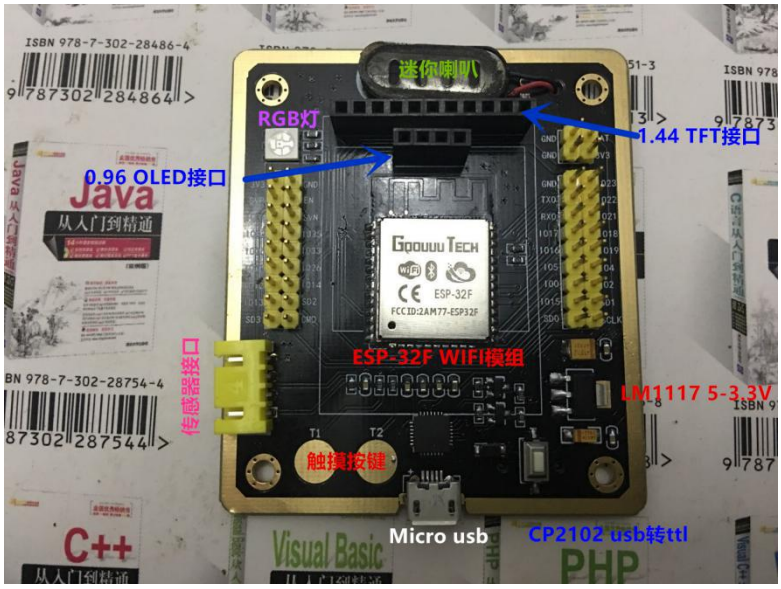
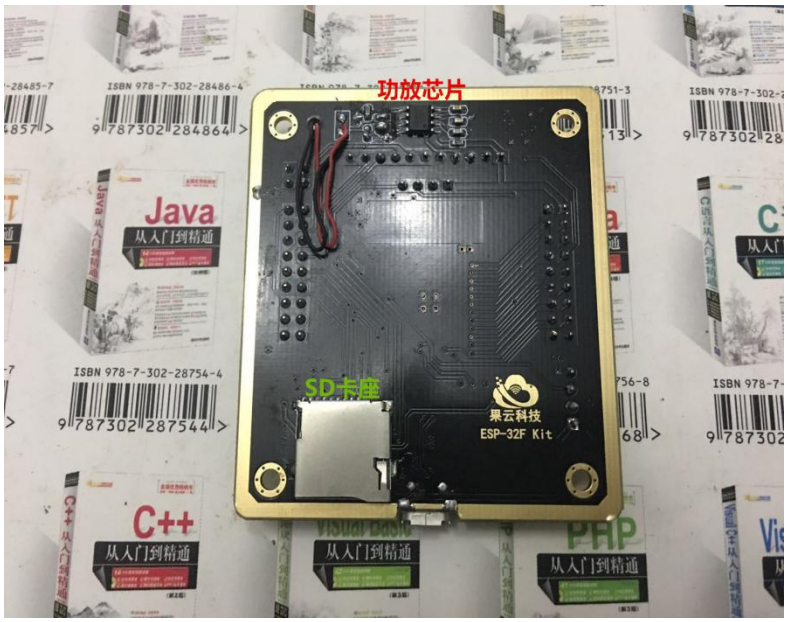
Arduino ESP-32F 开发板入门手册

果云科技 编写

目录

一、ESP-32F 开发板资源介绍






果云科技


果云科技

ESP-32F 模组特性

CPU

- Xtensa® 32-bit LX6 单 / 双核处理器，运算能力高达 600 DMIPS
- 448 KB ROM
- 520 KB SRAM
- RTC 中 16 KB SRAM
- QSPI 最多可连接 4 个 Flash / SRAM，每个 Flash 最大为 16 MB
- 供电电压：2.2V 到 3.6V
- 工作电流：平均：80 mA
- 封装尺寸：18 mm x 25.5 mm x 2.8 mm
- 温度范围：-40° C ~ +85° C *

时钟和定时器

- 内置 8 MHz 振荡器，支持自校准
- 内置 RC 振荡器，支持自校准
- 支持外置 2 MHz 至 40 MHz 的晶振
- 支持外置 32 kHz 晶振，用于 RTC，支持自校准
- 2 个定时器群组，每组包括 2 个 64-bit 通用定时器和 1 个主系统看门狗
- 具有次秒级精度的 RTC 定时器
- RTC 看门狗

外设接口

- 12-bit SAR ADC，多达 18 个通道
- 2 个 8-bit D/A 转换器
- 10 个触摸传感器
- 温度传感器
- 4 个 SPI
- 2 个 I2S
- 2 个 I2C
- 3 个 UART
- 1 个 Host SD / eMMC / SDIO
- 1 个 Slave SDIO / SPI
- 带有专用 DMA 的以太网 MAC 接口，支持 IEEE 1588
- CAN 2.0
- IR（TX / RX）

- 电机 PWM
- LED PWM，多达 16 个通道
- 霍尔传感器
- 超低噪声前置模拟放大器

Wi-Fi

标准：
- FCC/CE/IC/TELEC/KCC/SRRC/NCC

协议：
- 802.11 b/g/n/d/e/i/k/r（802.11n，速度高达 150 Mbps）
- A-PDU 和 A-MSDU 聚合，支持 0.4 µs 防护间隔

频率范围：
- 2.4 ~ 2.5 GHz

Wi-Fi 模式：
- Station/softAP/SoftAP+station/P2P

安全机制：
- WPA/WPA2/WPA2-Enterprise/WPS

加密类型：
- AES/RSA/ECC/SHA

蓝牙

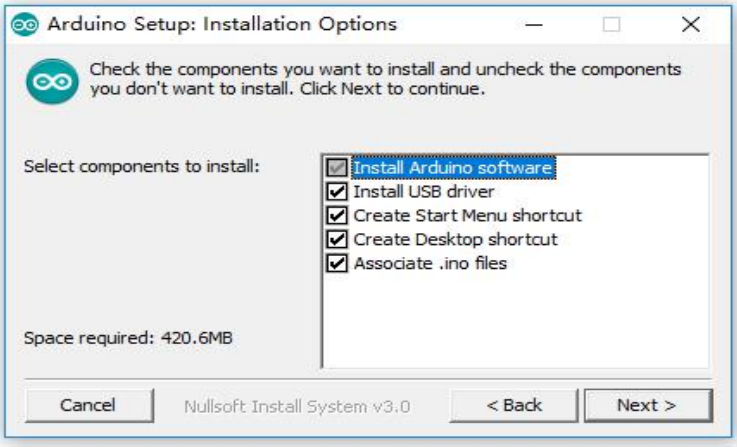
协议：
- 符合蓝牙 v4.2 BR/EDR 和 BLE 标准

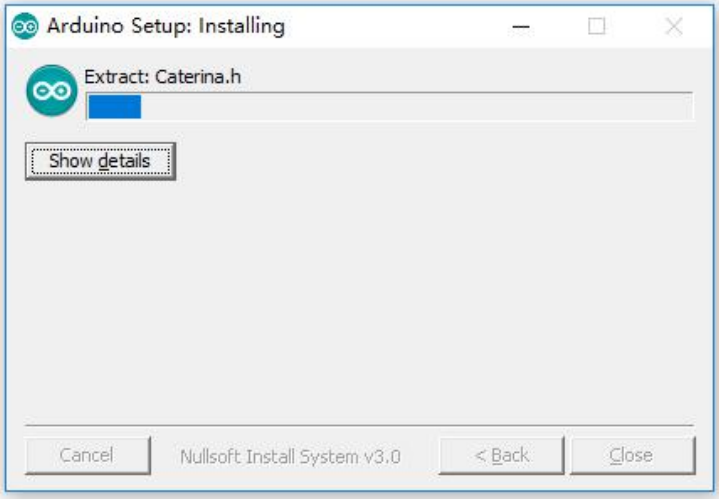
射频：
- 具有 -98 dBm 灵敏度的 NZIF 接收器
- Class-1, Class-2 和 Class-3 发射器
- AFH

音频：
- CVSD 和 SBC 音频

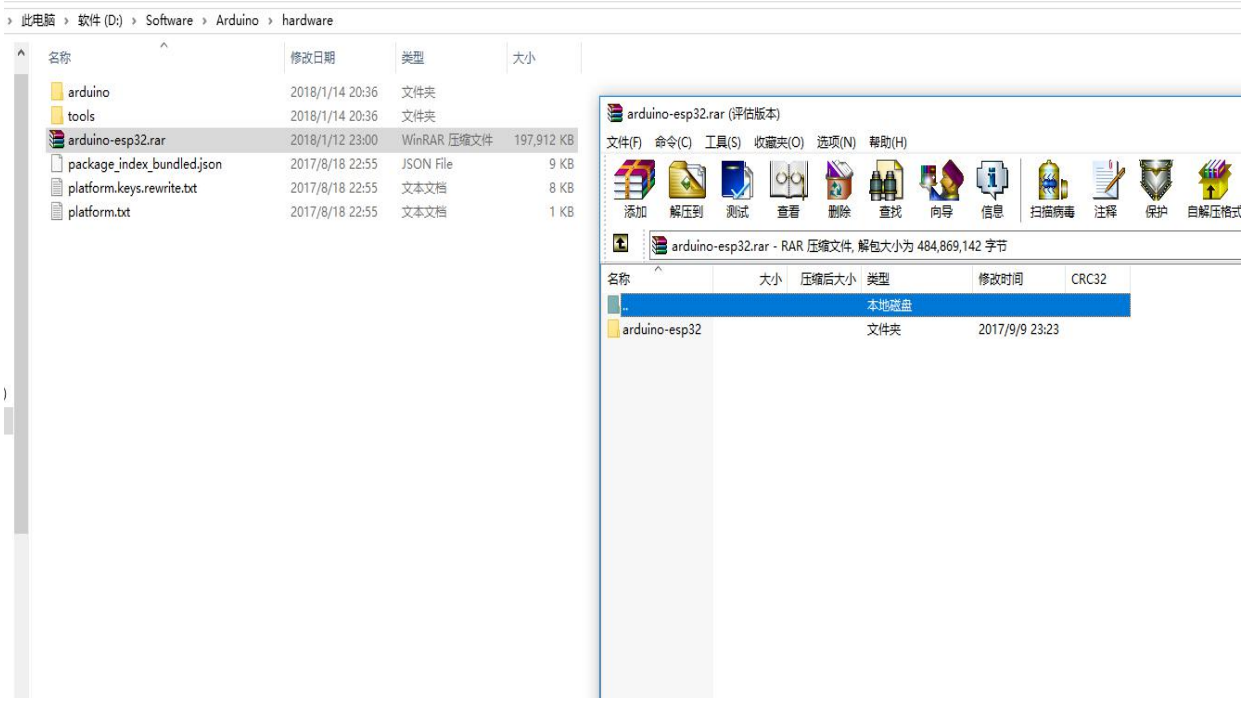
二、Arduino 环境的搭建

1.安装 Arduino IDE 环境





2. 安装完 arduino ide 后，将 arduino-esp32 这个压缩包解压到软件目录 hardware 文件夹下，arduino-esp32 内包含了 esp32 在 arduino 下的开发环境和工具。解压好后，我们就能在 arduino ide 下做 esp32 的开发了。

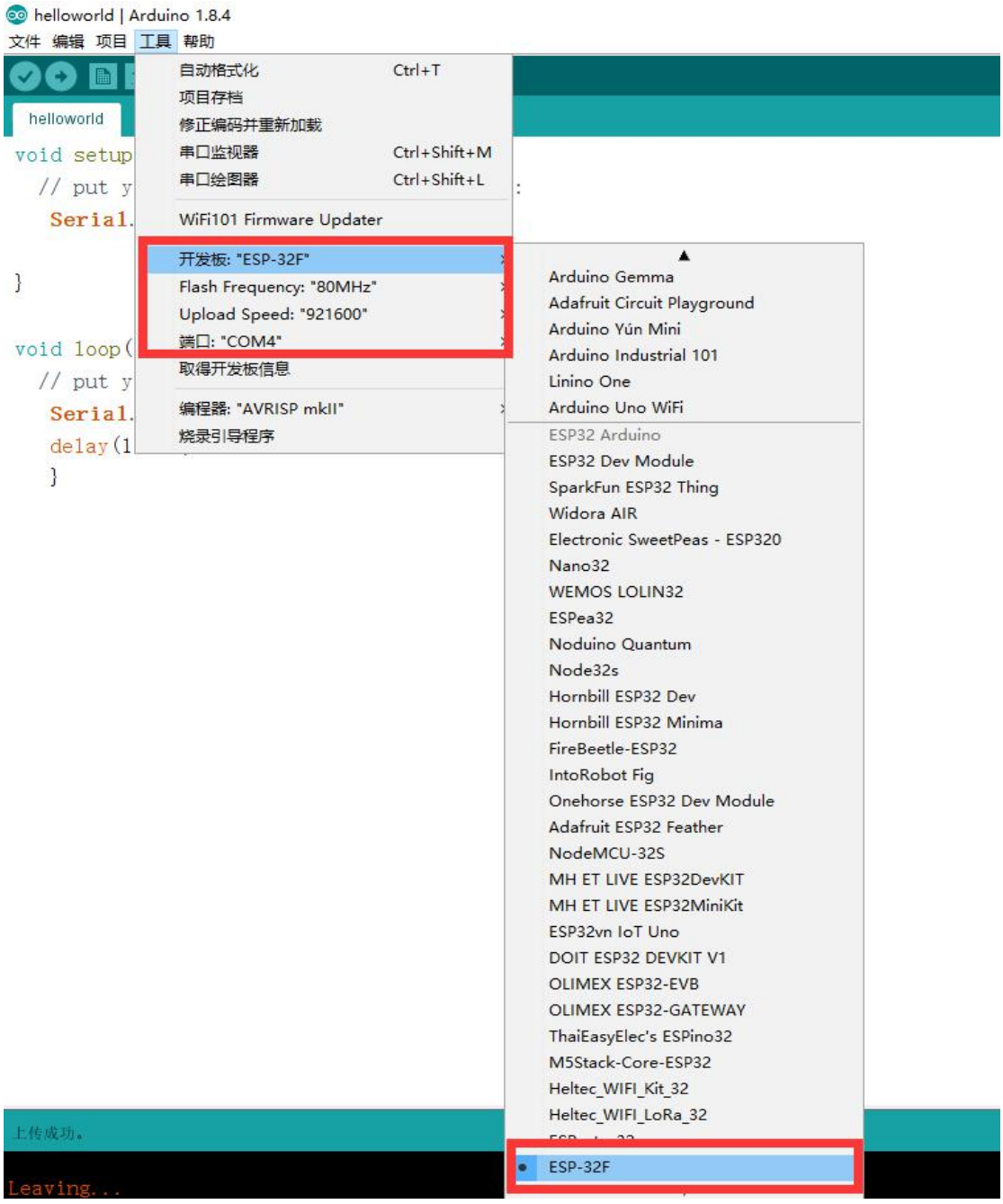


三、Hello World

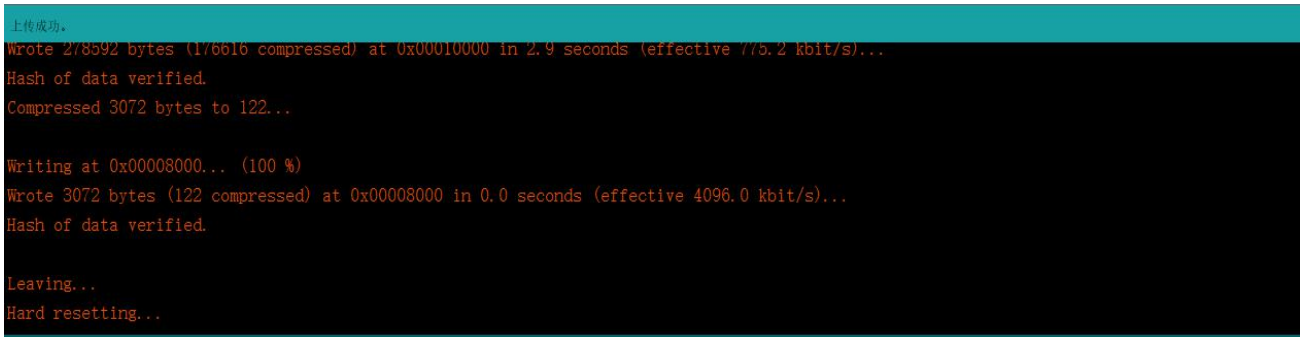
1. 打开 Arduino IDE 新建一个工程

Serial.begin(115200); //初始化串口 波特率 115200
Serial.println("Goouuu HelloWorld!"); //串口打印

