FIREBEETLE BOARD-ESP32 用户使用手册 V0.1

DFROBOT
DRIVE THE FUTURE

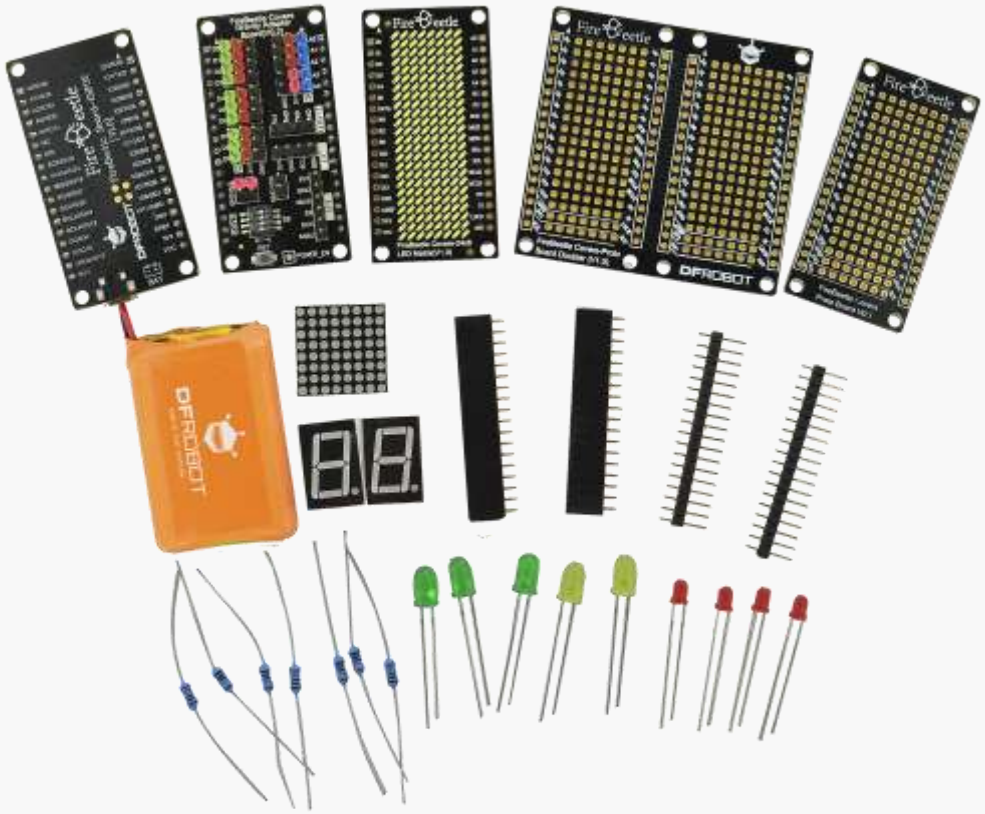
目录

第一章：简介

FireBeetle 介绍

FireBeetle，中文译为萤火虫，它是 DFRobot 新开发的产品线系列，采用超低功耗的外围硬件及小尺寸兼容性接口设计，能够方便、快速的搭建物联网平台。

简单来说，FireBeetle 是一款简单易用的物联网开发平台，是以低功耗为主题的系列开发硬件。旨在为智能硬件爱好者、交互艺术设计师以及电子软件工程师，提供简易的开发体验。例如，一个简单的物联网项目--智能浇花装置，实时的检测土壤湿度，当湿度小于 300（举例参数），装置就自动控制抽水泵给花浇水，并每 5 分钟上传数据到网络，手机端软件（如 Blynk）可以实时的检测数据，当然你也可以在浇花装置上安装一些温湿度传感器，以检测环境的温度。如是，一个智能浇花装置就搭建完成，是不是很简单呢？

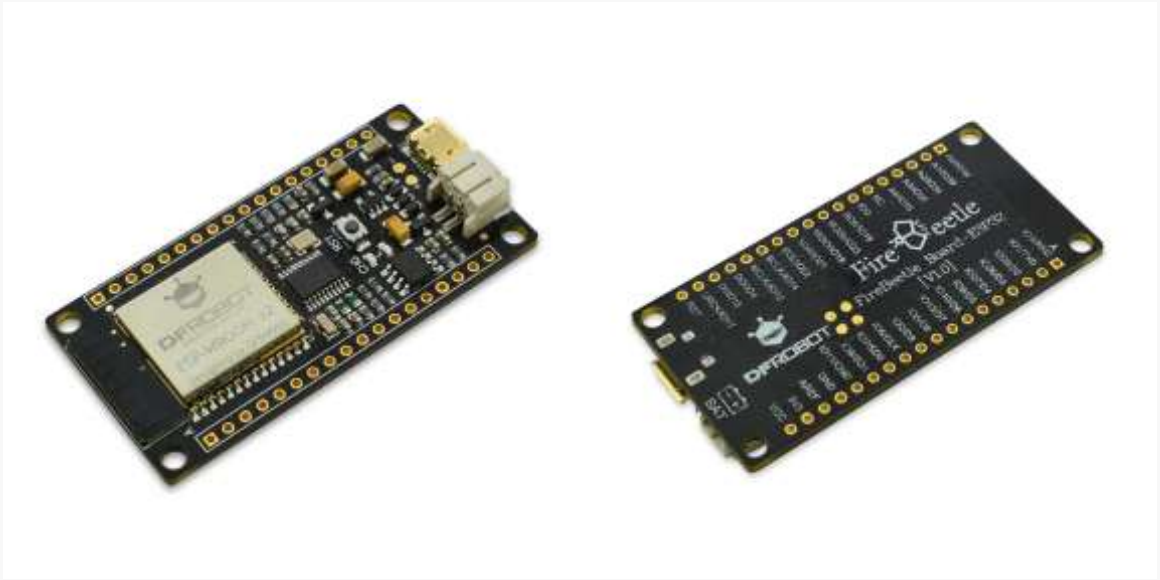


FireBeetle 系列，有三个大类，包括：Board（主控类）、Covers（扩展板类）、Accessories（周边配件）。其中，Board（主控类）主要以 ESP32、Bluno 等高性能、低功耗、具备无线传输功能的 MCU 为核心控制芯片，Covers 包括 LED 点阵、I2S 音频驱动、GPS/GPRS/SIM 等为主板提供丰富外设，Accessories 包括 2.54mm 间距排针、排母、LED 灯、按键等可扩展的外围硬件。

注意：**FireBeetle** 系列相关产品仍然在研发中。

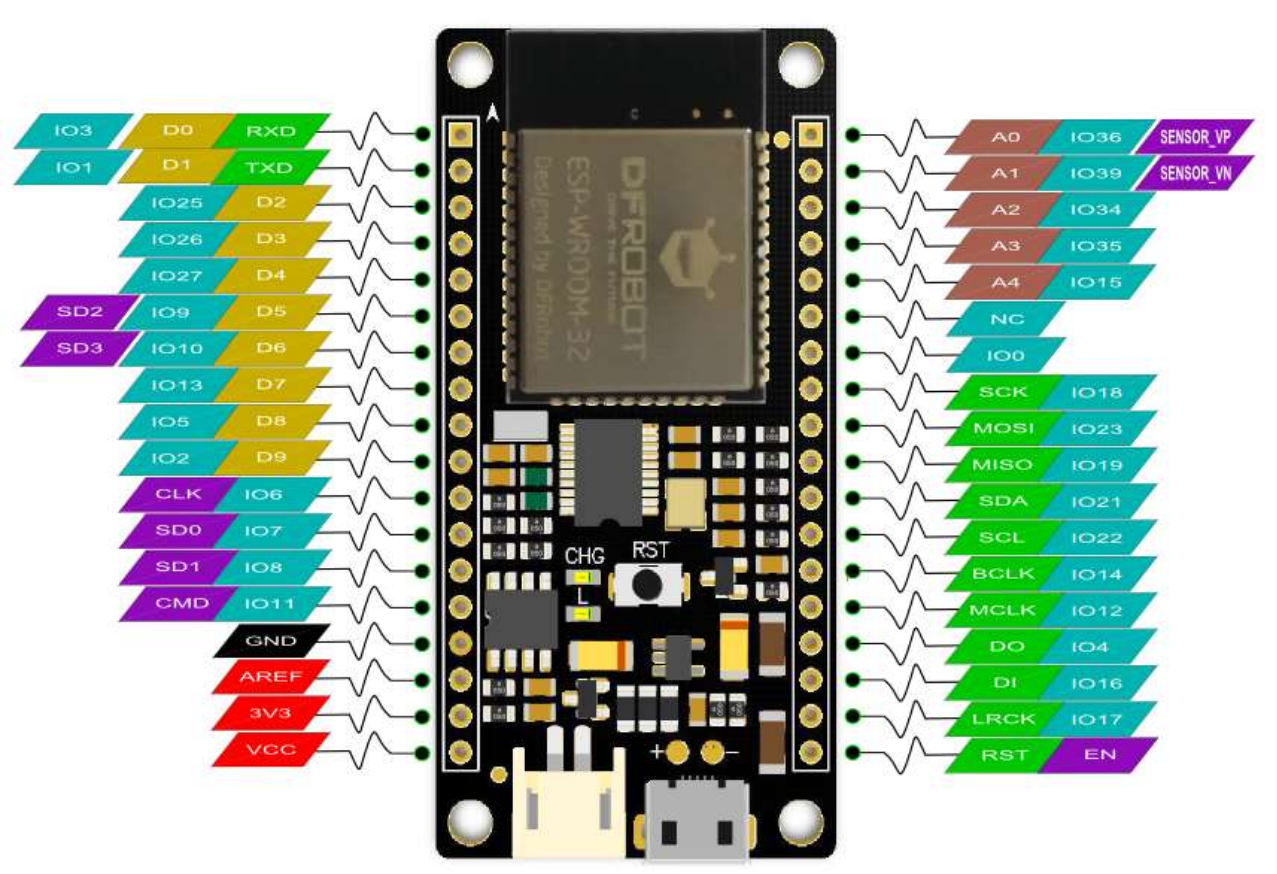
FireBeetle Board-ESP32 简介

FireBeetle Board-ESP32 是 FireBeetle 系列 Board（主控类）之一，该主控板采用 ESP32 芯片，是一款针对物联网方向的 SoC，它集成 WiFi&蓝牙、MCU 于一体。主控采用超低功耗外围硬件设计（实测低功耗模式下功耗为 10uA 左右），支持 USB 及锂电池供电，支持板载锂电池充电功能，是移动设备、可穿戴电子产品、IOT 应用的最佳选择，可以直接应用于低功耗项目。



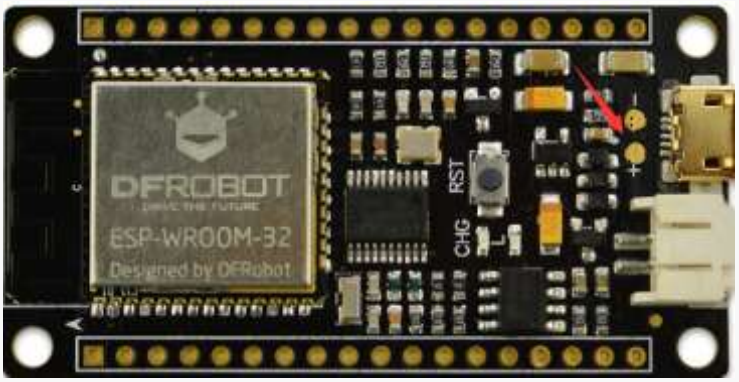
除此以外，FireBeetle Board-ESP32 还具备丰富的外设，ADC，I2C，I2S，SPI，UART 等，且编程方式完全兼容 Arduino IDE 编程。

注意：FireBeetle Board-ESP32 主控所有外设都可以配置到任意引脚，在使用 Arduino IDE 进行编程时，如果没有特殊配置，所有外设采用默认配置。默认配置及对应的硬件 IO 如下图所示：



