



## 物联网培训套件

20210904.V1

## 目录

第 1 课 安装 Arduino IDE.....	1
1.1 介绍.....	1
1.2 安装 Arduino.....	3
1.3 安装库.....	6
1.4 Arduino.....	8
1.5 RGB Nano 介绍.....	8
第 2 课 点亮 LED.....	11
2.1 概述.....	11
2.2 工作原理.....	11
2.3 连接描述.....	12
2.4 代码讲解.....	12
2.5 上传代码.....	13
第 3 课 按键控制 LED.....	15
3.1 概述.....	15
3.2 连接描述.....	15
3.3 代码讲解.....	15
第 4 课 有源蜂鸣器.....	17
4.1 概述.....	17
4.2 连接描述.....	17
4.3 代码讲解.....	18
第 5 课 无源蜂鸣器.....	19
5.1 概述.....	19
5.2 连接描述.....	19
5.3 代码讲解.....	20
第 6 课 交通灯.....	21
6.1 概述.....	21
6.2 工作原理.....	21
6.3 接线示意图.....	22

6.4 代码讲解.....	22
第 7 课流水灯.....	25
7.1 概述.....	25
7.2 工作原理.....	25
7.3 接线示意图.....	26
7.4 代码讲解.....	26
第 8 课 WS2812B.....	28
8.1 概述.....	28
8.2 工作原理.....	28
8.3 特征.....	29
8.4 接线示意图.....	30
8.5 代码讲解.....	30
第 9 课渐变 RGB.....	32
9.1 概述.....	32
9.2 工作原理.....	32
9.3 接线示意图.....	33
9.4 代码讲解.....	33
第 10 课 DS1307.....	36
10.1 概述.....	36
10.2 LCD1602 介绍.....	36
10.3 DS1307 介绍.....	37
10.4 接线示意图.....	38
10.5 代码讲解.....	38
第 11 课 显示温度.....	41
11.1 概述.....	41
11.2 热敏电阻简介.....	41
11.3 接线示意图.....	42
11.4 代码讲解.....	42
第 12 课显示温度和湿度.....	45

12.1 概述.....	45
12.2 接线示意图.....	45
12.3 代码讲解.....	45
第 13 课 超声波模块.....	48
13.1 概述.....	48
13.2 超声波传感器的介绍.....	48
13.3 接线示意图.....	49
13.4 代码讲解.....	49
第 14 课 光敏电阻.....	51
14.1 概述.....	51
14.2 组件的介绍.....	51
14.3 连接图.....	53
14.4 接线示意图.....	54
14.5 代码讲解.....	54
第 15 课 旋转编码器控制 RGB.....	56
15.1 概述.....	56
15.2 接线示意图.....	56
15.3 代码讲解.....	56
第 16 课 NRF24L01 无线接收发射模块.....	58
16.1 概述.....	58
16.2 接线示意图.....	58
16.3 代码讲解.....	58
第 17 课 LED 红外控制 LED.....	61
17.1 概述.....	61
17.2 接线示意图.....	61
17.3 代码讲解.....	62
第 18 课 红外控制 RGB.....	64
18.1 概述.....	64
18.2 接线示意图.....	64



18.3 代码讲解.....	64
第 19 课 蓝牙控制 RGB.....	67
19.1 概述.....	67
19.2 连接描述.....	67
19.3 代码讲解.....	67
19.4 蓝牙遥控.....	69
第 20 课 ESP8266 开发板.....	71
20.1 简介: .....	71
20.2 ESP8266 规格.....	71
20.3 ESP8266 版本.....	72
20.4 NodeMCU 引脚排列外设.....	73
20.5 NODEMCU ESP8266 中使用的引脚是什么? .....	73
第 21 课 Arduino IDE 中安装 ESP8266 开发板.....	75
21.1 在 Arduino IDE 中安装 ESP8266 插件.....	75
21.2 测试安装.....	78
21.3 接线图: .....	80
第 22 课 ESP8266 NodeMCU WiFi 控制红绿灯模块.....	81
22.1 异步网络服务器.....	81
22.2 示意图: .....	81
22.3 接线图: .....	82
22.4 ESP 异步 Web 服务器的代码.....	85
22.5 代码的工作原理.....	90
第 23 课 ESP8266 Node MCU 按键控制 LED.....	102
23.1 ESP8266 NodeMCU 控制数字输出.....	102
23.2 项目示例.....	102
23.3 接线示意图:.....	103
23.4 工作代码: .....	103
23.5 代码工作原理.....	104
23.6 上传代码.....	106

---

23.7 实物图: .....	106
第 24 课 ESP8266 控制 LED 亮度 (PWM) .....	107
24.1 ESP8266 NodeMCU PWM (脉宽调制) .....	107
24.2 示意图:.....	108
24.3 上传代码: .....	110
24.4 接线图: .....	111
第 25 课 ESP8266 Node MCU Web 服务器 控制 LED 亮度 (PWM).....	112
25.1 工作代码: .....	113
25.2 代码工作原理: .....	117
25.3 构建网页.....	118
25.4 网络服务器演示.....	125
第 26 课 ESP8266 通过 blinker 控制 WS2812 灯 .....	127
26.1 Arduino 配置.....	127
26.2 工作代码: .....	129
26.3 App 配置连接.....	135
26.4 接线图: .....	140
第 27 课 ESP8266 Nodemcu 结合 Blinker 显示温湿度.....	141
27.1 DHT11 传感器: .....	141
27.2 安装库.....	142
27.3 工作代码讲解: .....	142
27.4 接线图: .....	145
第 28 课 ESP8266 Nodem 结合 HC-SR04 超声波测距.....	149
28.1 超声波传感器.....	149
28.2 接线图: .....	150
28.3 工作代码讲解: .....	150

# 第 1 课 安装 Arduino IDE

## 1.1 介绍

Arduino 集成开发环境(IDE)是 Arduino 平台的软件端。

在本节课中，您将学习如何设置您的计算机使用 Arduino，以及如何设置有关的课程。

用于对 Arduino 进行编程的 Arduino 软件适用于 Windows、Mac 和 Linux。这三种平台的安装过程是不同的，安装软件需要您根据教程自己动手进行安装。

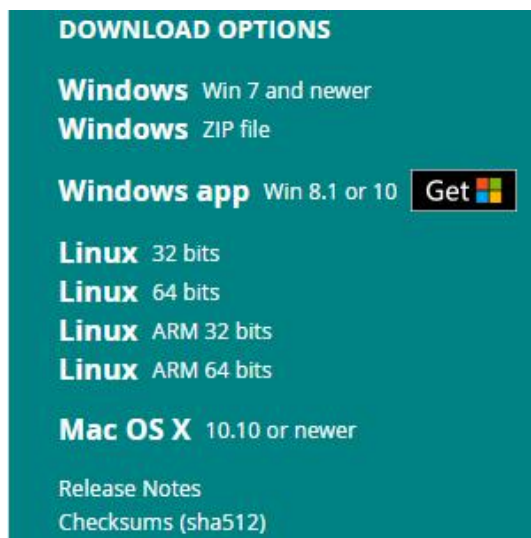
第一步：登录 <https://www.arduino.cc/en/software>。



The screenshot shows the Arduino IDE 1.8.13 download page. On the left, there's a section with the Arduino logo and text describing the IDE as open-source software for writing code and uploading it to the board. It mentions that the software can be used with any Arduino board and refers to the 'Getting Started' page for installation instructions. Below this, it says 'SOURCE CODE' and mentions that active development is hosted by GitHub, with instructions for building the code and links to the latest release source code archives. On the right, there's a teal sidebar titled 'DOWNLOAD OPTIONS' listing download links for Windows (Win 7 and newer, ZIP file), Windows app (Win 8.1 or 10, with a 'Get' button), Linux (32 bits, 64 bits, ARM 32 bits, ARM 64 bits), and Mac OS X (10.10 or newer). At the bottom of the sidebar are links for 'Release Notes' and 'Checksums (sha512)'.

该网站提供的版本通常是最新版本，实际版本可能比图片中的版本更新。

第二步：下载与您的计算机操作系统兼容的开发软件。此处以 Windows 为例。



单击 **Window** 下载软件。

## 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.](#)

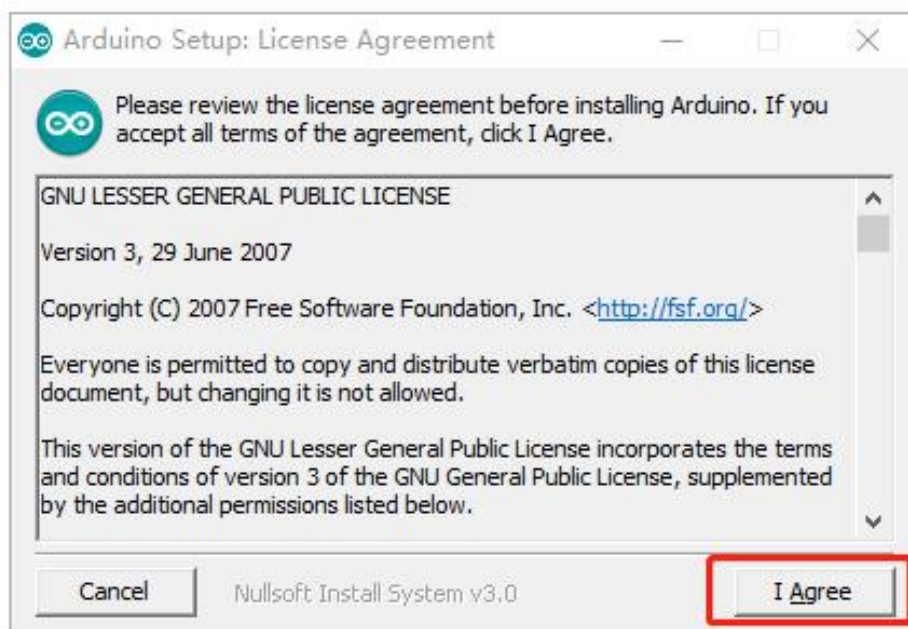


点击下载（**JUST DOWNLOAD**）。

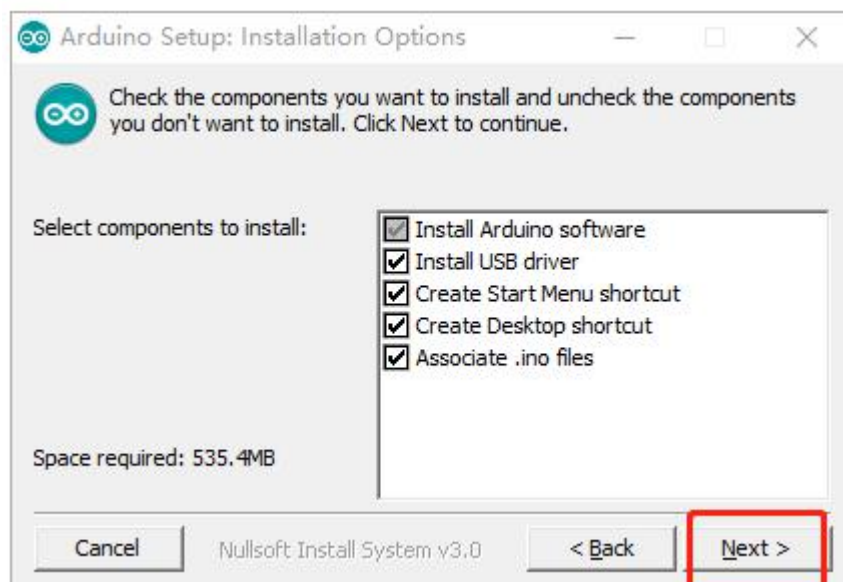
此外，1.8.9 版本可以在我们提供的材料中找到，我们的材料版本是本课程制作时的最新版本。

## 1.2 安装 Arduino

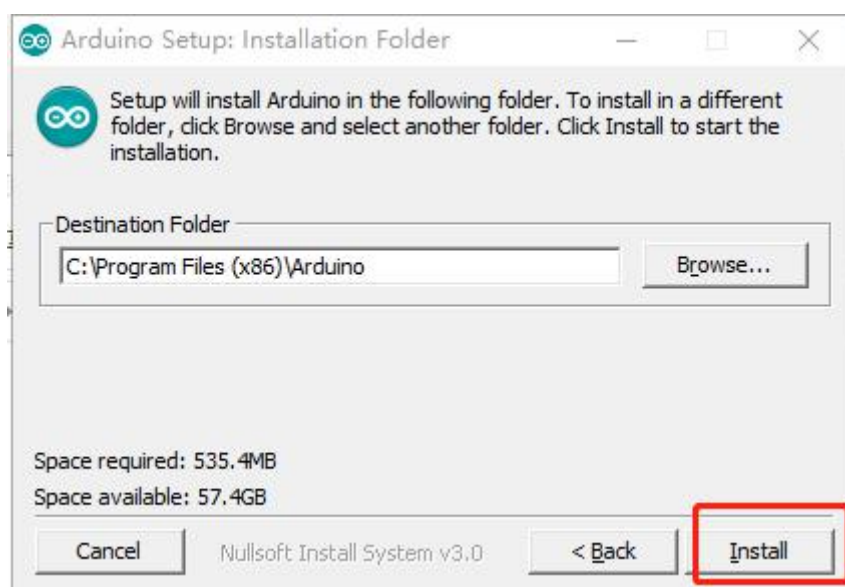
 arduino-1.8.13-windows



单击 I Agree 进行下一步操作。

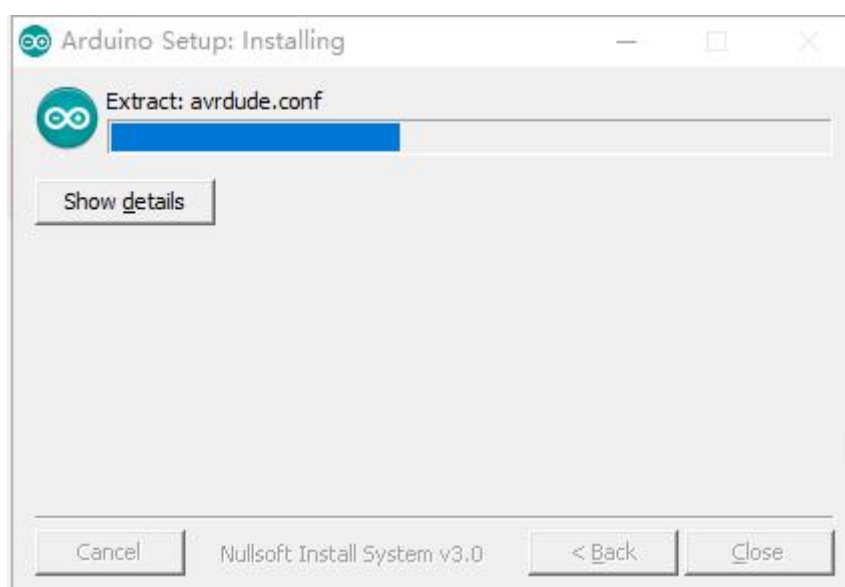


单击下一步（Next）。



您可以按“Browse...”来选择安装路径，或者直接输入您想要的目录。

单击 Install 开始安装。



最后，出现如下界面，单击 Install 完成安装。

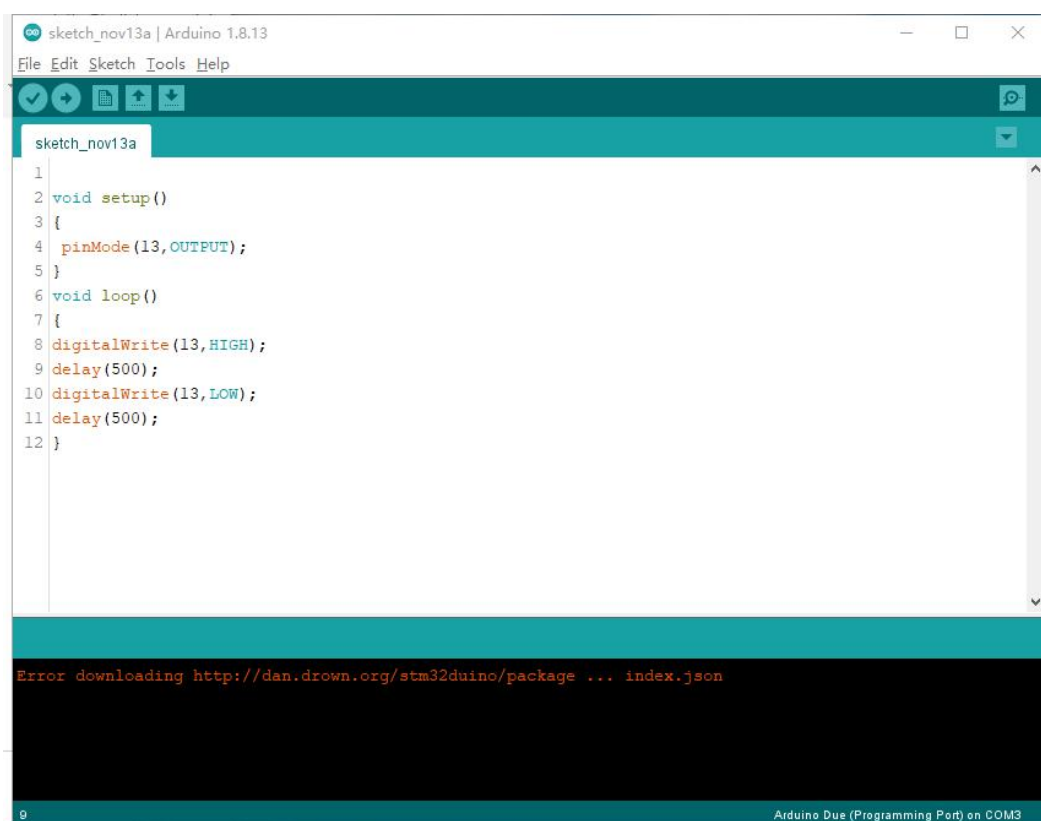


接下来，下面的图标出现在桌面上。



双击打开 Arduino IDE,进入所需的开发环境。

（软件自带中文，需要自行切换）



## 安装 Arduino (Mac OS X)

下载并解压压缩文件，双击 `Arduino.app` 进入 Arduino IDE；如果你的电脑中没有 Java 运行时库，系统会要求你安装它。一旦安装完成，您可以运行 Arduino IDE。

## 安装 Arduino (Linux)

您必须使用 `make install` 命令。如果您使用的是 Ubuntu 系统，建议从 Ubuntu 的软件中心安装 Arduino IDE。

安装额外的 Arduino 库。一旦您熟悉了 Arduino 软件并使用了内置功能，您可能想要使用额外的库来扩展您的 Arduino 功能。

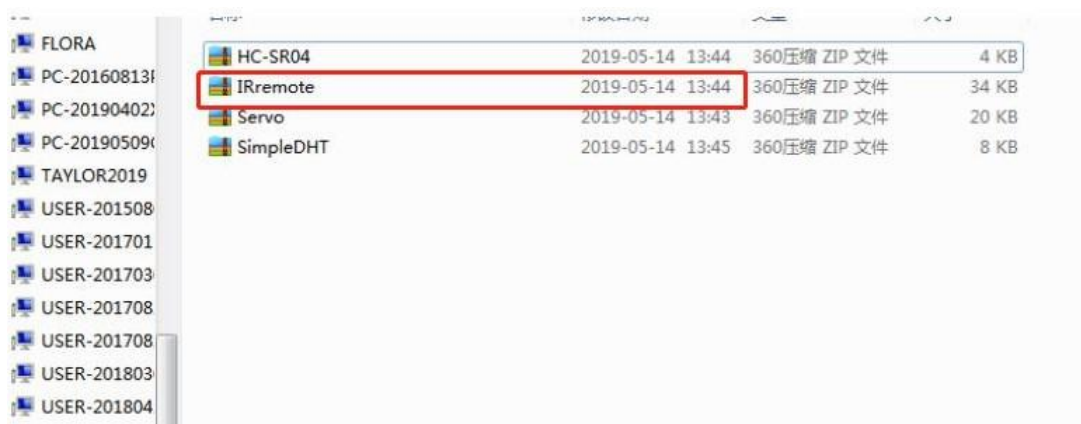
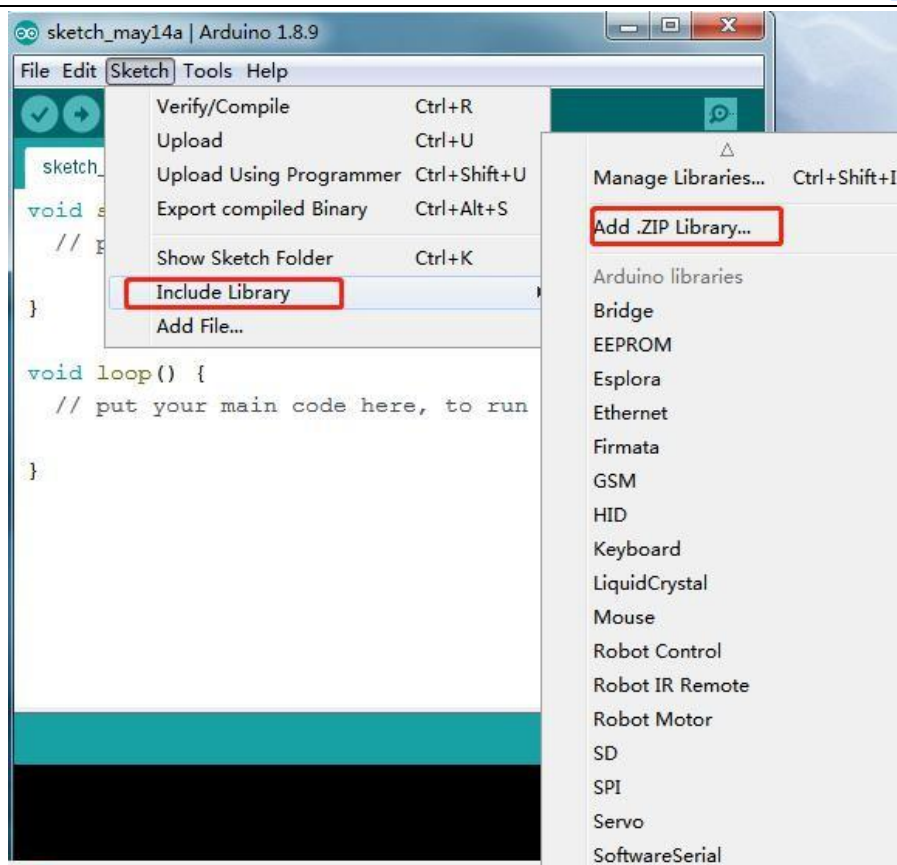
### 1.3 安装库

库是一组代码，可以方便地连接到传感器、显示器、模块等。例如，内置的液晶库使它很容易与字符液晶显示器对话。在互联网上还有数百个额外的图书馆可供下载。参考文献中列出了内置库和一些额外的库。要使用额外的库，您需要安装它们。

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.

如何安装库？使用图书馆管理器。要将新库安装到 Arduino IDE 中，可以使用库管理器(可从 IDE 版本 1.8.9 获得)。打开 IDE，点击“Sketch”菜单，然后包括库>管理库（Include Library > Manage Libraries）。





例如:IRremote

打开 Arduino 软件-项目-加载库-添加一个.zip 库。

添加方法二:

将 library 文件夹复制到 Arduino 安装目录中的 Libraries 文件夹,重启 Arduino,添加的库将生效。

## 1.4 Arduino

Arduino 是一个基于易用的硬件和软件的开源电子平台。适合任何从事互动项目的人。通常，Arduino 项目由电路和代码组成。

Arduino 板是一块集成了微控制器、输入输出接口等的电路板。Arduino 板可以利用传感器感知环境，接收用户操作，控制 LED 等。我们只需要组装电路并编写代码即可。目前 Arduino 开发板有几种型号，不同类型开发板之间的代码是通用的（由于硬件不同，部分开发板可能不完全兼容）包括流行的主控板。

## 1.5 RGB Nano 介绍

RGB Nano 的 14 个数字端口可以作为数字输入或输出，由程序中的 `pinMode()` 定义，由 `digitalWrite` 和 `digitalRead()` 功能块控制。它们在 5V 下工作。每个端口提供输出电流或接收 40 mA 电流。内部有一个上拉电阻，阻值为 20-50 k 欧姆。其他终端有特殊定义。

串行:0 (Rx)和 1 (TX)。用于接收(Rx)和发送(TX) TTL 串行数据。

外部中断:终端 2 和 3。这些外部接口可以配置为以后产生中断，也可以在外部的低电平发生时触发，或者在上升沿和下降沿发生时触发。有关更多信息，请参见 `attachInterrupt()` 函数。

PWM: 3、5、6、9、10、11，提供 8 位 PWM 输出，使用 `analogwrite()` 函数。

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)。这些引脚支持 SPI 通信。虽然硬件支持它们，但是 Arduino 软件中没有包含它们。

Led: 2-13，内置 Led，连接引脚 2-13，该引脚输出高电压时 LED 亮，输出低电压时 LED 灭。

按键:2，是内置按键，连接引脚 2，此引脚可作为上拉输入，当软件配置时，可检测按键是否按下。

蜂鸣器:8 为内置引脚，连接引脚 8。无源蜂鸣器模块连接到该引脚。当这个引脚输出一个频率级时，蜂鸣器可以发出不同的声音。

RGB: 13 是一个内置的引脚，连接到引脚 13，这个引脚集成了 WS2812RGB 光，可以用软件配置让它发出不同的颜色

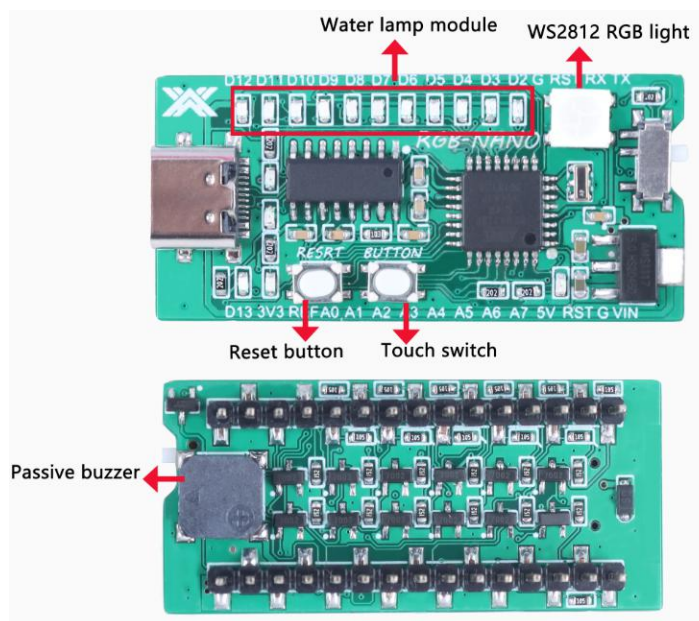
RGB Nano 有 8 个模拟输入，每一个都有 10 位的分辨率(即 1024 种不同的可

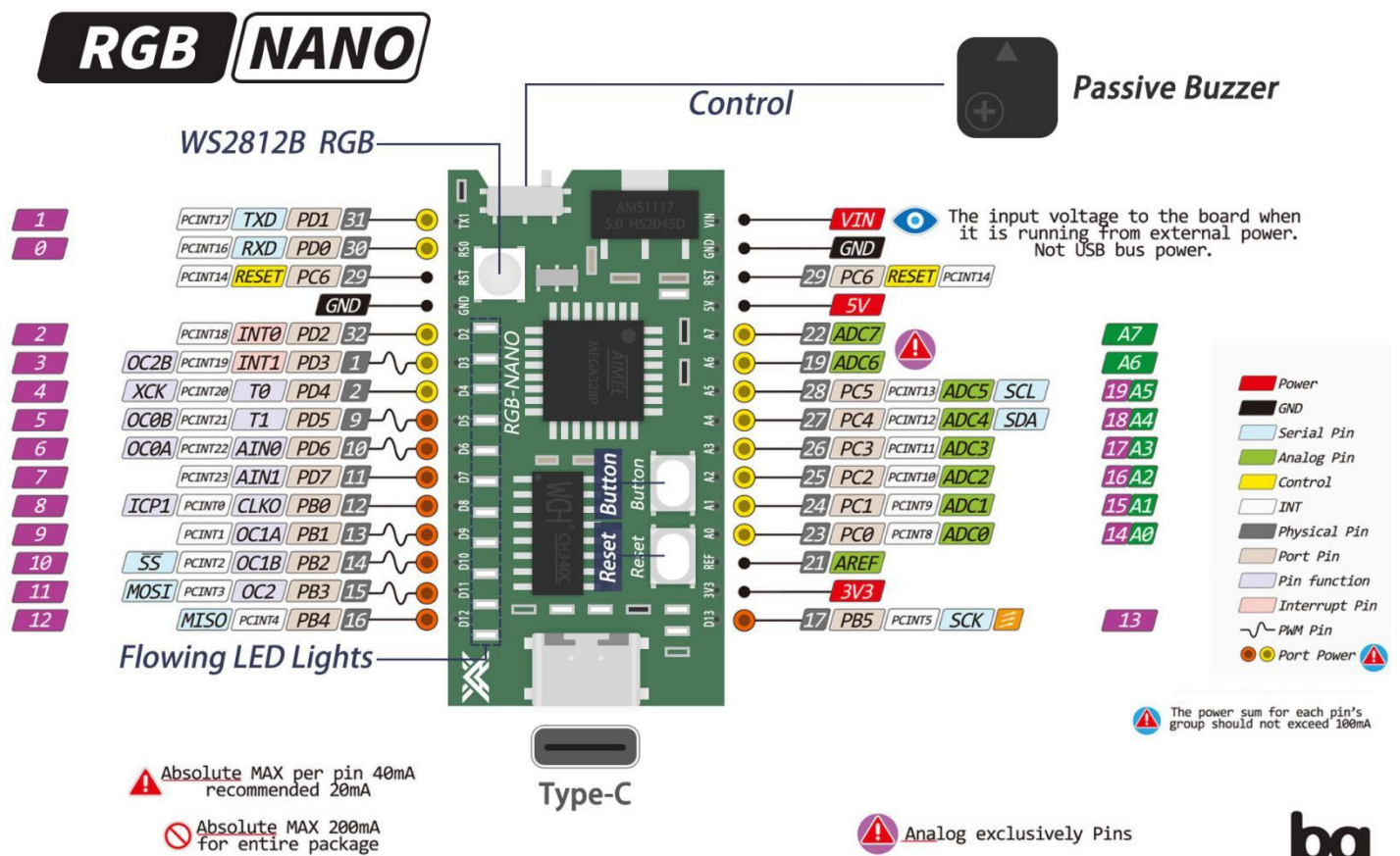
能性)。默认情况下，测量到的对地电压为 5V。当然，它的上限也可以通过 `analogreference()` 函数修改。模拟引脚 6、7 不能作为数字端口使用。此外，一些端口还有许多特殊功能。