2.我们先用 usb 线连接好开发板，然后点击工具，选择一下开发板型号、Flash 频率、下载波特率、串口号



2. 我们这里选编译并下载固件



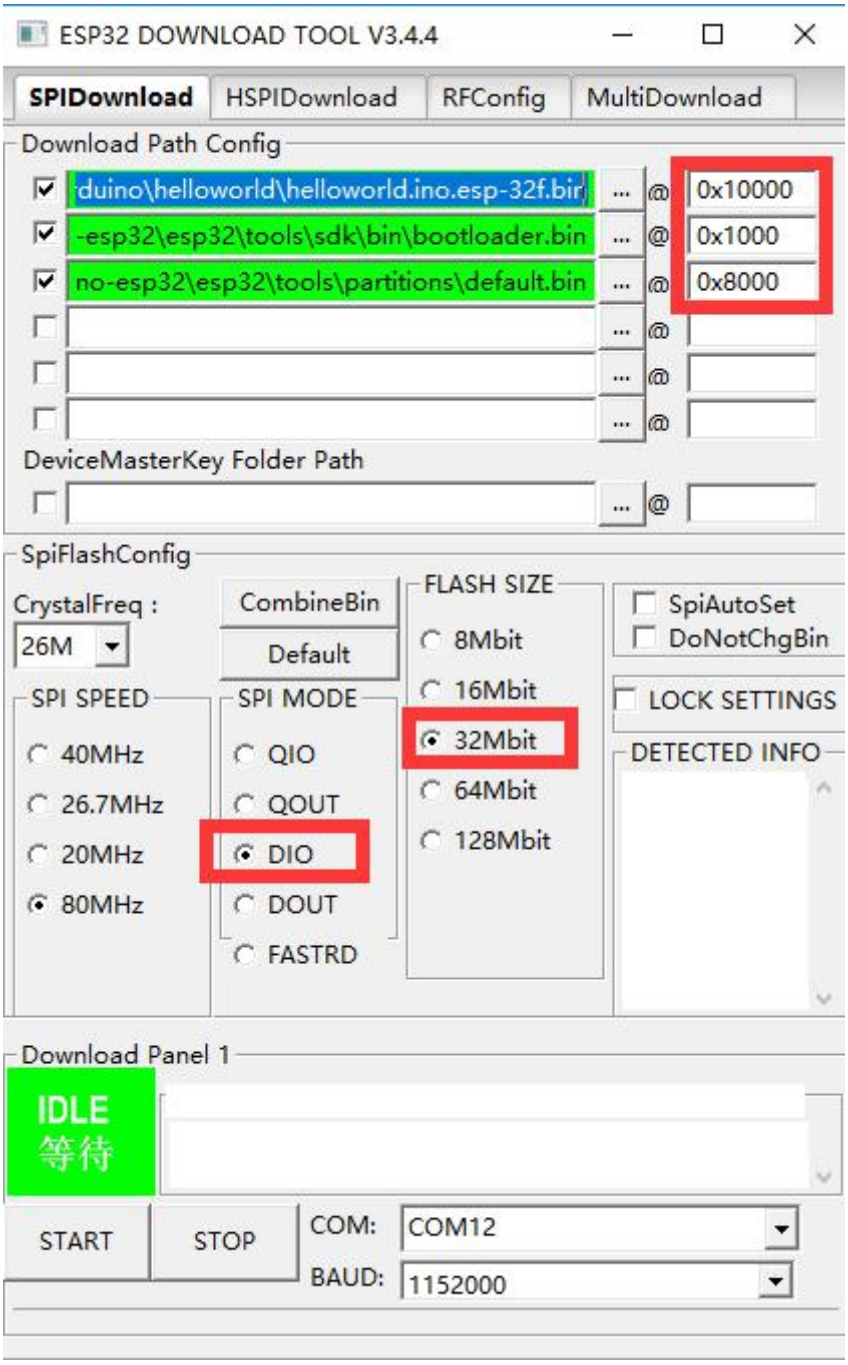
下载成功后按复位按钮运行程序。

3. 这里教大家如何用 Arduino IDE 生成一个工程的 bin 文件，然后用 ESPFlashDownloadTool 下载固件。

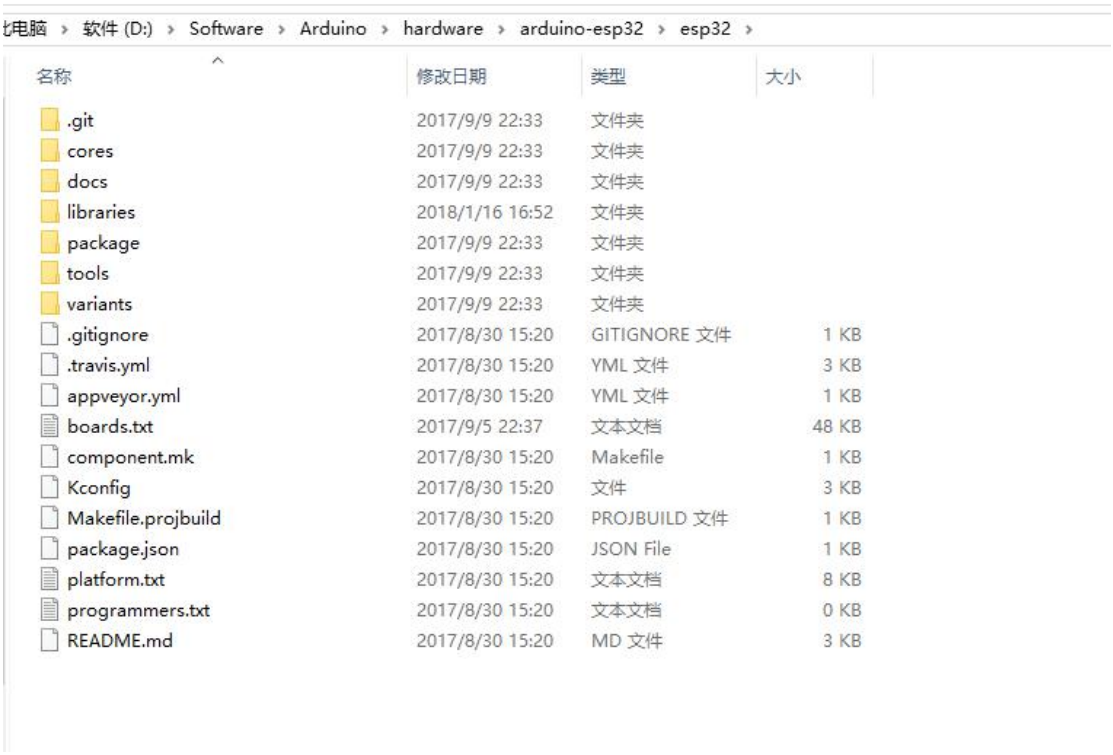
先点导出 bin 文件

除了生成的 bin 文件外，还需要 bootloader 和 default 这两个 bin 文件，程序才能正常运行。

打开 ESPFlashDownloadTool ，选择必要的文件，填写好地址和参数，点击下载



4.经过上的操作，基本掌握了建立工程，编译和下载了。接下来我们来熟悉下，Arduino ESP32 开发环境的文件结构。



Core 文件夹：乐鑫提供的 ESP32 底层内核文件，包括 gpio timer iic spi touch uarts 等外设驱动，还有二次封装好的 TCP UDP Server 等常用 arduino 库文件。

libraries 文件夹：包含各厂家的 arduino 例程，我们的例程文件夹是 ESP-32F

variants 文件夹：各种 esp32 开发板的引脚定义头文件 （如果需要添加的自己开发板，则需要在该文件夹内加入自己开发板的引脚定义头文件，添加方法可以参考其他开发板的头文件，依样画葫芦即可）

Tools 文件夹：esp32 sdk 和编译工具链

Boards.txt：定义开发板的各项参数，这些参数将会在 arduino ide 选择开发板型号和参数的面板中体现出来。

```
##############################################################

esp-32f.name=ESP-32F                    //定义开发板名称

esp-32f.upload.tool=esptool             //定义开发板下载工具
esp-32f.upload.maximum_size=1310720                 //定义Flash空间
esp-32f.upload.maximum_data_size=294912             //定义data区空间
esp-32f.upload.wait_for_upload_port=true

esp-32f.serial.disableDTR=true
esp-32f.serial.disableRTS=true

esp-32f.build.mcu=esp32
esp-32f.build.core=esp32
esp-32f.build.variant=esp-32f
esp-32f.build.board=ESP-32F

esp-32f.build.f_cpu=240000000L    //定义cpu频率
esp-32f.build.flash_mode=dio      //定义flash类型
esp-32f.build.flash_size=4MB
esp-32f.build.boot=bootloader
esp-32f.build.partitions=default

esp-32f.menu.FlashFreq.80=80MHz                 //定义可选flash频率
esp-32f.menu.FlashFreq.80.build.flash_freq=80m
esp-32f.menu.FlashFreq.40=40MHz
esp-32f.menu.FlashFreq.40.build.flash_freq=40m

esp-32f.menu.UploadSpeed.921600=921600          //定义可选波特率
esp-32f.menu.UploadSpeed.921600.upload.speed=921600
esp-32f.menu.UploadSpeed.115200=115200
esp-32f.menu.UploadSpeed.115200.upload.speed=115200
esp-32f.menu.UploadSpeed.256000.windows=256000
esp-32f.menu.UploadSpeed.256000.upload.speed=256000
esp-32f.menu.UploadSpeed.230400.windows.upload.speed=256000
esp-32f.menu.UploadSpeed.230400=230400
esp-32f.menu.UploadSpeed.230400.upload.speed=230400
esp-32f.menu.UploadSpeed.460800.linux=460800
esp-32f.menu.UploadSpeed.460800.macosx=460800
esp-32f.menu.UploadSpeed.460800.upload.speed=460800
esp-32f.menu.UploadSpeed.512000.windows=512000
esp-32f.menu.UploadSpeed.512000.upload.speed=512000
```

四、 实验 1：驱动 RGB

这里我们从最简单说起，照顾下不懂 Arduino 的同学。

steup 函数相当于 main 函数 ，程序从这里开始

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

}
```

Loop 函数相当予 while（1）{} 函数，

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

delay(x); arduino 系统延时函数 延时 x 毫秒

rand() 随机数函数，rand()%2 代表 0-1 这两个数取随机数

----------以下是实例代码----------

```
#include<stdio.h>
#include<stdlib.h>
void setup() {
  // put your setup code here, to run once:
    pinMode(32,OUTPUT);
    pinMode(33,OUTPUT);
    pinMode(27,OUTPUT);//IO27  IO32  IO33 三个引脚设置为输出模式
}

void loop() {
  // put your main code here, to run repeatedly:
    digitalWrite(32,rand()%2); //三个 LED 随机点亮，组成不同的颜色
    digitalWrite(33,rand()%2);
    digitalWrite(27,rand()%2);
    delay(1000);//延时 1000ms

}
```

五、实验 2：Serial 的使用

Arduino 串口通讯会用到 Stream 这个类

Stream 类是二进制数据或者字符串数据流传输的基础类，不能被直接调用，但可以被继承。

Stream 类

Stream 类 包含下列函数：

- available()
- read()
- flush()
- find()
- findUntil()
- peek()
- readBytes()
- readBytesUntil()
- readString()
- readStringUntil()
- parseInt()
- parseFloat()
- setTimeout()

Stream 的这些函数 都会被 Serial 库继承。

available()

说明（Description）：

该函数 available() 获取数据流中接收到的字节数

返回值（Returns）：

返回值是 int 类型

read()

说明（Description）：

该函数 read() 获取数据流中第一个字节数据，获取数据后会清除当前字节数据，与 peek()函数有区别

返回值（Returns）：

返回值是 读取数据字符的第一个字节（8bit）

flush()

说明（Description）：

该函数 flush() 清除数据流所有未向外发送的数据。

返回值（Returns）：

bool 类型

find()

说明（Description）：

该函数 find() 从数据流中查找目标字符串，找到目标字符串后返回值 = true，超时则返回值 = false

返回值（Returns）：

bool 类型

findUntil()

说明（Description）：

该函数 findUntil() 从数据流中读取目标字符串或者终止目标字符串，找到目标字符串后返回值 = true，超时则返回值 = false

语法（Syntax）：

stream.findUntil(target, terminal)

target：要搜索的字符串

terminal：终止目标字符串

返回值（Returns）：

bool 类型

peek()

说明（Description）：

该函数 peek() 从数据流中读取当前的一个字节，不会清除数据流中当前字节数据，与 read() 函数有区别。

返回值（Returns）：

当前缓存区数据流的第一个字节数据，如果缓存区无数据时返回 -1

readBytes()

说明（Description）：

该函数 readBytes() 从数据流中读取确定字节的数据到缓存区，读取确定长度数据或超时时终止

语法（Syntax）：

stream.readBytes(buffer, length)

stream：从 Stream 类 继承的实例

buffer：存放数据的缓存区（可以是 char[] 或 byte[] 这样的数组）

length：存放的字节数

返回值（Returns）：

已经存放在缓存区中的字节数

readBytesUntil()

说明（Description）：

该函数 readBytesUntil() 从数据流中读取确定字节的数据到指定缓存地址，读取确定长度数据、或读取到终止字符、或超时时终止

和 readBytes() 相比多了终止字符串

语法（Syntax）：

stream.readBytesUntil(character, buffer, length)

stream：从 Stream 类 继承的实例

character：终止字符（char 类型）

buffer：存放数据的指定缓存地址（可以是 char[] 或 byte[] 这样的数组）

length：存放的字节数（int 类型）

返回值（Returns）：

已经存放在缓存区中的字节数

readString()

说明（Description）：

该函数 readString() 从数据流中读取字符到字符串中，超时时终止

返回值（Returns）：

读取到的字符串（string）

readStringUntil()

说明（Description）：

该函数 readString() 从数据流中读取字符到字符串中，遇到终止字符，或超时时终止

语法（Syntax）：

stream.readString(terminator)

terminator：终止字符

返回值（Returns）：

读取到的字符串（string）

parseInt()

说明（Description）：

该函数 parseInt() 从数据流中读取第一个 长整型数（long），

语法（Syntax）：

返回值（Returns）：

长整型（long）

parseFloat()

说明（Description）：

该函数 parseInt() 从数据流中读取第一个有效的浮点数（float），

语法（Syntax）：

stream.parseFloat(list)

list：检查的数据流

返回值（Returns）：

浮点数（float）

setTimeout()

说明（Description）：

该函数 setTimeout() 设置等待数据流通讯超时时间，毫秒为单位

语法（Syntax）：

stream.setTimeout(time)

time：毫秒为单位的时间，长整型（long）

Serial 类

Serial 类 用于对串口数据流的读写。

Serial 继承 Stream 类，继承的函数方法参考 Stream 类；同时增加了几个新的函数，所有方法如下：

- if (Serial)
- available()
- availableForWrite()
- begin()
- end()
- find()
- findUntil()
- flush()
- parseFloat()
- parseInt()
- peek()
- print()
- println()
- read()
- readBytes()
- readBytesUntil()
- readString()
- readStringUntil()
- setTimeout()
- write()
- serialEvent()

begin()

说明（Description）：

该函数 begint() 设置串口数据传输的波特率。

语法（Syntax）：

Serial.begin(speed)
Serial.begin(speed, config)

Arduino Mega Only：

Serial1.begin(speed)

Serial2.begin(speed)

Serial3.begin(speed)

Serial1.begin(speed, config)

Serial2.begin(speed, config)

Serial3.begin(speed, config)

speed：波特率 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200

config：通讯格式 5N1, 6N1, 7N1, 8N1(默认), 5N2, 6N2, 7N2, 8N2, 5E1, 6E1, 7E1, 8E1, 5E2, 6E2, 7E2, 8E2, 5O1, 6O1, 7O1, 8O1, 5O2, 6O2, 7O2, 8O2,

返回值（Returns）：

nothing 无

end()

说明（Description）：

该函数 end() 禁用串口。禁用串口后，原串口所占用引脚被当做一般输入输出使用。

语法（Syntax）：

Serial.end()

返回值（Returns）：

nothing 无

print()

说明（Description）：

该函数 print() 将数据流通过串口以 ASCII 文本形式输出输出。

例子：

- Serial.print(78) gives "78"　　以 ASCII 码形式以此输出“7”和“8”
- Serial.print(1.23456) gives "1.23"　　浮点数默认只输出小数点后两位
- Serial.print('N') gives "N"
- Serial.print("Hello world.") gives "Hello world."
- Serial.print(78, BIN) gives "1001110"
- Serial.print(78, OCT) gives "116"
- Serial.print(78, DEC) gives "78"
- Serial.print(78, HEX) gives "4E"
- Serial.println(1.23456, 0) gives "1"
- Serial.println(1.23456, 2) gives "1.23"
- Serial.println(1.23456, 4) gives "1.2346"

语法（**Syntax**）：

Serial.print(val)

Serial.print(val, format)

val：需要输出的值，支持任何类型数据

format：指定数据格式

返回值（**Returns**）：

返回 print() 函数输出的字符数据个数，长整型（long）

println()

说明（**Description**）：

该函数 println() 将数据流通过串口以 ASCII 文本形式输出输出，并且在结尾输出换行符 (ASCII 13, 即 '\r')。

参考 print() 函数

语法（**Syntax**）：

Serial.println(val)

Serial.println(val, format)

val：需要输出的值，支持任何类型数据

format：指定数据格式

返回值（**Returns**）：

返回 println() 函数输出的字符数据个数，长整型（long）

write()

说明（Description）：

该函数 write() 将数据流通过串口以 二进制数据的形式发出，与 print() 函数是有区别的

语法（Syntax）：

Serial.write(val)

Serial.write(str)

Serial.write(buf, len)

Arduino Mega 允许使用下列串口： Serial1, Serial2, Serial3

val：单个字节的值

str：一连串字节的字符串

buf：定义的数组

len：指定的数组长度

返回值（Returns）：

返回 write() 函数通过写入串口的字节数

serialEvent()

说明（Description）：

该函数 serialEvent() 为串口中断事件函数，当串口有数据时被调用。可使用 Serial.read() 函数捕捉数据。

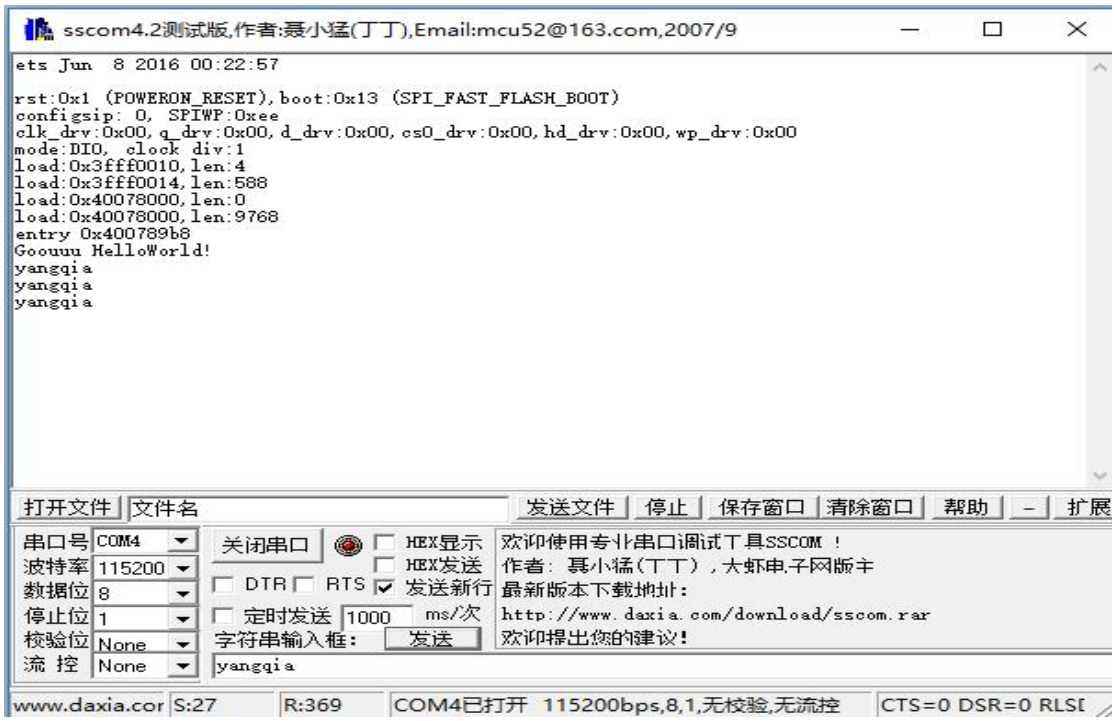
语法（Syntax）：

```
void serialEvent(){
    //statements
  }
```

我们的例程是用 Serial 类，这个类继承了 Stream 类

我们看下本节例程代码

```
/*----------------------------------------------------------------------------------------------------------------------
                                                果云科技
                                              ESP-32F 开发板

                                              Serial 实验
----------------------------------------------------------------------------------------------------------------------*/
String buf;
char temp;
void setup() {
  // put your setup code here, to run once:
    Serial.begin(115200);
    Serial.setTimeout(10);//设置串口接收超时时间 10ms （如果不明白可以改成 1000 ，用串口助手看效果）
    Serial.println("Goouuu HelloWorld!");