- IO3/RXD：D0(Arduino)，Serial 的接收引脚，连接到 ESP32 的 IO3
- IO1/TXD：D1(Arduino)，Serial 的发射引脚，连接到 ESP32 的 IO1
- IO25/D2：D2(Arduino)，GPIO 数字输入输出口，PWM 输出引脚，连接到 ESP32 的 IO25
- IO26/D3：D3(Arduino)，GPIO 数字输入输出口，PWM 输出引脚，连接到 ESP32 的 IO26
- IO27/D4：D4(Arduino)，GPIO 数字输入输出口，PWM 输出引脚，连接到 ESP32 的 IO27
- IO9/D5：D5(Arduino)，GPIO 数字输入输出口，PWM 输出引脚，连接到 ESP32 的 IO9
- IO10/D6：D6(Arduino)，GPIO 数字输入输出口，PWM 输出引脚，连接到 ESP32 的 IO10
- IO13/D7：D7(Arduino)，GPIO 数字输入输出口，PWM 输出引脚，连接到 ESP32 的 IO13
- IO5/D8：D8(Arduino)，GPIO 数字输入输出口，PWM 输出引脚，连接到 ESP32 的 IO5
- IO2/D9：D9(Arduino)，GPIO 数字输入输出口，PWM 输出引脚，连接到 ESP32 的 IO2
- CLK：时钟信号线，连接到 ESP32 的 IO6
- SD0：SD0 接口，连接到 ESP32 的 IO7
- SD1：SD1 接口，连接到 ESP32 的 IO8
- CMD：CMD 接口，连接到 ESP32 的 IO11
- GND：电源地
- AREF：参考电压输入，这里连接到 NC
- 3V3：3.3V 电源电压输出，能够提供最大 600mA 电流输出
- VCC：电源电压（输入/输出），5V（USB 供电时），3.7V（锂电池供电时）
- IO36/A0：A0(Arduino)，模拟量输入，连接到 ESP32 的 IO36
- IO39/A1：A1(Arduino)，模拟量输入，连接到 ESP32 的 IO39
- IO34/A2：A2(Arduino)，模拟量输入，连接到 ESP32 的 IO34
- IO35/A3：A3(Arduino)，模拟量输入，连接到 ESP32 的 IO35
- IO15/A4：A4(Arduino)，模拟量输入，连接到 ESP32 的 IO15
- NC：不连接
- IO0：数字口，连接到 ESP32 的 IO0
- SCK：SPI 的时钟线，连接到 ESP32 的 IO18
- MOSI：SPI 的 MOSI 数据线，连接到 ESP32 的 IO23
- MISO：SPI 的 MISO 数据线，连接到 ESP32 的 IO19
- SDA：I2C 数据线，连接到 ESP32 的 IO21
- SCL：I2C 时钟线，连接到 ESP32 的 IO22
- BCLK：I2S 时钟线 BCLK，连接到 ESP32 的 IO14
- MCLK：I2S 时钟线 MCLK，连接到 ESP32 的 IO12
- DO：I2S 的 DO 数据线，连接到 ESP32 的 IO4
- DI：I2S 的 DI 数据线，连接到 ESP32 的 IO16
- LRCK：I2S 的 LRCK 数据线，连接到 ESP32 的 IO17
- RST：RESET 复位接口，低电平复位

对于 FireBeetle 系列的主控板来说，为了适用更复杂的项目，还预留了外部 DC 接口，可以焊接其他外设电源，如无线充电模组、太阳能充电模组等。



注意: “+”连接外部电源的正极，“-”连接外部电源的负极，输入电压范围为 4.2V~6V。

第二章：快速入门

FireBeetle Board-ESP32 硬件

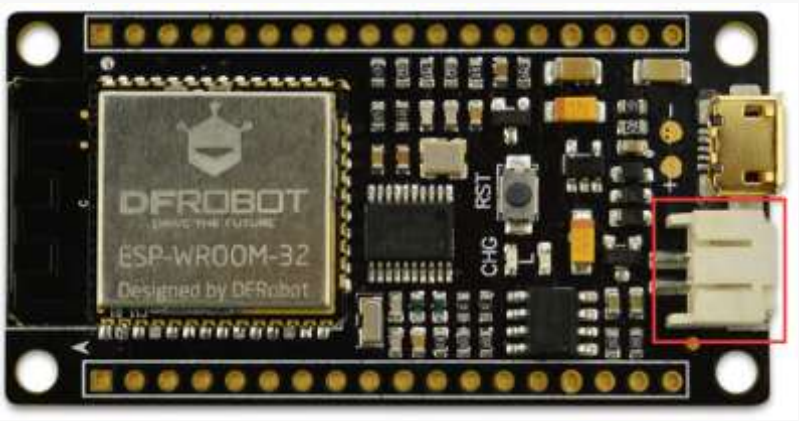
在第一章简介中，已经介绍了各个引脚的对应和默认 Arduino IDE 的外设配置引脚，这里就不再撰述，在使用时可以参考第一章简介内容进行相关引脚的特殊配置。

功能特点

- 工作电压：3.3V
- 输入电压：4.2V~6V
- 支持低功耗：10uA
- 支持最大放电电流：600mA@3.3V LDO
- 支持最大充电电流：500mA
- 支持 USB 充电
- 处理器：Tensilica LX6 双核处理器（一核处理高速连接；一核独立应用开发）
- 主频：240MHz
- SRAM：520KB
- Flash：16Mbit
- Wi-Fi 标准：FCC/CE/TELEC/KCC
- Wi-Fi 协议：802.11 b/g/n/d/e/i/k/r (802.11n，速度高达 150 Mbps)，A-MPDU 和 A-MSDU 聚合，支持 0.4us 防护间隔
- 频率范围：2.4~2.5 GHz
- 蓝牙协议：符合蓝牙 v4.2 BR/EDR 和 BLE 标准
- 蓝牙音频：CVSD 和 SBC 音频低功耗：10uA
- 工作电流：80mA（平均）
- 频率范围：2.4~2.5GHz
- 支持 Arduino 一键下载
- 支持 micropython
- 片上时钟：40MHz 晶振、32.768KHz 晶振
- 数字 I/O：10（arduino 默认）
- 模拟输入：5（arduino 默认）
- SPI：1（arduino 默认）
- I2C：1（arduino 默认）
- I2S：1（arduino 默认）
- LED_BUILTIN：D9
- 接口方式：FireBeetle 系列兼容

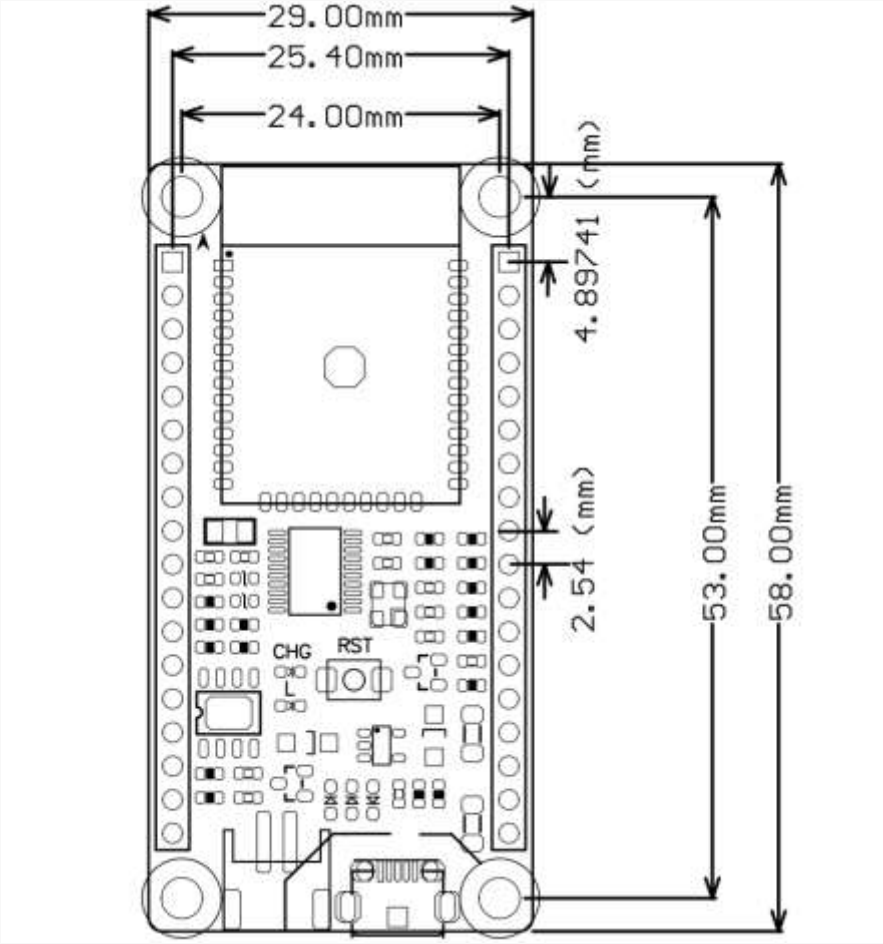
- 工作温度：-40℃~+85℃
- 模块尺寸：24 × 53(mm)/0.94 x 2.09(inches)
- 安装孔尺寸：内径 3.1mm/外径 6mm
- 板载复位按钮
- 硬件版本：V1.0

FireBeetle Board-ESP32 支持 3.7V 锂电池供电，接口如下图所示：



尺寸规格

FireBeetle Board-ESP32 主板在尺寸上完全兼容 FireBeetle 系列，尺寸参数如下图所示：



- pin 脚间距：2.54mm
- 安装孔间距：24mm/53mm
- 安装孔尺寸：3.1mm
- 主板尺寸：29.00 × 58.00mm
- 板厚：1.6mm

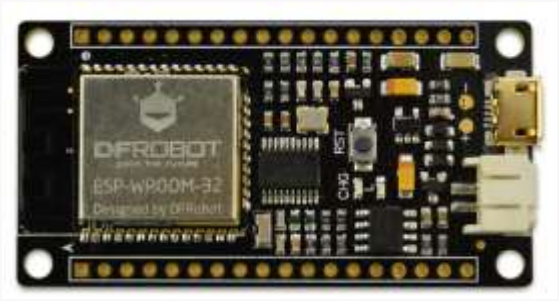
Arduino IDE for FireBeetle Board-ESP32

如果你是第一次接触 Arduino 开发平台，同样可以通过以下教程学习如何下载安装 Arduino IDE 软件，以及如何通过 Arduino IDE 编写 FireBeetle Board-ESP32 程序，完成您自己的项目。

开始之前，请确认您手中有一块 FireBeetle Board-ESP32 主板以及 USB 连接线，除此之外，您还需要一台运行 Windows/Mac OS/Linux 操作系统并且有网络连接的电脑。

我们需要的 FireBeetle Board-ESP32 及 USB 连接线，如下图所示：

USB 连接线



FireBeetle Board-ESP32

Arduino IDE for FireBeetle Board-ESP32 就是让您的 Arduino IDE 支持 FireBeetle Board-ESP32 控制板，通过以下步骤，可以快速搭建开发环境。

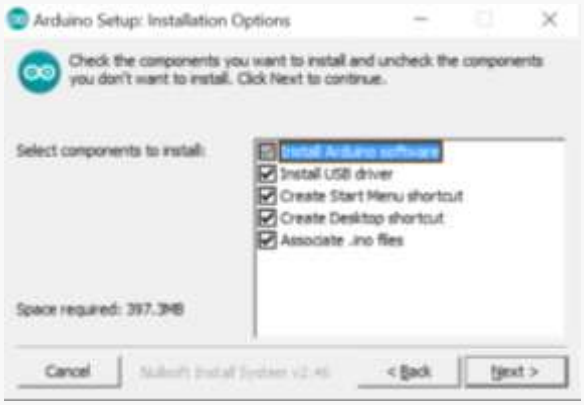
STEP1：下载 Arduino IDE 软件

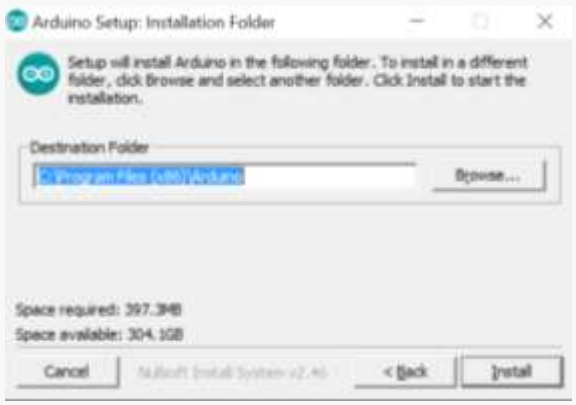
以下的步骤是基于 Windows 操作系统，如果你使用的是其他操作系统，可以将其作为参考。