I2C: A4 (SDA) and A5 (SCL). 主控板上还有其他端口。

Aref: 模拟输入的参考电压，与 `analogreference()` 一起使用。

Reset (复位): 下拉电势，复位微处理器。按下按钮后，整个系统可以复位。





## 第 2 课 点亮 LED

### 2.1 概述

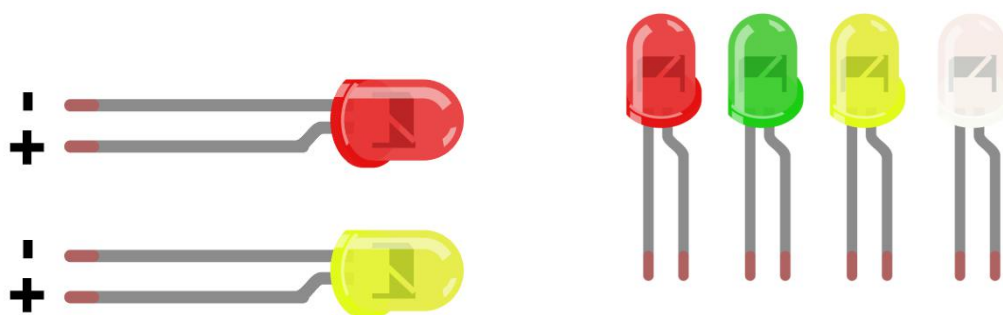
通过这个项目，您可以了解如何使用 RGB Nano 来点亮一个 10mm LED 模块。下载程序并连接好线路后，你会看到 LED 灯点亮成功，如果没有点亮，你需要检查线路是否连接正确，并检查所连接的微控制器的 pin 号是否与之对应。

### 2.2 工作原理

LED (Light Emitting Diode) 是一种将电能转换为光能的发光二极管，它也具有单向导电性，反向击穿电压约为 5V。其正向伏安特性曲线非常陡峭，限流电阻必须串联。在 5V 电路中，通常使用约 400 欧姆的电阻。LED 的两个引脚中较长的是正极。连接方式有两种，当引线的正极通过限流电阻和 Arduino 时。I/O 口已连接，另一端接地。此时 Arduino 输出高时，LED 亮，输出低时，LED 灭。

当 led 的负极连接到 Arduino 的 I/O 端口时，另一端通过限流+电阻连接到 5V 电压。

输出低时 LED 亮，输出高时 LED 灭。



如果你不使用带有电阻器的 LED，那么它可能会立即被破坏，因为太多的电流会流过，加热它并破坏产生光的“结”。

有两种方法来区分哪个是 LED 的正引线，哪个是负引线。

首先，正极引线较长。

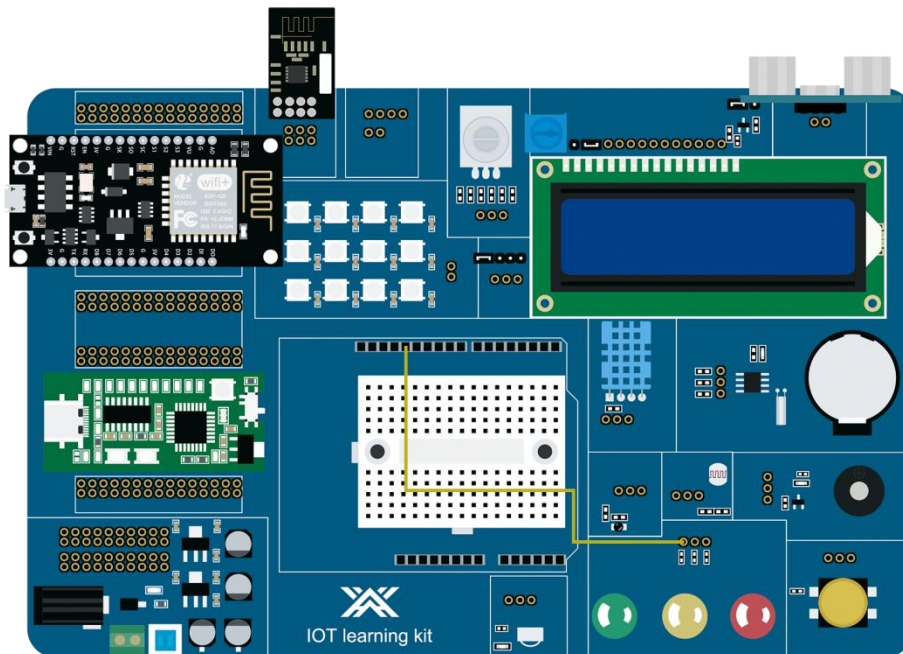
其次，在负极引线进入 LED 主体的地方，有一个平坦的边缘到 LED 的外壳。

如果你碰巧有一个 LED 的侧面是平的，靠近长引线，你应该假设长引线是正的。



## 2.3 连接描述

用杜邦线将微控制器的 D13 引脚引到 JP12 头的任意接口上，插入即可，线连接成功。接线图如下。



## 2.4 代码讲解

定义 LED 信号使能引脚。

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

函数初始化，定义 13 个引脚为输出。

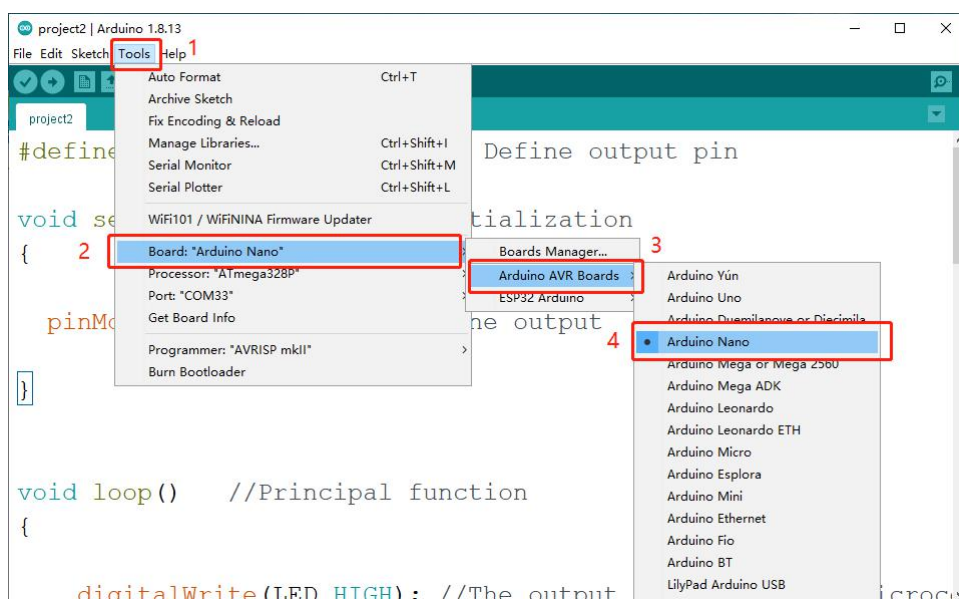
```
void setup() {
    // put your setup code here, to run once:
    pinMode(LED,OUTPUT);
    digitalWrite(LED , LOW); //Initialization,
}
```

主要功能，让 13 引脚输出高电平。

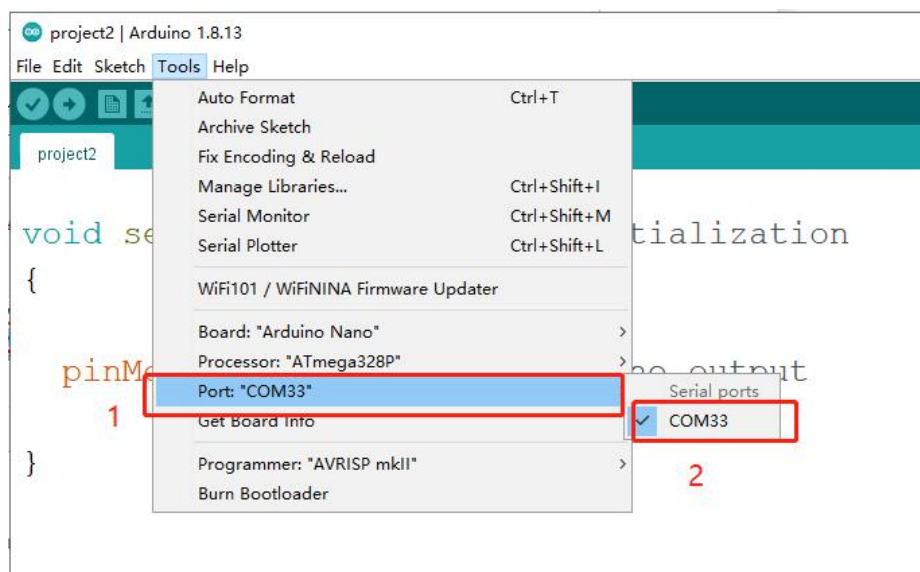
```
void loop() {
    // put your main code here, to run repeatedly:
    /*Since the negative pole of our LED pin has been
    connected, we only need to use the control pin of
    Arduino to output the high level, which is the positive
    pole, and the LED will be lit */
    digitalWrite(LED ,HIGH);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

## 2.5 上传代码

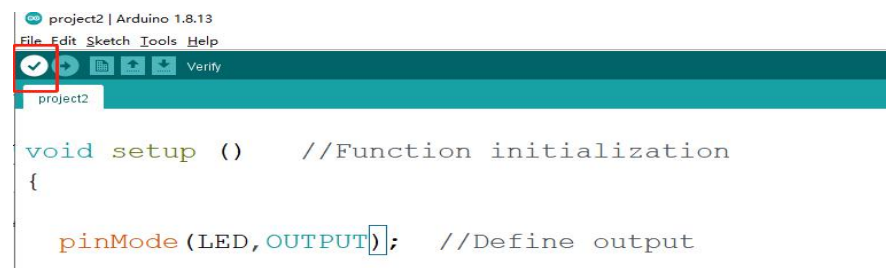
选择 NANO 开发板。



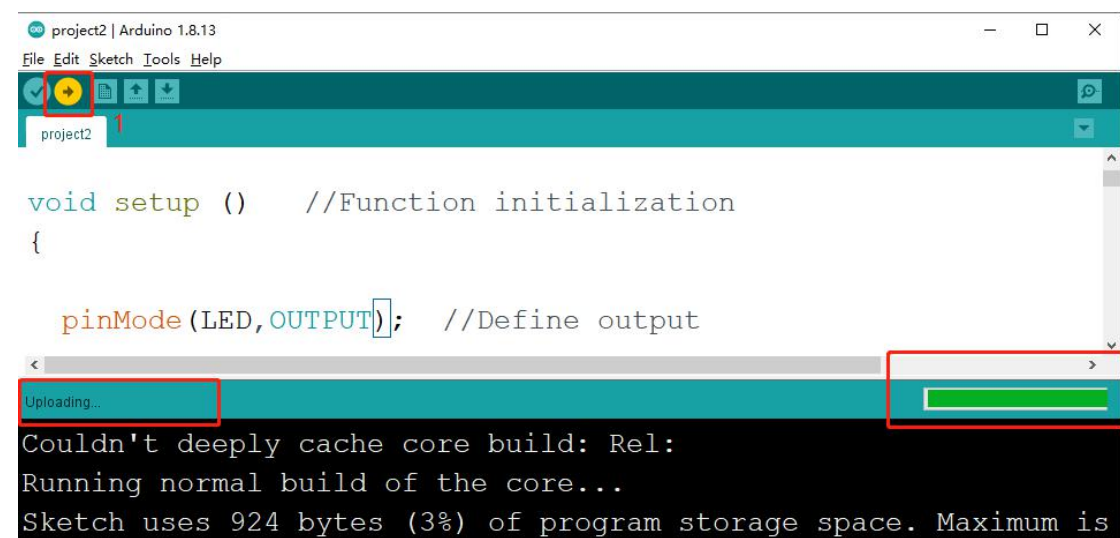
选择端口。



点击编译。



点击上传





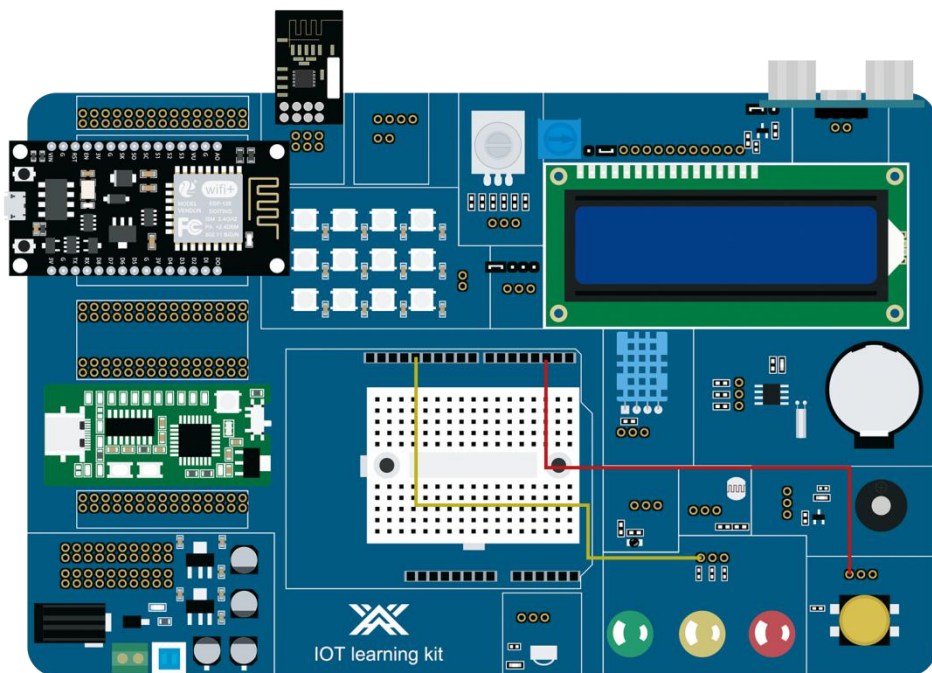
## 第 3 课 按键控制 LED

### 3.1 概述

通过这个项目，可以了解到如何使用 RGB Nano 控制按钮的输入来点亮 10mm 的 LED 灯。下载程序并连接线路后，需要按 WS1 按钮。按下后，可以看到 LED 灯点亮成功，再次按下后，LED 灯熄灭。如未点亮熄灭，则需检查电路是否连接正确，并检查所连接的单片机的引脚号是否与之对应。

### 3.2 连接描述

用杜邦线将单片机的 D13 引脚引到 JP12 头的任意接口上，插入即可连接成功，将单片机的 D2 引脚连接到 JP1 头的 S 端口上，按钮接线也完成。接线图如下。



### 3.3 代码讲解

定义 LED 信号使能引脚。

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

定义全局变量，用作按钮和 LED 的标志位，函数初始化，定义 13 脚作为输出，定义 2 脚作为输入状态。

```
#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);

}
```

主要功能是判断按钮是否被按下，按下后根据标志位判断按钮是否被按下，根据标志位判断 LED 是否点亮或熄灭。

```
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);

}
```

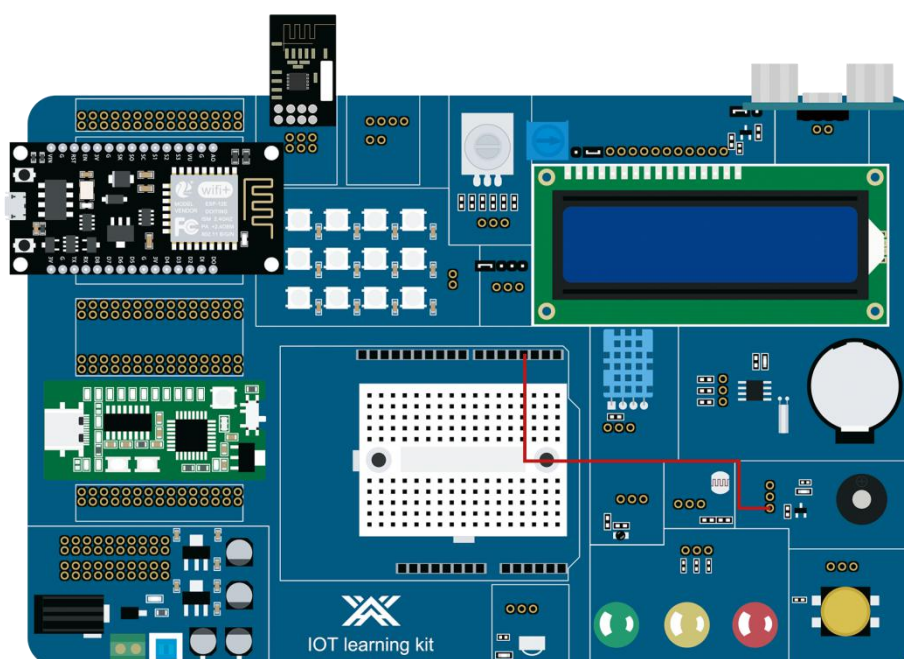
## 第 4 课 有源蜂鸣器

### 4.1 概述

通过这个项目，您可以了解如何使用 RGB Nano 使主动蜂鸣器发出警报。下载程序并连接线路后，会听到蜂鸣器的报警声音。如果蜂鸣器不响，则需要检查电路是否连接正确，并检查所连接的微控制器的引脚号是否与之对应。

### 4.2 连接描述

将单片机的 D3 引脚用杜邦电缆连接到 JP7 排座的 S 接口上，插入即可连接成功。接线图如下：



### 4.3 代码讲解

定义有源蜂鸣器信号使能引脚为引脚 3。

```
#define Buzzer 3
```

初始化，引脚 3 被定义为输出。

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

主要功能是让微控制器先输出 500 毫秒的高电平，再输出 500 毫秒的低电平，使有源蜂鸣器发出报警。

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

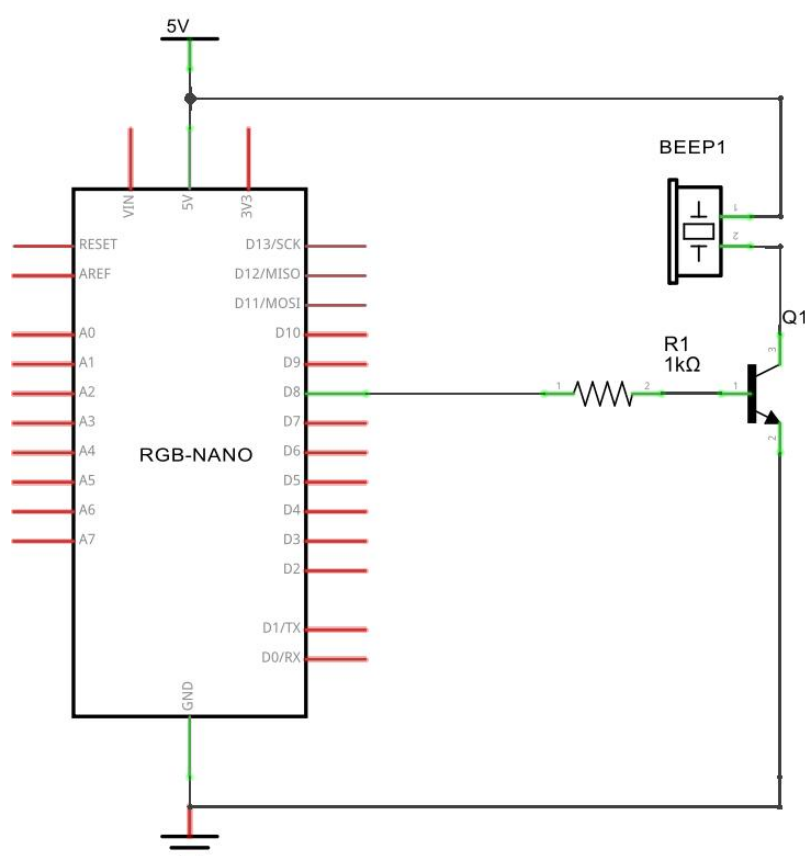
# 第 5 课 无源蜂鸣器

## 5.1 概述

通过这个项目，您可以了解如何使用 RGB Nano 使一个无源蜂鸣器发出警报声。你可以在下载程序后听到由无源蜂鸣器发出的声音，每 0.5S 发出一次。

## 5.2 连接描述

这节课不需要接线，因为无源蜂鸣器已经集成在 RGB Nano 上了，对应的无源蜂鸣器引脚是 D8 引脚。如果下载程序后没有声音，检查蜂鸣器上面是否有一个小拨动开关未打开，这个小拨动开关是 D8 引脚和无源蜂鸣器之间的连接开关，这里需要注意！接线图如下。



### 5.3 代码讲解

定义无源蜂鸣器信号使能引脚为 D8。

```
#define Buzzer 8
```

定义一个全局标志，同时初始化函数，并将引脚 D8 定义为输出。

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

主要功能是使单片机输出一个电平频率，使无源蜂鸣器发出响声。经过一段时间后，无源蜂鸣器可以发出报警声音。如果你想让无源蜂鸣器一直响，你可以让“beep\_bit”变量保持为 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;
    }
}
```



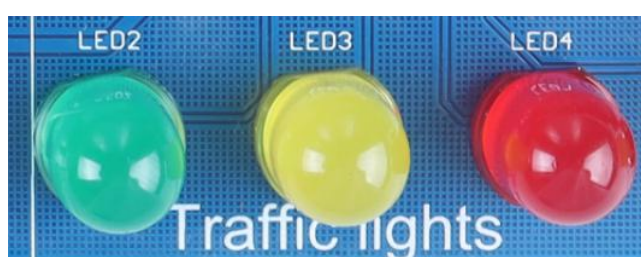
# 第 6 课 交通灯

## 6.1 概述

以上，我们已经完成了单个小光源的控制实验。接下来，让我们做一个稍微复杂一点的红绿灯实验。其实，聪明的朋友可以看出，这个实验是将单个小灯的实验扩展为 3 种颜色，这样就可以实现我们模拟的交通灯实验。

## 6.2 工作原理

信号灯是交通信号的重要组成部分，是道路交通的基本语言。



交通信号灯由红灯(表示禁止通行)、绿灯(表示允许通行)和黄灯(表示警告)组成。

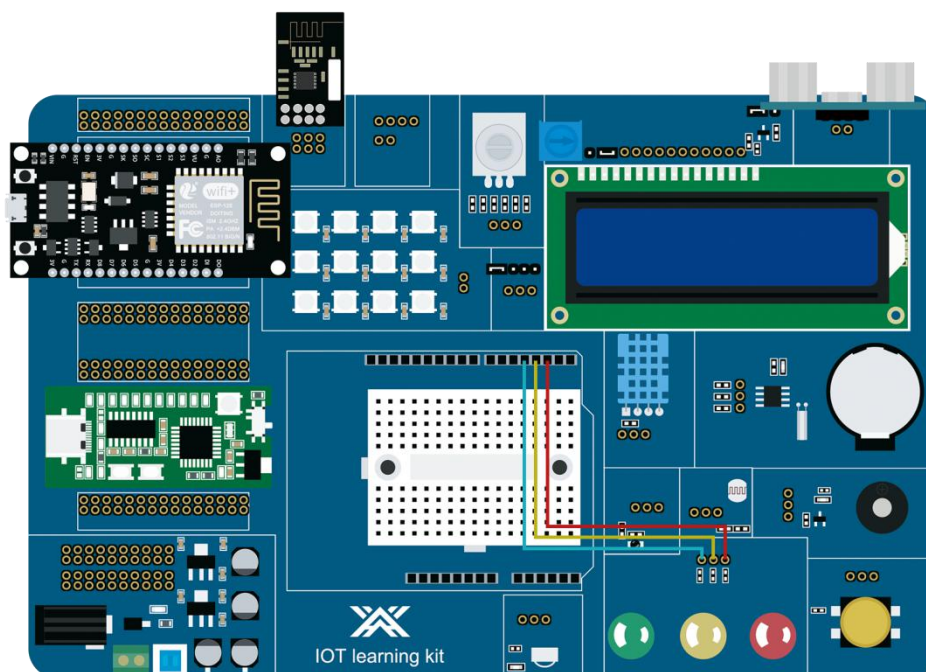
分为:机动车信号灯、非机动车信号灯、人行横道信号灯、车道信号灯、方向指示信号灯、闪烁警告信号灯、公路铁路平面道口信号灯。

道路交通信号灯是一类交通安全产品，是加强道路交通管理，减少交通事故的发生，提高道路使用效率，改善交通状况的重要工具。

适用于十字路口、丁字路口等十字路口。由道路交通信号控制机控制，引导车辆和行人安全有序通行。

交通信号灯的种有:机动车道路信号灯、人行横道信号灯、非机动车道信号灯、方向指示信号灯、移动交通信号灯、太阳能闪光警示灯、收费站顶棚信号灯。

## 6.3 接线示意图



## 6.4 代码讲解

单片机引脚 2、3、4 连接 LED 正极，并初始化设置为输出模式。

```
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
}
```

将数字引脚 2 设置为高，其余设置为低，延迟 5 秒，保持绿灯亮 5 秒。



```
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
```

绿灯每 500 毫秒闪一次，总共闪三次

```
//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);
```

打开黄灯，关闭红、绿灯 1 秒，然后红灯点亮五秒钟。

```
//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);
```

红灯每 500 毫秒闪三次。

```
//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);
```

## 第 7 课流水灯

### 7.1 概述

在本课程中，您将学习如何控制 LED 的开和关来实现流水灯功能。

### 7.2 工作原理

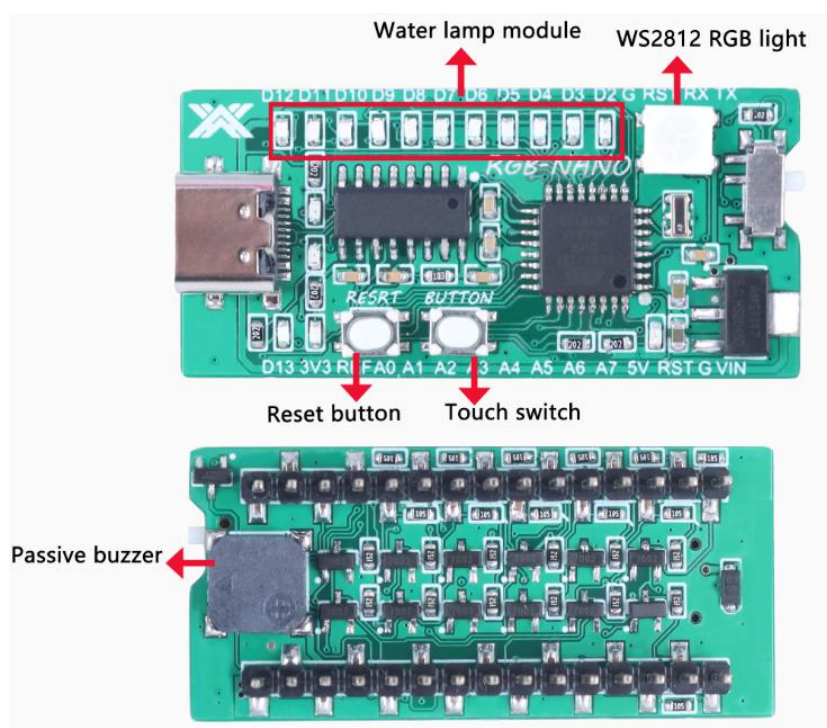
商场灯采用 RGB-Nano，可实现开灯、关灯、闪烁功能，LED 闪烁时间自行设定。

当引脚输出为低电平时，LED 灯不会亮，当引脚输出为高电平时，LED 灯会亮。

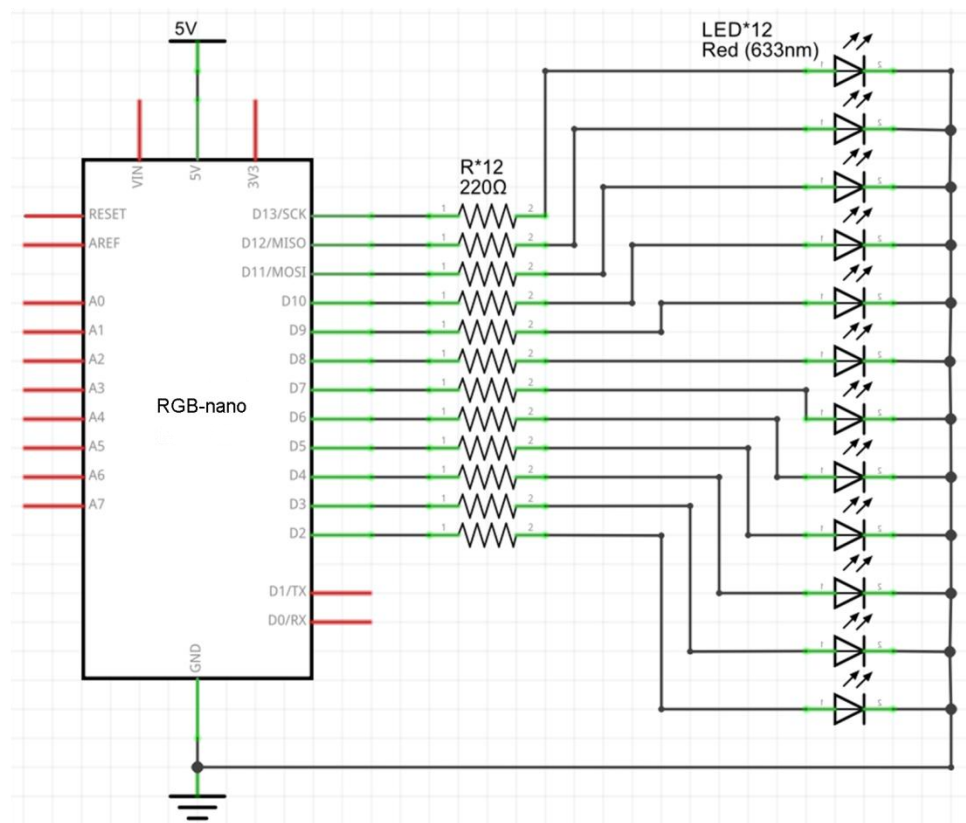
需要多个引脚控制，所以这里使用数字引脚 2 到 13。

分别控制灯的开、关时间，即可达到流水灯的效果。

流水灯的效果：所有的灯开始执行，然后点亮一个接一个的熄灭，执行一个接一个向右，更快的开始，随着时间的变化，越来越慢，到达最慢时间；流水灯将变得很快，这个过程是循环的，当然也可以其他效果，可以自行设定。



## 7.3 接线示意图



## 7.4 代码讲解

定义两个时间变量，然后将引脚 2 到 13 设置为输出模式。

```
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
    }
}
```