}

void loop() {
  // put your main code here, to run repeatedly:
     if(Serial.available()!=0)//如果接收缓冲区非空
     {
       buf=Serial.readString();//将收到的数据以字符串形式存到 buf
       Serial.print(buf);//串口打印 buf

     }
}
----------------------------------------------------------------------------------------------------------------------
```

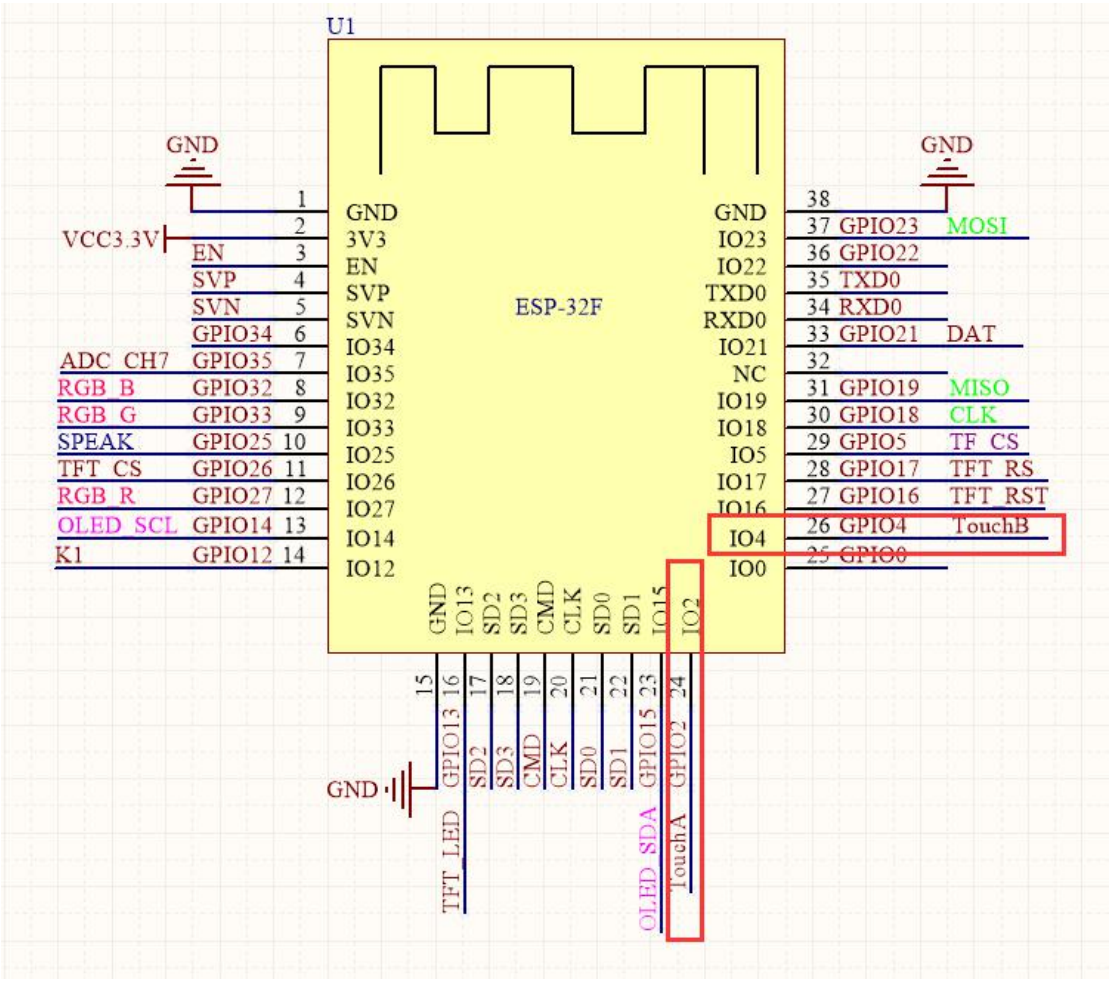


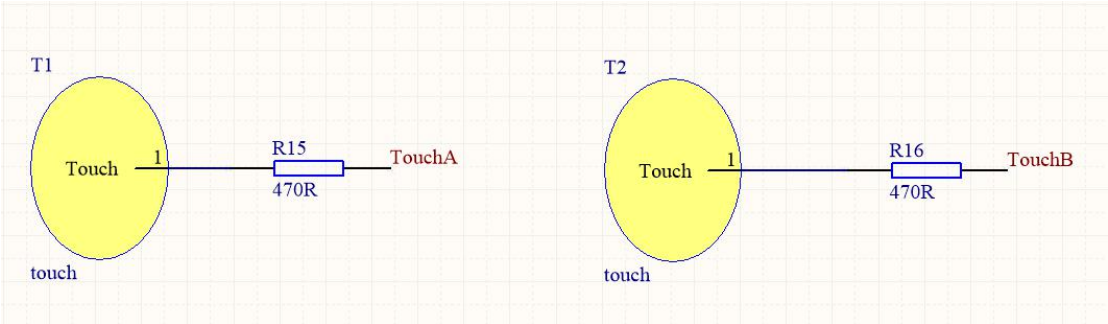
六、实验 3：触摸按键

ESP32 芯片共有 10 路 touch 通道。

12		32K_XP		VRTC	XTAL_32K_P	ADC1_CH4	TOUCH9
13		32K_XN		VRTC	XTAL_32K_N	ADC1_CH5	TOUCH8
14			GPIO25	VRTC	DAC_1	ADC2_CH8	
15			GPIO26	VRTC	DAC_2	ADC2_CH9	
16			GPIO27	VRTC		ADC2_CH7	TOUCH7
17			MTMS	VRTC		ADC2_CH6	TOUCH6
18			MTDI	VRTC		ADC2_CH5	TOUCH5
19	VDD3P3_RTC			VRTC supply in			
20			MTCK	VRTC		ADC2_CH4	TOUCH4
21			MTDO	VRTC		ADC2_CH3	TOUCH3
22			GPIO2	VRTC		ADC2_CH2	TOUCH2
23			GPIO0	VRTC		ADC2_CH1	TOUCH1
24			GPIO4	VRTC		ADC2_CH0	TOUCH0

ESP-32F 开发板上的触摸通道是用了 T0（IO4） 和 T2（IO2）





uint16_t touchRead(uint8_t pin); 获取某个通道采集的数值

```
/*----------------------------------------------------------------------------

                                果云科技
                              ESP-32F 开发板

                              触摸按键 实验
----------------------------------------------------------------------------*/
// Just test touch pin - Touch0 is T0 which is on GPIO 2.
int T0_Cul=0;
int T2_Cul=0;


void LED_Init(void)//LED 初始化
{
   pinMode(33,OUTPUT);
   pinMode(27,OUTPUT);
}



void setup()
{
  LED_Init();
  Serial.begin(115200);
  Serial.println("ESP32 Touch Test");
}
```

```
void Touch_Pad_Check(void)
{
  Serial.print("T0_Value:");
  Serial.println(touchRead(T0));  // get value using T0
  T0_Cul=touchRead(T0);
  Serial.print("T2_Value:");  // get value using T2
  Serial.println(touchRead(T2));  //
  T2_Cul=touchRead(T2);

    if(T0_Cul<=15)//通道采集数值低于 15 被认为按键按下
    {
      digitalWrite(33,0);//点亮 LEDA
    }
    else if(T0_Cul>20)//通道采集数值高于 20 被认为按键被释放
    {
     digitalWrite(33,1);//熄灭 LEDA
    }

     if(T2_Cul<=15)
   {
   digitalWrite(27,0);
   }
   else if(T2_Cul>20)
   {
    digitalWrite(27,1);
   }
}

void loop()
{
  Touch_Pad_Check();

}
```

我们可以打开串口助手查看 触摸通道实时输出的值，当我按住触摸按键 T1 不放时，T0 通道输出值都是保持在 15 以下，这时候这个通道触发的阀值就很清楚了。

七、实验 4：定时器

ESP32 提供了 4 个硬件定时器。

我们来看下官方提供的 timer 的库函数

hw_timer_t 定时器结构体 定义如下

```
typedef struct hw_timer_s {
        hw_timer_reg_t * dev;
        uint8_t num;
        uint8_t group;
        uint8_t timer;
        portMUX_TYPE lock;
} hw_timer_t;
```

我们来下看下我们要用到定时器的几个相关函数

hw_timer_t * timerBegin(uint8_t num, uint16_t divider, bool countUp){}
定时器初始化函数
参数： 定时器编号，分频数，计数模式是否是向上计数

void timerAttachInterrupt(hw_timer_t *timer, void (*fn)(void), bool edge){}
定时器中断配置函数
参数： 目标定时器，中断函数入口地址，是否跳变沿触发中断

定时器中断触发方式有 电平触发中断(level type) 边缘触发中断(edge type)

void timerAlarmWrite(hw_timer_t *timer, uint64_t alarm_value, bool autoreload){}
定时器设置函数
参数：目标定时器，计数触发阈值，是否自动重载

void timerAlarmEnable(hw_timer_t *timer){}
使能定时器函数

以下我们看下例程代码，先创建定时器 0，并且配置为 80 分频，向上计数，5 秒触发一次中断。进入中断后进入临界段，计数+1，退出临界段，释放二值信号量
创建一个定时器信号量并定义为二值信号量，退出定时器中断，在主函数循环中检测到二值信号量被置位，那么再次进入临界段，串口打印系统程序已经运行的时间和定时器计数的次数，退出临界段

```
/*-----------------------------------------------------------------------

                        果云科技
                    ESP-32F 开发板

                        定时器 实验
-----------------------------------------------------------------------*/

hw_timer_t *timer=NULL;//创建一个定时器结构体
volatile SemaphoreHandle_t timerSemaphore;//创建一个定时器信号量
int time_count=0;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
```

```
void IRAM_ATTR Timer_Hander()
{
  portENTER_CRITICAL_ISR(&timerMux);//进入临界段
  time_count++;
  portEXIT_CRITICAL_ISR(&timerMux);//退出临界段
  xSemaphoreGiveFromISR(timerSemaphore, NULL);//释放一个二值信号量 timerSemaphore

}
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  timerSemaphore=xSemaphoreCreateBinary();//定义信号量
  timer = timerBegin(0, 80, true);//初始化定时器 0 80 分频   向上计数

  //  配置定时器中断函数
  timerAttachInterrupt(timer, &Timer_Hander, true);

  // Set alarm to call onTimer function every second (value in microseconds).
  // Repeat the alarm (third parameter)
  timerAlarmWrite(timer, 5000000, true);//每计数 5000000 次触发定时器中断  自动重载开启

  // Start an alarm
  timerAlarmEnable(timer);//使能定时器函数
}

void loop() {
  // put your main code here, to run repeatedly:收到定时器触发后置位的二值信号量后打印
计数值
  if (xSemaphoreTake(timerSemaphore,0) == pdTRUE){
    portENTER_CRITICAL_ISR(&timerMux);
    Serial.println(millis());//打印系统已经运行了多少时间
    Serial.println(time_count);//计数值
    portEXIT_CRITICAL_ISR(&timerMux);
  }

}
```

Arduino -ESP32 官方移植了 FreeRTOS 系统，这里顺带了解下。

portENTER_CRITICAL_ISR(&timerMux);//进入临界段

portEXIT_CRITICAL_ISR(&timerMux);//退出临界段

在 portmacro.h 可以找到这两个函数的宏定义

#define portENTER_CRITICAL_ISR(mux)　　vTaskEnterCritical(mux)

#define portEXIT_CRITICAL_ISR(mux)　　vTaskExitCritical(mux)

taskENTER_CRITICAL()　用于进入临界区的宏。在临界区中不会发生上下文切换。是最紧急的临界区，因为中断被禁止后，tick 中断和任何硬件中断都不能响应，CPU 只能执行 taskENTER_CRITICAL()范围内的代码。

关于 portMUX_TYPE 结构体

```
typedef struct {
    volatile uint32_t mux;
#ifdef CONFIG_FREERTOS_PORTMUX_DEBUG
    const char *lastLockedFn;
    int lastLockedLine;
#endif
} portMUX_TYPE;
```

```
#define portMUX_MAGIC_VAL        0xB33F0000
#define portMUX_FREE_VAL         0xB33FFFFF
#define portMUX_MAGIC_MASK       0xFFFF0000
#define portMUX_MAGIC_SHIFT      16
#define portMUX_CNT_MASK         0x0000FF00
#define portMUX_CNT_SHIFT        8
#define portMUX_VAL_MASK         0x000000FF
#define portMUX_VAL_SHIFT        0

//Keep this in sync with the portMUX_TYPE struct definition please.
#ifndef CONFIG_FREERTOS_PORTMUX_DEBUG
#define portMUX_INITIALIZER_UNLOCKED {                  \
        .mux = portMUX_MAGIC_VAL|portMUX_FREE_VAL       \
    }
#else
#define portMUX_INITIALIZER_UNLOCKED {                  \
        .mux = portMUX_MAGIC_VAL|portMUX_FREE_VAL,      \
        .lastLockedFn = "(never locked)",               \
        .lastLockedLine = -1                            \
    }
#endif
```

关于二值信号量的理解

二值型信号量的使用方法见图1所示，二值型信号量可以理解为任务与中断间或者两个任务间的标志，该标志非"满"即"空"。Send操作相当把该标志置"满"，Receive操作相关与把该标志取"空"，经过send和receive操作实现任务与中断间或者两任务的操作同步。

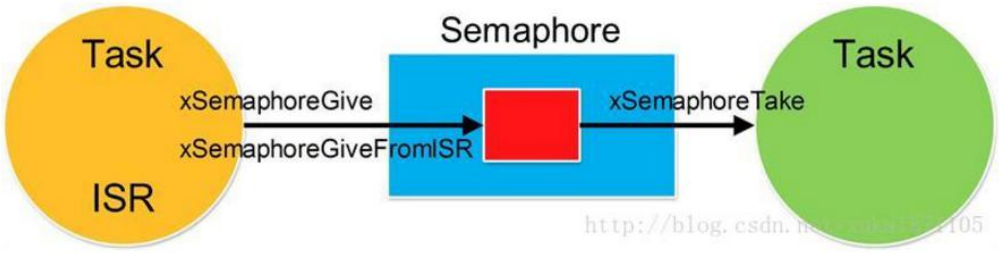
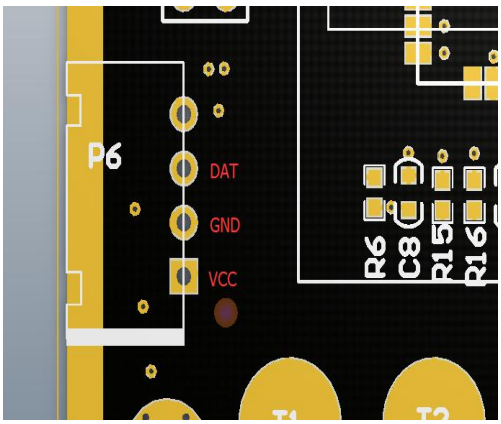


图1 二值型信号量使用示意图

八、实验 5：读取 DHT11 温湿度传感器



先用杜邦线将 DHT11 传感器和开发板上的传感器接口对应连接起来。

DHT11 输出两个字节湿度数据 两个字节温度数据 一个校验字节 共五个字节

库函数:

DHT11　dht11(21); 对 DHT11 这个类实例化 选择 IO21 作为单总线引脚

digitalRead(pin)　读取该引脚的电平状态

digitalWrite(pin,x) 该引脚输出高或低电平

delayMicroseconds(x)　arduino 系统微妙延时函数

DHT11.Start() 开始读数据

DHT11.ReadByte()　读取 dht11 一个字节的数据

DHT11.Read_Value() 读取 dht11 一帧数据 5 个字节

DHT11.Get_DHT11_Value()　获取一帧数据 并转化为 10 进制，并串口打印出来

都是比较简单的函数调用，不多讲，直接看例程即可。

我们看到这里面用到的库文件是 cpp 格式，aduino ide 是支持 c++类库的，我们来说下如何建立自己封装的 c++类库，方便自己日后调用。

1.点击下拉箭头 新建标签

输入文件名　xxx.cpp　后确定

同理，建立多一个 xxx.h 的头文件

好了之后我们先写头文件

创建一个类

需要外部调用和访问的就声明成 public,不需要外部访问的就声明为 private

构造函数：

在类实例化对象时自动执行，对类中的数据进行初始化。构造函数可以从载，可以有多个，但是只能有一个缺省构造函数。构造函数必须和类同名

析构函数：

在撤销对象占用的内存之前，进行一些操作的函数。析构函数不能被重载，只能有一个。

析构函数如果我们不写的话，C++ 会帮我们自动的合成一个，就是说：C++ 会自动的帮我们写一个析构函数。很多时候，自动生成的析构函数可以很好的工作，但是一些重要的事迹，就必须我们自己去写析构函数。

析构函数和构造函数是一对。构造函数用于创建对象，而析构函数是用来撤销对象。简单的说：一个对象出生的时候，使用构造函数，死掉的时候，使用析构函数。

```
class DHT11
{
  private:
         uint8  pin;
  public:  //公共方法
     DHT11(uint8  p);//构造函数
     ~DHT11(uint8  p);//析构函数
     void PortIN();//DHT11 引脚设置为输入模式
     void PortOUT();//DHT11 引脚设置为输出模式
     uint8  Start();//开始读取数据
     uint8  ReadByte();//读取一个字节的数据
     uint8  Read_Value(uint8  *dht);//读取5个字节，读取一帧温湿度数据
     void NumToString(uint8  dht,uint8  *str);
     void Get_DHT11_Value();//获取一帧数据并且打印
};
```

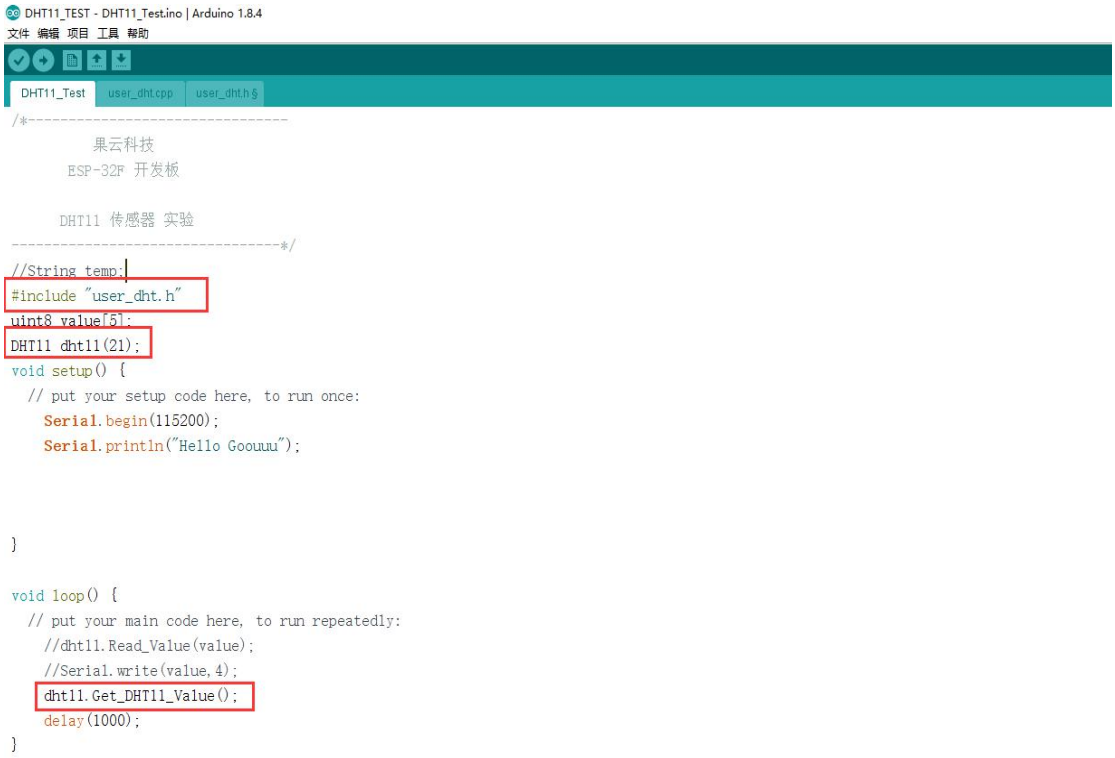
成员函数

定义好了头文件之后，我们看下 cpp 文件。我们逐一实现在头文件中定义的方法，注意类型要与定义类型相对应。所有的方法都要属于你定义的类名，格式如下

类名::方法名

