Any two-point transmission distance does not need to increase any circuit when it is less than 5 meters.
8. When the refresh rate is 30 frames/SEC, the cascade number of low speed mode is not less than 512 points, and high speed mode is not less than 1024 points. Data transmission speed up to 800Kbps.
- 9.The color of the light is highly consistent and cost-effective.

8.4 Wiring schematic



8.5 Code explanation

Reference to the library that drives the WS2812B RGB light, because the RGB light signal pin is connected to the development board processor pin 13, so the math pin 13 definition is connected to the RGB signal pin.

```
#include <Adafruit_NeoPixel.h>

// Which pin on the Arduino is connected to the NeoPixels?
#define PIN 13 // On Trinket or Gemma, suggest changing this to 1

// How many NeoPixels are attached to the Arduino?
#define NUMPIXELS 12 // Popular NeoPixel ring size

// When setting up the NeoPixel library, we tell it how many pixels,
// and which pin to use to send signals. Note that for older NeoPixel
// strips you might need to change the third parameter -- see the
// strandtest example for more information on possible values.
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

#define DELAYVAL 500 // Time (in milliseconds) to pause between pixels
```

Some Settings need to be initialized, such as the baud rate of the serial port is set to 9600 and the RGB light function is initialized.

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

    pixels.begin(); // INITIALIZE NeoPixel strip object (REQUIRED)

    Serial.println("Initialization completed!");
}
```

Call the clear function first, and in the for loop, turn the first light on and set the color to green.

Because we have 12 lights, so we're going through 12 cycles.

```
void loop()
{
    pixels.clear(); // Set all pixel colors to 'off'

    // The first NeoPixel in a strand is #0, second is 1, all the way up
    // to the count of pixels minus one.
    for(int i=0; i<NUMPIXELS; i++) { // For each pixel...
        ...
        // pixels.Color() takes RGB values, from 0,0,0 up to 255,255,255
        // Here we're using a moderately bright green color:
        pixels.setPixelColor(i-1, pixels.Color(0, 0, 0));
        pixels.setPixelColor(i, pixels.Color(0, 150, 0));

        pixels.show(); // Send the updated pixel colors to the hardware.

        delay(DELAYVAL); // Pause before next pass through loop
    }
}
```

Lesson 9 Gradient RGB

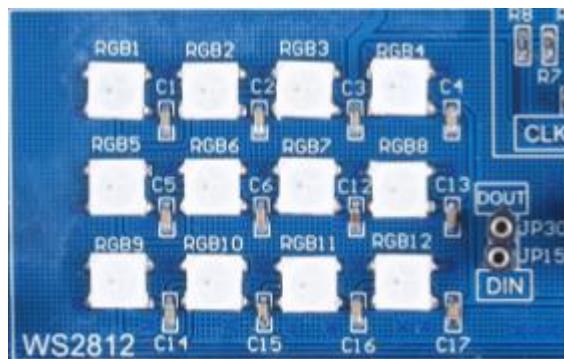
9.1 Overview

WS2812B is a built-in LED driver chip. In this course, you will learn how to control multiple LEDs, brightness adjustment, color adjustment, and other functions from one IO port.

9.2 Working principle

WS2812B is an intelligent external controlled LED light source integrating control circuit and luminous circuit.

The shape is the same as a 5050 LED lamp bead, and each element is a pixel point.



The pixel contains an intelligent digital interface data latch signal shaping amplifier drive circuit, also contains a high-precision internal oscillator and 5V voltage programmable fixed current control part, effectively ensure the pixel light color highly consistent.

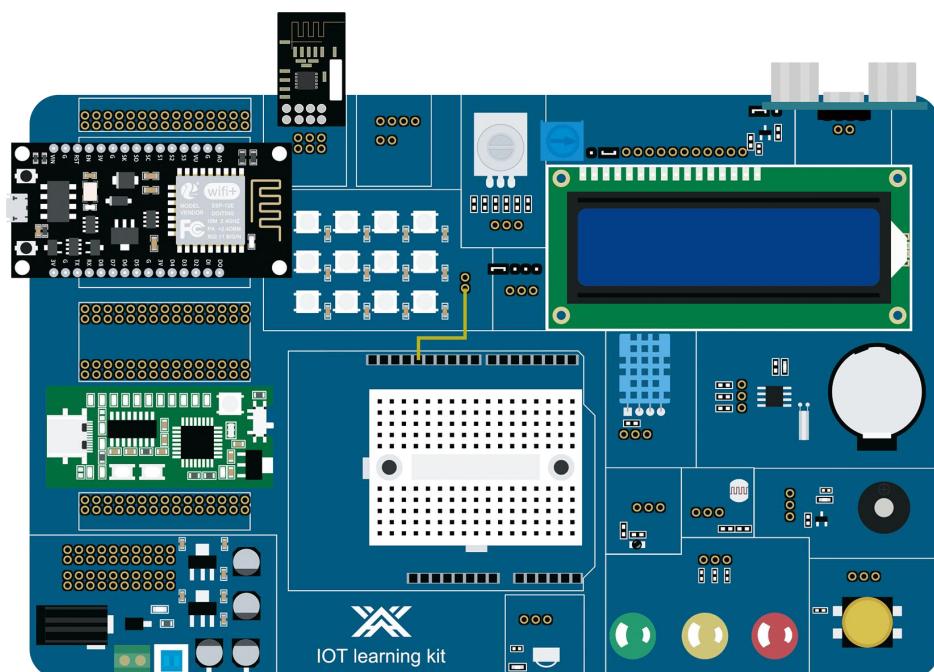
The data protocol adopts the communication mode of single-line return to zero code. After the pixel is powered on and reset, the DIN end accepts the data transmitted from the controller. The first sent 24bit data is extracted by the first pixel and sent to the data latch inside the pixel.

After internal shaping and circuit shaping, the remaining data is amplified and forwarded to the next cascaded pixel through the DO port. Every transmission of a pixel, the signal is reduced by 24bit.

Automatic shaping and forwarding technology is adopted for pixels, so that the number of cascaded pixels is not limited by signal transmission, but only limited by signal transmission speed.

RGB is integrated on the development board, and the control pin of the RGB light is connected to pin 13 of the RGB-Nano board.

9.3 Wiring schematic



9.4 Code explanation

Reference to the library that drives the WS2812B RGB light, because the RGB light signal pin is connected to the development board processor pin 13, so the math pin 13 definition is connected to the RGB signal pin. Since there are 12 lights, the number of all leds is set to 12 in the program.

```
#include <Adafruit_NeoPixel.h>
#ifndef __AVR__
    #include <avr/power.h> // Required for 16 MHz Adafruit Trinket
#endif

// Which pin on the Arduino is connected to the NeoPixels?
// On a Trinket or Gemma we suggest changing this to 1:
#define LED_PIN      13

// How many NeoPixels are attached to the Arduino?
#define LED_COUNT    12

// Declare our NeoPixel strip object:
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);
```

Do some initial setup.

```
void setup() {
    // These lines are specifically to support the Adafruit Trinket 5V 16 MHz.
    // Any other board, you can remove this part (but no harm leaving it):
#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000)
    clock_prescale_set(clock_div_1);
#endif
    // END of Trinket-specific code.

    strip.begin();           // INITIALIZE NeoPixel strip object (REQUIRED)
    strip.show();            // Turn OFF all pixels ASAP
    strip.setBrightness(50); // Set BRIGHTNESS to about 1/5 (max = 255)
}
```

Call the rainbow function with a parameter of 2 milliseconds, which represents the time delay for each gradient, and call the RGB function after the gradient ends.

```
void loop() {
    rainbow(2);           // Flowing rainbow cycle along the whole strip
    RGB(1,50);
    RGB(2,100);
    RGB(3,100);
}
```

```

void RGB(int num,int time)
{
    // The first NeoPixel in a strand is #0, second is 1, all the way up
    // to the count of pixels minus one.
    strip.clear(); // Set all pixel colors to 'off'
    for(int i=0; i<LED_COUNT; i++) { // For each pixel...

        // pixels.Color() takes RGB values, from 0,0,0 up to 255,255,255
        // Here we're using a moderately bright green color:
        strip.setPixelColor(i-1, strip.Color(0, 0, 0));
        for (int j = 0; j < 255; j++)
        {
            if(num == 1)
            {
                strip.setPixelColor(i, strip.Color(j, 0, 0));
            }
            else if(num == 2)
            {
                strip.setPixelColor(i, strip.Color(0, j, 0));
            }
            else if(num == 3)
            {
                strip.setPixelColor(i, strip.Color(0, 0, j));
            }
        }
        strip.show(); // Send the updated pixel colors to the hardware.
        delay(time); // Pause before next pass through loop
    }
}

// Rainbow cycle along whole strip. Pass delay time (in ms) between frames.
void rainbow(int wait) {
    // Hue of first pixel runs 5 complete loops through the color wheel.
    // Color wheel has a range of 65536 but it's OK if we roll over, so
    // just count from 0 to 5*65536. Adding 256 to firstPixelHue each time
    // means we'll make 5*65536/256 = 1280 passes through this outer loop:

    for(long firstPixelHue = 0; firstPixelHue < 5*65536; firstPixelHue += 256) {

        for(int i=0; i<strip.numPixels(); i++) { // For each pixel in strip...

            // Offset pixel hue by an amount to make one full revolution of the
            // color wheel (range of 65536) along the length of the strip
            // (strip.numPixels() steps):
            int pixelHue = firstPixelHue + (i * 65536L / strip.numPixels());
            // strip.ColorHSV() can take 1 or 3 arguments: a hue (0 to 65535) or
            // optionally add saturation and value (brightness) (each 0 to 255).
            // Here we're using just the single-argument hue variant. The result
            // is passed through strip.gamma32() to provide 'truer' colors
            // before assigning to each pixel:
            strip.setPixelColor(i, strip.gamma32(strip.ColorHSV(pixelHue)));

        }
        strip.show(); // Update strip with new contents
        delay(wait); // Pause for a moment
    }
}

```

Lesson 10 DS1307

10.1 Overview

In many electronic devices, operations must be run according to time.

When the main system is down, you should not stop calculating the time and date for devices such as computers, mobile phones, etc.

Therefore, the real-time clock (RTC) module is adopted.

In this section, you will learn how to use the RTC DS1307 module and the RGB-Nano development board to make a time prompter displayed by the LCD screen.

10.2 LCD1602 Introduction



Introduction to the pins of LCD1602:

VSS: A pin that connects to ground.

VDD: A pin that connects to a +5V power supply.

VO: A pin that adjust the contrast of LCD1602.

RS: A register select pin that controls where in the LCD's memory you are writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

R/W: A Read/Write pin that selects reading mode or writing mode

E: An enabling pin that, when supplied with low-level energy, causes the LCD module to execute relevant instructions.

D0-D7: Pins that read and write data.

A and K: Pins that control the LED backlight.

10.3 DS1307 Introduction

DS1307 is a low power, with 56 bytes of non-volatile RAM full BCD code clock calendar real-time clock chip, address and data transmission through a two-wire bidirectional serial bus, the chip can provide seconds, minutes, hours and other information, the days of each month can be automatically adjusted. It also has leap year compensation.

The Real Time Clock, or RTC, is a system that tracks the current Time and can be used with any device that needs to maintain an accurate Time.

You can also track the exact time without using an RTC system, but RTC has some important advantages.

Here are some of them:

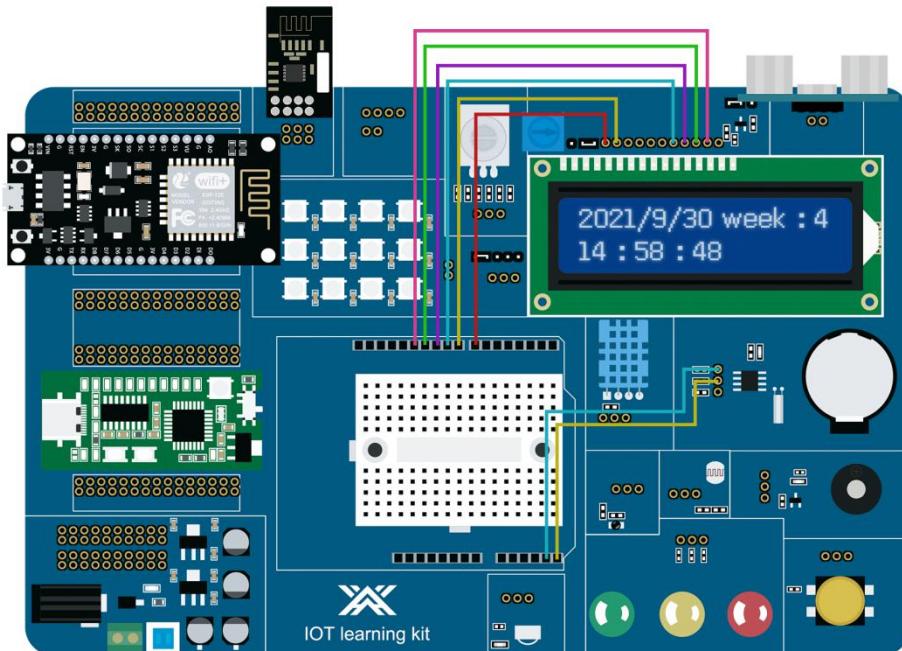
Release system time from time calculations (this feature is critical because in many cases the CPU is performing delicate tasks such as receiving sensor data).

If you do not use RTC, the CPU must also track time and it may interrupt the processor main task.

RTCS usually have a backup power supply, use CR2025 button batteries, so they can continue for a period of time when the main power supply is off or unavailable. RTC usually uses a 32.768khz crystal oscillator. But why 32,768? 32,768 is 2 to the 15th, so you can easily generate 1 second. In addition, the crystal must be small, moderate width, low power consumption, 32768 Hz can meet the requirements.

The higher the frequency, the weaker the crystal, and the lower the frequency, the greater the power consumption.

10.4 Wiring schematic



10.5 Code explanation

Declare some drive LCD display and DS1307 clock chip library, call these libraries .is our programming more simple and convenient, define the LCD display and development board connected pins, are digital port 7,8,9,10,11,12 pins.

(Note: When uploading the code, first change the existing code "DS1307_Write.ino" to correspond to the local time. After uploading the code, upload the code "DS1307_Write.ino" successfully, and then upload the code "DS1307.ino" successfully. After uploading, the normal local time will be displayed.)

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

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
#define ADDRESS_DS1307 0x68
```

Before the program starts to run, you need to set some initialization Settings, the baud rate of the serial port, the initialization of the display.

```
void setup()
{
    Wire.begin();
    Serial.begin(9600);
    lcd.begin(16,2);
    lcd.clear();
}
```

Call the function of DS1307, You need to write the date into DS1307 and then read the data.

```
void DS1307()
{
    //read the time
    Wire.beginTransmission(ADDRESS_DS1307);
    Wire.write(0x00);
    Wire.endTransmission();
    Wire.requestFrom(ADDRESS_DS1307, 7);
    if (Wire.available() >= 7)
    {
        for (int i = 0; i < 7; i++)
        {
            timeBcd[6-i] = Wire.read();
        }
    }
}
```

Print the date and time in the serial port.

```
//print
Serial.print("20"); Serial.print(timeBcd[0], HEX); Serial.print("/");
Serial.print(timeBcd[1], HEX); Serial.print("/"); Serial.print(timeBcd[2], HEX);
Serial.print(" "); Serial.print(BcdToDay(timeBcd[3])); Serial.print(" ");
Serial.print(timeBcd[4], HEX); Serial.print(":");
Serial.print(timeBcd[5], HEX); Serial.print(":");
Serial.print(timeBcd[6], HEX); Serial.println();
```

```
lcd.setCursor(0, 0);
lcd.print("20");
lcd.setCursor(2, 0);
lcd.print(timeBcd[0], HEX);
lcd.setCursor(4, 0);
lcd.print("/");
lcd.setCursor(5, 0);
lcd.print(timeBcd[1], HEX);
lcd.setCursor(7, 0);
lcd.print("/");
lcd.setCursor(8, 0);
lcd.print(timeBcd[2], HEX);
lcd.setCursor(10, 0);
lcd.print("week:");
lcd.setCursor(15, 0);
lcd.print(timeBcd[3], HEX);

lcd.setCursor(0, 1);
lcd.print(timeBcd[4], HEX);
lcd.setCursor(2, 1);
lcd.print(":");
lcd.setCursor(3, 1);
lcd.print(timeBcd[5], HEX);
lcd.setCursor(5, 1);
lcd.print(":");
lcd.setCursor(6, 1);
lcd.print(timeBcd[6], HEX);

// Convert binary coded decimal to day
String BcdToDay(byte val)
{
    String res;
    switch(val)
    {
        case 1: res = "Sunday"; break;
        case 2: res = "Monday"; break;
        case 3: res = "Tuesday"; break;
        case 4: res = "Wednesday"; break;
        case 5: res = "Thursday"; break;
        case 6: res = "Friday"; break;
        case 7: res = "Saturday"; break;
        default: res = "Error!";
    }
    return res;
}
```

Lesson 11 Show temp

11.1 Overview

In this lesson, you will learn how to use the LCD 1602 display to display temperature information.

The display is back lit by leds and can display two lines of up to 16 characters each.

You can see the rectangles of each character and the pixels that make up each character on the monitor.

The monitor is blue and white and is used to display text.

In this lesson, we will run the LCD library's RGB-Nano board temperature display routine.

11.2 Analog Temperature Sensor Introduction



A thermistor is a type of resistor whose resistance is dependent on temperature, more so than in standard resistors. The word is a portmanteau of thermal and resistor. Thermistors are widely used as inrush current limiter, temperature sensors (NTC type typically), self- resetting overcurrent protectors, and self-regulating heating elements.

Specification:

Model No: NTC-MF52 3950

3Pin

Temperature Range : ~55°C ~+125°C

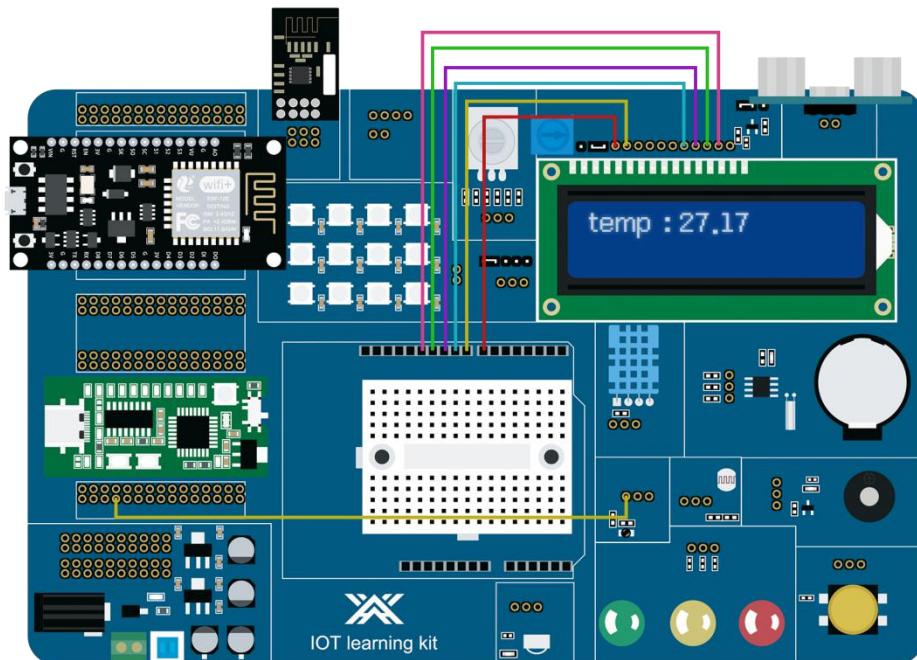
Accuracy : +/-0.5°C

Pull-up resistor : 10KΩ

PIN CONFIGURATION:

- 1、“S”: GND
- 2、“+” : +5V
- 3、“-” :Singal pin

11.3 Wiring schematic



11.4 Code explanation

Call LCD display library, define analog interface A0 to connect analog temperature sensor. Define the pins that connect the LCD display to the development board as digital ports 7, 8, 9, 10, 11, 12.

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

int NTCPin = A0;
#define SERIESRESISTOR 10000
#define NOMINAL_RESISTANCE 10000
#define NOMINAL_TEMPERATURE 25
#define BCOEFFICIENT 3950

//Do the initial setup
LiquidCrystal lcd(7, 8, 9, 10, 11, 12); //LCD pin
```

The serial port baud rate is set to 9600, and the LCD screen is initialized.

```
void setup()
{
    Serial.begin(9600);
    Wire.begin();
    lcd.begin(16,2); // Initialize LCD1602
    lcd.clear(); // Clear the LCD screen
}
```

The simulated data of the simulated temperature sensor is obtained and displayed on the serial port debugging window and LCD screen after calculation.

```
void loop()
{
    float ADCvalue;
    float Resistance;
    ADCvalue = analogRead(NTCPin);
    Resistance = (1023 / ADCvalue) - 1;
    Resistance = SERIESRESISTOR / Resistance;

    float steinhart;
    float temp1;
    steinhart = Resistance / NOMINAL_RESISTANCE; // (R/R0)
    steinhart = log(steinhart); // ln(R/R0)
    steinhart /= BCOEFFICIENT; // 1/B * ln(R/R0)
    steinhart -= 1.0 / (NOMINAL_TEMPERATURE + 273.15); // + (1/T0)
    steinhart = 1.0 / steinhart; // Invert
    steinhart += 273.15; // convert to C
    temp1 = steinhart - (steinhart * 2); // Turn negative numbers into positive numbers

    Serial.print("Temperature: ");// temperature
    Serial.print(temp1);
    Serial.println(" °C ");

    //Set LCD start display pointer position, 0 column 0 row
    lcd.setCursor(0, 0);
    lcd.print("temp:");
    lcd.setCursor(5, 0);
    lcd.print(temp1); //Display temperature data
    delay(1000);
}
```

Lesson 12 Show temp and humi

12.1 Overview

In this lesson, you will learn how to connect and use a display screen to display data measured by a temperature and humidity sensor.

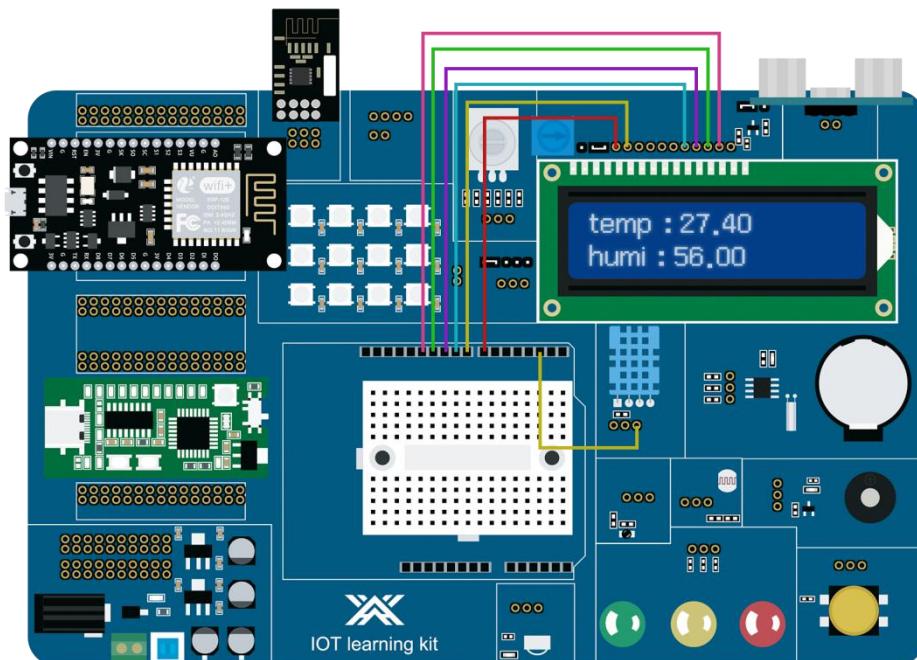
The display has LED backlighting and can display two lines of up to 16 characters each.

You can see the rectangles of each character and the pixels that make up each character on the monitor.

The display is blue and white and is used to display text.

In this lesson, we will run the LCD library's RGB-Nano sample program for measuring temperature and humidity.

12.2 Wiring schematic



12.3 Code explanation

Invoke the library of the T/H sensor (DHT11) and LCD display and define the processor number 2 to connect to the T/H sensor.define the LCD display and development board connected pins, are digital port 7,8,9,10,11,12 pins.

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

//Define the pins
#define DHTPIN 2
//Define the type, DHT11 or whatever
#define DHTTYPE DHT11
//Do the initial setup
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
```

The serial port baud rate is set to 9600, the DHT11 sensor is initialized, and the LCD screen is initialized.

```
void setup()
{
    Serial.begin(9600);
    dht.begin(); //DHT Start to work}
    Wire.begin();
    lcd.begin(16,2);
    lcd.clear();
}
```

Obtain the humidity and temperature data of the sensor and display them in real time on the serial port debugging window and LCD screen.

```
void loop()
{
    // It takes a few seconds between tests,
    //and this sensor is a little slow.
    delay(500);
    // It takes 250 milliseconds to read temperature or humidity
    float h = dht.readHumidity(); //Read the humidity
    float t = dht.readTemperature(); //Read temperature, default is Celsius
    Serial.print("Humidity: ");//humidity
    Serial.println(h);
    Serial.print("Temperature: ");// temperature
    Serial.print(t);
    Serial.println(" °C ");

    //Set LCD start display pointer position, 0 column 0 row
    lcd.setCursor(0, 0);
    lcd.print("temp:");
    lcd.setCursor(5, 0);
    lcd.print(t); //Display temperature data

    //Set LCD start display pointer position, 0 column 1 row
    lcd.setCursor(0, 1);
    lcd.print("humi:");
    lcd.setCursor(5, 1);
    lcd.print(h);
}
```

Lesson 13 Ultrasonic module

13.1 Overview

In this lesson, you will learn how to connect and use ultrasonic ranging sensors and LCD 1602 displays.

The display has LED backlighting and can display two lines of up to 16 characters each.

You can see the rectangles of each character and the pixels that make up each character on the monitor.

The display is blue and white and is used to display text.

In this lesson, we will run the RGB-Nano sample program to display the ultrasonic sensor measurement distance information on the LCD 1602 display.

13.2 Ultrasonic sensor Introduction

Ultrasonic sensor module HC-SR04 provides 2cm-400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

Using IO trigger for at least 10us high level signal,

The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.

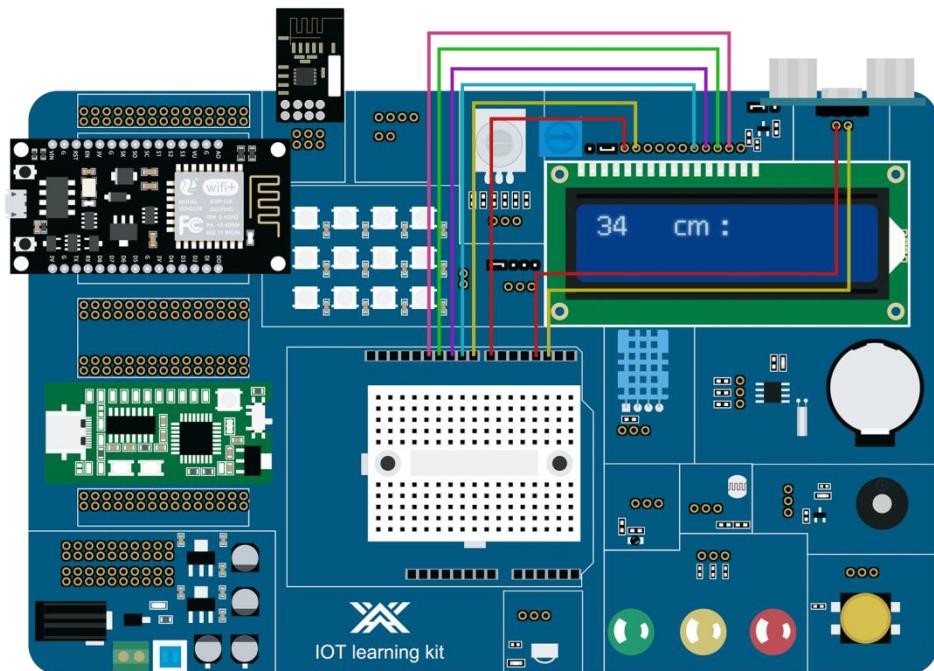
IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to turning.

