### 2.5.2 FreeRTOS任务构成

**1.任务定义**

FreeRTOS官方给出的任务函数模板如下:

void vATaskFunction(void\*pvParameters)

{

for(; ;){

任务功能实现;

vTaskDelayO;

}

vTaskDelete(NULL);

}

**2.任务控制**

typedef struct tskTaskControlBlock

{

volatile StackType\_t \*pxTopOfStack; //任务堆栈栈顶

#if(portUSING\_MPU\_WRAPPERS ==1)

xMPU\_SETTINGSxMPUSettings; //MPU相关设置

#endif

ListItem\_txStateListltem; //状态列表项

ListItem\_t xEventListItem; //事件列表项

UBaseType\_t uxPriority; //任务优先级

StackType\_t\*pxStack; //任务堆栈起始地址

char pcTaskName[ configMAX\_TASK\_NAME\_LEN]; //任务名字

#if ( portSTACK\_GROWTH >0 )

StackType\_t\*pxEndOfStack; //任务堆栈栈底

#endif

#if ( portCRITICAL\_NESTING\_IN\_TCB == 1 )

UBaseType\_tuxCriticalNesting; //临界区嵌套深度

#endif

#if ( configUSE\_TRACE\_FACILITY== 1 ) //trace或debug用

UBaseType\_t uxTCBNumber;

UBaseType\_t uxTaskNumber;

#endif

#if ( configUSE\_MUTEXES-= 1 )

UBaseType\_t uxBasePriority; //任务基础优先级

UBaseType\_t uxMutexesHeld; //任务获取互斥信号量个数

endif

#if(configUSE\_APPLICATION\_TASK\_TAG==1)//多任务不同任务的状态切换信息

TaskHookFunction\_t pxTaskTag;

#endif

#if( configNUM\_THREAD\_LOCAL\_STORAGE\_POINTERS>0 ) //本地存储

Void \*pvThreadLocalStoragePointers[ configNUM\_THREAD\_LOCAL\_STORAGE\_POINTERS ];

#endif

#if( configGENERATE\_RUN\_TIME\_STATS== 1 )

uint32\_t ulRunTimeCounter; //用来记录任务运行总时间

#endif

#if(configUSE\_NEWLIB\_REENTRANT ==1)

Struct\_reent xNewLib\_reent; //定义一个newlib 结构体变量

#endif

#if(configUSE\_TASK\_NOTIFICATIONS -=1 )//任务通知相关变量

volatile uint32\_t ulNotifiedValue; //任务通知值

volatile uint8\_t ucNotifyState; //任务通知状态

#endif

#if(tskSTATIC\_AND\_DYNAMIC\_ALLOCATION\_POSSIBLE !=0 )

//用来标记任务是动态创建的还是静态创建的，如果是静态创建的此变量就为pdTURE

//如果是动态创建的就为pdFALSE

uint8\_t ucStaticallyAllocated;#endif

#if(INCLUDE\_xTaskAbortDelay == 1 )

uint8\_t ucDelayAborted;

#endif

}tskTCB;

# 第3章 ESP32开发环境

hello\_world\_main源文件如下：

#include <stdio.h> //用到的头文件

#include "sdkconfig.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_spi\_flash.h"

void app\_main(void) //主程序入口

{

printf("Hello world!\n"); //打印Hello world！

/\*打印芯片信息\*/

esp\_chip\_info\_t chip\_info;

esp\_chip\_info(&chip\_info);

printf("This is %s chip with %d CPU cores, WiFi%s%s, ",

CONFIG\_IDF\_TARGET,

chip\_info.cores,

(chip\_info.features & CHIP\_FEATURE\_BT) ? "/BT" : "",

(chip\_info.features & CHIP\_FEATURE\_BLE) ? "/BLE" : "");

printf("silicon revision %d, ", chip\_info.revision);

printf("%dMB %s flash\n", spi\_flash\_get\_chip\_size() / (1024 \* 1024),

(chip\_info.features & CHIP\_FEATURE\_EMB\_FLASH) ? "embedded" : "external");

printf("Free heap: %d\n", esp\_get\_free\_heap\_size());

for (int i = 10; i >= 0; i--) { //延迟倒计时

printf("Restarting in %d seconds...\n", i);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

printf("Restarting now.\n"); //输出信息

fflush(stdout);