```
#include "user_dht.h"
#include "Arduino.h"

DHT11 :: DHT11(uint8  p)
{
    pin=p;
}

void DHT11:: PortIN()
{
   pinMode(pin,INPUT);
}

void DHT11:: PortOUT()
{
   pinMode(pin,OUTPUT);
}

uint8  DHT11:: Start()
{
   uint8  result = 0;
   PortOUT();
   DHT11_Pin_Low;
   delay(20);
   DHT11_Pin_Hig;
   delayMicroseconds(45);
   PortIN();
   delayMicroseconds(5);
   while(DHT11_Pin_In==0)
   {
```

.cpp 和 .h 文件都构建好了之后，我们在主文件调用.h 文件，然后对定义的类进行实例化，就可以调用类的方法了。是不是很简单呢！



九、实验 6：ADC 采集 GP2Y1014AU 粉尘传感器

ESP32　ADC 库函数

void analogSetSamples(uint8_t samples);
设置分频 1-255

bool adcAttachPin(uint8_t pin);
设置 ADC 采集引脚

uint16_t analogRead(uint8_t pin);
读取 ADC 通道值

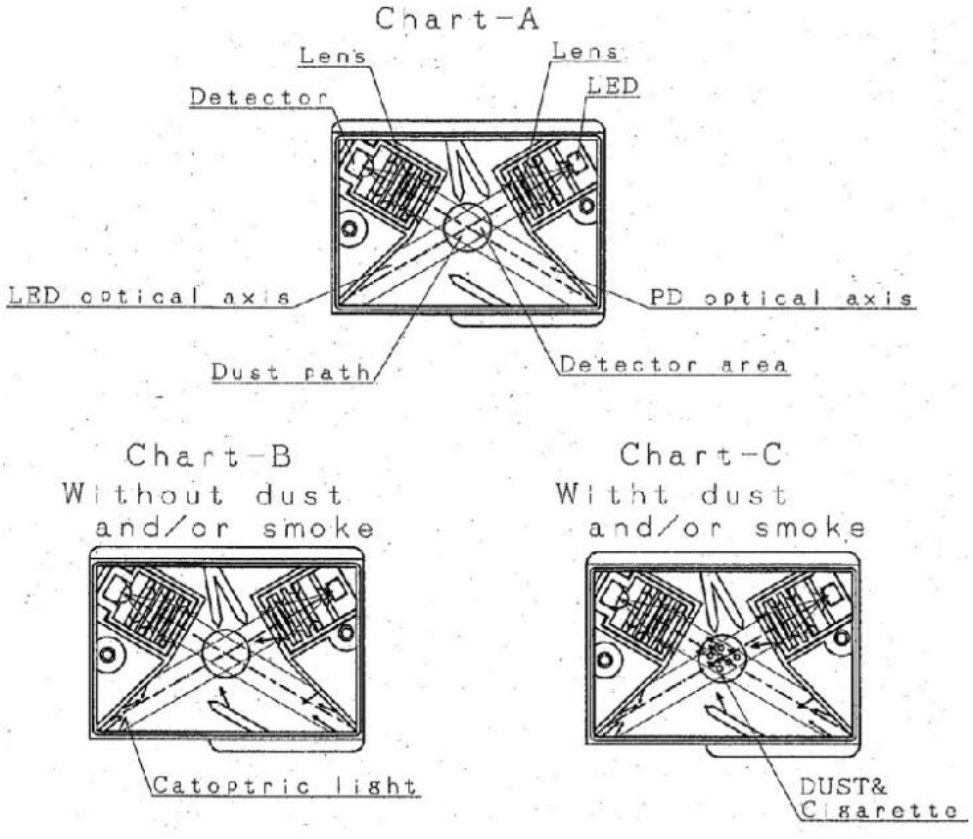
bool adcStart(uint8_t pin);
开始 ADC 采集

GP2Y1014AU 粉尘传感器

电源电压：5-7V
工作温度：-10-65 摄氏度
消耗电流：20mA 最大
最小粒子检出值：0.8 微米
灵敏度：0.5V/(0.1mg/m3)
清洁空气中电压：0.9V 典型值
工作温度：-10~65℃
存储温度：-20~80℃
使用寿命：5 年
尺寸大小：46mm×30mm×17.6mm

检测原理

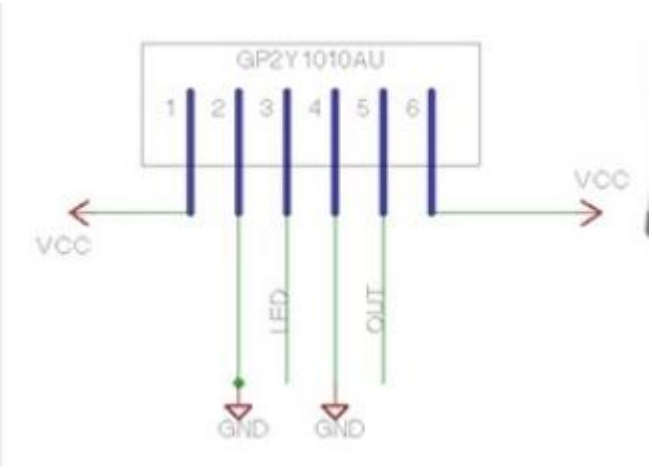
其原理如下图，传感器中心有个洞可以让空气自由流过，定向发射LED光，通过检测经过空气中灰尘折射过后的光线来判断灰尘的含量。



排线位置：


R寸:4.6*3.0*1.76cm
设置排列线
①V-LED
②LED-GND
③LED
④S-GND
⑤VO
⑥VCC
1 2 3 4 5 6

接线


GP2Y1010AU
1 2 3 4 5 6
VCC
VCC
LED
OUT
GND
GND

```
/*-------------------------------------------------------------------------------------------------------------------------

                                             果云科技
                                          ESP-32F 开发板

                                    ADC 采集 粉尘传感器 实验
-------------------------------------------------------------------------------------------------------------------------*/

#include "Arduino.h"
#define    dustPin    35    //定义 adc 采集引脚
#define    ledPower 21    //定义 led 发射引脚
int value;
float dustVal=0;


int delayTime=280;
int delayTime2=40;
float offTime=9680;

void setup() {
   // put your setup code here, to run once:
     pinMode(ledPower,OUTPUT);
     adcAttachPin(dustPin);
     adcStart(dustPin);
     Serial.begin(115200);


}

void loop() {
   // put your main code here, to run repeatedly:
          digitalWrite(ledPower,LOW);
          delayMicroseconds(delayTime);
          dustVal=analogRead(dustPin);
        // Serial.write(value>>8); Serial.write(value);
          delayMicroseconds(delayTime2);
          digitalWrite(ledPower,HIGH);
          delayMicroseconds(offTime);

          delay(1000);
          if (dustVal>36.455)
          Serial.println((float(dustVal/4096)-0.0356)*120000*0.035);
}
```

测试得到的数据和空气质量对照：

3000 + = 很差

1050-3000 = 差

300-1050 = 一般

150-300 = 好

75-150 = 很好

0-75 = 非常好

十、实验 7：ESP32 I2C 驱动 OLED

产品参数：

1、高分辨率：128*64（和12864LCD相同分辨率，但该OLED屏的单位面积像素点多）
2、超广可视角度：大于160°
3、超低功耗：正常显示时0.06W
4、宽供电范围：直流3.3V-5V
5、工业级：工作温度范围-30℃～70℃
6、体积小：27mm*27mm*2mm
7、通信方式：IIC、SPI（默认4线）
8:、亮度、对比度可以通过程序指令控制
9:、使用寿命不少于16000小时
10、OLED屏幕内部驱动芯片：SSD1306

0.96 OLED 和开发板的连线图

VCC ----3.3V
GND----GND
SCL----- GPIO14
SDA---- GPIO15

下面介绍下 OLED 的常用库函数

bool init();　液晶屏初始化函数

void resetDisplay(void); 液晶屏复位函数

void setColor(OLEDDISPLAY_COLOR color); 设置画笔颜色

void setPixel(int16_t x, int16_t y);　画点

void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1); 画线

void drawRect(int16_t x, int16_t y, int16_t width, int16_t height);　画矩形

void fillRect(int16_t x, int16_t y, int16_t width, int16_t height); 画填充

void drawCircle(int16_t x, int16_t y, int16_t radius);　画圆形

void clear(void);　清屏

void displayOn(void); 开显示

void displayOff(void);　关显示

void invertDisplay(void);　反显

void drawString(int16_t x, int16_t y, String text); 写字符串

void drawXbm(int16_t x, int16_t y, int16_t width, int16_t height, const char *xbm); 画一张图片

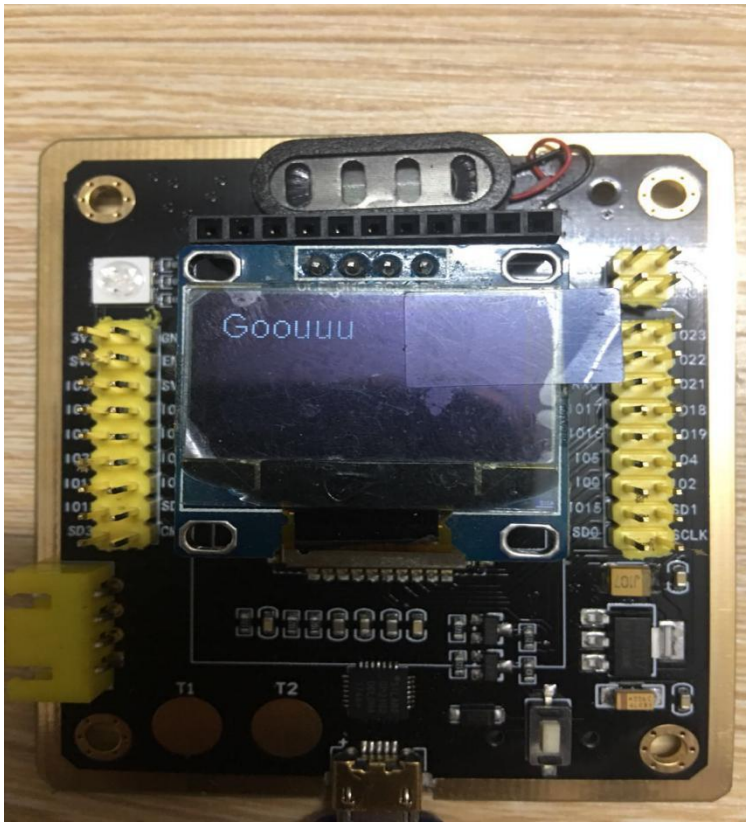
以下是画一组动态圆形的例程：

```
/*-----------------------------------------------------------------------------------------------------------------------

                                        果云科技
                                    ESP-32F 开发板

                                      OLED 实验
-----------------------------------------------------------------------------------------------------------------------*/
#include "SSD1306.h"
#include "images.h"
SSD1306 oled(0x3c,15,14);   //实例化 SSD1306 类

void drawCircle(void) {
  for (int16_t i=0; i<DISPLAY_HEIGHT; i+=2) {
    oled.drawCircle(DISPLAY_WIDTH/2, DISPLAY_HEIGHT/2, i);
    oled.display();
    delay(10);
  }
  delay(1000);
  oled.clear();

  // This will draw the part of the circel in quadrant 1
  // Quadrants are numberd like this:
  //      0010 | 0001
  //    ------|-----
  //      0100 | 1000
  //
  oled.drawCircleQuads(DISPLAY_WIDTH/2,DISPLAY_HEIGHT/2,DISPLAY_HEIGHT/4,0b00000001);
  oled.display();
  delay(200);
  oled.drawCircleQuads(DISPLAY_WIDTH/2,DISPLAY_HEIGHT/2,DISPLAY_HEIGHT/4,0b00000011);
  oled.display();
  delay(200);
  oled.drawCircleQuads(DISPLAY_WIDTH/2,DISPLAY_HEIGHT/2,DISPLAY_HEIGHT/4,0b00000111);
  oled.display();
  delay(200);
  oled.drawCircleQuads(DISPLAY_WIDTH/2,DISPLAY_HEIGHT/2,DISPLAY_HEIGHT/4,0b00001111);
  oled.display();
}
void setup() {
  // put your setup code here, to run once:
    oled.init();//初始化 OLED
    oled.flipScreenVertically();//
    oled.setContrast(255);//设置对比度 255
```

```
    drawCircle();//画动态圆形的函数
    //oled.drawXbm(0,0, Logo_width,Logo_height, Logo_bits);
    oled.display();//显示内容
}

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

}
```

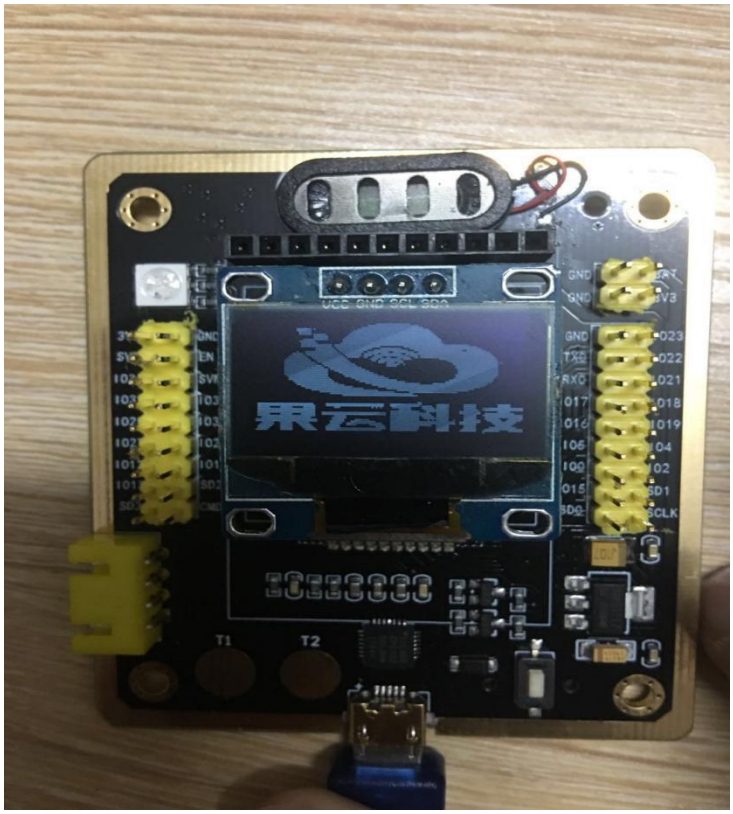
如果是要打印字符串可以在主函数下添加如下代码
oled.setFont(ArialMT_Plain_16);//设置字体为 16 号
如果想变其他字体 可以看下 OLEDDisplayFonts.h 这个文件，选择自己需要的字体。