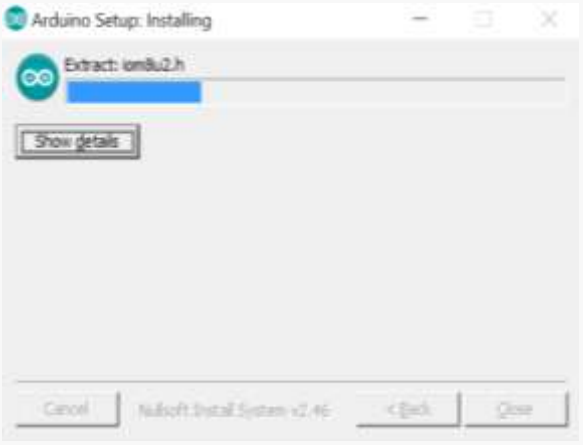
首先，你需要从官网下载最新版本的 Arduino IDE 软件。下载链接：*http://arduino.cc/en/Main/Software*

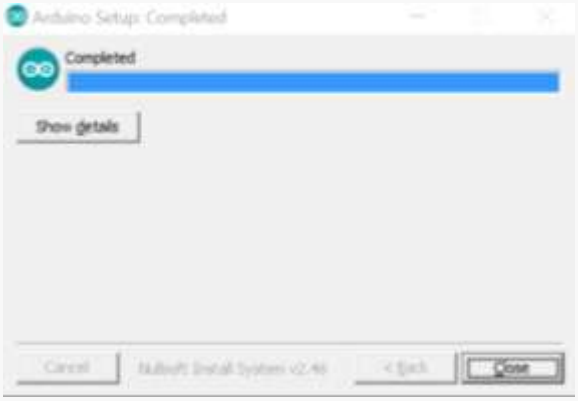
注意：建议使用 Arduino 1.8.0 以上版本，教程使用的是 Arduino IDE 1.8.0 版本。

在下载页右侧的列表中选择下载对应的安装包。对于 Windows 系统用户既可以选择下载 **Windows installer**（推荐初次使用者下载），也可以下载 Windows ZIP 安装包（需要手动安装驱动）。若选择的是 Windows installer，你可以直接执行安装程序，并跟随安装向导完成配置，驱动会在程序安装完成后自动安装。





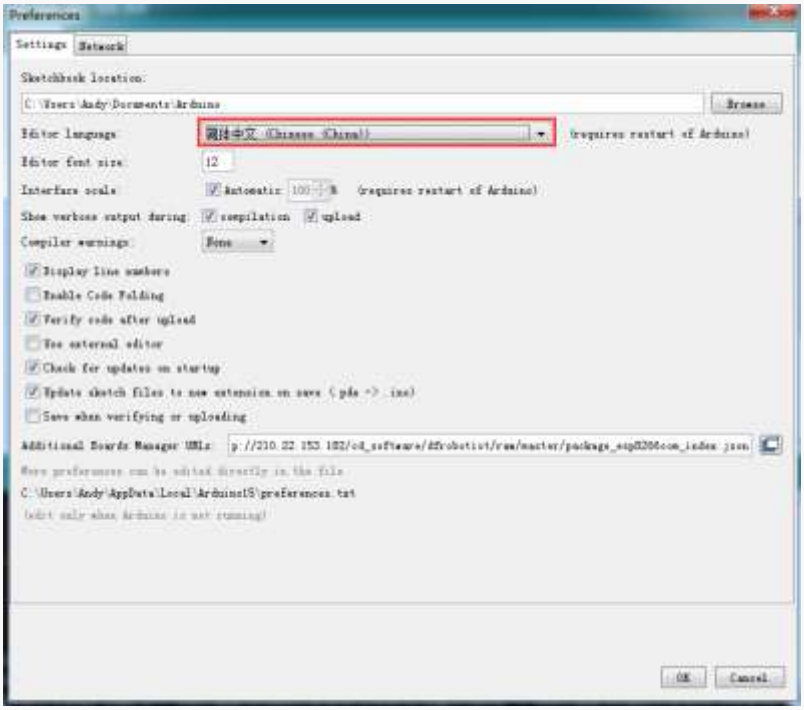




STEP2: Arduino IDE 汉化

Arduino IDE 本身支持多种语言（包括中文），我们只需要设置为中文即可。

打开 **File->Preferences->Editor language**，选择**简体中文（Chiness（China））**，然后重启 IDE。



STEP3: 安装 FireBeetle Board-ESP32 开发板核心

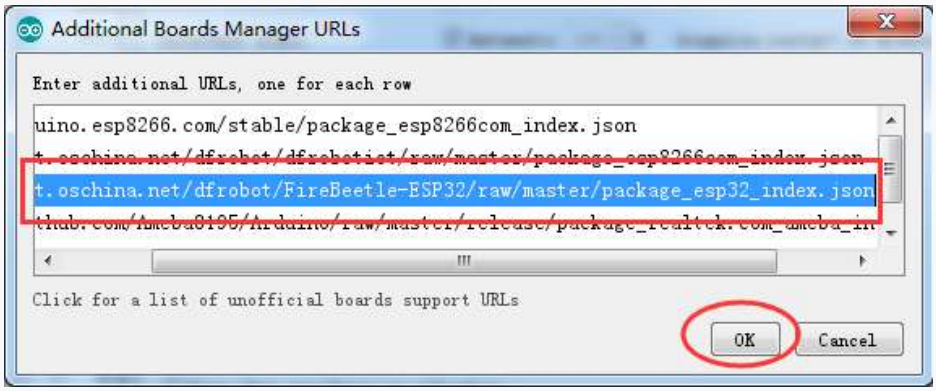
Arduino IDE 安装包中不包含 FireBeetle Board-ESP32 开发板核心，需要手动添加。首先，要添加 FireBeetle Board-ESP32 支持，需要在 Arduino 开发板管理器里手动安装 FireBeetle Board-ESP32 开发板核心。

（a）打开**文件 -> 首选项**，在**附加开发板管理网址**中，将以下网址复制进去：

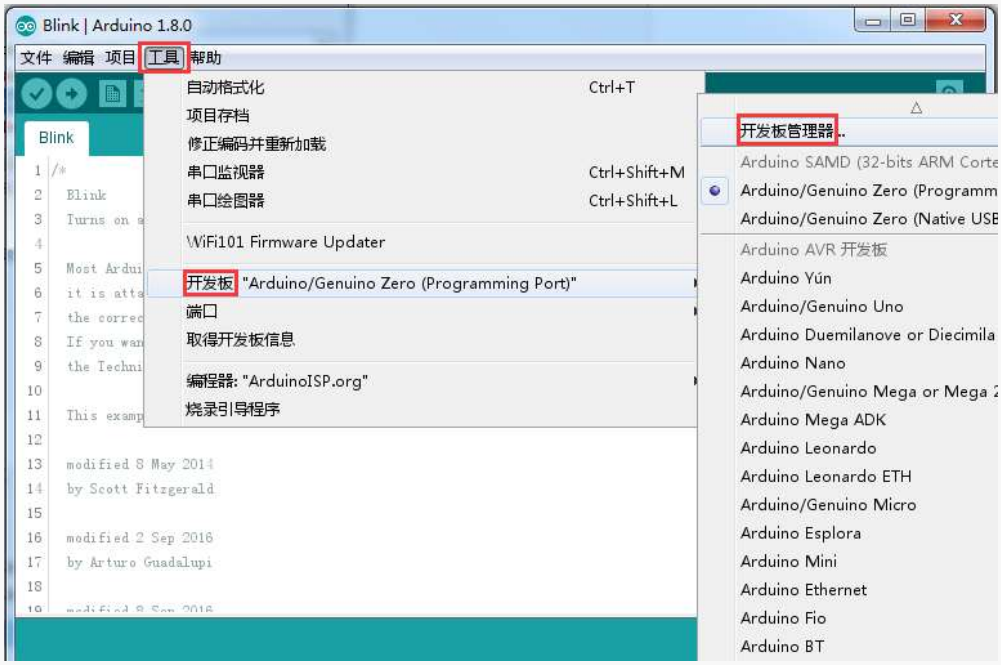
https://git.oschina.net/dfrobot/FireBeetle-ESP32/raw/master/package_esp32_index.json

点击**好**，完成设置。

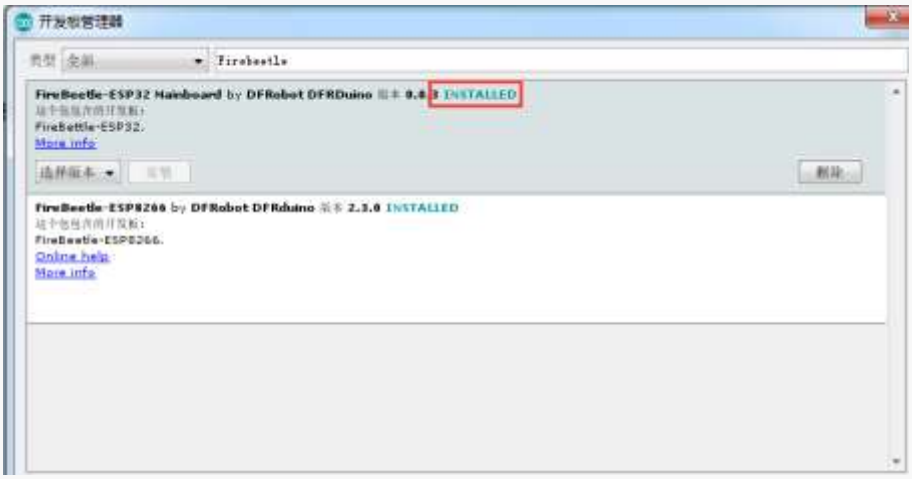
注意:如果这里已经输入了其他开发的 URL 地址，需要点击按钮，在地址添加框中添加即可，如下图所示：



（b）打开工具 -> 板子 -> 开发板管理器



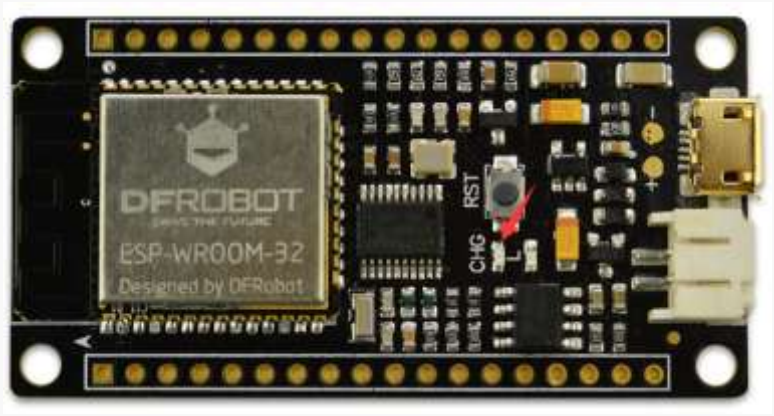
在打开的开发板管理器中，输入"FireBeetle"并等待信息加载完毕。版本号为 0.0.3（或者最新版本并点击"安装"后耐心等待安装完成。整个过程会因网络状况持续 5-10 分钟。安装完成后，开发板信息会被标注"INSTALLED"），如下图所示：



注意：如果发现安装过程中一直处于卡顿现象，可能是网络原因引起的，您可以尝试强制重启 Arduino IDE，重新执行之前的步骤，或者通过翻墙软件加速网络，直到安装完成。其次，在安装过程中，有些关键进程可能会被防火墙或者杀毒软件拦截，请选择允许更改并添加至白名单。