esp\_restart(); //重新启动程序

}

### 3.5.3 network模块

**3.示例**

import network

wlan = network.WLAN(network.STA\_IF) #创建站点接口

wlan.active(True) #激活接口

wlan.scan() #扫描接入点AP

wlan.isconnected() #检查站点是否接入AP

wlan.connect('essid', 'password') #连接到AP

wlan.config('mac') #获取接口的MAC地址

wlan.ifconfig() #获取接口的IP/netmask/gw/DNS地址

ap = network.WLAN(network.AP\_IF) #创建AP接口

ap.active(True) #激活接口

ap.config(essid='ESP-AP') #设置AP的ESSID

### 3.5.4 utime模块

示例如下：

import time

time.sleep(1) #睡眠1s

time.sleep\_ms(500) #睡眠500 ms

time.sleep\_us(10) #睡眠10us

start = time.ticks\_ms() #获取ms计数器

delta = time.ticks\_diff(time.ticks\_ms(), start) #计算时间差

# 第4章 基础外设开发

### 4.1.6 GPIO示例程序

1. 基于ESP IDF的VS Code开发环境实现方式一

代码如下：

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "sdkconfig.h"

#define LED 18 //定义输出引脚

void LED\_Task(void \*pvParameter)

{

gpio\_pad\_select\_gpio(LED); //选择芯片引脚

gpio\_set\_direction(LED, GPIO\_MODE\_OUTPUT); //设置该引脚为输出模式

while(1) {

gpio\_set\_level(LED, 0); //电平为低

vTaskDelay(1000 / portTICK\_PERIOD\_MS); //延迟1S

gpio\_set\_level(LED, 1); //电平为高

vTaskDelay(1000 / portTICK\_PERIOD\_MS); //延迟1S

}

}

void app\_main() //主函数

{

xTaskCreate(&LED\_Task,"LED\_Task",configMINIMAL\_STACK\_SIZE,NULL,5,NULL);//新建一个任务

}

2. 基于ESP IDF的VS Code开发环境实现方式二

代码如下：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/gpio.h"

#define GPIO\_OUTPUT\_IO\_0 18

#define GPIO\_OUTPUT\_PIN\_SEL 1ULL<<GPIO\_OUTPUT\_IO\_0

void app\_main(void)

{

gpio\_config\_t io\_conf; //定义GPIO结构体

io\_conf.intr\_type = GPIO\_PIN\_INTR\_DISABLE; //禁用中断

io\_conf.mode = GPIO\_MODE\_OUTPUT; //设置为输出模式

io\_conf.pin\_bit\_mask = GPIO\_OUTPUT\_PIN\_SEL; //比特掩码GPIO18

io\_conf.pull\_down\_en = 0; //禁用下拉模式

io\_conf.pull\_up\_en = 0; //禁用上拉模式

gpio\_config(&io\_conf); //使用上面的参数配置GPIO

int cnt = 0; //计数变量

while(1) {

printf("cnt: %d\n", cnt++); //串口可以看到计数输出

vTaskDelay(1000 / portTICK\_RATE\_MS); //延时1S

gpio\_set\_level(GPIO\_OUTPUT\_IO\_0, cnt % 2); //求余设置电平

}

}

3. Arduino开发环境

代码如下：

#define LED 18 //定义输出引脚

void setup() {

Serial.begin(115200); //设置串口监视器波特率

pinMode(LED, OUTPUT); //设置引脚状态为输出

}

void loop() { //主函数

digitalWrite(LED, 0); //电平为低

delay(1000); //延迟1s

digitalWrite(LED, 1); //电平为高

delay(1000); //延迟1s

}

4. MicroPython开发环境实现

代码如下：

import time

from machine import Pin

led=Pin(18,Pin.OUT) #设置输出引脚

while True:

led.value(0) #电平为低

time.sleep(1) #延时1s

led.value(1) #电平为高

time.sleep(1) #延时1s



### 4.2.4 中断示例程序

1. 基于ESP IDF的VS Code开发环境实现方式

代码如下：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/gpio.h"

#define GPIO\_OUTPUT\_IO\_0 18

#define GPIO\_OUTPUT\_PIN\_SEL (1ULL<<GPIO\_OUTPUT\_IO\_0)

#define GPIO\_INPUT\_IO\_0 4

#define GPIO\_INPUT\_PIN\_SEL (1ULL<<GPIO\_INPUT\_IO\_0)