```
oled.drawString(1,1,"Goouuu");//打印字符串
```

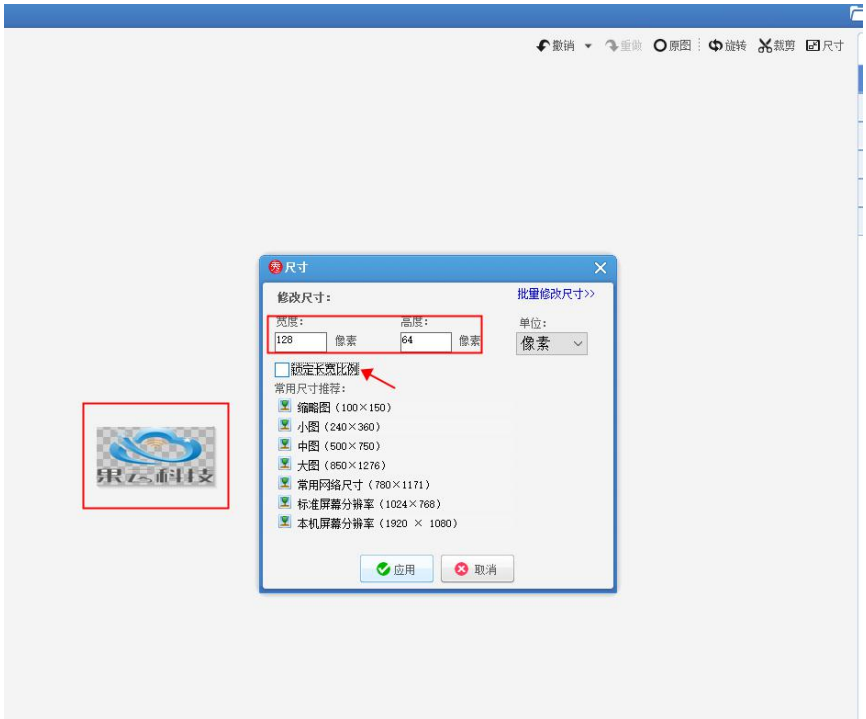


如果想打印一张位图 那么可以用这个函数

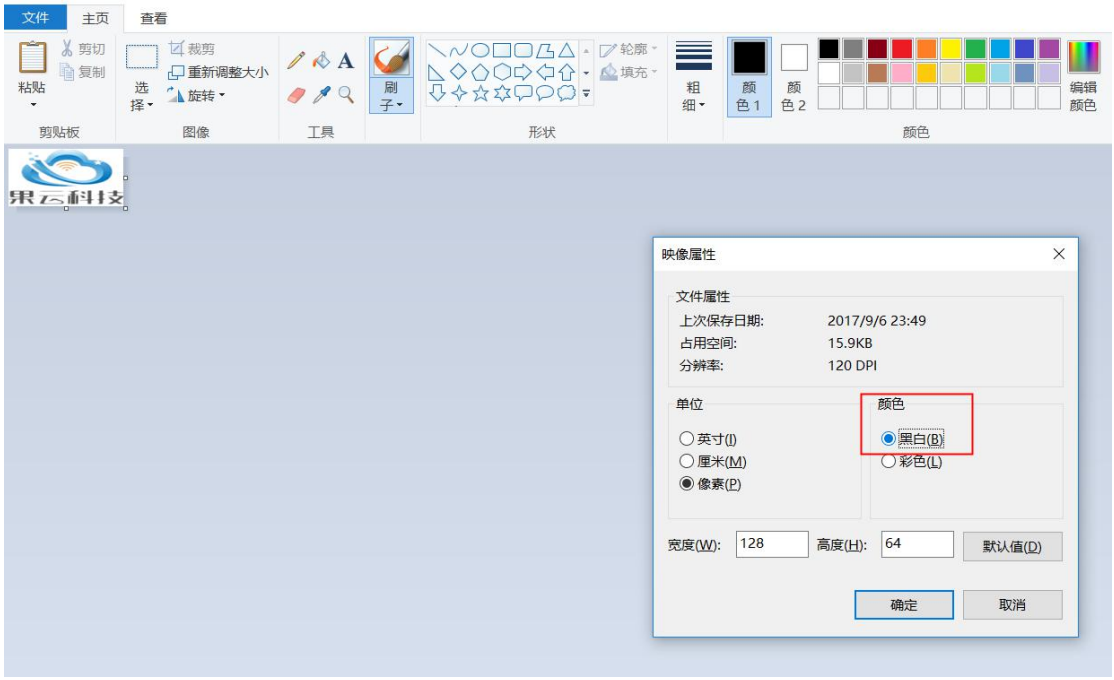
oled.drawXbm(0,0, Logo_width,Logo_height, Logo_bits);



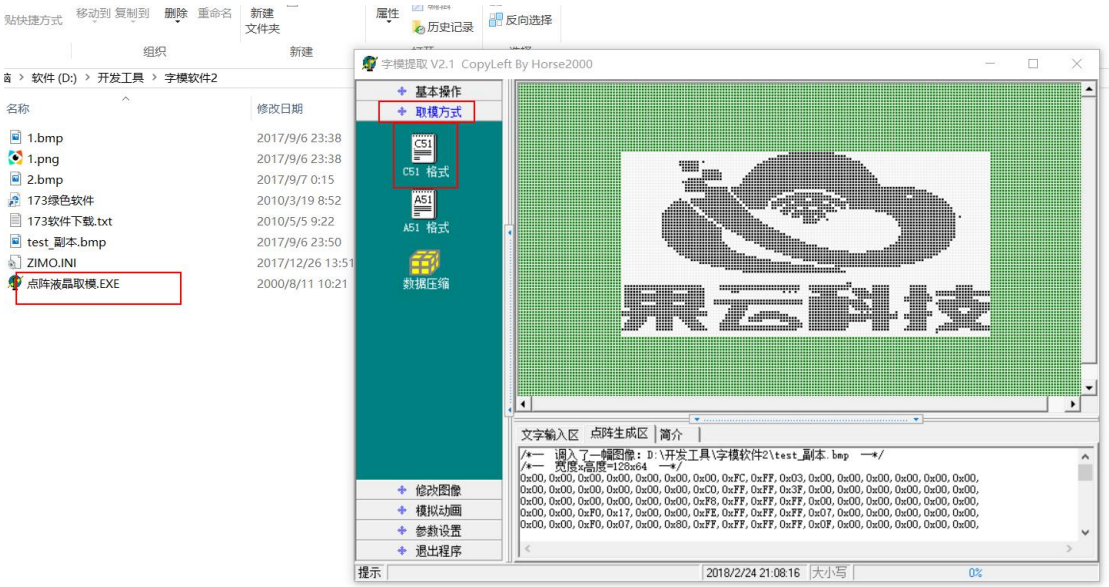
图片取模过程如下： 先用美图秀秀，将图片尺寸强制变成 128x64 并保存为 png 格式



用画图软件打开修改后的 png 图片 设置为黑白 点确定



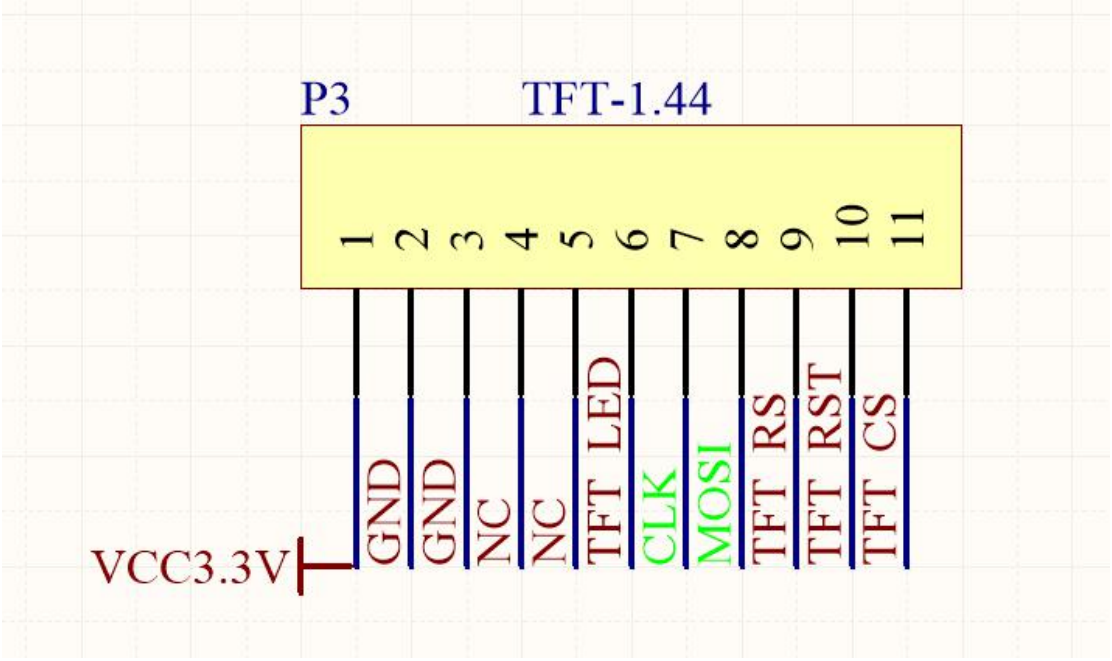
打开取模软件，调入黑白图像，点 c51 格式，然后把取到的数据复制到 arduino 里图像的数组即可。



十一、实验 8：ESP32　SPI 驱动 LCD

我们开发板上使用的 1.44 寸的 LCD 液晶屏

液晶屏和开发板的连接和 ESP32 所对应的引脚如下图




```
#define LCD_RS          17//RS
#define LCD_CS          26//CS
#define LCD_RST         16//复位
#define LCD_SCL         18//CLK
#define LCD_SDA         23//SDI
#define LCD_LED         13//背光
```

下面说下 LCD 驱动函数

```
void LCD:: LCD_GPIOInit(void)    //初始化引脚
{
   pinMode(LCD_RS, OUTPUT);
   pinMode(LCD_CS, OUTPUT);
   pinMode(LCD_RST, OUTPUT);
   pinMode(LCD_LED, OUTPUT);
   SPI.begin(LCD_SCL, 22, LCD_SDA, LCD_CS);//初始化 SPI
   //SPI.beginTransaction(SPISettings(500000, MSBFIRST, SPI_MODE2));
}
```

```
begin(int8_t sck, int8_t miso, int8_t mosi, int8_t ss)
```

我们看到 beigin 函数原型，4 个参数是 SPI 所要映射的 GPIO 口，我们上面 miso 填 22，是以为 LCD 没有用到 miso 这个脚，我们随便填了 1 个 22 作为参数。

```
//******************************************************************
//函数名：  LCD_WR_REG
//功能：    向液晶屏总线写入写 16 位指令
//输入参数：Reg:待写入的指令值
//返回值：  无
//修改记录：无
//******************************************************************


void LCD:: LCD_WR_REG(u16 data)
{
  LCD_CS_CLR;
  LCD_RS_CLR;

  SPI.write(data);
  LCD_CS_SET;
}

//******************************************************************
//函数名：  LCD_WR_DATA
//功能：    向液晶屏总线写入写 8 位数据
//输入参数：Data:待写入的数据
//返回值：  无
//修改记录：无
//******************************************************************

void LCD:: LCD_WR_DATA(u8 data)
{

  LCD_CS_CLR;
  LCD_RS_SET;

  // SPIv_WriteData(data);
  SPI.write(data);
  LCD_CS_SET;

}
//******************************************************************
//函数名：  LCD_DrawPoint_16Bit
//功能：    8 位总线下如何写入一个 16 位数据
//输入参数：(x,y):光标坐标
//返回值：  无
//修改记录：无
//******************************************************************
```

```
void LCD:: LCD_WR_DATA_16Bit(u16 data)
{
  LCD_CS_CLR;
  LCD_RS_SET;

  //SPIv_WriteData(data>>8);
  // SPIv_WriteData(data);
  SPI.write(data >> 8);
  SPI.write(data);
  LCD_CS_SET;
}

//*****************************************************************
//函数名：    LCD_WriteReg
//功能：      写寄存器数据
//输入参数：LCD_Reg:寄存器地址
//        LCD_RegValue:要写入的数据

//*****************************************************************

void LCD:: LCD_WriteReg(u16 LCD_Reg, u16 LCD_RegValue)
{
  LCD_WR_REG(LCD_Reg);
  LCD_WR_DATA(LCD_RegValue);
}

//*****************************************************************
//函数名：    LCD_WriteRAM_Prepare
//功能：      开始写 GRAM
//        在给液晶屏传送 RGB 数据前，应该发送写 GRAM 指令

//*****************************************************************

void LCD:: LCD_WriteRAM_Prepare(void)
{
  LCD_WR_REG(lcddev.wramcmd);
}
```

LCD 的底层驱动基本就是以上这些了，具体操作时序可以看下液晶屏控制器的 PDF。
下面我们接着看下 LCD 的一些参数宏定义和常用库函数，方便大家调用。

#if USE_HORIZONTAL==1　　//定义是否使用横屏　　0,不使用.1,使用.

extern u16　POINT_COLOR;//画笔颜色，默认红色
extern u16　BACK_COLOR; //背景颜色.默认为白色

```
//画笔颜色
#define WHITE         0xFFFF
#define BLACK         0x0000
#define BLUE          0x001F
#define BRED          0XF81F
#define GRED          0XFFE0
#define GBLUE         0X07FF
#define RED           0xF800
#define MAGENTA       0xF81F
#define GREEN         0x07E0
#define CYAN          0x7FFF
#define YELLOW        0xFFE0
#define BROWN         0XBC40
#define BRRED         0XFC07
#define GRAY          0X8430
```

void LCD_SetCursor(u16 Xpos, u16 Ypos);　//设置坐标函数

void LCD_SetWindows(u16 xStar, u16 yStar, u16 xEnd, u16 yEnd);//设置窗口大小

void LCD_Clear(u16 Color);　//LCD 清屏

void LCD_DisplayOn(void);//开显示

void LCD_DisplayOff(void);//关显示

void LCD_DrawPoint(u16 x, u16 y); //画一个点

u16　LCD_ReadPoint(u16 x, u16 y);　//读一个点

void LCD_DrawLine(u16 x1, u16 y1, u16 x2, u16 y2);//画线

void LCD_DrawRectangle(u16 x1, u16 y1, u16 x2, u16 y2);//LCD 画矩形

void LCD_Fill(u16 sx,u16 sy,u16 ex,u16 ey,u16 color);//画填充矩形

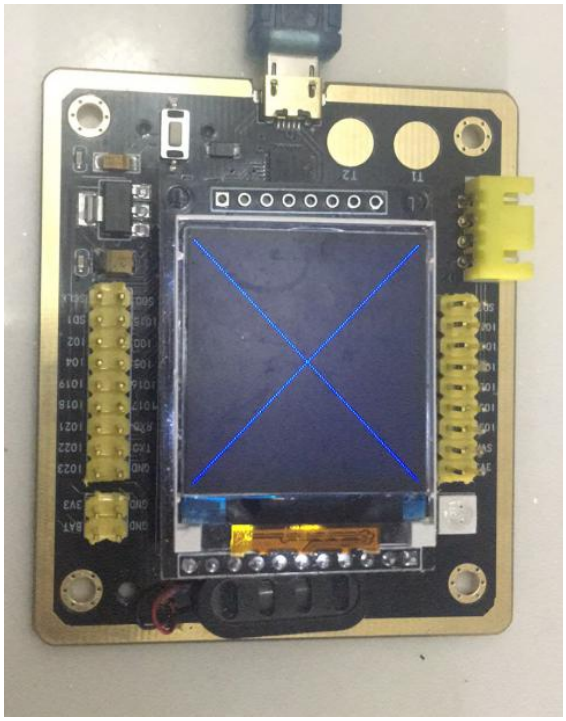
void Draw_Circle(u16 x0,u16 y0,u16 fc,u8 r); //画圆形

void LCD_ShowString(u16 x,u16 y,u8 lenth,const char *p,u8 mod);//写字符串函数

void Gui_Drawbmp16(u16 lenth,u16 width,u16 x,u16 y,const unsigned char *p); //画一张图片

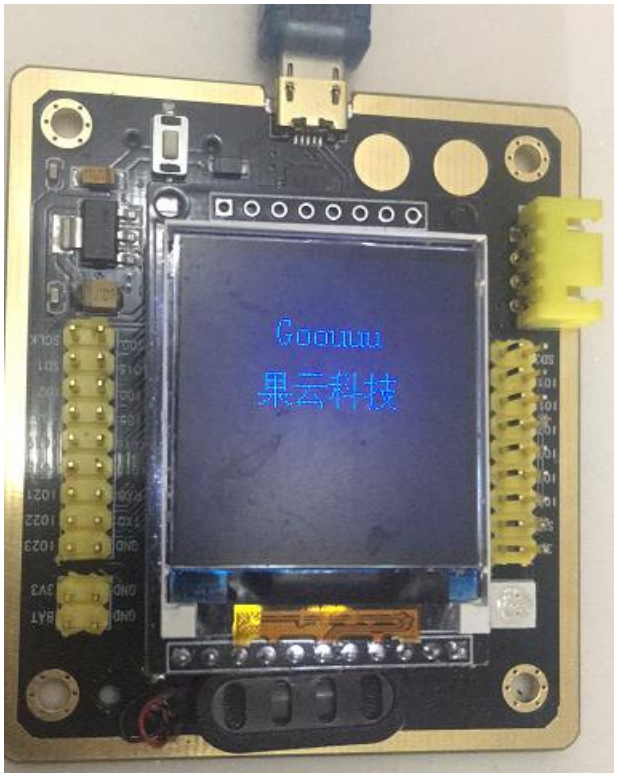
我们看下画线的例子：

```
#include"lcd.h"
#include"icon.h"
LCD lcd;  //实例化这个类

void setup() {
  // put your setup code here, to run once:
      lcd.LCD_GPIOInit();
      lcd.LCD_Init();
      lcd.LCD_Clear(BLACK); //清屏幕，并把背景设为黑色
      POINT_COLOR=BLUE;//设置画笔颜色蓝色
      lcd.LCD_DrawLine(0,127,127,0); //      画两条直线
      lcd.LCD_DrawLine(0,0,127,127); //
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

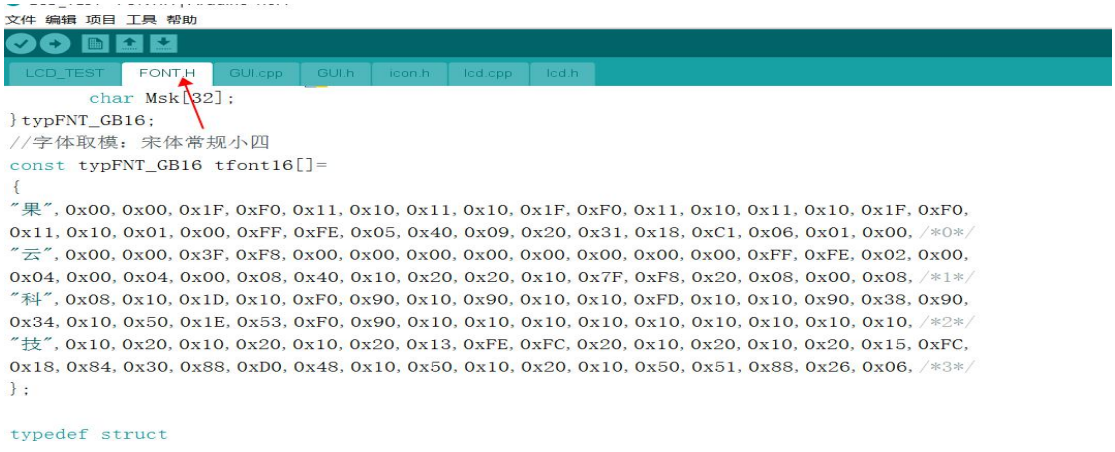


接下来我们修改下主函数，显示英文和中文字符

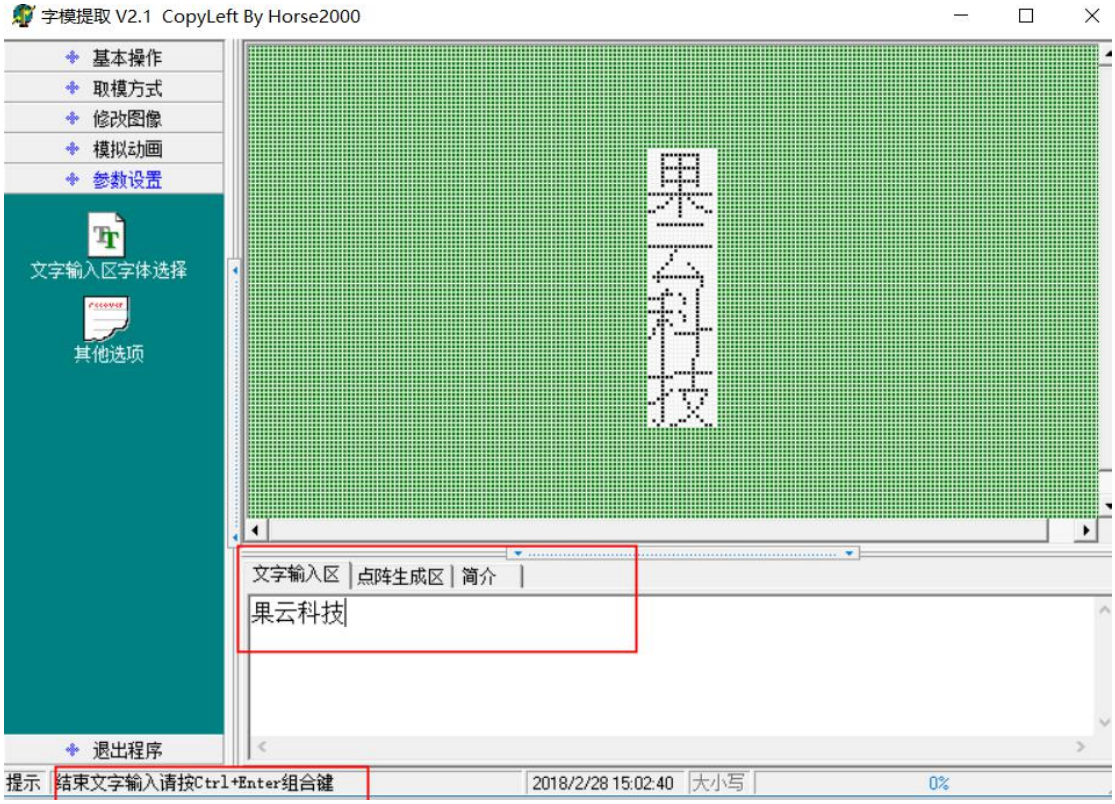
```
void setup() {
  // put your setup code here, to run once:
      lcd.LCD_GPIOInit();
      lcd.LCD_Init();
      lcd.LCD_Clear(BLACK);
      POINT_COLOR=BLUE;
      lcd.Show_Str(40,25,BLUE,BLACK,"Goouuu",16,1);
      lcd.Show_Str(32,50,BLUE,BLACK,"果 云 科 技 ",8,1);

}
```



