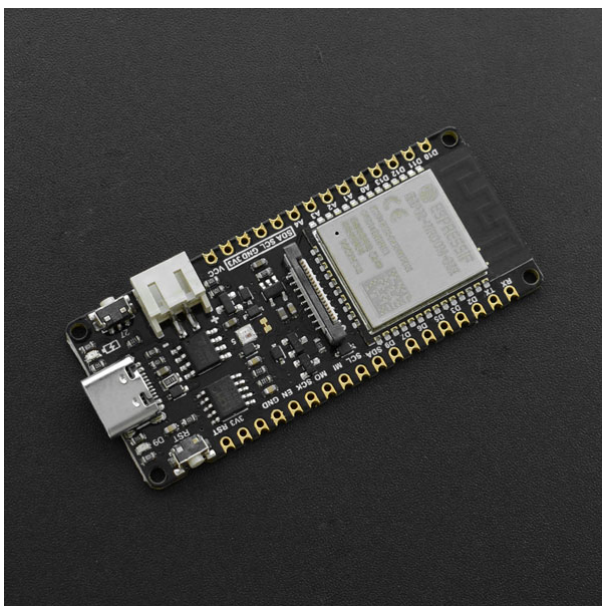


SKU:DFR0654 (<https://www.dfrobot.com/product-2195.html>)



(<https://www.dfrobot.com/product-2195.html>)

1. Introduction

FireBeetle ESP32-E, specially designed for IoT, is an ESP-WROOM-32E-based main controller board with dual-core chips. It supports WiFi and Bluetooth dual-mode communication, and features small size, ultra-low power consumption, on-board charging circuit and easy-to-use interface, which can be conveniently used for smart home IoT, industrial IOT applications, wearable devices and so on. You can easily create your own IoT smart home system when connecting it with an IoT platform like IFTTT. FireBeetle ESP32-E supports Arduino programming, and will support Scratch graphical programming and MicroPython programming very soon. We provide you with detailed online tutorials and application cases, and there are thousands of sensors with welding-free Gravity interface and actuators to help you get started easily. Besides,

the stamp hole design makes it able to be easily embedded in your PCB, greatly saving your costs and time to build and test prototype.

2. What is FireBeetle Series?

FireBeetle was originally designed to be a high-performance and more Mini Arduino open-source development board series. Now it is not only fully compatible with Arduino development environment, but also comes with abundant hardware and software resources. FireBeetle will support the various development environment like MakeCode, Mind+, Pingpong and MicroPython (to be improved soon), which allows you to program your hardware by graphical programming, C language, Python or JS.

This open source board of high-flexibility could bring you infinite possibilities! There are a large number of detailed tutorials and thousands of easy-to-use Gravity peripherals that provide you with the simplest way to program. No matter you are a student, an electronic enthusiast, an artist or a designer, this would be your best partner to open up the world of electronic without dealing with complicated circuits, brain-burning codings, and all complex communication protocols. Turn your worthy ideas into fantastic reality with this FireBeetle board!

3. Features

- Compatible with DFRobot FireBeetle V2 Series
- Small Size of 25.4×60 mm
- ESP32 Dual-core low power maincontroller, WiFi+BT4.0
- GDI Display Port, easy to connect
- Onboard Charging Circuit and PH2.0 li-ion Battery Interface

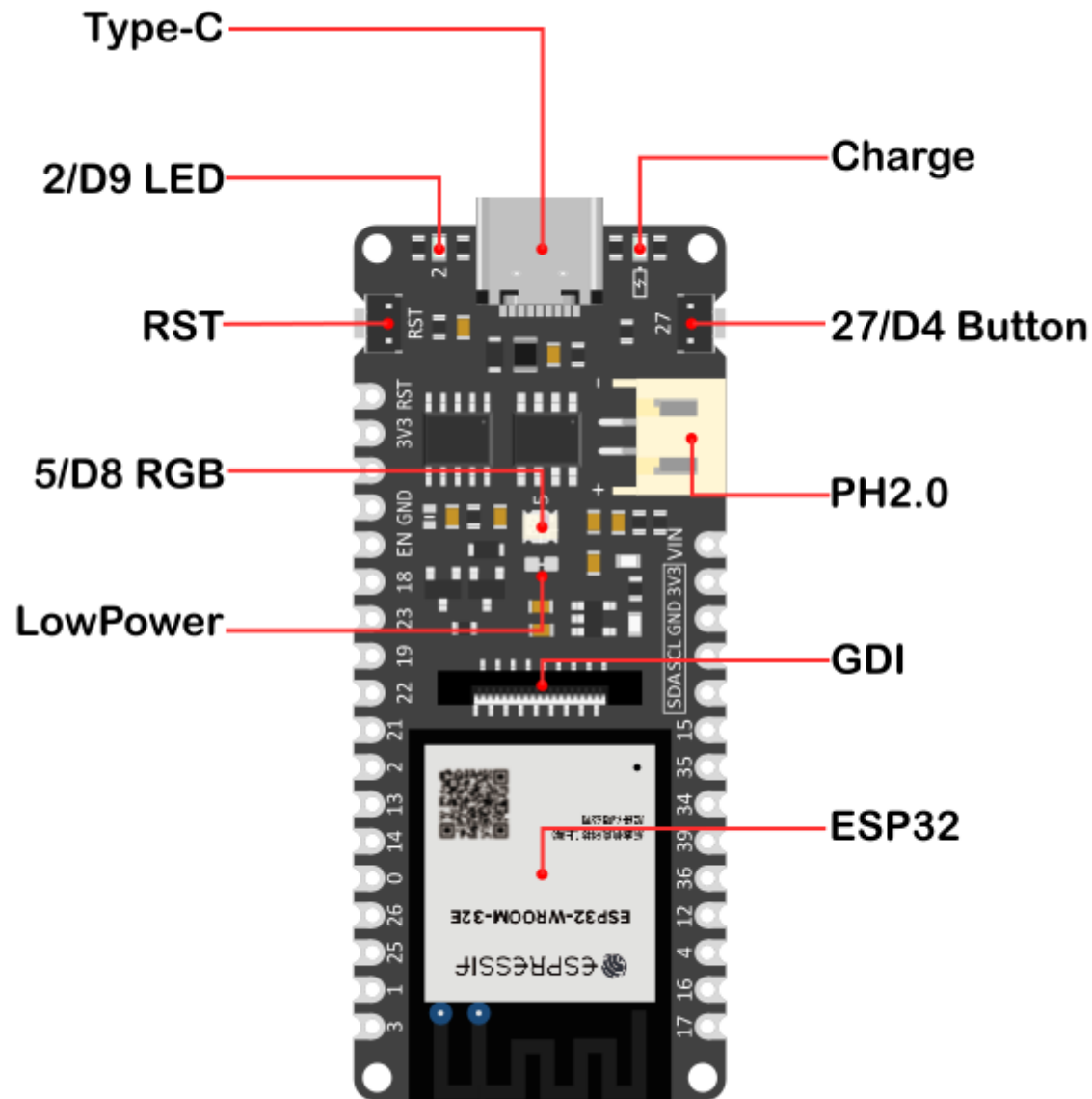
4. Specification

- Operating Voltage: 3.3V
- Input Voltage: 3.3V~5.5V
- Support Low-Power: 10uA
- Max Discharge Current: 600mA@3.3V (<mailto:600mA@3.3V>) LDO

- Max Charge Current: 500mA
- Support USB Charging
- Processor: Tensilica LX6 dual-core processor (One for high-speed connection; one for independent application development)

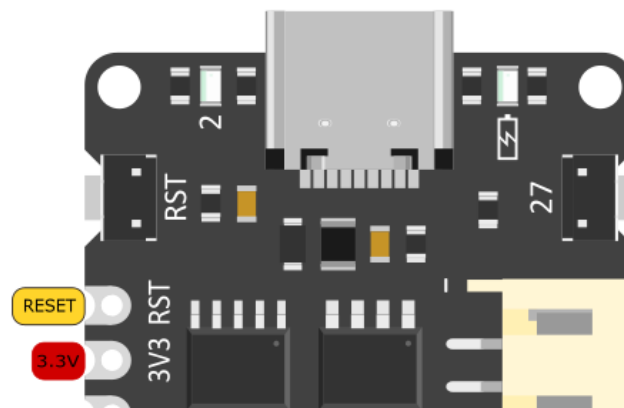
- Main Frequency: 240MHz
- SRAM: 520KB
- Flash: 32Mbit
- Wi-Fi Standard: FCC/CE/TELEC/KCC
- Wi-Fi Protocol: 802.11 b/g/n/d/e/i/k/r (802.11n, speed up to 150 Mbps), A-MPDU and A-MSDU Aggregation, support 0.4us guard interval)
- Frequency Range: 2.4~2.5 GHz
- Bluetooth Protocol: Bluetooth v4.2 BR/EDR and BLE standard compliant
- Bluetooth Audio: CVSD and SBC audio
- Operating Current: 80mA (Average)
- Support Arduino download with one-key
- Support MicroPython
- On-chip Clock: 40MHz crystal, 32.768KHz crystal
- Digital I/O x10(Arduino default)
- Analog Input x5(Arduino default)
- SPI x1(Arduino Default)
- IIC x1(Arduino Default)
- I2S x1(Arduino Default)
- RGB_LED: 5/D8
- Connector: FireBeetle V2 series compatible
- Operating Temperature: -40°C~+85°C
- Module Size: 25.4 × 60(mm)
- Mount Hole Size: M2 Mounting hole with diameter of 2.0mm

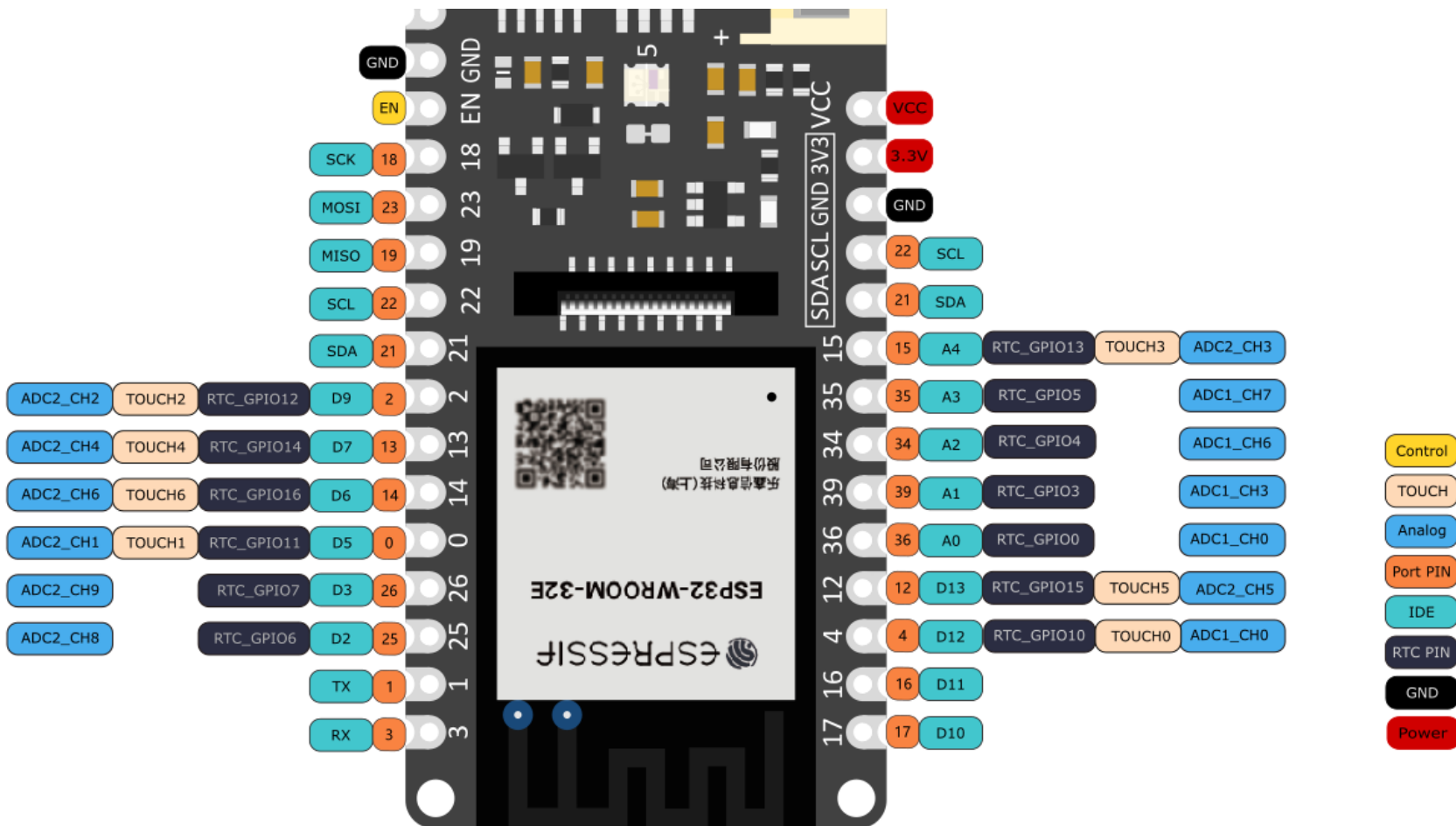
5. Board Overview



- Type-c: USB Interface: 4.75v-5.5v
- PH2.0 Li-ion Battery Connector: 3.5-4.2V
- 2/D9 LED: control LED via Pin 2/D9
- Charging Indicator: red LED for indicating charging status: 1. Off when fully charged or not charged; 2. On when charging; 3. Quick flash when powered by USB, and no battery connected.
- RST Reset Pin: click the reset button to reset program
- 5/D8 WS2812 Indicator: control WS2812 RGB LED via pin 5/D8
- Low Power Pad: This pad is specially designed for low power consumption. It is connected by default. You can cut off the thin wire in the middle with a knife to disconnect it. After disconnection, the static power consumption can be reduced by 500 μ A. The power consumption can be reduced to 13 μ A after controlling the maincontroller enter the sleep mode through the program. Note: when the pad is disconnected, you can only drive RGB LED light via the USB Power supply.
- GDI Display Interface: DFRobot dedicated display interface, details will be given later.
- ESP32 Module: the newest ESP32-e module launched by ESPRESSIF
- Button: control the button via pin 27/D4

6. Pinout





Overview

FireBeetle ESP32-E has up to 22 physical GPIOs, of which the pins 34-39 are only used as input pins, and others can be used as both input and output pins. All logic voltages are 3.3V.

- Control: FireBeetle Enable/reset pin
- Touch: pin with capacitive touch function
- Analog: analog pin

- Port Pin: the default physical pin number of the chip, which can be directly used to control the corresponding pin
- IDE: In Arduino IDE, the pin numbers have been remapped by FireBeetle, or you can directly use this symbol to control the corresponding pin
- RTC PIN: FireBeetle supports low power function, and in deepsleep mode, only RTC pins can be used.
- Power: FireBeetle leads out the power source voltage and the voltage stabilized 3.3V power supply through pins, which is convenient for users to use.
- GND: FireBeetle common ground pin

Power

- GND: common ground for all power and logic
- VCC: positive voltage of USB/Li-ion battery input(5V-output USB voltage when powered by USB; 3.7V-Output battery voltage when powered by Li-ion battery)
- 3V3: output of 3.3 voltage regulator, can provide 500mA peak current

Control

- RST: connected to the reset pin of ESP32, can reset program
- EN: enable pin of 3.3V voltage regulator. It has been pulled up, so grounding can disable the 3.3V regulator.