从第 2 盏灯开一段时间，然后关一段时间。

打开第 3 盏灯，等一会儿，关上第 3 盏灯，然后打开第 4 盏灯，以此类推。

```
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);
    }
}
```

从第 13 盏灯开始，等待一段时间，关闭第 13 盏灯，让第 12 盏灯打开，等待一段时间，关闭第二盏灯，继续点亮并关闭下一盏灯。

```
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);
}
}
```

## 第 8 课 WS2812B

### 8.1 概述

WS2812B 是一款内置 LED 驱动芯片。一个 IO 端口可以控制多个 led，亮度调节，颜色调节等功能。

### 8.2 工作原理

WS2812B 是一种集控制电路和发光电路于一体的智能外控 LED 光源。

形状与 5050 LED 灯珠相同，每个元素都是一个像素点。

所述像素包含智能数字接口数据锁存信号整形放大驱动电路，还包含高精度内部振荡器和 5V 电压可编程固定电流控制部分，有效保证像素的光色高度一致。

数据协议采用单行归零码通信方式。当像素上电并复位后，DIN 端接受来自控制器的数据传输。第一个发送的 24 位数据被第一个像素提取并发送到该像素内的数据锁存器。

经过内部整形和电路整形后，剩余的数据通过 DO 端口被放大转发到下一个级联像素。每传输一个像素，信号就减少 24 位。

像素采用自动整形和转发技术，级联像素的数量不受信号传输的限制，只受信号传输速度的限制。

RGB 集成在开发板上，RGB 灯的控制引脚连接到 RGB-nano 板的引脚 13。

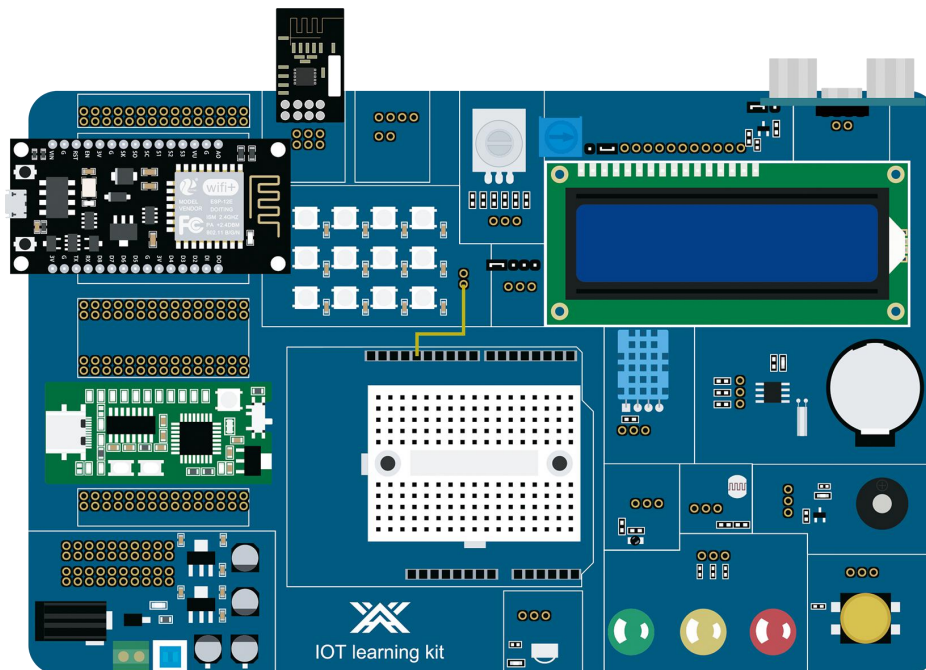


### 8.3 特征

1. 控制电路和 RGB 芯片集成在一个 5050 封装组件中, 形成一个完整的外部控制像素点。
2. 内置信号整形电路, 通过任意像素接收信号后的波形。
3. 类型输出, 保证线波形失真不会累积。
4. 内置上、下电复位电路。
5. 每个像素的三原色可实现 256 级亮度显示, 完成 16777216 种颜色的全真彩显示, 且扫描频率不低于 400Hz/s。
6. 串口级连接端口, 通过信号线完成数据的接收和解码。
7. 任何两点传输距离小于 5 米时不需要增加任何电路。
8. 当刷新率为 30 帧/秒时, 低速模式级联数不小于 512 点, 高速模式级联数不小于 1024 点。数据传输速度可达 800Kbps。
9. 光颜色的一致性。



## 8.4 接线示意图



## 8.5 代码讲解

参考驱动 WS2812B RGB 灯的库，因为 RGB 灯信号引脚连接到开发板处理器引脚 13，所以数字引脚 13 定义是连接到 RGB 信号引脚。

```
#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
```

有些设置需要初始化，如串口波特率设置为 9600，初始化 RGB 光功能。

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

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

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

首先调用 `clear` 函数，然后在 `for` 循环中打开第一盏灯并将颜色设置为绿色。

因为我们有 12 个灯，所以我们经过 12 个循环。

```
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
  }
}
```

## 第 9 课渐变 RGB

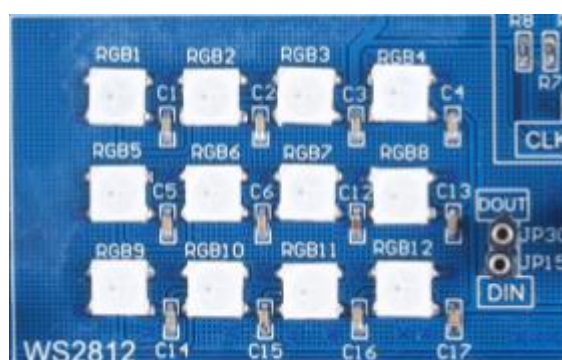
### 9.1 概述

WS2812B 是一款内置的 LED 驱动芯片。在本课程中，您将学习如何从一个 IO 端口控制多个 LED，亮度调整，颜色调整和其他功能。

### 9.2 工作原理

WS2812B 是一种集控制电路和发光电路于一体的智能外控 LED 光源。

形状与 5050 LED 灯珠相同，每个元素都是一个像素点。



所述像素包含智能数字接口数据锁存信号整形放大驱动电路，还包含高精度内部振荡器和 5V 电压可编程固定电流控制部分，有效保证像素的光色高度一致。

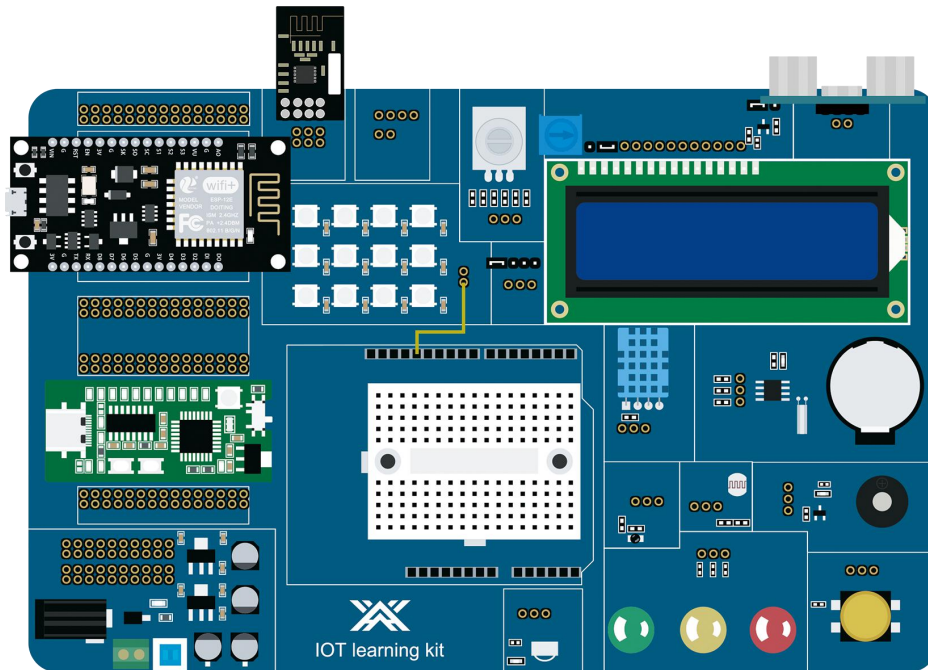
数据协议采用单行归零码通信方式。当像素上电并复位后，DIN 端接受来自控制器的数据传输。第一个发送的 24 位数据被第一个像素提取并发送到该像素内的数据锁存器。

经过内部整形和电路整形后，剩余的数据通过 DO 端口被放大转发到下一个级联像素。每传输一个像素，信号就减少 24 位。

像素采用自动整形和转发技术，级联像素的数量不受信号传输的限制，只受信号传输速度的限制。

RGB 集成在开发板上，RGB 灯的控制引脚连接到 RGB- nano 板的引脚 13。

## 9.3 接线示意图



## 9.4 代码讲解

参考驱动 WS2812B RGB 灯的库，因为 RGB 灯信号引脚连接到开发板处理器引脚 13，所以数学引脚 13 定义是连接到 RGB 信号引脚。因为有 12 个灯，所以程序中所有 LED 灯的数量都设置为 12。

```
#include <Adafruit_NeoPixel.h>
#ifdef __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);
```

做一些初始化设置。

```
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)  
}
```

调用参数为 2 毫秒的 `rainbow` 函数，表示每个梯度的时间延迟，在梯度结束后调用 `RGB` 函数。

```
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
    }
}
```

## 第 10 课 DS1307

### 10.1 概述

在许多电子设备中，操作必须按时间进行。

当主系统关闭时，你不应该停止计算电脑、手机等设备的时间和日期。

因此，采用实时时钟(RTC)模块。

在本节中，您将学习如何使用 RTC DS1307 模块和 RGB-Nano 开发板来制作一个由 LCD 屏幕显示的电子时钟。

### 10.2 LCD1602 介绍



LCD1602 引脚介绍:

连接到地面的引脚。

VDD:连接+5V 电源的引脚。

VO:调节 LCD1602 的对比度。

RS:一种寄存器选择引脚，它控制在 LCD 存储器中你写入数据的位置。你可以选择数据寄存器，它保存屏幕上的内容，或者指令寄存器，它是 LCD 控制器寻找下一步操作指令的地方。

R/W:可选择读或写模式的读/写引脚

E:一个使能引脚，当提供低能量时，使 LDC 模块执行相关指令。

D0-D7:读写数据的引脚。

A 和 K:控制 LED 背光的引脚。



## 10.3 DS1307 介绍

DS1307 是一种低功耗，具有 56 字节的非易失性 RAM 全 BCD 码时钟日历实时时钟芯片，地址和数据通过两线双向串行总线传输，该芯片可以提供秒、分、时等信息，每个月的天可以自动调整，它也有闰年补偿。

Real Time Clock，简称 RTC，是一种跟踪当前时间的系统，可用于任何需要保持准确时间的设备。

您也可以在不使用 RTC 系统的情况下跟踪准确的时间，但是 RTC 有一些重要的优点。

以下是其中一些：

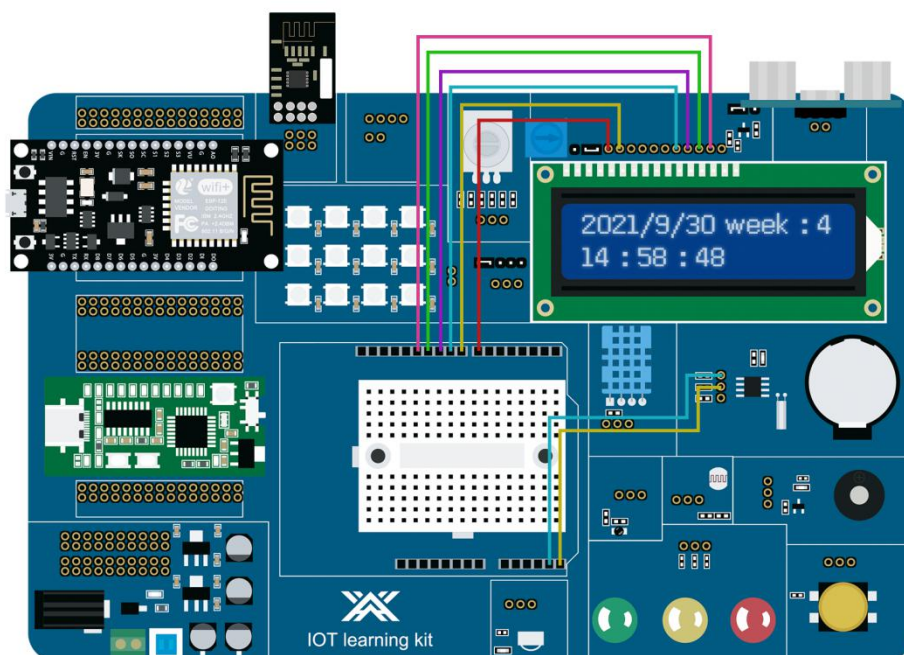
从时间计算中释放系统时间(这个特性非常关键，因为在许多情况下，CPU 正在执行微妙的任务，如接收传感器数据。

如果你不使用 RTC, CPU 也必须跟踪时间，它可能会中断处理器的主要任务。

RTCS 通常有一个备用电源，使用 CR2025 纽扣电池，这样当主电源关闭或不可用时，它们可以继续工作一段时间。RTC 通常使用 32.768kHz 晶体振荡器。但为什么是 32.768kHz 呢？32768Hz 是 2 的 15 次方，所以很容易就能得到 1 秒。此外，晶体必须小，宽度适中，功耗低，32876Hz 即可满足要求。

频率越高，晶体越弱，频率越低，功耗越大。

## 10.4 接线示意图



## 10.5 代码讲解

声明一些驱动液晶显示器和 DS1307 时钟芯片库,调用这些库使我们的编程更加简单、方便,定义液晶显示器和开发板连接,是数字接口 7, 8, 9, 10, 11, 12。

(注意:在上传代码时,首先更改现有的代码 DS1307\_Write。以当地时间为准上传代码后,上传代码 DS1307\_Write。成功,然后上传代码 DS1307。上传成功后,显示当地正常时间。)

```
#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
```

在程序开始运行之前,您需要设置一些初始化设置,串口的波特率,显示的初始化。

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

调用 DS1307 的函数，需要将日期写入 DS1307，然后读取数据。

```
void DS1307()
{
    //read the time
    Wire.beginTransaction(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();
        }
    }
}
```

在串口中打印日期和时间。

```
//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;
}
```

# 第 11 课 显示温度

## 11.1 概述

在本节课中，您将学习如何使用 LCD 1602 显示温度信息。

显示器由 LED 背光，可显示两行，每行最多 16 个字符。

你可以在显示器上看到每个字符的矩形和组成每个字符的像素。

显示器是蓝色和白色的，用来显示文本。

在本节课中，我们将运行 LCD 库的 RGB-Nano 板温度显示程序。

## 11.2 热敏电阻简介



热敏电阻是一种电阻比标准电阻更依赖于温度的电阻。这个词是 **thermal** 和 **resistor** 的合成词。热敏电阻广泛应用于涌流限制器、温度传感器(典型的 NTC 型)、自复位过流保护器和自调节加热元件。

规格:

型号: NTC-MF52 3950

3 个引脚

温度范围:  $\sim 55^{\circ}\text{C} \sim +125^{\circ}\text{C}$

准确性 :  $\pm 0.5^{\circ}\text{C}$

上拉电阻 :  $10\text{K}\Omega$

引脚配置:

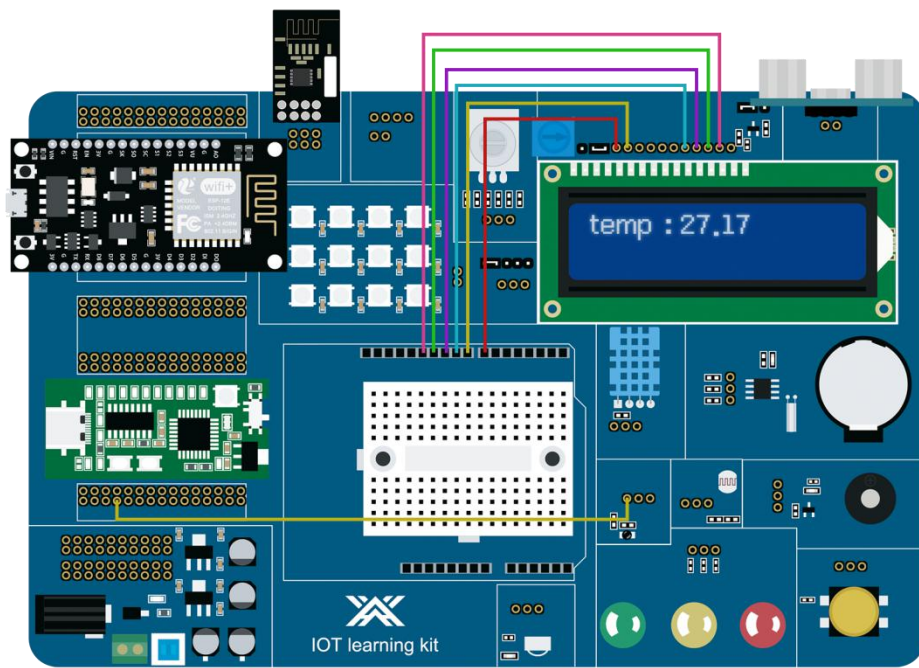
1、“S”: GND

2、“+”: +5V

3、“-”: 信号引脚



### 11.3 接线示意图



### 11.4 代码讲解

调用 LCD 显示库，定义模拟接口 A0 连接模拟温度传感器。定义连接 LCD 显示器到开发板的引脚为数字端口 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
```

串口波特率设置为 9600，初始化 LCD 界面。

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

得到热敏电阻的模拟数据，计算后显示在串口调试窗口和 LCD 屏幕上。



```
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/Ro)
    steinhart = log(steinhart); // ln(R/Ro)
    steinhart /= BCOEFFICIENT; // 1/B * ln(R/Ro)
    steinhart -= 1.0 / (NOMINAL_TEMPERATURE + 273.15); // + (1/To)
    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);
}
```

## 第 12 课显示温度和湿度

### 12.1 概述

在这节课中，你将学习如何使用 DHT11 和 LCD 显示屏来显示温度和湿度传感器测量的数据。

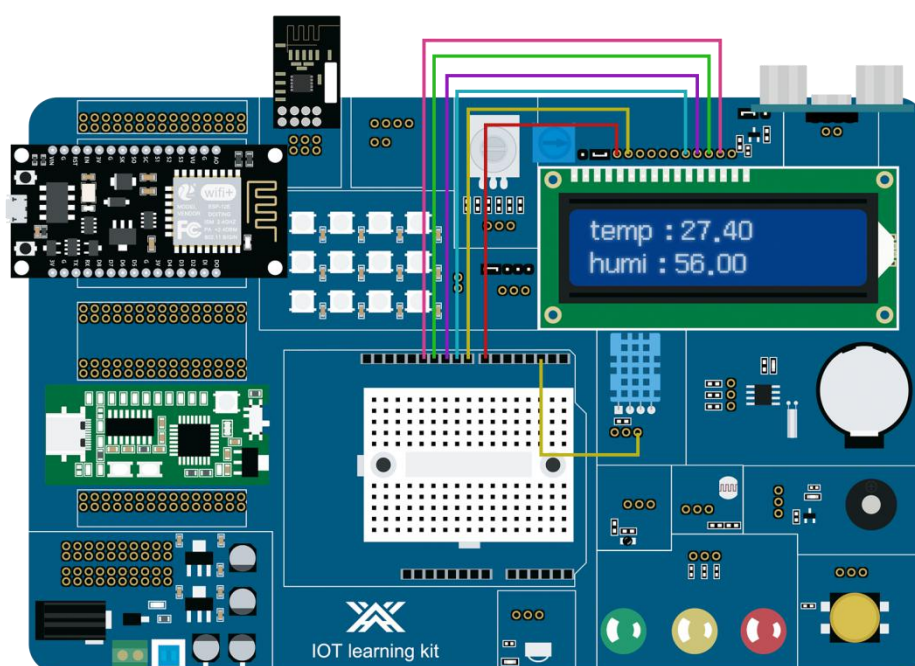
显示屏有 LED 背光，可以显示两行，每行最多 16 个字符。

你可以在显示器上看到每个字符的矩形和组成每个字符的像素。

显示器是蓝色和白色的，用来显示文本。

在本节课中，我们将运行 LCD 库的 RGB-Nano 板温度显示程序。

### 12.2 接线示意图



### 12.3 代码讲解

调用温湿度传感器(DHT11)和 LCD 显示库，并定义处理器编号 2 连接到温湿

度传感器。定义 LCD 显示和开发板连接的引脚，分别是数字端口 7、8、9、10、11、12 引脚。

```
#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);
```

串口波特率设置为 9600，初始化 DHT11 传感器，初始化 LCD 界面。

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

获取传感器的湿度和温度数据，并在串口调试窗口和 LCD 屏幕上实时显示。

```
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);
}
```

# 第 13 课超声波模块

## 13.1 概述

在本节课中，您将学习如何连接和使用超声波测距传感器和 LCD 1602 显示器。

显示屏有 LED 背光，可以显示两行，每行最多 16 个字符。

你可以在显示器上看到每个字符的矩形和组成每个字符的像素。

显示器是蓝色和白色的，用于显示文本。

在本节课中，我们将运行 RGB-Nano 样本程序，在 LCD 1602 显示屏上显示超声波传感器测量距离信息。



## 13.2 超声波传感器的介绍

超声波传感器模块 HC-SR04 提供 2cm-400cm 非接触式测量功能，测距精度可达 3mm。该模块包括超声波发射、接收和控制电路。工作的基本原则：

使用 IO 触发器对至少 10us 的高电平信号。

模块自动发送 8 个 40kHz，检测是否有脉冲信号返回。

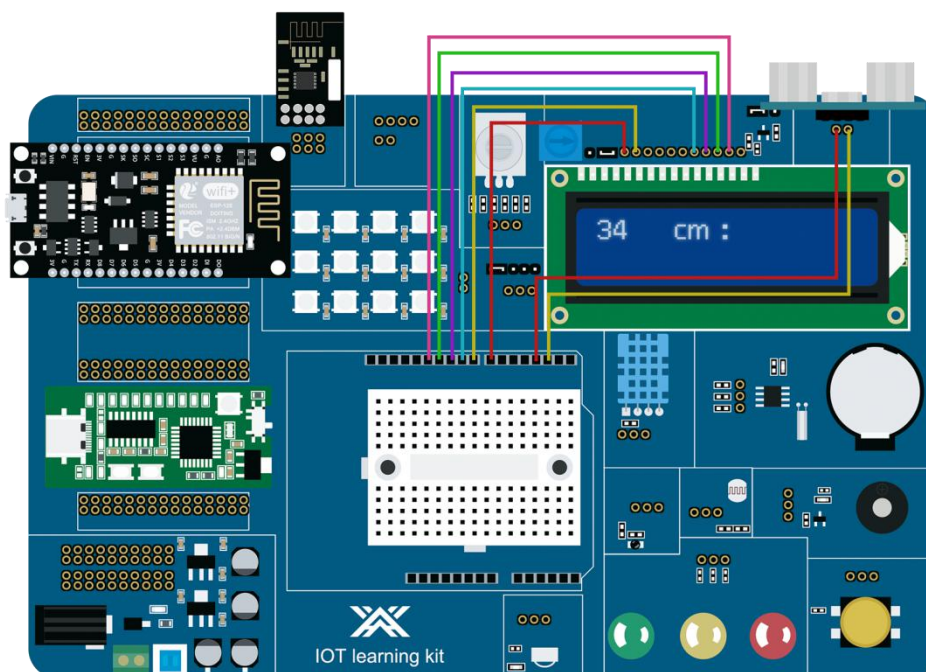
如果信号返回，通过高电平，高输出 IO 持续时间是从小发送超声波到反射回来的时间。

测试距离=(高电平时间×声速(340m/s) / 2。

你只需要提供一个短的 10us 脉冲触发输入开始测距，然后模块将发出一个 8 周期的 40 kHz 的超声脉冲，并提高其回声。回波是一个距离对象，它是脉冲宽度和范围的比例。你可以通过发送触发信号和接收回波信号之间的时间间隔来计算范围。公式:us / 58 =厘米或 us / 148 =英寸;或:量程=高电平时间\*速度(340M/S) / 2;我们建议使用 60ms 以上的测量周期，以防止触发信号到回波信号。



### 13.3 接线示意图



超声波传感器模块 HC-SR04 测得的数据经 RGB-Nano 板处理后显示在液晶屏上。

### 13.4 代码讲解

声明驱动液晶屏库，超声波传感器模块 HC-SR04 接口。

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

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

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

初始化串口和屏幕，设置超声波传感器模块 HC-SR04 接口。

```
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
}
```

通过超声波传感器模块 HC-SR04 获取测量数据,并在串行调试窗口和 LCD 屏幕上显示。

```
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;
}
```



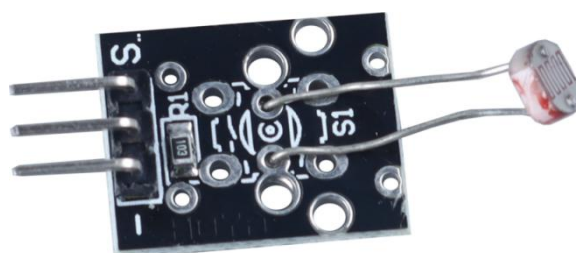
# 第 14 课 光敏电阻

## 14.1 概述

光敏电阻，是利用半导体光电效应制成的电阻值随入射光的强度而变化的电阻器；

当入射光强时，电阻减小，当入射光弱时，电阻增大。

光敏电阻通常用于光测量、光控制和光电转换(将光的变化转换为电的变化)。



## 14.2 组件的介绍

光敏电阻器可广泛应用于各种光控电路，如光控、调节等场合，也可用于光控开关。

在这个实验中，我们首先进行了一个比较简单的光敏电阻使用实验。

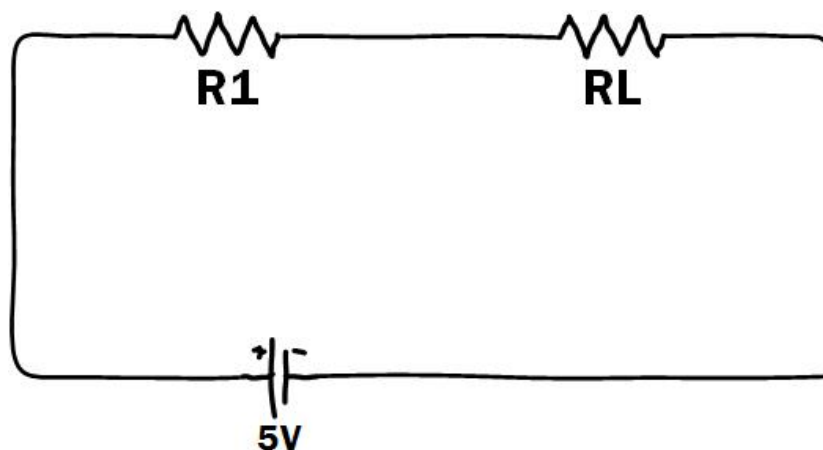
由于光敏电阻是可以根据光强改变电阻值的元件，自然也需要模拟口来读取模拟值。在这个实验中，我们可以借鉴 PWM 接口实验，将电位器改成光敏电阻，实现 LED 小灯的亮度在光强不同的情况下也会发生相应的变化。

光敏电阻器在黑暗条件下电阻很高。

光线越强，电阻越小。

通过测量光敏电阻两侧的电压变化，可以知道光敏电阻值的变化，从而得到光照强度值。

在连接图中，我们可以找到一个与光敏电阻串联的部分电压电阻。



上图中， $R_L$  为光敏电阻， $R_1$  为串联电阻。在黑暗中， $R_L$  会非常非常大，所以  $V_{out}$  会非常大，接近 5V。

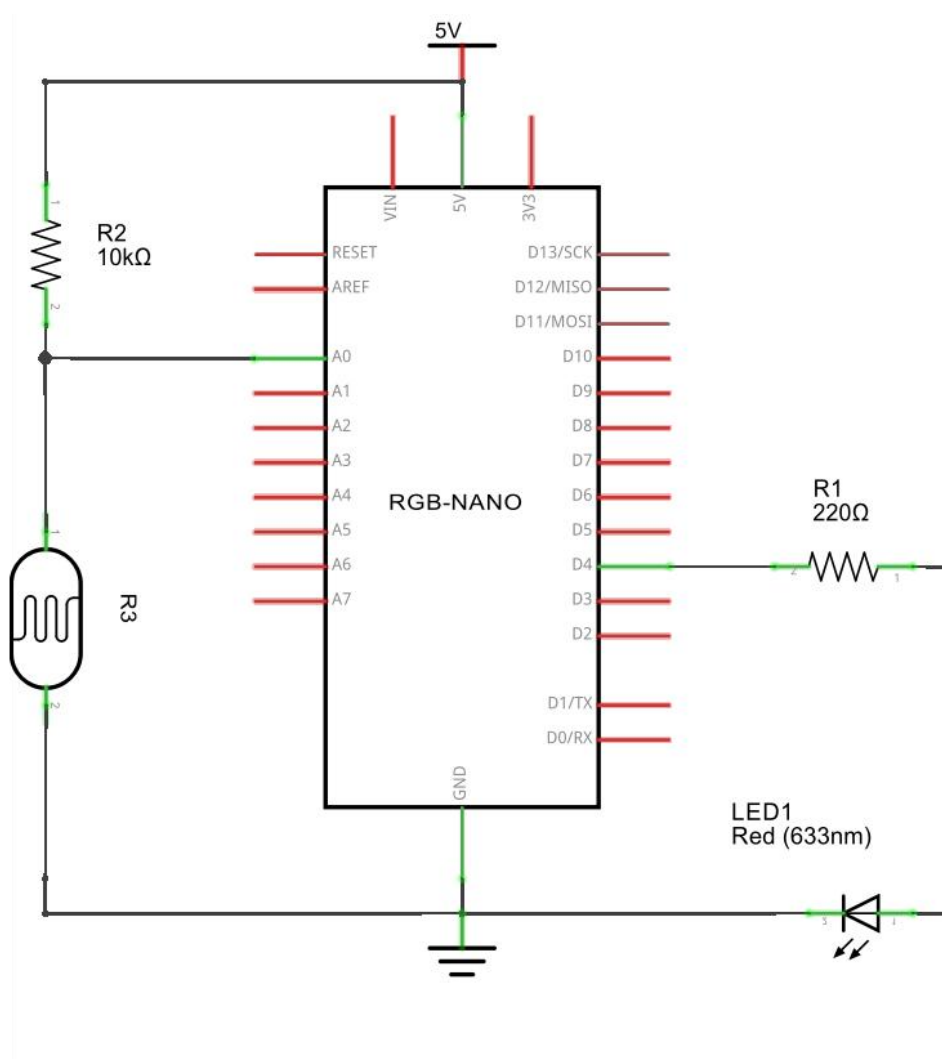
公式如下：

$$V_{out} = \frac{R_L}{R_1 + R_L} * V_{in}$$

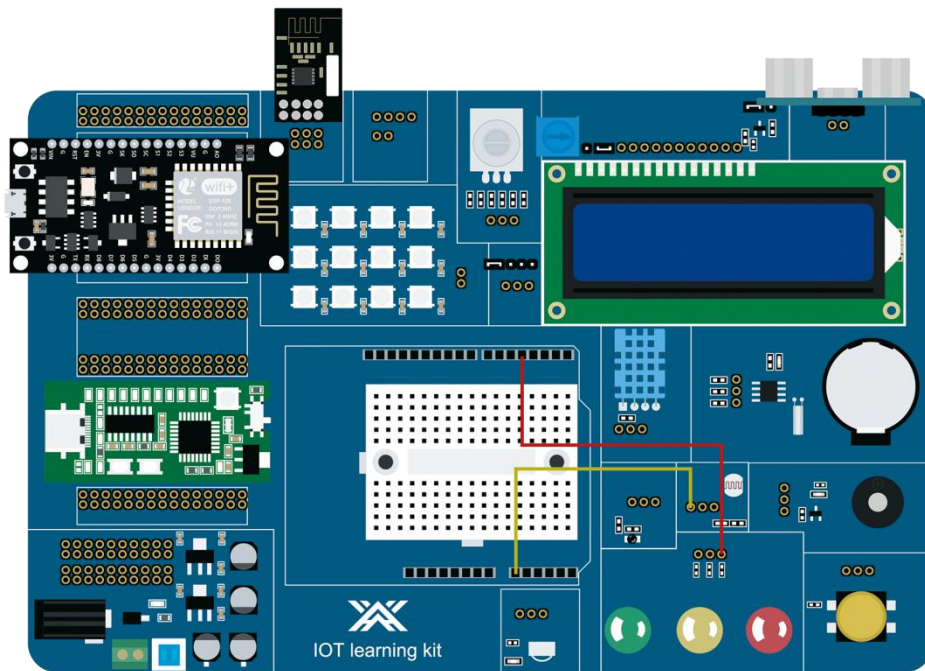
当光线照射时， $R_L$  的值迅速下降，所以  $V_{out}$  也随之下降。

由上式可知， $R_1$  的选择不宜过小，宜在 1K ~ 10K 左右，否则比值变化不大。

## 14.3 连接图



## 14.4 接线示意图



## 14.5 代码讲解

定义光敏电阻连接模拟引脚 A0, LED 连接数字引脚 4。