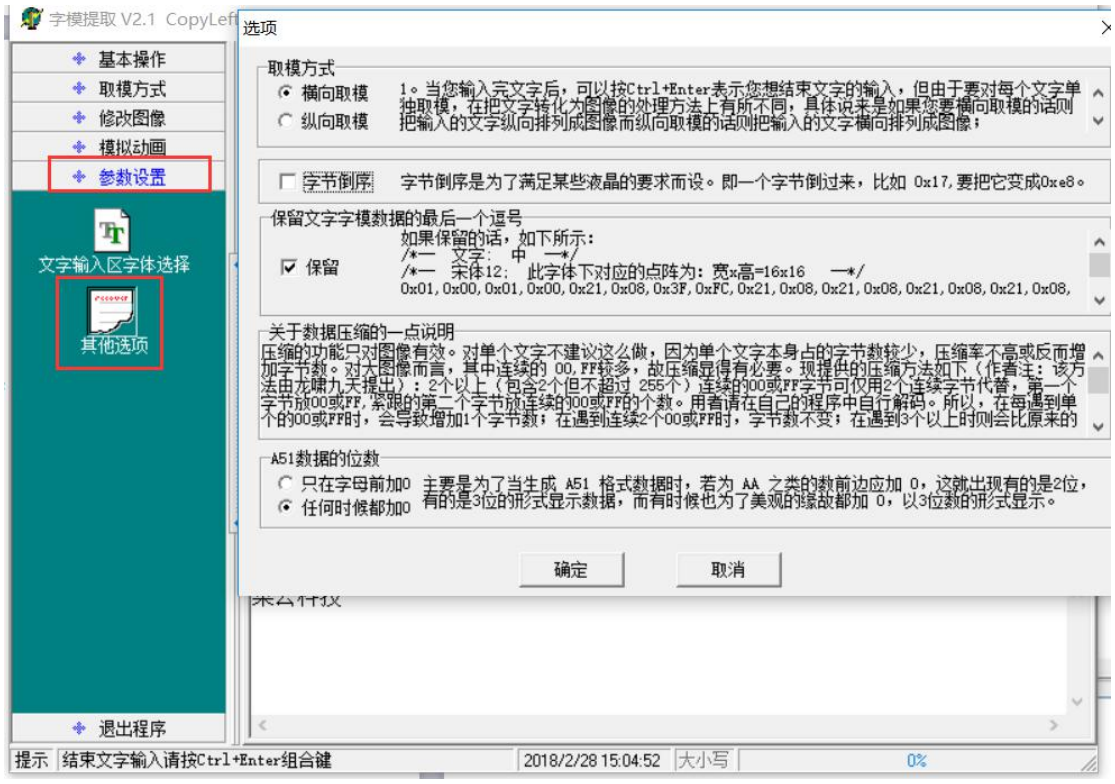
我们的字模都在 FONT.h 里，中文的话需要另外取模，下面我们说下步骤。



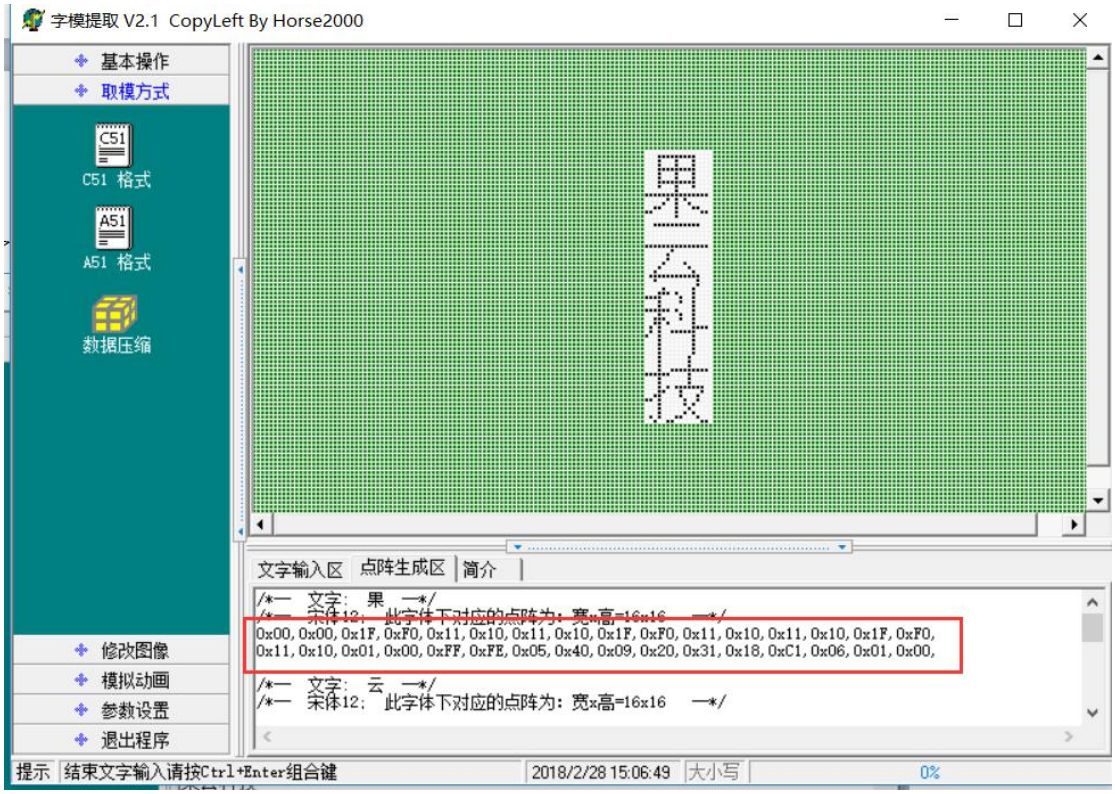
打开取模软件，在文字输入去输入要取模的中文，结束后按 Ctrl+Enter 组合键



然后点参数设置--->其他选项 设置参数如图



好了之后，点取模方式----->C51 格式



把获取到的取模数据分别复制到 font.h 下　对应字体，对应文字的数组里即可

```
//字体取模：宋体常规小四
const typFNT_GB16 tfont16[]=
{
"果", 0x00, 0x00, 0x1F, 0xF0, 0x11, 0x10, 0x11, 0x10, 0x1F, 0xF0, 0x11, 0x10, 0x11, 0x10, 0x1F, 0xF0,
0x11, 0x10, 0x01, 0x00, 0xFF, 0xFE, 0x05, 0x40, 0x09, 0x20, 0x31, 0x18, 0xC1, 0x06, 0x01, 0x00, /*0*/
"云", 0x00, 0x00, 0x3F, 0xF8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFE, 0x02, 0x00,
0x04, 0x00, 0x04, 0x00, 0x08, 0x40, 0x10, 0x20, 0x20, 0x10, 0x7F, 0xF8, 0x20, 0x08, 0x00, 0x08, /*1*/
"科", 0x08, 0x10, 0x1D, 0x10, 0xF0, 0x90, 0x10, 0x90, 0x10, 0x10, 0xFD, 0x10, 0x10, 0x90, 0x38, 0x90,
0x34, 0x10, 0x50, 0x1E, 0x53, 0xF0, 0x90, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, /*2*/
"技", 0x10, 0x20, 0x10, 0x20, 0x10, 0x20, 0x13, 0xFE, 0xFC, 0x20, 0x10, 0x20, 0x10, 0x20, 0x15, 0xFC,
0x18, 0x84, 0x30, 0x88, 0xD0, 0x48, 0x10, 0x50, 0x10, 0x20, 0x10, 0x50, 0x51, 0x88, 0x26, 0x06, /*3*/
};
```

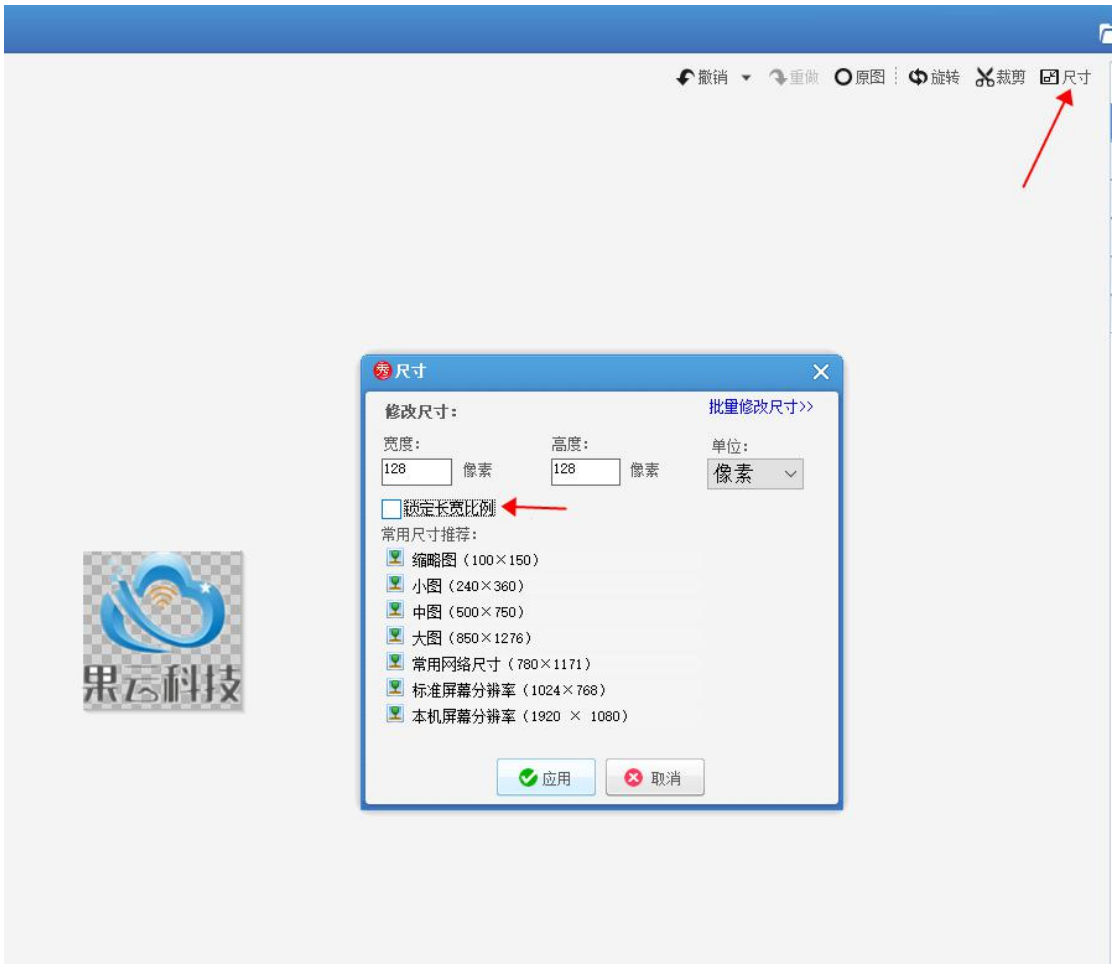
下面继续，我们用 LCD 全屏显示 1 张真彩图片，我们调用图片显示函数

lcd.Gui_Drawbmp16(128,128,0,0,gImage_123);

因为我们这个 1.44 寸屏幕是 128x128 点阵，那么全屏的参数就是图片长度 128　宽度 128，远点坐标 0,0.

gImage_123 这个图片数组怎么取模，我们看下一步骤

1.用美图秀秀打开目标图片，点击设置尺寸，把尺寸锁定去掉，强制改图片尺寸为 128x128


撤销
重做
原图
旋转
裁剪
尺寸
尺寸
修改尺寸：
批量修改尺寸>>
宽度：
高度：
单位：
128
像素
128
像素
像素
锁定长宽比例
常用尺寸推荐：
缩略图（100×150）
小图（240×360）
中图（500×750）
大图（850×1276）
常用网络尺寸（780×1171）
标准屏幕分辨率（1024×768）
本机屏幕分辨率（1920 × 1080）
应用
取消
果云科技

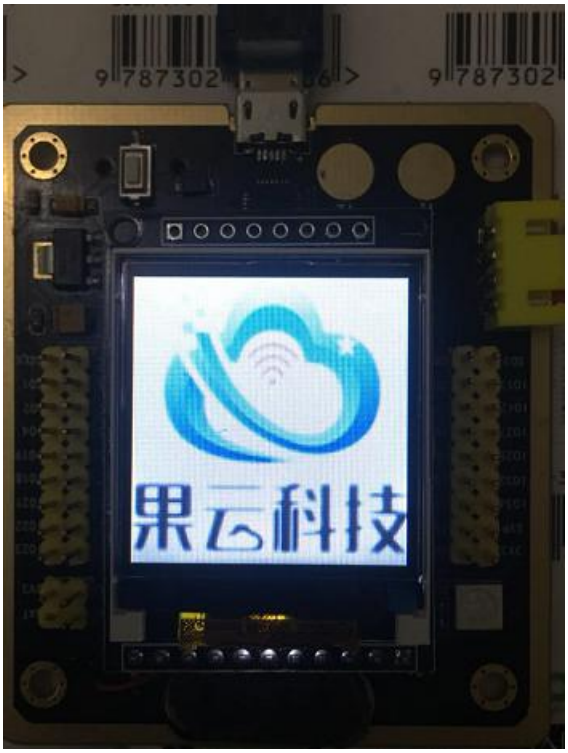
保存为 jpg 格式


保存与分享
分享此图片
果云科技
保存路径
桌面
覆盖原
自定
e:\项目文档\果云科技\logo\
更改
test 副本
.jpg
画质 100
保 存
另存为
美图秀秀iPhone版：帮你快速美化照片，轻松搞定拼图！

用 lcd 取模软件打开图片，参数按照下面的图片来

确认无误后点击保存，把原来的数组替换掉


```
#ifndef __ICON_H
#define __ICON_H
const unsigned char gImage_123[32768] = { /* 0X00,0X10,0X80,0X00,0X80,0X00,0X01,0X1B, */
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,
```



十三、实验 9：ESP32 DAC 驱动扬声器播放 pcm 音乐

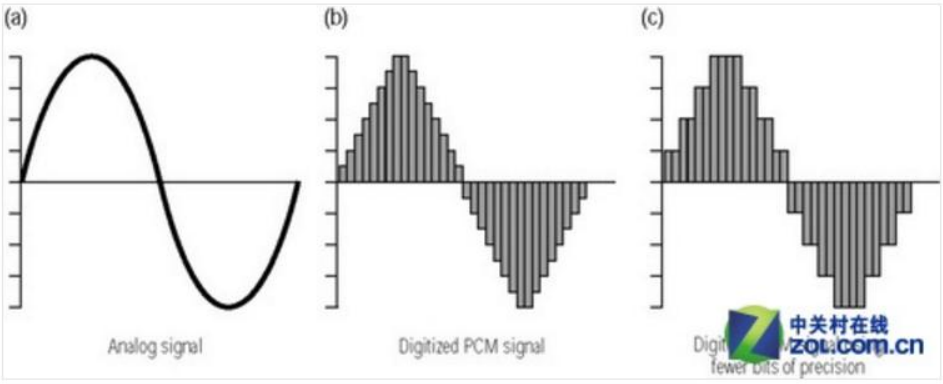
ESP32 有 2 个 8-bit DAC 通道，将 2 个数字信号分别转化为 2 个模拟电压信号输出。DAC 电路由内置电阻串 和 1 个缓冲器组成。这 2 个 DAC 可以作为参考电压使用，也可以作为其它电路的电源使用。这是 2 个独立的 DAC。

这里为什么用到 DAC 我们就要来看下 PCM 编码了，什么是 PCM 编码？（以下是百度收集资料）

PCM 中文称脉冲编码调制(Pulse Code Modulation)，是 70 年代末发展起来的，记录媒体之一的 CD，在 80 年代初由飞利浦和索尼公司共同推出。脉码调制的音频格式也被 DVD-A 所采用，它支持立体声和 5.1 环绕声，1999 年由 DVD 讨论会发布和推出的。脉冲编码调制的比特率，从 14-bit 发展到 16-bit、18-bit、20-bit 直到 24-bit；采样频率从 44.1kHz 发展到 192kHz。PCM 脉码调制这项技术可以改善和提高的方面则越来越来小。只是简单的增加 PCM 脉码调制比特率和采样率，不能根本的改善它的根本问题。其原因是 PCM 的主要问题在于：

（1）任何脉冲编码调制数字音频系统需要在其输入端设置急剧升降的滤波器，仅让 20Hz-22.05kHz 的频率通过（高端 22.05kHz 是由于 CD44.1kHz 的一半频率而确定）。

（2）在录音时采用多级或者串联抽选的数字滤波器（减低采样频率），在重放时采用多级的内插的数字滤波器（提高采样频率），为了控制小信号在编码时的失真，两者又都需要加入重复定量噪声。这样就限制了 PCM 技术在音频还原时的保真度。



码率越高的PCM录音就越接近模拟信号的圆滑正弦波

对于我们最常说的“无损音频”来说，一般都是指传统 CD 格式中的 16bit/44.1kHz 采样率的文件格式，而知所以称为无损压缩，也是因为其包含了 20Hz-22.05kHz 这个完全覆盖人耳可闻范围的频响频率而得名，当然现在的各种 PCM 格式编码高码率文件已经层出不穷非常常见，但是就像上文中所说的，高码率并不能有效地提升 PCM 编码采样率的频响范围，而只能增加其采样点来得到更加类似模拟录音的平滑波形。