GPIO

- D2 to D13: these are general purpose pins, which are usually used as digital pins or multiplexed function
- A0 to A4: these are analog input pins, of which A0-A3 can only be used as input pins.
- SDA-IIC(line) data pin
- SCL-IIC(line) clock pin
- SCK/MOSI/MISO: hardware SPI pins, you can use them as normal GPIO pins (but it is recommended to leave them idle as they are best suited for high-speed SPI hardware)

UART

ESP32 has two UART ports, of which UART0 is for PC communication.

SerialPort Name	Arduino	TX	RX
UART0	Serial	Pin1	Pin3
UART2	Serial2	Pin17	Pin16

7. Getting Started (Use for first time)

7.1 Arduino IDE Configuration

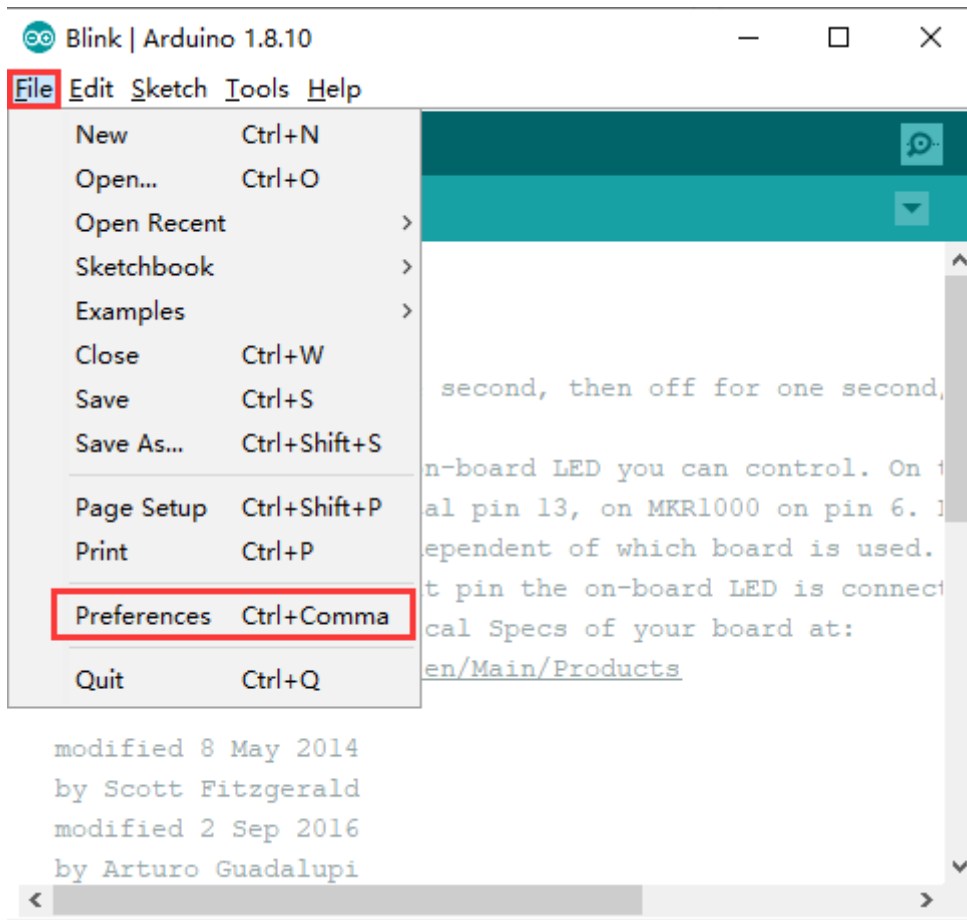
When using FireBeetle maincontroller for the first time, you need know the following steps:

- Add the json link in IDE
- Download the core of the maincontroller
- Select development board and serial port
- Get to know serial monitor

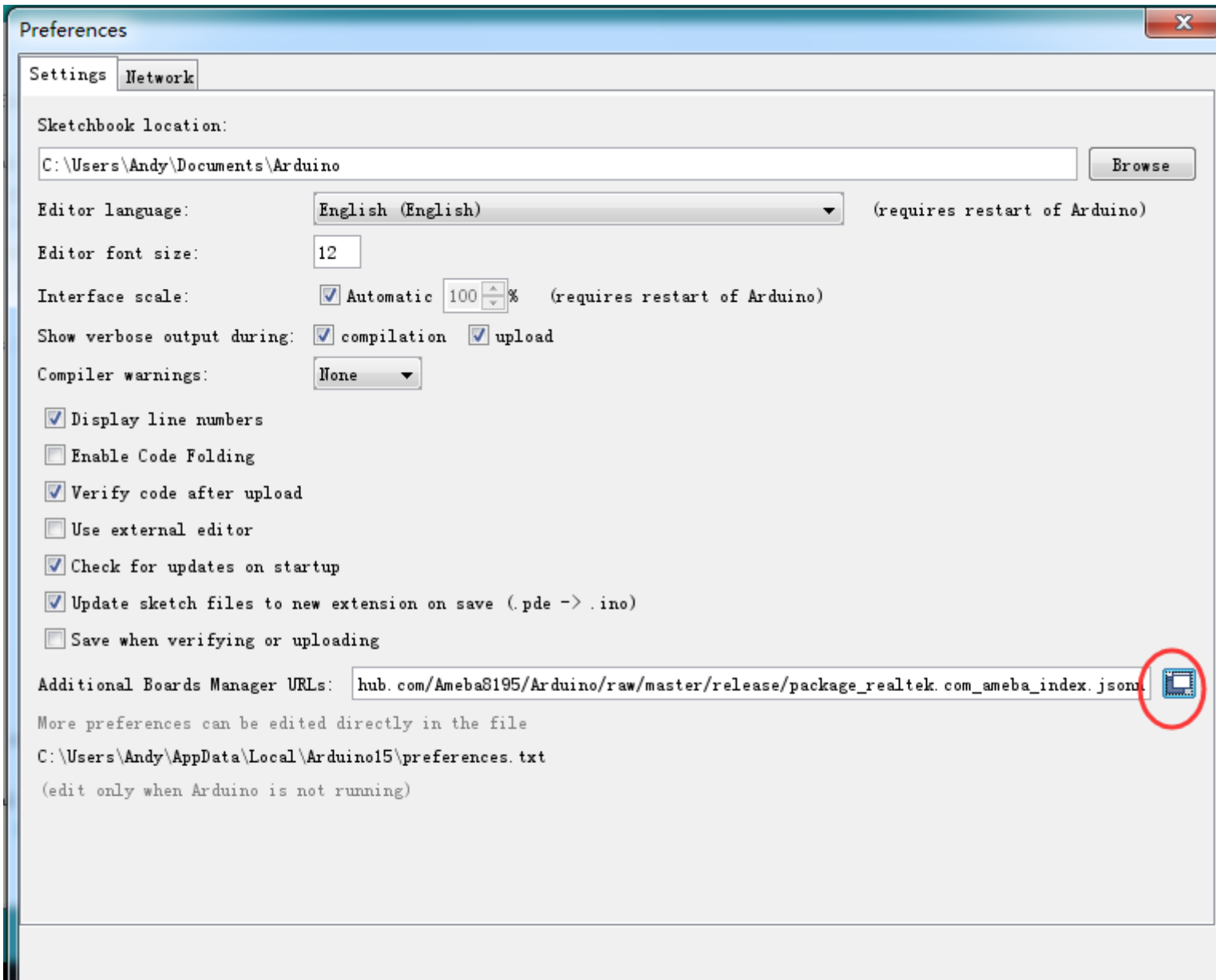
Arduino IDE Setup

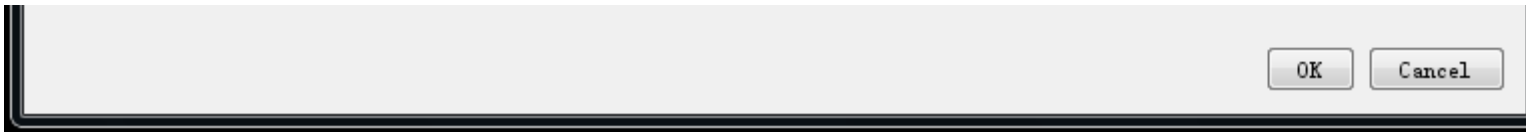
- Add URL to Arduino IDE

Open Arduino IDE, click File->Preferences, as shown below:

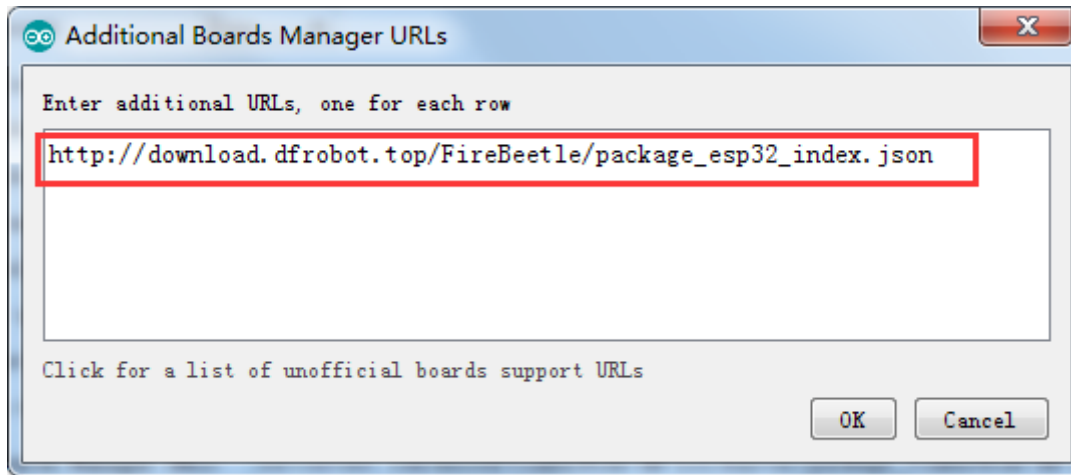


Click the icon marked with red below.



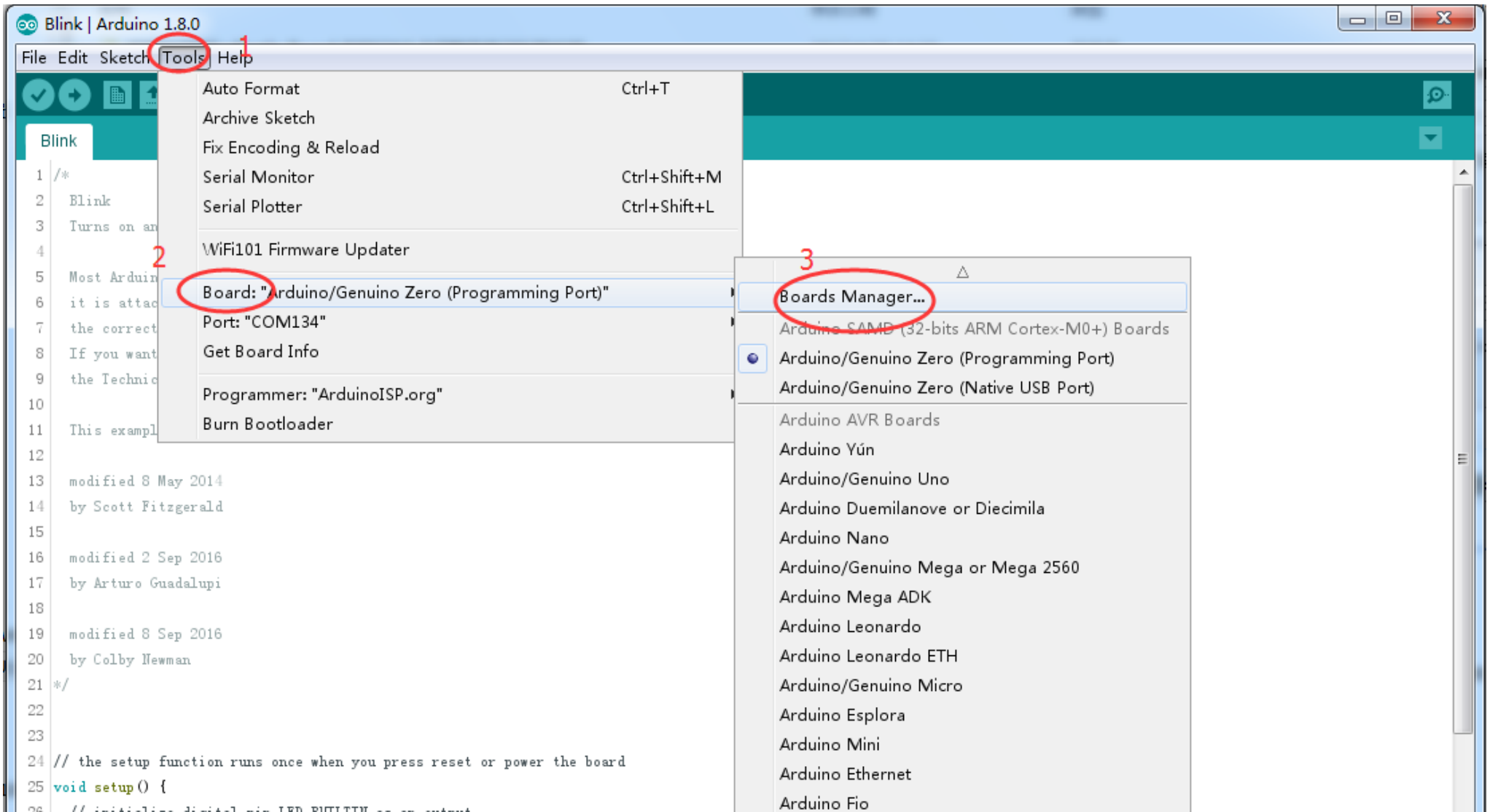


Copy the address to the newly popped up box: http://download.dfrobot.top/FireBeetle/package_DFRobot_index.json
(http://download.dfrobot.top/FireBeetle/package_DFRobot_index.json)