```
int inputValue = 0;

//Define the pins
#define RES_Pin A0 //A0 Connect the photoresistor
#define LED_Pin 4  //D4 Connect the LED pins
```

当光敏电阻器检测到光线昏暗时，它将 LED 灯点亮，将连接 LED 灯的引脚设置为输出模式。

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

利用模拟读数功能读取光敏电阻的数据，然后将该数据作为日夜的判断条件。如果检测到的数据小于 500，说明当前环境已经变暗，所以让连接到 LED 的引脚输出高电平，并使 LED 亮起。

```
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);
    }
}
```

# 第 15 课 旋转编码器控制 RGB

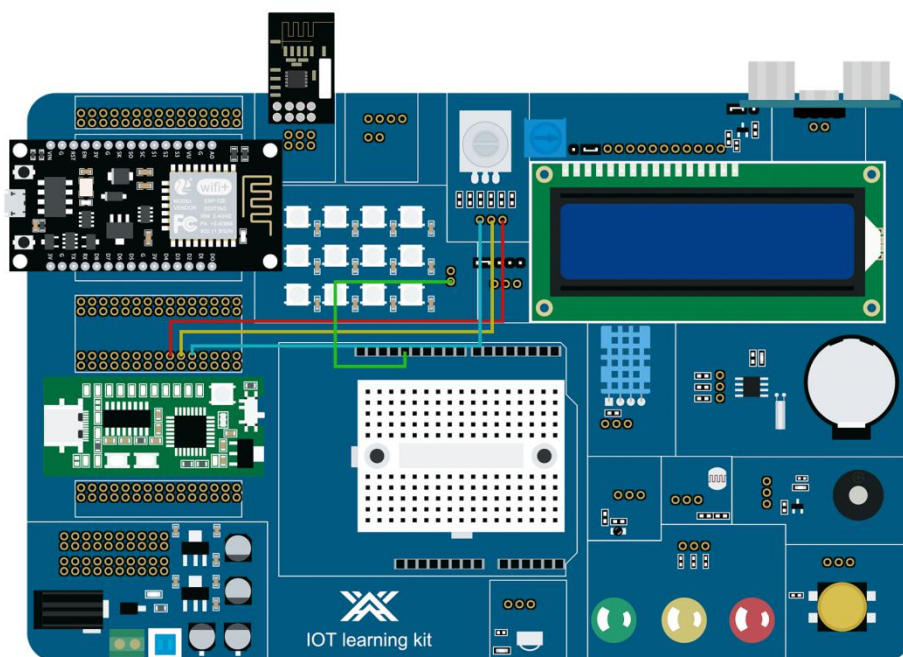
## 15.1 概述

通过这个项目，可以学习使用旋转编码器在 RGB Nano 微控制器上控制 RGB 显示运行光。

连接描述:

将单片机的 D13 引脚用杜邦电缆连接到 JP15 排的 DIN 接口上，插入即可连接成功。然后用杜邦线将微控制器的 D2 引脚连接到 JP13 排座的“CLK”上，D3 引脚连接到 JP3 排座的“DT”上，D4 引脚连接到 JP13 排座的“SW”上。连接成功。

## 15.2 接线示意图



## 15.3 代码讲解

包含 RGB 库文件，定义 RGB 灯的数量和引脚编号，并定义函数引脚和函数所需的全局变量。



```
#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
```

同时初始化功能，确定引脚的工作类型，同时使能中断 1 功能。

```
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);
}
```

主要功能，编码器发送的值通过显示功能使相应的 RGB 光被点亮。通过改变“计数”可以改变被点亮的光。

```
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);
}
```

编码器处理功能将决定编码器是向前旋转还是向后旋转，同时改变“count”的值。

```
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
    }
  }
}
```

# 第 16 课 NRF24L01 无线接收发射模块

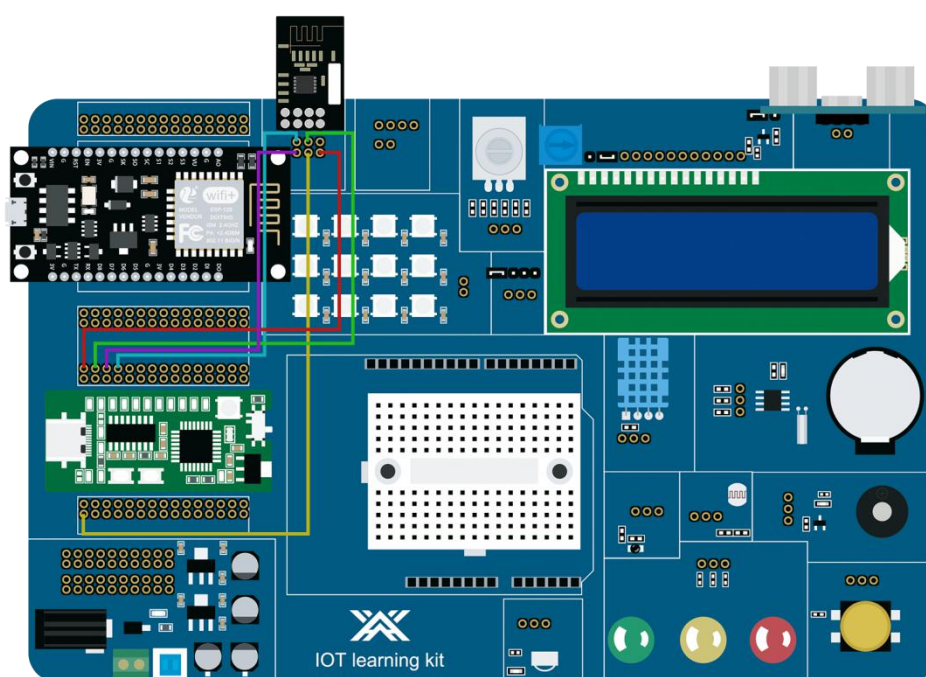
## 16.1 概述

通过这个项目，您将学习使用 NRF24L01 发送数据和使用 IDE 编译器的串口显示数据。

连接描述：

将单片机的 D9、D10、D11、D12、D13 引脚用杜邦线缆连接到 JP3 下面头的“CSN”、“CE”、“MOSI”、“MISO”、“SCK”上，同时将 NRF24L01 模块连接到 JP3 母线上，连接完成，连接成功。

## 16.2 接线示意图



## 16.3 代码讲解

包含项目必须使用的库文件。

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

程序的函数脚是在全局变量定义中定义的。

```
//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;
```

初始化函数以确定工作波特率为“57600”，需要与串口波特率保持一致，否则会产生乱码。

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

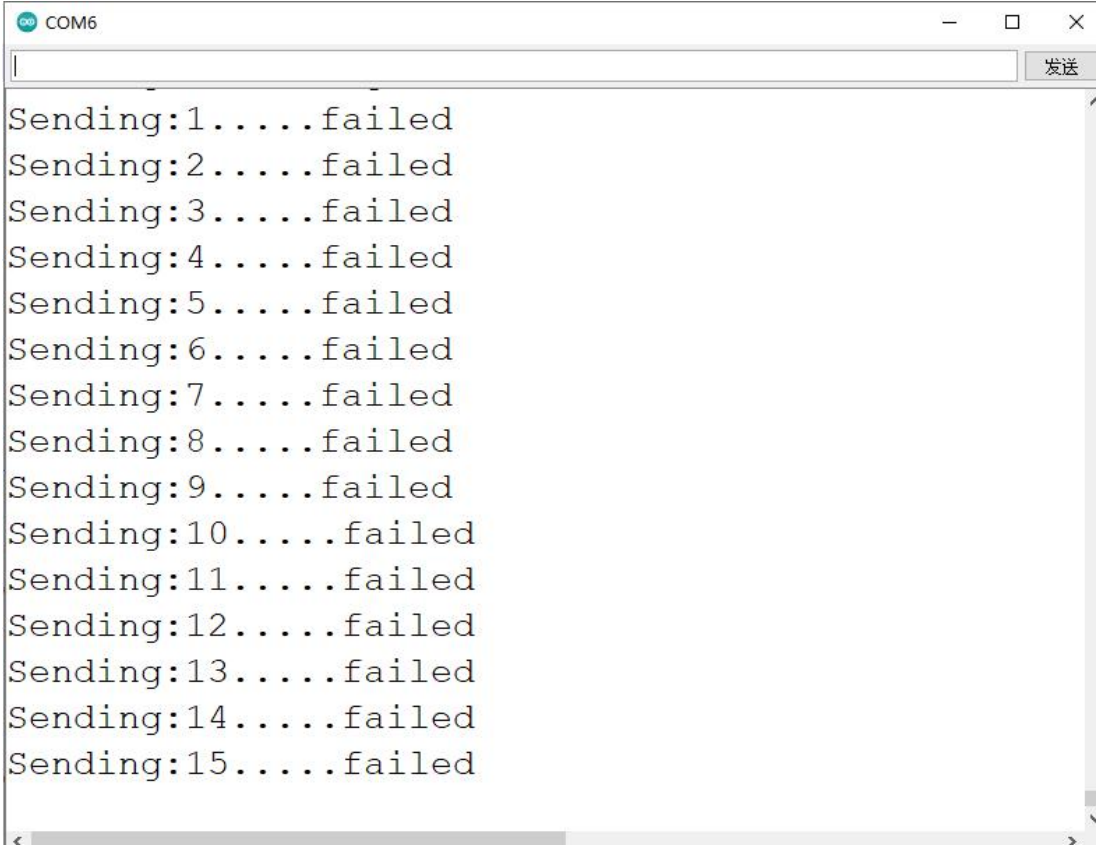
串口连续打印发送的数据“data”来下载程序。连接杜邦线缆后，可以打开 IDE 的调试窗口查看。请特别注意波特率的设置。默认波特率为 9600，需要修改为 57600。

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

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

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

运行结果。



```
COM6
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
```

# 第 17 课 LED 红外控制 LED

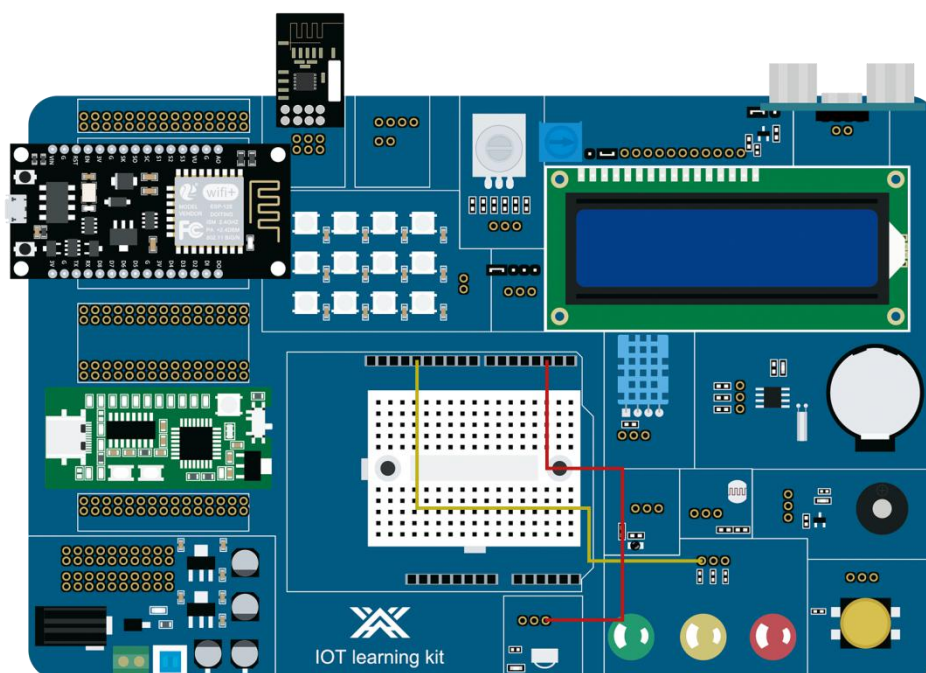
## 17.1 概述

通过本项目，您将学习使用红外遥控器远程打开 LED。下载完程序后，连接杜邦电缆，按遥控器上对应的的按键键，即可开启和关闭 LED。

### 连接描述:

用杜邦电缆将单片机 D2 连接到 JP16 引脚“S”上，将 D13 连接到 JP12 的任意引脚上，连接成功。

## 17.2 接线示意图





## 17.3 代码讲解

函数的初始化和函数的函数脚定义已经在前面的课程中讨论过了,但它们都是相同的,所以我在这里不进行描述。我主要讲一下红外处理功能。当红外接收功能接收到键值时,进行判断。该键值可以通过串口打印出来,并在串口调试器中查看。当我们指定的键值被判断时,我们定义 LED 的状态为反转,这样 LED 就可以开或关。

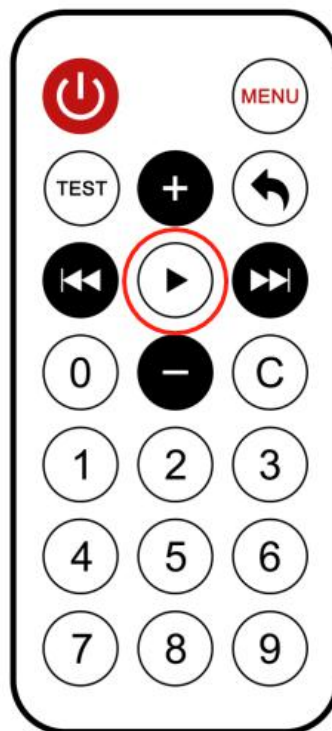
```
#include "IRremote.h"
#define HW 2
#define LED 13
int BIT_LED=0;
IRrecv irrecv(HW);
decode_results results;
void setup() {
    // put your setup code here, to run once:
    irrecv.enableIRIn();
    Serial.begin(9600);
    pinMode(LED,OUTPUT);
}

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);
  }
}
```

遥控器红框内按钮已在程序中设定具有指定功能。



## 第 18 课 红外控制 RGB

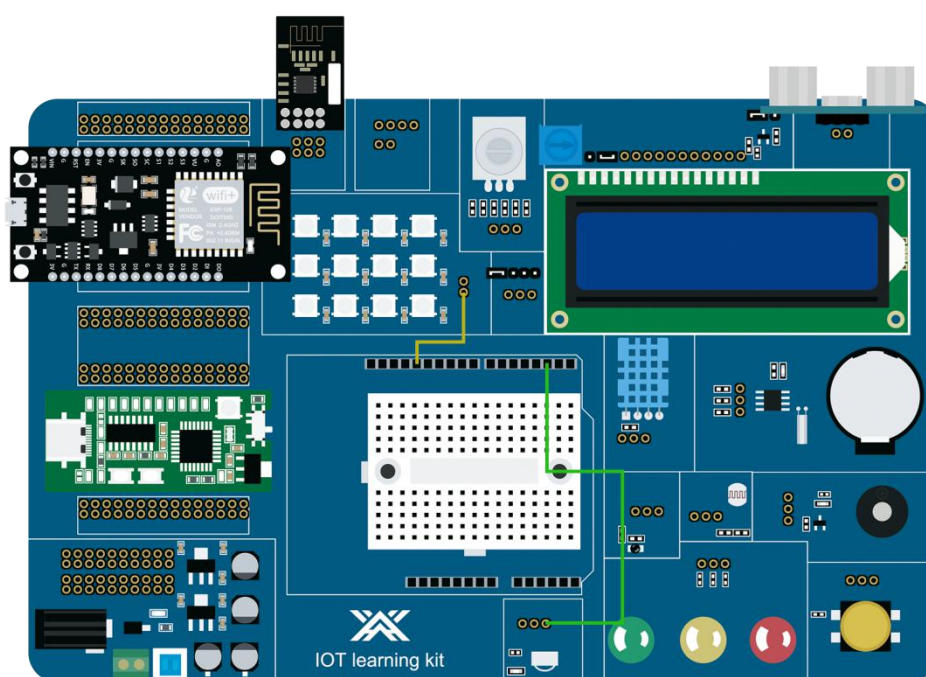
### 18.1 概述

通过这个项目，您将学习使用红外遥控器远程打开 RGB。下载程序后，连接杜邦电缆，按遥控器上的 1、2、3、4、5、6 显示红、绿、蓝、黄、白五种颜色，最后关闭 RGB。

连接描述：

用杜邦线将单片机 D2 连接到 JP16 引脚“S”上，将 D13 连接到 JP15 的“DIN”引脚上，连接成功。

### 18.2 接线示意图



### 18.3 代码讲解

函数的初始化和函数的函数脚定义已经在前面的课程中讨论过，但它们是相同的，所以我在这里不再重复它们。我主要讲的是红外处理功能。当红外接收功能接收到键值时，进行判断。该键值可以通过串口打印出来，并在串口调试器中查看。当判断我们指定的键值时，我们定义每个数字对应的 RGB 颜色。当按相

应的数字时，就会点亮相应的颜色。这些键值也可以自定义或使用自己定义的数字来控制 RGB 的颜色或 RGB 的数量。

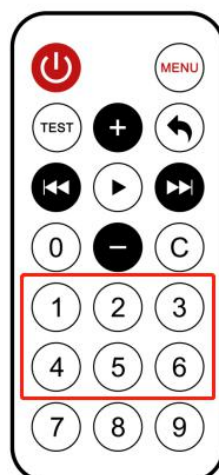
```
#include "IRremote.h"
#include <Adafruit_NeoPixel.h>
#define NUMPIXELS 12 // Number of 2812 lamps
#define RGB_PIN 13 // 2812 pin definition
#define HW 2
IRrecv irrecv(HW);
decode_results results;

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==0XFF5AA5) //6
    {
        for(int i=0;i<12;i++)
        {
            pixels.setPixelColor(i, pixels.Color(0,0, 0)); //Receive digital 6RGB light off
            pixels.show();
        }
    }
}
```

遥控器红框内按钮已在程序中设定具有指定功能。



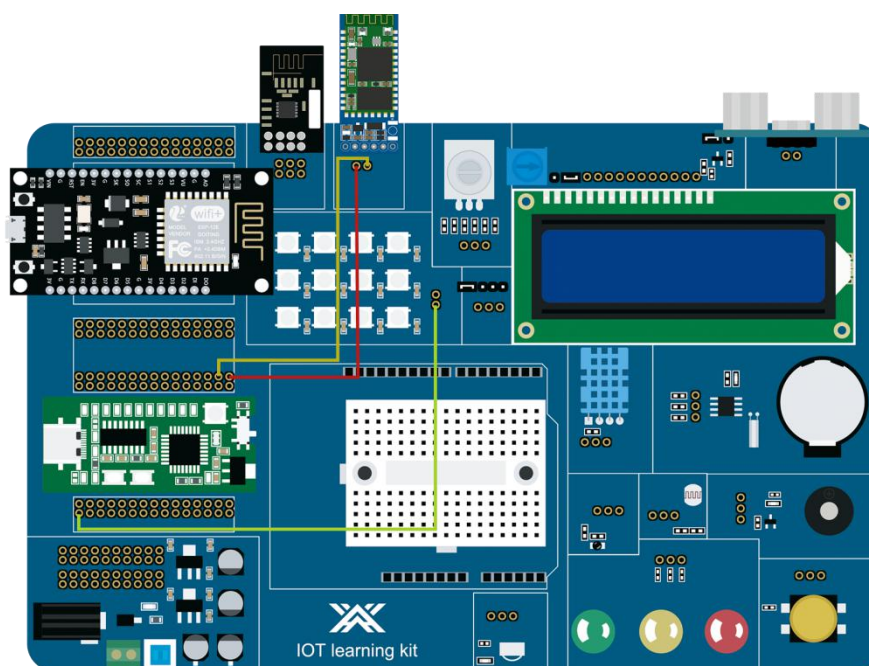
## 第 19 课 蓝牙控制 RGB

### 19.1 概述

通过这个项目，你将学会使用手机 APP 使用蓝牙连接板上的蓝牙模块来实现 RGB 的远程开启。下载程序后，连接杜邦线，连接蓝牙，使用手机 APP 连接蓝牙，通过 APP 键盘发送 R、G、B、Y、W、O，显示红、绿、蓝、黄、白五种颜色。最后一个关闭 RGB。注意，这里的字符都是大写的。

### 19.2 连接描述

将单片机的 D13 用杜邦线连接到 JP15 的“DIN”引脚，将单片机上的 TX 连接到 JP32 上的 RX，将单片机上的 RX 连接到 JP32 上的 TX。所有连接完成后，连接成功。项目接线图：



### 19.3 代码讲解

函数的初始化和函数的函数脚定义已经在前面的课程中讨论过，但它们都是相同的，所以我在这里不再重复它们。我主要讲蓝牙的处理功能。当蓝牙接收功能接收到相应的字符时，就会做出判断。这个字符可以通过串口打印出来，并在



串口调试器中查看。当判断我们指定的字符值时，我们定义每个数字对应的 RGB 颜色。当按下相应的数字时，相应的颜色就会亮起来。这些按键也可以定制或使用自定义数字来控制 RGB 的颜色或 RGB 的数量。

```
#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();
            }
        }
    }
}
```



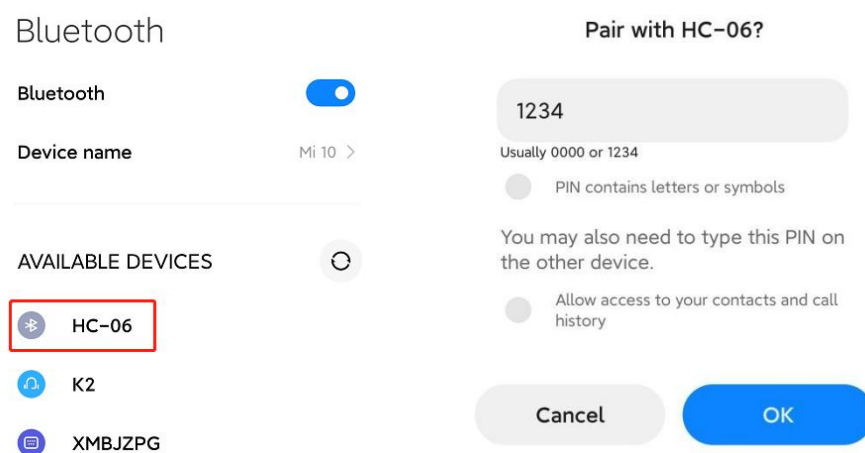
```