#define ESP\_INTR\_FLAG\_DEFAULT 0

static xQueueHandle gpio\_evt\_queue = NULL; //FreeRTOS的队列句柄

static void IRAM\_ATTR gpio\_isr\_handler(void\* arg) //函数gpio\_isr\_handler的调用规范

{

uint32\_t gpio\_num = (uint32\_t) arg;

xQueueSendFromISR(gpio\_evt\_queue, &gpio\_num, NULL);

}

static void gpio\_task\_example(void\* arg) //构建任务

{

uint32\_t io\_num;

for(;;) {

if(xQueueReceive(gpio\_evt\_queue, &io\_num, portMAX\_DELAY)) { //接收队列

printf("GPIO[%d] intr, val: %d\n", io\_num, gpio\_get\_level(io\_num));

}

}

}

void app\_main(void) //主函数

{

gpio\_config\_t io\_conf; //定义结构体

io\_conf.intr\_type = GPIO\_PIN\_INTR\_DISABLE; //禁用中断

io\_conf.mode = GPIO\_MODE\_OUTPUT; //设置输出模式

io\_conf.pin\_bit\_mask = GPIO\_OUTPUT\_PIN\_SEL; //GPIO18的比特掩码

io\_conf.pull\_down\_en = 0; //禁用下拉模式

io\_conf.pull\_up\_en = 0; //禁用上拉模式

gpio\_config(&io\_conf); //使用以上参数初始化GPIO

io\_conf.intr\_type = GPIO\_PIN\_INTR\_POSEDGE; //上升沿触发中断

io\_conf.pin\_bit\_mask = GPIO\_INPUT\_PIN\_SEL; //GPIO4的比特掩码

io\_conf.mode = GPIO\_MODE\_INPUT; //设置输入模式

io\_conf.pull\_up\_en = 1; //使能上拉模式

gpio\_config(&io\_conf); //使用以上参数配置

gpio\_evt\_queue = xQueueCreate(10, sizeof(uint32\_t)); //创建队列处理中断

xTaskCreate(gpio\_task\_example, "gpio\_task\_example", 2048, NULL, 10, NULL); //开启任务

gpio\_install\_isr\_service(ESP\_INTR\_FLAG\_DEFAULT); //安装GPIO中断服务

gpio\_isr\_handler\_add(GPIO\_INPUT\_IO\_0, gpio\_isr\_handler, (void\*) GPIO\_INPUT\_IO\_0);

//GPIO引脚挂钩ISR处理程序

int cnt = 0;

while(1) {

printf("cnt: %d\n", cnt++); //打印计数

vTaskDelay(1000 / portTICK\_RATE\_MS); //延时1S

gpio\_set\_level(GPIO\_OUTPUT\_IO\_0, cnt % 4); //每隔4个计数，打印一次中断

//gpio\_set\_level(GPIO\_OUTPUT\_IO\_1, cnt % 2);

}

}

读者可以使用gpio\_set\_intr\_type(GPIO\_INPUT\_IO\_0, GPIO\_INTR\_ANYEDGE)语句，改变中断类型，看看有什么中断效果。

2. Arduino开发环境实现

代码如下：

void callBack(void)

{

Serial.printf("GPIO 4 Interrupted\n");

}

void setup()

{

Serial.begin(115200); //设置串口监视器波特率

Serial.println();

pinMode(18, OUTPUT); //GPIO18为输出模式

pinMode(4, INPUT); //GPIO4为输入模式

attachInterrupt(4, callBack, RISING); //上升沿触发中断

}

int cnt = 0;

void loop() //主函数

{

Serial.printf("cnt: %d\n", cnt++); //打印计数

digitalWrite(18, cnt % 4); //每隔4个计数，打印一次中断

delay(1000); //延时1S

//detachInterrupt(4); //关闭中断

}

3.MicroPython开发环境实现

代码如下：

import time

import machine

from machine import Pin

GPIO\_OUTPUT=Pin(18,Pin.OUT)

GPIO\_INPUT=Pin(4,Pin.IN, Pin.PULL\_UP)

cnt=0 #定义计数

interrupt = 0

interruptsCounter = 0 #计算中断事件次数

def callback(pin): #定义回调函数

global interrupt, interruptsCounter #声明为全局变量

interrupt = 1

interruptsCounter = interruptsCounter+1

GPIO\_INPUT.irq(trigger=Pin.IRQ\_RISING, handler=callback)

while True:

GPIO\_OUTPUT.value(cnt%4)

time.sleep(1)

cnt=cnt+1

if interrupt:

#state = machine.disable\_irq() #禁用计数器

interrupt = 0

#machine.enable\_irq(state) #重新启动计数器

print("Interrupt has occurred: " + str(interruptsCounter))

### 4.3.3 ADC示例程序

代码如下：

#include <stdio.h>

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/uart.h"

#include <driver/adc.h>

void app\_main()

{

adc1\_config\_width(ADC\_WIDTH\_BIT\_12); //设置位宽

adc1\_config\_channel\_atten(ADC1\_CHANNEL\_0, ADC\_ATTEN\_DB\_0);//设置衰减

while(1){

int val = hall\_sensor\_read(); //读取霍尔传感器的值

printf("The hall val: %d\n",val); //串口打印值

vTaskDelay(100); //延时

}

}

2. 基于ESP IDF的VS Code开发环境ADC2

/\*ADC2示例\*/

#include <stdio.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "driver/adc.h"

#include "esp\_adc\_cal.h"

#include <esp\_log.h>

//ADC初始化

//ADC\_ATTEN\_DB\_0:表示参考电压为1.1V

//ADC\_ATTEN\_DB\_2\_5:表示参考电压为1.5V

//ADC\_ATTEN\_DB\_6:表示参考电压为2.2V

//ADC\_ATTEN\_DB\_11:表示参考电压为3.9V

void adc\_Init()

{

adc2\_config\_channel\_atten(ADC2\_CHANNEL\_0,ADC\_ATTEN\_DB\_6);

//ADC2设置通道0和2.2V参考电压

}

void app\_main() //用户函数入口，相当于main()函数

{

int read\_raw;

printf("APP Start......\n");

adc\_Init();

while(1){

adc2\_get\_raw(ADC2\_CHANNEL\_0, ADC\_WIDTH\_12Bit, &read\_raw);

//采集ADC

//ADC的结果转换成电压

//参考电压是2.2V，所以是2200mV，12位分辨率，所以是4096

printf("ADV\_Value: %d Voltage: %d mV \r\n", read\_raw,(read\_raw\*2200)/4096);

vTaskDelay(1000 / portTICK\_RATE\_MS); //延迟

}

}

3. Arduino开发环境实现

代码如下：

#include "driver/gpio.h"

#include "driver/adc.h"

#include "esp\_adc\_cal.h"

void setup() {

Serial.begin(115200); //设置串口监视器波特率

adc2\_config\_channel\_atten(ADC2\_CHANNEL\_0,ADC\_ATTEN\_DB\_6);

//ADC2设置通道0和2.2V参考电压

}

void loop() {

int read\_raw;

Serial.printf("APP Start......\n");

adc2\_get\_raw(ADC2\_CHANNEL\_0, ADC\_WIDTH\_12Bit, &read\_raw);

//ADC的结果转换成电压，参考电压是2.2V，所以是2200mV，12位分辨率，所以是4096

Serial.printf("ADV\_Value: %d Voltage: %d mV \r\n", read\_raw, (read\_raw\*2200)/4096);

delay(1000); //延迟

}

4. MicroPython开发环境实现

代码如下：

from machine import ADC, Pin

from time import sleep\_ms

adc = ADC(Pin(32)) #在引脚32实例化ADC（MicroPython只支持在32~39号引脚对ADC实例化）

adc.atten(ADC.ATTN\_6DB)

adc.width(ADC.WIDTH\_12BIT) #12位分辨率，范围0~4095

while 1:

adc.read()

sleep\_ms(1000) #延时

print("ADC Value: " + str(adc.read()\*2200/4096))

### 4.4.3 DAC示例程序

本部分包括基于ESP IDF的VS Code、Arduino和MicroPython环境的三种代码实现。

1. 基于ESP IDF的VS Code开发环境的DAC控制LED

本示例将GPIO26（也就是DAC2）接到LED上，通过DAC输出变化的电压值，实现DAC完成LED的亮灭，并将DAC的信息打印在串口，电路如图4-12所示。



图4-12 DAC电路

#include <stdio.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/gpio.h"

#include "driver/adc.h"

#include "driver/dac.h"

#include "esp\_system.h"

#include "esp\_adc\_cal.h"

void app\_main(void)