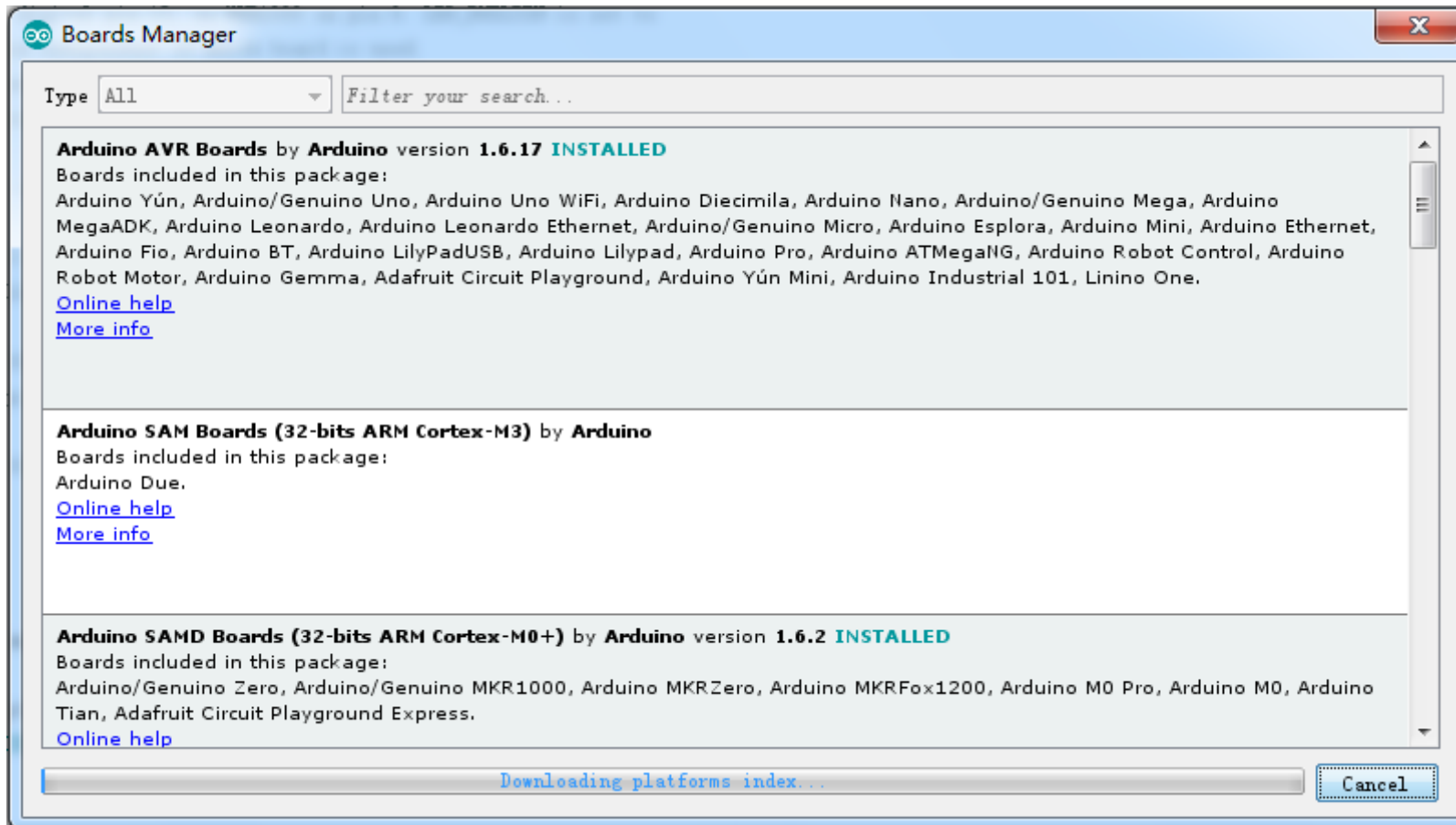


Click OK.

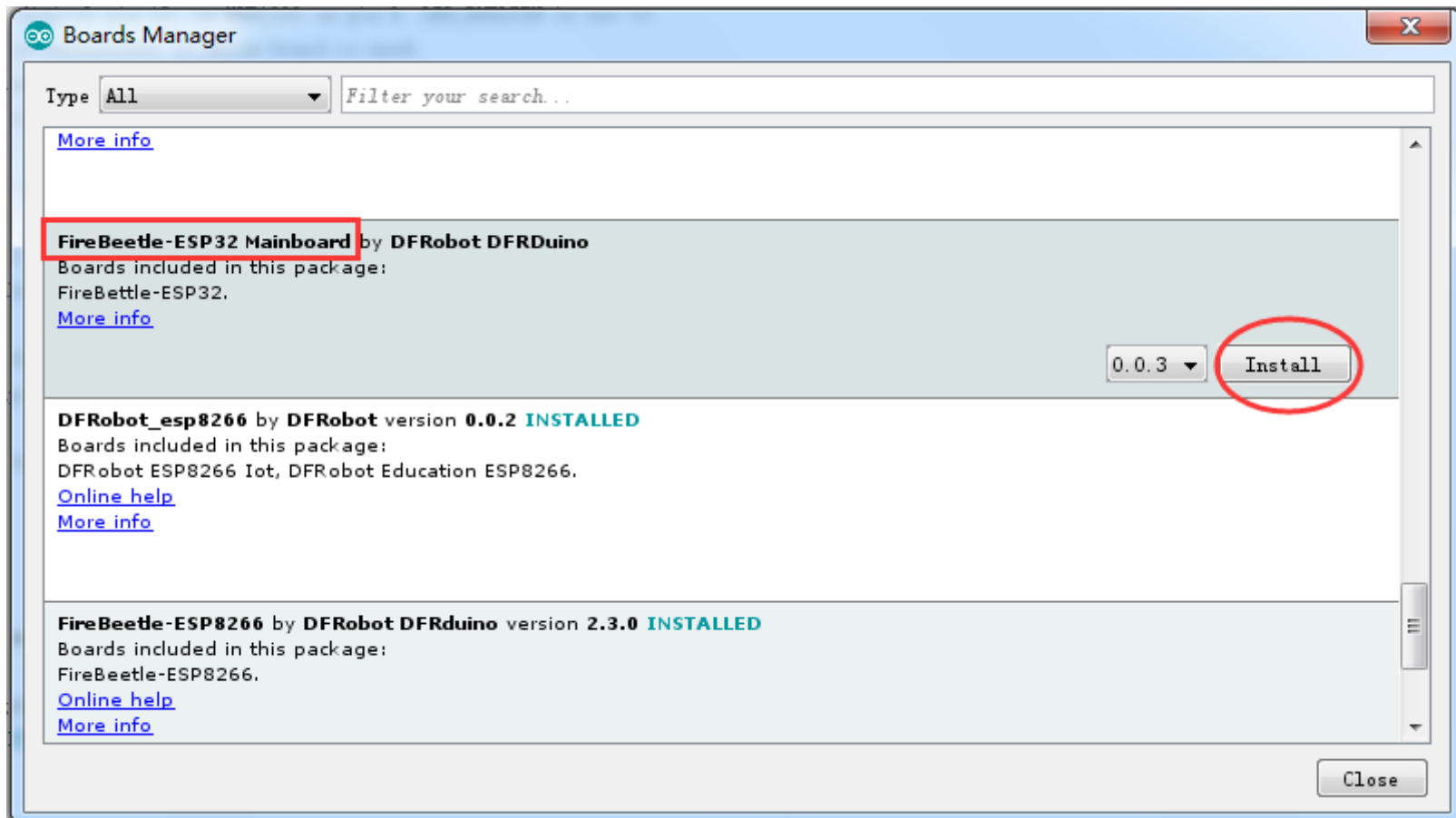
- Update board. Open Tools->Board->Boards Manager.



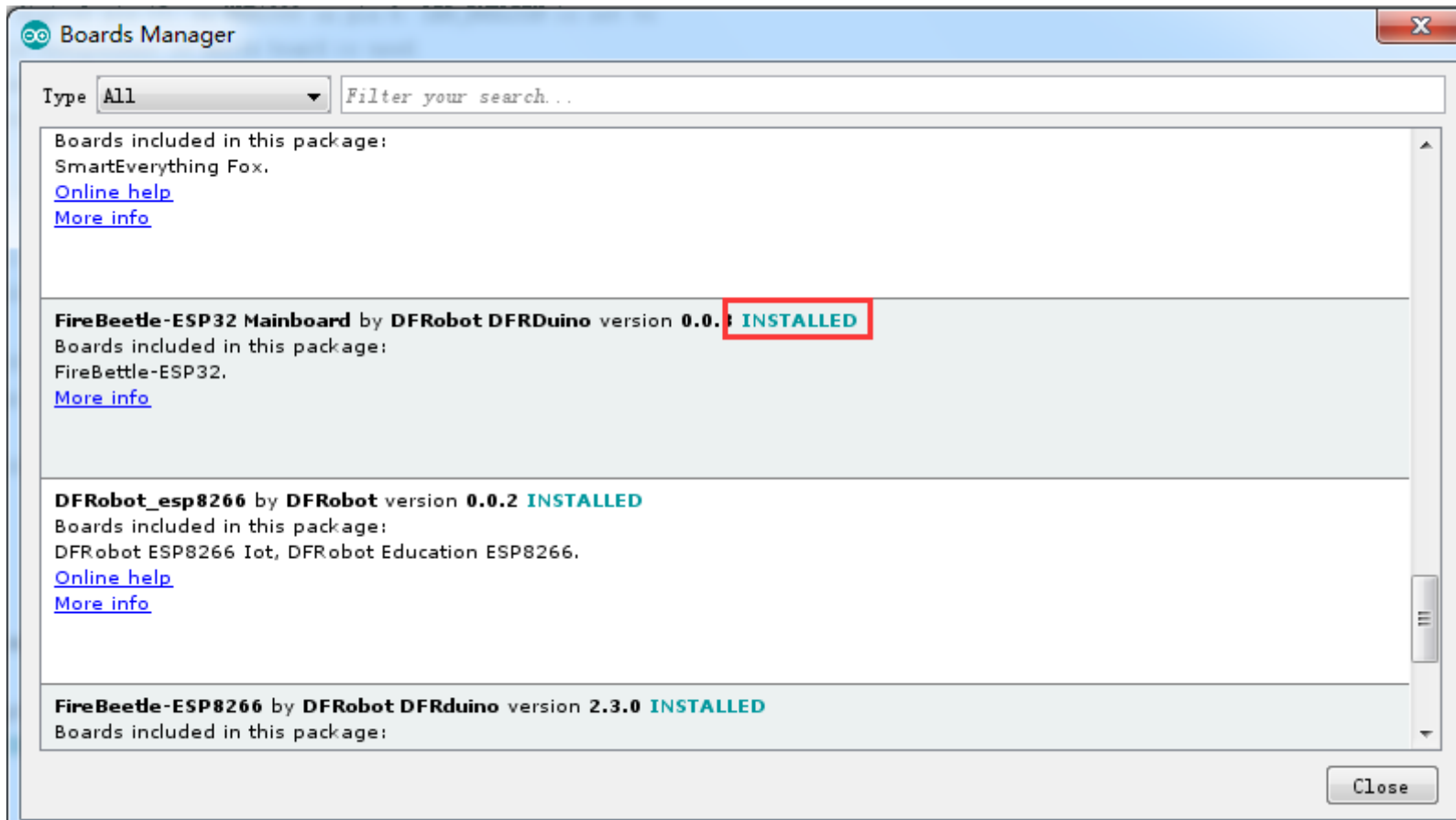
The board will be automatically updated.



Wait for while, then you will find the FireBeetle-ESP32(V0.0.8 Available now) in the list. Click "Install":



Done! You can find the installed FireBeetle-ESP32 board in the list now.



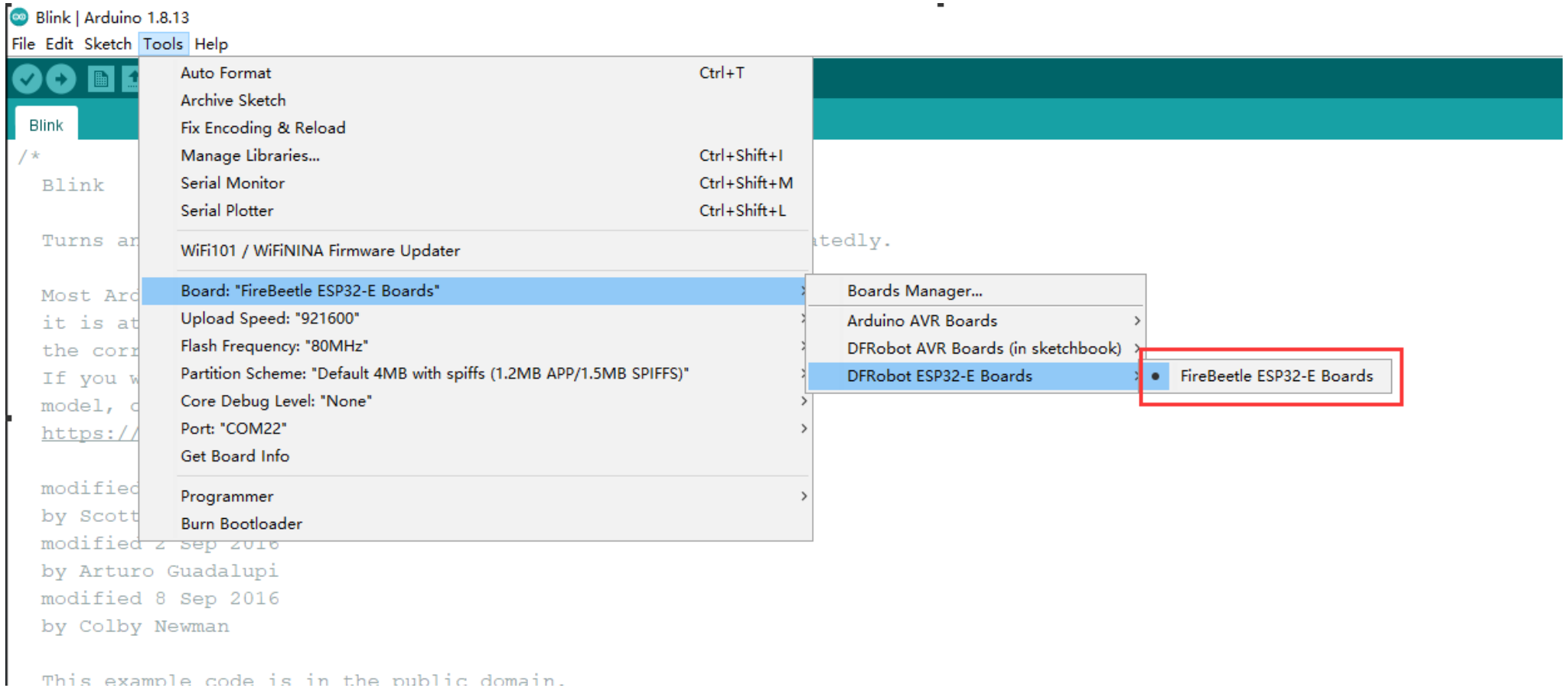
7.2 Blink

This is a blink program for users who use Arduino for the first time. The LED will blink regularly when burning codes into your board. The default blink LED for FireBeetle-ESP32 board is D9/2.

- Select Board and Port

• Click on the Board: Select FireBeetle-ESP32

- Click TOOLS>BOARD; Select FIREBEETLE-ESP32-E
- Click port to select the corresponding port



- Programming


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

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

- Copy the codes above to the Arduino IDE.
- Click the arrow to compile and burn the codes into your board.

Burning Completed

上传成功。

Hash of data verified.

Compressed 3072 bytes to 128...

Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.0 seconds (effective 4915.3 kbit/s)...

Hash of data verified.

Leaving...

Hard resetting via RTS pin...