Test distance = (high level time × velocity of sound (340m/s) /2.

You only need to supply a short 10us pulse to 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. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: us / 58 = centimeters or us

/ 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.

13.3 Wiring schematic



The data measured by the ultrasonic sensor module HC-SR04 is processed by the RGB-Nano board, and the data is displayed on the LCD screen.

13.4 Code explanation

Declaration drives LCD screen library, ultrasonic sensor module HC-SR04 interface.

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

//echo:D3 trig:D2
#define Trig 2
#define Echo 3

LiquidCrystal lcd(7, 8, 9, 10, 11, 12); //LCD pin
```

Initialize the serial port and screen, and set the ultrasonic sensor module HC-SR04 interface .

```
void setup()
{
    Serial.begin(9600);
    Wire.begin();
    lcd.begin(16,2); // Initialize LCD1602
    lcd.clear(); // Clear the LCD screen
    pinMode(Trig,OUTPUT); //Set Trig pin to output
    pinMode(Echo,INPUT); //Set the Echo pin as input
}
```

Obtain the data measured by the ultrasonic sensor module HC-SR04 and display it in the serial debugging window and LCD screen.

```
void loop()
{
    int dis = GetDistance(); //Assign ultrasonic data to dis
    Serial.print(dis);
    Serial.print("cm");
    Serial.println();
    //Set LCD start display pointer position, 0 column 0 row
    lcd.setCursor(0, 0);
    lcd.print(dis); //Display range data
    if(dis<10){
        lcd.setCursor(1, 0);
        lcd.print(" ");
    }else if( (dis>9) && (dis<100) ){
        lcd.setCursor(2, 0);
        lcd.print(" ");
    }
    lcd.setCursor(5, 0);
    lcd.print("cm:");
    delay(200);
}

float GetDistance() //Read ultrasonic sensor data
{
    float distance;
    digitalWrite(Trig, LOW);
    delayMicroseconds(2);
    digitalWrite(Trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(Trig, LOW);
    distance = pulseIn(Echo, HIGH) / 58.00;
    return distance;
}
```

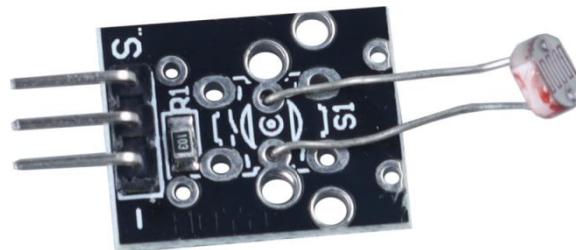
Lesson 14 Photosensitive resistance

14.1 Overview

Photoinductive resistance, is the use of semiconductor photoelectric effect made of a resistance value with the intensity of the incident light and change the resistor;

When the incident light is strong, the resistance goes down, when the incident light is weak, the resistance goes up.

Photosensitive resistors are commonly used for light measurement, light control and photoelectric conversion (to convert light changes into electrical changes).



14.2 Component Introduction

Photosensitive resistors can be widely used in a variety of light control circuits, such as light control, regulation and other occasions, can also be used for light control switches.

In this experiment, we first carried out a relatively simple use experiment of photosensitive resistance.

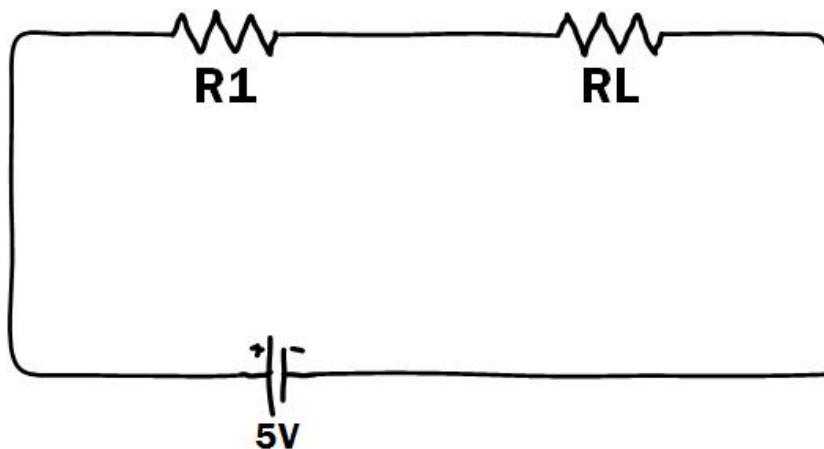
Since photosensitive resistor is a component that can change the resistance value according to the light intensity, it also needs the analog port to read the analog value naturally. In this experiment, we can learn from the PWM interface experiment and change the potentiometer into a photosensitive resistor to realize that the brightness of the LED small lamp will also change accordingly when the light intensity is different.

The resistance of photosensitive resistors is very high in dark and dark conditions.

The stronger the light, the smaller the resistance.

By measuring the voltage change on both sides of the photosensitive resistor, the change of the photosensitive resistance value can be known and the illumination intensity value can be obtained.

In the connection diagram, we can find a partial voltage resistor in series for the photosensitive resistor.



In the figure above, RL is a photosensitive resistor, and R1 is a series resistor. In the dark, RL is going to be very, very large, so Vout is going to be very large, close to 5V.

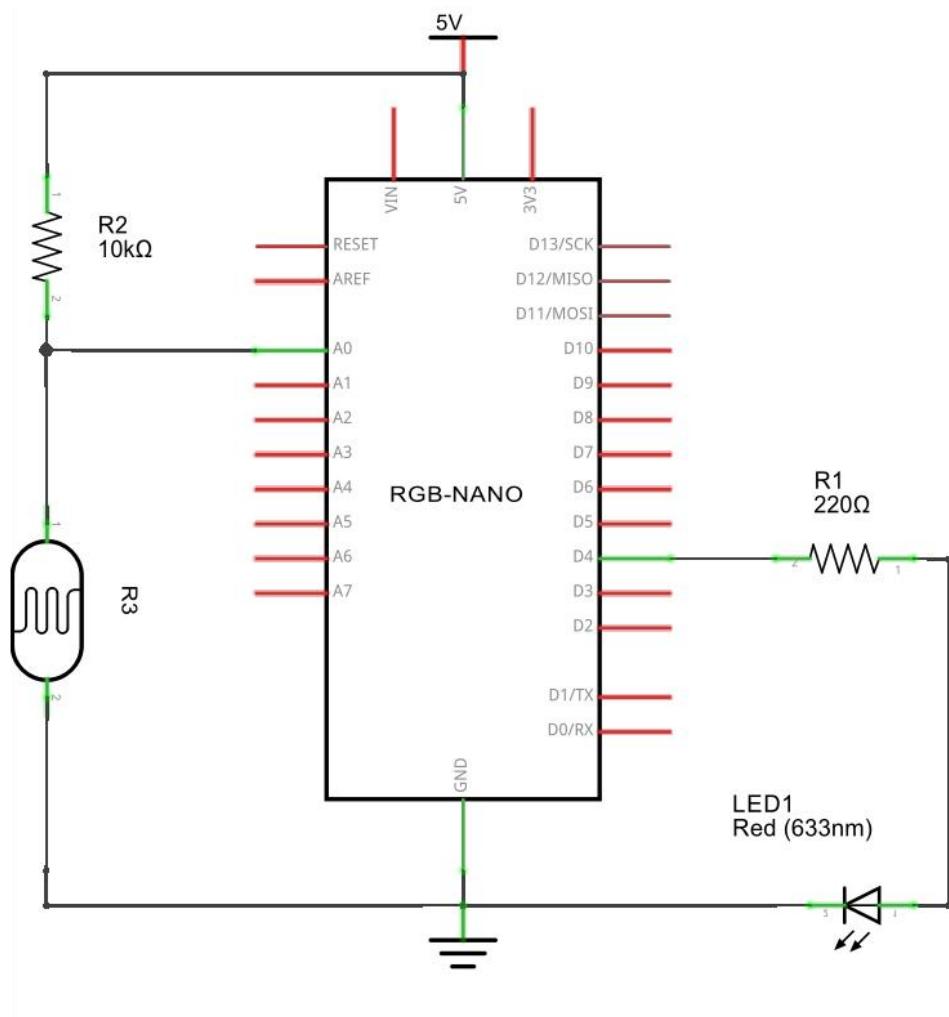
The formula is as follows:

$$V_{out} = \frac{RL}{R1 + RL} * V_{in}$$

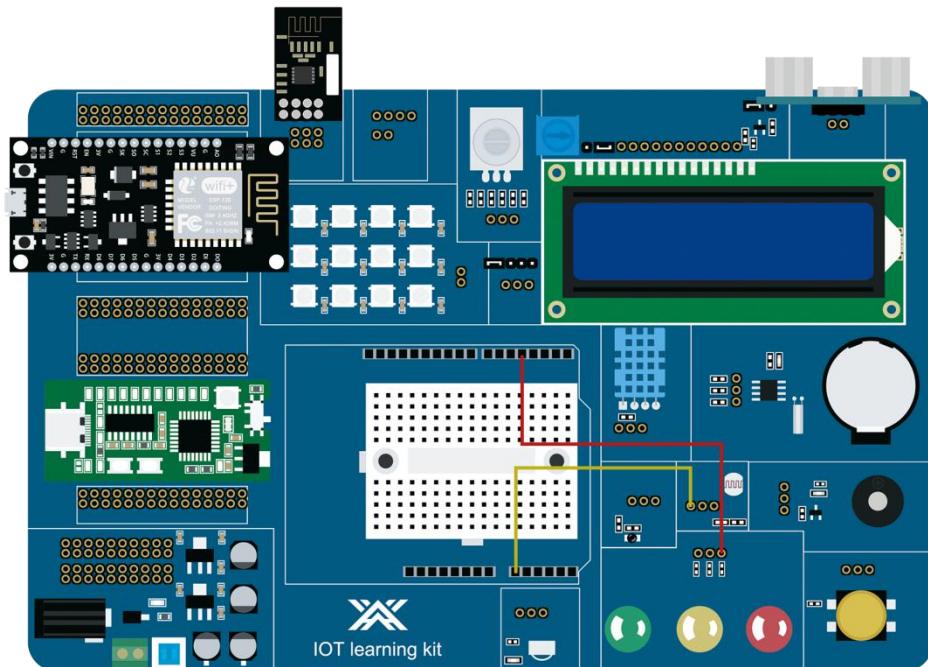
As soon as the light hits, the value of RL decreases rapidly, so Vout decreases with it.

It can be seen from the above formula that the selection of R1 should not be too small, preferably around 1K ~ 10K, otherwise the ratio will not change significantly.

14.3 Connection Diagram



14.4 Wiring schematic



14.5 Code explanation

Define the photoresistor to connect to analog pin A0 and the LED to connect to digital pin 4.

```
int inputValue = 0;  
  
//Define the pins  
#define RES_Pin A0 //A0 Connect the photoresistor  
#define LED_Pin 4 //D4 Connect the LED pins
```

When the photoresistor detects that the light is dim, it puts the LED light is on, so set the pin connecting the LED light to output mode.

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

Use the analog reading function to read the data of the photosensitive resistor, and then take this data as the judgment condition of day and night. If the detected data is less than 500, it indicates that the current environment has become dark, so let the pin connected to the LED output high level, and make the LED light up.

```
void loop()
{
    //Read the photosensitive resistance value
    inputValue = analogRead(RES_Pin);

    Serial.print("Value=");
    Serial.println(inputValue);

    //When the light is low, turn on the light
    if (inputValue < 500)
    {
        digitalWrite(LED_Pin, HIGH);
    }
    else //When the light is bright, turn off the light
    {
        digitalWrite(LED_Pin, LOW);
    }
}
```

Lesson 15 Rotary encoder control RGB

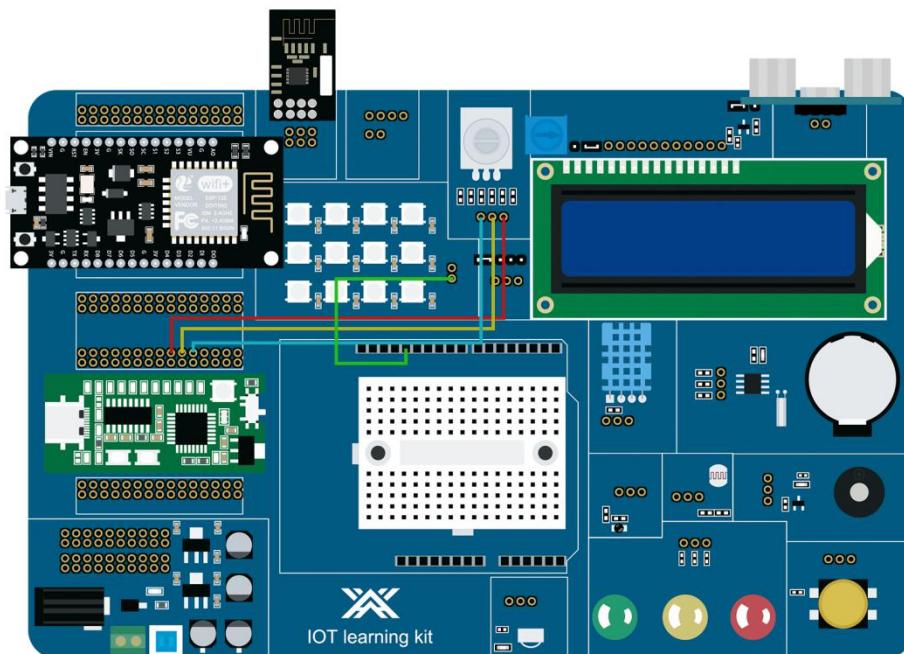
15.1 Overview

Through this project, you can learn to use a rotary encoder to control the RGB display running light on the micro controller RGB Nano.

Connection description:

Connect the D13 pin of the single-chip microcomputer to the DIN interface of the JP15 row with a DuPont cable, and the connection is successful when it is plugged in. Next, connect the D2 pin of the micro controller to the "CLK" of the JP13 row seat with a Dupont line, connect the D3 pin to the "DT" of the JP3 row seat, and connect the D4 pin to the "SW" of the JP13 row seat. You can connect successfully.

15.2 Project wiring diagram



15.3 Code explanation

Contains RGB library files, defines the number of RGB lights and pin numbers, and defines the function pins and the global variables required by the function.

```
#include <Adafruit_NeoPixel.h>
#define NUMPIXELS 12 // Number of 2812 lamps
#define RGB_PIN 13 // 2812 pin definition
//Define the pin connection
int CLK = 2;//CLK->D2
int DT = 3;//DT->D3
int SW = 4;//SW->D4
const int interrupt0 = 0;//Interrupt 0 on pin 2
int count = 0;//Define the count
int lastCLK = 0;//CLK initial value

Adafruit_NeoPixel pixels(NUMPIXELS, RGB_PIN, NEO_GRB + NEO_KHZ800); //Creating light objects
```

At the same time, the function is initialized, the work type of the pin is determined, and the interrupt 1 function is enabled at the same time.

```
void setup()
{
    pinMode(SW, INPUT_PULLUP);
    pinMode(CLK, INPUT_PULLUP);
    pinMode(DT, INPUT_PULLUP);

    pixels.begin(); // Initialize 2812 library functions
    pixels.show();
    pixels.clear(); // Lighting function
    attachInterrupt(interrupt0, ClockChanged, CHANGE); //Set the interrupt 0 handler,
    Serial.begin(9600);
}
```

The main function, the value sent by the encoder through the display function to make the corresponding RGB light be lit. By changing the "count" you can change the light that is lit

```
void loop()
{
    if(count<=0) count=0;
    if(count>=12) count=12;
    if (!digitalRead(SW)) //Read the button press and the count value to 0 when the counter reset
    {
        count = 0;
    }
    //Light up the corresponding RGB light
    //Change the "count" variable to change the corresponding light
    pixels.setPixelColor(count, pixels.Color(0, 255, 0));
    pixels.show();
    pixels.clear(); // Lighting function
    attachInterrupt(interrupt0, ClockChanged, CHANGE);
}
```

The encoder processing function will determine whether the encoder is rotating forward or backward, and at the same time change the value of "count".

```
void ClockChanged()
{
    if(digitalRead(SW)) //Judgment is not pressed
    {
        int clkValue = digitalRead(CLK); //Read the CLK pin level
        int dtValue = digitalRead(DT); //Read the DT pin level
        if (lastCLK != clkValue)
        {
            lastCLK = clkValue;
            count += (clkValue != dtValue ? 1 : -1); //CLK and inconsistent DT + 1, otherwise - 1
            Serial.print("count:");
            Serial.println(count); //Serial print rotation value
        }
    }
}
```

Lesson 16 NRF24L01 launch

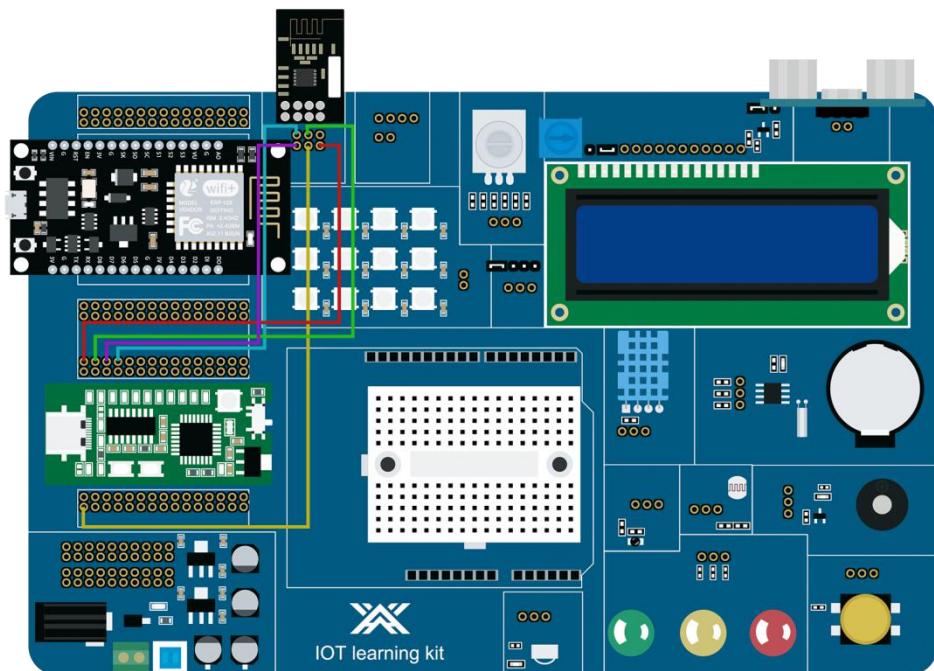
16.1 Overview

Through this project, you will learn to use NRF24L01 to send data and display it using the serial port of the IDE compiler.

Connection description:

Connect the D9, D10, D11, D12, and D13 pins of the MCU to the "CSN", "CE" "MOSI" "MISO" "SCK" on the header below JP3 with DuPont cables, and connect the NRF24L01 module to the JP3 row at the same time Mother, the connection is complete, the connection is successful.

16.2 Project wiring diagram



16.3 Code explanation

Contains the library files that the project must use.

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
```

The function pins of the program are defined in the global variable definition

```
//Initial RF24(cePin, csnPi)
RF24 radio(9,10);

//This is the transmission channel code we are about to establish
///To be consistent with another module
const uint64_t pipe = 0xE8E8F0F0E1LL;

//Data to be transferred
int data = 0;
```

Initialize the function to determine that the working baud rate is "57600". This needs to be the same as the baud rate of the serial port, otherwise garbled characters will be generated.

```
void setup(void) {
    Serial.begin(57600);
    //Boot chip
    radio.begin();
    //Open write channel
    radio.openWritingPipe(pipe);
}
```

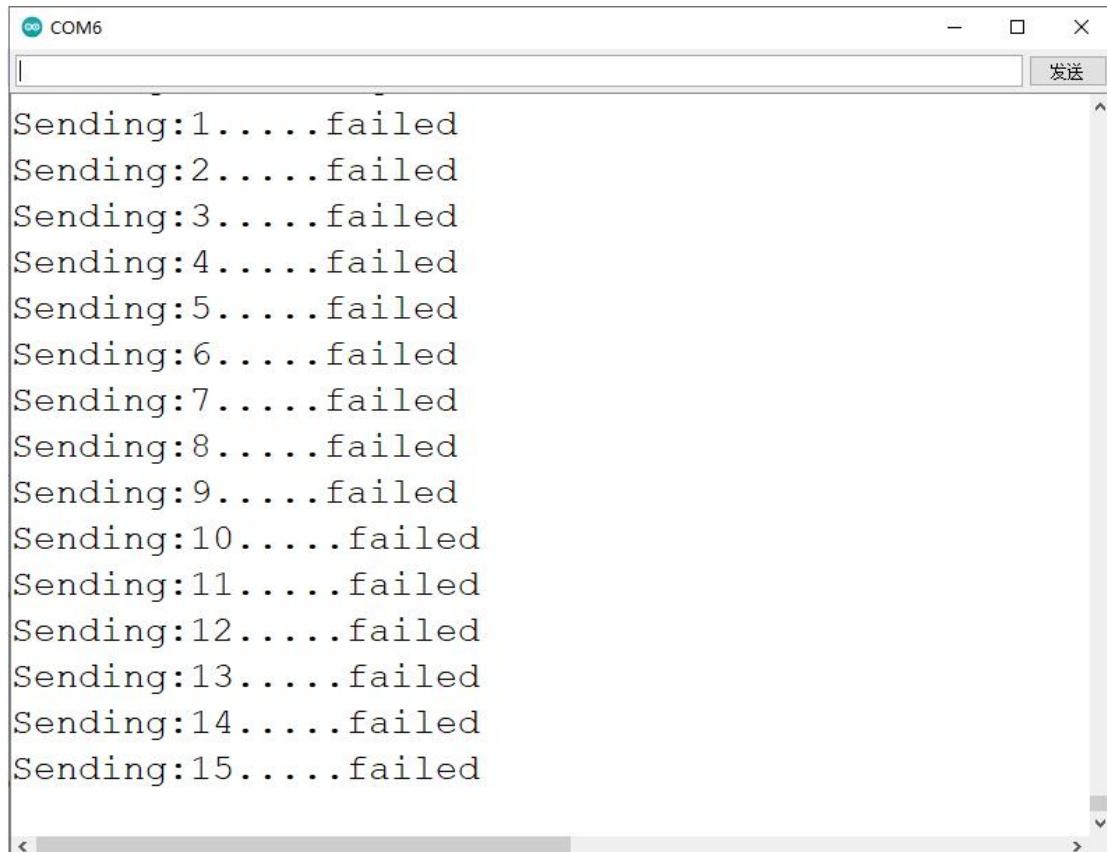
The serial port continuously prints the sent data "data" to download the program. After connecting the DuPont cable, you can open the debugging window of the IDE to view it. Pay special attention to setting the baud rate. The default baud rate is 9600 and needs to be changed to 57600.

```
void loop(void)
{
    Serial.print("Sending:");
    Serial.print(data);

    bool ok = radio.write(&data,sizeof(int));
    if(ok)
        Serial.println(".....successed");
    else
        Serial.println(".....failed");

    data++;    //Add 1 every 200ms
    delay(200);
}
```

Experimental phenomena.



The screenshot shows a terminal window titled "COM6". The window contains a list of 15 lines of text, each starting with "Sending:" followed by a number from 1 to 15 and ending with ". failed". The window has a standard title bar with minimize, maximize, and close buttons. A "发送" (Send) button is located in the top right corner. The text area is scrollable, indicated by a vertical scrollbar on the right side.

```
Sending:1.....failed
Sending:2.....failed
Sending:3.....failed
Sending:4.....failed
Sending:5.....failed
Sending:6.....failed
Sending:7.....failed
Sending:8.....failed
Sending:9.....failed
Sending:10.....failed
Sending:11.....failed
Sending:12.....failed
Sending:13.....failed
Sending:14.....failed
Sending:15.....failed
```

Lesson 17 Infrared control LED

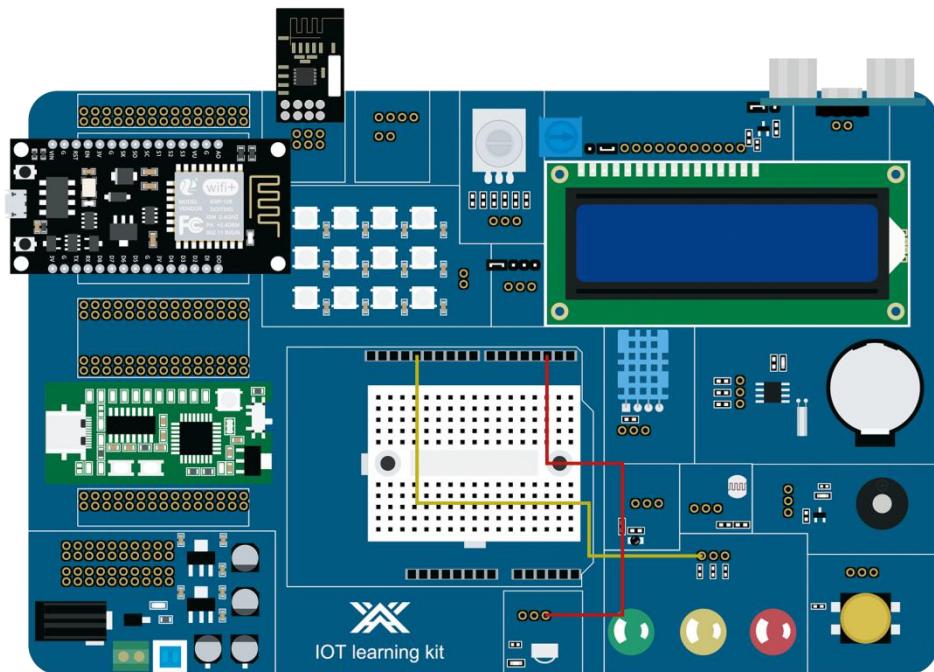
17.1 Overview

Through this project, you will learn to use an infrared remote control to remotely turn on the LED. After downloading the program, connect the DuPont cable and press the remote control's play button to turn the LED on and off.

Connection description:

Connect D2 of the single-chip microcomputer to pin "S" of JP16 with a DuPont cable, and connect D13 to any pin of JP12, and the connection is successful.