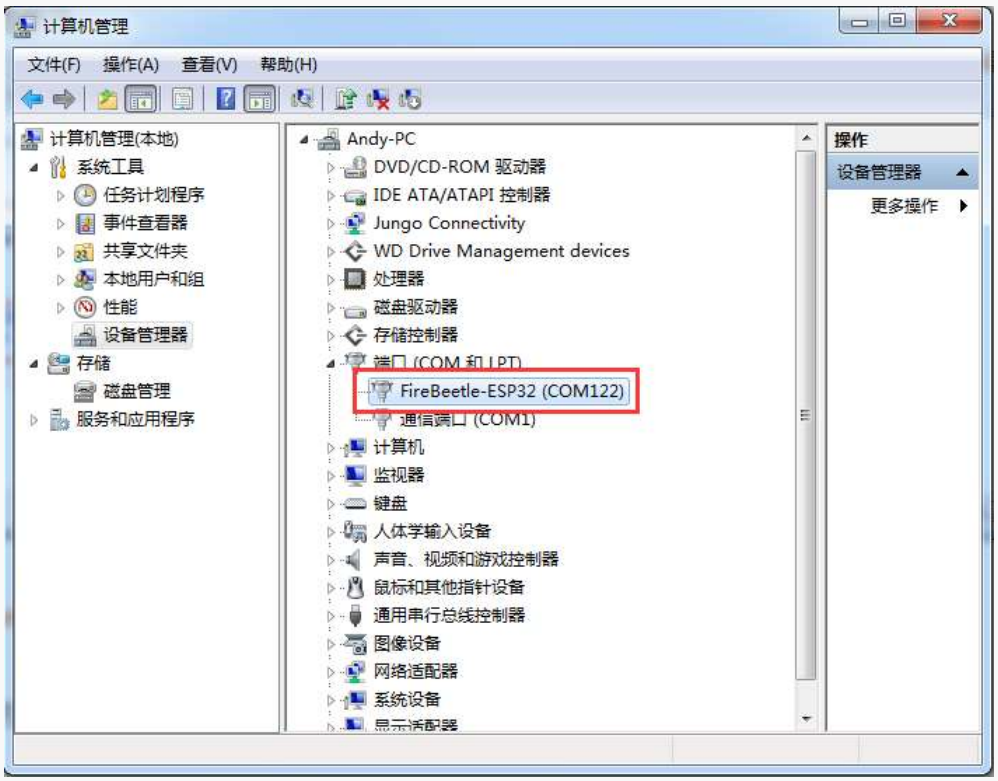
STEP4: 连接 FireBeetle Board-ESP32 至电脑

正确安装完成 Arduino IDE 和 FireBeetle Board-ESP32 开发板核心后，即可将 FireBeetle Board-ESP32 通过 USB 数据线连接至电脑。正确连接时 FireBeetle Board-ESP32 的 CHG 电源指示灯会闪烁（这是在查询有没有接入锂电池）。



在编程之前，我们需要确保开发板被电脑识别并且找出连接了哪一个 COM 口（用于提供串口通信交互）。可以在接下来的步骤中确认。

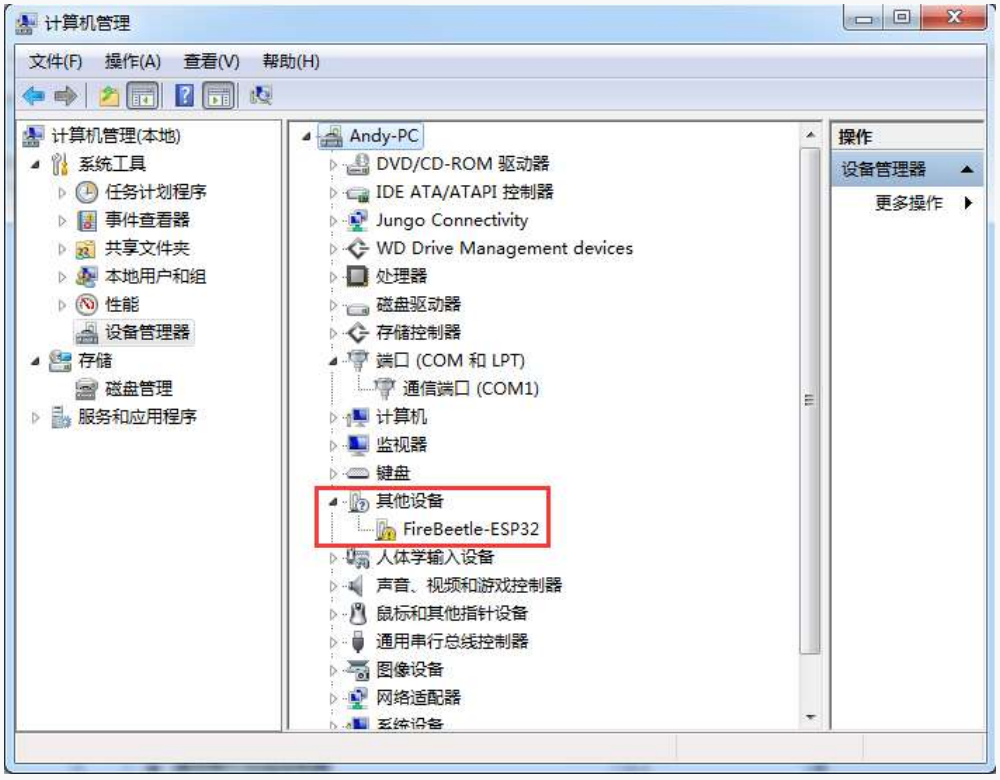
首先打开“控制面板”，打开“设备管理器”，点开“端口（COM 和 LPT）”。接上 FireBeetle Board-ESP32 的端口就会在列表中显示（这里是 COM122）。



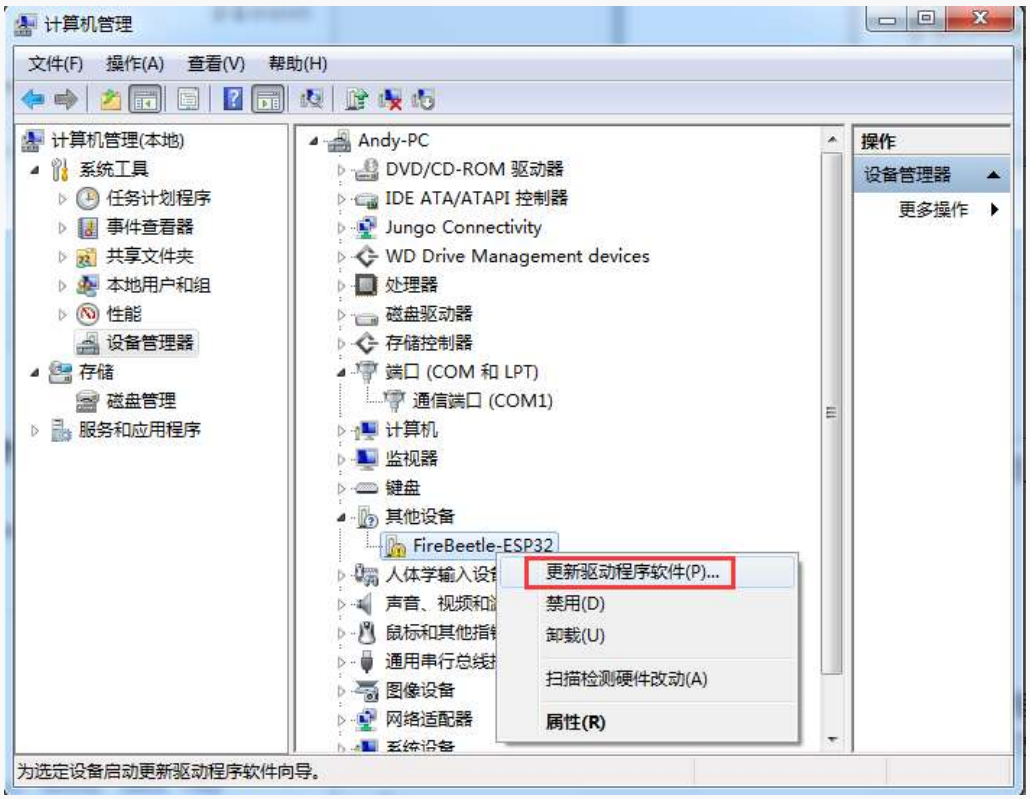
如果提示无法识别的设备，您需要下载 FireBeetle Board-ESP32 驱动到本机，并安装驱动，下载地址：

https://github.com/Chocho2017/FireBeetle-Board-ESP32.git

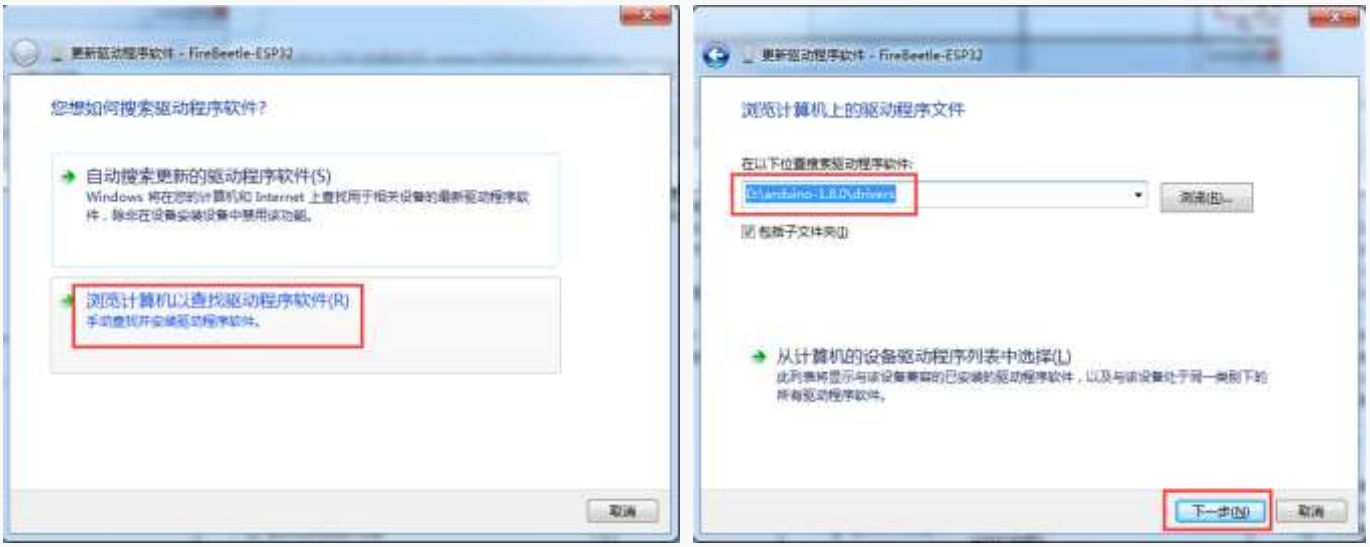
错误提示信息如下图所示：



将下载后的 FireBeetle-ESP32.inf 驱动文件保存到您的电脑（任意位置都可以），右键点击 **FireBeetle-ESP32**，选择更新驱动程序软件，如下图所示：



选择浏览计算机以查找驱动程序软件，在地址栏中输入您刚刚保存的 FireBeetle-ESP32.inf 文件目录，点击下一步，如下图所示：



注意：这里 FireBeetle-ESP32.inf 文件保存在 Arduino IDE 的 drivers 文件夹下。

然后根据提示完成驱动文件的安装。

STEP5：在 Arduino IDE 中进行编程

Arduino IDE 软件安装完成后，运行软件打开编程窗口。你可以在这个窗口里编辑并上传代码到 Arduino 开发板上，或是使用内置的串口监视器通过串口与开发板通信。现在让我们仔细看下 Arduino IDE 界面。