7.3 Bluetooth Tutorial

The ESP32 supports Bluetooth function. This part will mainly demonstrate how to use two FireBeetle-ESP32-E for realizing Bluetooth Data Transparent Transmission.

```

//This example code is in the Public Domain (or CC0 licensed, at your option.)
//By Victor Tchistiak - 2019
//
//This example demonstrates master mode bluetooth connection and pin
//it creates a bridge between Serial and Classical Bluetooth (SPP)
//this is an extension of the SerialToSerialBT example by Evandro Copercini - 2018
//

#include "BluetoothSerial.h"

BluetoothSerial SerialBT;

String MACadd = "AA:BB:CC:11:22:33";
uint8_t address[6] = {0xAA, 0xBB, 0xCC, 0x11, 0x22, 0x33};
//uint8_t address[6] = {0x00, 0x1D, 0xA5, 0x02, 0xC3, 0x22};
String name = "ESP32test";
char *pin = "1234"; //<- standard pin would be provided by default
bool connected;

void setup() {
  Serial.begin(115200);
  //SerialBT.setPin(pin);
  SerialBT.begin("ESP32master", true);
  //SerialBT.setPin(pin);
  Serial.println("The device started in master mode, make sure remote BT device is on!");

  // connect(address) is fast (upto 10 secs max), connect(name) is slow (upto 30 secs max) as it needs
  // to resolve name to address first, but it allows to connect to different devices with the same name.
  // Set CoreDebugLevel to Info to view devices bluetooth address and device names
  connected = SerialBT.connect(name);
  //connected = SerialBT.connect(address);

```

```
if(connected) {
  Serial.println("Connected Succesfully!");
} else {
  while(!SerialBT.connected(10000)) {

    Serial.println("Failed to connect. Make sure remote device is available and in range, then restart app.");
  }
}
// disconnect() may take upto 10 secs max
if (SerialBT.disconnect()) {
  Serial.println("Disconnected Succesfully!");
}
// this would reconnect to the name(will use address, if resolved) or address used with connect(name/address).
SerialBT.connect();
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

```
//This example code is in the Public Domain (or CC0 licensed, at your option.)
//By Evandro Copercini - 2018
//
//This example creates a bridge between Serial and Classical Bluetooth (SPP)
//and also demonstrate that SerialBT have the same functionalities of a normal Serial

#include "BluetoothSerial.h"

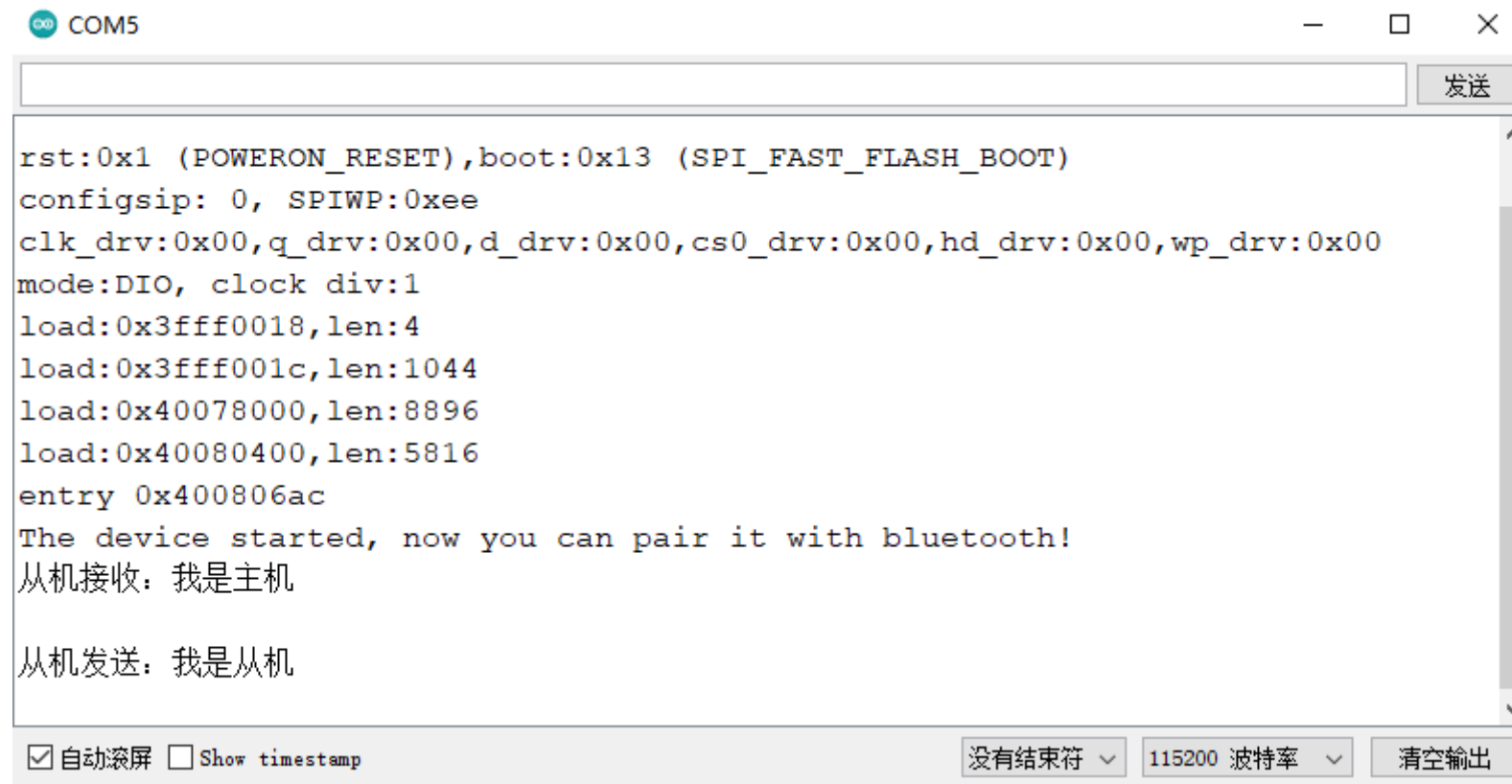
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

Result



COM5

发送

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1044
load:0x40078000,len:8896
load:0x40080400,len:5816
entry 0x400806ac
The device started, now you can pair it with bluetooth!
从机接收: 我是主机

从机发送: 我是从机
```

自动滚屏 Show timestamp

没有结束符 115200 波特率 清空输出

```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1044
load:0x40078000,len:8896
load:0x40080400,len:5816
entry 0x400806ac
The device started in master mode, make sure remote BT device is on!
Connected Successfully!
Disconnected Successfully!
主机发送: 我是主机

主机接收: 我是从机
```

Send "I'm the master" from the mater port.

The slave port displays "The slave receives: I'm the master"

Send "I'm the slave" from the slave port.

The master port displays "The master receives: I'm the slave".

Member Functions

- **SerialBT.begin()**
Description: init Bluetooth module
- **SerialBT.disconnect()**
Description: disconnect device
Return: ture/false
- **SerialBT.connect()**
Description: connect device
Return: ture/false
- **SerialBT.available()**
Description: judge if the Bluetooth received data
- **SerialBT.read()**
Description: read the information received by the Bluetooth
Return: string
- **SerialBT.write()**
Description: send message by Bluetooth

7.4 WiFi Tutorial

The ESP32 supports WiFi function. Here we build a WiFi server with the ESP32, and use the client to connect it to control an LED remotely.


```

/*
  WiFiAccessPoint.ino Create a wifi hotspot, and provide a web service

  Steps:
  1. Connect to the wifi "yourAp"
  2. Visit http://192.168.4.1/H to turn on the LED; Visit http://192.168.4.1/L to turn off the LED
     OR
     Run raw TCP "GET /H" and "GET /L" on PuTTY terminal with 192.168.4.1 as IP address and 80 as port
*/

#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiAP.h>

// Set your wifi and password
const char *ssid = "esp32";
const char *password = "";

WiFiServer server(80);

void setup() {
  pinMode(LED_BUILTIN, OUTPUT); //Set pin LED to output mode
  Serial.begin(115200);
  Serial.println();
  Serial.println("Configuring access point...");

  // Configure wifi and get IP address
  WiFi.softAP(ssid, password);
  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);
}

```

```

Serial.println(myIP);
server.begin();

Serial.println("Server started");
}

void loop() {
  WiFiClient client = server.available(); // listen for incoming clients

  if (client) { // if you get a client,
    Serial.println("New Client."); // print a message out the serial port
    String currentLine = ""; // make a String to hold incoming data from the client
    while (client.connected()) { // loop while the client's connected
      if (client.available()) { // if there's bytes to read from the client,
        char c = client.read(); // read a byte, then
        Serial.write(c); // print it out the serial monitor
        if (c == '\n') { // if the byte is a newline character

          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();

            // the content of the HTTP response follows the header:
            client.print("Click <a href=\"/H\">here</a> to turn ON the LED.<br>");
            client.print("Click <a href=\"/L\">here</a> to turn OFF the LED.<br>");

            // The HTTP response ends with another blank line:
            client.println();
            // break out of the while loop:
            break;
          } else { // if you got a newline, then clear currentLine:
            currentLine = "";
          }
        }
      }
    }
  }
}

```

```
    }
  } else if (c != '\r') { // if you got anything else but a carriage return character,
    currentLine += c;    // add it to the end of the currentLine
  }

  // Check to see if the client request was "GET /H" or "GET /L":
  if (currentLine.endsWith("GET /H")) {
    digitalWrite(LED_BUILTIN, HIGH); // GET /H turns the LED on
  }
  if (currentLine.endsWith("GET /L")) {
    digitalWrite(LED_BUILTIN, LOW); // GET /L turns the LED off
  }
}
}
// close the connection:
client.stop();
Serial.println("Client Disconnected.");
}
}
```

Result

Connect to the WiFi with a phone, and access 192.168.4.1 through the browser. As shown in the figure, the IP address is 192.168.4.1, and the server has been started.

The image shows a terminal window titled "COM22" with standard window controls (minimize, maximize, close). The terminal output is as follows:

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1044
load:0x40078000,len:8896
load:0x40080400,len:5816
entry 0x400806ac

Configuring access point...
AP IP address: 192.168.4.1
Server started
```

At the bottom of the terminal window, there are several controls: a checked checkbox for "自动滚屏" (Auto scroll), an unchecked checkbox for "Show timestamp", a dropdown menu set to "没有结束符" (No terminator), a dropdown menu set to "115200 波特率" (115200 baud rate), and a "清空输出" (Clear output) button.

Use the browser to access the IP address, then you will get the result as shown in the figure below

Click [here](#) to turn ON the LED.
Click [here](#) to turn OFF the LED.

Click to turn the LED on/off.

Member Functions

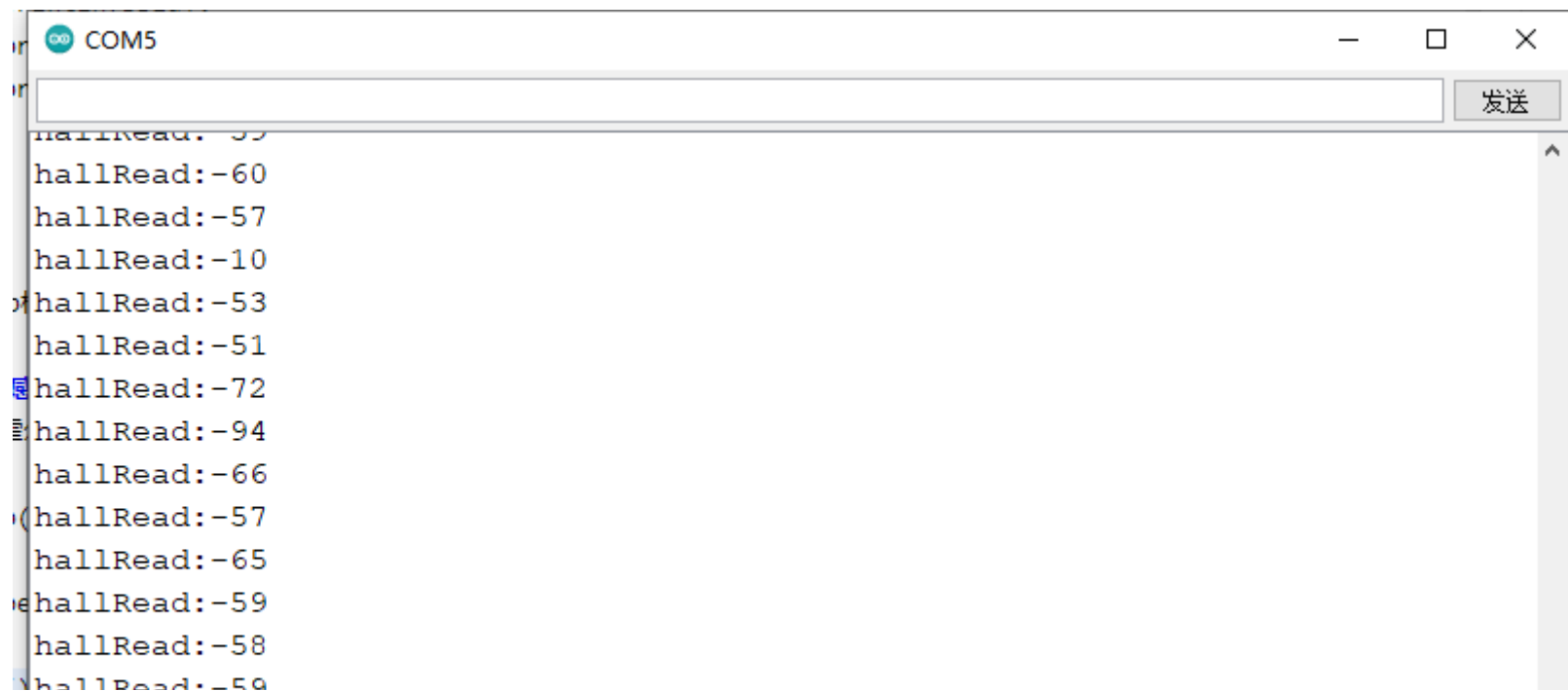
- **begin()**
Description: Init WiFi module
- **softAP(ssid,password)**
Description: Configure WiFi to AP mode, and set name and password
Parameter:
 - ssid: WiFi name in AP mode
 - password: WiFi password in AP mode
- **disconnect()**
Description: disconnect client
- **connect()**
Description: connect client
- **read()**
Description:
Read the data received by WiFi
- **write()**
Description: Send data by WiFi

7.5 Hall Sensor

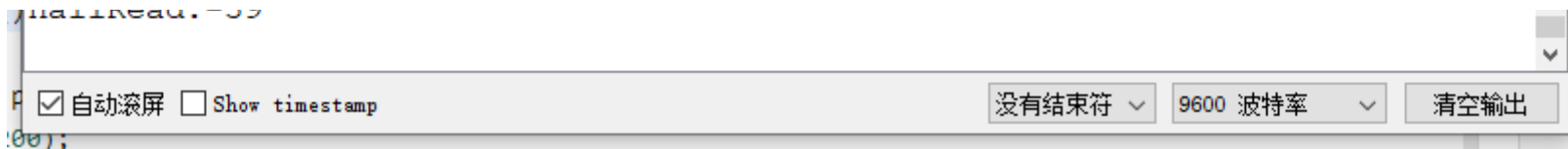
The ESP32 comes with a hall sensor that presents positive/negative number when approaching a magnetic field.

```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.printf("hallRead:%d\n",hallRead());
  delay(200);
}
```

Result



```
COM5
hallRead:-60
hallRead:-57
hallRead:-10
hallRead:-53
hallRead:-51
hallRead:-72
hallRead:-94
hallRead:-66
hallRead:-57
hallRead:-65
hallRead:-59
hallRead:-58
hallRead:-59
```



Member Function

- hallRead()

Description: read the value of built-in hall sensor **Return:** return integer 0-255; Positive number for North pole; Negative number for South pole. The stronger the magnetic field, the greater the absolute value