17.2 Project wiring diagram



17.3 Code explanation

The initialization of the function and the function pin definition of the function have been discussed in the previous lessons, but they are all the same, so I won't describe them here. I will mainly talk about the infrared processing function. When the infrared receiving function receives the key value, it will judge. This key value can be printed through the serial port and viewed in the serial debugger. When the key value we specified is judged, we define the state of the LED to be inverted, so that the LED can be turned on and off.

```
#include "IRremote.h"
#define HW 2
#define LED 13
int BIT_LED=0;
IRrecv irrecv(HW);
decode_results results;
void setup() {
    // put your setup code here, to run once:
    irrecv.enableIRIn();
    Serial.begin(9600);
    pinMode(LED,OUTPUT);

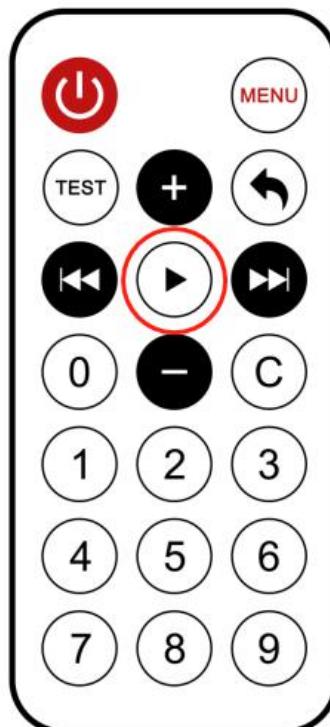
}

void loop() {
    // put your main code here, to run repeatedly:
    if (irrecv.decode(&results)) // have we received an IR signal?
    {
        translateIR();
        Serial.println( results.value,HEX );
        irrecv.resume(); // receive the next value
    }

}
```

```
void translateIR() // takes action based on IR code received
{
    if(results.value==0xFFA857) //Determine the received key value as an
    {
        if(BIT_LED)BIT_LED=0;
        else BIT_LED=1;
        if(BIT_LED)digitalWrite(LED, HIGH); //Invert the state of the LED
        else digitalWrite(LED, LOW);
        //delay(50);
    }
}
```

The button in the red box of the remote control has been set with the specified function in the program.



Lesson 18 Infrared control RGB

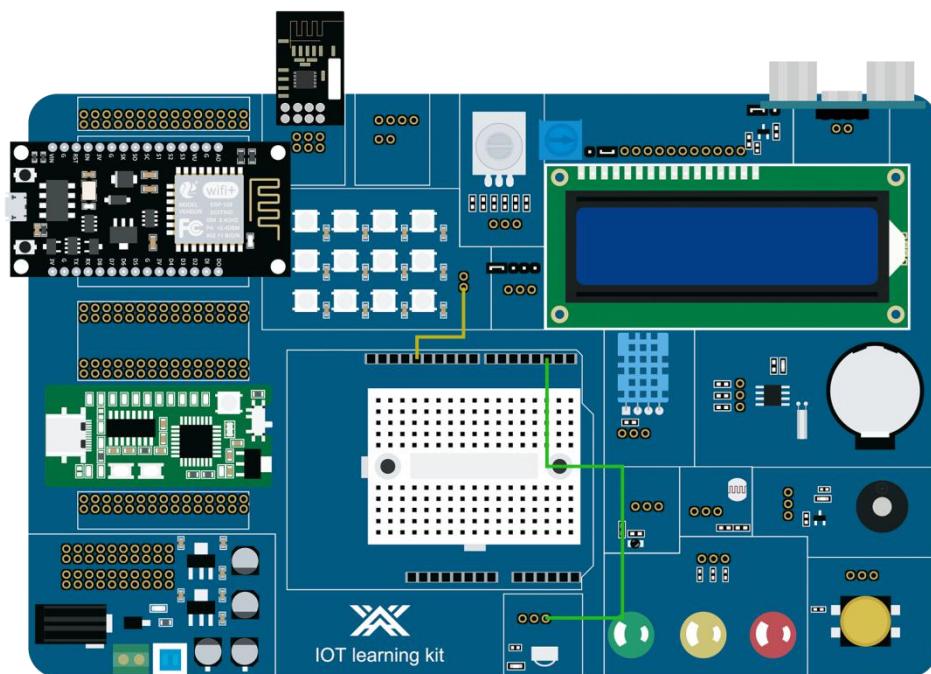
18.1 Overview

Through this project, you will learn to use an infrared remote control to turn on RGB remotely. After downloading the program, connect the DuPont cable and press 1, 2, 3, 4, 5, and 6 on the remote control to display five colors of red, green, blue, yellow, white, and the last one is to turn off RGB.

Connection description:

Connect D2 of the single-chip microcomputer to the pin "S" of JP16 with a Dupont wire, and connect D13 to the "DIN" pin of JP15, and the connection is successful.

18.2 Project wiring diagram



18.3 Code explanation

The initialization of the function and the function pin definition of the function have been discussed in the previous course, but they are the same, so I won't repeat

them here. I mainly talk about infrared processing functions. When the infrared receiving function receives the key value, it will judge. This key value can be printed out through the serial port and viewed in the serial debugger. When the key value we specify is judged, we define the RGB color corresponding to each number. When the corresponding number is pressed, the corresponding color will be lit. These key values can also be customized or Use your own defined number to control the color of RGB or the number of RGB.

```
#include "IRremote.h"
#include <Adafruit_NeoPixel.h>
#define NUMPIXELS 12 // Number of 2812 lamps
#define RGB_PIN 13 // 2812 pin definition
#define HW 2
IRrecv irrecv(HW);
decode_results results;

Adafruit_NeoPixel pixels(NUMPIXELS, RGB_PIN, NEO_GRB + NEO_KHZ800);
void setup() {
    // put your setup code here, to run once:
    irrecv.enableIRIn();
    Serial.begin(9600);
    pixels.begin(); // Initialize 2812 library functions
    pixels.show();
    pixels.clear(); // Lighting function
}

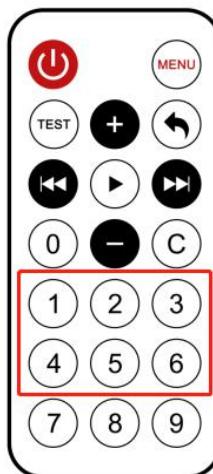
void loop() {
    // put your main code here, to run repeatedly:
    if (irrecv.decode(&results)) // have we received an IR signal?
    {
        translateIR();
        Serial.println(results.value,HEX);
        irrecv.resume(); // receive the next value
    }
}
```

```

void translateIR()
{
    if(results.value==0xFF30CF) //1
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(255,0, 0)); //Received digital 1RGB display red
            pixels.show();
        }
    }
    else if(results.value==0xFF18E7) //2
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(0,255, 0)); //Received digital 2RGB display green
            pixels.show();
        }
    }
    else if(results.value==0xFF7A85) //3
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(0,0, 255)); //Received digital 3RGB display blue
            pixels.show();
        }
    }
    else if(results.value==0xFF10EF) //4
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(255,255, 0)); //Received digital 4RGB display yellow
            pixels.show();
        }
    }
    else if(results.value==0xFF38C7) //5
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(255,255, 255)); //Received digital 5RGB display white
            pixels.show();
        }
    }
    else if(results.value==0xFFFFAA5) //6
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(0,0, 0)); //Receive digital 6RGB light off
            pixels.show();
        }
    }
}

```

The button in the red box of the remote control has been set with the specified function in the program.



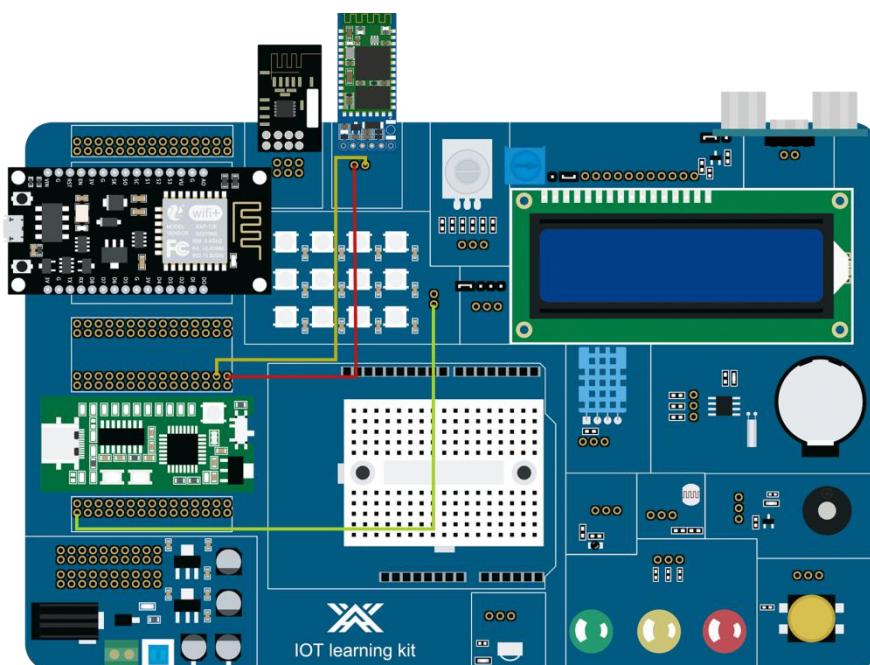
Lesson 19 Bluetooth control RGB

19.1 Overview

Through this project, you will learn to use the mobile phone APP to use Bluetooth to connect with the Bluetooth module on the board to realize remote opening of RGB. After downloading the program, connect to the DuPont cable, connect to Bluetooth, use the mobile APP to connect to Bluetooth, send R, G, B, Y, W, O through the APP keyboard, and display five colors of red, green, blue, yellow, and white. The last one One is to turn off RGB. Note that the characters here are all capitalized in English.

19.2 Connection description

Connect D13 of the single-chip microcomputer to the "DIN" pin of JP15 with a Dupont line, connect TX on the single-chip microcomputer to RX on JP32, and connect RX on the single-chip microcomputer to TX on JP32. After all connections are made, the connection is successful. Project wiring diagram:



19.3 Code explanation

The initialization of the function and the function pin definition of the function have been discussed in the previous course, but they are all the same, so I won't repeat them here. I mainly talk about Bluetooth processing functions. When the Bluetooth receiver function receives the corresponding character, it will make a judgment. This character can be printed out through the serial port and viewed in the serial debugger. When the character value we specify is judged, we define the RGB color corresponding to each number. When the corresponding number is pressed, the corresponding color will light up. These keys can also be customized or use self-defined numbers to control the color of RGB or the number of RGB.

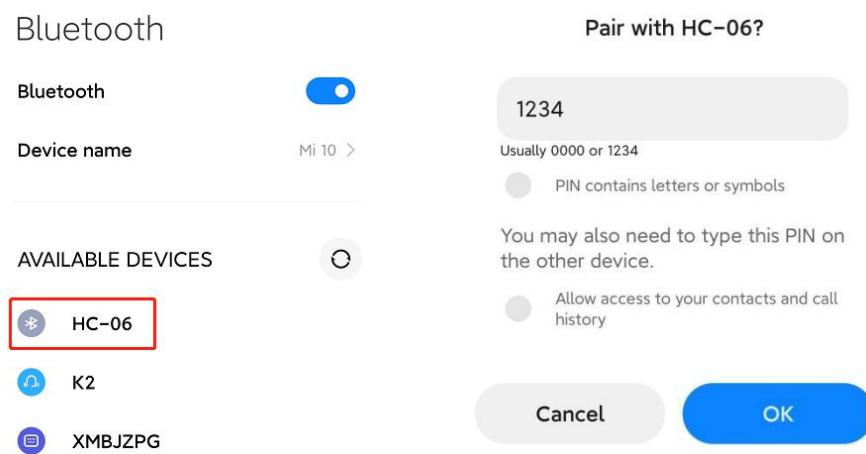
```
#include <Adafruit_NeoPixel.h>
#define NUMPIXELS 12 // Number of 2812 lamps
#define RGB_PIN 13 // 2812 pin definition
Adafruit_NeoPixel pixels(NUMPIXELS, RGB_PIN, NEO_GRB + NEO_KHZ800); //Creating light objects
void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);
    pixels.begin(); // Initialize 2812 library functions
    pixels.show();
    pixels.clear(); // Lighting function
}

void loop() {
    // put your main code here, to run repeatedly:
    HC_06(); //Run Bluetooth processing function
}
```

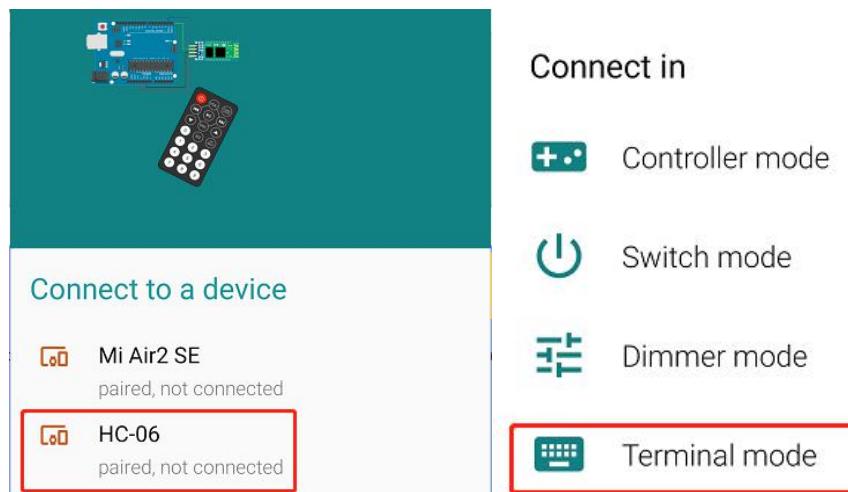
```
void HC_06()
{
    int serialData;
    if(Serial.available() > 0)      //Determine whether the received data is greater than 0
    {
        //Serial.println(Serial.read());
        serialData = Serial.read(); //Receive order function
        if('R' == serialData )  //R
        {
            Serial.println("ok");
            for(int i=0;i<12;i++)
            {
                pixels.setPixelColor(i, pixels.Color(255,0, 0)); //The bluetooth receives the "R" character RGB from the mobile phone APP and displays red
                pixels.show();
            }
        }
        else if('G' == serialData )  //G
        {
            for(int i=0;i<12;i++)
            {
                pixels.setPixelColor(i, pixels.Color(0,255, 0)); //The bluetooth receives the "G" character RGB from the mobile phone APP and displays green
                pixels.show();
            }
        }
        else if('B' == serialData )  //B
        {
            for(int i=0;i<12;i++)
            {
                pixels.setPixelColor(i, pixels.Color(0,0, 255)); //Bluetooth receives the "B" character RGB from the mobile phone APP and displays blue
                pixels.show();
            }
        }
    }
    else if('Y' == serialData )  //Y
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(255,255, 0)); //The bluetooth receives the "Y" character RGB from the mobile phone APP and displays yellow
            pixels.show();
        }
    }
    else if('W' == serialData )  //W
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(255,255, 255)); //Bluetooth receives the "W" character RGB from the mobile phone APP and displays white
            pixels.show();
        }
    }
    else if('O' == serialData )  //O
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(0,0, 0)); //Bluetooth receives the "O" character RGB from the mobile phone APP is turned off
            pixels.show();
        }
    }
}
```

19.4 Bluetooth remote control

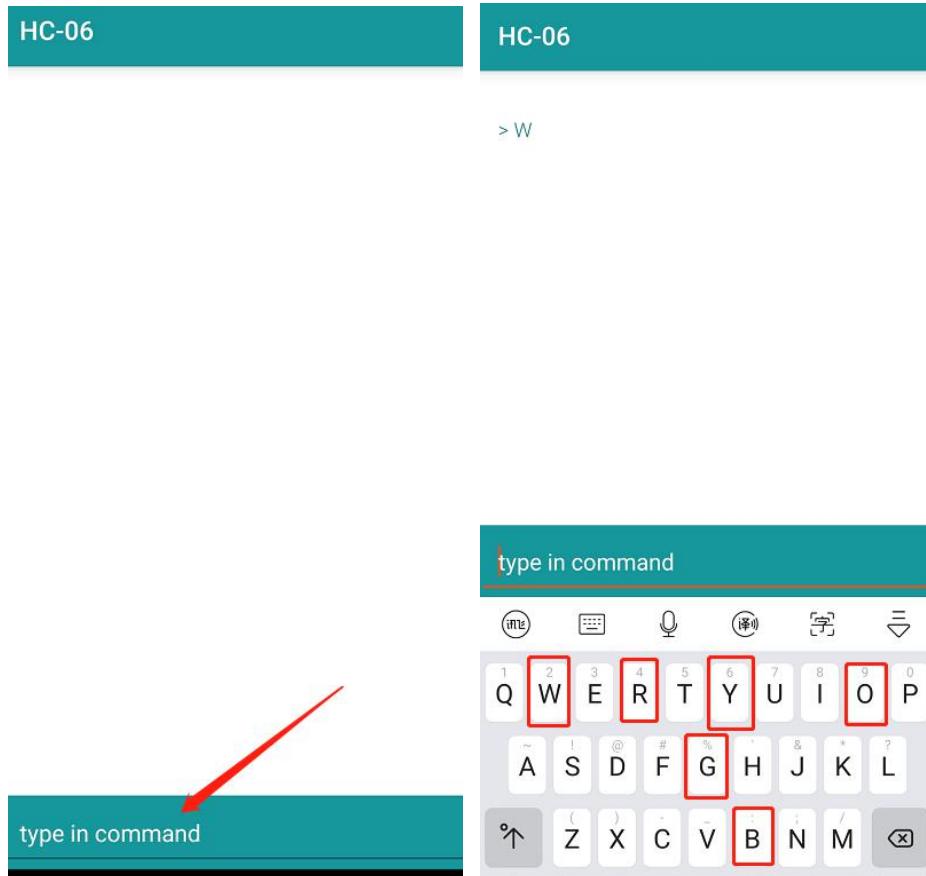
Equipped with bluetooth serial port assistant for remote control of car;First, turn on bluetooth and search for Bluetooth devices to find hc-06 connection. The default connection pairing password is 1234.



Open bluetooth Assistant and find the bluetooth module name hc-06 just configured. After successful connection, a remote control main interface will appear.



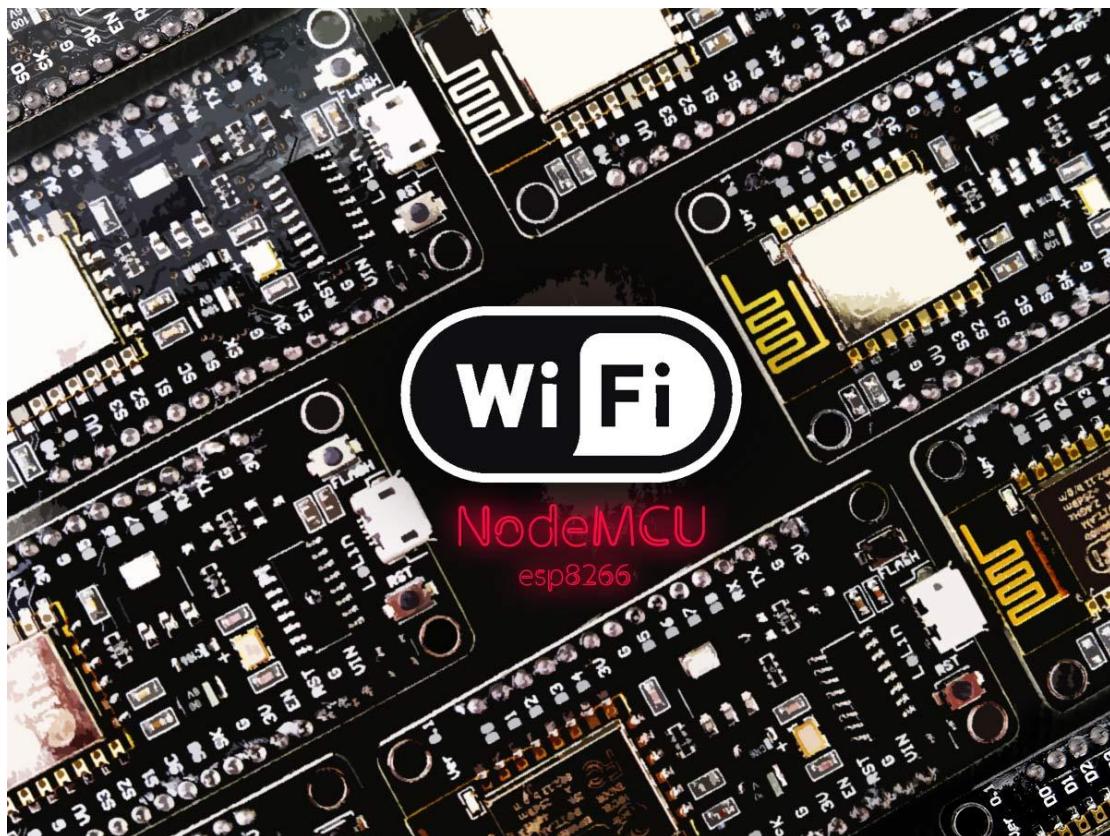
Select the keyboard mode, after entering, you can enter the characters we specified. After entering "R", the system will return an "OK" to indicate that our communication is normal, and you can enter other characters to display different colors.



Lesson 20 ESP8266 Development board

20.1 introduction:

The ESP8266 is a Wi-Fi module ideal for Internet of Things and home automation projects. This article is a beginner's guide to the ESP8266 development board.

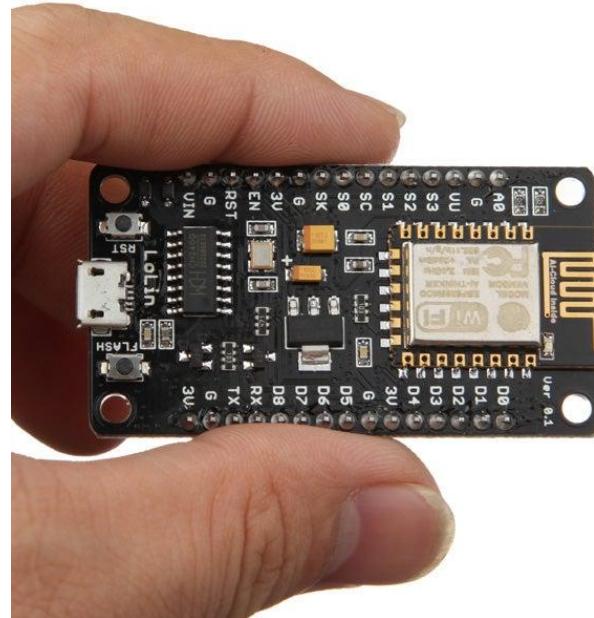


20.2 ESP8266 specifications

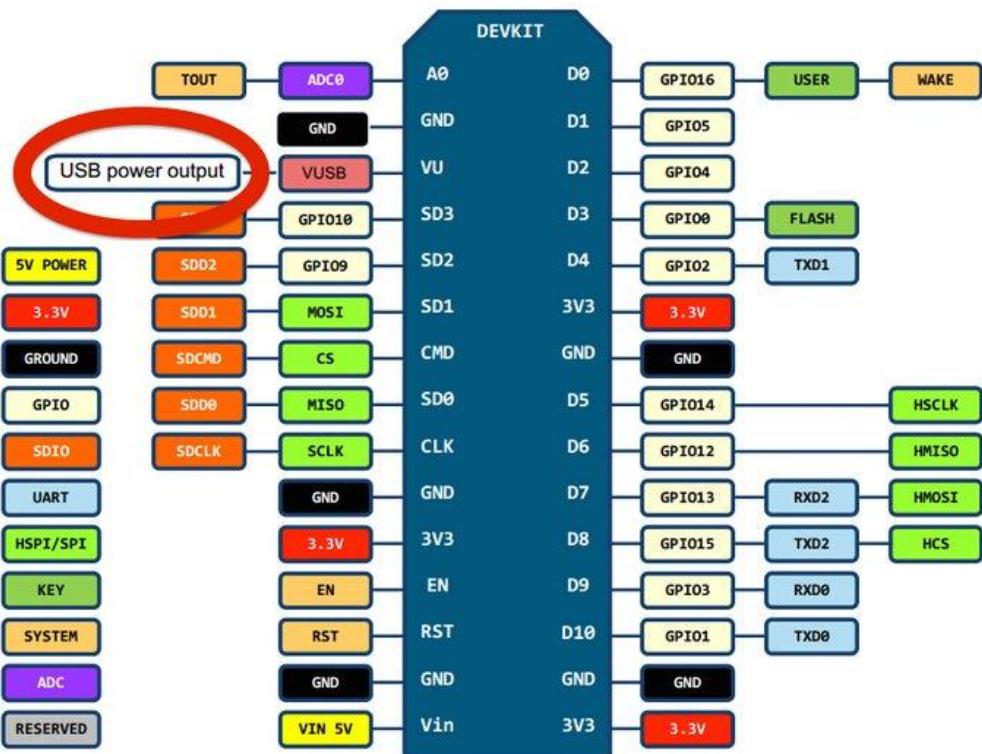
- 11 b/g/n agreement
- Wi-Fi Direct (P2P)、soft AP
- Integrates TCP/IP protocol stack
- Built-in low-power 32-bit CPU
- SDIO 2.0、SPI、UART

20.3 ESP8266 version

The ESP8266 is available in several versions (see figure below). In our opinion, the ESP-12E or more commonly known as THE ESP-12E NodeMCU suite is the most practical version available today.



ESP8266 Development board pin schematic diagram:



20.4 NodeMCU pin arrangement peripherals

NodeMCU peripherals include

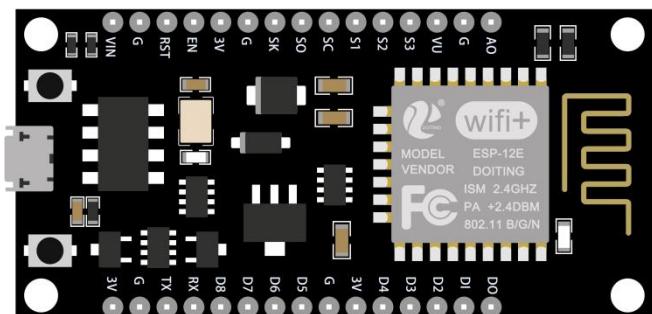
17 universal I/O pins

SPI

I2C

A serial port

10 bit ADC



20.5 What pins are used in NODEMCU ESP8266?

The GPIO number does not match the label on the pin diagram. For example, D1 corresponds to GPIO5, D2 corresponds to GPIO5

Can be used as input/output pins without problem

GPIO5 labeled D1 is commonly used as SCL (I2C)

GPIO4 labeled D2 is commonly used as SDA (I2C)

GPIO0 flagged as D3 connects to the FLASH button if pulled low startup fails

GPIO2 marked AS D4 connected to the onboard LED, if pulled low - high at startup, startup fails

GPIO14 is labeled D5 SPI (SCLK)

GPIO12 is labeled D6 SPI (MISO)

GPIO13 is labeled D7 SPI (MOSI)

ADO is marked as AO

The following are the pins that can be used, but you need to be careful because they can have unexpected behavior primarily at startup.

- **GPIO16 is marked D0 HIGH at startup to wake up from deep sleep**
- **GPIO15 is marked D8 pull to GND: if we pull high, startup fails**
- **GPIO3 在 boo 标记为 RX HIGH**

GPIO1 is marked as TX debug output at startup, if pulled down startup fails

Note:

It is recommended that pins labeled RX and ADO be used as outputs and not TX pins as inputs.

Pins called GPIO6 through GPIO11 connect to the flash chip in the ESP8266.

Therefore, it is not recommended to use these pins for input/output functions.

If you want to operate relays, GPIO4 and GPIO5 are the safest GPIO pins to use.

Lesson 21 Installing the ESP8266 development board in the Arduino IDE

The ESP8266 community has created an add-on for the Arduino IDE that allows you to program ESP8266 using the Arduino IDE and its programming language.