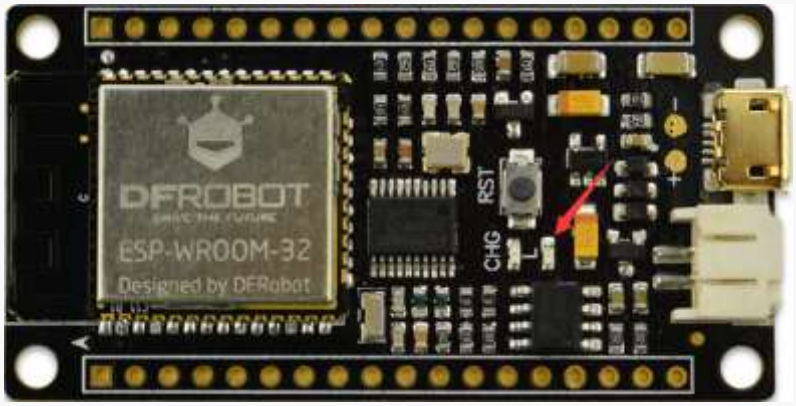
与常规 C 语言程序不同的是，一段用于 Arduino 的程序通常由 void setup() 部分与 void loop() 两部分构成。

"void setup()"用于放置初始化程序的代码，这部分代码在开发板上电后仅运行一次。需要重复运行的代码需要放置在"void loop()"中，的这些代码会一直重复运行，使得开发板时时与外部进行交互。

白色区域是编程区域。编程区域以下的黑色部分是信息窗口，用于显示代码上传以及编译的信息。你可以通过文件->首选项，设置相关打印信息。

STEP6: 上传代码至 FireBeetle Board-ESP32 主板

在这一步，我们将演示如何上传示例程序"Blink"到 FireBeetle Board-ESP32 主板。"Blink"程序的功能是控制 D9 引脚上的 LED 灯间隔一秒闪烁一次。FireBeetle Board-ESP32 主板与大部分 Arduino 相同，有一个板载的 D9 LED 信号灯，这意味着在本例中我们不需要连接其他的外设元件。LED 状态指示灯可以在 FireBeetle Board-ESP32 主板的找到，如下图所示：



在上传之前，你应该首先确认代码中没有错误。点击"编译"确认。

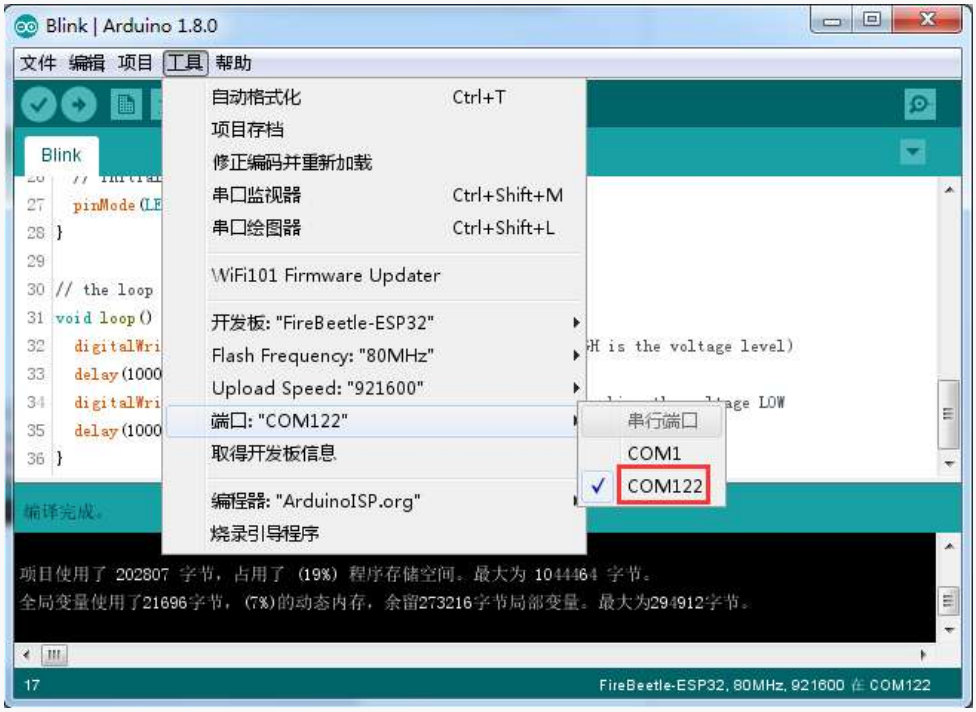
注意: Blink 示例代码在*文件->示例->Basics->Blink*。

等待几秒钟，如果没有错误的话，一条"编译完成"信息会在信息窗口显示,表示 "编译成功"。如果出现错误可以返回检查程序是否完整。

编程成功后，选择工具->开发板->FireBeetle-ESP32，通常在不切换到使用其他型号的开发板时，这个步骤仅需在第一次使用时执行一次即可。

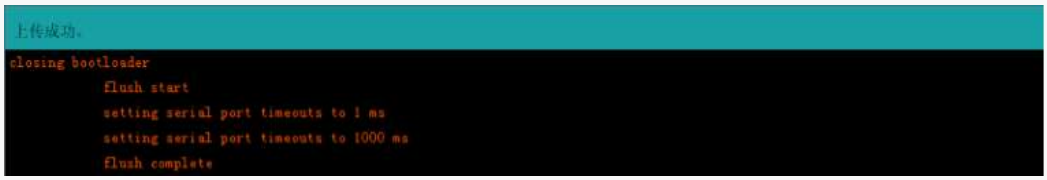
切换到工具栏>工具>端口，根据 STEP4 中显示的 FireBeetle Board-ESP32 所占用串口序号，我们应该选择"COM122"作为通信端口。通信端口只有当开发板连接至电脑并被成功识别时才会出现。同一块开发板在插拔后可能会占用不同的端口上，所以我们每次上传前都要重复确认。

COM 选择完成后，开发板的信息和端口就会在窗口右下角显示。



最后，点击“上传”烧写代码到 FireBeetle Board-ESP32 中。

成功上传后，“上传成功”消息会出现在信息窗口。此时 FireBeetle Board-ESP32 板载的 D9 LED 灯会开始闪烁。



简而言之，为 Arduino 上传代码可以分为以下三个步骤：

- 编译代码；
- 选择开发板型号和端口；
- 上传！

以上就是使用 FireBeetle Board-ESP32 主板在 Arduino IDE 上面的一些基本的方法。如果在使用中存在任何疑问或者建议，欢迎访问我们的论坛联系我们。

注意：FireBeetle Board-ESP32 仍然针对开发人员，并不是所有的外设都具有示例代码的完整功能，还有一些 bug 需要被发现和修复。在 Arduino IDE 下，IO 管脚功能以及 I2C/SPI 可以直接调用，但其他依旧在开发中。

论坛链接: *http://www.dfrobot.com.cn/community/forum.php*

记得来逛 DFRobot 的社区看看更多的教程和精彩的项目哦。我们同样希望您能够把你自己的项目或者想法发在论坛上分享。欢迎成为我们的一员！

DF 官方链接: *http://www.dfrobot.com.cn*

第三章: FireBeetle Board-ESP32 外设使用（基础篇）

本章主要是通过一些示例项目，阐述 FireBeetle Board-ESP32 主控板的外设基本使用方法，通过下面的项目，您可以进行修改完成您的自己的项目。

其中 FireBeetle Board-ESP32 外设主要包括：UART、I2C、SPI、ADC、PWM、DAC，以及内部集成的霍尔传感器等。

项目一 串口实验

在最开始的章节中，我们上传了一个 Blink 闪烁程序来测试板子上的 LED 状态灯。现在，我们使用 UART 串口，每秒打印一次计时数据。

所需元件

1 x FireBeetle Board-ESP32 主板

硬件连接

该项目不需要连接其他传感器，只需要使用 USB 线连接 FireBeetle Board-ESP32 和电脑即可。

输入代码

打开 Arduino IDE。尽管您可以打开 ***Course->Item-1*** 代码，我们还是建议您自己手动输入代码熟悉下。

代码如下:

```
void setup() {
```

```
        Serial.begin(115200);
}
void loop() {
        static unsigned long i = 0;
        Serial.println(i++);
        delay(1000);
}
```

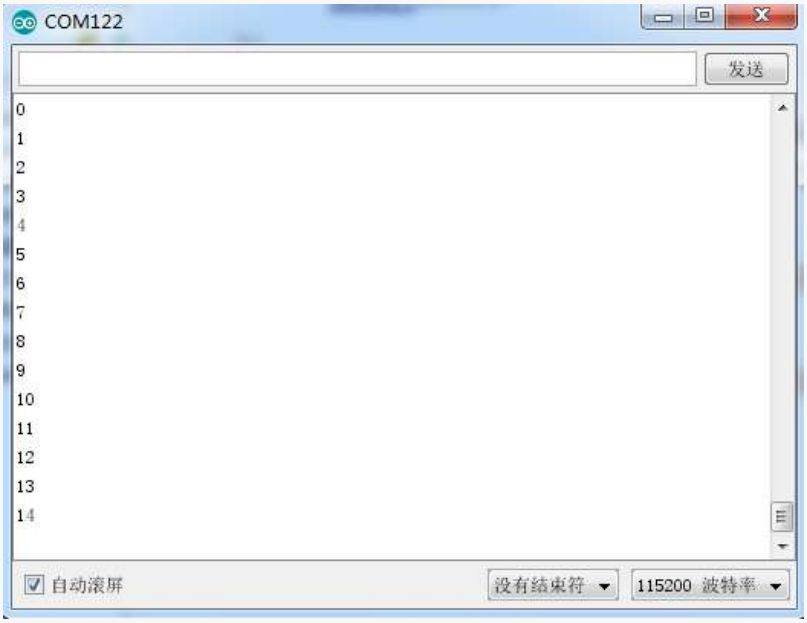
输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。

注意：如果你忘了怎么编译和上传代码，请参考前面的入门教程。

样例程序在《Course》文件夹下的 *Item-1* 程序

实验现象

完成之前步骤的上传后，打开 Arduino IDE 自带的串口监视器，可以看到如下的打印信息：



项目二 PWM(呼吸灯)

呼吸灯，即让 FireBeetle Board-ESP32 通过 PWM 驱动 LED 灯，实现 LED 的亮度渐变，看起来就像是在呼吸一样。关于 PWM 的解释，请阅览知识扩展部分。

所需元件

1 x FireBeetle Board-ESP32 主板

硬件连接

该项目不需要连接其他传感器，LED 灯可以使用 FireBeetle Board-ESP32 主板上面的 L 灯，硬件连接方面只需要使用 USB 线连接 FireBeetle Board-ESP32 和电脑即可。

输入代码

打开 Arduino IDE。您可以打开 ***Course->Item-2*** 代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

```
//设置通道 0
#define LEDC_CHANNEL_0     0

//设置 13 位定时器
#define LEDC_TIMER_13_BIT  13

//设置定时器频率位 5000Hz
#define LEDC_BASE_FREQ     5000
//设置 LED 灯
#define LED_PIN            D9
```

```
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

//设置 led 灯的亮度
void ledcAnalogWrite(uint32_t value, uint32_t valueMax = 255) {
  //计算占空比
  uint32_t duty = (LEDC_BASE_FREQ / valueMax) * min(value, valueMax);
  //设置占空比
  ledcWrite(LEDC_CHANNEL_0, duty);
}

void setup() {
  ledcSetup(LEDC_CHANNEL_0, LEDC_BASE_FREQ, LEDC_TIMER_13_BIT);
  ledcAttachPin(LED_PIN, LEDC_CHANNEL_0);
}

void loop() {
  ledcAnalogWrite(brightness);
  brightness += fadeAmount;

  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  delay(30);
}
```

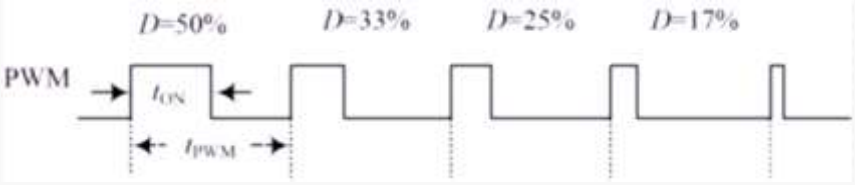
输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。上传完成后您就可以看到 L 灯开始“呼吸”了！