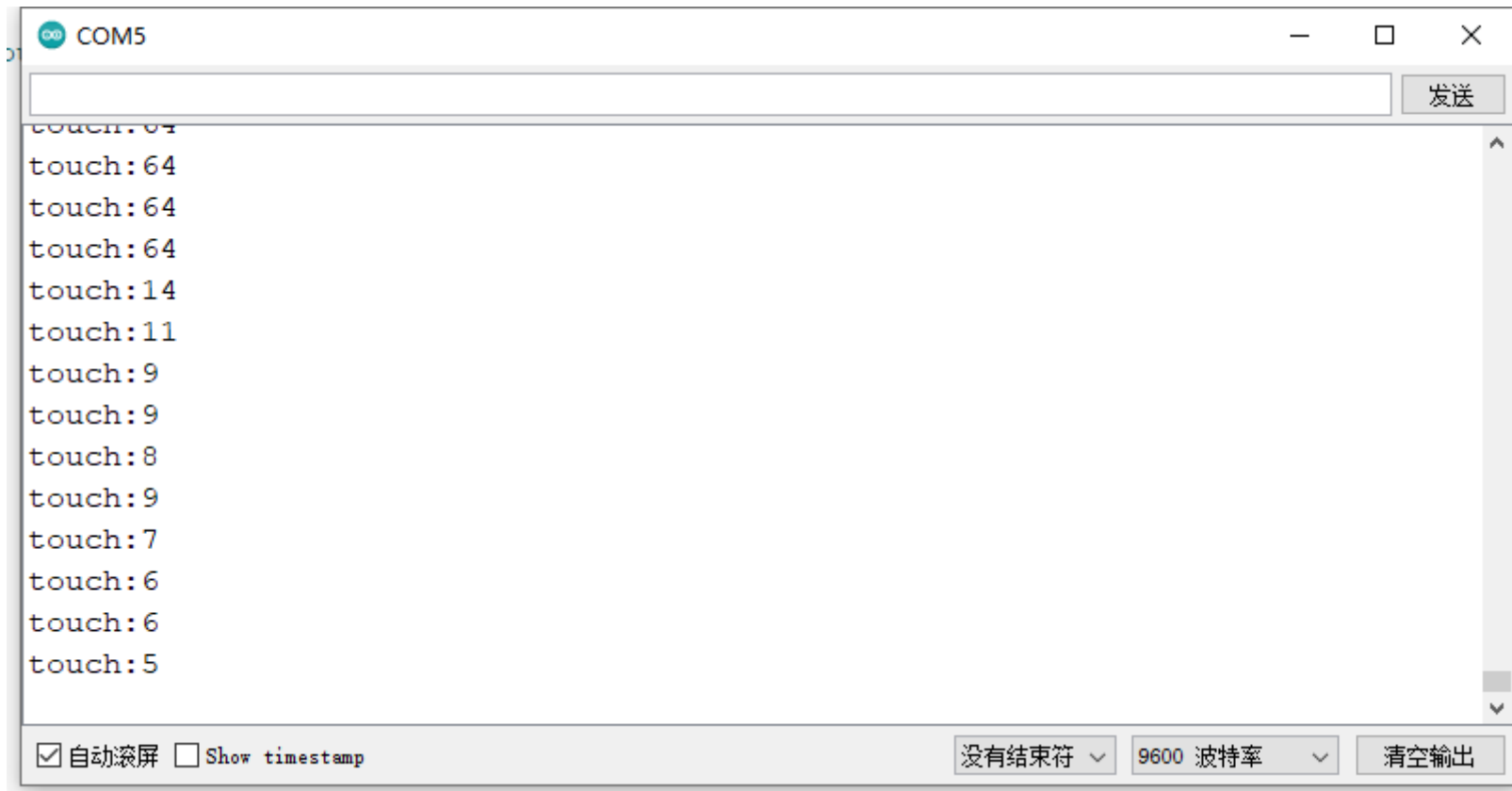
7.5 Capacitive Keys

ESP32 provides the function of capacitive touch sensor. There are 9 touch sensors (T0, T2 ~ T9) available, corresponding to pins 4, 2, 15, 13, 12, 14, 27, 33 and 32 respectively. There is no need to set PinMode. The return value of touchRead() is within 0 ~ 255. The greater the touch force, the smaller the return value. Burning this sample code into your board, use the pin 4/D12 as the touch key, the touch value will be returned through the serial port monitor.

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

void loop()
{
  Serial.printf("touch:%d\n",touchRead(4));
}
```

Result



Member Functions

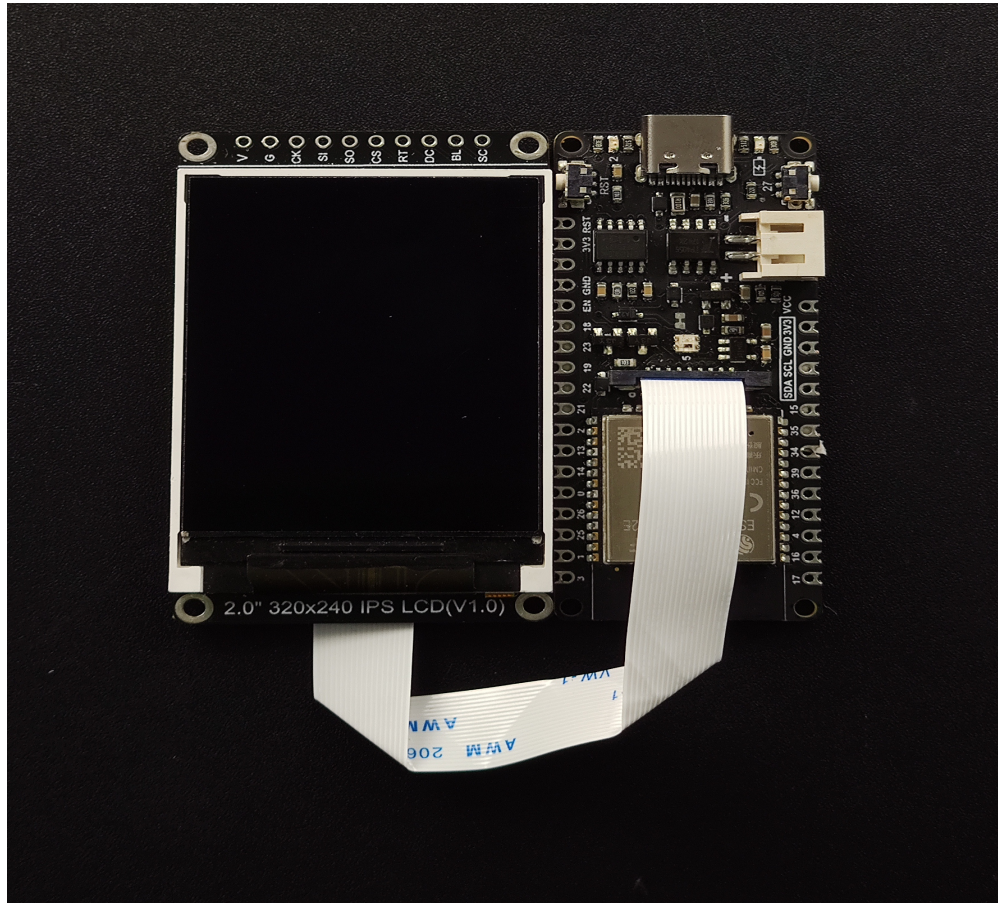
- **TouchRead(pin)**

Description: no need to set PinMode

Parameter:

- pin: touch sensor pin to be called
- *Return**: range 0~255. The stronger the touch force, the greater the return value.

7.6 GDI



This is a DFRobot special GDI display interface. It can be conveniently connected to a screen with 18pin-FPC cable, easy to get started.

The pin list for GDI:

FPC PINS	FireBeetle M0 PINS	Description
VCC	3V3	3.3V

BLK (PWM adjustment) FPC PINS	12/D13 FireBeetle M0 PINS	Backlit Description
GND	GND	GND
SCLK	18/SCK	SPI clock

MOSI	23/MOSI	Master output, slave input
MISO	19/MISO	Master input, slave output
DC	25/D2	Data/command
RES	26/D3	Reset
CS	14/D6	TFT chip select
SDCS	13/D7	SD card chip select
FCS	0/D5	Font library
TCS	4/D12	Touch
SCL	22/SCL	I2C Clock
SDA	21/SDA	I2C Data
INT	16/D11	INT
BUSY-TE	17/D10	Anti-tearing Pin
X1	NC	User-defined pin 1
X2	NC	User-defined pin 2

When using FPC to connect a screen, configure the corresponding pins according to GDL demo. Generally, you only need to configure the three pins for different maincontrollers.

GDI supported screen:

1. 1.54inch 240x240 IPS TFT LCD Display with MicroSD Card Breakout (<https://www.dfrobot.com/product-2072.html>)
2. 2.0inch 0x240 IPS TFT LCD Display with MicroSD Card Breakout (<https://www.dfrobot.com/product-2071.html>)
3. 2.8inch 320x240 IPS TFT LCD Touchscreen with MicroSD (<https://www.dfrobot.com/product-2106.html>)
4. 3.5inch 480x320 TFT LCD Capacitive Touchscreen (<https://www.dfrobot.com/product-2107.html>)

```
/*ESP32 and ESP8266*/  
#elif defined(ESP32) || defined(ESP8266)  
#define TFT_DC 25  
#define TFT_CS 14  
#define TFT_RST 26
```

For more details, please refer to: https://wiki.dfrobot.com/2.0_Inches_320_240_IPS_TFT_LCD_Display_with_MicroSD_Card_Breakout_SKU_DFR0664
(https://wiki.dfrobot.com/2.0_Inches_320_240_IPS_TFT_LCD_Display_with_MicroSD_Card_Breakout_SKU_DFR0664)

7.7 RGB_LED

FastLED is a powerful but easy-to-use Arduino third-party library for controlling LED strips such as WS2812 and LPD8806. At present, FastLED is recognized as one of the most widely used LED controlling libraries by Arduino developers. FireBeetle integrates FastLED into the core library. The following code demonstrates how to use the 5/D8 conneted RGB_LED.

```

#include <FastLED.h>

// How many leds in your strip?
#define NUM_LEDS 1

// For led chips like WS2812, which have a data line, ground, and power, you just
// need to define DATA_PIN.  For led chipsets that are SPI based (four wires - data, clock,
// ground, and power), like the LPD8806 define both DATA_PIN and CLOCK_PIN
// Clock pin only needed for SPI based chipsets when not using hardware SPI
#define DATA_PIN 5
#define CLOCK_PIN 13

// Define the array of leds
CRGB leds[NUM_LEDS];

void setup() {
    // Uncomment/edit one of the following lines for your leds arrangement.
    // ## Clockless types ##
    FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS); // GRB ordering is assumed
}

void loop() {
    //LED light up in red
    leds[0] = CRGB::Red;
    FastLED.show();
    delay(500);
    // LED light up in green
    leds[0] = CRGB::green;
    FastLED.show();
    delay(500);
    // LED light up in blue
    leds[0] = CRGB::Blue;
    FastLED.show();
    delay(500);
}

```

```
    leds[0] = CRGB::Blue;  
    FastLED.show();  
    delay(500);  
}
```

Member Functions

- **leds[0] = CRGB::Red**
Description: set the LED No.0 to red
- **FastLED.show()**
Description: light up or change LED color
- **leds[0].r = 255**
Description: Set the R value of the first LED on the LED strip to 255
- **leds[0].g = 125**
Description: Set the G value of the first LED on the LED strip to 125
- **leds[0].b = 0**
Description: Set the B value of the first LED on the LED strip to 0

7.8 Sleep Mode

In sleep mode, the power consumption can be reduced to 10 μ A (disconnect the low-power pad). The following will demonstrate how to enter the sleep mode at a set time.

```

#define uS_TO_S_FACTOR 1000000ULL /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP 5 /* Time ESP32 will go to sleep (in seconds) */

RTC_DATA_ATTR int bootCount = 0;

/*
Method to print the reason by which ESP32
has been awoken from sleep
*/
void print_wakeup_reason(){
    esp_sleep_wakeup_cause_t wakeup_reason;

    wakeup_reason = esp_sleep_get_wakeup_cause();

    switch(wakeup_reason)
    {
        case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
        case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
        case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
        case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break;
        case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
        default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); break;
    }
}

void setup(){
    Serial.begin(115200);
    delay(1000); //Take some time to open up the Serial Monitor

    //Increment boot number and print it every reboot
    ++bootCount;
    Serial.println("Boot number: " + String(bootCount));
}

```

```
Serial.println( boot number:  + String(bootcount));

//Print the wakeup reason for ESP32
print_wakeup_reason();

/*
First we configure the wake up source
We set our ESP32 to wake up every 5 seconds
*/
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +
" Seconds");

/*
Next we decide what all peripherals to shut down/keep on
By default, ESP32 will automatically power down the peripherals
not needed by the wakeup source, but if you want to be a poweruser
this is for you. Read in detail at the API docs
http://esp-idf.readthedocs.io/en/latest/api-reference/system/deep\_sleep.html
Left the line commented as an example of how to configure peripherals.
The line below turns off all RTC peripherals in deep sleep.
*/
//esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
//Serial.println("Configured all RTC Peripherals to be powered down in sleep");

/*
Now that we have setup a wake cause and if needed setup the
peripherals state in deep sleep, we can now start going to
deep sleep.
In the case that no wake up sources were provided but deep
sleep was started, it will sleep forever unless hardware
reset occurs.
*/
Serial.println("Going to sleep now");
Serial.flush();
esp_deep_sleep_start();
Serial.println("This will never be printed");
```