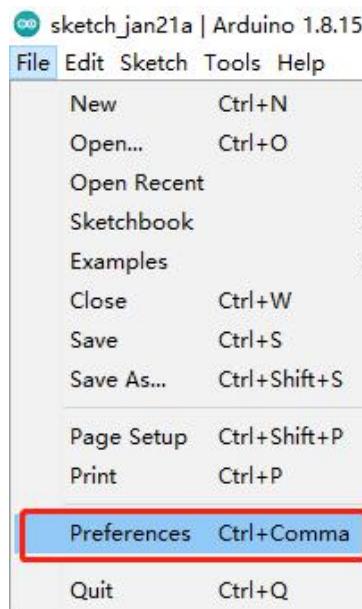
This tutorial shows how to install the ESP8266 development board in the Arduino IDE, whether you are using Windows, Mac OS X, or Linux.

Before starting this installation process, make sure you have the latest version of the Arduino IDE installed on your computer. If not, uninstall it and reinstall it. Otherwise, it might not work.

You can click on the link:<https://www.arduino.cc/en/software> Install the latest Arduino IDE software.

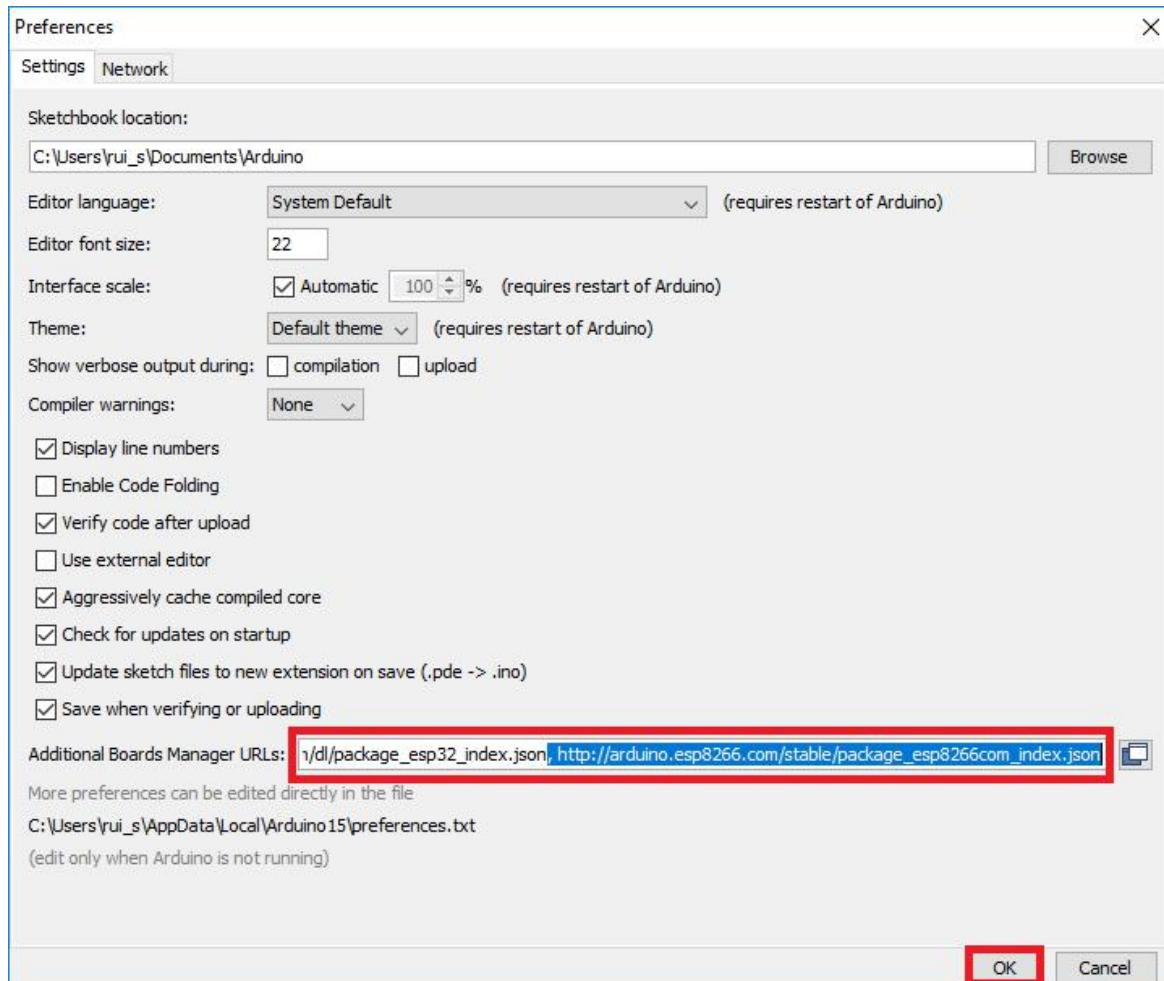
21.1 Install the ESP8266 plug-in in the Arduino IDE

To install the ESP8266 board in your Arduino IDE, follow the instructions below:
In your Arduino IDE, go to file > Preferences.



Enter it in the Additional Boards Manager URLs field

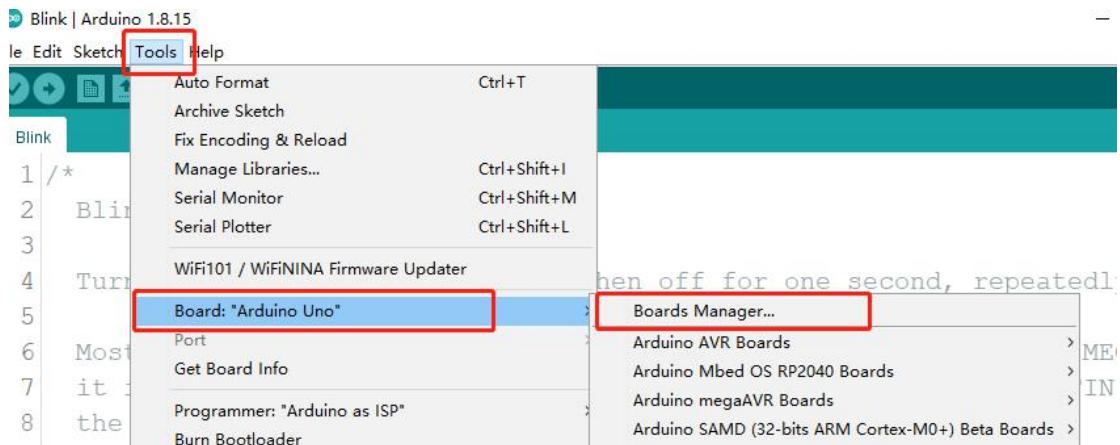
http://arduino.esp8266.com/stable/package_esp8266com_index.json, As shown in the figure below. Then, click the ok button:



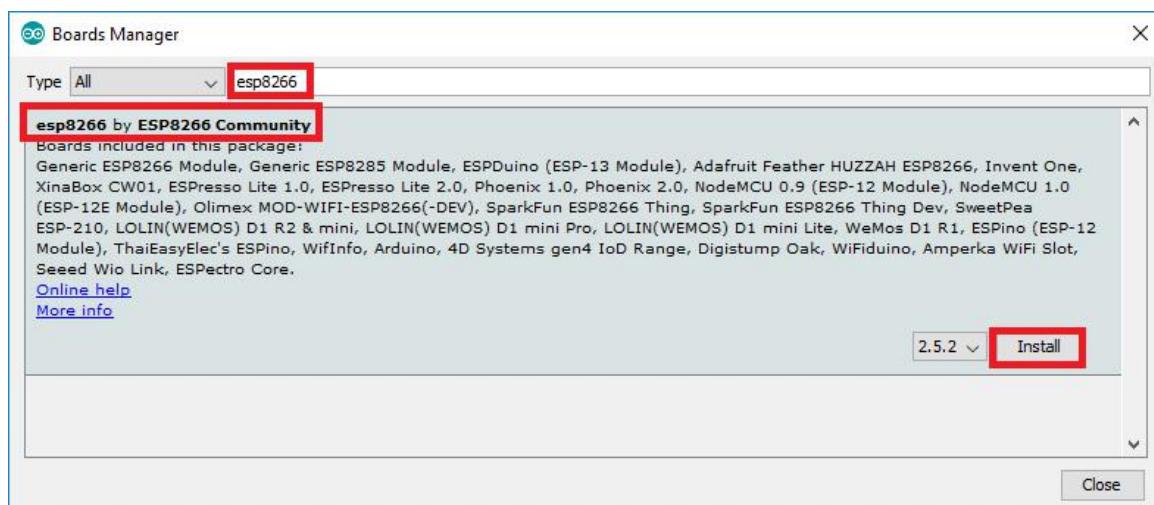
Note: If you already have ESP32 board urls, you can use commas to separate the urls, as shown below:

https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

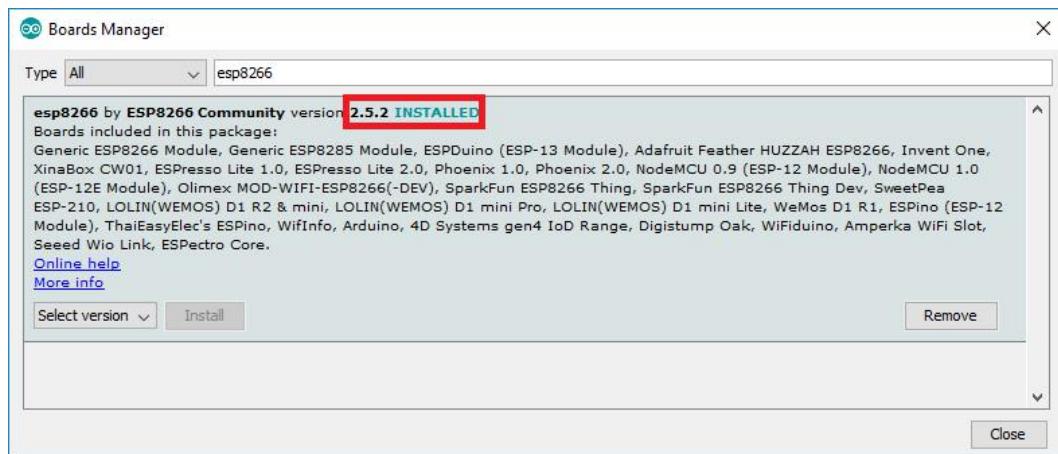
Open board Manager. Go to tools → board → board manager...



Search for ESP8266 and press the install button of "ESP8266 by ESP8266 Community" :



Wait a few seconds to install.



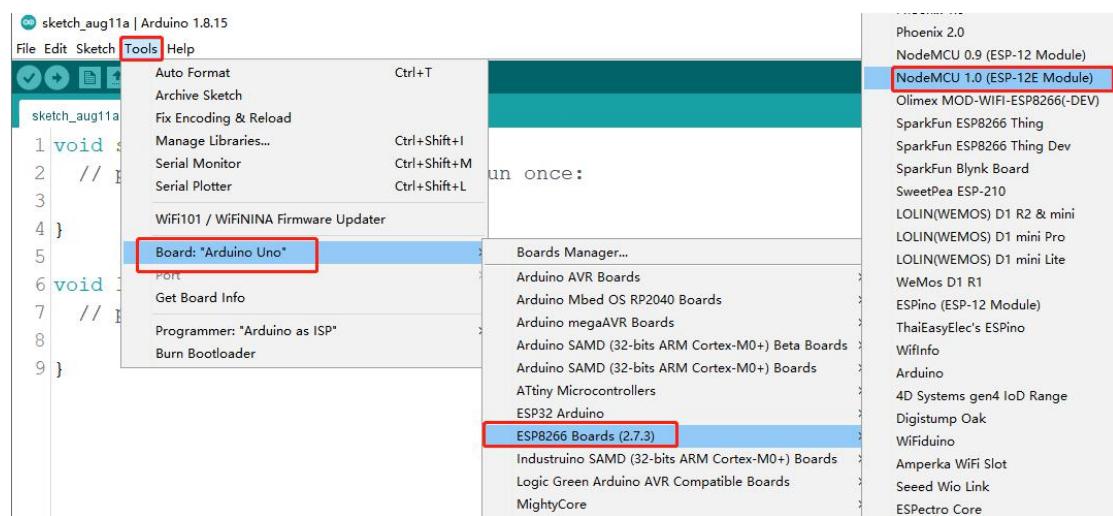
21.2 Test the installation

To test the ESP8266 plug-in installation, let's see if we can make the LED blink through the ESP8266 using the Arduino programming language.

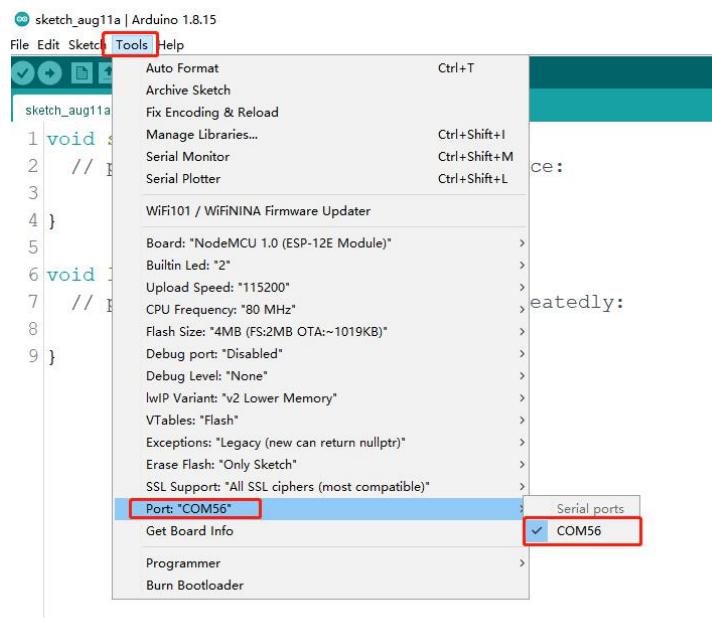
Upload a sketch

Upload the sketch to ESP-12E

Plug your development board into your computer. Make sure you choose the right circuit board:



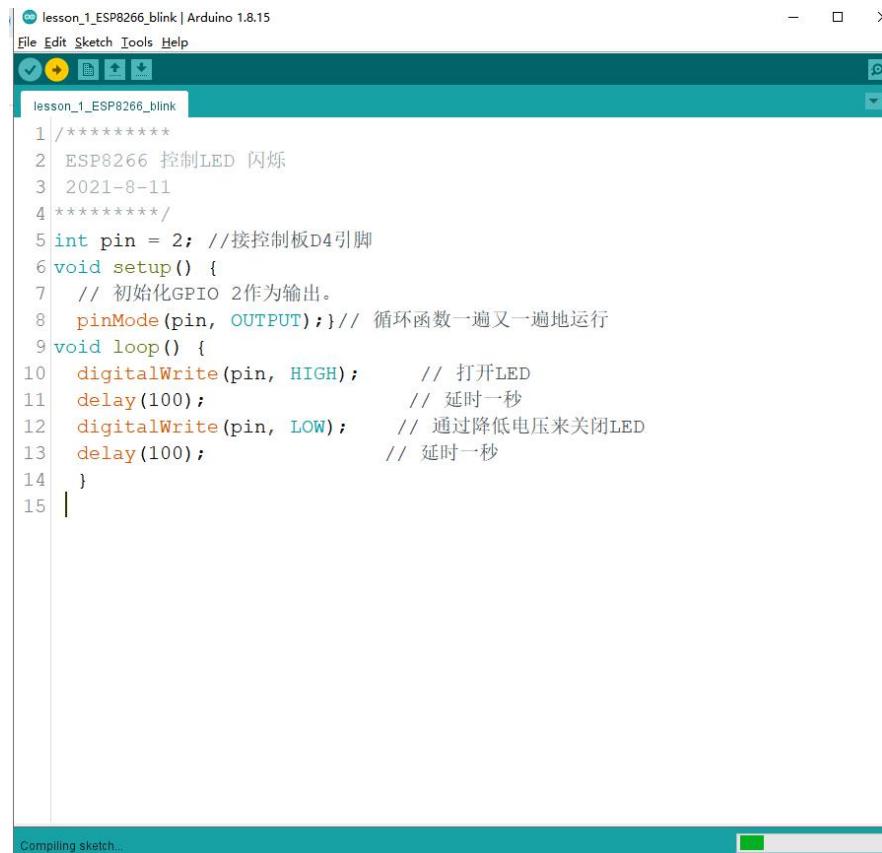
You also need to select ports:



Then, copy the supplied code:

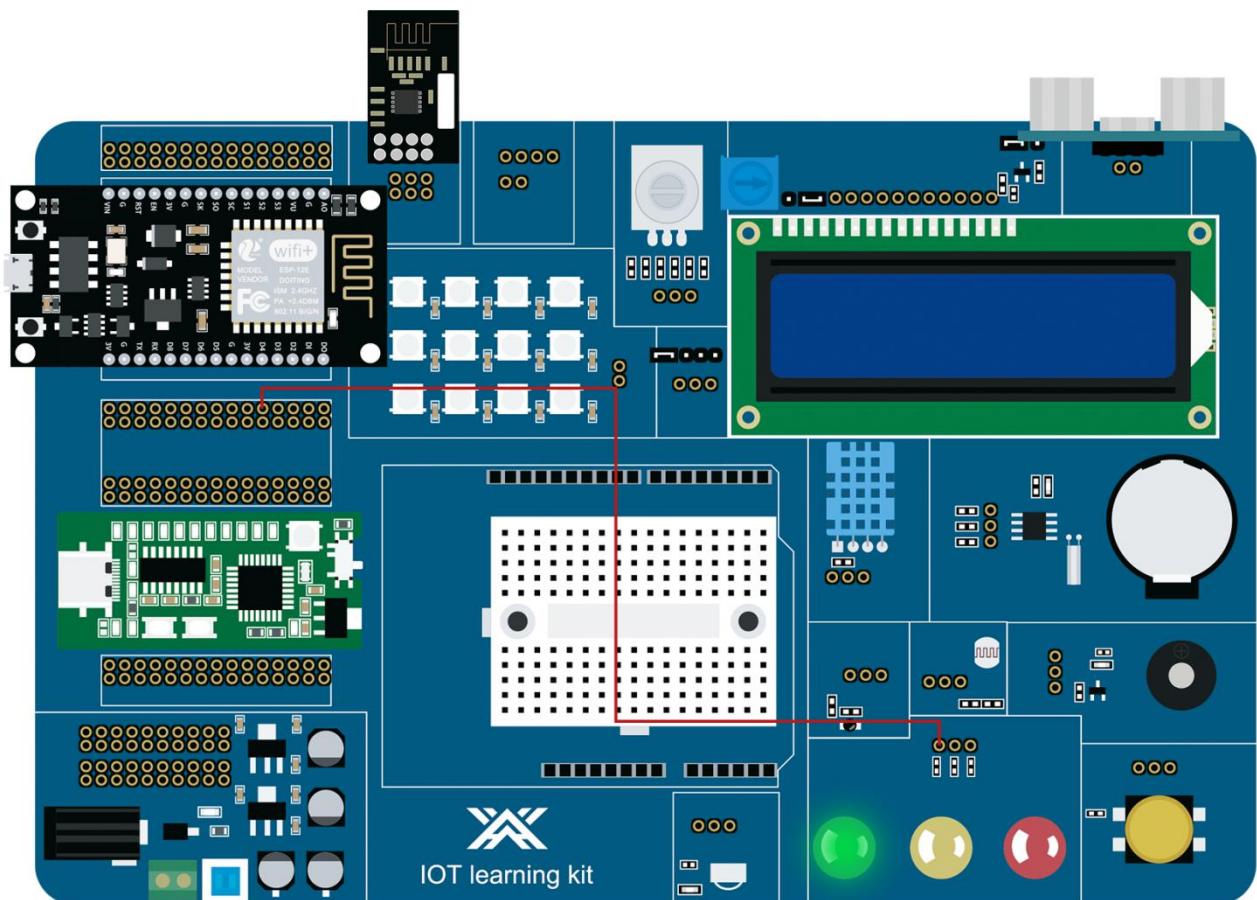
```
int pin = 2;void setup() {  
    // Initialize GPIO 2 as output.  
    pinMode(pin, OUTPUT);}// The loop function runs over and over again  
  
void loop() {  
    digitalWrite(pin, HIGH);    // Open the LED  
    delay(1000);                // Delay for a second  
    digitalWrite(pin, LOW);     // Turn off the LED by lowering the voltage  
    delay(1000);                //Delay for a second  
}
```

Click the "Upload" button in the Arduino IDE and wait a few seconds until you see the "Upload Completed" message. It's in the lower left corner.



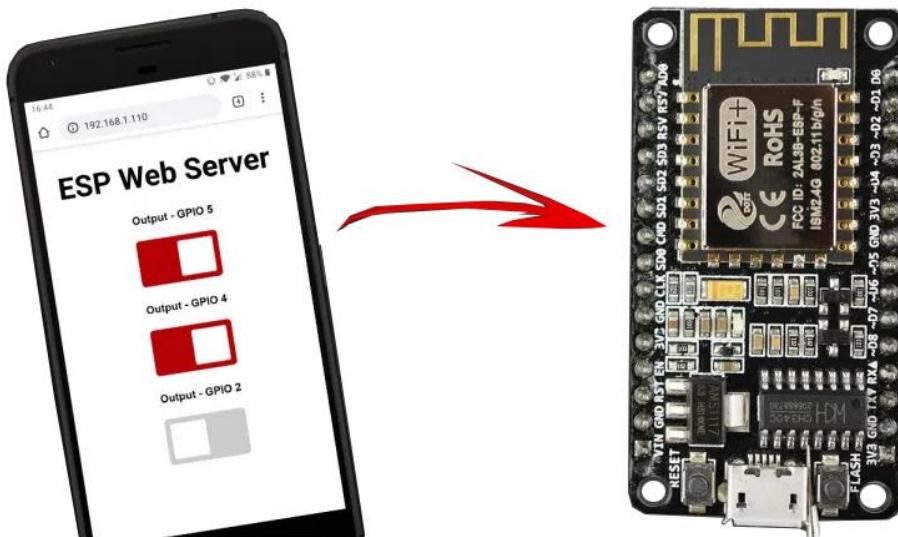
21.3 Wiring Diagram

When D4 is connected to the green LED pin in the traffic light module, the LED flashes.



Lesson 22 ESP8266 NodeMCU WiFi control traffic light module

In this tutorial, you will learn how to use the ESP8266 NodeMCU board to build an asynchronous Web server to control its output. The board will be programmed using the Arduino IDE and we will use the ESPAsyncWebServer library.

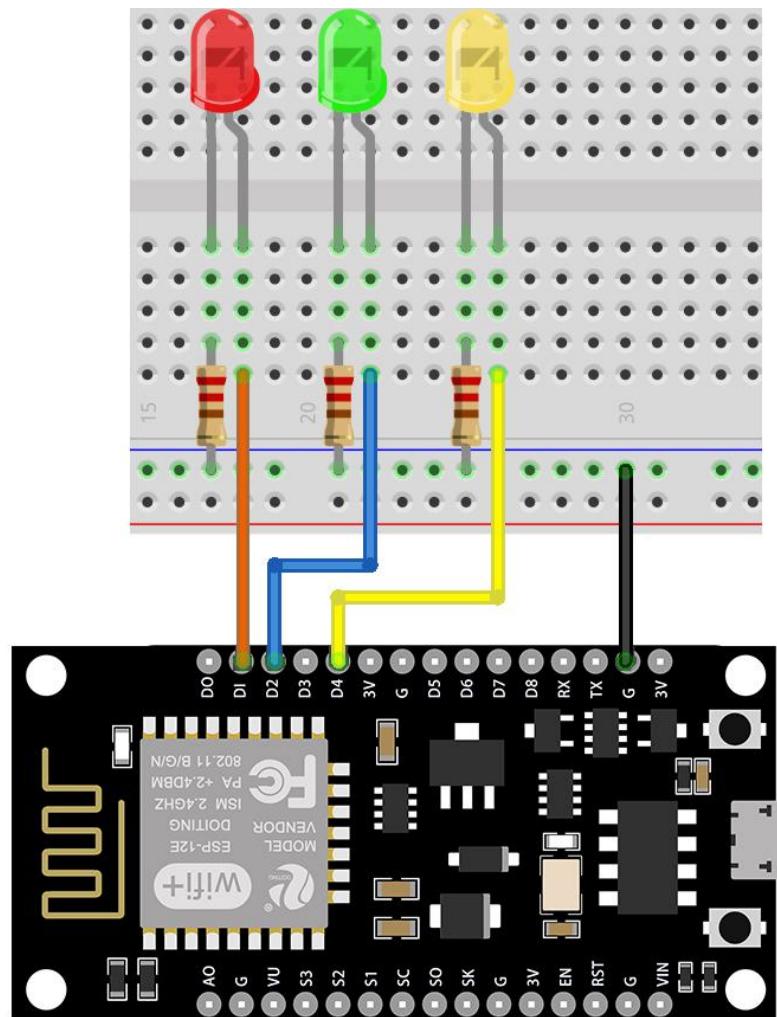


22.1 Asynchronous Network Server

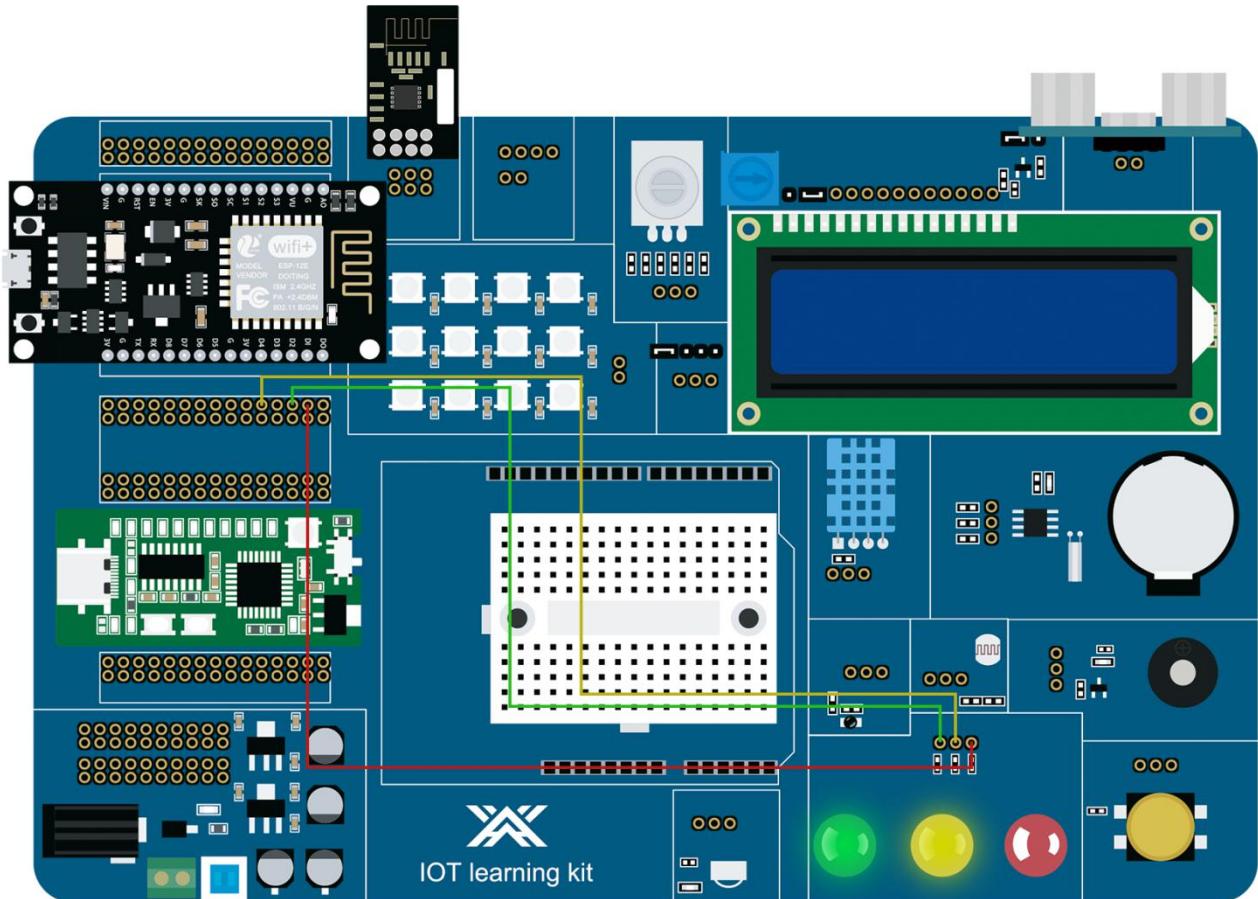
To build a Web server, we'll use the ESPAsyncWebServer library, which provides an easy way to build an asynchronous Web server.