现在让我们来回顾一下代码和硬件，看看它是如何工作。

知识学习

什么是 PWM 控制信号？

PWM（pulse-width modulation）脉冲宽度调制，MCU（微控制器）通过对开关器件的通断进行控制，使输出端得到一系列幅值相等的脉冲，用这些脉冲来代替正弦波或所需的波形。如下图所示：



其中，t_{ON} 是高电平持续时间，t_{PWM} 是 PWM 波的周期，t_{PWM}-t_{ON} 是低电平持续时间，占空比是指高电平持续时间占整个周期的比例，即 $D=t_{on}/t_{PWM}$ 。

代码分析

FireBeetle Board-ESP32 的 PWM 比普通的 Arduino UNO 高级的多，设置上不能简单的使用 analogWrite 函数来驱动 PWM，而是需要设置 timer 函数，以及相关的频率参数等才能工作。

```
#define LEDC_CHANNEL_0      0
```

定义了定时器使用的通道，FireBeetle Board-ESP32 总共有 16 个通道，这里用的是通道 0。

```
#define LEDC_TIMER_13_BIT  13
```

定义了定时器为 13 位定时器，即定时器最大计数为 2 的 13 次方。

```
#define LEDC_BASE_FREQ     5000
```

这是设置定时器的频率，单位是 Hz。接下来的 brightness 和 fadeAmount 参数分别表示 PWM 的占空比和每次变化的数值。

```
void ledcAnalogWrite(uint32_t value, uint32_t valueMax = 255)
```

这个函数是计算 PWM 占空比和设置 PWM 占空比，类似 Arduino 的 analogWrite 函数，可以看到，传递参数的最大值是 255，这是为了和 analogWrite 兼容。

```
ledcSetup(LEDC_CHANNEL_0, LEDC_BASE_FREQ, LEDC_TIMER_13_BIT);
ledcAttachPin(LED_PIN, LEDC_CHANNEL_0);
```

这两个函数是 FireBeetle Board-ESP32 定时器设置函数，函数原型及原理这里不讲述，如果您感兴趣可以看看底层源码（源码地址：C:\Users\"your-PC"\AppData\Local\Arduino15\packages\esp32\hardware\DFRobot_FireBeetle-ESP32\0.0.3\libraries\ESP32\），这里只需要知道怎么用这些函数来设置相关的 timer 就可以了。

关于什么是 PWM 信号，在前面已经阐述过了，这里不再说明。

注意：FireBeetle Board-ESP32 的任何引脚都可以配置成 PWM 输出，您可以尝试着修改代码，完成您的项目。

项目三 ADC

ADC（模数转换器即 A/D 转换器），是指将模拟信号转变成数字信号。FireBeetle Board-ESP32 的 ADC 是 12 位的，最大输出值为 4095，而 Arduino UNO 是 10 位的，最大输出值是 1023，因此，在精度上比普通的 UNO 要高，而且转换速率快，且在使用上兼容 Arduino analogRead 函数，直接读取即可。

所需元件

1 × 模拟角度传感器

注意：为了方便您的使用，传感器采用 Gravity 接口，即插即用，不用您去焊接电线，您可以点击链接进行购买。

1 × FireBeetle Covers-Gravity Adapter Board

注意：如果您是初学者，建议使用 Graviry 适配器板，传感器即插即用。

1 x FireBeetle Board-ESP32 主板

硬件连接

把 FireBeetle Covers-Gravity Adapter Board 插接到 FireBeetle Board-ESP32 主板上，然后将模拟角度传感器插接到 A0（实验中用的是 A0 模拟口）。

元件连接好后，使用 USB 线连接 FireBeetle Board-ESP32 和电脑。

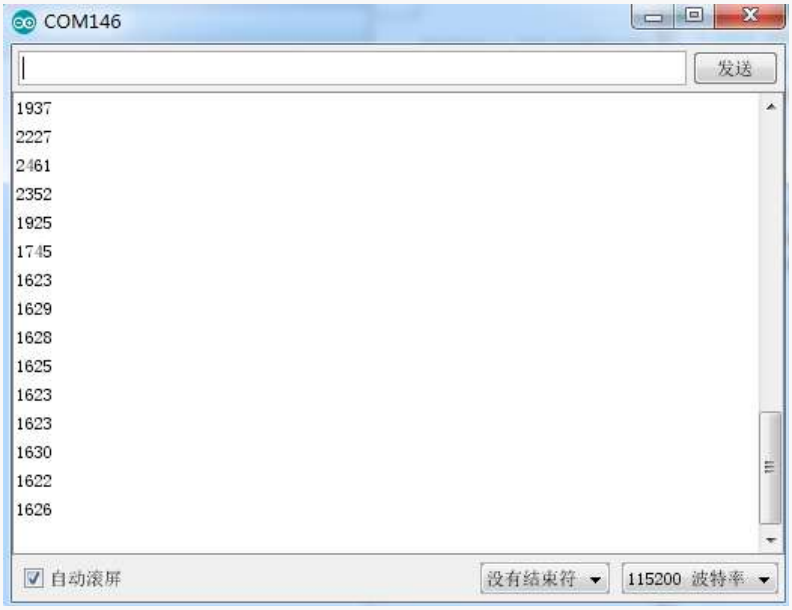
输入代码

打开 Arduino IDE。您可以打开 ***Course-> Item-3*** 代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

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

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println(analogRead(A0));
  delay(100);
}
```

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。上传完成后，打开 Arduino IDE 的串口监视器，旋转模拟角度传感器，可以看到串口监视器中的数值变化，如下图所示：



注意：如果您忘了怎么编译和上传代码，请参考前面的入门教程。

代码分析

由于 FireBeetle Board-ESP32 的 ADC 在使用上完全兼容 Arduino，因此这里不再对 analogRead 函数进行过多的讲解。

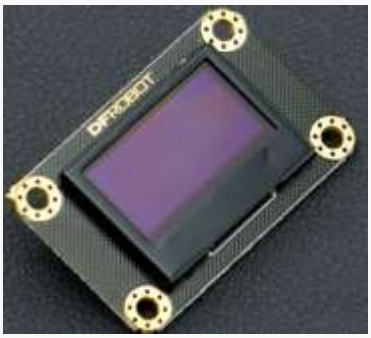
注意：如果您对 Arduino 的基本函数不是特别熟悉，您可以点击链接进行学习。

项目四 I2C

FireBeetle Board-ESP32 的 I2C 可以配置到任意 I/O 口，您可以通过传递相关参数进行配置。为了方便使用，我们已经将 I2C 进行了默认配置，在使用上完全兼容 Arduino，默认配置引脚可以在第一章简介中查看到。本项目是基于 I2C 默认配置，驱动 OLED 显示屏。

所需元件

1 x Gravity I2C OLED-2864 显示屏



注意：为了方便您的使用，传感器采用 Gravity 接口，即插即用，不用您去焊接电线，您可以点击链接进行购买。

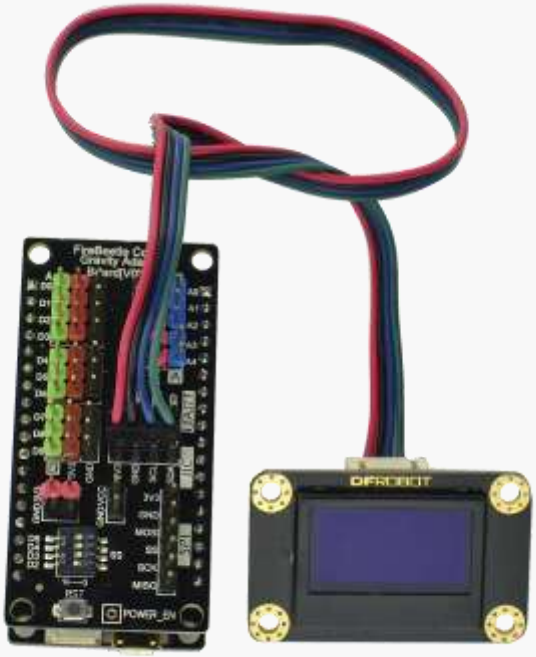
1 x FireBeetle Covers-Gravity Adapter Board

注意：如果您是初学者，建议使用 Graviry 适配器板，传感器即插即用。

1 x FireBeetle Board-ESP32 主板

硬件连接

把 FireBeetle Covers-Gravity Adapter Board 插接到 FireBeetle Board-ESP32 主板上，然后将 OLED 显示屏插接到 I2C 接口。如下图所示



元件连接好后，使用 USB 线连接 FireBeetle Board-ESP32 和电脑。

输入代码

首先，我们需要知道模拟角度传感器输出数据的范围，以方便后面做数据映射。打开 Arduino IDE。您可以打开 ***Course->Item-4*** 打开代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

```
#include <Wire.h>

int UG2864Address = 0x3C;//OLED UG2864 器件 7 位地址

unsigned char  show2[] ={
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//*page0
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x08,0xF8,0x08,0x08,0x08,0x10,0xE0,0x00,0x08,0xF8,0x88,0x88,0xE8,0x08,0x00,0x00, //DF 上半
  0x08,0xF8,0x88,0x88,0x88,0x88,0x70,0x00,0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,//Ro
  0x08,0xF8,0x00,0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,//bo
  0x00,0x80,0x80,0xE0,0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//t
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//*page1
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x20,0x3F,0x20,0x20,0x20,0x10,0x0F,0x00,0x20,0x3F,0x20,0x00,0x03,0x00,0x00,0x00,//DF 下半
  0x20,0x3F,0x20,0x00,0x03,0x0C,0x30,0x20,0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,//Ro
  0x00,0x3F,0x11,0x20,0x20,0x11,0x0E,0x00,0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,//bo
  0x00,0x00,0x00,0x1F,0x20,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//t
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//*page2
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x80,0x80,0x00,0x80,0x00,0x80,0x80,0x80,0x80,0x80,0x00,0x80,0x00,0x80,0x80,0x80,//*page3
  0x80,0x80,0x00,0x80,0x00,0x80,0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,//
  0x80,0x88,0xF8,0x00,0x00,0x80,0x80,0xF0,0x88,0x88,0x88,0x18,0x80,0x80,0x80,0x00,//
  0x80,0x80,0x80,0x00,0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,0x08,0xF8,0x00,0x80,//ro 上半
  0x80,0x00,0x00,0x00,0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,0x00,0x80,0x80,0xE0,//bo
  0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x80,0x80,0x00,0x00,//t.c
  0x00,0x00,0x80,0x80,0x80,0x80,0x00,0x00,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x00,//om
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//.
  0x0F,0x30,0x0C,0x03,0x0C,0x30,0x0F,0x00,0x0F,0x30,0x0C,0x03,0x0C,0x30,0x0F,0x00,//*page4
  0x0F,0x30,0x0C,0x03,0x0C,0x30,0x0F,0x00,0x00,0xc0,0xc0,0x00,0x00,0x0E,0x11,0x20,//
  0x20,0x10,0x3F,0x20,0x00,0x20,0x20,0x3F,0x20,0x20,0x00,0x00,0x20,0x20,0x3F,0x21,//
  0x20,0x00,0x01,0x00,0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,0x00,0x3F,0x11,0x20,//ro 下半
  0x20,0x11,0x0E,0x00,0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,0x00,0x00,0x00,0x1F,//bo
  0x20,0x20,0x00,0x00,0x00,0xc0,0xc0,0x00,0x00,0x0E,0x11,0x20,0x20,0x20,0x11,0x00,//t.c
  0x00,0x1F,0x20,0x20,0x20,0x20,0x1F,0x00,0x20,0x3F,0x20,0x00,0x3F,0x20,0x00,0x3F,//om
  0x00,0x00,0xc0,0xc0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//.
```