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();
    }
}
}
}

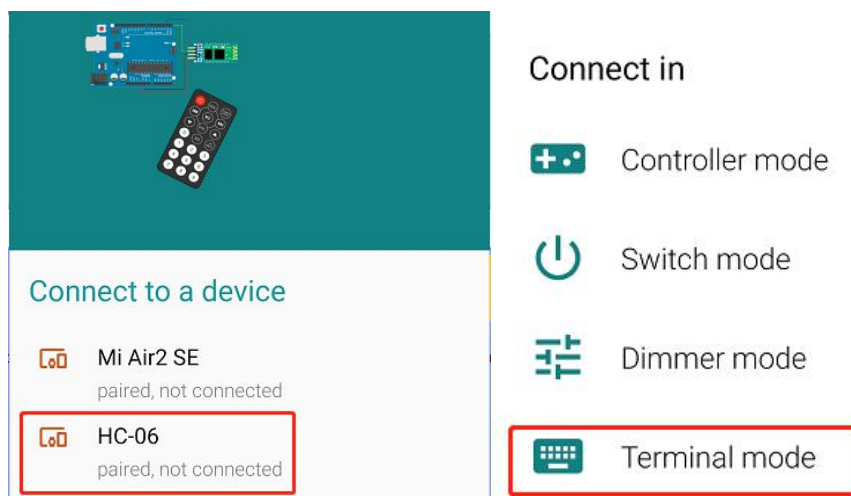
```

## 19.4 蓝牙遥控

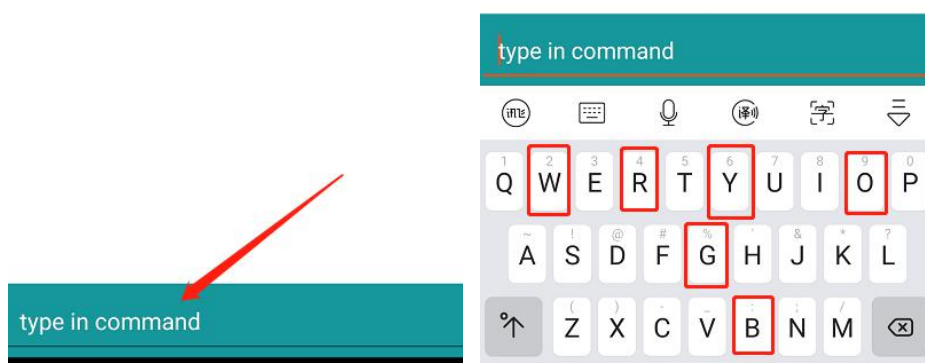
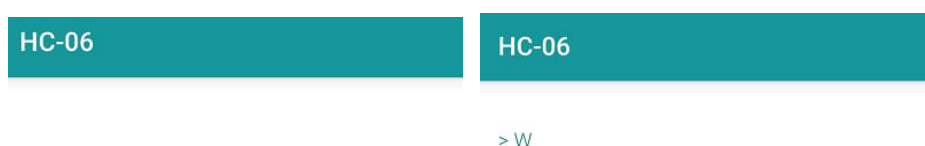
配备蓝牙串口助手，远程控制汽车；首先打开蓝牙，搜索蓝牙设备，找到 hc-06 连接。连接配对密码默认为“1234”。



打开蓝牙助手，找到刚刚配置的蓝牙模块名称 hc-06。连接成功后，会出现远程控制主界面。



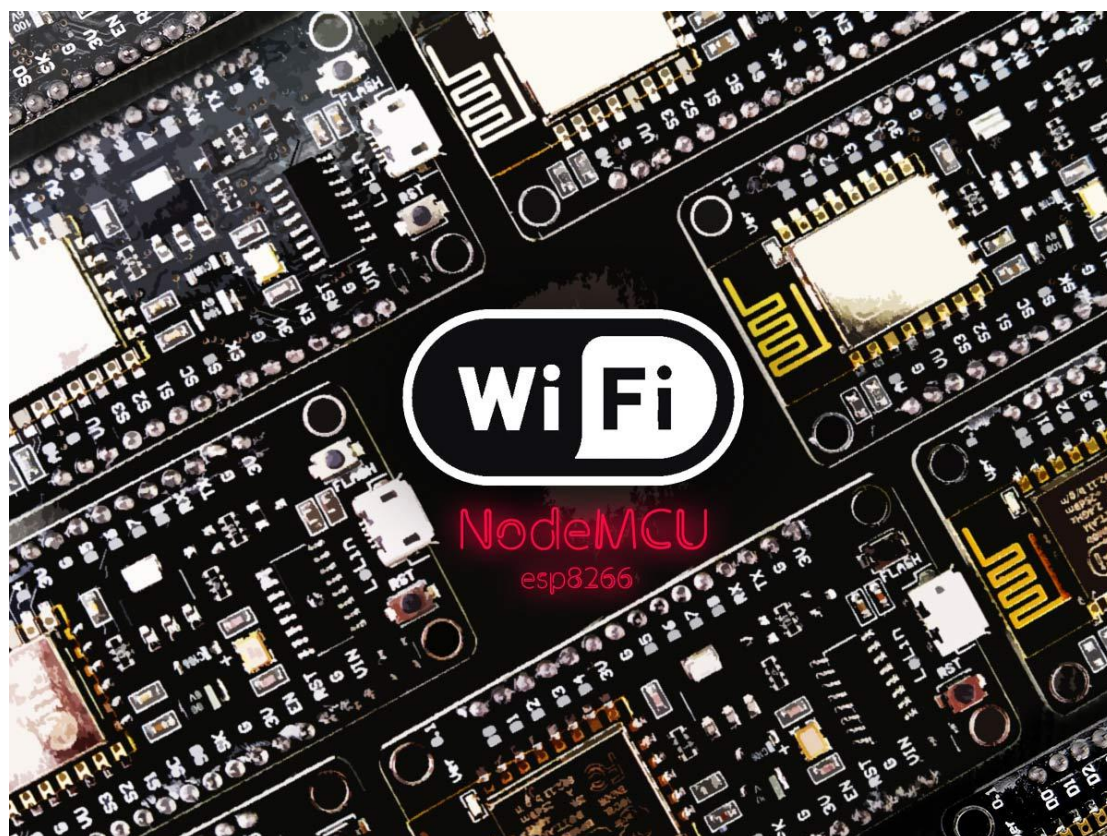
选择键盘模式，输入后可以输入我们指定的字符。输入“R”后，系统会返回“OK”，表示我们的通信正常，你可以输入其他字符，显示不同的颜色。



## 第 20 课 ESP8266 开发板

### 20.1 简介:

ESP8266 是一款非常适合物联网和家庭自动化项目的 Wi-Fi 模块。本文是 ESP8266 开发板的入门指南。

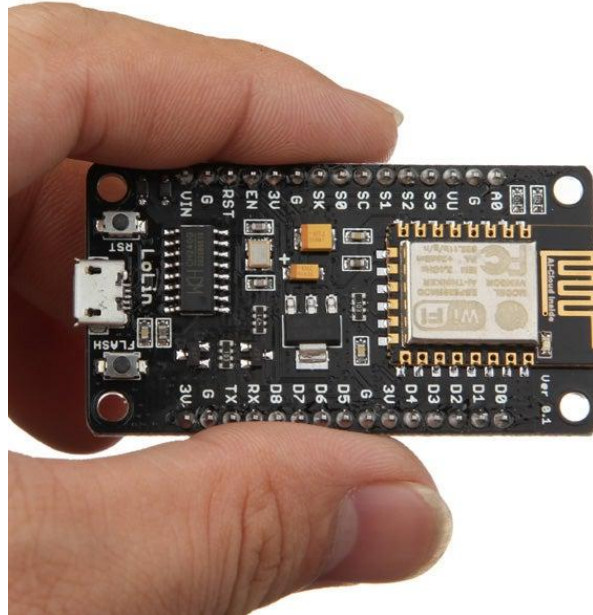


### 20.2 ESP8266 规格

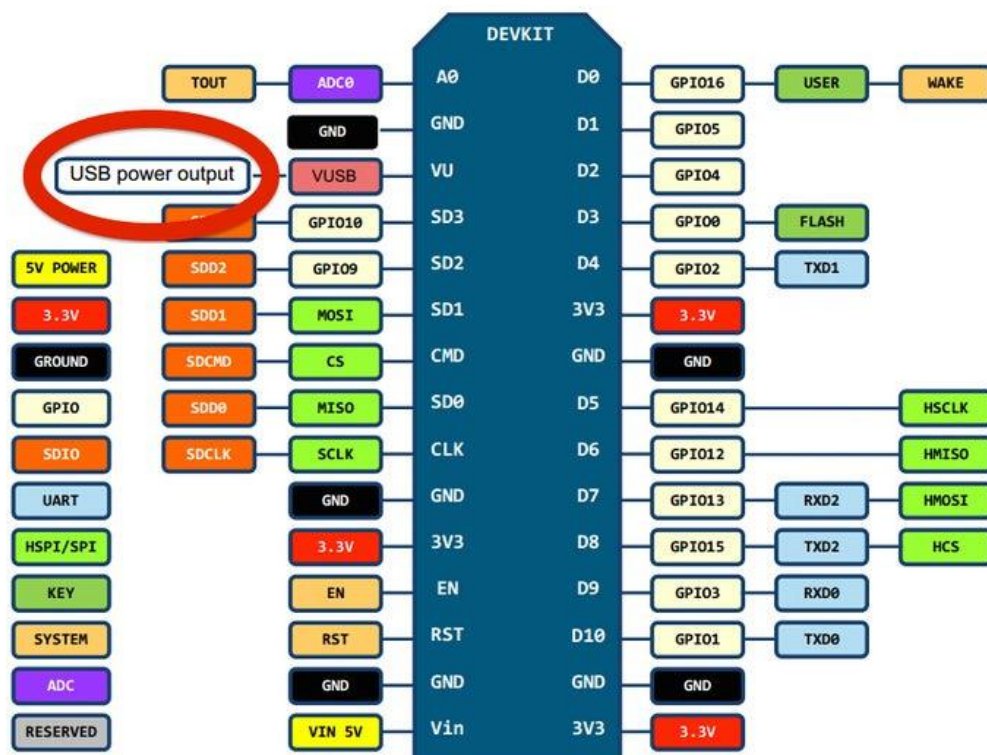
- 11 b/g/n 协议
- Wi-Fi Direct (P2P)、软 AP
- 集成 TCP/IP 协议栈
- 内置低功耗 32 位 CPU
- SDIO 2.0、SPI、UART

## 20.3 ESP8266 版本

ESP8266 有多种版本（如下图所示）。在我们看来，ESP-12E 或通常称为 ESP-12E NodeMCU 套件是目前最实用的版本。



ESP8266 开发板引脚原理图：





## 20.4 NodeMCU 引脚排列外设

NodeMCU 外设包括

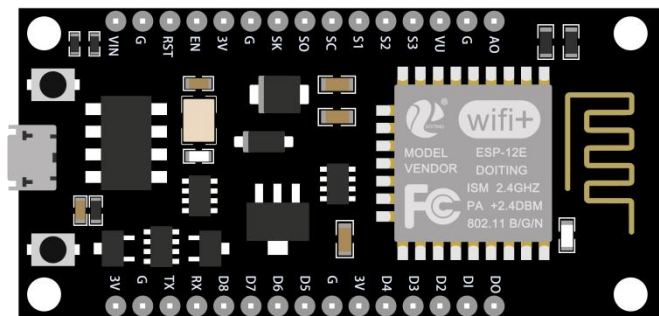
17 个通用输入输出引脚

SPI

I2C

串口

10 位 ADC



## 20.5 NODEMCU ESP8266 中使用的引脚是什么？

GPIO 编号与引脚图上的标签不匹配。比如 D1 对应 GPIO5，D2 对应 GPIO5

可以毫无问题地用作输入/输出的引脚

标记为 D1 的 GPIO5 通常用作 SCL (I2C)

标记为 D2 的 GPIO4 通常用作 SDA (I2C)

GPIO0 标记为 D3 连接到 FLASH 按钮，如果拉低启动失败

GPIO2 标记为 D4 连接到板载 LED，如果在启动时拉低 - 高，则启动失败

GPIO14 标记为 D5 SPI (SCLK)

GPIO12 标记为 D6 SPI (MISO)

GPIO13 标记为 D7 SPI (MOSI)

ADO 标记为 AO

以下是可以使用的引脚，但您需要注意，因为它们可能主要在启动时出现意外行为。

- **GPIO16** 在启动时 标记为 D0 HIGH 用于从深度睡眠中唤醒
- **GPIO15** 标记为 D8 拉至 GND: 如果我们拉高，则启动失败
- **GPIO3** 在 boo 标记为 RX HIGH

**GPIO1** 在启动时 标记为 TX 调试输出，如果拉低启动失败

---

## 注意:

建议将标记为 RX 和 ADO 的引脚用作输出，不建议将 TX 引脚用作输入

被称为 GPIO6 到 GPIO11 的引脚连接到 ESP8266 中的闪存芯片。因此，

不建议将这些引脚用作输入/输出功能。

如果要操作继电器，GPIO4 和 GPIO5 是最安全的 GPIO 使用引脚。



## 第 21 课 Arduino IDE 中安装 ESP8266 开发板

ESP8266 社区为 Arduino IDE 创建了一个附加组件，允许您使用 Arduino IDE 及其编程语言对 ESP8266 进行编程。

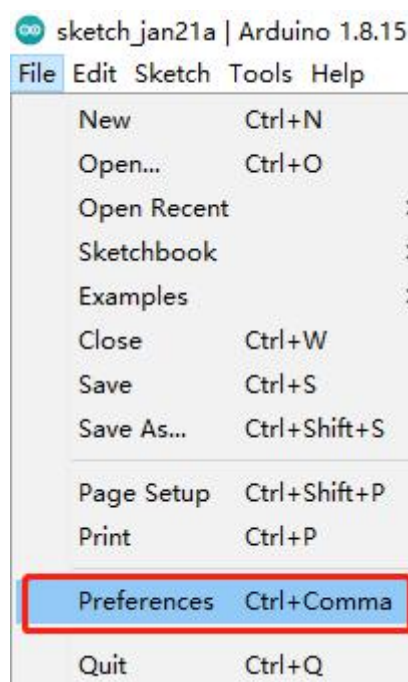
本教程展示了如何在 Arduino IDE 中安装 ESP8266 开发板，无论您使用的是 Windows、Mac OS X 还是 Linux。

在开始此安装过程之前，请确保您的计算机中安装了最新版本的 Arduino IDE。如果没有，请将其卸载并重新安装。否则，它可能无法工作。

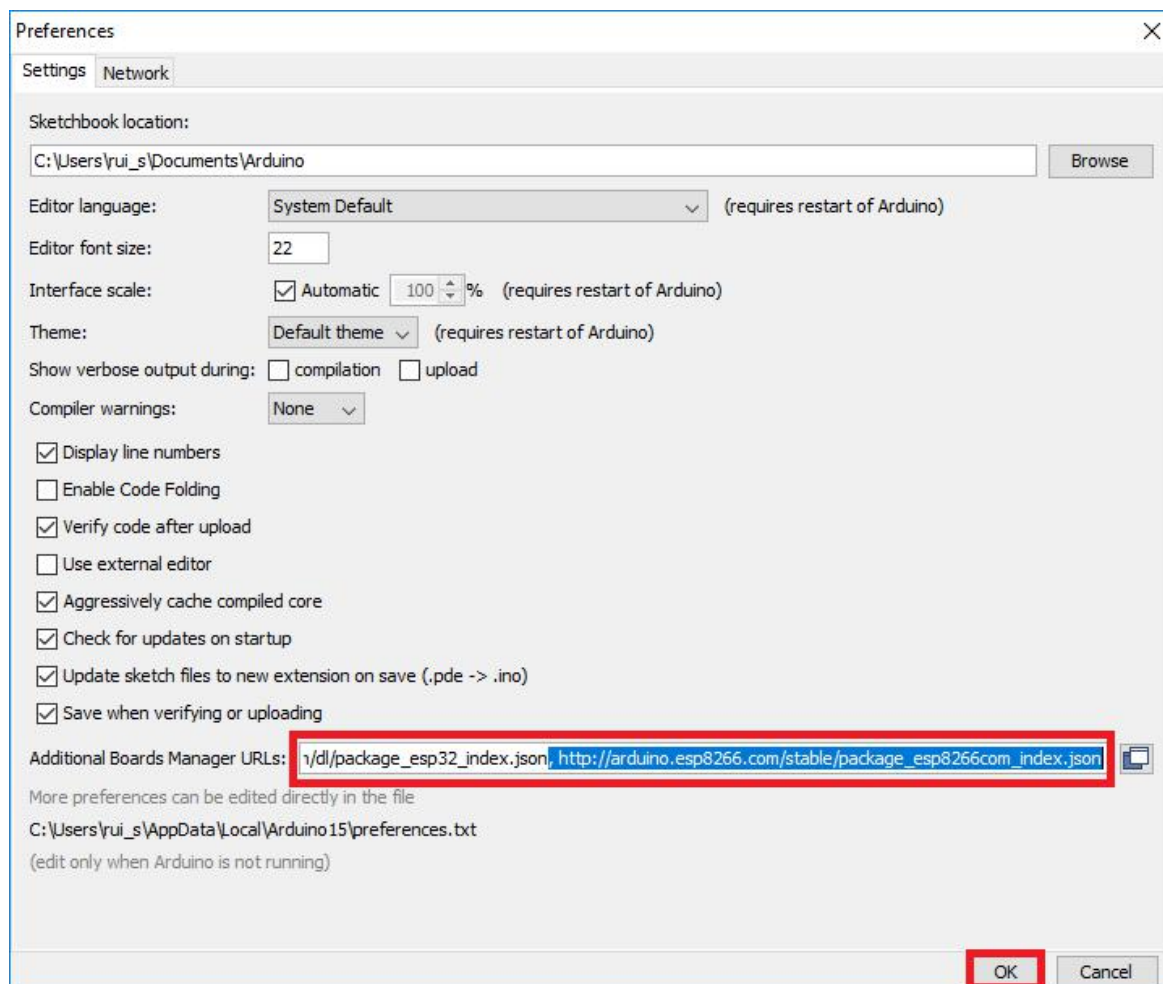
可以点击链接：<https://www.arduino.cc/en/software> 安装最新的 Arduino IDE 软件。

### 21.1 在 Arduino IDE 中安装 ESP8266 插件

要在您的 Arduino IDE 中安装 ESP8266 板，请按照以下说明操作：在您的 Arduino IDE 中，转到文件>首选项。



在“Additional Boards Manager URLs”字段中输入  
[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json), 如下图所示。  
 然后, 单击“确定”按钮:

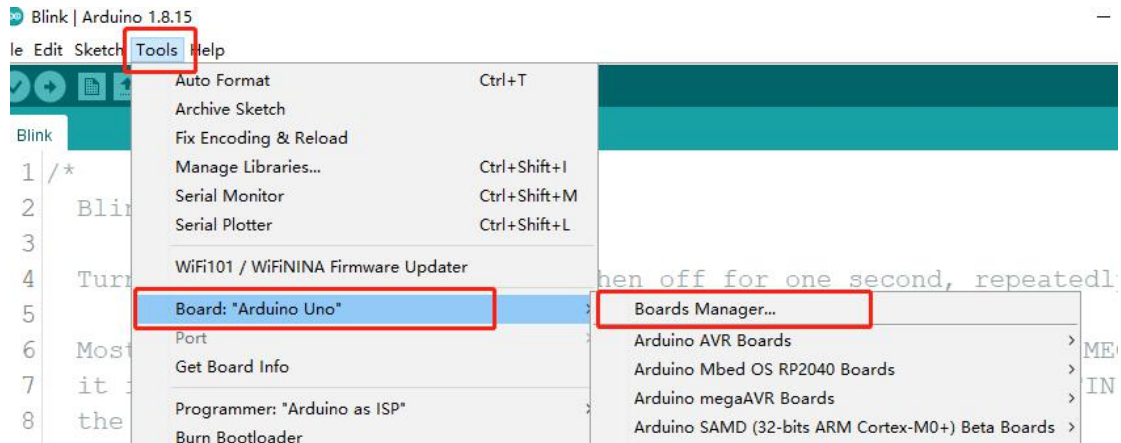


注意: 如果您已经有 ESP32 板 URL, 您可以使用逗号分隔 URL, 如下所示:

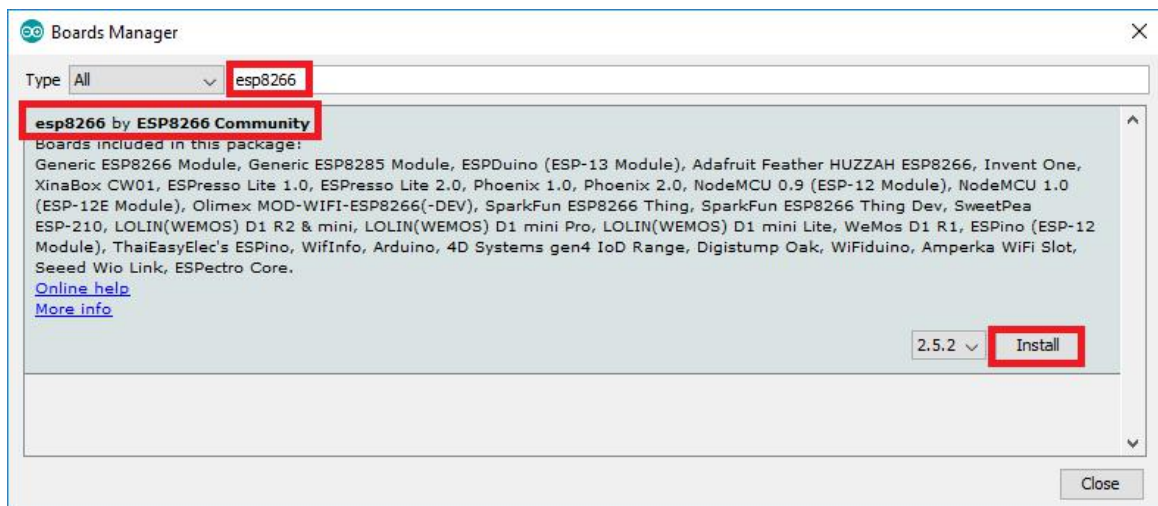
[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json),

[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

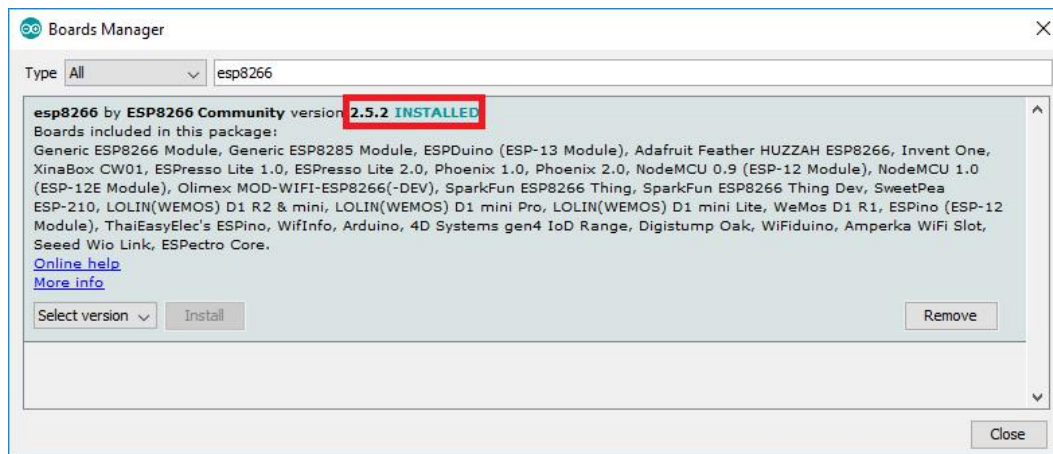
打开板管理器。转到工具>板>板管理器...



搜索 ESP8266 并按“ESP8266 by ESP8266 Community”的安装按钮：



等待几秒钟后安装。



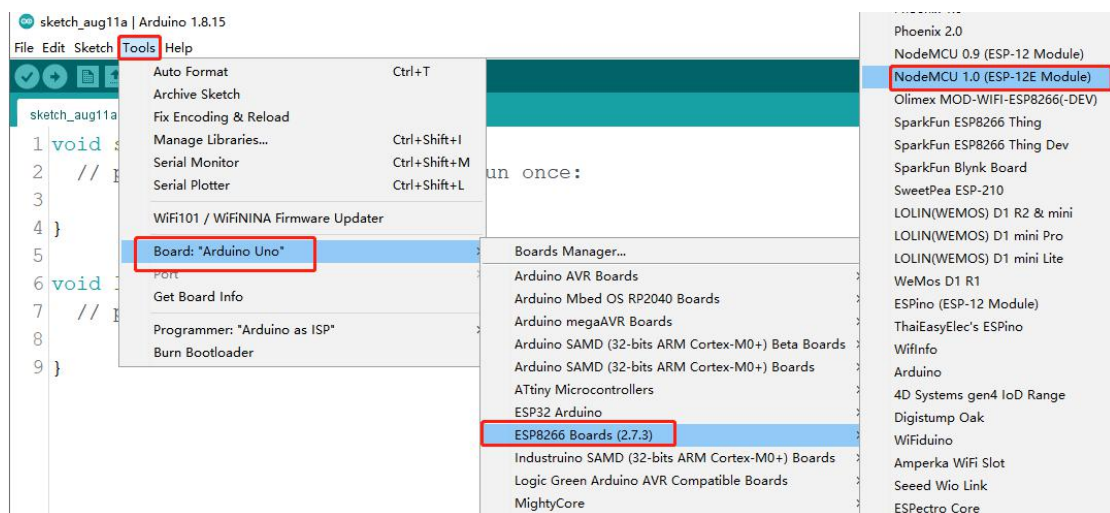
## 21.2 测试安装

为了测试 ESP8266 插件安装，让我们看看我们是否可以使用 Arduino 编程语言通过 ESP8266 使 LED 闪烁。

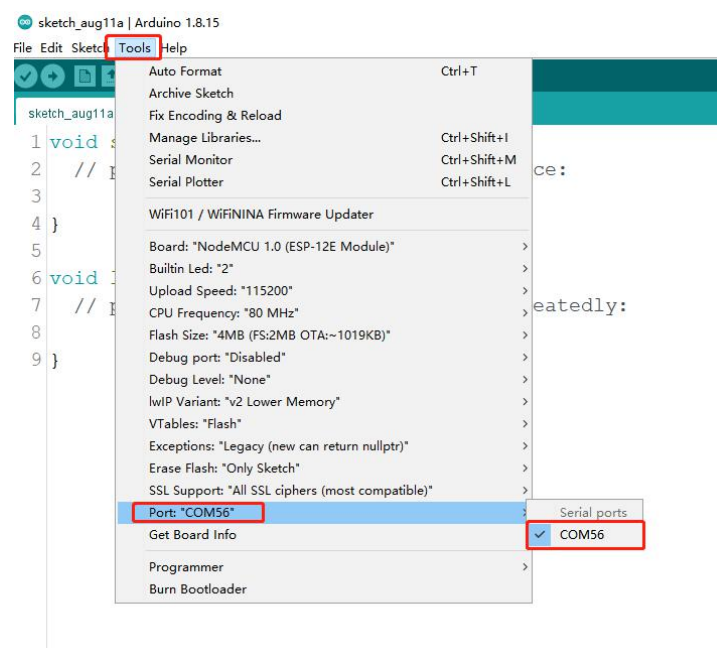
上传草图

将草图上传到 ESP-12E

将您的开发板插入您的计算机。确保您选择了正确的电路板：



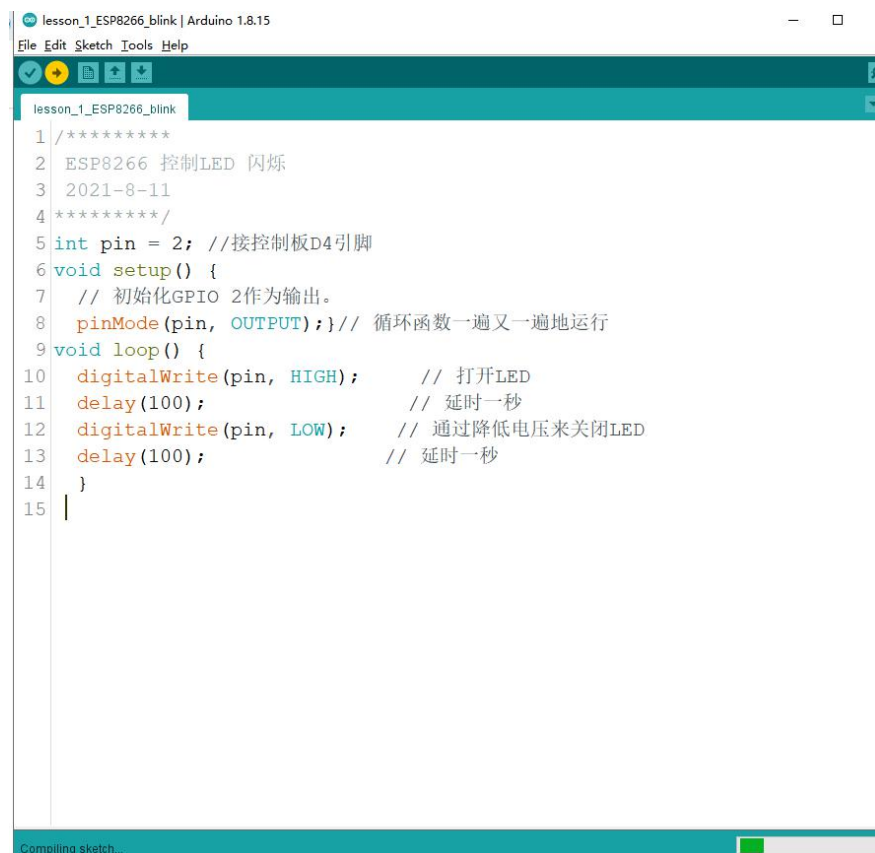
您还需要选择端口：



然后，复制提供的代码：