```
}  
  
void loop(){  
  //This is not going to be called  
}
```

Member Functions

- **esp_sleep_get_wakeup_cause()**
Description: Check which wake-up source triggered a wake-up from sleep mode
- **esp_deep_sleep_start()**
Description: Enter sleep mode
- **esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR)**
Description: user timer to start wake-up from deep sleep.

8. Arduino Tutorial Basics

8.1 GPIO

Digital IO

- **digitalRead(pin)**
Description: Reads the value from a specified digital pin, either HIGH or LOW.
Parameter:
 - pin: the Arduino pin number you want to read
- **digitalWrite (pin,value)**
Description: Write a HIGH or a LOW value to a digital pin. Parameter:

- pin: the Arduino pin number.
 - value: HIGH or LOW.
-
- **pinMode(pin, mode)**
Description: Configures the specified pin to behave either as an input or an output
Parameter:
 - pin: the Arduino pin number to set the mode of.
 - mode: INPUT, OUTPUT, or INPUT_PULLUP.

Control LED via Keys

Analog IO

- **AnalogRead (pin)**
Description: Reads the value from the specified analog pin.
Parameter:
 - pin: the name of the analog input pin to read
- **AnalogReference (type)**
Description: Configures the reference voltage used for analog input
Parameter:
 - type
- **AnalogWrite (pin, value)**
Description: Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin.

Parameter:

- pin: the Arduino pin to write to. Allowed data types: int.
- value: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int.

8.2 Serial

- **Serial.begin(speed)**

Description: Sets the data rate in bits per second (baud) for serial data transmission. For communicating with Serial Monitor, make sure to use one of the baud rates listed in the menu at the bottom right corner of its screen.

Parameter:

- speed: in bits per second (baud). Allowed data types: long.

- **Serial.available()**

Description: Get the number of bytes (characters) available for reading from the serial port.

Input

- **Serial.read()**

Description: Reads incoming serial data.

- **Serial.peek()**

Description: Returns the next byte (character) of incoming serial data without removing it from the internal serial buffer.

Output

- **Serial.print()**

Description: Prints data to the serial port

- **Serial.println()**

Description: Prints data to the serial port followed by a carriage return character and a newline character

Software Serial

- `SoftwareSerial()`

Running time Function

- `micros ()`

Description: Returns the number of microseconds since the Arduino board began running the current program.

- `millis ()`

Description: Returns the number of milliseconds passed since the Arduino board began running the current program.

Delay Functions

- `delay (ms)`

Description: Pauses the program for the amount of time (in milliseconds) specified as parameter.

Parameter: ms: the number of milliseconds to pause. Allowed data types: unsigned long.

- `delayMicroseconds (us)`

Description: Pauses the program for the amount of time (in microseconds) specified by the parameter. There are a thousand microseconds in a millisecond and a million microseconds in a second.

Parameter: us: the number of microseconds to pause. Allowed data types: unsigned int.

8.3 Tone Functions

- `tone(pin, frequency, duration)`

Description: Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to `noTone()`. The pin can be connected to a piezo buzzer or other speaker to play tones.

Parameter:

- pin: the Arduino pin on which to generate the tone.
- frequency: the frequency of the tone in hertz. Allowed data types: unsigned int.

- duration: the duration of the tone in milliseconds (optional). Allowed data types: unsigned long.

- **noTone(pin)**

Description: Stops the generation of a square wave triggered by tone(). Has no effect if no tone is being generated.

Parameter:

- pin: the Arduino pin on which to stop generating the tone
- frequency: the frequency of the tone in hertz. Allowed data types: unsigned int.
- duration: the duration of the tone in milliseconds (optional). Allowed data types: unsigned long.

8.4 Interrupt

- **attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)**

Description: External Interrupts

Parameter:

- pin: the Arduino pin number.
- ISR: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.
- mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

- **detachInterrupt(digitalPinToInterrupt(pin))**

Description: Turns off the given interrupt.

Parameter:

- interrupt: the number of the interrupt to disable
- pin: the Arduino pin number of the interrupt to disable

- **interrupts ()**

Description: Re-enables interrupts (after they've been disabled by `noInterrupts()`). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.。

- **`noInterrupts ()`**

Description: Disables interrupts (you can re-enable them with `interrupts()`). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.

8.5 I2C

IIC Master/Slave Pin

Different from the one-to-one communication mode of serial port, bus communication is usually divided into master and slave. During communication, the master is responsible for starting and terminating data transmission, and also outputs clock signal; the slave is addressed by the host and responds to the communication request of the host. The communication rate is controlled by the host, and the master outputs clock signal for all slaves on the bus through SCL pin. At the same time, I2C is a half duplex communication mode, that is, the devices on the bus transmit communication data through SDA pins, and the sending and receiving of data are controlled by the host computer. Esp32 has two I2C controllers (also known as ports) that handle communication on both I2C buses. Each I2C controller can run as a master or slave. Pin 21 is default to SDA, pin 22 to SCL.

- **`begin(address)`**

Description: Initiate the IIC and join the I2C bus as a master or slave.

Parameter:

- **`read()`**

Description: In the host, when the `requestfrom()` function is used to send the data request signal, the `read()` function is needed to obtain the data; in the slave machine, the function is used to read the data sent by the host.

Parameter:

- **`available()`**

Description: Get the number of bytes (characters) of the received value

Parameter:

- **write: ()**

Description: When it is in host state, the host will add the data to be sent to the sending queue; when it is in the slave state, the slave will send the data to the requesting host.

Parameter:

- valuc: send as a single byte
- string: send as a series of bytes
- data: an array to send as a series of bytes

- **requestFrom(address, quantity)**

Description: The host sends a data request signal to the slave. After using requestfrom(), the slave can use onRequest() to register an event to respond to the host's request. The host can read the data through the available() and read() functions.

Parameter:

- quantity: the number of bytes to request
- address: ddress of the device to request bytes from

- **beginTransmission(address)**

Description: Begin a transmission to the slave device with the given address. Subsequently, queue bytes for transmission with the write() function and transmit them by calling endTransmission().

Parameter:

- address: address of the device to transmit to

- **endTransmission**

Description: Ends a transmission

Parameter:

- stop: boolean. true will send a stop message, releasing the bus after transmission. false will send a restart, keeping the IIC connection active.

- **onReceive() Description:** 8. onReceive()

Function: Registers an event to be triggered when a slave device receives a transmission from a master.

Grammar: Wire.onReceive(handler)

Parameter:

- handler: the event to be triggered when the slave receives data; this should take a single int parameter (the number of bytes read from the master) and return nothing, e.g.: void myHandler(int numBytes)

- **onRequest(handler)**

Description: Register an event to be triggered when a master requests data from this slave device. **Grammar:** Wire.onRequest(handler)

Parameter:

- handler: the event to be triggered, takes no parameters and returns nothing, e.g.: void myHandler()

8.6 SPI

SPI Pin

ESP32 has four SPI peripherals: SPI0, SPI1, HSPI and VSPI.

1. SPI0 is used for flash cache, and ESP32 maps the attached SPI flash device to memory.
2. SPI1 and SPI0 share one hardware line, SPI1 is used to write flash chip.
3. HSPI and VSPI can be used arbitrarily.
4. SPI1, HSPI and VSPI have three chip selection lines, so as SPI host, ESP32 is allowed to drive up to three SPI devices.

- **begin()**

Description: initialize SPI communication. after calling this function, SCK.MOSI, and SS pins will be set to the output mode, and the SCK and

MOSI pins will be pulled down and the SS pin will be pulled up.

- **end()**

Description: turn off SPI BUS communication

- **setBitOrder()**

Description: Set transmission order

- **setBitOrder()**

Description: Set communication clock. The clock signal is generated by the master, and the slave is not configured. But the SPI clock frequency of the master should be within the processing speed range allowed by the slave.

9. Advanced Tutorials

9.1 How to use SD Library

SD Class

- **begin(cspin)**

Description: Initializes the SD library and card. This begins use of the SPI bus (digital pins 11, 12, and 13 on most Arduino boards; 50, 51, and 52 on the Mega) and the chip select pin, which defaults to the hardware SS pin (pin 10 on most Arduino boards, 53 on the Mega). Note that even if you use a different chip select pin, the hardware SS pin must be kept as an output or the SD library functions will not work.

Parameter: cspin: the Arduino pin connected to the chip select line of the SD card.

Return: boolean type. True on success; false on failure

- **exists()**

Description: Tests whether a file or directory exists on the SD card. **Grammar:** SD. exists(filename)

Parameter:

- filename: the name of the file to test for existence, which can include directories (delimited by forward-slashes, /)
- *Return**: boolean type, true if the file or directory exists, false if not

- **open()**

Description: Opens a file on the SD card. If the file is opened for writing, it will be created if it doesn't already exist (but the directory containing it must already exist). **Grammar:** SD.open(filename) SD.open(filename, mode)

Parameter:

filename: the name the file to open, which can include directories (delimited by forward slashes, /) - char * mode (optional): the mode in which to open the file, defaults to FILE_READ - byte. one of:

FILE_READ: open the file for reading; FILE_WRITE: open the file for reading and writing.

Return: a File object referring to the opened file: if the file couldn't be opened, this object will evaluate to false in a boolean

FILE_WRITE: open the file for reading and writing.

Return: a File object referring to the opened file;Return false if the file cannot be opened.

- **remove()**

Description: Remove a file from the SD card. If the file didn't exist, the return value is unspecified, so it is better to use SD.Exists (file name) to detect whether the file exists before removing the file.

Grammar: SD.remove(filename)

Parameter:

- filename: the name of the file to remove, which can include directories (delimited by forward-slashes, /)
- *Return*:* boolean type. True if the removal of the file succeeded, false if not.

- **mkdir(filename)**

Description: Create a directory on the SD card.

Parameter:

- filename,the name of the directory to create, with sub-directories separated by forward-slashes, /
- *Return**: boolean type. True if the creation of the directory succeeded, false if not.

- **rmdir(filename)**

Description: Remove a directory from the SD card. The directory must be empty. **Grammar:** SD.rmdir(filename)

Parameter:

- filename: the name of the directory to remove, with sub-directories separated by forward-slashes, /
- *Return**: booleantype. True if the removal of the directory succeeded, false if not.

File Class

The file class provides the function of reading / writing files. The function of this class is very similar to the that of serial port related functions used before. The member functions are as follows.

- **available()**

Description: Check if there are any bytes available for reading from the file. **Grammar:** file. available()

Parameter:

- file:an instance of the File class
- *Return**: the number of bytes available

- **close()**

Description: Close the file, and ensure that any data written to it is physically saved to the SD card. **Grammar:** file. close()

Parameter:

- file:an instance of the File class
- *Return**: none

- **flush()**

Description: Ensures that any bytes written to the file are physically saved to the SD card. This is done automatically when the file is closed.

Syntax: file.flush

Parameter:

- file: an instance of the File class
- *Return**: none

- **peek()**

read()

Description: Read a byte from the file without advancing to the next one.

Parameter:

- file: an instance of the File class
- *Return**: The next byte (or character), or -1 if none is available.

- **position()**

Description: Get the current position within the file (i.e. the location to which the next byte will be read from or written to). **Syntax:** file.

position()

Parameter:

- file: an instance of the File class
- *Return**: the position within the file

- **print()**

Description: Print data to the file, which must have been opened for writing. **Syntax:** file. print(data)file. print(data, BASE)

Parameter:

- file: an instance of the File class
- data: the data to print (char, byte, int, long, or string)
- BASE(optional): the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

- **println()**

Description: Print data, followed by a carriage return and newline, to the File, which must have been opened for writing. **Syntax:** file.

println(data)file,println(data, BASE)

Parameter:

- file:an instance of the File class
- data (optional): the data to print (char, byte, int, long, or string)
- BASE (optional): the base in which to print numbers: BIN for binarv (base 2). DEC for decimal (base 10). OCT for octal (base 8). HEX for

hexadecimal (base 16).

- **seek()**

Description: Seek to a new position in the file, which must be between 0 and the size of the file (inclusive). **Syntax:** file.seek(pos)

Parameter:

- file: an instance of the File class
- pos: the position to which to seek
- *Return**: true for success, false for failure (boolean)

- **size()**

Description: Get the size of the file. **Syntax:** file.size()

Parameter:

- file: an instance of the File class
- *Return**: the size of the file in bytes

- **read()**

Description: Read from the file. **Syntax:** file.read **Parameter:**

- file: an instance of the File class
- *Return**: The next byte (or character), or -1 if none is available.

- **write()**

Description: Write data to the file.

Syntax: file.write(data)file.write(buf, len)

Parameter:

- file: an instance of the File class

- data: the byte, char, or string (char*) to write
- buf: an array of characters or bytes
- len: the number of elements in buf

- *Return**: the number of bytes written

- **isDirectory()**

Description: Reports if the current file is a directory or not **Syntax:** file.isDirectory()

Parameter:

- file: an instance of the File class
- *Return**:boolean. True if the file is a directory, false if not

- **openNextFile()**

Description: Reports the next file or folder in a directory. **Syntax:** file.openNextFile()

Parameter:

- file: an instance of the File class that is a directory
- *Return**: the next file or folder in the path

- **rewindDirectory()**

Description: Back to the first file in the directory **Syntax:** file.rewindDirectory()

Parameter:

- file: an instance of the File class.
- *Return**: None

9.2 IR Remote Control

IRrecv Class

IRrecv class can be used to receive and decode infrared signals. Before using this class, you need to instantiate an object of the class. Its

IRrecv class can be used to receive and decode infrared signals. Before using this class, you need to instantiate an object of the class. Its member functions are as follows.

- **IRrecv()**

Description: construct function of IRrecv class. Specify the pin the IR receiver connects to on Arduino. **Syntax:** IRrecvobject(recvpin)

Parameter:

- object: user-defined object name
- recvpin: Arduino pin connect to IR receiver

- **enableIRIn()**

Description: Init IR decoder **Syntax:** IRrecv.enableIRIn()

Parameter:

- IRrecv: an object of IRrecv class

- **decode()**

Description: detect if an IR signal is received **Syntax:** IRrecv.decode(&.results)

Parameter:

- IRrecv: an object of IRrecv class
- results: an object of decode_results class
- *Return**: int type. Returns 0 if a code was received, or 1 if nothing received yet

- **resume()**

Description: Receive the next ir code **Syntax:** IRrecv.resume()

Parameter:

- IRrecv: an object of IRrecv class.
- *Return**: none

IRsend class

The IRsend class can encode and send infrared signals.

- **IRsend(object)**

Description: Construct function of IRsend class **Parameter:**

- object: an object of IRsend class

- **sendNEC()**

Description: Sends the specified value in NEC encoded format. **Syntax:** IRsend.sendNEC(data, nbits)

Parameter:

- IRsend: an object of IRsend class. data: encode value to send
- nbits: number of encoding bits

- **sendSony()**

Description: Send a code in Sony format. **Syntax:** IRsend.sendSony(data, nbits)

Parameter:

- IRsend: an object of IRsend class. data: encode value to send
- nbits: number of encoding bits

- **sendRaw()**

Description: Send a raw code. **Syntax:** IRsend.sendRaw(buf,len,hz)

object:

- IRsend:an object of IRsend class
- buf: store the array of original code
- len: the length of the array
- hz: ir transmitting frequency

9.3 WIFI

ESP32 supports WiFi connection of both STA and AP mode.

- STA mode: ESP32 module connects Internet through router, and mobile phone or computer realizes remote control of equipment through Internet.
- AP mode: ESP32 module acts as a hot spot to enable the communicate between the module and mobile phone/computer, and realize the wireless LAN controlling.
- STA+AP mode: The coexistence mode of the two modes can realize seamless switching through Internet control, which is convenient for operation.