也正因为几乎所有的有损压缩格式都是从 WAV 格式压缩、转换而来，其实内部的编码依然是 PCM，所以曾经很多 MP3 设备并不支持 FLAC、APE、AAC 等等格式，是因为它们不支持这些文件的解压缩，但是从没有一款播放器不支持 WAV 格式，因为 WAV 格式本身，就等于 PCM 码流。所以我们用 DAC 可以播放 wav 和 pcm 格式的文件。

我们顺便说下扬声器和喇叭的区别：
扬声器就是通常我们说的音箱。可以播放各种声音，例如音乐。
而蜂鸣器则是单纯只发出一种声音的报警装置。它会通过声音的长短来报告错误

我们先看下扬声器的驱动--SPEAKER 类的库函数
里面驱动方式包括 2 种：DAC 方式 和 PWM 方式，我们只讲 DAC 方式用到的函数

```
void begin();   //初始化
void mute();   //静音
void write(uint8_t value);//DAC 通道写 8 位数据
void setVolume(uint8_t volume);//设置音量 1-10
void playMusic(const uint8_t* music_data, uint16_t sample_rate);//播放 pcm 格式数据，并设定采样率
```

程序代码很简单，播放我们预先取好的 pcm 音频数据。

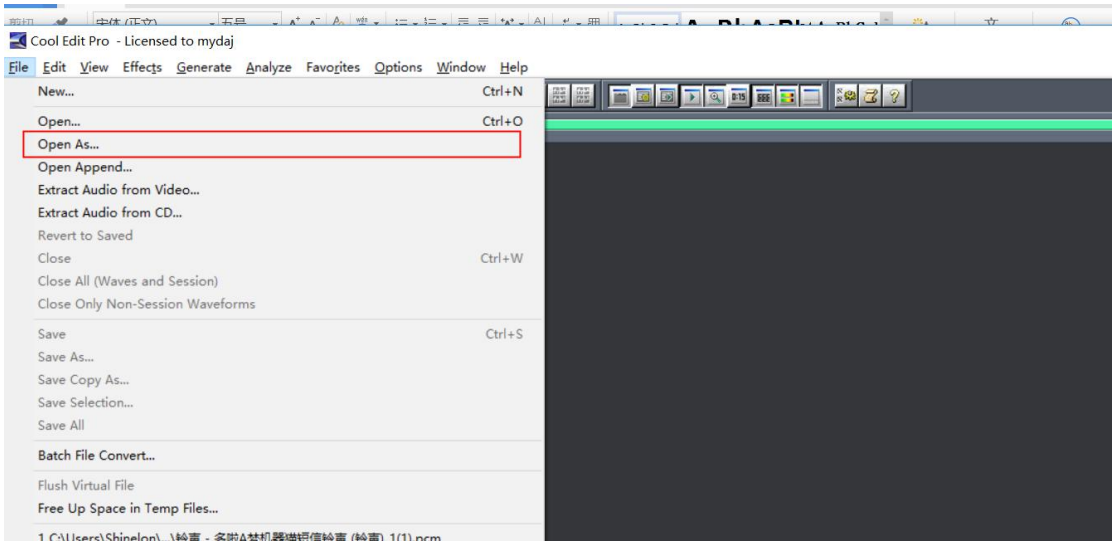
```
#include "Speaker.h"
#include "music_8bit.h"
SPEAKER Speaker;

void setup() {
   // put your setup code here, to run once:
     Speaker.begin();
     Speaker.setVolume(10);
     Speaker.playMusic(data, 22500);
}

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

我们这里讲一下这个 pcm 数据怎么提取来的。

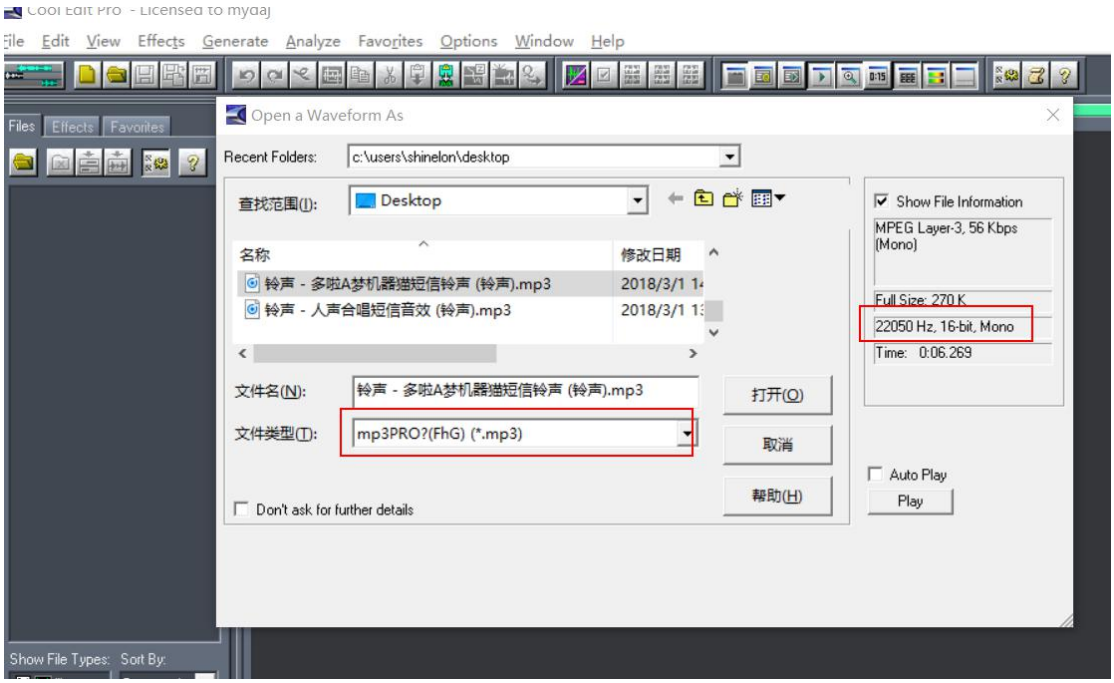
1.我们要安装一个音频处理软件 Cool Edit Pro 。

2.准备要用的 mp3 文件

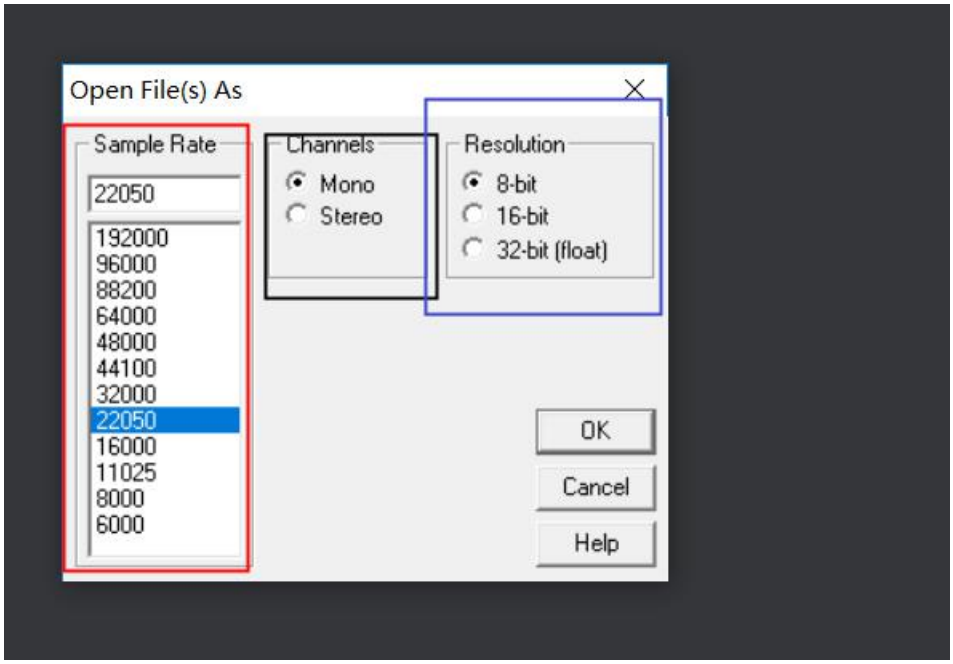
3.打开 Cool Edit Pro 软件，点击 open as



选择要打开的目标 mp3 文件

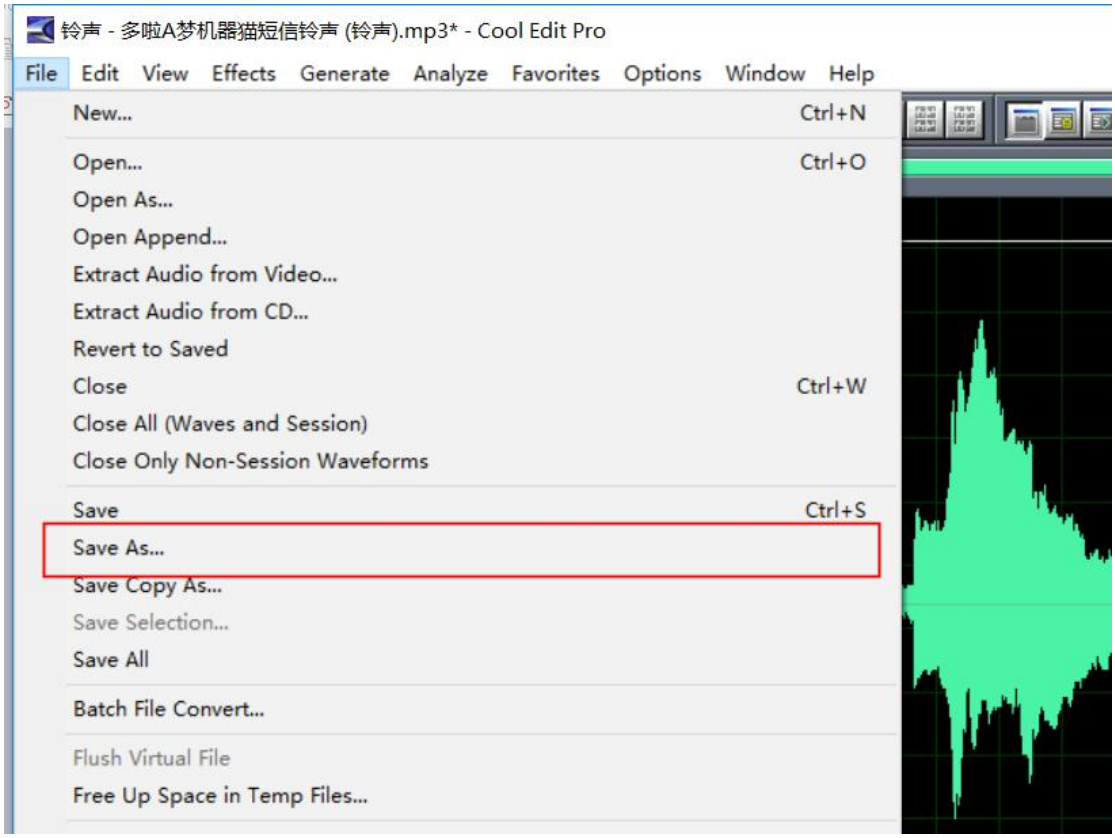


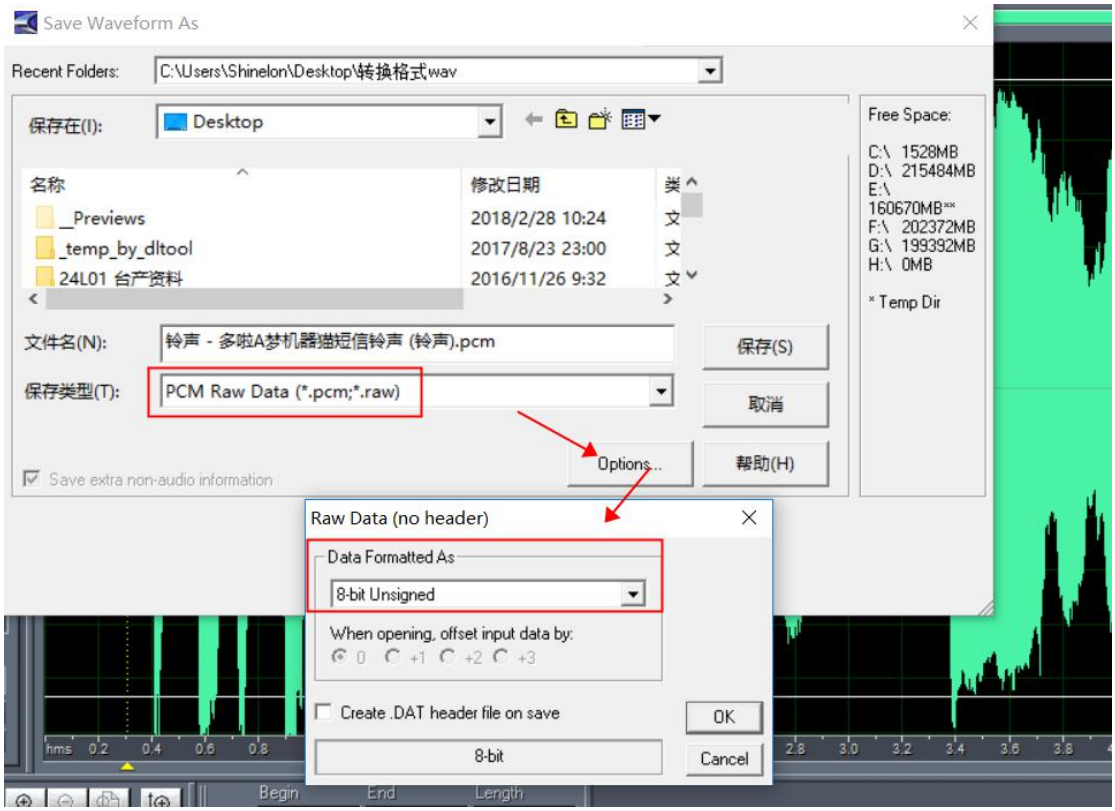
4.把文件打开并修改为采样率 22050，单声道，8 位数据格式的文件，好了点击 ok



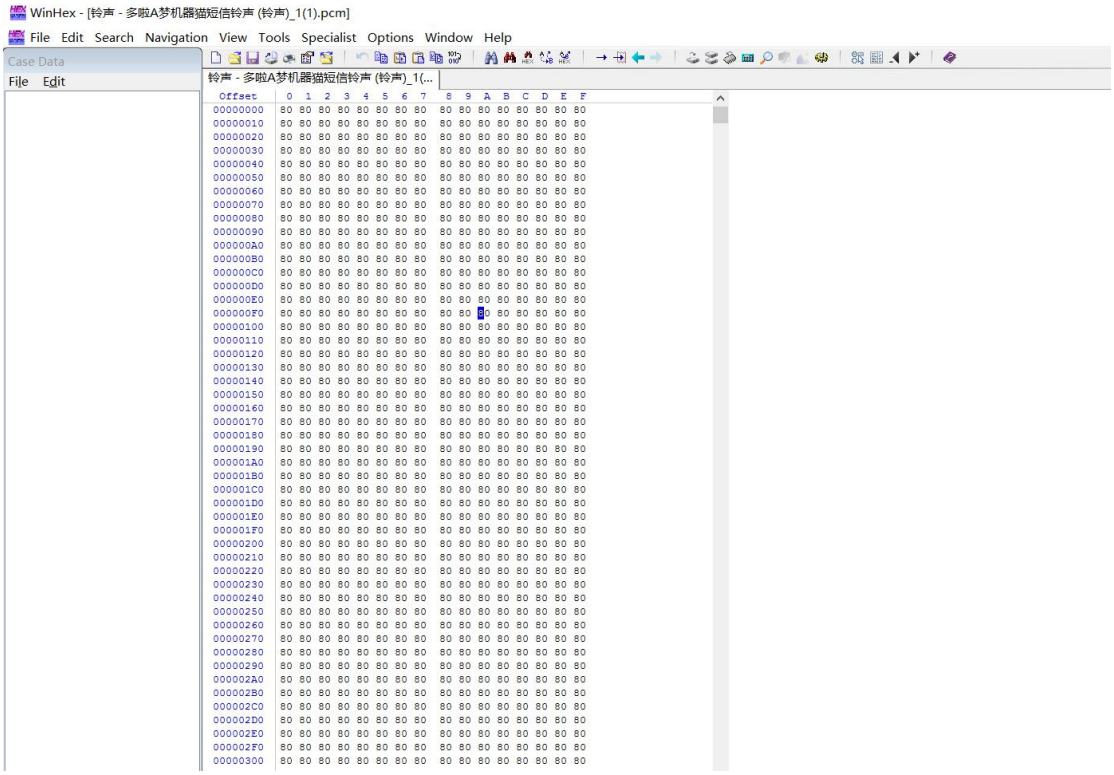
为什么要这么做，因为大多数 mp3 文件是双声道，32 位的数据，我们要强制转换为单声道的 8 位格式文件，才能给 8 位 DAC 输出播放。

5.点击 Save as ， 把文件另存为 pcm 格式的文件

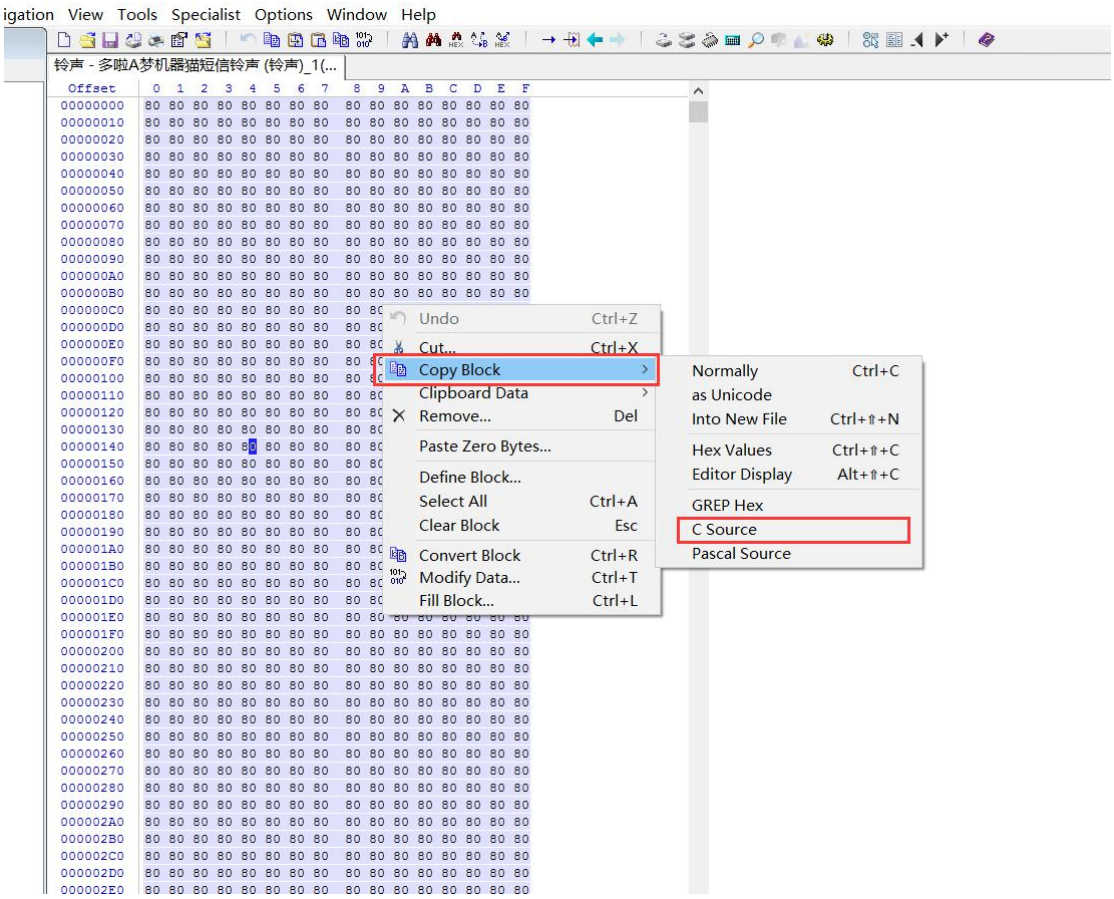




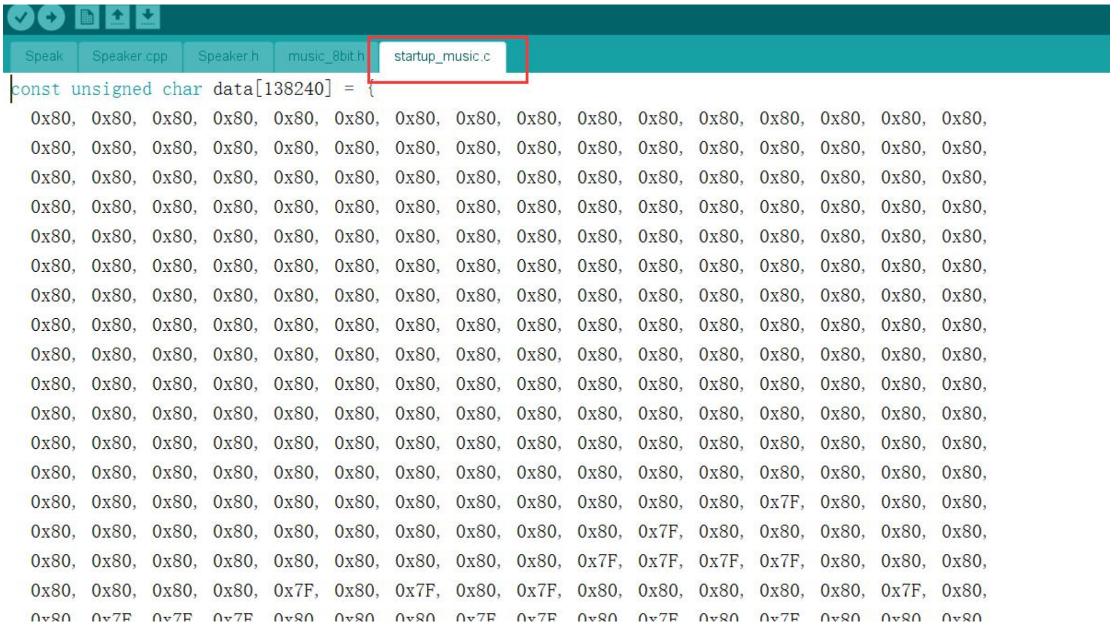
6. 转好后，我们用 winhex 工具来打开，这个 pcm 格式文件，可以直接看到音频的 8 位数据，我们全部复制出来。



7.全选后把数据复制为 c 语言数据的形式。



8.把数据保存到下图文件即可。