```
int pin = 2; void setup() {  
    // 初始化 GPIO 2 作为输出。  
    pinMode(pin, OUTPUT); } // 循环函数一遍又一遍地运行  
void loop() {  
    digitalWrite(pin, HIGH); // 打开 LED  
    delay(1000); // 延时一秒  
    digitalWrite(pin, LOW); // 通过降低电压来关闭 LED  
    delay(1000); // 延时一秒  
}
```

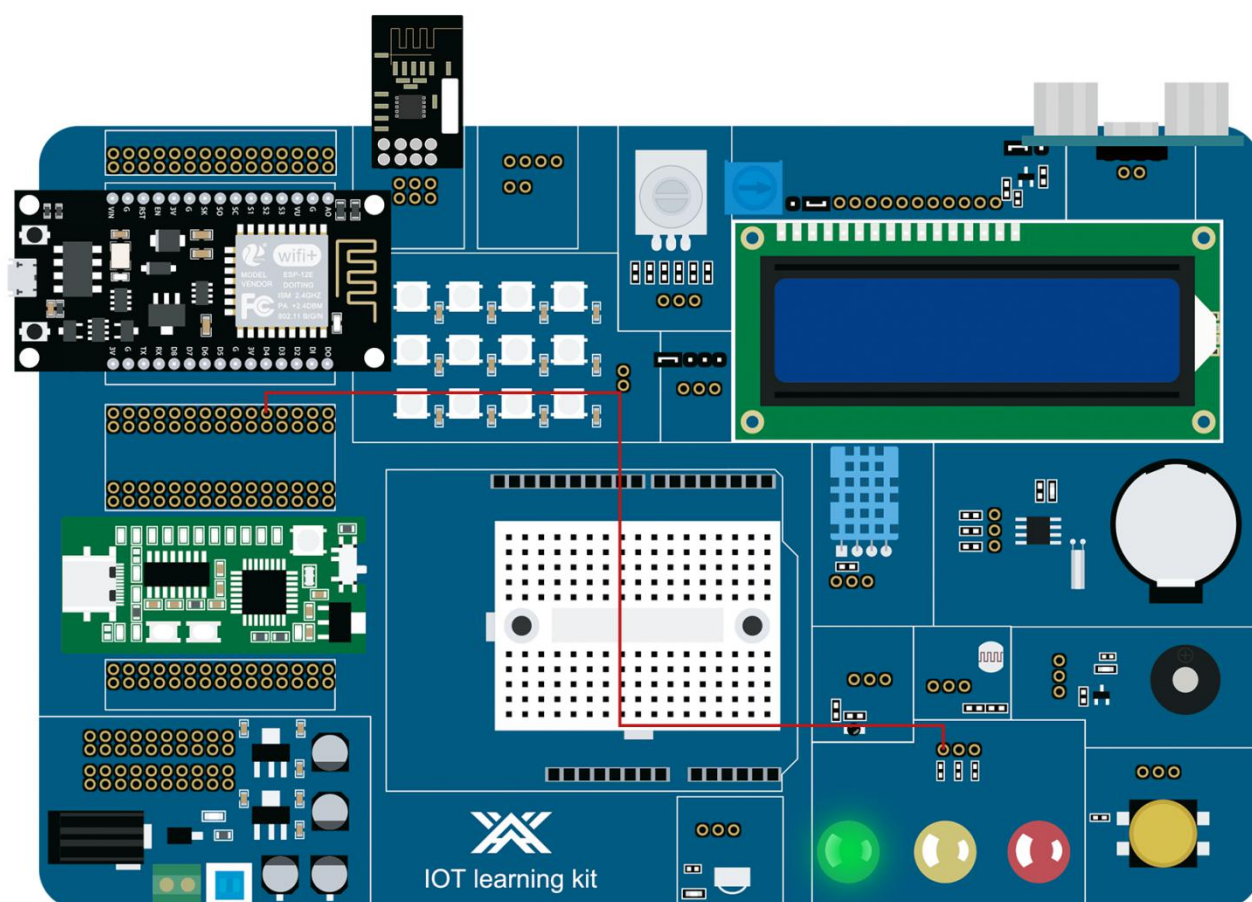
单击 **Arduino IDE** 中的“上传”按钮并等待几秒钟，直到您看到“完成上传”消息。在左下角。





### 21.3 接线图:

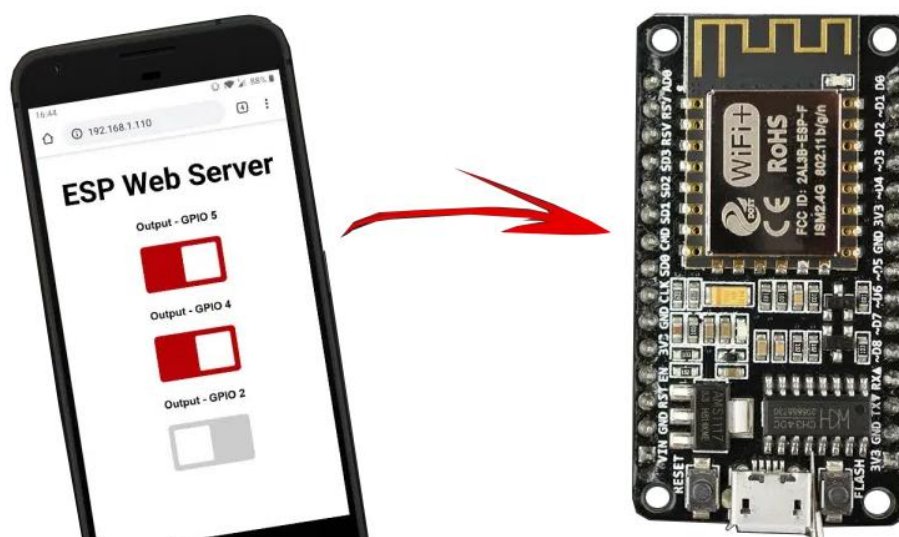
D4 接到红绿灯模块中绿色 LED 引脚上，LED 出现闪烁现象。





## 第 22 课 ESP8266 NodeMCU WiFi 控制红绿灯模块

在本教程中，您将学习如何使用 ESP8266 NodeMCU 板构建异步 Web 服务器以控制其输出。该板将使用 Arduino IDE 进行编程，我们将使用 ESPAsyncWebServer 库。

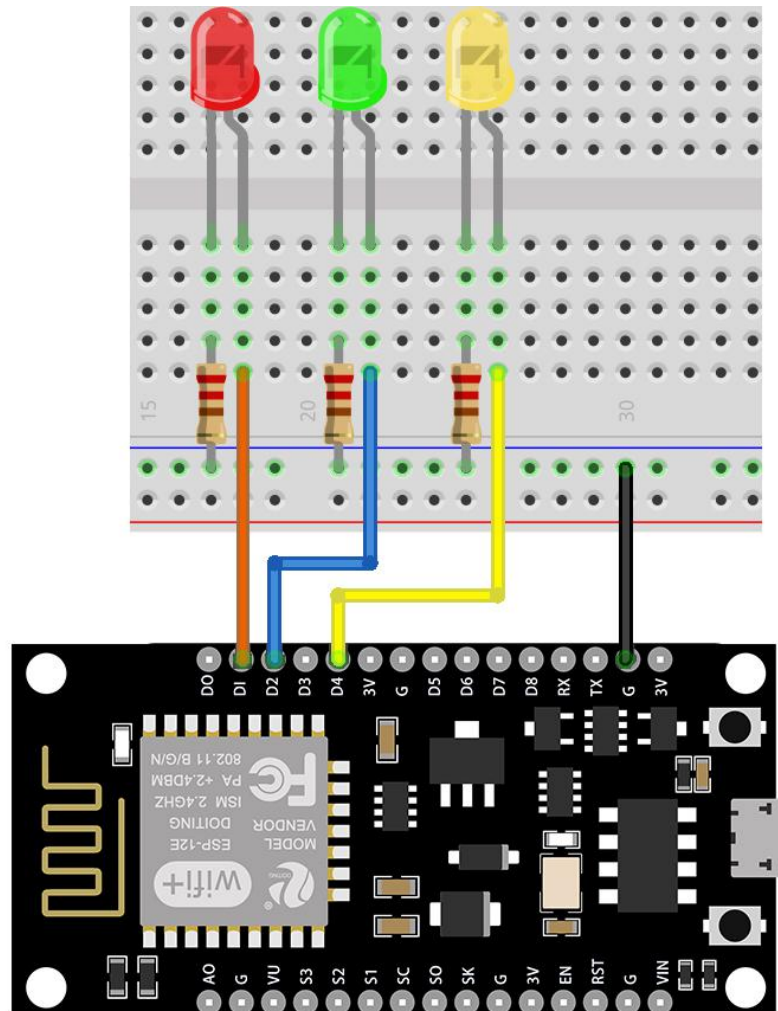


### 22.1 异步网络服务器

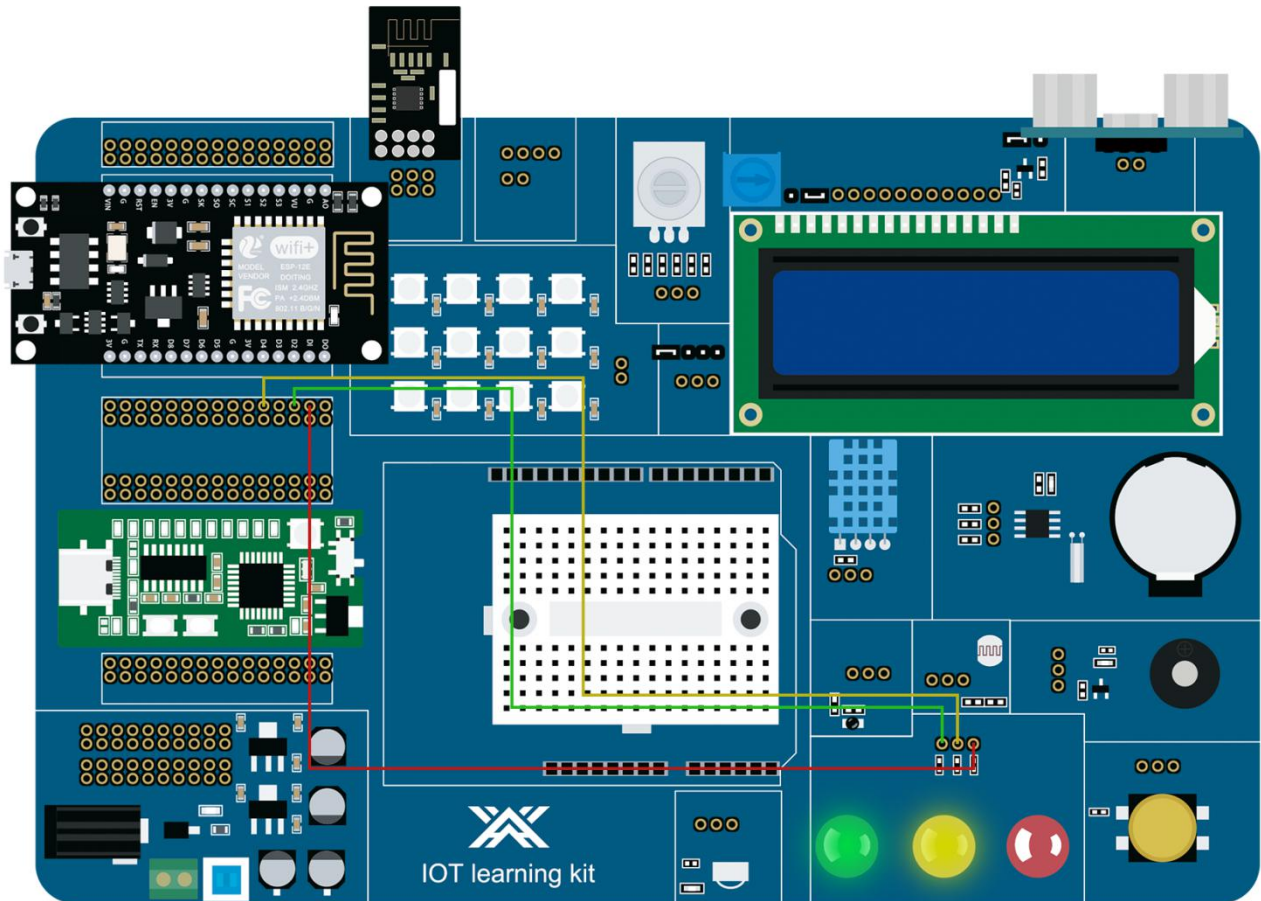
为了构建 Web 服务器，我们将使用 [ESPAsyncWebServer](#) 库，它提供了一种构建异步 Web 服务器的简单方法。

### 22.2 示意图：

在继续编写代码之前，将 3 个 LED 连接到 ESP8266。我们将 LED 连接到 GPIO 5、4 和 2。



## 22.3 接线图:



安装库 - ESP 异步 Web 服务器

要构建 Web 服务器，您需要安装以下库。

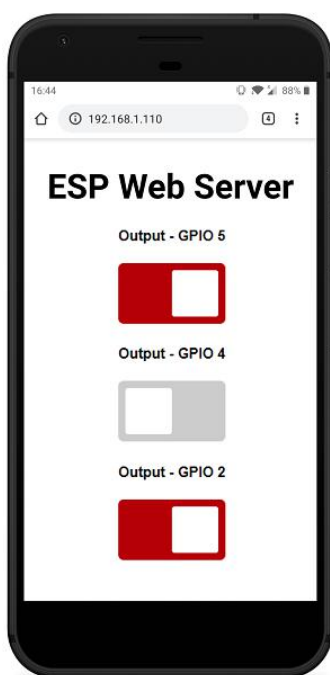
- **ESPAsyncWebServer**

## ▪ ESPAsyncTCP

这些库无法通过 **Arduino** 库管理器安装，因此您需要将库文件复制到 **Arduino** 安装库文件夹。或者，在您的 **Arduino IDE** 中，您可以转到 **Sketch > Include Library > Add .zip Library** 并选择您刚刚下载的库。

### 项目概况

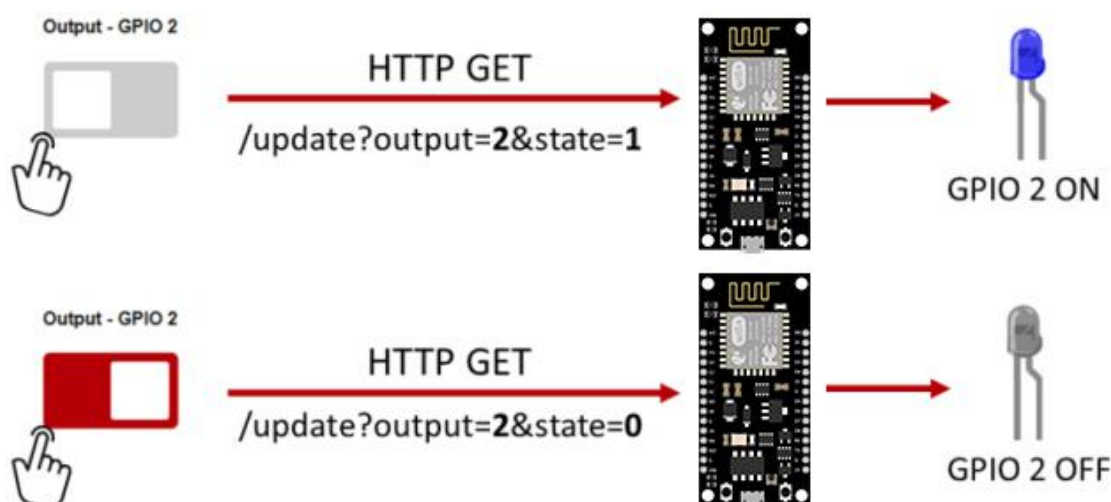
为了更好地理解代码，让我们看看 **Web** 服务器是如何工作的。



**Web** 服务器包含一个标题“**ESP Web Server**”和三个按钮（切换开关）来控制三个输出。每个滑块按钮都有一个标签，指示 **GPIO** 输出引脚。您可以轻松删除/添加更多输出。

当滑块为红色时，表示输出已打开（其状态为 HIGH）。如果您切换滑块，它将关闭输出（将状态更改为 LOW）。

当滑块为灰色时，表示输出关闭（其状态为 LOW）。如果您切换滑块，它会打开输出（将状态更改为 HIGH）。



让我们看看切换按钮时会发生什么。我们将看到 GPIO 2 的示例。其他按钮的工作方式类似。

1.在第一种情况下，您切换按钮以打开 GPIO 2。发生这种情况时，它会在/更新？输出= 2 &状态= 1 网址。根据该 URL，我们将 GPIO 2 的状态更改为 1 (HIGH) 并打开 LED。

2.在第二个示例中，当您切换按钮以关闭 GPIO 2 时。发生这种情况时，在/更新？输出= 2 &状态= 0 网址。根据该 URL，我们将 GPIO 2 的状态更改为 0 (LOW) 并关闭 LED。

## 22.4 ESP 异步 Web 服务器的代码

```
// 需要安装的库文件

#include <ESP8266WiFi.h>
```

```
#include <ESPAsyncTCP.h>

#include <ESPAsyncWebServer.h> // 用您的网络凭据替换

const char* ssid = "REPLACE_WITH_YOUR_SSID"; // 输入您的 wifi 名称

const char* password = "REPLACE_WITH_YOUR_PASSWORD"; // 输入 wifi 密码

const char* PARAM_INPUT_1 = "output";

const char* PARAM_INPUT_2 = "state";

// 在端口 80 上创建 AsyncWebServer 对象

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>

    %BUTTONPLACEHOLDER%

<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";// 将网页中的占位符替换为按钮部分

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(){

    // 用于调试的串口

    Serial.begin(115200);
```

```
pinMode(5, OUTPUT);

digitalWrite(5, LOW);

pinMode(4, OUTPUT);

digitalWrite(4, LOW);

pinMode(2, OUTPUT);

digitalWrite(2, LOW);


// 连接到无线网络

WiFi.begin(ssid, password);

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

    delay(1000);

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

}

// 输出“ESP 本地 IP 地址”

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");

});

//启动伺服器

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

## 22.5 代码的工作原理

在本节中，我们将解释代码的工作原理。

## 导入库

首先，导入所需的库。你需要包括无线上网, ESP8266 同步网络服务器 和 ESP8266 同步 TCP 库。

```
#include <ESP8266WiFi.h>
```

```
#include <ESPAsyncTCP.h>
```

```
#include <ESPAsyncWebServer.h>
```

## 设置网络凭据

在以下变量中插入您的网络凭据，以便 ESP8266 可以连接到您的本地网络。

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

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

## 输入参数

为了检查通过 URL 传递的参数（GPIO 编号及其状态），我们创建了两个变量，一个用于输出，另一个用于状态。

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

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

请记住，ESP8266 接收这样的请求： /更新? 输出=2&状态=0

## AsyncWebServer object

创建一个 异步 Web 服务器 端口 80 上的对象。

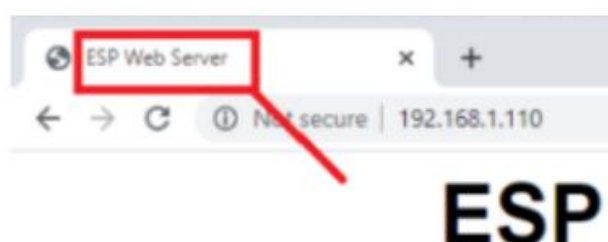
```
AsyncWebServer server(80);
```

## 构建网页

所有带有样式和 JavaScript 的 HTML 文本都存储在 index\_html 多变的。现在我们将浏览 HTML 文本并查看每个部分的作用。

标题位于<title>和</tile>标签内。标题就是它听起来的样子：您的文档的标题，它显示在您的 Web 浏览器的标题栏中。在这种情况下，它是“ESP Web 服务器”。

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



下面的<meta>标签使您的网页在任何浏览器（笔记本电脑、平板电脑或智能手机）中都能响应。

```
<meta name="viewport"
```

```
content="width=device-width, initial-scale=1">
```

下一行阻止对网站图标的请求。在这种情况下，我们没有网站图标。网站图标是显示在 Web 浏览器选项卡标题旁边的网站图标。如果我们不添加以下行，ESP 将在我们每次访问 Web 服务器时收到对 favicon 的请求。

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



在<style></style> 标签之间，我们添加了一些 CSS 来设置网页的样式。我们不会详细介绍这种 CSS 样式是如何工作的。

```
<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 正文

里面的<body> </ body>标签是我们添加的网页内容。

该 <h2> </ h2> 标签标题添加到网页。在这种情况下，“ESP Web Server”文本，但您可以添加任何其他文本。

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

在标题之后，我们有按钮。按钮在网页上的显示方式（红色：如果 GPIO 开启；或灰色：如果 GPIO 关闭）取决于当前的 GPIO 状态。

当您访问 Web 服务器页面时，您希望它显示正确的当前 GPIO 状态。因此，我们将添加一个占位符，而不是添加 HTML 文本来构建按钮%按钮占位符%。当网页加载时，这个占位符将被实际的 HTML 文本替换，以构建具有正确状态的按钮。

```
%BUTTONPLACEHOLDER%
```

## JavaScript

然后，正如我们之前解释的那样，当您切换按钮时，有一些 JavaScript 负责发出 HTTP GET 请求。

```
<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>
```

这是发出请求的行：

```
if(element.checked){xhr.open("GET",
```

```
"/update?output="+element.id+"&state=1", true); }
```

元素.id 返回一个 HTML 元素的 id。每个按钮的 id 将是控制的 GPIO，我们将在下一节中看到：

GPIO 5 按钮 » element.id = 5

GPIO 4 按钮 » element.id = 4

GPIO 2 按钮 » element.id = 2

## 处理器

现在，我们需要创建处理器（）函数，用我们定义的内容替换 HTML 文本中的占位符。

当请求网页时，检查 HTML 是否有任何占位符。如果它发现%按钮占位符% 占位符，它返回 HTML 文本以创建按钮。

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

#### 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(); }
```

您可以轻松删除或添加更多行以创建更多按钮。

让我们来看看按钮是如何创建的。我们创建了一个名为的 String 变量纽扣包含用于构建按钮的 HTML 文本。我们将 HTML 文本与当前输出状态连接起来，这样切换按钮要么是灰色的，要么是红色的。当前输出状态由输出状态(<GPIO>) 函数（它接受 GPIO 编号作为参数）。见下文：

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

使用\ 以便我们可以在字符串中传递 “ ” 。

这 输出状态() 函数返回 “检查” 如果 GPIO 处于打开状态或字段为空 “ ” 如果 GPIO 关闭。

```
StringoutputState(intoutput)
```

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

```
{ return "checked"; }
```

```
else { return ""; }
```

```
}
```

因此，GPIO 2 打开时的 HTML 文本将是：

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

让我们把它分解成更小的部分来理解它是如何工作的。

在 HTML 中，切换开关是一种输入类型。所述<输入>标记指定的输入栏，其中用户可以输入数据。拨动开关是一个输入字段类型复选框。还有许多其他输入字段类型。

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

复选框可以被选中或不选中。检查时，您有以下内容：

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

这变化中是当我们更改元素（复选框）的值时发生的事件属性。每当您选中或取消选中切换开关时，它都会调用切换复选框（） 该特定元素 id 的 JavaScript 函数（this）。

This ID 为该 HTML 元素指定一个唯一的 id。id 允许我们使用 JavaScript 或 CSS 操作元素。

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

## setup()

在里面 setup()初始化监视器以进行调试。

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

使用 pinMode（）功能并在 ESP8266 首次启动时将它们设置为 LOW。如果您添加了更多 GPIO，请执行相同的过程。

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

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

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

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

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

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

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

连接到您的本地网络并打印 ESP8266 IP 地址。

```
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());
```

在里面 `setup()`，您需要处理 ESP8266 收到请求时发生的情况。正如我们之前看到的，您会收到此类请求：

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

因此，我们检查请求是否包含 `PARAM_INPUT1` 变量值（输出）和 `PARAM_INPUT2`（状态）并将相应的值保存在 `输入 1 消息` 和 `输入 2 消息` 变量。



```
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();
```

然后，我们控制对应的 GPIO 对应的状态（输入消息 1 变量保存 GPIO 编号和输入消息 2 保存状态 -0 或 1）

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

下面是处理 HTTP GET /update 请求的完整代码：

```
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");});
```

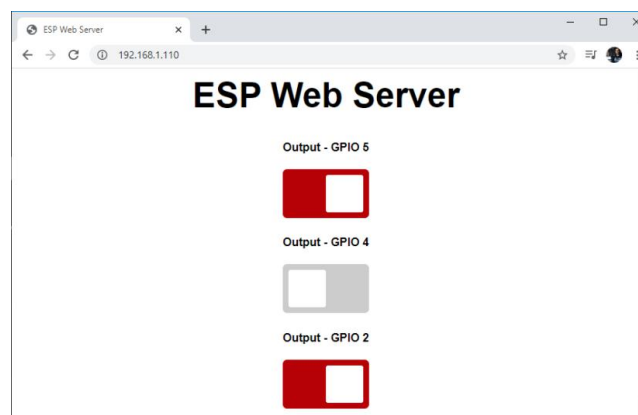
最后，启动服务器：

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

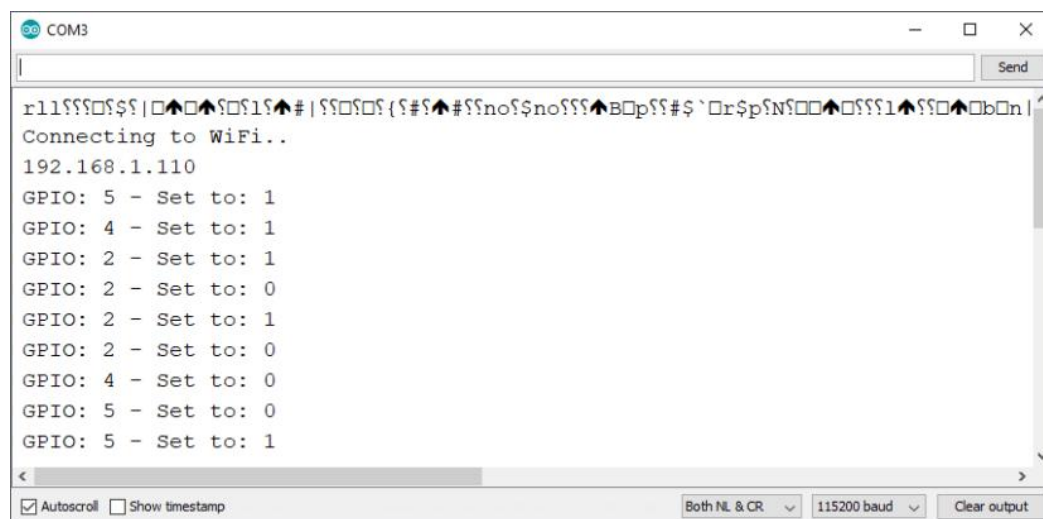
## 示范

将代码上传到 ESP8266 后，以 115200 的波特率打开串行监视器。按下板载 RST/EN 按钮。你应该得到它的 IP 地址。

打开浏览器并键入 ESP IP 地址。您将可以访问类似的网页。



按下切换按钮以控制 ESP GPIO。同时，您应该在串行监视器中收到以下消息，以帮助您调试代码。



您还可以从智能手机中的浏览器访问网络服务器。每当您打开 Web 服务器时，它都会显示当前的 GPIO 状态。红色表示 GPIO 开启，灰色表示 GPIO 关闭。



## 第 23 课 ESP8266 Node MCU 按键控制 LED

在本入门课程中，您将学习如何使用带有 Arduino IDE 的 ESP8266 NodeMCU 板读取按钮开关等数字输入和控制 LED 等数字输出。

### 23.1 ESP8266 NodeMCU 控制数字输出

首先，您需要将要控制的 GPIO 设置为 输出。使用 `pinMode()` 功能如下：

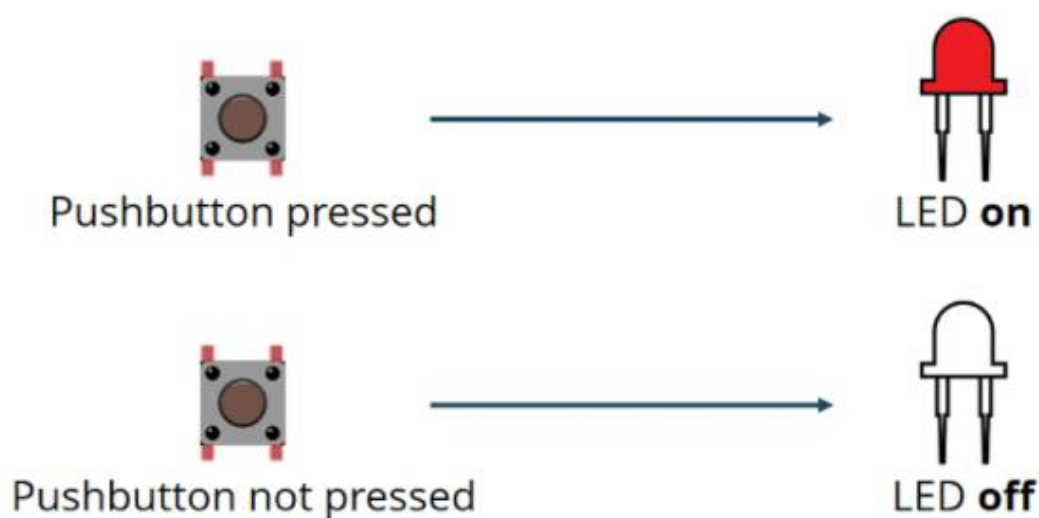
```
pinMode(GPIO, OUTPUT);
```

要读取数字输入，例如按钮，您可以使用 `digitalRead()` 函数，它接受您所指的 GPIO（整数）作为参数。

```
digitalRead(GPIO);
```

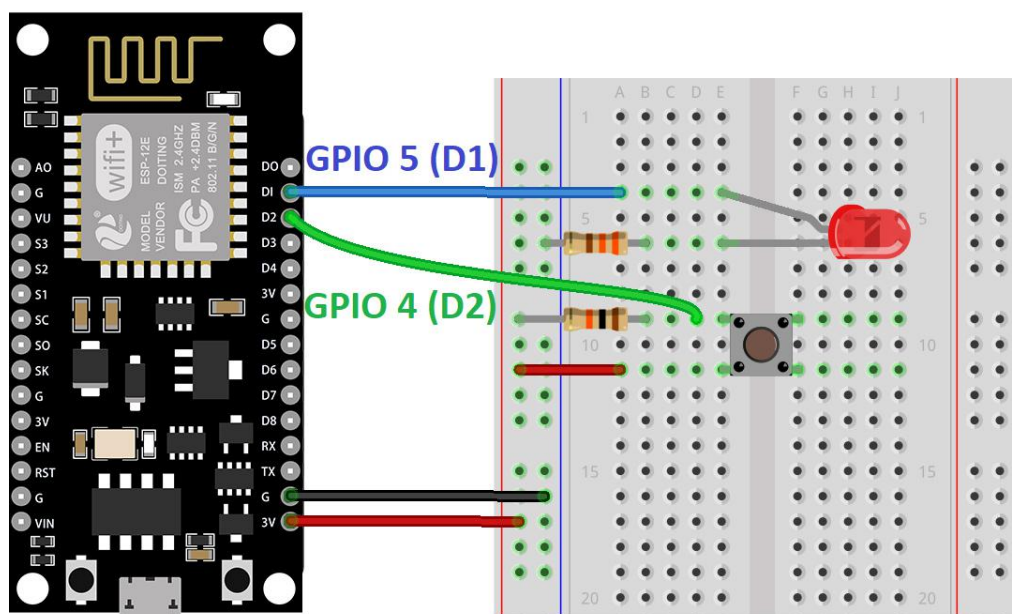
### 23.2 项目示例

为了向您展示如何使用数字输入和数字输出，我们将构建一个带有按钮和 LED 的简单项目示例。我们将读取按钮的状态并相应地点亮 LED，如下图所示。



### 23.3 接线示意图:

在继续之前，您需要组装一个带有 LED 和按钮的电路。我们将 LED 连接到通用输入输出 5（D1）和按钮 通用输入输出 4（D2）。



### 23.4 工作代码:

```
// 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 代码工作原理:

在以下两行中，您创建变量以分配引脚：

```
const int buttonPin = 4;

const int ledPin = 5;
```



该按钮连接到 通用输入输出口 4 并且 LED 连接到 通用输入输出口 5. 使用 Arduino IDE 搭配 ESP8266 时, 4 对应于通用输入输出口 4 和 5 对应于 通用输入输出口 5.

接下来, 创建一个变量来保存按钮状态。默认情况下, 它是 0 (未按下)。

```
int buttonState = 0;
```

在里面 `setup()`, 您将按钮初始化为 输入, 而 LED 作为 输出. 为此, 您使用 `pinMode()` 接受您所指的引脚和模式的函数: 输入 要么 输出.

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

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

在里面 `loop()` 是您读取按钮状态并相应地设置 LED 的地方。

在下一行中, 您读取按钮状态并将其保存在 按钮状态多变的。正如我们之前所见, 您使用 `digitalRead()` 功能。

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

下面的 if 语句, 检查按钮状态是否为 HIGH。如果是, 它使用 `digitalWrite()` 函数打开 LED, 该函数接受 `ledPin` 作为参数, 并将状态设置为 HIGH。

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