22.2 Schematic Diagram:

Before continuing with the code, connect the three leds to the ESP8266. We connected the LED to GPIO 5, 4 and 2.



22.3 Wiring Diagram



Installation library - ESP Exotic Web server

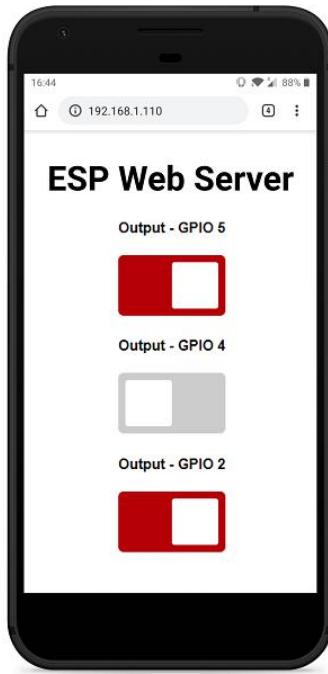
To build a Web server, you need to install the following libraries.

- **ESPAsyncWebServer**
- **ESPAsyncTCP**

These libraries cannot be installed through the Arduino library manager, so you need to copy the library files to the Arduino installation library folder. Or, in your Arduino IDE, you can go to **Sketch > Include Library > Add .zip Library** and select the library you just downloaded.

Project overview

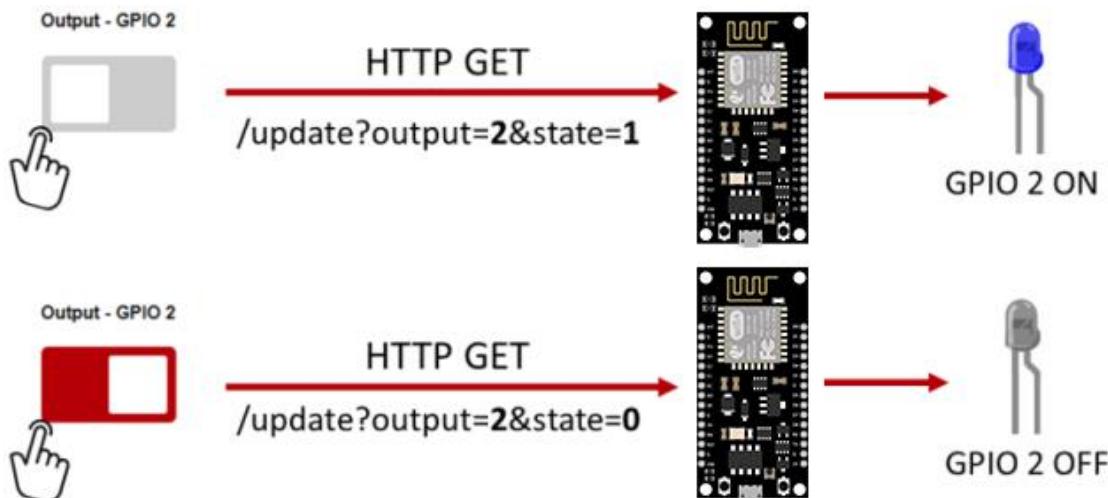
To better understand the code, let's take a look at how the Web server works.



The Web Server contains a title "ESP Web Server" and three buttons (toggle switches) to control the three outputs. Each slider button has a label indicating the GPIO output pin. You can easily remove/add more output.

When the slider is red, the output is on (its status is HIGH). If you switch the slider, it will turn off the output (changing the state to LOW).

When the slider is gray, the output is off (its status is LOW). If you switch the slider, it turns on the output (changing the state to HIGH).



Let's see what happens when we toggle buttons. We will see an example of GPIO 2. The other buttons work similarly.

1. In the first case, you toggle the button to turn on GPIO 2. When this happens, will it update in /? Output = 2 & status = 1 URL. Based on this URL, we change the state of GPIO 2 to 1 (HIGH) and turn on the LED.

2. In the second example, when you toggle the button to turn off GPIO 2. When this happens in/update? Output = 2 & status = 0 URL. Based on this URL, we change the state of GPIO 2 to 0 (LOW) and turn off the LED.

22.4 ESP asynchronous Web server code

```
// Library files that need to be installed

#include <ESP8266WiFi.h>

#include <ESPAsyncTCP.h>

#include <ESPAsyncWebServer.h> // Replace with your network credentials

const char* ssid = "REPLACE_WITH_YOUR_SSID"; //Enter your wifi name

const char* password = "REPLACE_WITH_YOUR_PASSWORD"; //Enter wifi password

const char* PARAM_INPUT_1 = "output";

const char* PARAM_INPUT_2 = "state";

//Create an AsyncWebServer object on port 80

AsyncWebServer server(80);

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html><head>

<title>ESP Web Server</title>

<meta name="viewport" content="width=device-width, initial-scale=1">
```

```
<link rel="icon" href="data:,">

<style>

    html {font-family: Arial; display: inline-block; text-align: center; }

    h2 {font-size: 3.0rem; }

    p {font-size: 3.0rem; }

    body {max-width: 600px; margin: 0px auto; padding-bottom: 25px; }

    .switch {position: relative; display: inline-block; width: 120px; height: 68px}

    .switch input {display: none}

    .slider {position: absolute; top: 0; left: 0; right: 0; bottom: 0; background-color: #ccc; border-radius: 6px}

    .slider:before {position: absolute; content: ""; height: 52px; width: 52px; left: 8px; bottom: 8px; background-color: #fff; -webkit-transition: .4s; transition: .4s; border-radius: 3px}

    input:checked+.slider {background-color: #b30000}

    input:checked+.slider:before {-webkit-transform: translateX(52px); -ms-transform: translateX(52px); transform: translateX(52px)}

</style>

</head>

<body>

    <h2>ESP Web Server</h2>

    %BUTTONHOLDER%

    <script>function toggleCheckbox(element) {

        var xhr = new XMLHttpRequest();
```

```
    if(element.checked){ xhr.open("GET",
"/update?output="+element.id+"&state=1", true); }

    else { xhr.open("GET", "/update?output="+element.id+"&state=0",
true); }

    xhr.send();}

</script>

</body>

</html>

)rawliteral;// Replace placeholders in web pages with button sections

String processor(const String& var){

//Serial.println(var);

if(var == "BUTTONPLACEHOLDER"){

    String buttons = "";

    buttons += "<h4>Output - GPIO 5</h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"5\" " +
outputState(5) + "><span class=\"slider\"></span></label>";

    buttons += "<h4>Output - GPIO 4</h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"4\" " +
outputState(4) + "><span class=\"slider\"></span></label>";

    buttons += "<h4>Output - GPIO 2</h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"2\" " +
outputState(2) + "><span class=\"slider\"></span></label>";

    return buttons;

}

return String();

}

String outputState(int output){
```

```
if(digitalRead(output)){  
  
    return "checked";  
  
}  
  
else {  
  
    return "";  
  
}  
  
}  
  
void setup(){  
  
    // Serial port for debugging  
  
    Serial.begin(115200);  
  
    pinMode(5, OUTPUT);  
  
    digitalWrite(5, LOW);  
  
    pinMode(4, OUTPUT);  
  
    digitalWrite(4, LOW);  
  
    pinMode(2, OUTPUT);  
  
    digitalWrite(2, LOW);  
  
    // Connect to a wireless network  
  
    WiFi.begin(ssid, password);  
  
    while (WiFi.status() != WL_CONNECTED) {  
  
        delay(1000);  
  
        Serial.println("Connecting to WiFi..");  
    }  
}
```

```
}

// ESP local IP address is displayed"

Serial.println(WiFi.localIP());

// Route for root / web page

server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){

    request->send_P(200, "text/html", index_html, processor);

});

// Send a GET request to
<ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>

server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {

    String inputMessage1;

    String inputMessage2;

    // GET input1 value on
<ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>

    if (request->hasParam(PARAM_INPUT_1) &&
request->hasParam(PARAM_INPUT_2)) {

        inputMessage1 = request->getParam(PARAM_INPUT_1)->value();

        inputMessage2 = request->getParam(PARAM_INPUT_2)->value();

        digitalWrite(inputMessage1.toInt(), inputMessage2.toInt());

    }

    else {

        inputMessage1 = "No message sent";

        inputMessage2 = "No message sent";    }

```

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

Serial.print(inputMessage1);

Serial.print(" - Set to: ");

Serial.println(inputMessage2);

request->send(200, "text/plain", "OK");

});

//Start server

server.begin();}void loop() {}
```

22.5 How the code works

In this section, we explain how the code works.

Import libraries

First, import the required libraries. You need to include wireless Internet access, ESPA sync network server and ESPA sync TCP library.

```
#include <ESP8266WiFi.h>

#include <ESPAsyncTCP.h>

#include <ESPAsyncWebServer.h>
```

Set up network credentials

Insert your network credentials in the following variables so that ESP8266 can connect to your local network.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
```

```
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

The input parameters

To check the parameters passed through the URL (GPIO number and its state), we create two variables, one for output and one for state.

```
const char* PARAM_INPUT_1 = "output";  
const char* PARAM_INPUT_2 = "state";
```

Keep in mind that ESP8266 receives requests like: / Update? Output =2& state =0

AsyncWebServer object

Create an object on port 80 of the asynchronous Web server.

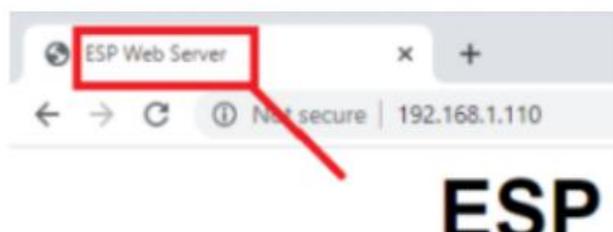
```
AsyncWebServer server(80);
```

Build a web page

All HTML text with styles and JavaScript is stored in index_HTML variable. Now we'll go through the HTML text and see what each section does.

The title is located at < title> And </title> Tags. The title is exactly what it sounds like: the title of your document, which appears in the title bar of your Web browser. In this case, it is the "ESP Web server."

```
<title>ESP Web Server</title>
```



The following <meta>; Tabs make your web pages responsive in any browser (laptop, tablet, or smartphone).

```
<meta name="viewport"  
      content="width=device-width, initial-scale=1">
```

The next line blocks requests for web site ICONS. In this case, we don't have a website icon. A website icon is a website icon that appears next to the title of a Web browser TAB. If we don't add the following line, ESP will receive a request for FavIcon every time we visit the Web server.

```
<link rel="icon" href="data:,">
```

In <style> </style>, Between the tabs, we added some CSS to set the style of the page. We won't go into the details of how this CSS style works.

```
<style> html {font-family: Arial; display: inline-block;  
text-align: center;}  
  
h2 {font-size: 3.0rem;}  
  
p {font-size: 3.0rem;}  
  
body {max-width: 600px; margin: 0px auto;  
padding-bottom: 25px;}  
  
.switch {position: relative; display: inline-block;  
width: 120px; height: 68px}  
  
.switch input {display: none}  
  
.slider {position: absolute; top: 0; left: 0; right: 0;  
bottom: 0; background-color: #ccc; border-radius: 6px}  
  
.slider:before {position: absolute; content: ""; height:
```

```
52px; width: 52px; left: 8px; bottom: 8px; background-color: #fff; -webkit-transition: .4s; transition: .4s; border-radius: 3px}

input:checked+.slider {background-color: #b30000}

input:checked+.slider:before { -webkit-transform: translateX(52px); -ms-transform: translateX(52px); transform: translateX(52px) }

</style>
```

HTML text

The inside of the<body> </ body>Tags are the content we add to the page.

The < h2 >< /h2 >; Add TAB titles to web pages. In this case, "ESP Web Server" text, but you can add any other text.

```
<h2>ESP Web Server</h2>
```

After the title, we have the button. How buttons appear on web pages (red: if GPIO is on; Or gray: if GPIO is off) depends on the current GPIO state.

When you visit a Web server page, you want it to display the correct current GPIO state. So, instead of adding HTML text, we'll add a placeholder to build the button % button placeholder %. When the web page loads, this placeholder is replaced with the actual HTML text to build the button with the correct state.

%BUTTONPLACEHOLDER%

JavaScript

Then, as we explained earlier, there is some JavaScript responsible for making HTTP GET requests when you switch buttons.

```
<script>function toggleCheckbox(element) {  
  
    var xhr = new XMLHttpRequest();  
  
    if(element.checked){ xhr.open("GET",  
"/update?output="+element.id+"&state=1", true); }  
  
    else { xhr.open("GET", "/update?output="+element.id+"&state=0",  
true); }  
  
    xhr.send();}</script>
```

Here is the line that made the request:

```
if(element.checked){xhr.open("GET",  
"/update?output="+element.id+"&state=1", true); }
```

Element. Id Returns the ID of an HTML element. The ID of each button will be the GPIO of the control, as we will see in the next section:

GPIO 5 button » element. id = 5

GPIO 4 button » element. id = 4

GPIO 2 button » element. id = 2

The processor

Now we need to create the processor () function to replace the placeholders in the HTML text with what we define.

When requesting a web page, check the HTML for any placeholders. If it finds the % button placeholder % placeholder, it returns HTML text to create the button.

```
String processor(const String& var){  
  
    //Serial.println(var);  
  
    if(var == "BUTTONPLACEHOLDER")  
  
    { String buttons = "";
```

```

buttons+="<h4>Output-GPIO5</h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"5\" " +
outputState(5) + "><span class=\"slider\"></span></label>"; buttons +=
"<h4>Output - GPIO 4</h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"4\" " +
outputState(4) + "><span class=\"slider\"></span></label>"; buttons +=
"<h4>Output - GPIO 2</h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"2\" " +
outputState(2) + "><span class=\"slider\"></span></label>"; return
buttons; } return String(); }

```

You can easily delete or add more rows to create more buttons.

Let's look at how buttons are created. We create a String variable button named String that contains the HTML text used to build the button. We concatenate the HTML text with the current output state so that the toggle button is either gray or red. The current output state is determined by the output state (<GPIO>) Function (which takes a GPIO number as an argument). See below:

```

buttons += "<h4>Output - GPIO 2</h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"2\" " + outputState(2) +
"><span class=\"slider\"></span></label>";

```

Use \ so that we can pass "" in the string.

This output status () function returns "check" if GPIO is in the open state or the field is empty if GPIO is off.

```
StringoutputState(intoutput)
```

```
{ if(digitalRead(output))
```

```
{ return "checked"; }

else { return ""; }

}
```

Therefore, when GPIO 2 opens, the HTML text will be:

```
<h4>Output - GPIO 2</h4> <label class="switch"> <input
type="checkbox" onchange="toggleCheckbox(this)" id="2"
checked><span class="slider"></span> </label>
```

Let's break it down into smaller pieces to understand how it works.

In HTML, a toggle switch is an input type. Described `<input>` ; Flags the specified input field where the user can enter data. Toggle switch is an input field type check box. There are many other input field types.

```
<input type="checkbox">
```

The check box can be checked or unchecked. When you check, you have the following:

```
<input type="checkbox" checked>
```

Among these changes are the event attributes that occur when we change the value of an element (check box). Every time you check or uncheck the toggle switch, it calls the JavaScript function (this) that toggles the checkbox () for that particular element ID.

This ID specifies a unique ID for the HTML element. Id allows us to manipulate elements using JavaScript or CSS.

```
<input type="checkbox" onchange="toggleCheckbox(this)" id="2" checked>
```

setup()

Initialize the monitor in setup() for debugging.

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

Use the pinMode () function and set them to LOW when the ESP8266 is first started. If you add more GPIO, follow the same process.

```
pinMode(2, OUTPUT);
```

```
pinMode(5, OUTPUT);
```

```
digitalWrite(5, LOW);
```

```
pinMode(4, OUTPUT);
```

```
digitalWrite(4, LOW);
```

```
pinMode(2, OUTPUT);
```

```
digitalWrite(2, LOW);
```

Connect to your local network and print the ESP8266 IP address.

```
WiFi.begin(ssid, password);  
while (WiFi.status() != WL_CONNECTED) {  
    delay(1000);
```

```
Serial.println("Connecting to WiFi.."); }
```

```
// Print ESP Local IP Address
```

```
Serial.println(WiFi.localIP());
```

In Setup (), you need to handle what happens when ESP8266 receives the request. As we saw earlier, you get requests like this:

```
<ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>
```

Therefore, we check if the request contains the PARAM_INPUT1 variable values (output) and PARAM_INPUT2(state) and store the corresponding values in the input 1 message and input 2 message variables.

```
if(request->hasParam(PARAM_INPUT_1)&&request->hasParam(PARAM_INPUT_2)) { inputMessage1 = request->getParam(PARAM_INPUT_1)->value();  
inputMessage2 = request->getParam(PARAM_INPUT_2)->value();
```

We then control the corresponding state of the corresponding GPIO (input message 1 variable holds the GPIO number and input message 2 holds the status-0 or 1)

```
digitalWrite(inputMessage1.toInt(),inputMessage2.toInt());
```

Here is the complete code for handling HTTP GET/UPDATE requests:

```
server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {  
  
    String inputMessage1;  
  
    String inputMessage2;  
  
    // GET input1 value on  
<ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>  
  
    if (request->hasParam(PARAM_INPUT_1) &&  
request->hasParam(PARAM_INPUT_2)) {
```

```

inputMessage1 = request->getParam(PARAM_INPUT_1)->value();

inputMessage2 = request->getParam(PARAM_INPUT_2)->value();

digitalWrite(inputMessage1.toInt(), inputMessage2.toInt());

}

else {

    inputMessage1 = "No message sent";

    inputMessage2 = "No message sent";

}

Serial.print("GPIO: ");

Serial.print(inputMessage1);

Serial.print(" - Set to: ");

Serial.println(inputMessage2);

request->send(200, "text/plain", "OK");});

```

Finally, start the server:

```
server.begin();
```

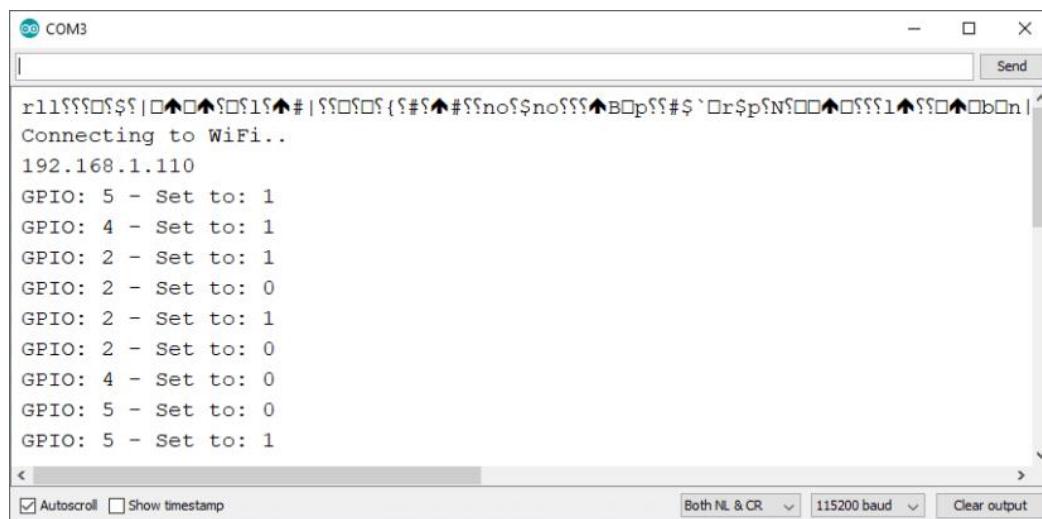
demonstration

After uploading the code to ESP8266, turn on the serial monitor at 115200 baud rate. Press the onboard RST/EN button. You should get its IP address.

Open a browser and type ESP IP address. You will be able to access similar web pages.



Press the toggle button to control ESP GPIO. In the meantime, you should receive the following message in the serial monitor to help you debug your code.



The screenshot shows a terminal window titled "COM3" with the following text output:

```
r11??#o$?|o#o#o#o#1?#|?#o#o# {?#?#o#no#no#?#o#B#o#p#?#?`#o#r$p#N#o#o#o#1#o#b#o#n  
Connecting to WiFi..  
192.168.1.110  
GPIO: 5 - Set to: 1  
GPIO: 4 - Set to: 1  
GPIO: 2 - Set to: 1  
GPIO: 2 - Set to: 0  
GPIO: 2 - Set to: 1  
GPIO: 2 - Set to: 0  
GPIO: 4 - Set to: 0  
GPIO: 5 - Set to: 0  
GPIO: 5 - Set to: 1
```

At the bottom of the window, there are checkboxes for "Autoscroll" and "Show timestamp", and dropdown menus for "Both NL & CR", "115200 baud", and "Clear output".

You can also access the web server from a browser in your smartphone. Every time you open the Web server, it displays the current GPIO state. Red indicates THAT GPIO is on and gray indicates that GPIO is off.



Lesson 23 ESP8266 Node MCU button control LED

In this introductory course, you will learn how to read digital inputs such as button switches and control digital outputs such as leds using the ESP8266 NodeMCU board with Arduino IDE.

23.1 ESP8266 NodeMCU controls digital output

First, you need to set the GPIO you are going to control as output. Use pinMode () as follows:

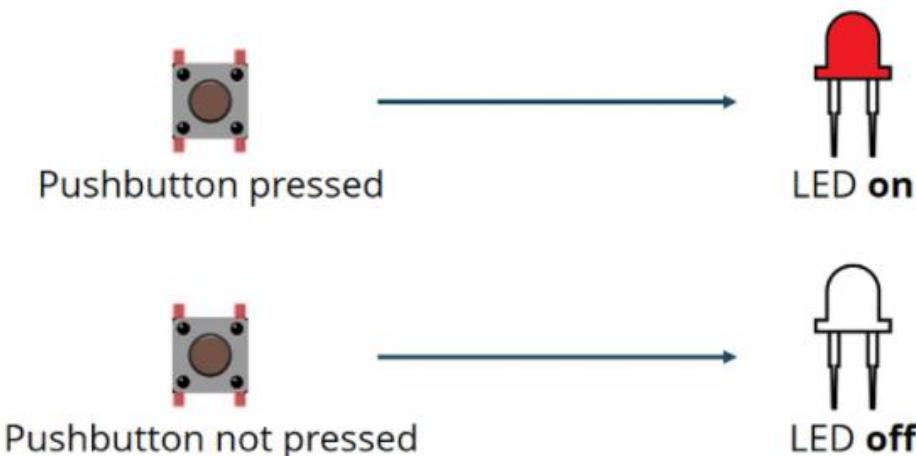
```
pinMode(GPIO, OUTPUT);
```

To read numeric input, such as buttons, you can use the digitalWrite () function, which takes the GPIO (integer) you point to as an argument.

```
digitalRead(GPIO);
```

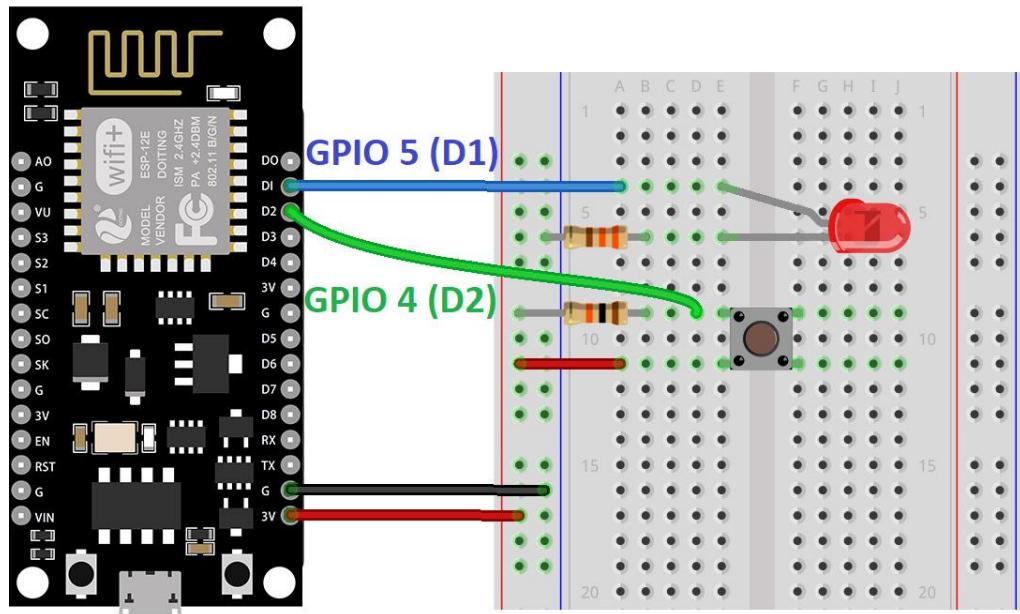
23.2 Project example

To show you how to use digital input and digital output, we'll build an example of a simple project with buttons and leds. We will read the status of the button and light up the LED accordingly, as shown below.



23.3 Wiring diagram:

Before proceeding, you need to assemble a circuit with leds and buttons. We connected the LED to the universal I/O outlet 5 (D1) and the button universal I/O outlet 4 (D2).



23.4 Working Code:

```
// set pin numbers

const int buttonPin = 4;          // the number of the pushbutton pin

const int ledPin = 5;             // the number of the LED pin

// variable for storing the pushbutton status

int buttonState = 0;

void setup() {

    // initialize the pushbutton pin as an input

    pinMode(buttonPin, INPUT);
```

```
// initialize the LED pin as an output  
  
pinMode(ledPin, OUTPUT);  
  
}  
  
void loop() {  
  
    // read the state of the pushbutton value  
  
    buttonState = digitalRead(buttonPin);  
  
    // check if the pushbutton is pressed.  
  
    // if it is, the buttonState is low  
  
    if (buttonState == LOW) {  
  
        // turn LED on  
  
        digitalWrite(ledPin, HIGH);  
  
    }  
  
    else {  
  
        // turn LED off  
  
        digitalWrite(ledPin, LOW);  
  
    }  
  
}
```

23.5 Code working principle:

In the following two lines, you create variables to assign pins:

```
const int buttonPin = 4;  
  
const int ledPin = 5;
```

The button is connected to universal I/O outlet 4 and the LED is

connected to universal I/O outlet 5. When using an Arduino IDE with ESP8266, 4 corresponds to the generic I/O 4 and 5 corresponds to the generic I/O 5.

Next, create a variable to hold the button state. By default, it is 0 (not pressed).