```

#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
HTTPClient http;
const char* ssid="dfrobotOffice";
const char* password="dfrobot2011";
const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 28800;
const int  daylightOffset_sec = 0;
DynamicJsonDocument doc(1024);
DynamicJsonDocument doc1(1024);

void printLocalTime(){
  struct tm timeinfo;
  if(!getLocalTime(&timeinfo)){
    Serial.println("Failed to obtain time");
    return ;
  }
  Serial.println(&timeinfo,"%A, %B %d %Y %H:%M:%S");
}

void printLocalWeather(){
  http.begin("http://www.weather.com.cn/data/cityinfo/101270101.html");
  int httpCode = http.GET();
  if(httpCode == HTTP_CODE_OK){
    String pageData = http.getString();
    //Serial.println(pageData);
    deserializeJson(doc,pageData);
    JsonObject obj = doc.as<JsonObject>();
    String weatherInfo = obj["weatherinfo"];
    deserializeJson(doc1,weatherInfo);
    JsonObject obj1 = doc1.as<JsonObject>();

```

```

    JSONObject obj1 = doc1.as<JSONObject>();
    String city = obj1["city"];
    String temp1 = obj1["temp1"];
    String temp2 = obj1["temp2"];
    String weather = obj1["weather"];

    String cityInfo = "Address: " + city;
    String tempInfo = " Temperature: " + temp1 + "~" + temp2;
    String cityWeatherinfo = " Weather: " + weather;
    Serial.println("The weather conditions obtained are as follows: ");
    printLocalTime();
    Serial.print(cityInfo);
    Serial.print(tempInfo);
    Serial.println(cityWeatherinfo);
} else {
    Serial.println("GET ERR");
}
http.end();
}

```

```

void setup() {
Serial.begin(115200);
    Serial.printf("Connecting to %s",ssid);
    WiFi.begin(ssid,password);
    while(WiFi.status() != WL_CONNECTED){
        delay(500);
        Serial.print(".");
    }
    Serial.println(" CONNECTED");
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

    // printLocalWeather();
}

```

```

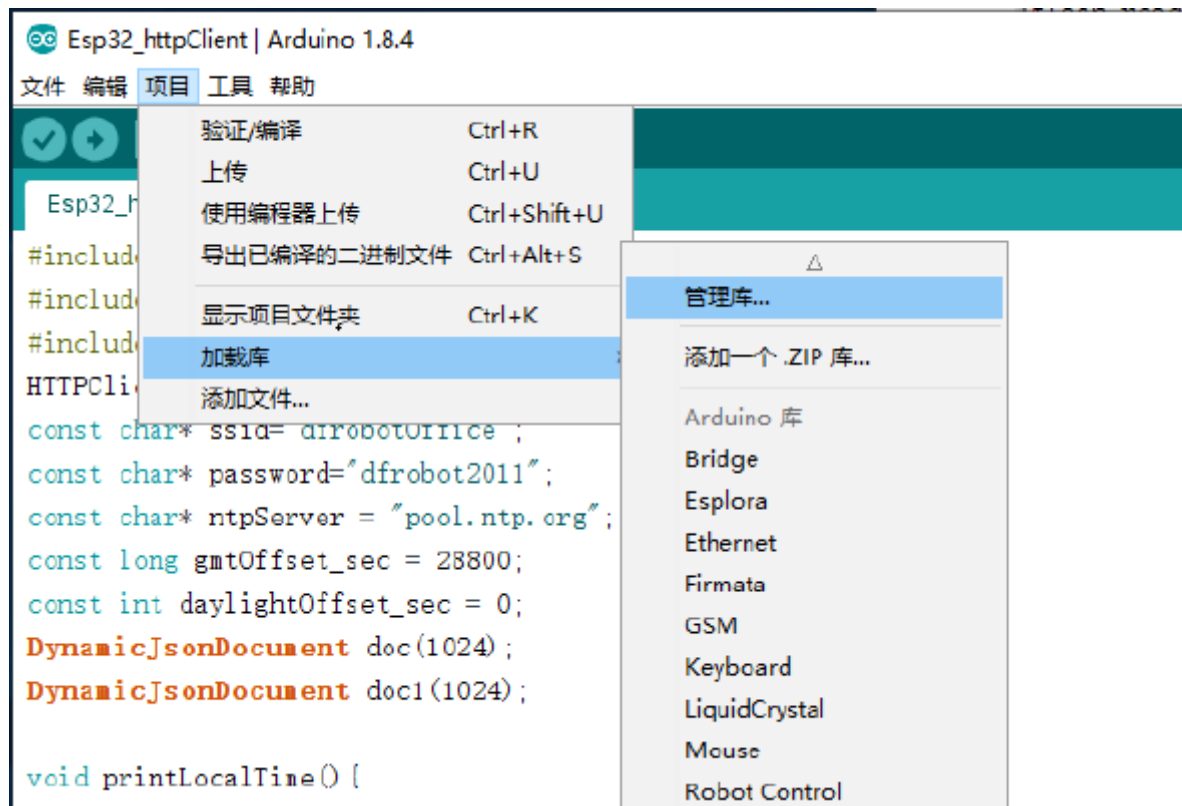
void loop() {
    if(WiFi.status() == WL_CONNECTED){
        printLocalWeather();
    } else {

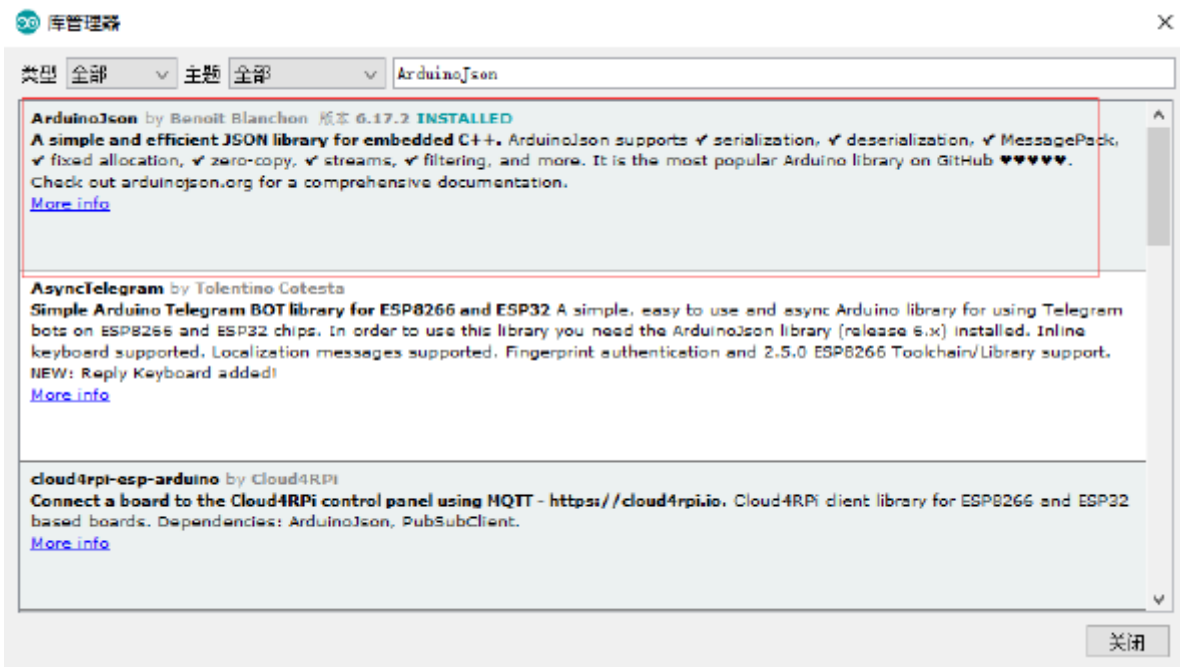
```

```
Serial.println("WiFi Disconnect");
}
}
```

Description: This demo shows how to obtain the network time through WiFi function and get the weather by visiting the <http://www.weather.com.cn/data/cityinfo/101010100.html> (<http://www.weather.com.cn/data/cityinfo/101010100.html>) . "101010100" in this interface is the city code.

Note: you need to download arduino json library in this example, as shown below.





Result

```
ets Jun 8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun 8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00, q_drv:0x00, d_drv:0x00, cs0_drv:0x00, hd_drv:0x00, wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018, len:4
load:0x3fff001c, len:1044
load:0x40078000, len:8896
load:0x40080400, len:5816
entry 0x400806ac
```

Start

```
Connecting to dfrobotOffice. CONNECTED
```

获得天气情况如下：

```
Thursday, November 26 2020 13:43:34
```

```
地点：成都 温度：16℃~28℃ 天气状况：阵雨转晴
```

→ Demo result

WiFiClass

- **begin()**
Description: enable WiFi and connect to the specified wifi network
- **status()**
Description: Get WiFi status

HTTPClient

- **begin()**
Description: Analyze the incoming URL parameter information
- **GET()**
Description: Send get request to server
- **end()** Description: end this connection

DynamicJsonDocument

- **deserializeJson ()**
Description: analyze Json
- **as()**
Description: Get the top node and convert it to T-type

9.4 Bluetooth

This demo creates a BLE_Server that can provide data and send notification for the client. When the server receives the data from the client, it will send the received data to the client in the form of notification. That is, the notification service provided by BLE server in this demo only serves to return the received client data.

```

#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#define SERVICE_UUID          "DFCD0001-36E1-4688-B7F5-EA07361B26A8"
#define CHARACTERISTIC1_UUID  "DFCD000A-36E1-4688-B7F5-EA07361B26A8"
bool deviceConnected = false;
BLEServer *pServer;
BLEService *pService;
BLECharacteristic* pCharacteristic;
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
        deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
        deviceConnected = false;
    }
};
class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        std::string value = pCharacteristic->getValue();

        if (value.length() > 0) {
            Serial.println("*****");
            Serial.print("New value: ");
            for (int i = 0; i < value.length(); i++){
                Serial.print(value[i]);
            }
            Serial.println();
            Serial.println("*****");
            pCharacteristic->write(value);
        }
    }
};

```

```

        pCharacteristic->notify();
    }
}
};
void setupBLE()
{
    BLEDevice::init("DFRobot_ESP32"); //Create BLE device
    pServer = BLEDevice::createServer(); //Create BLE server
    pServer->setCallbacks(new MyServerCallbacks()); //Set the callback function of the server
    pService = pServer->createService(SERVICE_UUID); //Create BLE service
    pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC1_UUID,
        BLECharacteristic::PROPERTY_READ|
        BLECharacteristic::PROPERTY_NOTIFY|
        BLECharacteristic::PROPERTY_WRITE); //Create the characteristic value of the service
    pCharacteristic->setCallbacks(new MyCallbacks()); //Set the callback function of the characteristic value
    pCharacteristic->addDescriptor(new BLE2902());
    pCharacteristic->setValue("Hello DFRobot");
    pService->start();
    BLEAdvertising *pAdvertising = pServer->getAdvertising();
    pAdvertising->start();
}
void setup() {
    Serial.begin(115200);
    setupBLE();
}
void loop() {
    delay(3000);
}

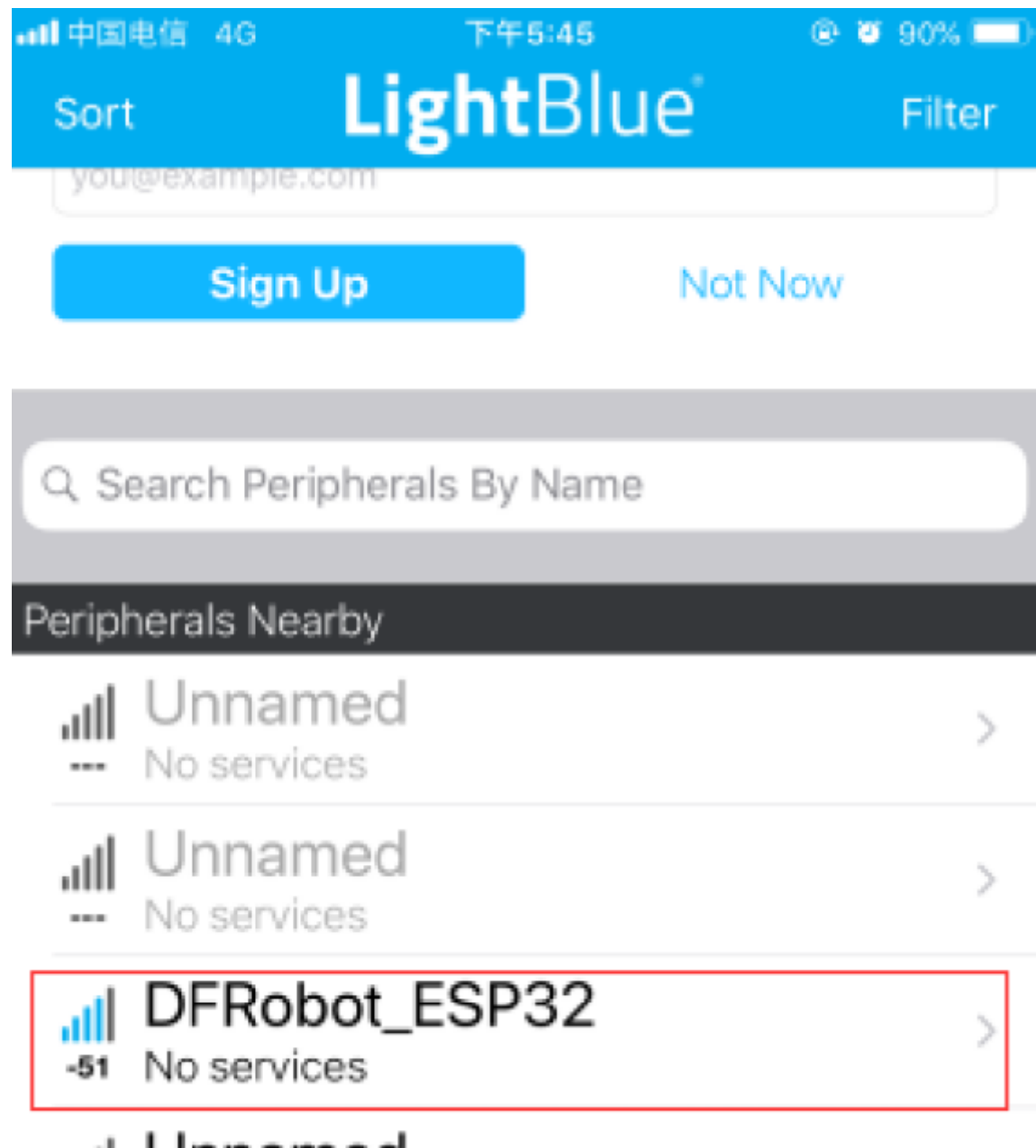
```





BLE Usage





In this demo, the module FireBeetle ESP32-E acts as the BLE server, and the client could be a mobile phone. Install a BLE helper on the phone to establish BLE connection with the ESP32 module. Here we use the Light Blue on iPhone to show you how to do that, such kind of Bluetooth


software can also be found on Android phones.

The operation on the client is as follows:



 Unnamed -78 1 service	>
 Unnamed No services	>
 Unnamed No services	>
 Unnamed No services	>

 **Peripherals**  **Virtual Devices**  **Log**  **More**

中国电信 4G 下午5:45 90% 

[Back](#) **Peripheral** [Clone](#)

DFRobot_ESP32

UUID: A033AC3A-92A7-3170-6AF5-2FCF70DD5469

Connected

ADVERTISEMENT DATA [Show](#)

ADVERTISEMENT DATA

SHOW

UUID:

DFCD0000-0000-...00-00805F9B34FB

0xDFCD000A-0000-1000-8000-00805F9B34FB >

Properties: Read Write Notify



中国电信 4G 下午5:46 90%

< DFRobot_ESP32 0xDFCD000A-0000-1... Hex

DFRobot_ESP32 Set the format of data to receive/send

0xDFCD000A-0000-1000-8...

UUID: DFCD000A-0000-1000-8000-00805F9B34FB

Connected


Read data from the server

READ/NOTIFIED VALUES

Read again

Receive notification

Listen for notifications

 Cloud Connect



0x48656C6C6F204446526F626F74

17:46:00.424

WRITTEN VALUES

Write new value

Send data to the server from here

DESCRIPTORS

1

Client Characteristic Configuration

PROPERTIES

Read

Write



Peripherals



Virtual Devices



Log



More

COM3



发送

```
ets Jun  8 2016 00:22:57
```

```
rst:0x1 (POWERON_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
```

```
flash read err, 1000
```

```
ets_main.c 371
```

```
ets Jun  8 2016 00:22:57
```

```
rst:0x10 (RTCWDT_RTC_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
```