```
}
```

如果按钮状态不为“HIGH”，则将 LED 设置为关闭状态。只需将 LOW 设置为 `digitalWrite()` 函数中的第二个参数。

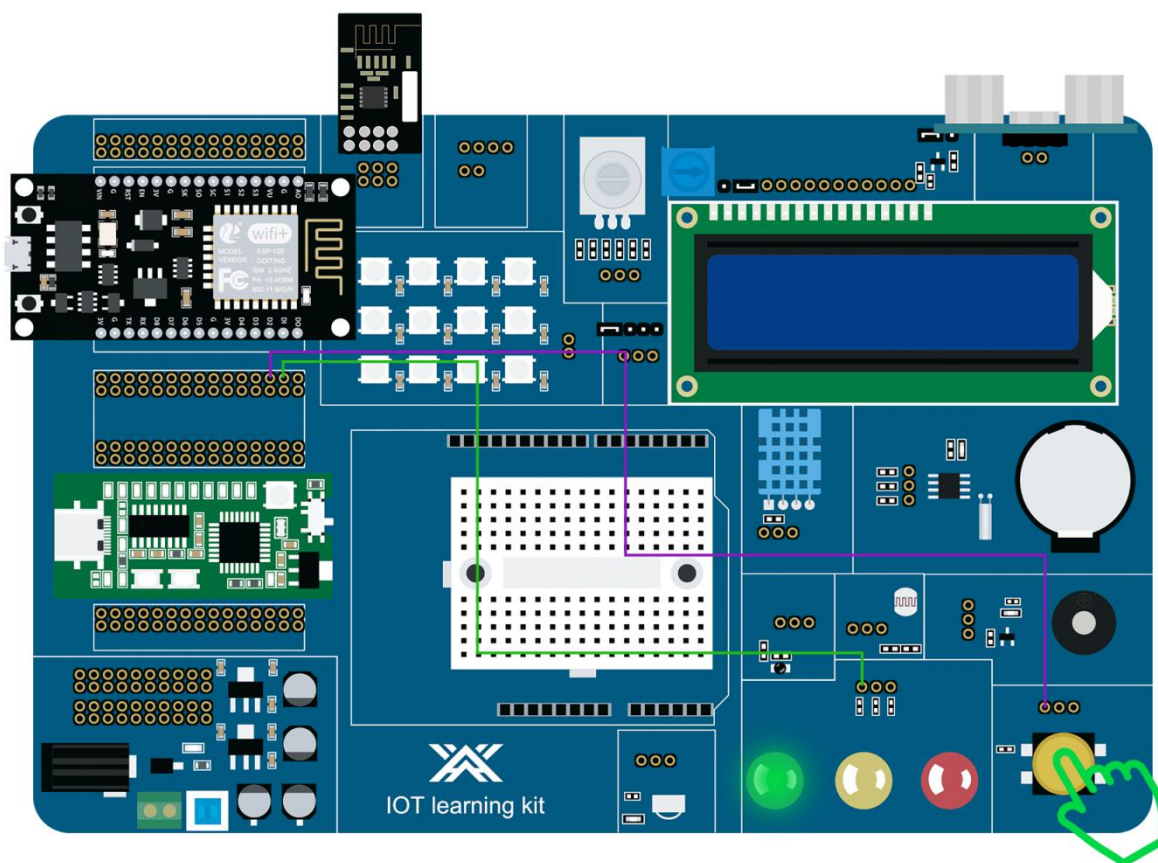
```
else { digitalWrite(ledPin, LOW); }
```

## 23.6 上传代码

在单击上传按钮之前，转到 `Tools` > `Board`，然后选择您正在使用的 board。NodeMCU 1.0 (ESP-12 E Module)。

转到工具>端口并选择 ESP8266 连接到的 COM 端口。然后，按上传按钮并等待“完成上传”消息。

## 23.7 实物图：



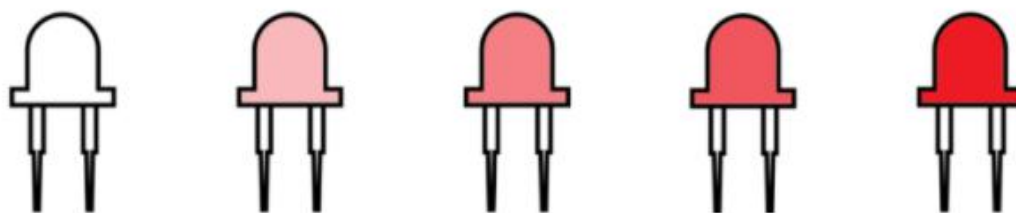
## 第 24 课 ESP8266 控制 LED 亮度（PWM）

本教程展示了如何使用 Arduino IDE 通过 ESP8266 NodeMCU 生成 PWM 信号。例如，我们将通过随时间改变占空比来调暗 LED 亮度。

### 24.1 ESP8266 NodeMCU PWM（脉宽调制）

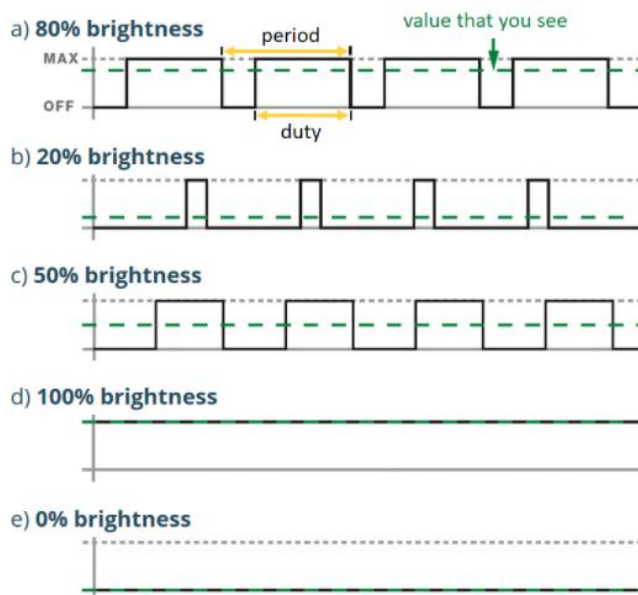
ESP8266 GPIO 可以设置为输出 0V 或 3.3V，但它们之间不能输出任何电压。但是，您可以使用脉宽调制（PWM）输出“假”中级电压，这就是您为此项目生成不同级别 LED 亮度的方式。

如果您在高电平和低电平之间快速交替 LED 的电压，您的眼睛就跟不上 LED 开关的速度；你只会看到亮度的一些渐变。



这基本上就是 PWM 的工作原理——通过产生一个以非常高的频率在 HIGH 和 LOW 之间变化的输出。

占空比是 LED 设置为高电平的时间段的一部分。下图说明了 PWM 的工作原理。



占空比为 50% 的 LED 亮度为 50%，占空比为 0 表示 LED 完全关闭，占空比为 100 表示 LED 完全打开。改变占空比是您产生不同亮度级别的方式。

## analogWrite()

要在给定引脚上产生 PWM 信号，请使用以下函数：

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

Pin: PWM 可用于引脚 0 到 16

value: 应该在 0 到 PWM 范围，默认为 1023。当值为 0 时，该引脚上的 PWM 被禁用。值 1023 对应于 100% 占空比

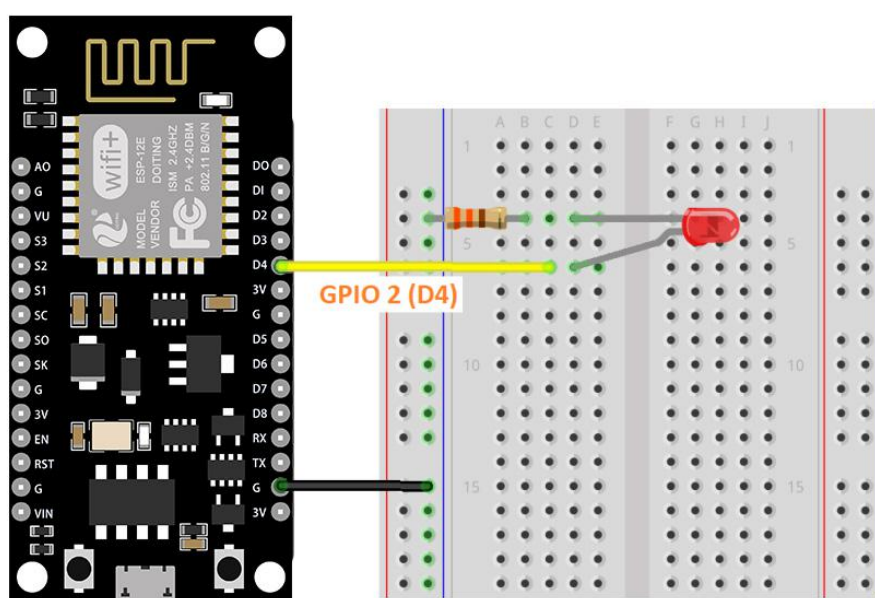
您可以通过调用更改 PWM 范围：

```
analogWriteRange(new_range);
```

默认情况下，ESP8266 PWM 频率为 1kHz。您可以通过以下方式更改 PWM 频率：

```
analogWriteFreq(new_frequency);
```

## 24.2 示意图:



ESP8266 NodeMCU PWM 代码:

```
const int ledPin = 2; //定义了 led 引脚 (D4)

void setup() {

}

void loop() {

    //提高 LED 亮度

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

        // 用 PWM 改变 LED 亮度

        analogWrite(ledPin, dutyCycle);

        delay(1);

    }

    // 降低 LED 亮度

    for(int dutyCycle = 1023; dutyCycle > 0; dutyCycle--){

        // 用 PWM 改变 LED 亮度

        analogWrite(ledPin, dutyCycle);

        delay(1);

    }

}}
```

代码工作原理:

首先定义引脚 LED 连接到。在这种情况下, LED 连接到通用输入输出口 2 (D4)。

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

在里面 `loop()`, 您可以在 0 和 1023 之间改变占空比以增加 LED 亮度。

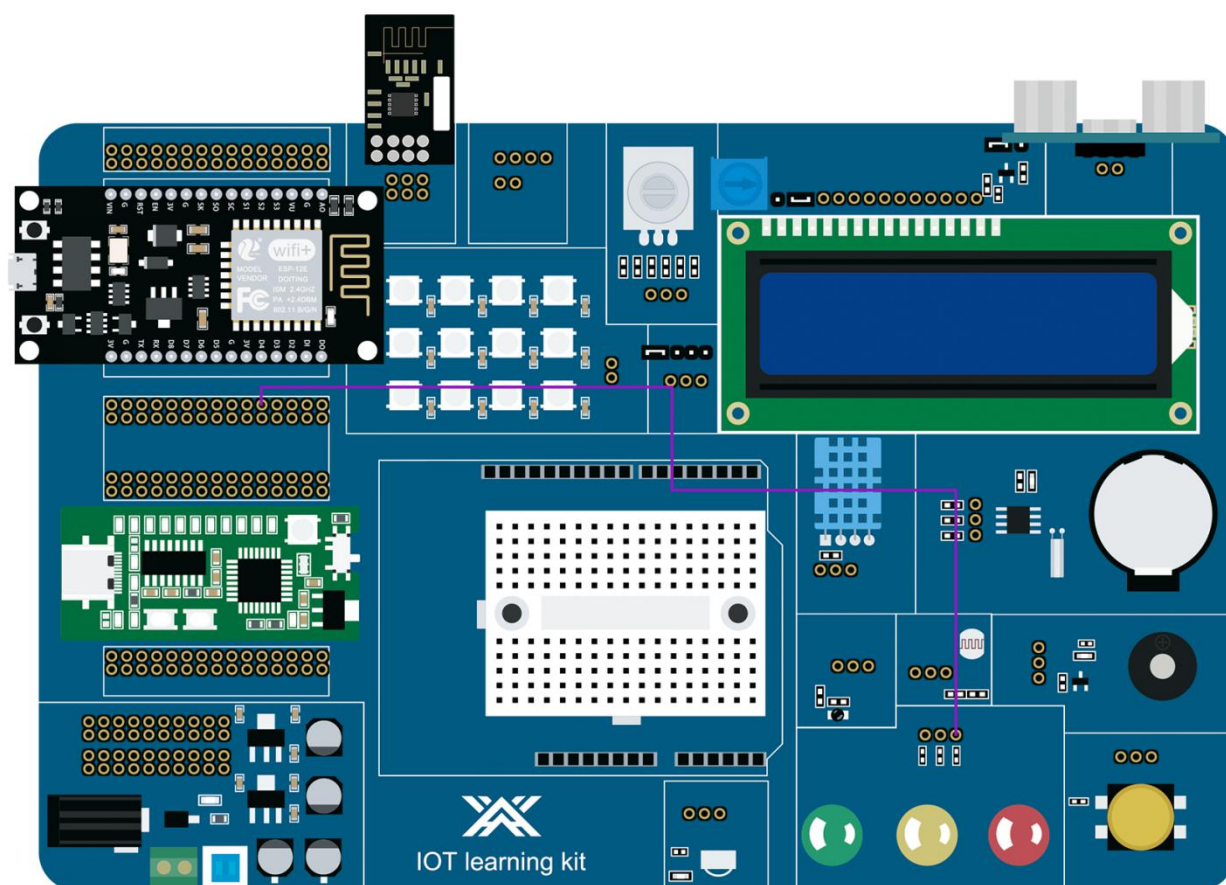
```
for(int dutyCycle = 0; dutyCycle < 1023; dutyCycle++){  
  
    // 用 PWM 改变 LED 亮度  
  
    analogWrite(ledPin, dutyCycle);  
  
    delay(1);};
```

要设置 LED 亮度, 您需要使用 `analogWrite()` 接受 GPIO 作为参数的函数, 您想在其中获取 PWM 信号和 0 到 1023 之间的值以设置占空比。

### 24.3 上传代码:

在您的 Arduino IDE 中, 转到 Tools > Board 并选择您的 ESP8266 型号。

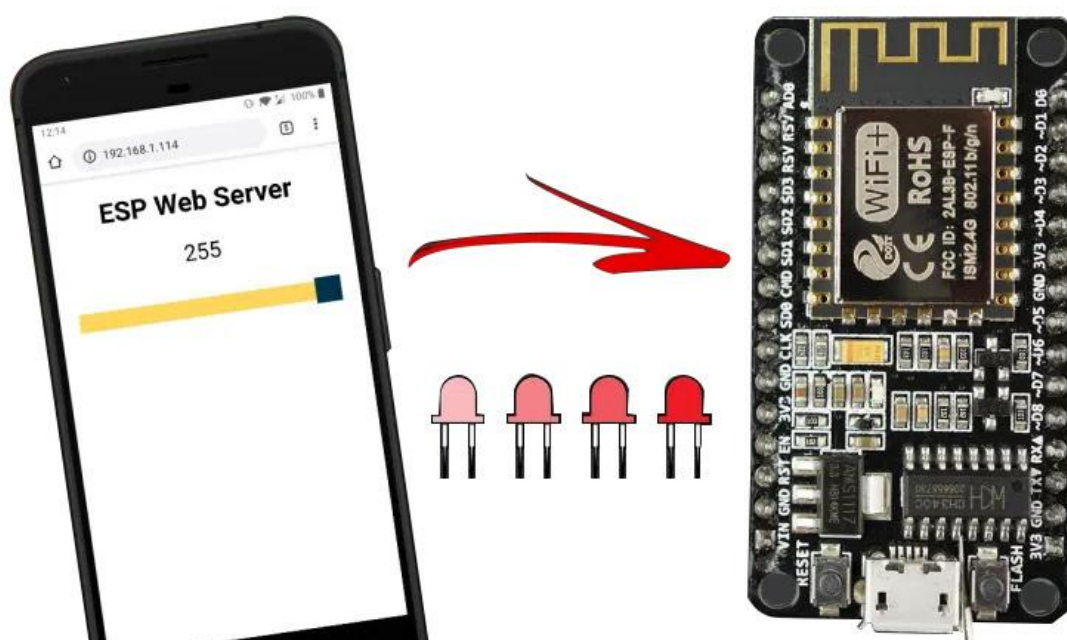
## 24.4 接线图:



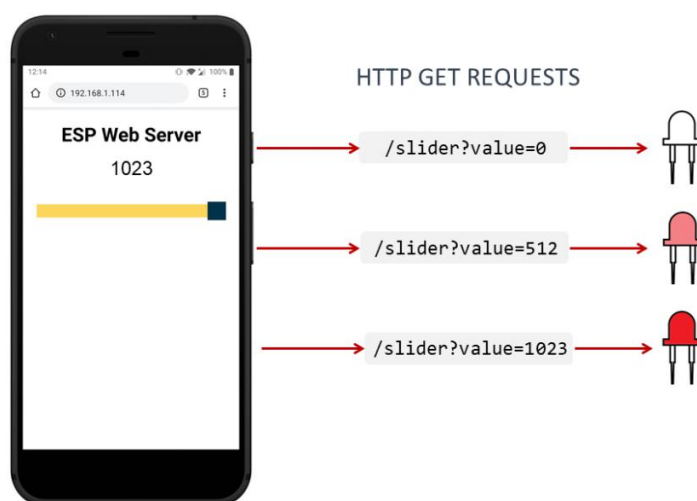


# 第 25 课 ESP8266 Node MCU Web 服务器 控制 LED 亮度 (PWM)

本教程展示了如何构建带有滑块的 ESP8266 NodeMCU Web 服务器来控制 LED 亮度。您将学习如何将滑块添加到您的网络服务器项目，获取其值并将其保存在 ESP8266 可以使用的变量中。我们将使用该值来控制 PWM 信号的占空比并改变 LED 的亮度。例如，您也可以控制伺服电机而不只是 LED。



此外，您还可以修改本教程中的代码，为您的项目添加一个滑块，以设置阈值或您需要在代码中使用的任何其他值。



ESP8266 托管一个 Web 服务器，该服务器显示带有滑块的网页；  
当您移动滑块时，您会使用新的滑块值向 ESP8266 发出 HTTP 请求；  
HTTP 请求采用以下格式：GET/slider?value=SLIDERVALUE，其中 滑块值 是一个介于 0 和 1023 之间的数字。您可以修改滑块以包括任何其他范围；  
ESP8266 从 HTTP 请求中获取滑块的当前值；  
ESP8266 根据滑块值调整 PWM 占空比；  
这对于控制 LED 的亮度（如我们将在本例中所做的那样）、伺服电机、设置阈值或其他应用程序非常有用。

## Arduino IDE

我们将使用 Arduino IDE 对 ESP8266 NodeMCU 板进行编程，因此在继续本教程之前，请确保您的 Arduino IDE 中安装了 ESP8266 板。

### 25.1 工作代码：

```
// 添加所需的库

#include <ESP8266WiFi.h>

#include <ESPAsyncTCP.h>

#include <ESPAsyncWebServer.h>

// 用您的网络凭据替换（输入您的 WiFi 名称和 WiFi 密码）

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";

// 在端口 80 上创建 AsyncWebServer 对象
```

```

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";

// 将网页中的占位符替换为按钮部分

String processor(const String& var){

    //Serial.println(var);

    if (var == "SLIDERVALUE"){

        return sliderValue;

    }

    return String();}

void setup(){

    // 用于调试的串口

    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..");

    }

    // 输出“ESP 本地 IP 地址”
```

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

// Route for root / 网页

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

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

});

// 发送一个 GET 请求到<ESP_IP>/slider?值= < inputMessage >
server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest *request) {

    String inputMessage;

    // 在<ESP_IP>/slider 上获取 input1 值?值= < 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 代码工作原理:

首先, 导入所需的库。这 ESP8266WiFi, ESPAsyncWebServe 和 ESPAsyncTCP 所需构建 Web 服务器。

```
#include <ESP8266WiFi.h>  
  
#include <ESPAsyncTCP.h>  
  
#include <ESPAsyncWebServer.h>
```

### 设置网络凭据:

在以下变量中插入您的网络凭据, 以便 ESP8266 可以连接到您的本地网络。

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

### 变量定义:

我们将控制 ESP8266 内置 LED 的亮度。内置 LED 对应于通用输入输出口 2。将我们要控制的 GPIO 保存在输出变量。

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

这 滑块值变量将保存滑块值。开始时, 它被设置为零。

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

### 输入参数:

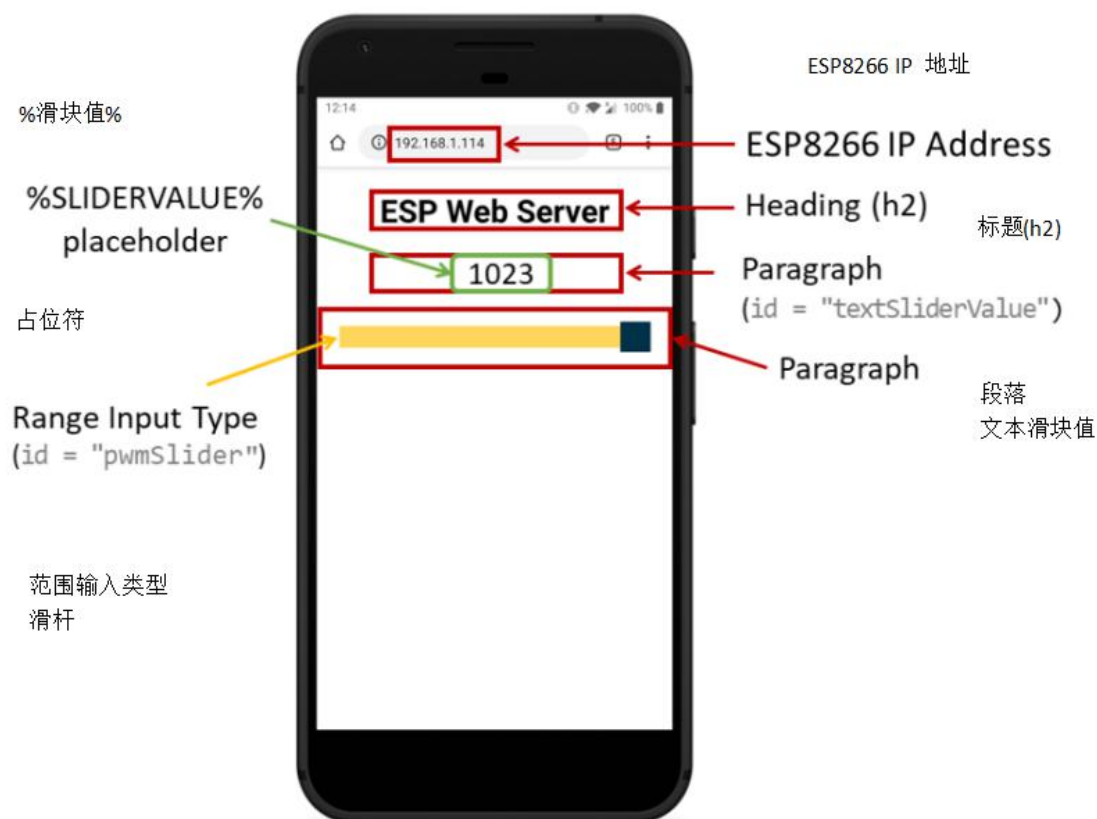
这参数输入当滑块移动时, 变量将用于“搜索”ESP8266 收到的请求中的滑块值。(记住: ESP8266 会收到这样的请求 GET/slider?value=SLIDERVALUE)

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

它会搜索 value 在 URL 上并获取分配给它的值。

## 25.3 构建网页

现在让我们进入 Web 服务器页面。



这个项目的网页非常简单。它包含一个标题、一个段落和一个类型范围的输入。

让我们看看网页是如何创建的。

包含样式的所有 HTML 文本都存储在 `index_html` 变量。现在我们将浏览 HTML 文本并查看每个部分的作用。

下面的`<meta>`标签使您的网页在任何浏览器中都能响应。



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

之间的<title> </TITLE>标签进入我们的 Web 服务器的名称。标题是显示在 Web 浏览器选项卡上的文本。

样式

在<style></style>标签之间，我们添加了一些 CSS 来设置网页的样式。

```
<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>
```

基本上，我们将 HTML 页面设置为在没有边距的块中显示带有 Arial 字体的文本，并在中心对齐。

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

面的行设置标题(h2)和段落(p)的字体大小。

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

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

设置 HTML 主体属性。

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

下面的行自定义滑块：

```
.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; }
```

HTML 正文

里面的<body> </ body>标签是我们添加的网页内容。

该<h2> </ h2>标签标题添加到网页。在这种情况下，“ESP Web Server”文本，但您可以添加任何其他文本。

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

第一段将包含当前滑块值。那个特定的 HTML 标签有 id textSliderValue 分配给它，以便我们以后可以引用它。

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

这 %滑块值% 是滑块值的占位符。当 ESP8266 将其发送到浏览器时，它将被实际值替换。这对于在您第一次访问浏览器时显示当前值很有用。

## 创建滑块

要在 HTML 中创建滑块，请使用 <input> 标签。所述 <输入> 标签指定一个字段，其中用户可以输入数据。

有多种输入类型。要定义滑块，请使用“类型”属性和“范围”值。在滑块中，您还需要使用“ min ”和“ max ”属性定义最小和最大范围（在这种情况下，0 和 1023，分别）。

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

您还需要定义其他属性，例如：

在步骤属性指定有效号码之间的间隔。在我们的例子中，它被设置为 1;

在类的风格滑块（类=“滑块”）;

用于更新网页上显示的当前位置的 id;

调用函数的 onchange 属性 (updateSliderPWM(this)) 在滑块移动时向 ESP8266 发送 HTTP 请求。这 this 关键字指的是滑块的当前值。

```
<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>
```

下一行通过其 `id` 获取当前滑块值并将其保存在滑块值 JavaScript 变量。以前，我们已将滑块的 `id` 分配给 PWM 滑块。所以，我们得到它如下：

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

之后，我们设置滑块标签（其 `id` 为 文本滑块值）到保存在滑块值变量。

最后，发出一个 HTTP GET 请求。

```
var xhr = new XMLHttpRequest();
```

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

```
xhr.send();
```

例如，当滑块位于 **0** 时，您对以下 URL 发出 HTTP GET 请求：

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

当滑块值为 **200** 时，您将在关注 URL 上收到请求。

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

这样，当 ESP8266 收到 GET 请求时，它可以检索 URL 中的 `value` 参数并相应地控制 PWM 信号，我们将在下一节中介绍：

## 处理器：

现在，我们需要创建 处理器（） 函数，当您第一次在浏览器中访问它时，它将用当前滑块值替换我们 HTML 文本中的占位符。

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

当请求网页时，我们检查 HTML 是否有任何占位符。如果它找到 **%SLIDERVALUE%** 占位符，我们将返回保存在 **滑块值** 变量。

## setup()

在 setup(), 初始化串行监视器进行调试。

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

将 PWM 信号的占空比设置为保存在 滑块值 (当 ESP8266 启动时, 它被设置为 0)。

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

连接到您的本地网络并打印 ESP8266 IP 地址。

```
// 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());
```

### 处理请求:

最后, 添加下几行代码来处理 Web 服务器。

```
// 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"); });
```

当我们对根 URL 发出请求时，我们发送存储在 index\_html 变量中的 HTML 文本。我们还需要传递 processor() 函数，它将用正确的值替换所有占位符。

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

我们需要另一个处理程序，将保存当前滑块的值，并设置相应的 LED 亮度。

```
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"); });
```

基本上，我们在以下几行获得滑块值：

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

然后，使用以下命令更新 LED 亮度（PWM 占空比）`ledcWrite()` 接受您要控制的通道和值作为参数的函数。

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

最后，启动服务器。

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

因为这是一个异步 web 服务器，所以我们不需要在 `loop()` 中写任何东西。

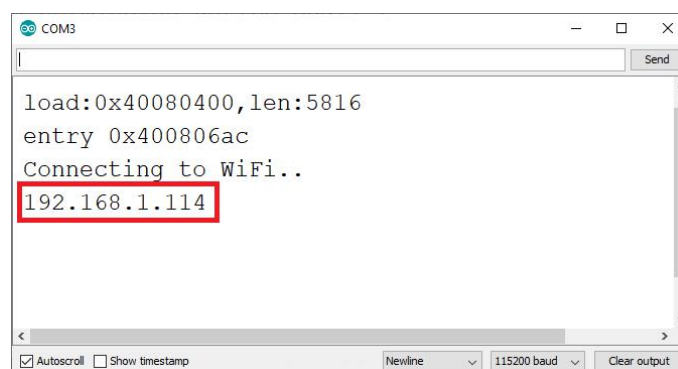
```
void loop(){ }
```

这就是代码的工作原理。

### 上传代码：

现在，将代码上传到您的 ESP8266。确保您选择了正确的电路板和 COM 端口。

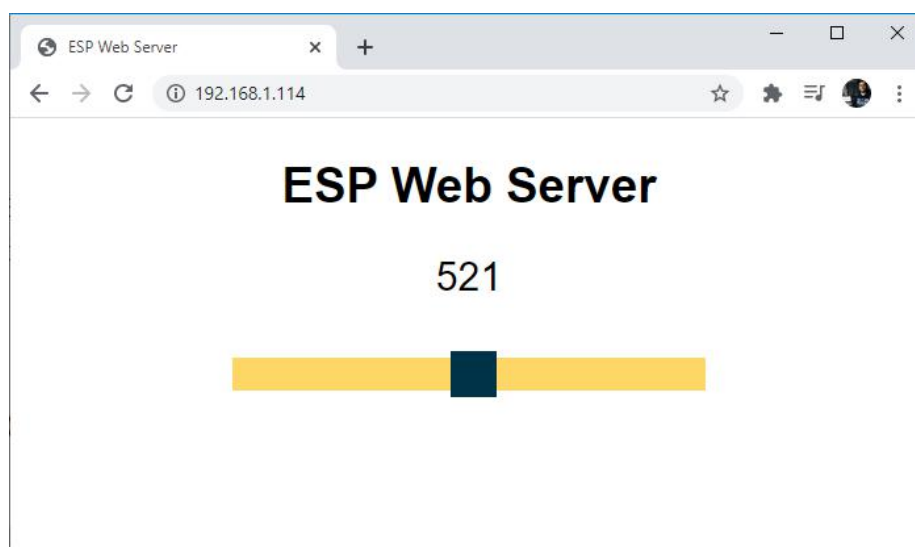
上传后，以 115200 的波特率打开串口监视器，按下 ESP8266 复位按钮。ESP8266 IP 地址应打印在串行监视器中。



## 25.4 网络服务器演示

打开浏览器并输入 ESP8266 IP 地址。您的 Web 服务器应显示滑块及其当前值。

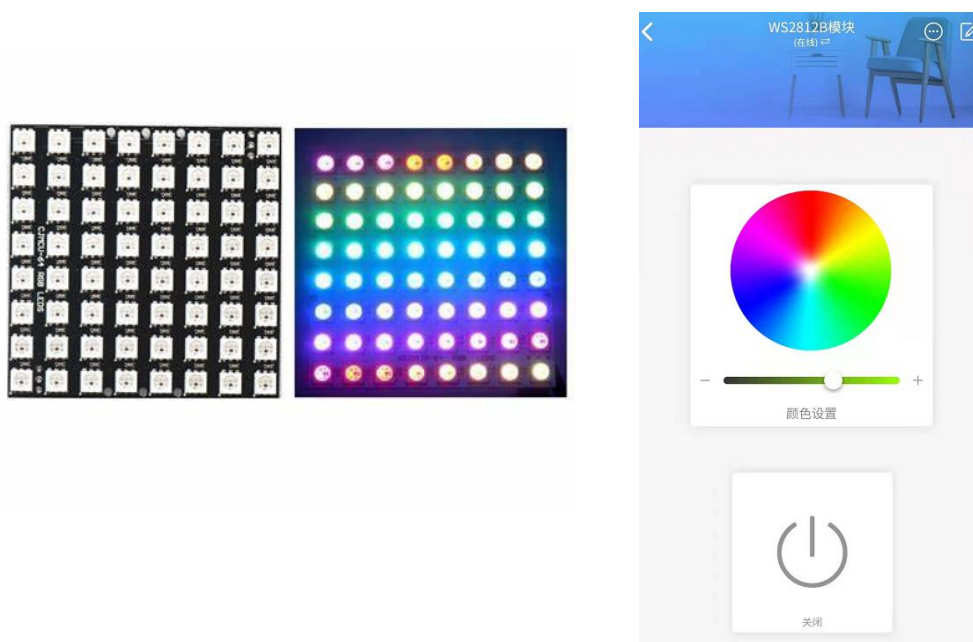




移动滑块，可以看到 ESP8266 内置 LED 的亮度增加和减少。

# 第 26 课 ESP8266 通过 blinker 控制 WS2812 灯

本教程展示 Nodemcu ESP8266 如何通过 Blinker 来控制 RGB 改变颜色。您将学习如何使用 Blinker APP 通过物联网从世界各地控制它；例如，您也可以控制伺服电机或直流电机而不只是 RGB。



## 26.1 Arduino 配置

安装 blinker Arduino

1、打开教程的库文件找到 blinker 库、Adafruit\_NeoPixel 库解压到 我的电脑 > 文档 > Arduino > libraries 文件夹中；

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、在 App 中添加设备，获取 Secret Key

1. 进入 App，点击右上角的 “+ ” 号，然后选择 **添加设备**；
2. 点击选择 **Arduino** > **WiFi 接入**；
3. 选择要接入的服务商；
4. 复制申请到的 **Secret Key**；

( 注意：如果你没有使用 blinker,你需要用手机号码注册相关账号即可操作。)



## 26.2 工作代码:

(注意: 修改源码中 `auth[]` 的值为在 App 中获取到的 `Secret Key` , 其他配置可根据自身条件设置。)

```
#define BLINKER_PRINT Serial

#define BLINKER_MQTT_LIGHT

#define BLINKER_WIFI

#include <Blinker.h>

#include <Adafruit_NeoPixel.h>

char auth[] = "*****";    /****输入您在 blinker 获取的密钥***/

#define PIN 15    //  DIN PIN  (GPIO15, D8)

#define NUMPIXELS 60    //  定义需要点亮的 RGB 数量

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB
+ NEO_KHZ800);

// 新建组件对象

BlinkerRGB RGB1("RGB");

int LED_R=0,LED_G=0,LED_B=0,LED_Bright=180; // 定义 RGB 和亮度

bool WIFI_Status = true;

void smartConfig()    //配置网函数

{

    WiFi.mode(WIFI_STA);

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

    WiFi.beginSmartConfig();//等待手机端发出的用户名与密码
```

```
while (1)
{
    Serial.print(".");
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
    if (WiFi.smartConfigDone())//退出等待
    {
        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 正在连接");
    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 连接失败，请用手机进行配网");
        }
    }
else
{
    smartConfig(); //微信智能配网
}