```
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//*page5
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//*page6
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x80,0x80,0x80,0x00,0x00,0x80,0x80,0x00,0x80,0x80,0x80,0x00,0x00,//cn
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//*page7
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x0E,0x11,0x20,0x20,0x20,0x11,0x00,0x20,0x3F,0x21,0x00,0x00,0x20,0x3F,0x20,//cn
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,//
  };

void Writec(unsigned char COM){
   Wire.beginTransmission(UG2864Address);
   Wire.write(0x00);
   Wire.write(COM);
   Wire.endTransmission();
}

void Writed(unsigned char DATA){
   Wire.beginTransmission(UG2864Address);
   Wire.write(0x40);
   Wire.write(DATA);
   Wire.endTransmission();
}

void SSD1306(){
   Writec(0XAE);//display off
   Writec(0X00);//set lower column address
   Writec(0X10);//set higher column address
   Writec(0X40);//set display start line
   Writec(0XB0);//set page address
   Writec(0X81);//set contract control
```

```
    Writec(0XCF);// VCC Generated by Internal DC/DC Circuit
    Writec(0XA1);//set segment remap  column address 127 is mapped to SEG0
    Writec(0XA6);//normal / reverse   normal display
    Writec(0XA8);//multiplex ratio
    Writec(0X3F);//1/64
    Writec(0XC8);//Com scan direction remapped mode. Scan from COM[N-1] to COM0
    Writec(0XD3);//set display offset
    Writec(0X00);
    Writec(0XD5);//set osc division
    Writec(0X80);
    Writec(0XD9);//set pre-charge period
    Writec(0X11);
    Writec(0XDa);//set COM pins
    Writec(0X12);
    Writec(0X8d);/*set charge pump enable*/
    Writec(0X14);
    Writec(0Xdb);//Set VcomH
    Writec(0X20);
    Writec(0XAF);//display ON
}

void fill(unsigned char dat){
    unsigned char i,j;
    Writec(0x00);
    Writec(0x10);
    Writec(0xB0);
    for(j=0;j<8;j++){
      Writec(0xB0+j);
      Writec(0x00);
      Writec(0x10);
      for(i=0;i<128;i++){
          Writed(dat);
      }
    }
}

void show(){
    unsigned char x,y;
    unsigned int j=0;

    Writec(0x00);//set lower column address
    Writec(0x10);//set higher column address

    for(y=0;y<8;y++){
        Writec(0xB0+y);
        Writec(0x00);
        Writec(0x10);
```

```
            for(x=0;x<128;x++)
            Writed(show2[j++]);
        }
    }

    void setup(){
       Wire.begin();
       SSD1306();
       fill(0xff);//点亮屏幕
       delay(100);
       fill(0x00);//清屏
       delay(100);
    }

    void loop(){
      show();
    }
```

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。上传完成后，OLED 显示屏会显示“DFRobot www.dfrobot.com.cn”字样。

代码分析

本项目的代码相对于前面的项目较多，主要是基于 I2C 通信对 OLED 显示屏底层寄存器的直接驱动，而对于 OLED 显示屏更高级的应用您可以点击链接，阅览 DF 官方 WiKi。

void Writec(unsigned char COM)

设置寄存器函数，通过 I2C 对 OLED 显示屏设置，I2C 使用方法完全兼容 Arduino。

void Writed(unsigned char DATA)

写数据函数，I2C 使用方法完全兼容 Arduino。

注意：FireBeetle Board-ESP32 的 I2C 与 Arduino 完全兼容，主要是调用 Wire 库文件使用。

项目五 SPI

在很多传感器中，都使用 SPI 通信，因为 SPI 通信速率相对于 I2C 更快，没有地址冲突的弊端。SPI，是一种高速的、全双工、同步的通信总线，而 FireBeetle Board-ESP32 的 SPI 可以配置到所有 I/O，您可以阅览底层代码进行使用（初学者不建议使用）。为了更好的使用体验，FireBeetle Board-ESP32 默认情况下配置了 IO18、IO19、IO23 为 SPI 口，在使用上完全兼容 Arduino。

本项目使用 FireBeetle Board-ESP32，通过 SPI 读取 BME280 温湿度传感器的数据，示例中使用的是 BME280 库文件，关于 SPI 驱动您可以阅览 BEM280 库文件，点击链接下载 BME280 库文件。

所需元件

1 × BME280 温湿度传感器

注意：BME280 传感器本身支持 I2C 和 SPI 通信，这里我们采用 SPI 通信。

1 × FireBeetle Covers-Gravity Adapter Board

注意：如果您是初学者，建议使用 Graviry 适配器板，传感器即插即用。

1 x FireBeetle Board-ESP32 主板

输入代码

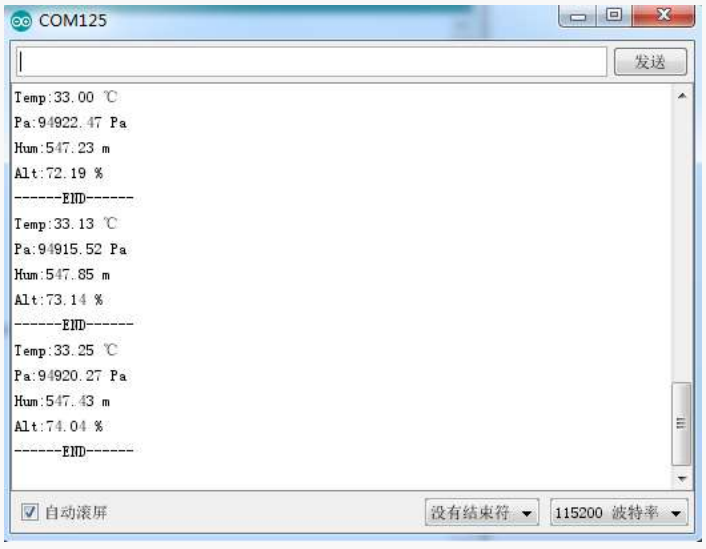
打开 Arduino IDE。你可以打开 ***Course-> Item-5*** 打开代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

```
#include <DFRobot_BME280.h>

#define SEA_LEVEL_PRESSURE 1013.25f
#define BME_CS D2

DFRobot_BME280 bme(BME_CS); //SPI

float temp, pa, hum, alt;

void setup() {
    Serial.begin(115200);

    // I2c default address is 0x77, if the need to change please modify bme.begin(Addr)
    if (!bme.begin()) {
        Serial.println("No sensor device found, check line or address!");
        while (1);
    }

    Serial.println("-- BME280 DEMO --");
}

void loop() {
     temp = bme.temperatureValue();
     pa = bme.pressureValue();
     hum = bme.altitudeValue(SEA_LEVEL_PRESSURE);
```

```
        alt = bme.humidityValue();

        Serial.print("Temp:");
        Serial.print(temp);
        Serial.println(" ℃");

        Serial.print("Pa:");
        Serial.print(pa);
        Serial.println(" Pa");

        Serial.print("Hum:");
        Serial.print(hum);
        Serial.println(" m");

        Serial.print("Alt:");
        Serial.print(alt);
        Serial.println(" %");

        Serial.println("------END------");

        delay(1000);
    }
```

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。打开 Arduino 串口监视器，可以看到打印信息如下：

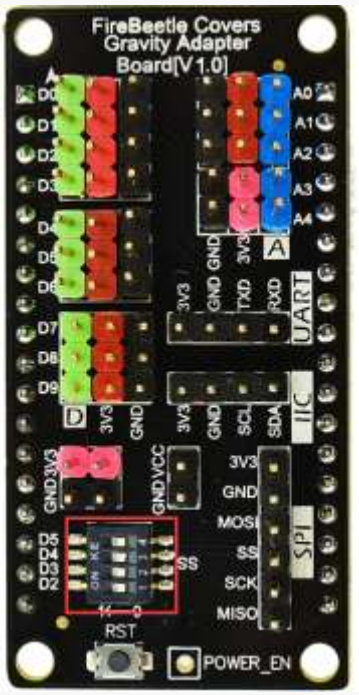


代码分析

本项目采用的是 BME280 库文件，在 Item-5.ino 文件中并没有对 SPI 底层进行操作，不过，FireBeetle Board-ESP32 的 SPI 使用完全兼容 Arduino。

```
#define BME_CS D2
```

设置 SPI 片选引脚，这里和 Arduino 有不同，需要传递 D2，而不是 2。其次，需要将 Graviry 适配板的 SS 拨码开关的 D2 选通，并将 BME280 通过杜邦线连接到 Gravity 适配板的 SPI 接口，如下图所示：



项目六 霍尔传感器

FireBeetle Board-ESP32 集成的霍尔传感器是基于空穴（N-carrier）电阻设计。当芯片至于电磁场中时，霍尔传感器会在电阻上横向产生一个小电压，这个小电压可由 ADC 直接采集测量，也可以经过超低噪声前置模拟放大器放大后再由 ADC 测量。

本项目是基于 FireBeetle Board-ESP32 的霍尔传感器进行的实验，当磁铁靠近 FireBeetle Board-ESP32 主板的正面（有 logo 一面）时，打印出来的数值为负数，并随着磁铁的靠近，值越小；当磁铁靠近 FireBeetle Board-ESP32 的反面时，打印出来的数值为证书，并随着磁铁的靠近，值越大。

所需元件

1 x FireBeetle Board-ESP32 主板

注意：该项目不需要连接其他传感器。

输入代码

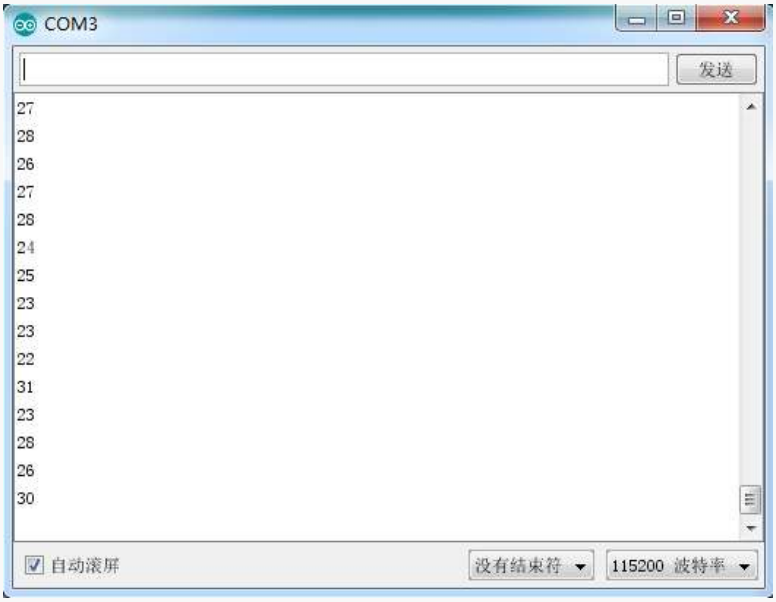
打开 Arduino IDE。你可以打开 ***Course-> Item-6*** 打开代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

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

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println(hallRead());
  delay(100);
```

```
}
```

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。打开 Arduino 串口监视器，可以看到打印信息如下：



注意：该您可以移动磁铁与 FireBeetle Board-ESP32 距离，观察打印数值的变化。

代码分析

本项目使用 FireBeetle Board-ESP32 集成的霍尔传感器，底层采用 IO36 和 IO39 读取霍尔传感器上面的电压（如果您对底层感兴趣，可以阅览源码），这里我们只需要调用 hallRead 函数即可。

```
hallRead()
```

获取霍尔传感器上面的电压转换值，返回类型是 int 型。您可以尝试着修改代码，将霍尔传感器应用到您的项目中。

项目七 DAC

DAC（数/模转换器），与项目三的 ADC 恰恰相反，DAC 是将数字信号转换成模拟信号输出。FireBeetle Board-ESP32 集成了两个 8-bit DAC 通道，这两个数字信号分别转换为两个模拟电压信号输出。DAC 电路由内置电阻串和 1 个缓冲器组成。并且这两个 DAC 可以作为参考电压使用，也可以作为其他电路的电源使用，是两个独立的 DAC。