```
int buttonState = 0;
```

In setup (), you initialize the button as input and the LED as output. To do this, you use pinMode () to accept the pin and mode function you point to: input or output.

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

Inside loop () is where you read the button state and set the LED accordingly.

In the next line, you read the button state and save it in the variable button state. As we saw earlier, you use the digitalRead () function.

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

The following if statement checks whether the button state is HIGH. If so, it turns on the LED using the digitalWrite() function, which takes ledPin as an argument and sets the state to HIGH.

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

If the button state is not "HIGH", the LED is set to off. Simply set LOW to the second argument in the digitalWrite() function.

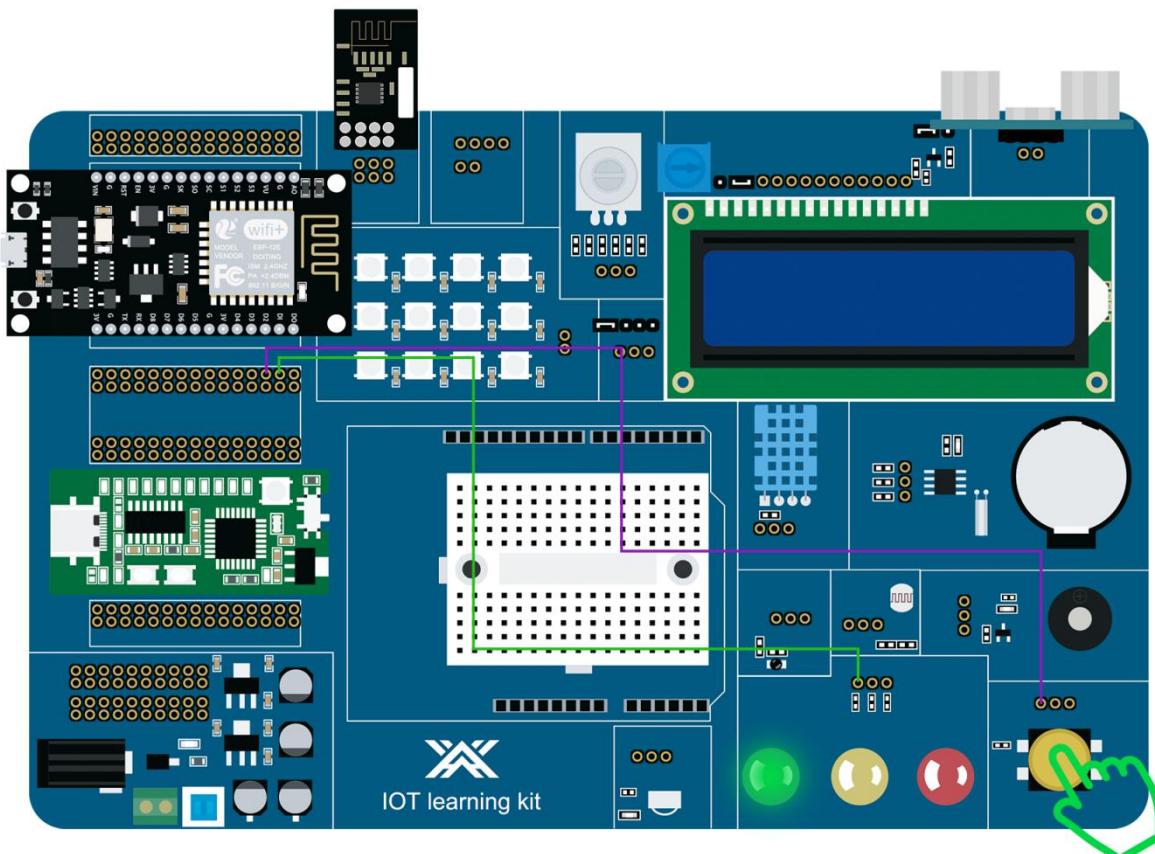
```
else { digitalWrite(ledPin, LOW); }
```

23.6 Upload code

Before clicking the upload button, go to Tools > Board, and then select the Board you are using. NodeMCU 1.0 (ESP-12 E Module).

Go to tools > Port and select the COM port to which ESP8266 is connected. Then, press the upload button and wait for the "Upload completed" message.

23.7 Object diagram:



Lesson 24 ESP8266 Controlling LED

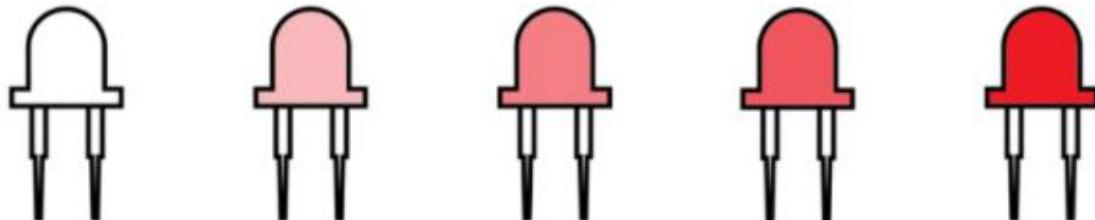
Brightness (PWM)

This tutorial shows how to generate PWM signals through ESP8266 NodeMCU using the Arduino IDE. For example, we will dim the LED brightness by changing the duty cycle over time.

24.1 ESP8266 NodeMCU PWM

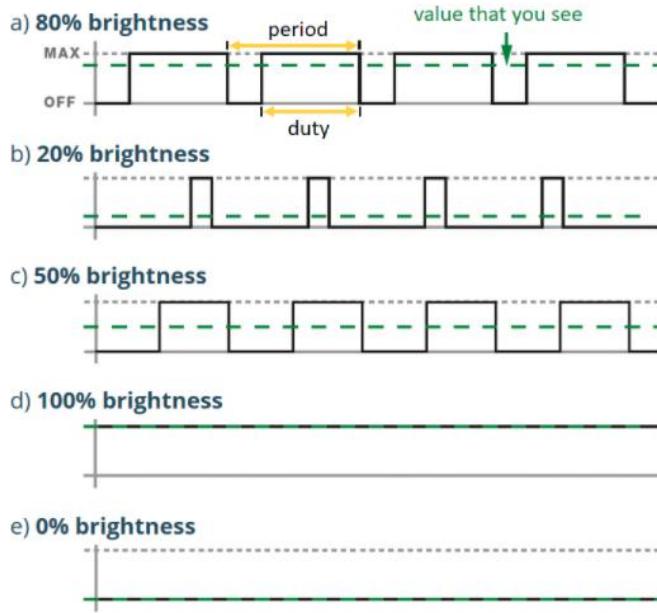
The ESP8266 GPIO can be set to output 0V or 3.3V, but cannot output any voltage between them. However, you can use pulse width modulation (PWM) to output "false" intermediate voltages, which is how you generate different levels of LED brightness for this project.

If you quickly alternate the voltage of the LED between high and low levels, your eye can't keep up with the speed of the LED switch; You'll just see some gradients in the brightness.



This is basically how PWM works -- by producing an output that varies between HIGH and LOW at a very HIGH frequency.

Duty cycle is part of the period during which the LED is set to high level. The following figure illustrates how PWM works.



An LED with a duty cycle of 50% has a brightness of 50%, a duty cycle of 0 means the LED is fully off, and a duty cycle of 100 means the LED is fully on. Changing duty cycle is how you produce different brightness levels.

analogWrite()

To generate a PWM signal on a given pin, use the following function:

```
analogWrite(pin, value);
```

Pin: PWM can be used with pins 0 to 16

value: Should be in the 0 to PWM range, default is 1023. When the value is 0, the PWM on this pin is disabled. The value 1023 corresponds to 100% duty cycle

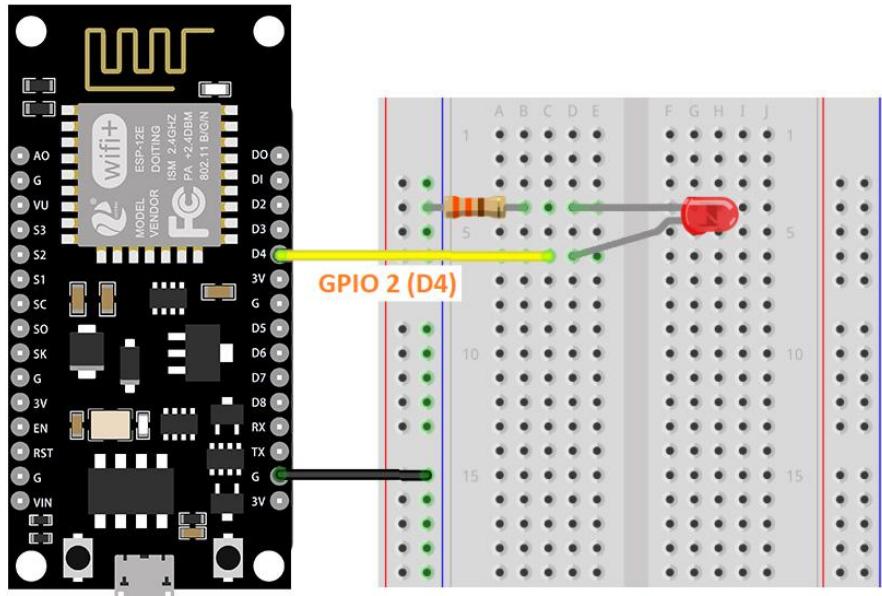
You can change the PWM range by calling:

```
analogWriteRange(new_range);
```

By default, the ESP8266 PWM frequency is 1kHz. You can change the PWM frequency in the following ways:

```
analogWriteFreq(new_frequency);
```

24.2 Schematic diagram:



ESP8266 NodeMCU PWM code:

```

const int ledPin = 2; //Defines led pins (D4)

void setup() {

}

void loop() {
    //Improve LED brightness

    for(int dutyCycle = 0; dutyCycle < 1023; dutyCycle++){

        // Use PWM to change LED brightness

        analogWrite(ledPin, dutyCycle);

        delay(1);

    }
}

```

```
// Reduce LED brightness

for(int dutyCycle = 1023; dutyCycle > 0; dutyCycle--){
    // Use PWM to change LED brightness

    analogWrite(ledPin, dutyCycle);

    delay(1);
}
```

Code working principle:

First define the pin LED connected to. In this case, the LED is connected to the universal INPUT/output port 2 (D4).

```
const int ledPin = 2;
```

In Loop (), you can change the duty cycle between 0 and 1023 to increase the LED brightness.

```
for(int dutyCycle = 0; dutyCycle < 1023; dutyCycle++){
    // Use PWM to change LED brightness

    analogWrite(ledPin, dutyCycle);

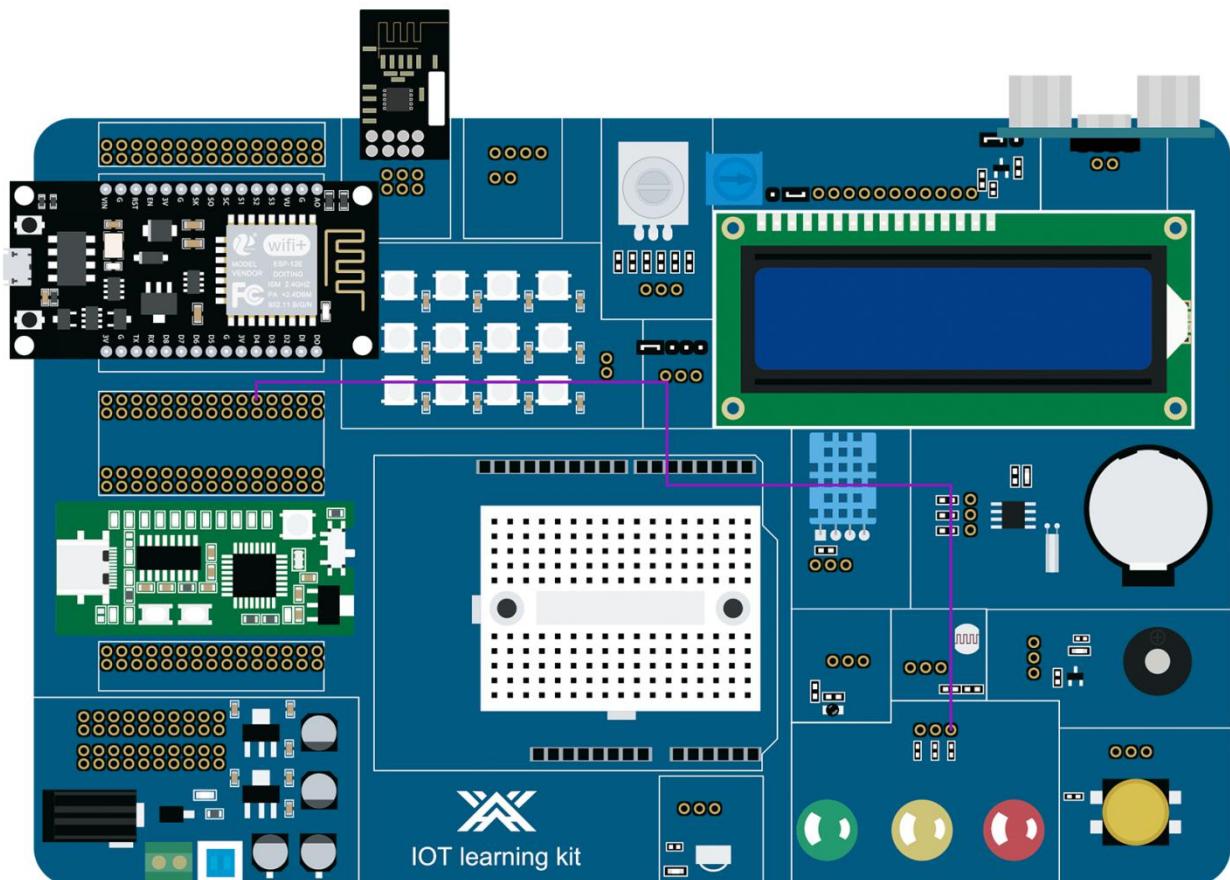
    delay(1);}
```

To set the LED brightness, you need to use the function that `analogWrite()` accepts GPIO as an argument, where you want to get the PWM signal and a value between 0 and 1023 to set the duty cycle.

24.3 Upload code:

In your Arduino IDE, go to Tools > Board and select your ESP8266 model.

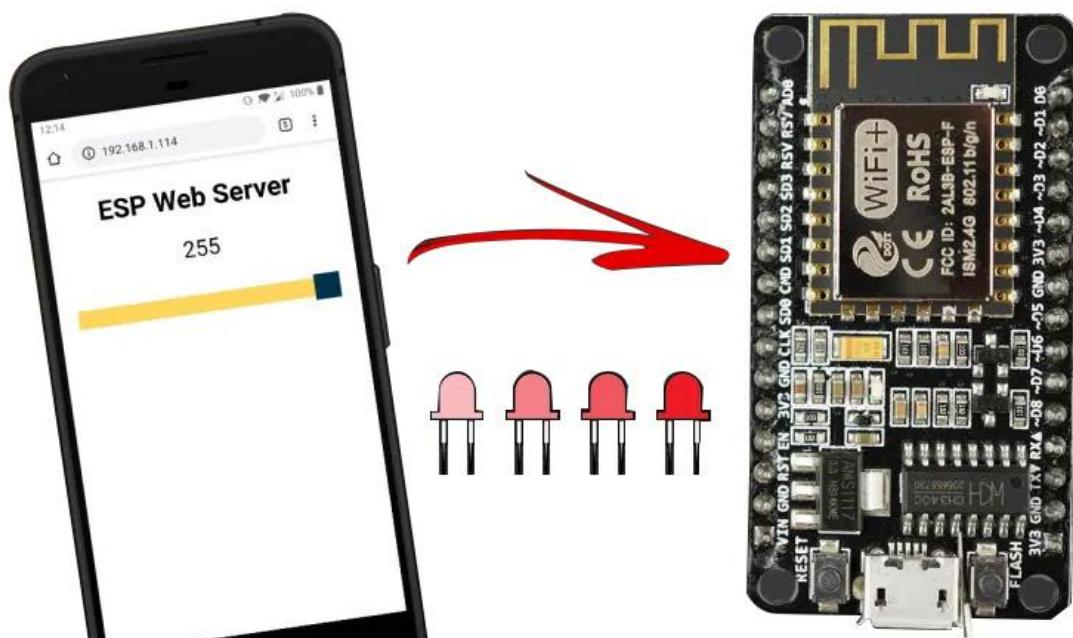
24.4 Wiring Diagram:



Lesson 25 ESP8266 Node MCU Web Server

Control LED Brightness (PWM)

This tutorial shows how to build the ESP8266 NodeMCU Web server with sliders to control LED brightness. You'll learn how to add a slider to your Web server project, get its value, and save it in a variable that ESP8266 can use. We will use this value to control the duty cycle of the PWM signal and change the brightness of the LED. For example, you can also control servo motors instead of just leds.



In addition, you can modify the code in this tutorial to add a slider for your project to set thresholds or any other values you need to use in your code.



ESP8266 hosts a Web server that displays Web pages with sliders;

When you move the slider, you make an HTTP request to ESP8266 with the new slider value;

HTTP requests take the following format: GET/slider? Value =SLIDERVALUE, where the SLIDERVALUE is a number between 0 and 1023. You can modify the slider to include any other scope;

ESP8266 gets the current value of the slider from the HTTP request;

ESP8266 adjusts PWM duty ratio according to slider value;

This is useful for controlling the brightness of leds (as we will do in this example), servomotors, setting thresholds, or other applications.

Arduino IDE

We will program the ESP8266 NodeMCU board using the Arduino IDE, so make sure you have the ESP8266 board installed in your Arduino IDE before continuing with this tutorial.

25.1 Working Code:

```
// Add the required libraries

#include <ESP8266WiFi.h>

#include <ESPAsyncTCP.h>

#include <ESPAsyncWebServer.h>

// Replace with your network credentials (enter your WiFi name and WiFi password)

const char* ssid = "REPLACE_WITH_YOUR_SSID";

const char* password = "REPLACE_WITH_YOUR_PASSWORD";

const int output = 2;

String sliderValue = "0";

const char* PARAM_INPUT = "value";
```

```
// Create an AsyncWebServer object on port 80

AsyncWebServer server(80);

const char index_html[] PROGMEM = R"rawliteral(<!DOCTYPE
HTML><html><head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>ESP Web Server</title>

<style>

    html {font-family: Arial; display: inline-block; text-align:
center;}

    h2 {font-size: 2.3rem; }

    p {font-size: 1.9rem; }

    body {max-width: 400px; margin:0px auto; padding-bottom: 25px; }

    .slider { -webkit-appearance: none; margin: 14px; width: 360px;
height: 25px; background: #FFD65C;

outline: none; -webkit-transition: .2s; transition: opacity .2s; }

    .slider::-webkit-slider-thumb { -webkit-appearance: none;
appearance: none; width: 35px; height: 35px; background: #003249; cursor:
pointer; }

    .slider::-moz-range-thumb { width: 35px; height: 35px; background:
#003249; cursor: pointer; }

</style></head><body>

<h2>ESP Web Server</h2>

<p><span id="textSliderValue">%SLIDERVALUE%</span></p>

<p><input type="range" onchange="updateSliderPWM(this)"
id="pwmSlider" min="0" max="1023" value="%SLIDERVALUE%" step="1"
class="slider"></p><script>

function updateSliderPWM(element) {
```

```
var sliderValue = document.getElementById("pwmSlider").value;

document.getElementById("textSliderValue").innerHTML = sliderValue;

console.log(sliderValue);

var xhr = new XMLHttpRequest();

xhr.open("GET", "/slider?value="+sliderValue, true);

xhr.send(); }</script></body></html>)rawliteral";

// Replace placeholders in web pages with button sections

String processor(const String& var){

//Serial.println(var);

if (var == "SLIDERVALUE"){

return sliderValue;

}

return String();}

void setup(){

// Serial port for debugging

Serial.begin(115200);

analogWrite(output, sliderValue.toInt());

// Connect to Wi-Fi

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi..");

}

}
```

```
// ESP local IP address is displayed"

Serial.println(WiFi.localIP());

// Route for root / 网页

server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){

    request->send_P(200, "text/html", index_html, processor);

});

// Send a GET request to<ESP_IP>/slider? value= <inputMessage >
server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest *request) {

    String inputMessage;

// In<ESP_IP>Get input1 value on /slider? value= <inputMessage >

    if (request->hasParam(PARAM_INPUT)) {

        inputMessage = request->getParam(PARAM_INPUT)->value();

        sliderValue = inputMessage;

        analogWrite(output, sliderValue.toInt());

    }

    else {

        inputMessage = "No message sent";

    }

    Serial.println(inputMessage);

    request->send(200, "text/plain", "OK");

});

// Start server
```

```
server.begin();}

void loop() {

}
```

25.2 Code working principle:

First, import the required libraries. This ESP8266WiFi, ESPAsyncWebServer and ESPAsyncTCP are required to build Web servers.

```
#include <ESP8266WiFi.h>

#include <ESPAsyncTCP.h>

#include <ESPAsyncWebServer.h>
```

Set network credentials:

Insert your network credentials in the following variables so that ESP8266 can connect to your local network.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";

const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Variable definition:

We will control the brightness of the ESP8266's built-in LED. 2. Save the GPIO we want to control in the output variable.

```
const int output = 2;
```

This slider value variable will hold the slider value. At the beginning, it is set to zero.

```
String sliderValue = "0";
```

Input parameters:

This parameter inputs that the variable will be used to "search" for the slider value in the request received by ESP8266 when the slider moves.

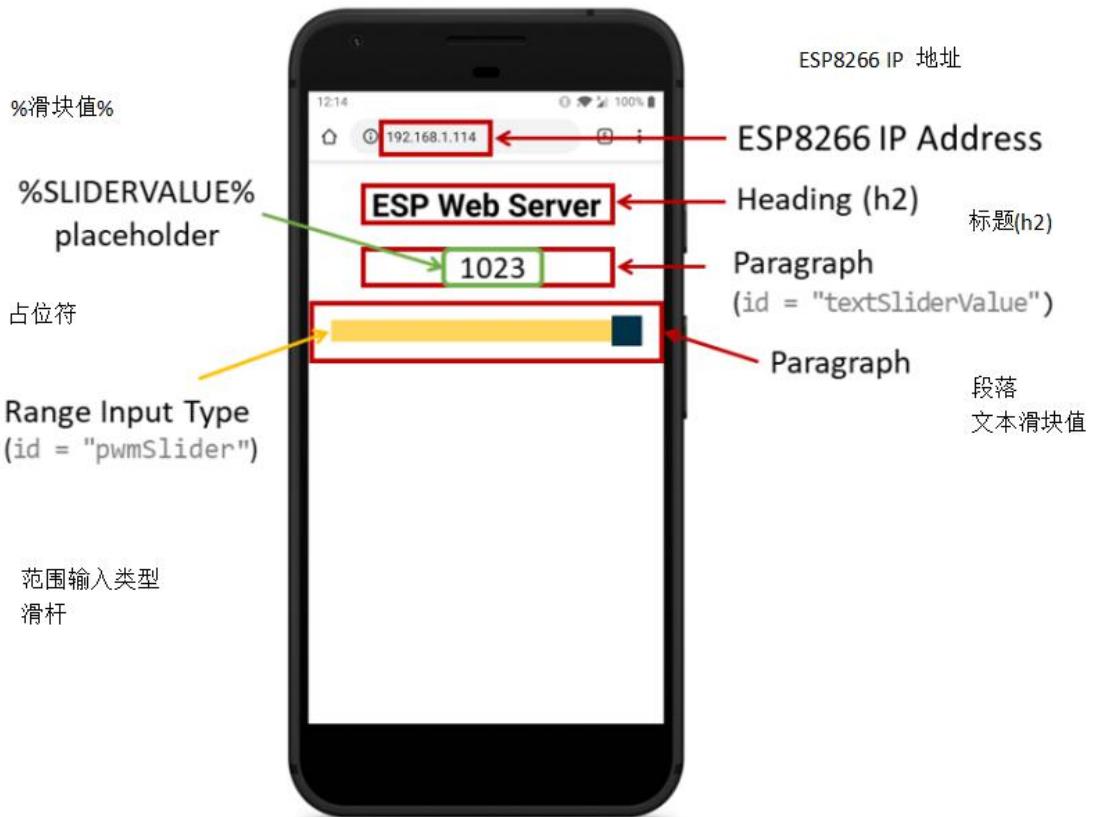
(Remember: ESP8266 will receive a request like GET/sliders? value=SLIDERVALUE)

```
const char* PARAM_INPUT = "value";
```

It searches for value on the URL and gets the value assigned to it.

25.3 Build a web page

Now let's go to the Web server page.



The web page for this project is very simple. It contains a heading, a paragraph, and a type range of input.

Let's look at how the web page is created.

All HTML text containing styles is stored in the index_HTML variable. Now we'll go through the HTML text and see what each section does.

The following <meta> Tags make your web page responsive in any browser.

```
<meta name="viewport" content="width=device-width,  
initial-scale=1">
```

Between<title> </TITLE>Tags enter the name of our Web server. A title is text that appears on a Web browser TAB.

Style

In<style></style>Between the tabs, we added some CSS to set the style of the page.

```
<style> html {font-family: Arial; display: inline-block;  
text-align: center;} h2 {font-size: 2.3rem;} p {font-size:  
1.9rem;} body {max-width: 400px; margin:0px auto;  
padding-bottom: 25px;} .slider { -webkit-appearance: none;  
margin: 14px; width: 360px; height: 25px; background:  
#FFD65C; outline: none; -webkit-transition: .2s;  
transition: opacity .2s;} .slider::-webkit-slider-thumb  
{-webkit-appearance: none; appearance: none; width: 35px;  
height: 35px; background: #003249; cursor:  
pointer;} .slider::-moz-range-thumb { width: 35px; height:  
35px; background: #003249; cursor: pointer; } </style>
```

Basically, we set up the HTML page to display Arial text in a block with no margins, aligned in the center.

```
html {font-family: Arial; display: inline-block;  
text-align: center;}
```

The lines of the face set the font size for the heading (h2) and paragraph (p).

```
h2 {font-size: 2.3rem;}
```

```
p {font-size: 1.9rem;}
```

Sets the HTML body properties.

```
body {max-width: 400px; margin: 0px auto; padding-bottom:  
25px;}
```

The following row custom slider:

```
.slider { -webkit-appearance: none; margin: 14px; width:  
360px; height: 25px; background: #FFD65C; outline: none;  
-webkit-transition: opacity .2s; transition:  
opacity .2s; } .slider::-webkit-slider-thumb  
{-webkit-appearance: none; appearance: none; width: 35px;  
height: 35px; background: #003249; cursor:  
pointer;} .slider::-moz-range-thumb { width: 35px; height:  
35px; background: #003249; cursor: pointer; }
```

HTML text

The inside of the<body></ body>Tags are the content we add to the page.

The<h2></ h2>Add TAB titles to web pages. In this case, "ESP Web Server" text, but you can add any other text.

<h2>ESP Web Server</h2>

The first segment will contain the current slider value. That particular HTML tag has the ID textSliderValue assigned to it so that we can reference it later.