```
const unsigned char data[138240] = {
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7F, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7F, 0x80, 0x80, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7F, 0x7F, 0x7F, 0x7F, 0x80, 0x80, 0x80,
  0x80, 0x80, 0x80, 0x80, 0x7F, 0x80, 0x7F, 0x80, 0x7F, 0x80, 0x80, 0x80, 0x80, 0x80, 0x7F, 0x80,
  0x80, 0x7F, 0x7F, 0x7F, 0x80, 0x80, 0x80, 0x7F, 0x7F, 0x80, 0x7F, 0x80, 0x7F, 0x80, 0x80, 0x80
```

十四、实验 10：ESP32 读写 MicroSD 卡操作

开发板 SD 卡接口：

GPIO5:---SD_DAT3

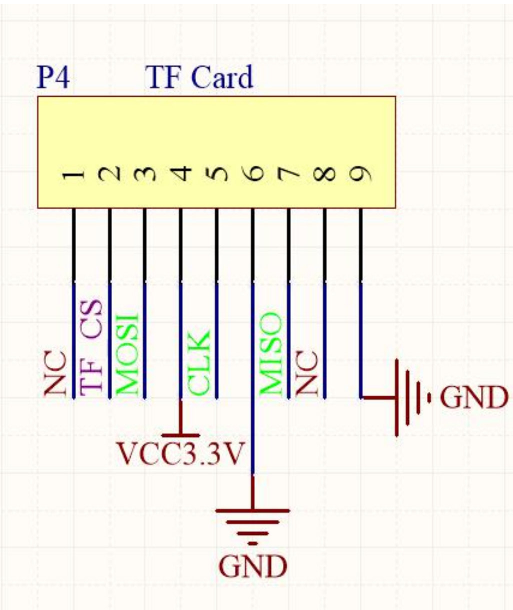
GPIO18:---SD_CLK

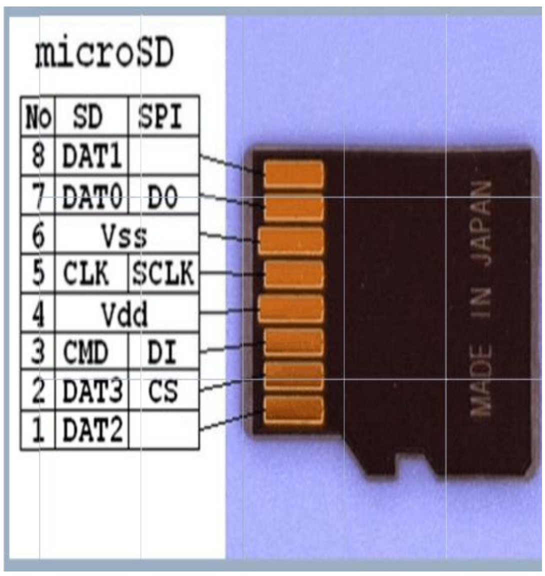
GPIO19:---SD_DAT0

GPIO23:---SD_CMD

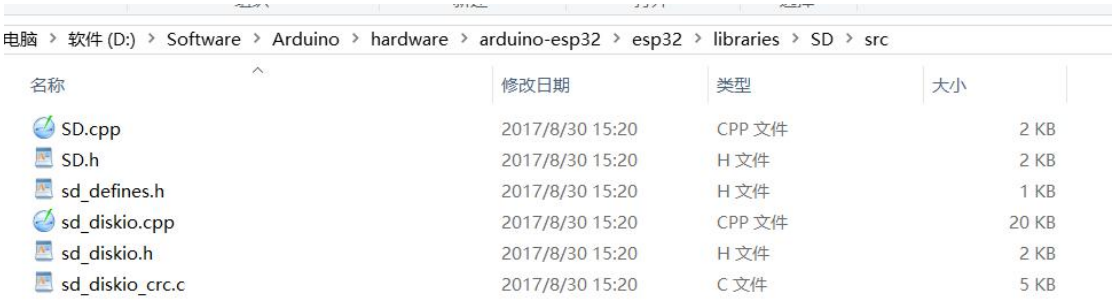
3.3V:---SD_VCC

GND:---SD_GND





我们在 ESP32 库可以找到 SD 卡的底层驱动，找到关键的几个库函数



SDFS(FSImplPtr impl); //SD 类的构造函数

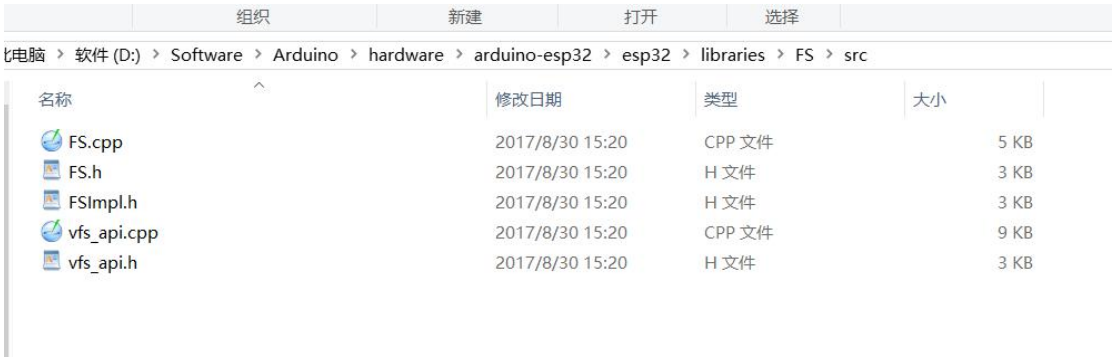
bool begin(uint8_t ssPin=SS, SPIClass &spi=SPI, uint32_t frequency=4000000, const char * mountpoint="/sd"); //初始化 SD 卡，参数可以设置所在 SPI 的的引脚和频率

void end(); //结束 SD 卡操作

sdcard_type_t cardType(); //识别 SD 卡类型，判断是 MMC，SD，SDHC，UNKNOWN，或者非 SD 卡

uint64_t cardSize();//获取 SD 卡的容量大小

有了 SD 卡的常规操作后，我们看下文件系统的库函数，文件系统顾名思义，file system，简称 fs。



FS 类的方法

File open(const String& path, const char* mode = FILE_READ); //打开指定的某个文件

bool exists(const String& path); //判断某个文件是否存在

bool remove(const String& path); //移除某个文件

bool rename(const String& pathFrom, const String& pathTo); //重命名某个文件

bool mkdir(const String &path); //创建指定目录

bool rmdir(const String &path); //移除指定目录

我们看下读写 SD 卡测试的代码，这个程序对 FS 类下的函数进行重写，因为原来的 fs 下的方法没有添加串口输出信息，不方便我们观察现象。

```
#include "FS.h"
#include "SD.h"
#include "SPI.h"
#include "Speaker.h"
SPEAKER Speaker;
/*----------------------------------------------------
        列出指定目录下的文件
----------------------------------------------------*/
void listDir(fs::FS &fs, const char * dirname, uint8_t levels){
    Serial.printf("Listing directory: %s\n", dirname);

    File root = fs.open(dirname);
    if(!root){
        Serial.println("Failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println("Not a directory");
        return;
    }

    File file = root.openNextFile();
    while(file){
        if(file.isDirectory()){
            Serial.print("    DIR : ");
            Serial.println(file.name());
            if(levels){
                listDir(fs, file.name(), levels -1);
            }
        } else {
            Serial.print("    FILE: ");
            Serial.print(file.name());
            Serial.print("    SIZE: ");
            Serial.println(file.size());
        }
        file = root.openNextFile();
    }
}
```

```
/*----------------------------------------------------
                    创建目录
----------------------------------------------------*/

void createDir(fs::FS &fs, const char * path){
      Serial.printf("Creating Dir: %s\n", path);
      if(fs.mkdir(path)){
            Serial.println("Dir created");
      } else {
            Serial.println("mkdir failed");
      }
}
/*----------------------------------------------------
                    移除目录
----------------------------------------------------*/

void removeDir(fs::FS &fs, const char * path){
      Serial.printf("Removing Dir: %s\n", path);
      if(fs.rmdir(path)){
            Serial.println("Dir removed");
      } else {
            Serial.println("rmdir failed");
      }
}
/*----------------------------------------------------
            读取指定文件的内容
----------------------------------------------------*/

void readFile(fs::FS &fs, const char * path){
      uint8_t buf[2];
      uint16_t delay_interval = ((uint32_t)1000000/25000);
      Serial.printf("Reading file: %s\n", path);

      File file = fs.open(path);
      if(!file){
            Serial.println("Failed to open file for reading");
            return;
      }

      Serial.print("Read from file: ");
      while(file.available()){
          buf[0]=file.read();
                dacWrite(SPEAKER_PIN,buf[0]);
                delayMicroseconds(delay_interval-20);
```

```
    }

    file.close();
}
/*------------------------------------------------------
      写入指定内容到指定文件
------------------------------------------------------*/

void writeFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file){
        Serial.println("Failed to open file for writing");
        return;
    }
    if(file.print(message)){
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
    file.close();
}
/*------------------------------------------------------
写入指定内容到指定文件，如果这个文件不存在，则创建一个新的指定名字的文件，再写入内容。
------------------------------------------------------*/

void appendFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file){
        Serial.println("Failed to open file for appending");
        return;
    }
    if(file.print(message)){
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
    file.close();
}
```

```
/*----------------------------------------------------
                    重命名文件
----------------------------------------------------*/

void renameFile(fs::FS &fs, const char * path1, const char * path2){
    Serial.printf("Renaming file %s to %s\n", path1, path2);
    if (fs.rename(path1, path2)) {
        Serial.println("File renamed");
    } else {
        Serial.println("Rename failed");
    }
}
/*----------------------------------------------------
                    删除指定文件
----------------------------------------------------*/

void deleteFile(fs::FS &fs, const char * path){
    Serial.printf("Deleting file: %s\n", path);
    if(fs.remove(path)){
        Serial.println("File deleted");
    } else {
        Serial.println("Delete failed");
    }
}

/*----------------------------------------------------
                    读写文件测试
----------------------------------------------------*/
void testFileIO(fs::FS &fs, const char * path){
    File file = fs.open(path);
    static uint8_t buf[512];
    size_t len = 0;
    uint32_t start = millis();
    uint32_t end = start;
    if(file){
        len = file.size();
        size_t flen = len;
        start = millis();
        while(len){
```

```
            size_t toRead = len;
            if(toRead > 512){
                toRead = 512;
            }
            file.read(buf, toRead);
            len -= toRead;
        }
        end = millis() - start;
        Serial.printf("%u bytes read for %u ms\n", flen, end);
        file.close();
    } else {
        Serial.println("Failed to open file for reading");
    }


    file = fs.open(path, FILE_WRITE);
    if(!file){
        Serial.println("Failed to open file for writing");
        return;
    }

    size_t i;
    start = millis();
    for(i=0; i<2048; i++){
        file.write(buf, 512);
    }
    end = millis() - start;
    Serial.printf("%u bytes written for %u ms\n", 2048 * 512, end);
    file.close();
}


void setup(){
    Serial.begin(115200);
    if(!SD.begin()){            //初始化 SD 卡
        Serial.println("Card Mount Failed");
        return;
    }

    uint8_t cardType = SD.cardType();    //读取 SD 卡类型
```

```
    //判断 SD 卡类型
      if(cardType == CARD_NONE){
            Serial.println("No SD card attached");
            return;
      }

      Serial.print("SD Card Type: ");
      if(cardType == CARD_MMC){
            Serial.println("MMC");
      } else if(cardType == CARD_SD){
            Serial.println("SDSC");
      } else if(cardType == CARD_SDHC){
            Serial.println("SDHC");
      } else {
            Serial.println("UNKNOWN");
      }
    //读取 SD 卡尺寸
      uint64_t cardSize = SD.cardSize() / (1024 * 1024);
      Serial.printf("SD Card Size: %lluMB\n", cardSize);

      listDir(SD, "/", 0);//列出目录
      createDir(SD, "/mydir");//创建 1 个 mydir 的文件夹目录
      listDir(SD, "/", 0);//列出目录
      removeDir(SD, "/mydir");//删除 mydir 的文件夹目录
      listDir(SD, "/", 2);//列出目录
      writeFile(SD, "/hello.txt", "Hello ");//对 hello.txt 这个文件写内容 Hello
      appendFile(SD, "/hello.txt", "World!\n");//检测 hello.txt 这个是否存在，不存在则创建并写
//入内容，如果存在则直接写入内容
      readFile(SD, "/hello.txt");//读取 hello.txt 文件
      deleteFile(SD, "/foo.txt");//删除 foo.txt 文件
      renameFile(SD, "/hello.txt", "/foo.txt");//把 hello.txt 这个文件重命名为 foo.txt
      readFile(SD, "/jqm.pcm");//读取 jqm.pcm 这个文件，并播放

}

void loop(){

}
```

十五、ESP32　WiFi 的使用

ESP32 支持 TCP/IP 协议，完全遵循 802.11 b/g/n/e/i WLAN MAC 协议和 Wi-Fi Direct 标准，支持分布式控制功 能 (DCF) 下的基本服务集 (BSS) STA 和 SoftAP 操作，也支持最新 Wi-Fi P2P 协议的 P2P GO 模式。 通过适当的命令，ESP32 会自动实施主动扫描、被动扫描以及 P2P 发现。通过最小化主机交互来优化有效工作 周期，以实现功耗管理。数据率高达 150 Mbps ,发射功率高达 20.5 dBm 。

STA 模式下的 TCP 通讯实验：
ESP32 作为 Station 模式，连接路由器，然后连接我们的公网测试服务器端，惊醒 tcp 通讯

```
#include<WiFi.h>
#include<WiFiMulti.h>
WiFiMulti WiFiMulti;
WiFiClient client;
#define WIFISSID "yq"
#define Password "goouuu8266"
uint8_t test[6]={0x12,0x34,0x56,0x78,0x0c,0x0b};//6 个字节的测试数据
const uint16_t port = 8266;//主机端口
const char * host = "39.106.163.29"; // ip or dns 远程主机 ip

void Connect_Router(void);//连接到你的路由器
void Connect_Host(void);//连接到远程服务器


void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Connect_Router();//连接到你的路由器



}

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

    Connect_Host();//连接到远程服务器
    // This will send the request to the server
    client.write(test,6);//发送 6 个字节数据
    client.print("Hello_Goouuu!\r\n");//发送字符串
    //read back one line from server
```

```
        String line = client.readStringUntil('\r');读取服务器发来的数据
        Serial.println(line);//打印服务器发来的数据

        client.stop();//关闭端口
        Serial.println("closing connection");
        Serial.println("wait 5 sec...");
        delay(5000);
}


void Connect_Router(void)
{
        // We start by connecting to a WiFi network
        WiFiMulti.addAP(WIFISSID,Password);

        Serial.println();
        Serial.println();
        Serial.print("Wait for WiFi... ");

        while(WiFiMulti.run() != WL_CONNECTED) {
                Serial.print(".");
                delay(500);
        }

        Serial.println("");
        Serial.println("WiFi connected");
        Serial.println("IP address: ");
        Serial.println(WiFi.localIP());
        delay(500);
}

void Connect_Host(void)
{
        if (!client.connect(host, port)) {
                Serial.println("connection failed");
                Serial.println("wait 5 sec...");
                delay(5000);
                return;
        }

}
```

十六、 ESP32 蓝牙的使用