/* Serial.println("连接成功");
   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++) //把灯条变色
    {
        pixels.setPixelColor(i,R,G,B);
    }

    pixels.setBrightness(bright);//亮度

    pixels.show();    //送出显示
}

//APP RGB 颜色设置回调

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() {
    // 初始化串口
    Serial.begin(115200);

    pixels.begin();//WS2812 初始化
    pixels.show();
    pinMode(LED_BUILTIN, OUTPUT);
    #if defined(BLINKER_PRINT)
        BLINKER_DEBUG.stream(BLINKER_PRINT);
    #endif

    WIFI_Set();

    // 初始化 blinker
    Blinker.begin(auth, WiFi.SSID().c_str(), WiFi.psk().c_str());

    RGB1.attach(rgb1_callback);//注册调节颜色的回调函数

}

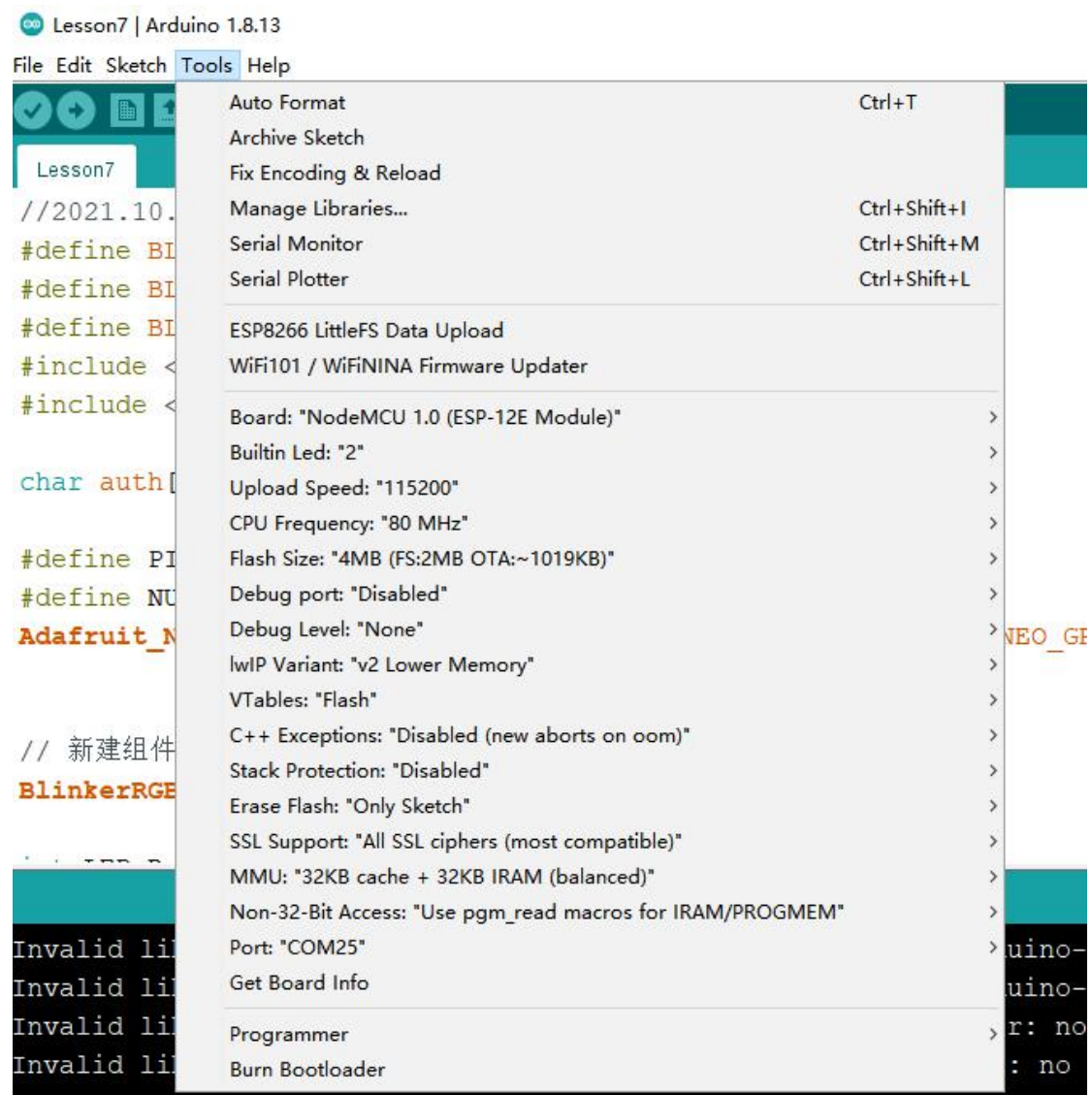
void loop() {
```

```
Blinker.run();

}
```

其中源码中只实现单色显示，更多颜色或酷炫的效果显示请自己修改源码实现。

上传代码：



## 26.3 App 配置连接

### 连接设备

打开点灯 App，点击侧边栏按钮 -> 开发者 -> 开发工具 -> EspTouch/SmartConfig，输入 WiFi 密码后单击开始配置；



## 输入 WIFI 密码

EspTouch快捷配置

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

## 配置成功

22:02

EspTouch快捷配置

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

WiFi 密码.....

配置成功

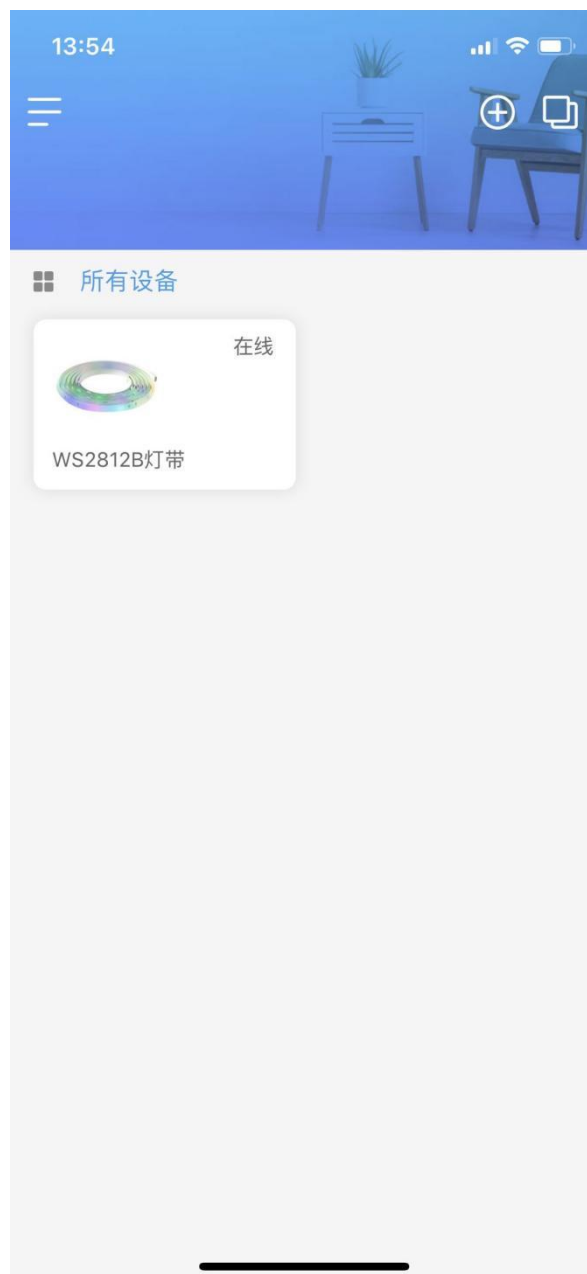
MAC: d8f15b11480f

IP: 192.168.3.107

返回

## 设备配置

1.返回 点灯 blinker App 主页，可看到设备已在线，点击设备进入配置页面；



2. 点击设备编辑按钮；



3. 点击编辑按钮后会进入组件编辑状态，点击底部组件列表的 颜色 组件，添加到界面中，然后点击界面中新增的 颜色 组件进入组件编辑界面，将 数据键名 改为 RGB，点击 保存；

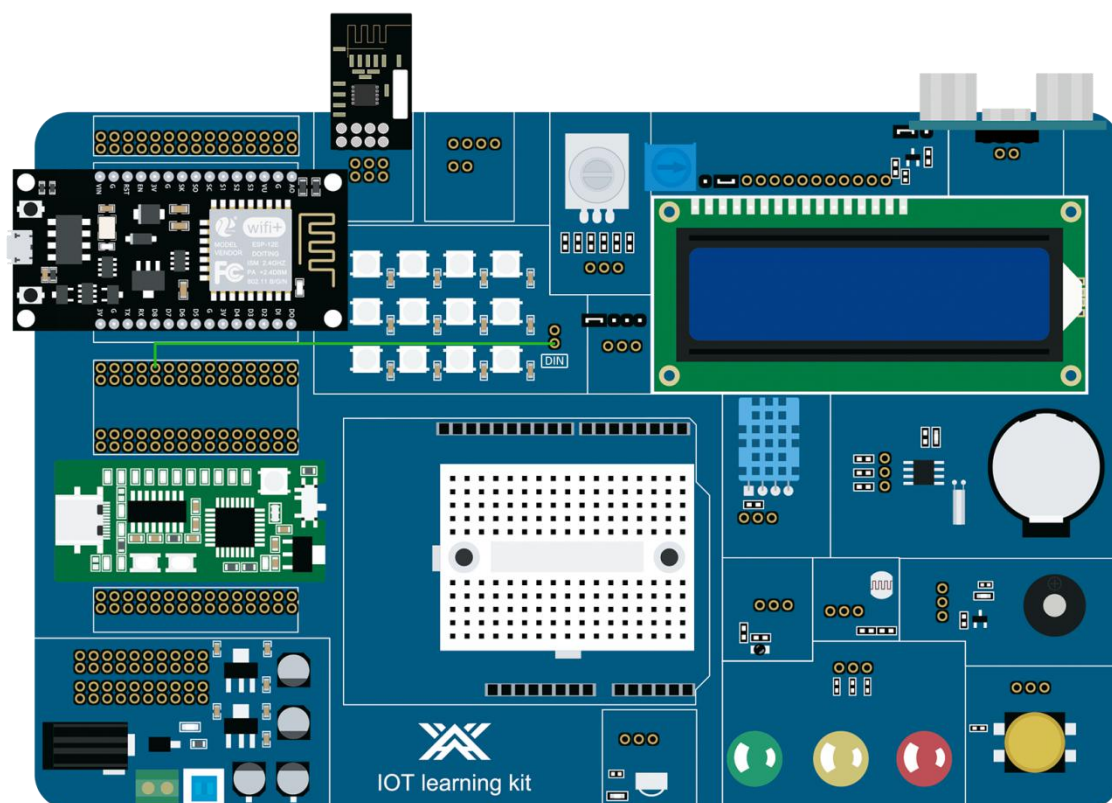


4. 编辑完成后, 点击右上角的 锁定 按钮完成编辑, 然后就可以控制灯带了。





## 26.4 接线图:

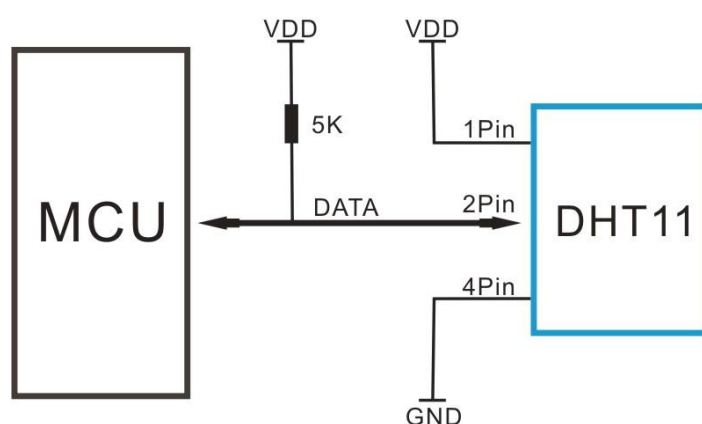
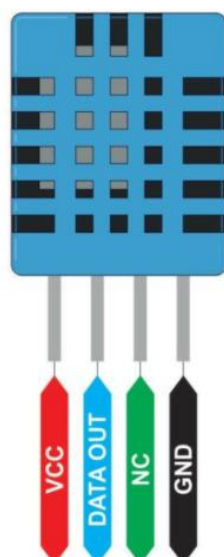


# 第 27 课 ESP8266 Nodemcu 结合 Blinker 显示温湿度

本节课程我们将使用 DHT11 温湿度传感器连接 ESP8266 Nodemcu 实现 blinker APP 显示温湿度。

## 27.1 DHT11 传感器：

DHT11 传感器提供湿度和温度数据。它有以下引脚接口。



典型应用电路

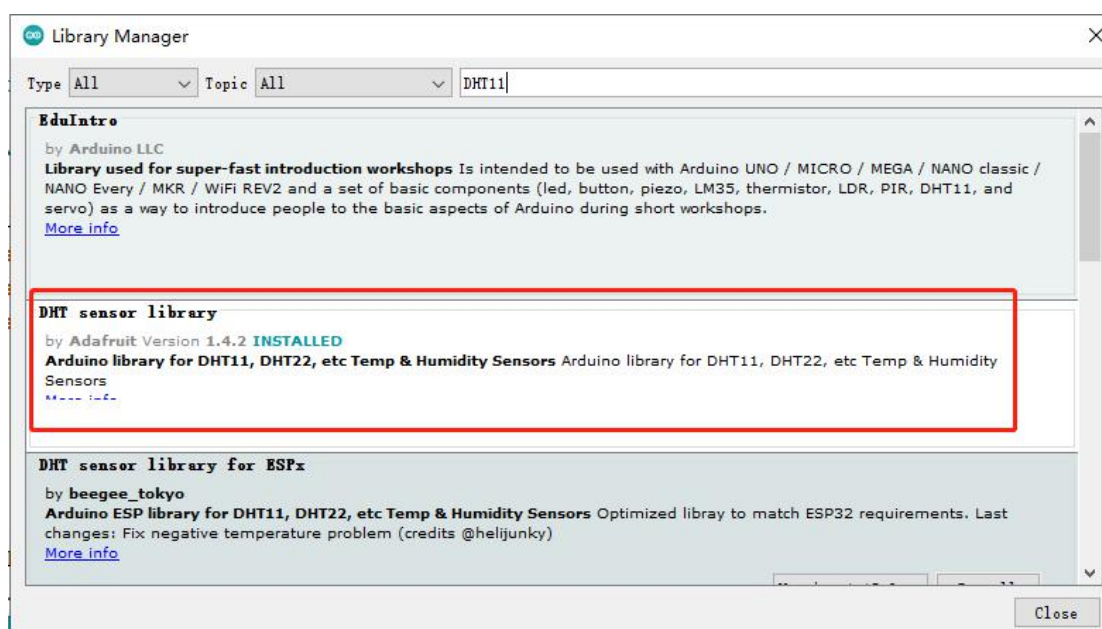
在上几节我们已经学会使用 Arduino IDE 开发 esp8266 所需要的设置，

## 27.2 安装库

如前所述，我们假设使用 Arduino IDE 对 ESP8266 进行编程。如果您尚未将其配置为支持 ESP8266 板，请回顾第 20 节课内容。

在 Arduino IDE 里添加 DHT11 的库文件。

该库可以通过 Arduino IDE 库管理器轻松安装，如图 3 所示。



## 27.3 工作代码讲解：

(如果你有小爱同学设备此代码也可以实现小爱同学查看家中温湿度)

```
#define BLINKER_WIFI

#define BLINKER_MQTT_SENSOR //小爱同学定义为传感器设备

#include <Blinker.h>

#include <DHT.h>

char auth[] = "*****"; //点灯 app 获取的设备

char ssid[] = "*****"; //WiFi 名称

char pswd[] = "*****"; //WiFi 密码

BlinkerNumber HUMI("humi"); //定义湿度数据键名
```

```
BlinkerNumber TEMP("temp"); //定义温度数据键名
#define DHTPIN 2 //定义 DHT11 模块连接管脚 GPIO2 (D4)
#define DHTTYPE DHT11 // 使用 DHT 11 温度湿度模块
// #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); //给 blinkerapp 回传湿度数据
  TEMP.print(temp_read); //给 blinkerapp 回传温度数据
}
void miotQuery(int32_t queryCode) //小爱同学语音命令反馈
{ BLINKER_LOG("MIOT Query codes: ", queryCode);
  int humi_read_int=humi_read; //去掉湿度浮点
  BlinkerMIOT.humi(humi_read_int); //小爱接收湿度
  BlinkerMIOT.temp(temp_read); //小爱接收温度
  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 读取显示温度
  BLINKER_LOG("Temperature: ", t, " *C"); //blinker APP 读取显示湿度

  humi_read = h;

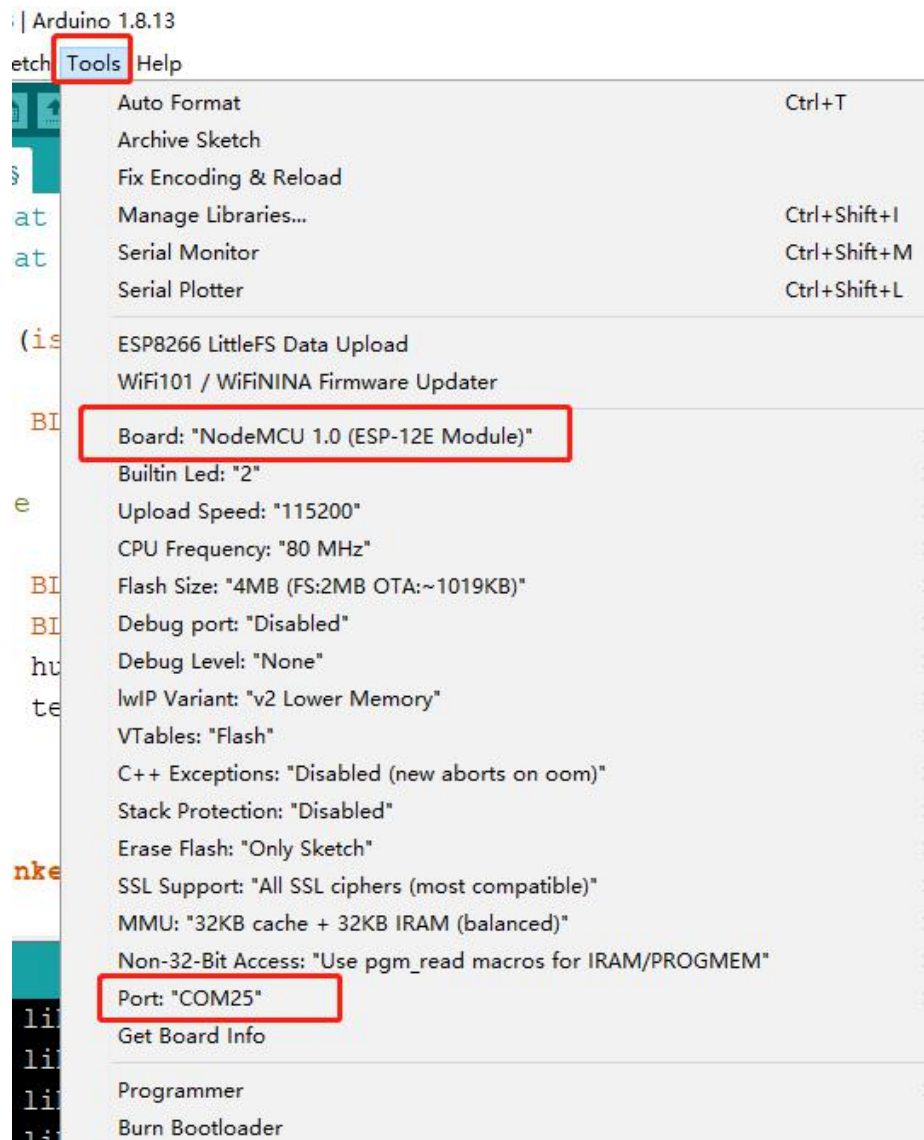
  temp_read = t;
}

Blinker.delay(2000);
}
```

在代码的注释内容下我们可以很容易的找到用 esp8266 接收数据的引脚是哪一个,但是在这里指的是 GPIO2, 并不是我们 esp8266d nodemcu 的 D2 而是 D4.



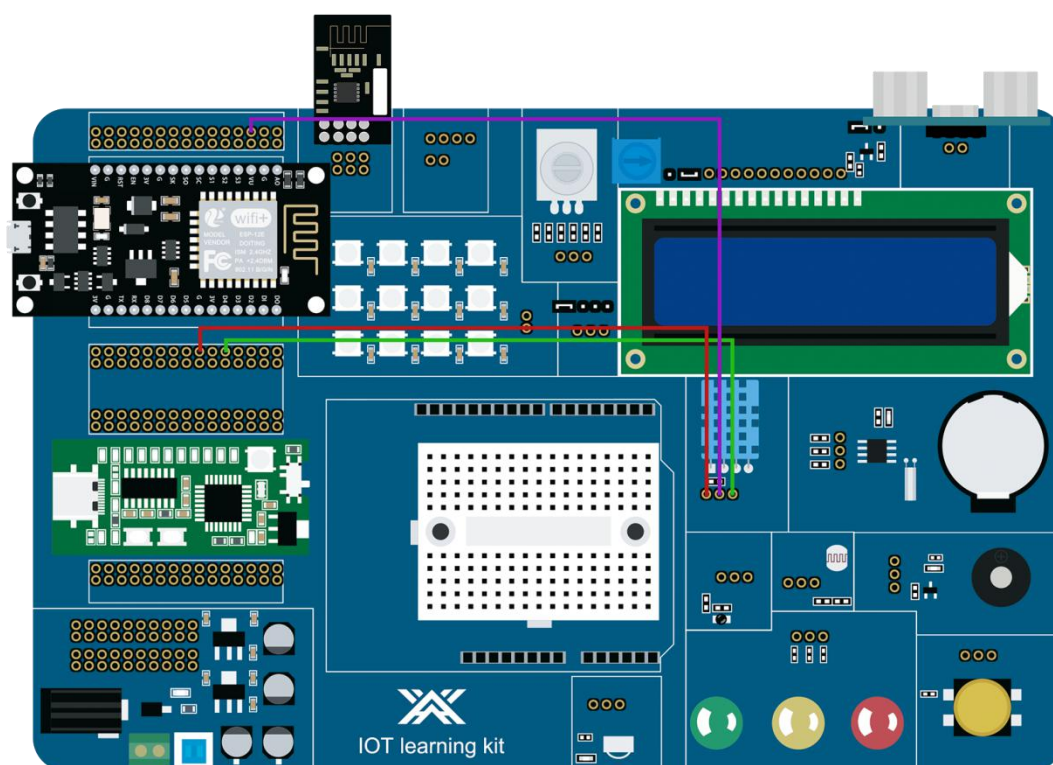
打开 Arduino IDE 写入代码, 编译上传, 烧录程序。



## 27.4 接线图:

我们的 DHT11 模块有三个引脚: V、G 以及 S.

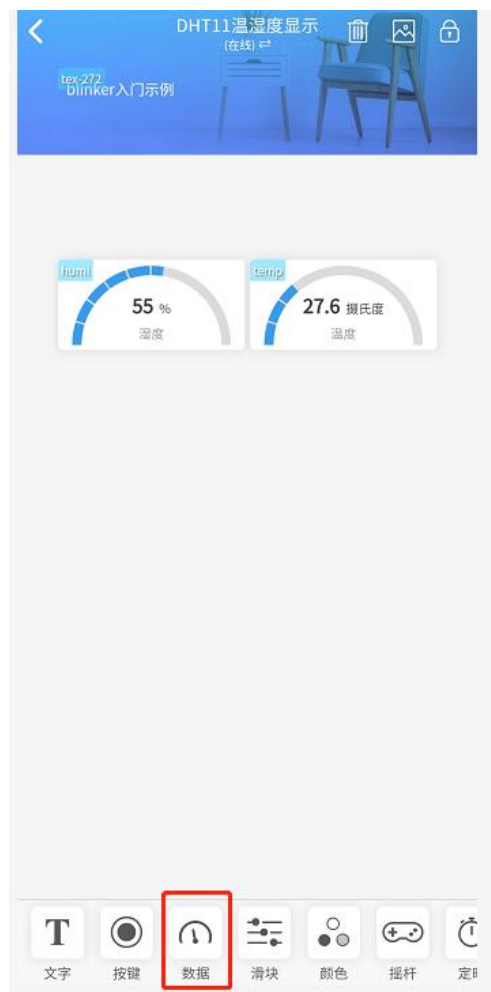




重启开发板，我们可以看到点灯 app 上设备在线，我们还可以设置点灯 app 内的组件设置，在点灯 app 内查看温度湿度：







例如我们的湿度数据，数据键名为代码前定义的 `humi`，显示文本为湿度，单位为%，最大值为 100；同理我们设置温度也是相同操作。

下午 3:50
组件编辑
删除
保存

样式设置

?
63%
湿度

63%
湿度

63%
湿度

参数设置

基础设置

数据键名
humi

显示文本1
湿度

数据单位
%

最大值
100

显示图标
修改图标

图标颜色

背景设置
☒ 正常
☐ 半透明
☐ 透明

最终显示正常的温湿度，如图：

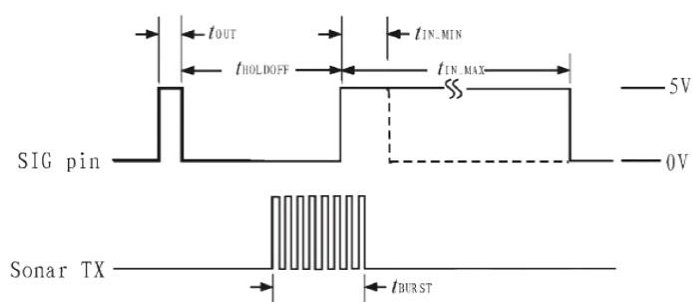
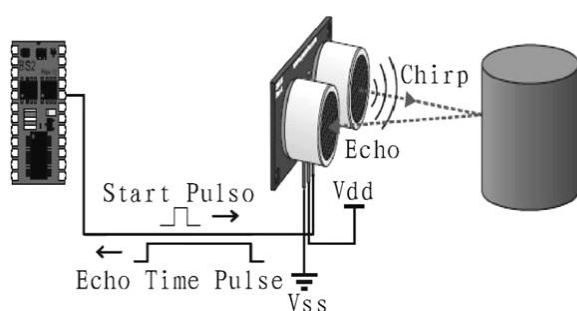


# 第 28 课 ESP8266 Nodem 结合 HC-SR04 超声波测距

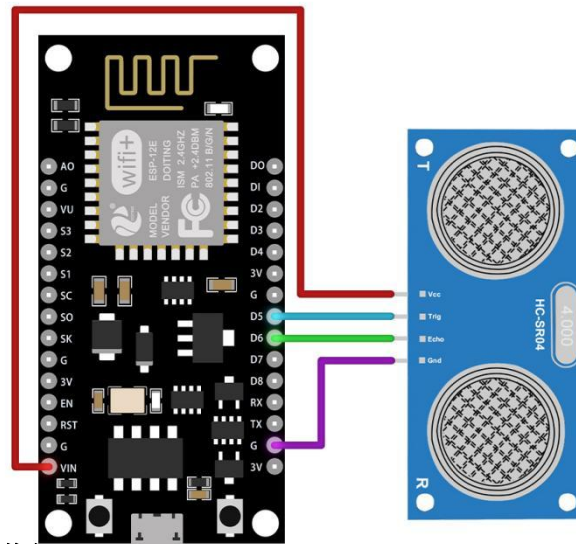
本节课程我们将使用 ESP8266 Nodem 结合 HC-SR04 超声波传感器模块制作测距工具。

## 28.1 超声波传感器

超声波测距原理是超声波发射探头发出的超声波脉冲，经媒质(空气)传到物体表面，反射后通过媒质(空气)传到接收探头，测出超声脉冲从发射到接收所需的时间，根据媒质中的声速，求得从探头到物体表面之间的距离。设探头到物体表面的距离为  $L$ ，超声在空气中的传播速为  $v$ ，从发射到接收所需的传播时间为  $t$ ，则有： $L=vt/2$ 。由此可见，被测距离  $L$  与传播时间之间具有确定的函数关系，只要能测出时间  $t$ ，即可求出距离  $L$ ，通过软件实现直接在显示器上显示  $L$  的值。



## 28.2 接线图:



## 28.3 工作代码讲解:

使用 HC-SR04 超声波传感器和 ESP8266 获得与物体的距离

首先，定义触发器和回声引脚。

```
const int trigPin = 12;

const int echoPin = 14;
```

在这个例子中，我们使用通用输入输出口 12 和通用输入输出口 14。你也可以任意修改其他 GPIO 引脚。

这声音速度变量保存了 20°C 空气中的声速。我们使用以 cm/uS 为单位的值。

```
#define SOUND_SPEED 0.034
```

这 CM\_TO\_INCH 变量允许我们将以厘米为单位的距离转换为英寸。

```
#define CM_TO_INCH 0.393701
```

然后，初始化以下变量。

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

这 `duration` 变量保存超声波的传播时间（从脉冲波的发送和接收开始经过的时间）。这 `distanceCm` 和 `distanceInch`，顾名思义，以厘米和英寸为单位保存到对象的距离。

## setup()

在里面 `setup()`，以 115200 的波特率初始化串行通信，以便我们可以在串行监视器上打印测量值。

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

将触发引脚定义为 **输出**——触发引脚发出超声波。并将回声引脚定义为 **输入**——回波引脚接收反射波，并向 **ESP8266** 发送与行程时间成正比的信号。

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

## loop()

在 `loop()` 中，以下行在触发器引脚上产生 10uS HIGH 脉冲——这意味着引脚将发射超声波。注意，在发送脉冲之前，我们给一个短的低脉冲，以确保您将得到一个干净的高脉冲。

```
// 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);
```

我们使用 `pulseIn()` 函数来获取声波的传播时间:

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

函数的作用是:读取引脚上的 HIGH 或 LOW 脉冲。它接受引脚和脉冲状态(HIGH 或 LOW)作为参数。它以微秒为单位返回脉冲长度。脉冲长度等于到达物体所花费的时间加上返回的时间。

然后,我们简单地计算到一个物体的距离考虑到声速。

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

转换距离为英寸:

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

最后,在串行监视器上打印结果。

```
Serial.print("Distance (cm): ");
```

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

```
Serial.print("Distance (inch): ");
```

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

把代码上传到你的板上。不要忘记选择你正在使用的板子在工具>板子。另外,不要忘记在 Tools > port 中选择正确的 COM 端口。

上传完成后,以波特率 115200 打开 Serial Monitor。按下车载 RST 按钮重新启动板,它将开始打印到串行监视器上最近的对象距离。如下图所示。