```
<p><span>
```

```
id="textSliderValue">%SLIDERVALUE%</span></p>
```

The % slider value % is a placeholder for the slider value. When ESP8266 sends it to the browser, it will be replaced with the actual value. This is useful for displaying the current value when you first visit the browser.

Create the slider

To create sliders in HTML, use < input>. The label. Described & lt; Input & gt; The tag specifies a field in which the user can enter data.

There are multiple input types. To define the slider, use the Type attribute and the range value. In the slider, you also need to define the minimum and maximum ranges (in this case, 0 and 1023, respectively) using the "min" and "Max" attributes.

```
<p><input
```

```
type="range" onchange="updateSliderPWM(this)"  
id="pwmSlider" min="0" max="1023" value="%SLIDERVALUE%"  
step="1" class="slider"></p>
```

You also need to define other properties, such as:

Specify the interval between valid numbers in the step property. In our case, it is set to 1;

Style slider in class (class = "slider");

The ID used to update the current location displayed on the web page;

The onchange property of the calling function (updateSliderPWM(this)) sends an HTTP request to ESP8266 as the slider moves. The this keyword refers to the current value of the slider.

```
<script>  function  updateSliderPWM(element)  {  var
sliderValue = document.getElementById("pwmSlider").value;
document.getElementById("textSliderValue").innerHTML =
sliderValue;  console.log(sliderValue);  var  xhr  =  new
XMLHttpRequest();  xhr.open("GET",
"/slider?value="+sliderValue,  true);  xhr.send();  }
</script>
```

The next line gets the current slider value by its ID and saves it in the slider value JavaScript variable. Previously, we assigned the id of the slider to the PWM slider. So we get it as follows:

```
Var sliderValue= document.getElementById("pwmSlider").value;
```

After that, we set the slider label (whose ID is the text slider value) to the variable saved in the slider value.

Finally, an HTTP GET request is issued.

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "/slider?value="+sliderValue, true);
xhr.send();
```

For example, when the slider is at 0, you make an HTTP GET request to the following URL:

```
http://ESP-IP-ADDRESS/slider?value=0
```

When the slider value is 200, you will receive the request on the concern URL.

```
http://ESP-IP-ADDRESS/slider?value=200
```

This way, when ESP8266 receives a GET request, it can retrieve the value parameter in the URL and control the PWM signal accordingly, as we'll see in the next section:

Processor:

Now we need to create the processor () function that will replace the placeholders in our HTML text with the current slider value when you first access it in your browser.

```
// Replaces placeholder with button section in your web  
page      String      processor(const      String&  
var){      //Serial.println(var);      if      (var      ==  
"SLIDERVALUE"){ return sliderValue; } return String(); }
```

When requesting a web page, we check for any placeholders in the HTML. If it finds the %SLIDERVALUE% placeholder, we will return the variable saved in the SLIDERVALUE.

setup()

Under Setup (), initialize the serial monitor for debugging.

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

Set the duty cycle of the PWM signal to save at the slider value (it is set to 0 when ESP8266 starts).

```
analogWrite(output, sliderValue.toInt());
```

Connect to your local network and print the ESP8266 IP address.

```
// Connect to Wi-Fi WiFi.begin(ssid, password); while  
(WiFi.status() != WL_CONNECTED) { delay(1000);  
Serial.println("Connecting to WiFi.."); } // Print ESP  
Local IP Address Serial.println(WiFi.localIP());
```

Processing requests:

Finally, add the next few lines of code to handle the Web server.

```
// Route for root / web page

server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){ request->send_P(200, "text/html", index_html, processor); });

//Send a GET request to <ESP_IP>/slider?value=<inputMessage>

server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest *request) { String inputMessage; // GET input1 value on
<ESP_IP>/slider?value=<inputMessage> if
(request->hasParam(PARAM_INPUT)) { inputMessage =
request->getParam(PARAM_INPUT)->value(); sliderValue =
inputMessage; ledcWrite(ledChannel,
sliderValue.toInt()); } else { inputMessage = "No message
sent"; } Serial.println(inputMessage); request->send(200,
"text/plain", "OK"); });
}
```

When we make a request for the root URL, we send the HTML text stored in the index_HTML variable. We also need to pass in the Processor () function, which will replace all placeholders with the correct values.

```
// Route for root / web page server.on("/", HTTP_GET,
[](AsyncWebServerRequest *request){ request->send_P(200,
"text/html", index_html, processor); });
}
```

We need another handler that will save the value of the current slider and set the corresponding LED brightness.

```
server.on("/slider", HTTP_GET, []  
(AsyncWebRequest *request) { String inputMessage; //  
GET input1 value on <ESP_IP>/slider?value=<inputMessage>  
if (request->hasParam(PARAM_INPUT)) { inputMessage =  
request->getParam(PARAM_INPUT)->value(); sliderValue =  
inputMessage; ledcWrite(ledChannel,  
sliderValue.toInt()); } else { inputMessage = "No message  
sent"; } Serial.println(inputMessage); request->send(200,  
"text/plain", "OK"); } );
```

Basically, we get the slider value in the following lines:

```
if (request->hasParam(PARAM_INPUT)) { inputMessage =  
request->getParam(PARAM_INPUT)->value(); sliderValue =  
inputMessage;
```

Then, update the LED brightness (PWM duty cycle) with the following command. LedcWrite () accepts the channel and value you want to control as a function of parameters.

```
ledcWrite(ledChannel, sliderValue.toInt());
```

Finally, start the server.

```
server.begin();
```

Since this is an asynchronous Web server, we don't need to write anything in loop().

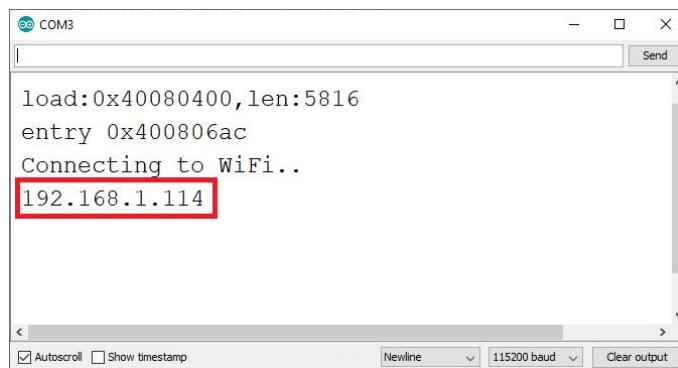
```
void loop(){ }
```

That's how the code works.

Upload code:

Now, upload the code to your ESP8266. Make sure you select the correct circuit board and COM port.

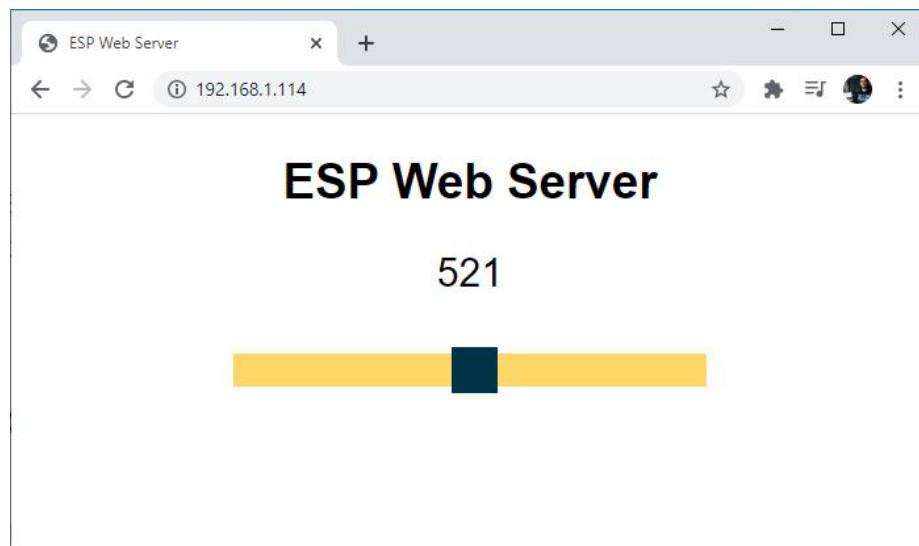
After uploading, turn on the serial port monitor at 115200 baud rate and press the ESP8266 reset button. The ESP8266 IP address should be printed on the serial monitor.



A screenshot of a Windows-style serial monitor window titled "COM3". The window shows the following text:
load:0x40080400,len:5816
entry 0x400806ac
Connecting to WiFi..
192.168.1.114

25.4 Web server Demo

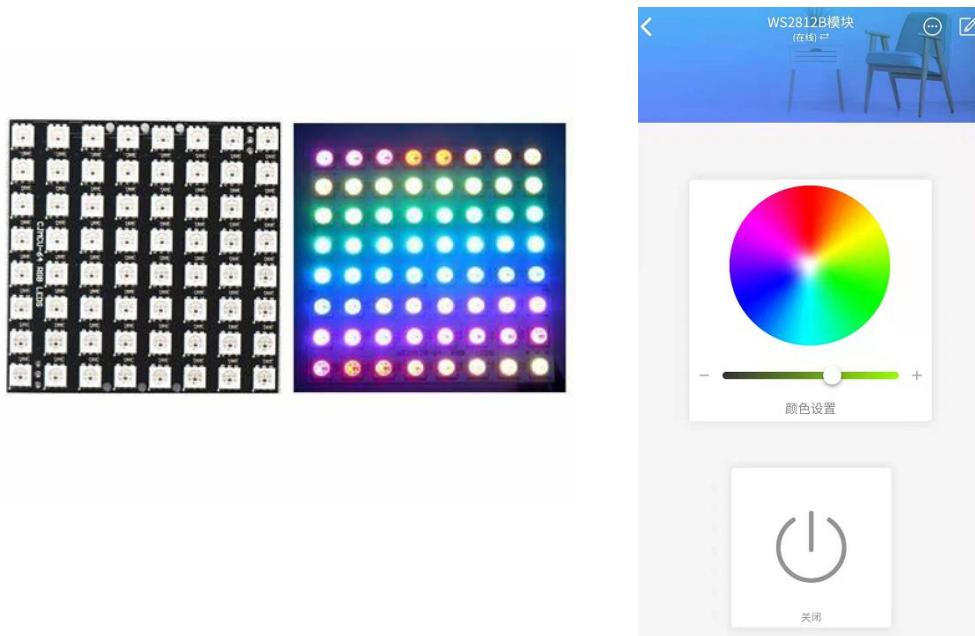
Open a browser and enter the ESP8266 IP address. Your Web server should display the slider and its current value.



Move the slider to see the brightness of the ESP8266's built-in LEDs increase and decrease.

Lesson 26 ESP8266 controls WS2812 lights via Blinker

This tutorial shows how Nodemcu ESP8266 can control RGB color changes via Blinker. You'll learn how to use Blinker APP to control it from around the world via the Internet of Things; For example, you can also control servo motors or DC motors instead of just RGB.



26.1 Arduino configuration

Install the blinker Arduino

- 1、Open the tutorial library files to find blinker library、Adafruit_NeoPixel library
unzip to my computer > Document > Arduino > libraries folder;

文档 > Arduino > libraries			
名称	修改日期	类型	大小
Attiny84_IO	2021-04-14 12:59	文件夹	
BalanceCar	2020-08-10 13:44	文件夹	
BatReader	2020-12-12 10:19	文件夹	
blinker	2021-10-07 11:48	文件夹	

2、Add a device to the App and obtain the Secret Key

1. Enter the App, click "+" in the upper right corner, and then select Add device;
2. Click **Arduino** > **WiFi access**;
3. Select a service provider to access;
4. Copy the requested file **Secret Key**;

(Note: If you haven't used Blinker before, you'll need to sign up for an account with your mobile number.)



26.2 Working Code:

(Note: the value of `auth[]` in the source code is the **Secret Key** obtained in the App. Other configurations can be set according to their own conditions.)

```
#define BLINKER_PRINT Serial  
  
#define BLINKER_MIOT_LIGHT  
  
#define BLINKER_WIFI  
  
#include <Blinker.h>  
  
#include <Adafruit_NeoPixel.h>  
  
char auth[] = "*****"; //****Enter the secret key you obtained  
in Blinker****/  
  
#define PIN 15 // DIN PIN (GPIO15, D8)  
  
#define NUMPIXELS 60 // Defines the number of RGB to be lit  
  
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB  
+ NEO_KHZ800);  
  
// Creating a Component Object  
  
BlinkerRGB RGB1("RGB");  
  
int LED_R=0,LED_G=0,LED_B=0,LED_Bright=180; // Define RGB and  
brightness  
  
bool WIFI_Status = true;  
  
void smartConfig() //Configuration network function  
  
{
```

```
WiFi.mode(WIFI_STA);

Serial.println("\r\nWait for Smartconfig...");

WiFi.beginSmartConfig();//Waiting for the user name and password
sent by the mobile phone

while (1)

{

    Serial.print(".");

    digitalWrite(LED_BUILTIN, HIGH);

    delay(1000);

    digitalWrite(LED_BUILTIN, LOW);

    delay(1000);

    if (WiFi.smartConfigDone())//Exit the waiting for

    {

        Serial.println("SmartConfig Success");

        Serial.printf("SSID:%s\r\n", WiFi.SSID().c_str());

        Serial.printf("PSW:%s\r\n", WiFi.psk().c_str());

        break;

    }

}

void WIFI_Set()//



{
```

```
//Serial.println("\r\n Are connected");

int count = 0;

while(WiFi.status()!=WL_CONNECTED)

{

    if(WIFI_Status)

    {

        Serial.print(".");

        digitalWrite(LED_BUILTIN, HIGH);

        delay(500);

        digitalWrite(LED_BUILTIN, LOW);

        delay(500);

        count++;

        if(count>=5)//5s

        {

            WIFI_Status = false;

            Serial.println("WiFi connection failed, please use

mobile phone to configure network");

        }

    }

else

{



    smartConfig(); //Wechat intelligent distribution network
```

```
    }

}

/* Serial.println("The connection is successful");

Serial.print("IP:");

Serial.println(WiFi.localIP());*/



}

void SET_RGB(int R,int G,int B,int bright)

{

    for (uint16_t i = 0; i < NUMPIXELS; i++) //Change the color of the

light strip

    {

        pixels.setPixelColor(i,R,G,B);

    }

    pixels.setBrightness(bright);//brightness

    pixels.show();      //Send display

}

//APP RGB Color setting callback

void rgb1_callback(uint8_t r_value, uint8_t g_value,

                  uint8_t b_value, uint8_t bright_value)

{
```

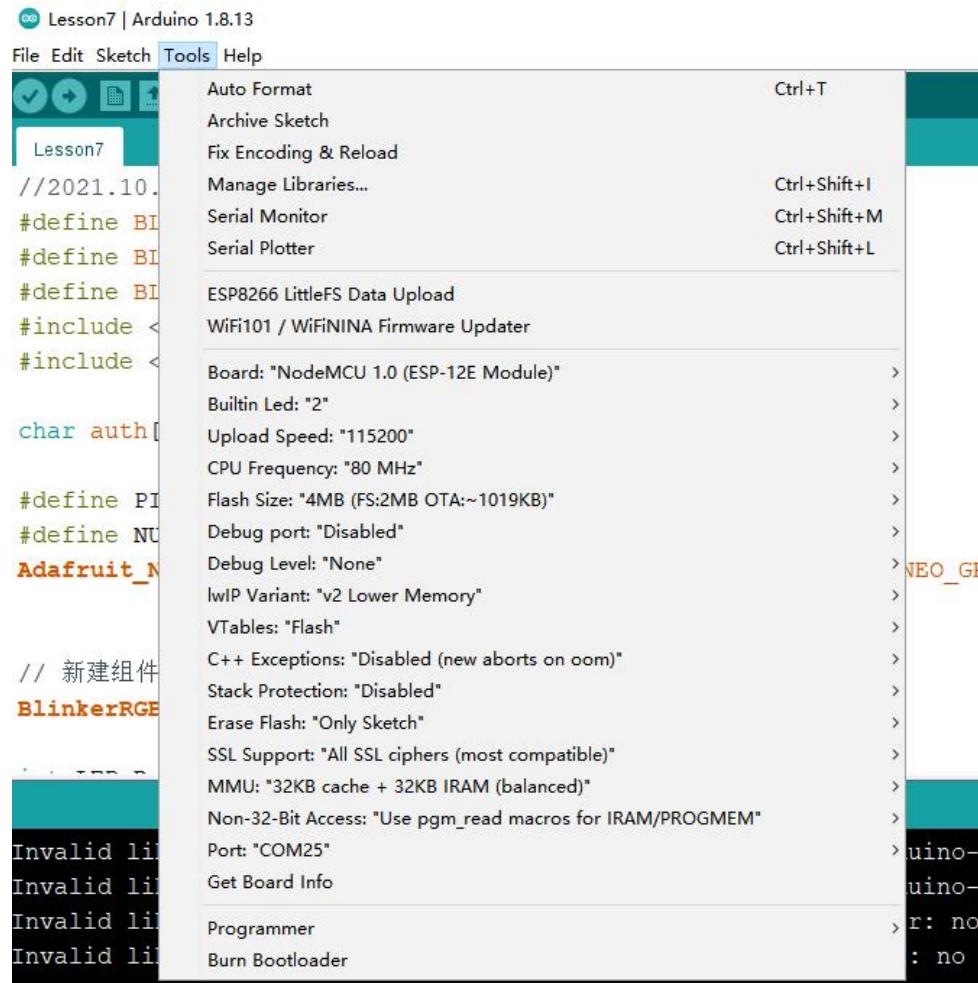
```
//digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));  
  
BLINKER_LOG("R value: ", r_value);  
  
BLINKER_LOG("G value: ", g_value);  
  
BLINKER_LOG("B value: ", b_value);  
  
BLINKER_LOG("Rrightness value: ", bright_value);  
  
LED_Bright = bright_value;  
  
SET_RGB(r_value,g_value,b_value,LED_Bright);  
  
}  
  
void setup() {  
  
    // Initializing the serial port  
  
    Serial.begin(115200);  
  
  
  
    pixels.begin(); //WS2812 initialization  
  
    pixels.show();  
  
    pinMode(LED_BUILTIN, OUTPUT);  
  
    #if defined(BLINKER_PRINT)  
  
        BLINKER_DEBUG.stream(BLINKER_PRINT);  
  
    #endif  
  
  
  
    WIFI_Set();  
  
    // blinker initialization  
  
    Blinker.begin(auth, WiFi.SSID().c_str(), WiFi.psk().c_str());
```

```
RGB1.attach(rgb1_callback); //Registers callback functions that
adjust colors
}
```

```
void loop() {
    Blinker.run();
}
```

The source code only to achieve monochrome display, more color or cool effect display please modify the source code.

Upload code:



26.3 App Connection Configuration

Connected devices

Open the Lighting App and click the sidebar button - > Developers - >Development tools - > EspTouch/SmartConfig, enter the WiFi password and click to start the configuration.



Enter WIFI password



- 1.确保设备已进入等待配置状态
- 2.输入WiFi密码，并点击开始配置

WiFi 名称 zhiyi1

WiFi 密码 · · · · · · · ·

 记住密码

开始配置

选择其他 WiFi

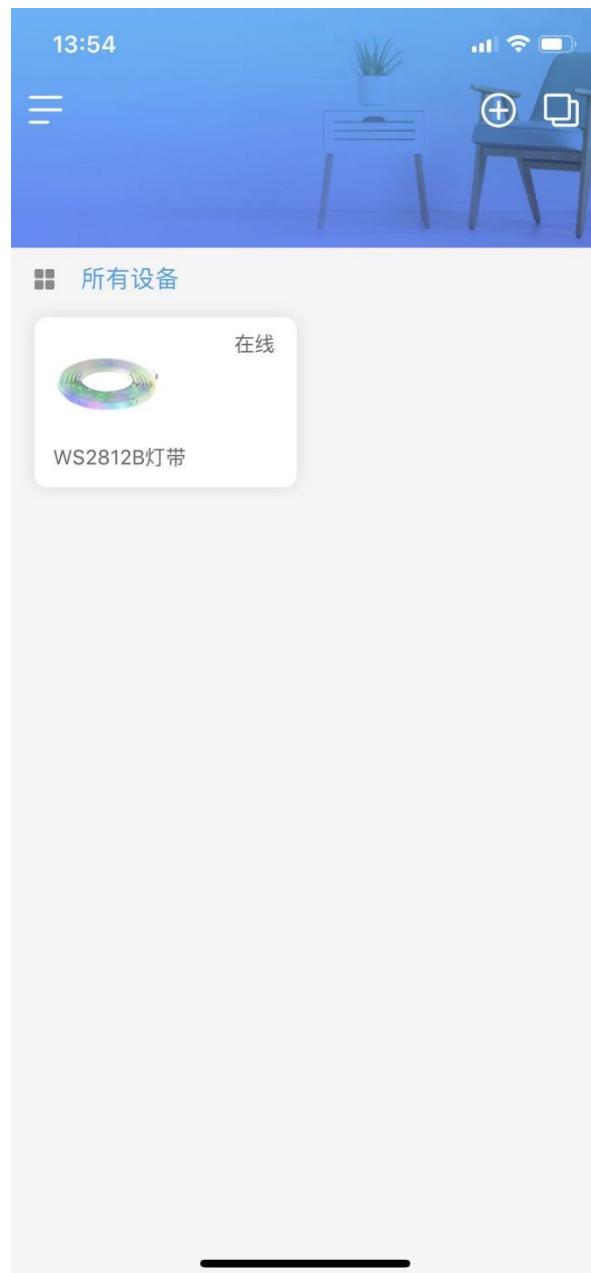
Configuration is successful



Device configuration

1.Return to the blinker App home page and you can see that the device is online.

Click the device to enter the configuration page.



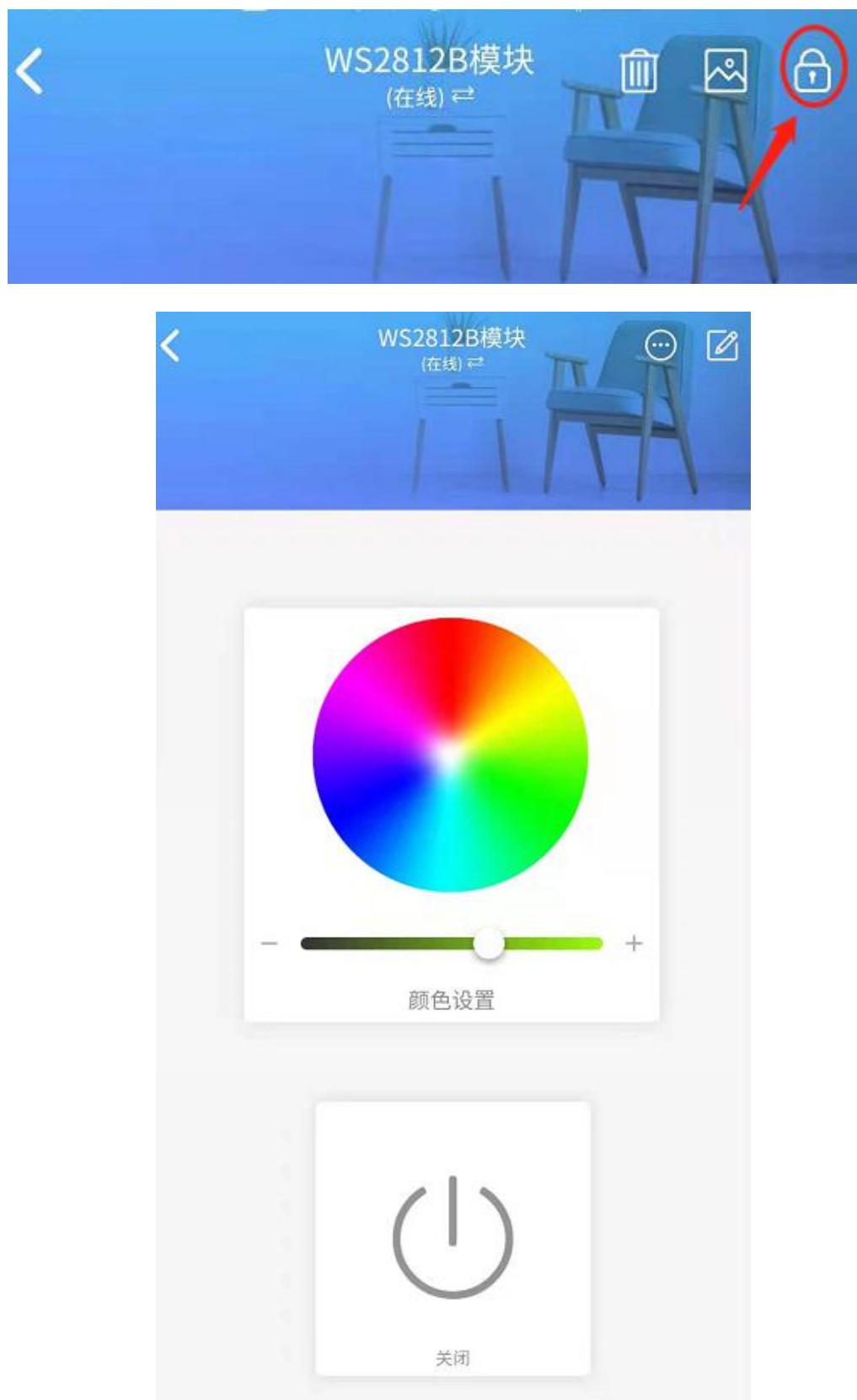
2. Click the device Edit button;



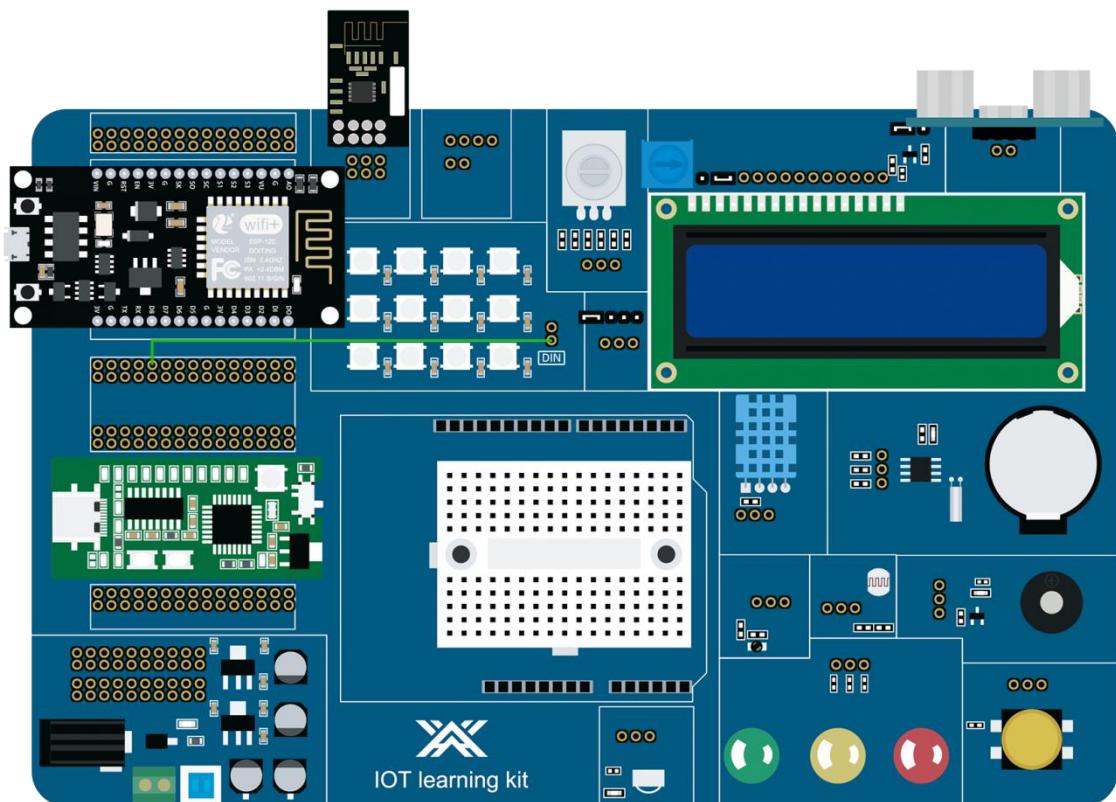
3. Click the Edit button to enter the component editing state. Click the color component at the bottom of the component list to add it to the interface, then click the new color component to enter the component editing interface, change the data key name to RGB, and click Save;



4. After editing, click the lock button in the upper right corner to finish editing, and then you can control the lamp belt.