本项目将阐述如何使用 DAC 输出阶梯波形。

所需元件

1 x FireBeetle Board-ESP32 主板

注意：该项目不需要连接其他传感器。

输入代码

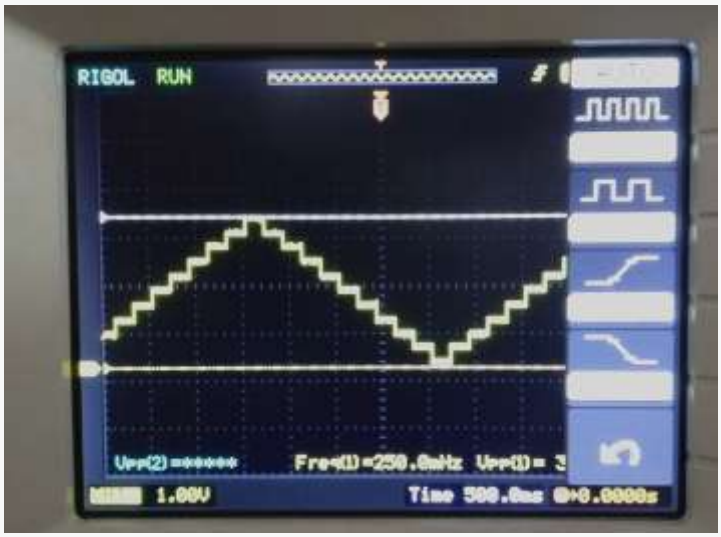
打开 Arduino IDE。你可以打开 ***Course-> Item-7*** 打开代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

```
void setup() {
}

void loop() {
  // put your main code here, to run repeatedly:
  for(int i=0;i<10;i++){
    dacWrite(D2,i*25);
    delay(200);
  }
```

```
    for(int j=10;j>0;j--){
      dacWrite(D2,j*25);
      delay(200);
    }
  }
```

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。用示波器测试 D2 口的电压，可以看到如下图所示的阶梯波形：



代码分析

可以看到 Arduino 代码中，使用 DAC 是很方便的，只需要调用 dacWrite 函数，函数原型如下：

```
void dacWrite(uint8_t pin, uint8_t value)
```

其中，pin 脚为输出 DAC 的数字口，这里只能传递 D2 或者 D3；value 是输出的值，范围是 0~255，输出后对应的电压值是 0~Vcc。

项目八 触摸传感器

FireBeetle Board-ESP32 提供了多达 10 个电容式传感器 GPIO，能够探测由手指或其他物品直接接触或接近而产生的电容差异。这种低噪声特性和电路的高灵敏度设计适用于较小的触摸板，可以直接用于触摸开关。

本项目阐述了如何通过 Arduino 代码获取 FireBeetle Board-ESP32 的触摸传感器状态，并打印状态。

所需元件

1 x FireBeetle Board-ESP32 主板

注意：该项目不需要连接其他传感器。

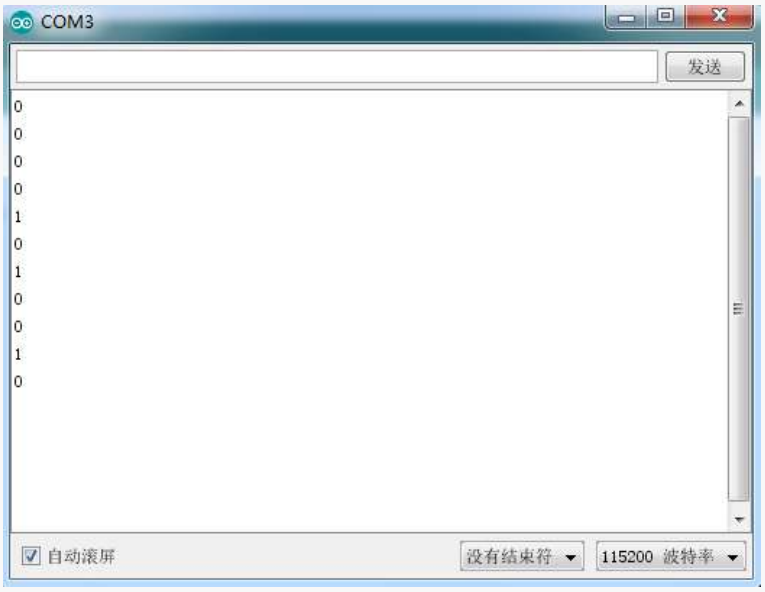
输入代码

打开 Arduino IDE。您可以打开 ***Course-> Item-8*** 打开代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

```
void setup()
{
  Serial.begin(115200);
  delay(1000); // give me time to bring up serial monitor
  Serial.println("FireBeetle Board-ESP32 Touch Test");
}

void loop(){
  Serial.println(touchRead(T2));  // get value using T0->D9
  delay(1000);
}
```

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。打开 Arduino IDE 串口监视器，并用手触摸 D9（T2 对应的是数字口 D9），可以看到会打印出“1”，如下图所示：



代码分析

获取触摸传感器的 GPIO 状态，只需要调用 touchRead 函数，函数原型如下：

```
uint16_t touchRead(uint8_t pin)
```

返回“0”表示没有触摸，“1”表示触摸。其中 pin 是 T0~T9，对应到 FireBeetle 的引脚如下表所示：

触摸传感器序号	对应的 ESP32 硬件	FireBeetle Board-ESP32
T0	GPIO4	DO/IO4
T1	GPIO0	IO0
T2	GPIO2	IO2/D9
T3	MTDO	A4/IO15
T4	MTCK	IO13/D7
T5	MTD1	MCLK/IO12
T6	MTMS	BCLK/EO14
T7	GPIO27	IO27/D4
T8	32K_XN	外部晶振
T9	32K_XP	外部晶振

注意：T8 和 T9 已经连接到外部晶振 32.768KHz，请不要使用 T8 和 T9 作为触摸输入。

项目九 Deep_Sleep（低功耗）

FireBeetle Board-ESP32 拥有先进的电源管理技术，可以切换到不同的省电模式，以满足不同的应用环境对低功耗的要求。在进入低功耗后，可以通过外部中断唤醒 FireBeetle Board-ESP32 主板，也可以通过 RTC 定时器，让 FireBeetle Board-ESP32 在设置的时间后重启唤醒。

本项目阐述了如何通过 Arduino 代码让 FireBeetle Board-ESP32 进入低功耗 Deep_sleep 模式。

所需元件

1 x FireBeetle Board-ESP32 主板

注意：该项目不需要连接其他传感器。

FireBeetle Board-ESP32 省电模式：

- Active 模式：芯片视频处于工作状态，芯片可以接受、发射和监听信号
- Modem-sleep 模式：CPU 可运行，时钟可被配置，WiFi/蓝牙视频关闭
- Light-sleep 模式：CPU 暂停运行。RTC 和 ULP 协处理器运行
- Deep-sleep 模式：仅 RTC 处于工作状态，WiFi 和蓝牙连接数据存储在 RTC 中，ULP 协处理器工作
- Hibernate 模式：内置的 8MHz 晶振和 ULP 协处理器被禁止，RTC 内存回收电源被切断

不同省电模式下的功耗：

省电模式	描述	功耗
Active（射频工作）	Wi-Fi Tx packet 13 dBm ~ 21 dBm	160 ~ 260 mA
	Wi-Fi / BT Tx packet 0 dBm	120 mA
	Wi-Fi / BT Rx 和侦听	80 ~ 90 mA
	关联睡眠方式（与 Light-sleep 模式关联）	0.9 mA@DTIM3, 1.2 mA@DTIM1
Modem-sleep	CPU 处于工作状态	最大速度：20 mA
		正常速度：5 ~ 10 mA
		慢速：3 mA
Light-sleep	-	0.8 mA
Deep-sleep	ULP 协处理器处于工作状态	0.5 mA
	超低功耗传感器监测方式	25 μA @1% duty
	RTC 定时器 + RTC 存储器	10 μA
Hibernate	仅有 RTC 定时器处于工作状态	2.5 μA

注意：由于 Arduino 支持度还在完善中，示例中进入低功耗模式只能支持 Deep_sleep 模式，FireBeetle Board-ESP32 功耗在 40uA 左右，如果您熟悉 linux 编程，可以阅览 ESP-idf 源码进行更深层次的开发。

输入代码(一)

打开 Arduino IDE。您可以打开 ***Course-> Item-9-1*** 打开代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(D9,OUTPUT);
  digitalWrite(D9,HIGH);
  delay(1000);
  digitalWrite(D9,LOW);
  ESP.deepSleep(5000000);
}

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

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。

代码分析

示例中使用了 ESP.deepSleep 函数，该函数是 Arduino 底层库封装好了的，不用额外的配置 sleep 模式，函数原型如下：

```
void deepSleep(uint32_t time_us)
```

传递的参数是 uint32_t 类型的数据，是一个设置时间值，单位是 uS（微秒），示例程序中，让 FireBeetle Board-ESP32 sleep 5 秒后重启，可以看到 FireBeetle Board-ESP32 主板上面的 L 灯每 5s 闪烁一次。

输入代码(二)

打开 Arduino IDE。您可以打开 ***Course-> Item-9-2*** 打开代码，但我们还是建议您自己手动输入代码熟悉下。

代码如下:

```
void setup() {
```

```
  // put your setup code here, to run once:
  pinMode(D9,OUTPUT);
  digitalWrite(D9,HIGH);
  delay(1000);
  digitalWrite(D9,LOW);
  esp_deep_sleep_enable_ext0_wakeup(GPIO_NUM_25,LOW);
  esp_deep_sleep_start();
}

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

输入完成后，点击“编译”检查代码有无错误。确保没有错误后就可以开始上传了，点击“上传”之后 IDE 会把代码发送给 FireBeetle Board-ESP32 主板。

代码分析

本示例和 Item-9-1 实现的方法不同，是将 FireBeetle Board-ESP32 一直进入睡眠模式，通过外部中断唤醒主板，并将 GPIO25 设置为外部中断唤醒口，并设置为低电平中断：

```
esp_deep_sleep_enable_ext0_wakeup(GPIO_NUM_25,LOW);
```

这里的 esp_deep_sleep_enable_ext0_wakeup 函数是底层设置函数，您不必在意函数本身的实现原理以及底层的相关配置，只需要了解怎么用就可以了。

可以配置的中断口，以及对应的 FireBeetle Board-ESP32 主板的引脚，如下表所示：

中断参数名称	对应的 ESP32 硬件	FireBeetle Board-ESP32
GPIO_NUM_0	GPIO0	IO0
GPIO_NUM_1	GPIO1	IO1/TXD
GPIO_NUM_2	GPIO2	IO2/D9
GPIO_NUM_3	GPIO3	IO3/RXD
GPIO_NUM_4	GPIO4	DO/IO4
GPIO_NUM_5	GPIO5	IO5/D8
GPIO_NUM_6	GPIO6	IO6/CLK
GPIO_NUM_7	GPIO7	IO7/SD0
GPIO_NUM_8	GPIO8	IO8/SD1
GPIO_NUM_9	GPIO9	IO9/D5
GPIO_NUM_10	GPIO10	IO10/D6
GPIO_NUM_11	GPIO11	IO11/CMD
GPIO_NUM_12	GPIO12	MCLK/IO12
GPIO_NUM_13	GPIO13	IO13/D7
GPIO_NUM_14	GPIO14	BCLK/IO14
GPIO_NUM_15	GPIO15	A4/IO15
GPIO_NUM_16	GPIO16	DI/IO16
GPIO_NUM_17	GPIO17	LRCK/IO17
GPIO_NUM_18	GPIO18	SCK/IO18

GPIO_NUM_19	GPIO19	MISO/IO19
GPIO_NUM_21	GPIO21	SDA/IO21
GPIO_NUM_22	GPIO22	SCL/IO22
GPIO_NUM_23	GPIO23	MOSI/IO23
GPIO_NUM_25	GPIO25	IO25/D2
GPIO_NUM_26	GPIO26	IO26/D3
GPIO_NUM_27	GPIO27	IO27/D4
GPIO_NUM_34	GPIO34	A2/IO34
GPIO_NUM_35	GPIO35	A3/IO35
GPIO_NUM_36	GPIO36	A0/IO36
GPIO_NUM_39	GPIO39	A1/IO39

注意：在使用外部中断唤醒 **FireBeetle Board-ESP32** 时，配置中断引脚只能使用 **GPIO_NUM_X** 来传递引脚号参数，这是由于代码中调用的是底层函数，不能在 **Arduino ino** 文件中直接传递 **Dx**。