```
configsip: 0, SPIWP:0xee
```

```
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
```

```
mode:DIO, clock div:1
```

```
load:0x3fff0018,len:4
```

```
load:0x3fff001c,len:1044
```

```
load:0x40078000,len:8896
```

```
load:0x40080400,len:5816
```

```
entry 0x400806ac
```

```
*****
```

```
New value: Hello DFRobot
```

```
*****
```



Received data from the client

自动换屏

没有结束符

115200 波特率

Clear output

BLEDevice

- **init()**
Description: create a BLE device
- **createServer()**
Description: create BLE server

BLEServer

- **createService()**
Description: create a BLE service
- **setCallbacks()**
Description: create server callback function
- **start()**
Description: turn off server
- **getAdvertising()**
Description: configure advertising function

BLEService

- **createCharateristic()**
Description: create the characteristic value of the service

BLECharacteristic

- **setCallbacks()**
Description: set characteristic value callback function

Description: set characteristic value callback function

- **addDescriptor()**

Description:

- **setValue()**

Description: Set the value of the characteristic vlaue

- **getValue()**

Description: get the value of the characteristic vlaue

- **notify()**

Description: send notification

BLEAdvertising

- **start()** Description: start advertising

10 Using with IFTTT

What is IFTTT?

If This Then That (commonly known as IFTTT, /ift/), is a web-based service that allows users to create chains of conditional statements triggered by changes that occur within other web services. It is both a website and a mobile app of free service with the following slogan: "Put the Internet to work for you". IFTTT aims to help people use the open API of various websites to monitor the triggers set by users. If triggers are triggered, actions set by users will be executed. Usually, we can create n applets to meet our various automation needs.

IFTTT

Recipe

If This Then That

Trigger

Action

Email Sending

Requirements

- Hardware
 - FireBeetle ESP32-E x1
 - FireBeetle Gravity IO Expansion Board x1
- Software
 - Download IFTTT Library and Sample Code (<https://dfimg.dfrobot.com/nobody/wiki/52ca4c18a70580ad4fa9766be442cfda.rar>)

Configure IFTTT

- Configure IoT platform
 1. Enter IFTTT website (<https://ifttt.com/>), register an account if you don't have one. Then Sign in.

For business Explore Sign in **Sign up**

Make your work more productive



The image shows a sign-up form for IFTTT. It features a text input field with the placeholder text "Enter your email" and a "Get started" button. Below the input field, there is a horizontal line with the word "or" in the center. Underneath the line, there are three social media login options: "Continue with Apple" (with the Apple logo), "Continue with Google" (with the Google logo), and "Continue with Facebook" (with the Facebook logo).

2. The following interface will appear when you signed in.



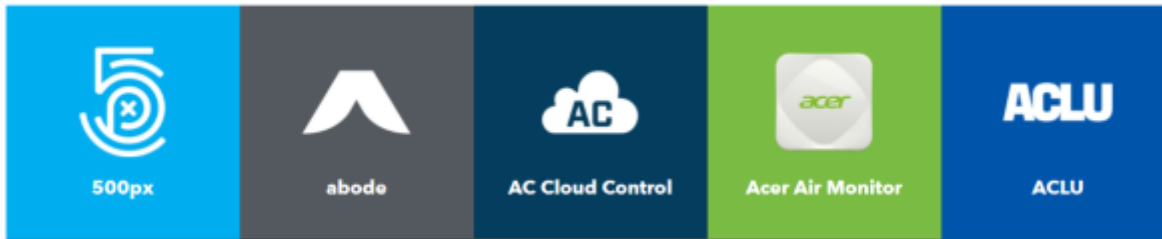
3. Click "Create" to enter the interface below.



4. Click "if this" and input "webhooks" in the search bar.

Choose a service

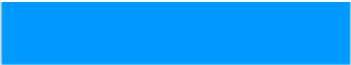
Q Search services



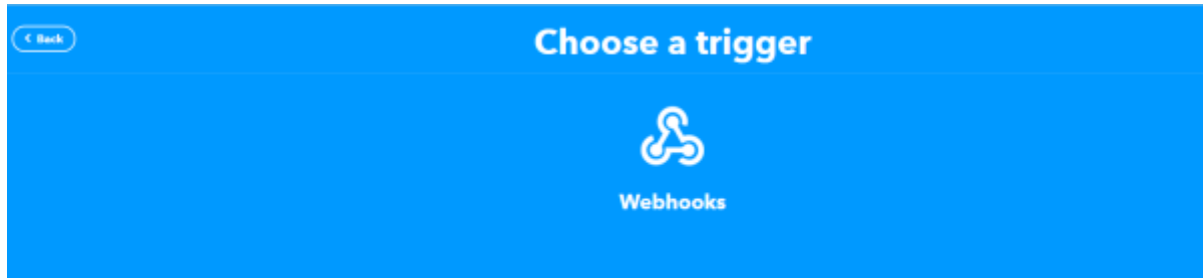
Choose a service

Q webhooks





5. The following interface when entering for the first time, click "Receive a web request".



Receive a web request
This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

6. Fill in the Event Name with "Message", then click "create this".



Receive a web request

This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Event Name

message

The name of the event, like "button_pressed" or "front_door_opened"

 **Create trigger**

7. The webpage will return back automatically. Click "that" and select "Email". Then click "send me an email".

Choose action service

Step 3 of 6

Q email



Email



Email Digest

Choose an action



Email

Send me an email

**This Action will send you
an HTML based email.
Images and links are
supported.**

8. Click "Connect", fill in your email address, and click "send PIN" to send a PIN code to your email box.



Connect Email

Enter the email address you would like to use for all of your Email Applets.

Email address



Send PIN

Enter the email address you would like to use for all of your Email Applets.

Email address

Please enter the 4-digit PIN you received below.

PIN

Connect

Retry

9. Check your email to find the PIN code, and fill it in the webpage, then click "Connect".



Connect Email

Enter the email address you would like to use for all of your Email Applets.

Email address

Please enter the 4-digit PIN you received below.
PIN

Connect

Retry

10. Click "Send me an Email" to edit the email.



Choose action

Step 4 of 6

Send me an email

This Action will send you an HTML based email. Images and links are supported.

Don't see what you're looking for?

Suggest a new action

11. You can write the content to be sent to your email box in the interface below. Click "Create action".

Complete action fields

Step 5 of 6

Subject

The event named "
EventName" occurred on
the Maker Webhooks
service

Add ingredient

Body

What: EventName


When: OccurredAt

Extra Data: Value1 ,
Value2 , Value3 ,


Add ingredient

Create action

12. Click "Continue" to review, then click "Finish".

If  Receive a web request [Delete](#)



Then  Send me an email [Delete](#)



Continue

Review and finish



Applet Title

If Maker Event "message", then Send me an email at



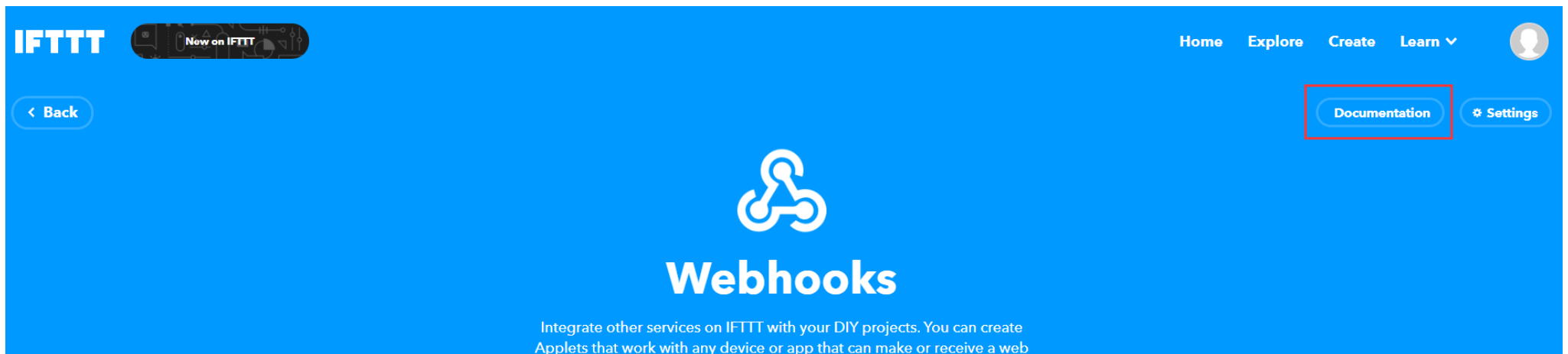
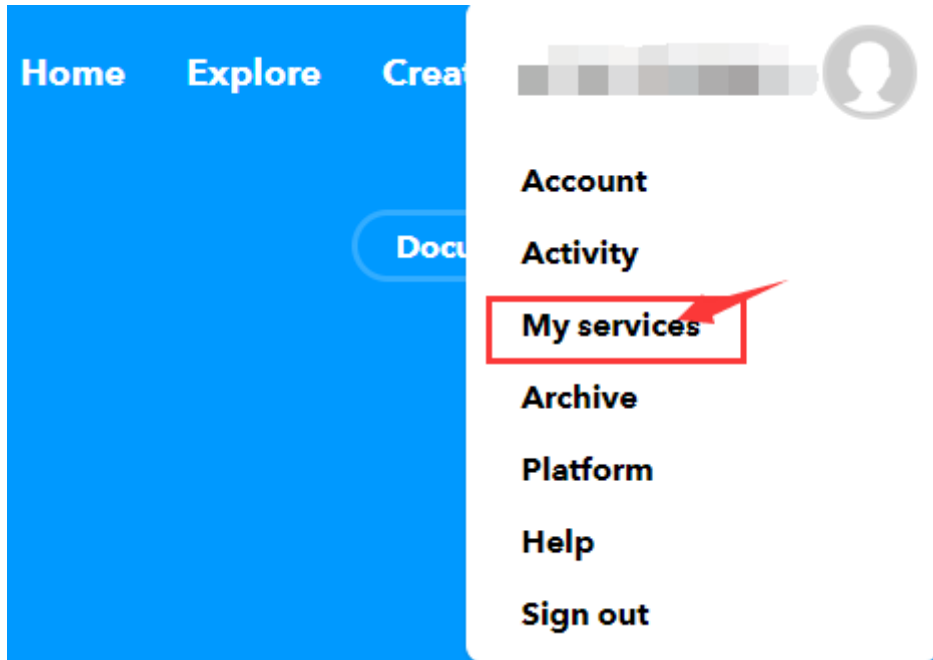
by [g714173948](#)

67/140



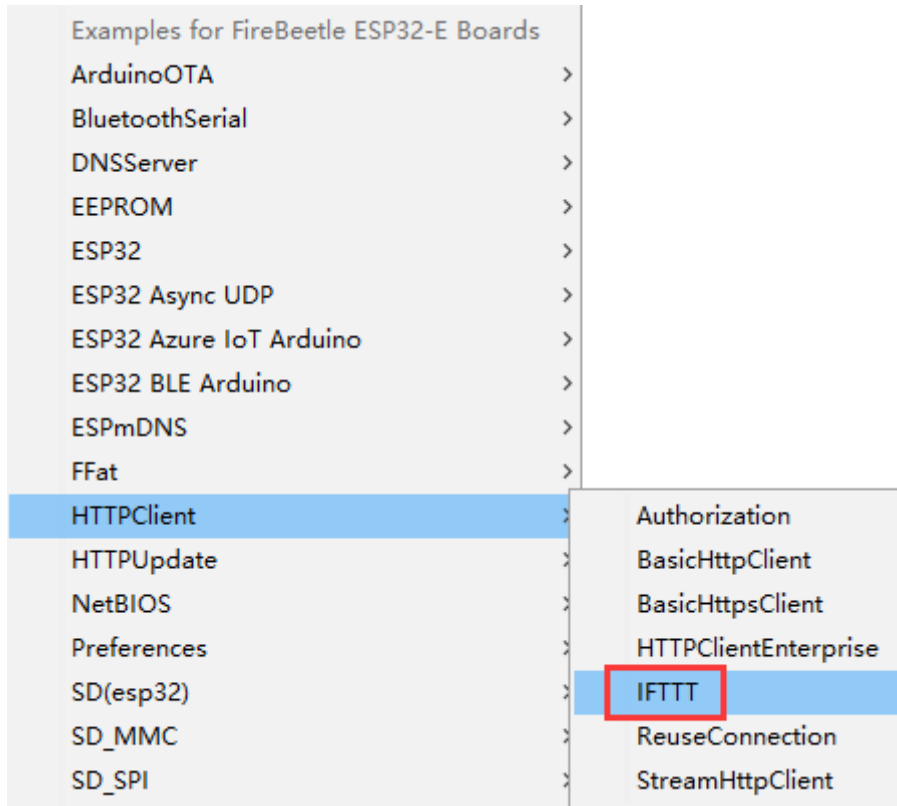
Finish

13. Check IFTTT_Key: click your avatar, click "My services"- "Webhooks"- "Documentation", then copy your key.



request. If you'd like to build your own service and Applets, [check out the IFTTT platform](#).

- Burning Arduino Codes
 - Open the built-in sample code



Sample Code


```

#include <WiFi.h>
#include <HTTPClient.h>
//Configure WiFi name and password
char *WIFI_SSID          = "WIFI_SSID";
char *WIFI_PASSWORD     = "WIFI_PASSWORD";
//Configure IFTTT
char *IFTTT_ENVENT      = "Your_Event";
char *IFTTT_KEY         = "Your_Key";
//IFTTT Send Message
char *IFTTT_VALUE_1     = "Value1";
char *IFTTT_VALUE_2     = "Value2";
char *IFTTT_VALUE_3     = "Value3";
HTTPClient ifttt;
unsigned long lastTime = 0;
unsigned long timerDelay = 10000;
void setup() {
  Serial.begin(115200);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Wifi Connect Success");
}
void loop() {
  //Send an HTTP POST request every 10 seconds
  if ((millis() - lastTime) > timerDelay) {
    //Check WiFi connection status
    if(WiFi.status()== WL_CONNECTED){
      //CALL IFTTT API (IFTTT EVENT, IFTTT KEY)

```

```

    IFTTT.IFTTTBegin(IFTTT_ENVENT,IFTTT_KEY);
    int dataSendState = ifttt.IFTTTSend(IFTTT_VALUE_1,IFTTT_VALUE_2,IFTTT_VALUE_3);
    Serial.println(dataSendState);//Whether the printing data is sent successfully
}else {
    Serial.println("WiFi Disconnected");

}
lastTime = millis();
}
}

```

- Configure Parameters in Arduino Code

```

//Configure WiFi name and password
char *WIFI_SSID          = "WIFI_SSID";//Input WiFi name
char *WIFI_PASSWORD      = "WIFI_PASSWORD";//Input WiFi Password
//Configure IFTTT
char *IFTTT_ENVENT       = "Your_Event";//Input Event Name
char *IFTTT_KEY          = "Your_Key";//Input the key you found in IFTTT
//IFTTT Send Message
char *IFTTT_VALUE_1      = "Value1";
char *IFTTT_VALUE_2      = "Value2";
char *IFTTT_VALUE_3      = "Value3";//Configure the three values in email information

```

Result

Receive the data from FireBeele-ESP32-E in the Email box.

What: message

When: December 8, 2020 at 02:26PM

Extra Data: Value1, Value2, Value3,



[Manage](#)



[Unsubscribe](#) from these notifications or sign in to manage your [Email service](#).

Dimension

- Pin Pitch: 2.54mm
- Mounting Hole Pitch:
- Mounting Hole Size: 2mm
- Board Dimension: 25.400mm×60.00mm
- Thickness: 1.6mm

FAQ


1. Install Driver

FireBeetle-ESP32 adopts CH340 serial chip that can be used without driver among most devices. If you find the driver is not installed automatically after plugging into the device, you can install it manually:click to download the CH340 driver program (<https://dfimg.dfrobot.com/nobody/wiki/0e0d6b3864f7163833ec5d7ad4af7632.EXE>)

For any questions, advice or cool ideas to share, please visit the **DFRobot Forum** (<https://www.dfrobot.com/forum/>).

More Documents

- FireBeetle Schematic (<https://dfimg.dfrobot.com/nobody/wiki/fd28d987619c16281bdc4f40990e5a1c.PDF>)

 Get **FireBeetle_Board_ESP32_E** (<https://www.dfrobot.com/product-2195.html>) from DFRobot Store or **DFRobot Distributor**. (<https://www.dfrobot.com/index.php?route=information/distributorslogo>)

Turn to the Top