26.4 The wiring diagram:

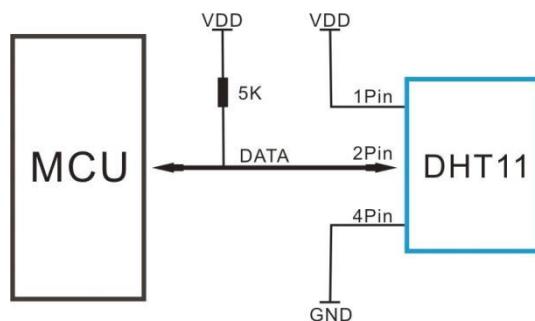
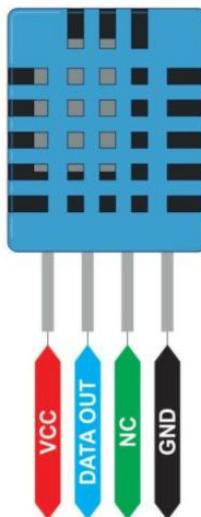


Lesson 27 ESP8266 Nodemcu displays temperature and humidity in combination with Blinker

In this course, we will use the DHT11 temperature and humidity sensor connected to ESP8266 Nodemcu to display the temperature and humidity in Blinker APP.

27.1 DHT11 Sensor:

The DHT11 sensor provides humidity and temperature data. It has the following pin interface.



典型应用电路

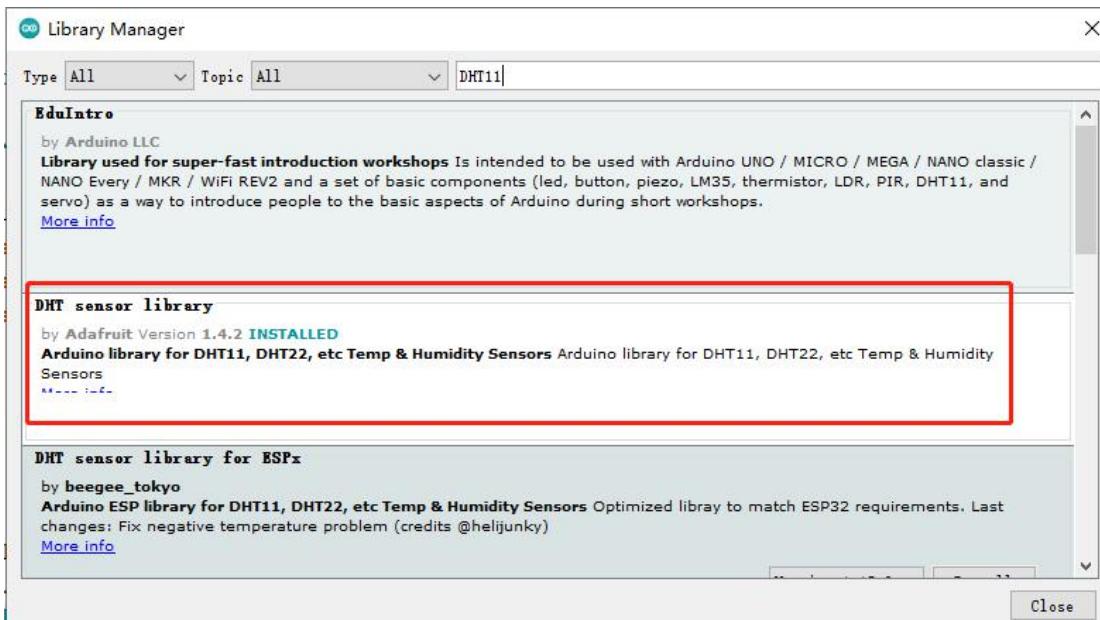
In the last few sections we learned to use the Arduino IDE to develop the Settings required for esp8266.

27.2 Install the library

As mentioned earlier, we assume that ESP8266 is programmed using an Arduino IDE. If you have not already configured it to support ESP8266 boards, review lesson 20.

Add DHT11 libraries to the Arduino IDE.

The library can be easily installed through the Arduino IDE library manager, as shown in Figure 3.



27.3 Working code explanation:

(If you have a Xiao Ai device, this code can also be used to check the temperature and humidity in your home.)

```
#define BLINKER_WIFI
#define BLINKER_MIOT_SENSOR //Xiao Ai defined it as sensor equipment
#include <Blinker.h>
#include <DHT.h>
char auth[] = "*****"; //Device obtained by lighting APP
char ssid[] = "*****"; //The name of the WiFi
```

```
char pswd[] = "*****"; //WiFi password
BlinkerNumber HUMI("humi"); //Define the humidity data key name
BlinkerNumber TEMP("temp"); //Define the temperature data key name
#define DHTPIN 2 //Define DHT11 module connection pin GPIO2 (D4)
#define DHTTYPE DHT11 // Use the DHT 11 temperature and humidity module
//#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
//#define DHTTYPE DHT21 // DHT 21 (AM2301)
DHT dht(DHTPIN, DHTTYPE); //定义 dht
float humi_read = 0, temp_read = 0;
void heartbeat()
{
    HUMI.print(humi_read); //Send humidity data back to BlinkerApp
    TEMP.print(temp_read); //Return temperature data to BlinkerApp
}
void miotQuery(int32_t queryCode) //Xiao Ai voice command feedback
{
    BLINKER_LOG("MIOT Query codes: ", queryCode);
    int humi_read_int=humi_read; //Remove humidity floating point
    BlinkerMIOT.humi(humi_read_int); //Xiao Ai receives humidity
    BlinkerMIOT.temp(temp_read); //Xiao Ai receives the temperature
    BlinkerMIOT.print();
}
void setup()
{
    Serial.begin(115200);
    BLINKER_DEBUG.stream(Serial);
    BLINKER_DEBUG.debugAll();
    Blinker.begin(auth, ssid, pswd);
    Blinker.attachHeartbeat(heartbeat);
    dht.begin();
}
```

```

BlinkerMIOT.attachQuery(miotQuery);

}

void loop()

{ Blinker.run();

float h = dht.readHumidity();

float t = dht.readTemperature();

if (isnan(h) || isnan(t))

{

BLINKER_LOG("Failed to read from DHT sensor!");

}

else

{ BLINKER_LOG("Humidity: ", h, "%"); //blinker APP Read display temperature

BLINKER_LOG("Temperature: ", t, " *C"); //blinker APP Read display humidity

humi_read = h;

temp_read = t;

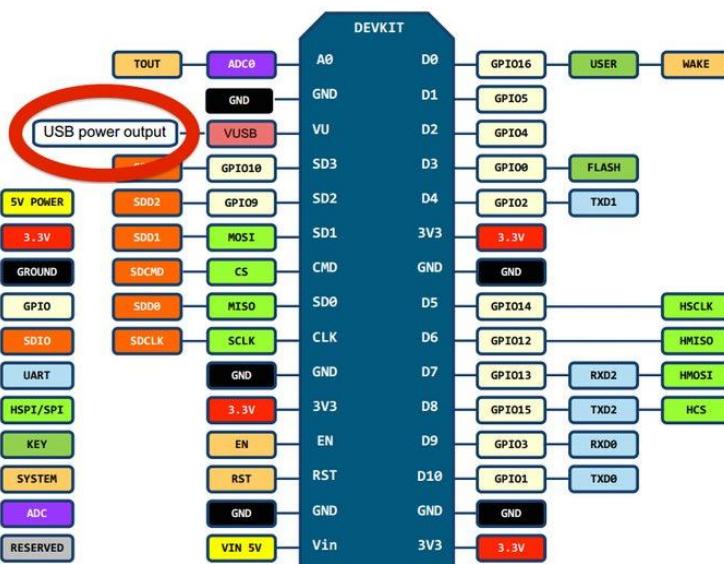
}

Blinker.delay(2000);

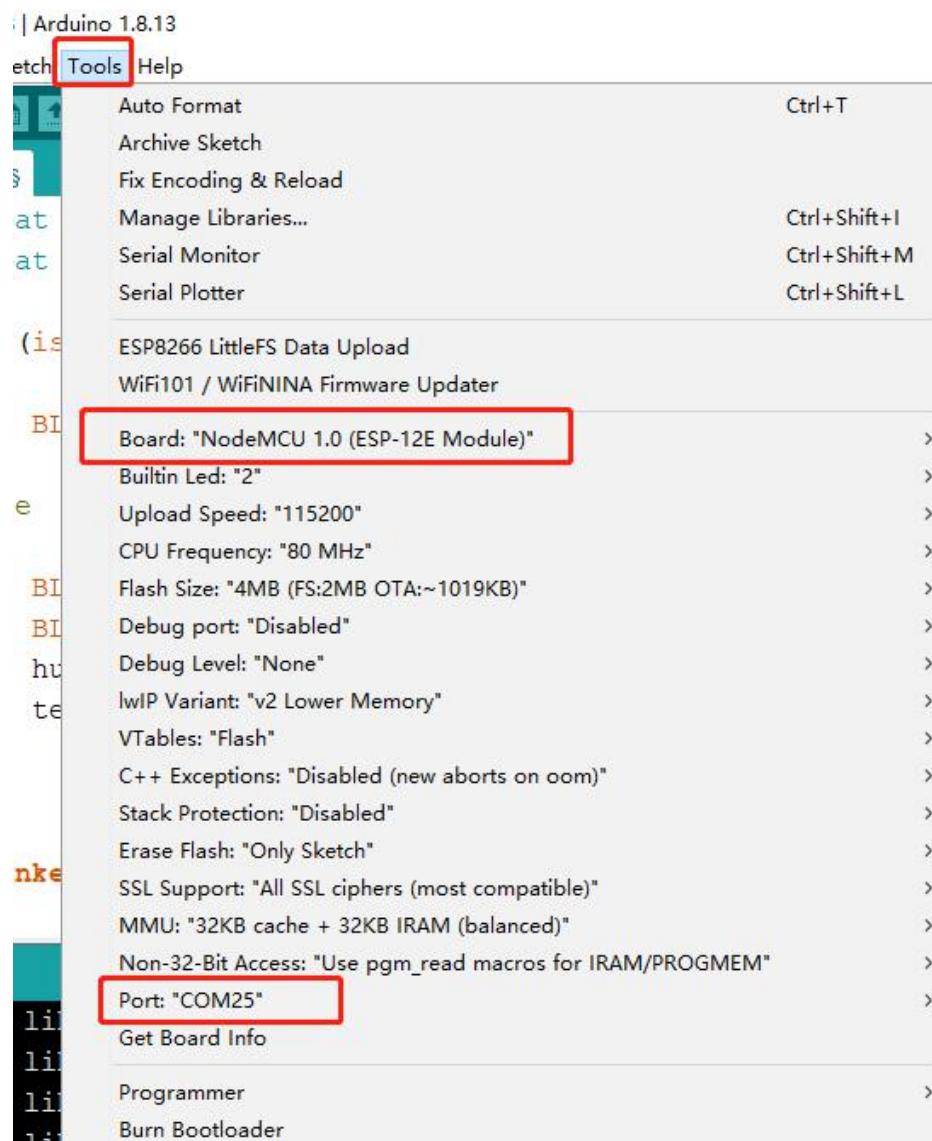
}

```

In the comments of the code we can easily find which pin is used to receive data with esp8266, but this refers to GPIO2, not D2 of our ESP8266d Nodemcu, but D4.

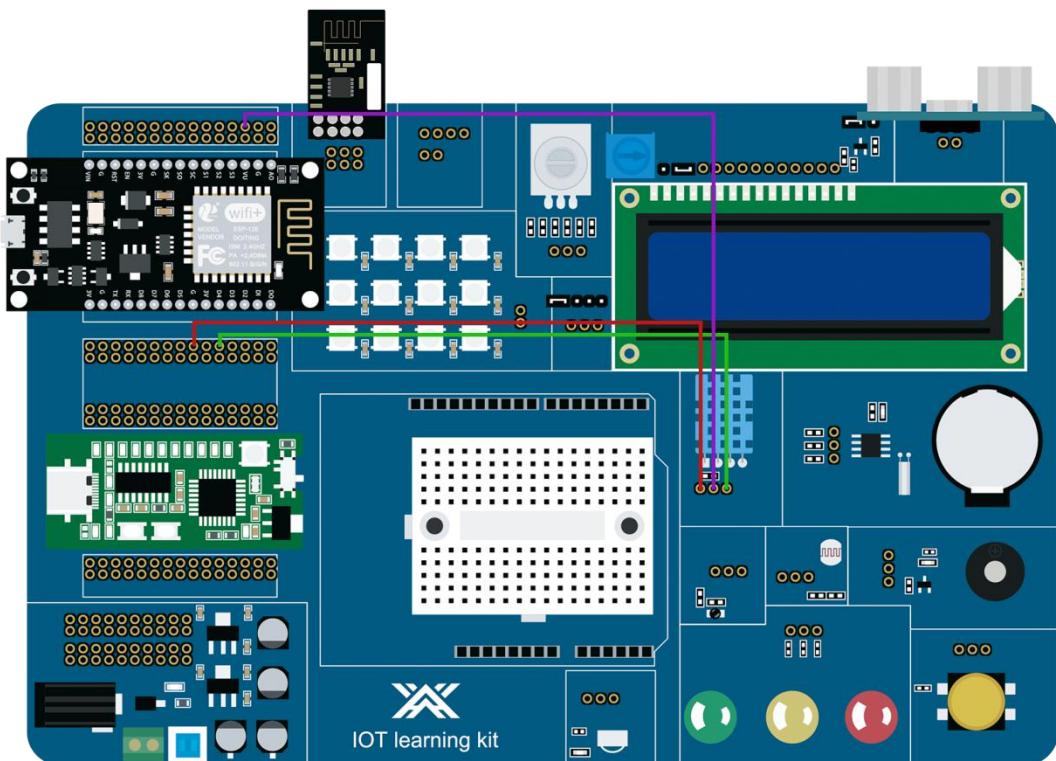


Open Arduino IDE to write code, compile and upload, and burn programs.



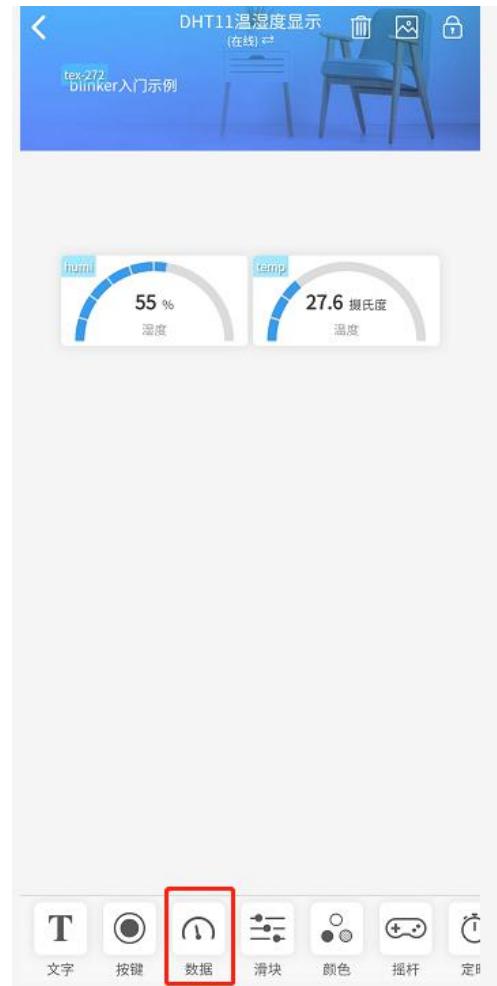
27.4 Wiring Diagram

Our DHT11 module has three pins: V, G and S.



After restarting the development board, we can see that the device is online in the lighting app. We can also set the component Settings in the lighting app and check the temperature and humidity in the lighting app:





For example, for our humidity data, the data key name is humi defined before the code, the display text is humidity, the unit is %, the maximum value is 100; The same goes for setting the temperature.



Finally, normal temperature and humidity are displayed, as shown in the figure:

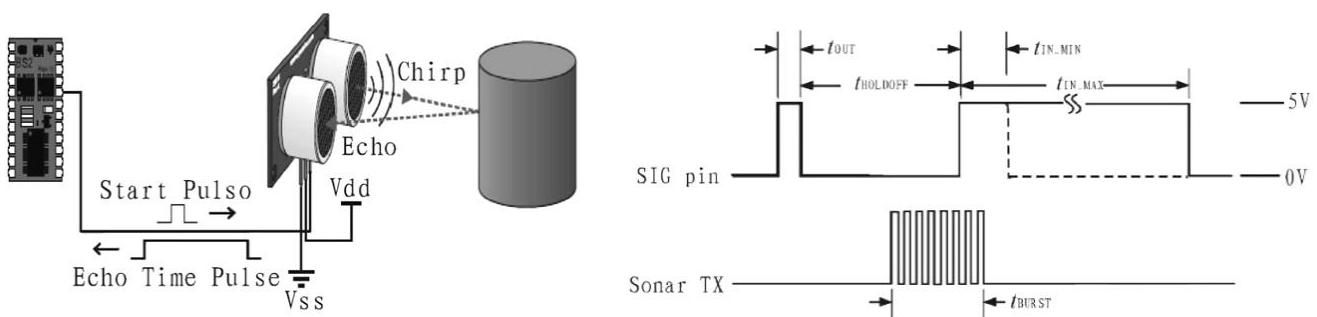


Lesson 28 ESP8266 Nodem combined with HC-SR04 Ultrasonic Ranging

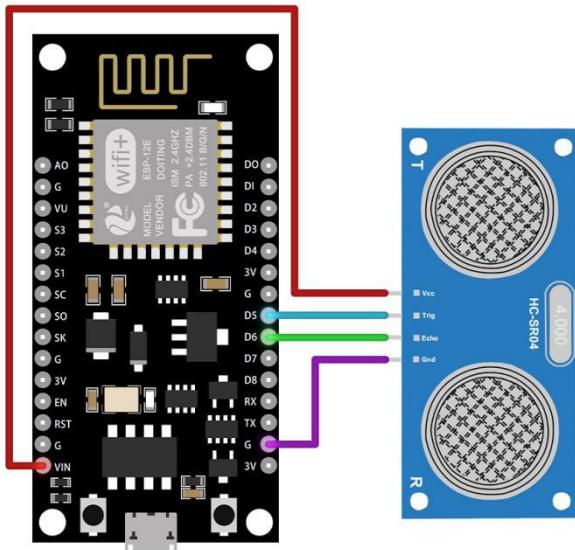
In this course, we will use ESP8266 Nodem combined with HC-SR04 ultrasonic sensor module to build ranging tools.

28.1 Ultrasonic transducer

The principle of ultrasonic ranging is the ultrasonic pulse emitted by the ultrasonic probe, transmitted to the surface of the object through the medium (air), reflected through the medium (air) to receive the probe, measured the ultrasonic pulse from the transmission to receive the required time, according to the speed of sound in the medium, obtained from the probe to the surface of the distance between the object. If the distance between the probe and the surface of the object is L, the propagation speed of ultrasonic in the air is V, and the propagation time from transmitting to receiving is T, then $L=vt/2$. Thus it can be seen that there is a definite functional relationship between the measured distance L and the propagation time. As long as the time T can be measured, the distance L can be calculated, and the value of L can be directly displayed on the monitor by software.



28.2 Wiring Diagram



28.3 Explanation of working Code:

The hC-SR04 ultrasonic sensor and ESP8266 were used to obtain the distance from the object

First, define the trigger and echo pins.

```
const int trigPin = 12;
const int echoPin = 14;
```

In this example, we use generic I/O 12 and generic I/O 14. You can also modify other GPIO pins as you like.

This sound velocity variable preserves the speed of sound in air at 20°C. We use values in cm/uS.

```
#define SOUND_SPEED 0.034
```

The CM_TO_INCH variable allows us to convert a distance in centimeters to inches.

```
#define CM_TO_INCH 0.393701
```

Then, initialize the following variables.

```
long duration;  
  
float distanceCm;  
  
float distanceInch;
```

The duration variable holds the propagation time of the ultrasonic wave (the time elapsed since the pulse wave was sent and received). The distanceCm and distanceInch, as the name implies, save distances to objects in centimeters and inches.

setup()

In setup (), initialize the serial communication at 115200 baud rate so that we can print the measurements on the serial monitor.

```
Serial.begin(115200); // Starts the serial communication
```

Define the trigger pin as the output -- the trigger pin emits ultrasonic waves. An echo pin is defined as an input-echo pin that receives reflected waves and sends a signal proportional to the travel time to ESP8266.

```
pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output  
  
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
```

loop()

In loop (), the following line produces a 10uS HIGH pulse on the trigger pin -- which means that the pin emits ultrasonic waves. Note that before sending the pulse, we give a short low pulse to ensure that you will get a clean high pulse.

```
// Clears the trigPin  
digitalWrite(trigPin, LOW);  
delayMicroseconds(2);  
  
// Sets the trigPin on HIGH state for 10 micro seconds  
digitalWrite(trigPin, HIGH);  
delayMicroseconds(10);  
  
digitalWrite(trigPin, LOW);
```

We use the `pulseIn()` function to obtain the propagation time of sound waves:

```
duration = pulseIn(echoPin, HIGH);
```

The `LOW()` function reads a HIGH or LOW pulse from a pin. It accepts pins and pulse states (HIGH or LOW) as parameters. It returns the pulse length in microseconds. The pulse length is equal to the time it takes to reach the object plus the time it takes to return.

Then, we simply calculate the distance to an object considering the speed of sound.

```
distanceCm = duration * SOUND_SPEED/2;
```

Convert distance to inches:

```
distanceInch = distanceCm * CM_TO_INCH;
```

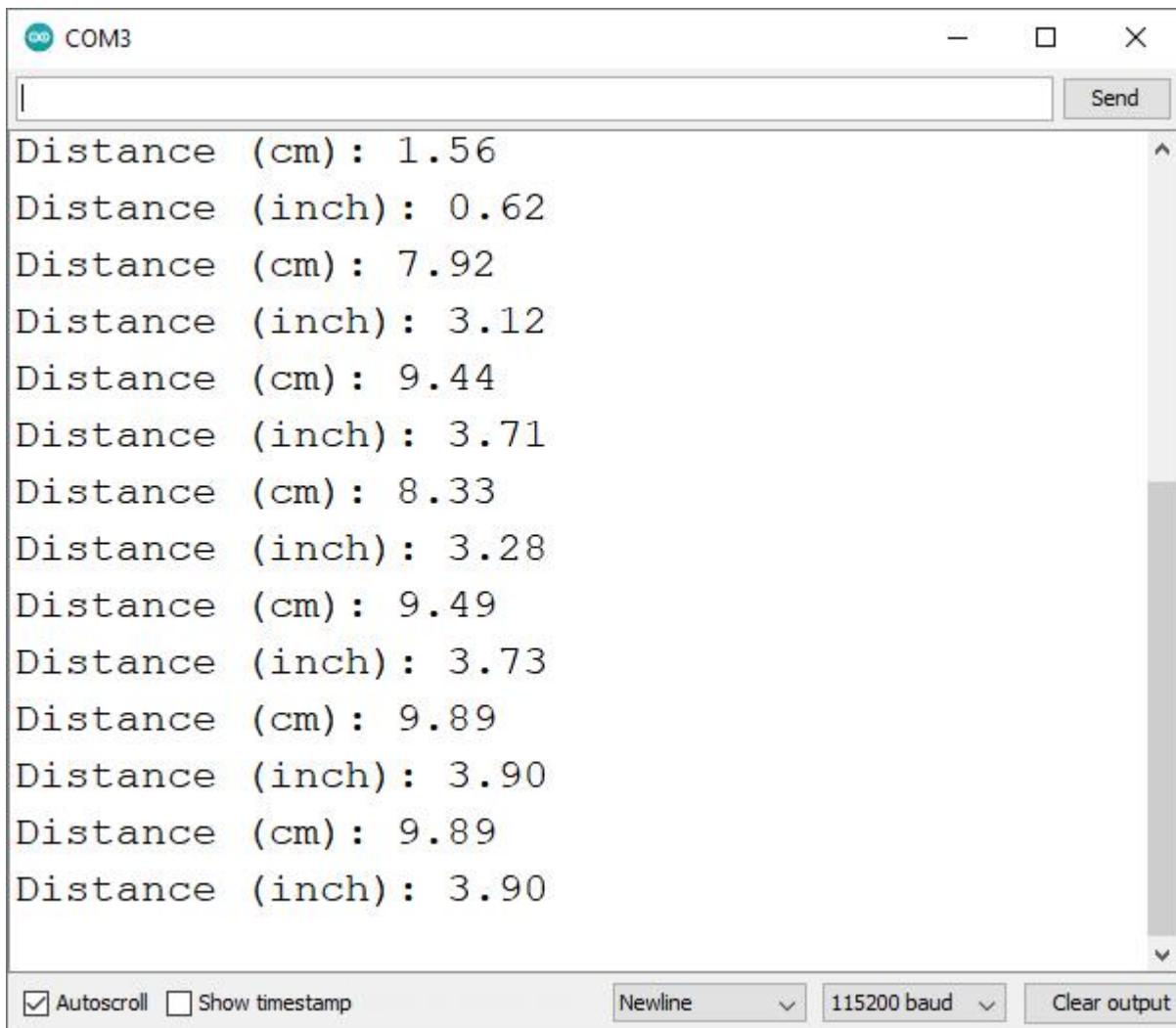
Finally, the results are printed on a serial monitor.

```
Serial.print("Distance (cm): ");  
  
Serial.println(distanceCm);  
  
Serial.print("Distance (inch): ");  
  
Serial.println(distanceInch);
```

Upload the code to your board. Don't forget to select the board you are using in the tool > The board. Also, don't forget to use Tools & GT; Select the correct

COM port from port.

After the upload, open the Serial Monitor at baud rate 115200. Press the on-board RST button to restart the board and it will begin printing the distance to the nearest object on the serial monitor. As shown in the figure below.



The screenshot shows the Arduino Serial Monitor window titled "COM3". The window displays a series of distance measurements printed by the code. The measurements alternate between centimeters (cm) and inches (inch). The text in the monitor is as follows:

```
Distance (cm) : 1.56
Distance (inch) : 0.62
Distance (cm) : 7.92
Distance (inch) : 3.12
Distance (cm) : 9.44
Distance (inch) : 3.71
Distance (cm) : 8.33
Distance (inch) : 3.28
Distance (cm) : 9.49
Distance (inch) : 3.73
Distance (cm) : 9.89
Distance (inch) : 3.90
Distance (cm) : 9.89
Distance (inch) : 3.90
```

At the bottom of the window, there are several configuration options: "Autoscroll" (checked), "Show timestamp" (unchecked), "Newline" dropdown, "115200 baud" dropdown, and "Clear output" button.