{

uint8\_t output\_data=0; //输出数据变量

esp\_err\_t r; //判断结果变量

gpio\_num\_t dac\_gpio\_num; //引脚变量

r = dac\_pad\_get\_io\_num( DAC\_CHANNEL\_2, &dac\_gpio\_num ); //获取引脚信息

assert( r == ESP\_OK ); //正确与否

printf("DAC channel %d @ GPIO %d.\n",DAC\_CHANNEL\_2, dac\_gpio\_num );

dac\_output\_enable( DAC\_CHANNEL\_2 ); //DAC输出使能

vTaskDelay(2 \* portTICK\_PERIOD\_MS); //延时

printf("start conversion.\n");

while(1) {

dac\_output\_voltage( DAC\_CHANNEL\_2, output\_data++ ); //输出数据

printf("output\_data %d @ GPIO %d.\n",output\_data, dac\_gpio\_num );

vTaskDelay(10);

}

}

2. Arduino开发环境实现

代码如下：

#include "driver/gpio.h"

#include "driver/adc.h"

#include "driver/dac.h"

#include "esp\_system.h"

#include "esp\_adc\_cal.h"

uint8\_t output\_data=0; //输出数据变量

esp\_err\_t r; //判断结果变量

gpio\_num\_t dac\_gpio\_num; //引脚变量

void setup() {

Serial.begin(115200); //设置串口监视器波特率

}

void loop()

{

r = dac\_pad\_get\_io\_num( DAC\_CHANNEL\_2, &dac\_gpio\_num ); //获取引脚信息

assert( r == ESP\_OK ); //正确与否

Serial.printf("DAC channel %d @ GPIO %d.\n",DAC\_CHANNEL\_2, dac\_gpio\_num );

dac\_output\_enable( DAC\_CHANNEL\_2 ); //DAC输出使能

delay(2 \* portTICK\_PERIOD\_MS); //延时

Serial.printf("start conversion.\n");

dac\_output\_voltage( DAC\_CHANNEL\_2, output\_data++ ); //输出数据

Serial.printf("output\_data %d @ GPIO %d.\n",output\_data, dac\_gpio\_num );

delay(10);

}

3. MicroPython开发环境实现

代码如下：

from machine import DAC, Pin

import utime, math

dac\_pin = 26

dac = DAC(Pin(dac\_pin, Pin.OUT), bits=12)

def pulse(dac, period, gears):

'''

呼吸灯函数参数:

dac {[DAC]}为[DAC对象], period {[type]}为[周期 ms], gears {[type]}为[亮度档位]

'''

for i in range(2 \* gears):

dac.write(int(math.sin(i / gears \* math.pi) \* 127) + 128)

#延时

utime.sleep\_ms(int(period / (2 \* gears)))

print("DAC gpio\_num: " + str(dac\_pin)) #呼吸十次

for i in range(10):

print('第' + str(i + 1) + '次')

pulse(dac, 1000, 50)

4. 基于ESP IDF的VS Code的ADC-DAC变换

本示例完成ADC到DAC的变换，ESP32平台默认使用ADC1\_CHANNEL\_7（GPIO27）和DAC\_CHANNEL\_1（GPIO25），将两个GPIO短路。ESP32-S2系列默认使用ADC1\_CHANNEL\_7（GPIO18）和DAC\_CHANNEL\_1（GPIO17），将两个GPIO短路。程序运行的结果可以看到DAC从0~255变化，ADC从0~4095变化。代码如下：

/\*ADC-DAC示例\*/

#include <stdio.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/gpio.h"

#include "driver/adc.h"

#include "driver/dac.h"

#include "esp\_system.h"

#define DAC\_EXAMPLE\_CHANNEL CONFIG\_EXAMPLE\_DAC\_CHANNEL

#define ADC2\_EXAMPLE\_CHANNEL CONFIG\_EXAMPLE\_ADC2\_CHANNEL

#if CONFIG\_IDF\_TARGET\_ESP32

static const adc\_bits\_width\_t width = ADC\_WIDTH\_BIT\_12;

#elif CONFIG\_IDF\_TARGET\_ESP32S2

static const adc\_bits\_width\_t width = ADC\_WIDTH\_BIT\_13;

#endif

void app\_main(void)

{

uint8\_t output\_data=0; //DAC数据变量

int read\_raw; //ADC数据变量

esp\_err\_t r; //初始化返回变量

gpio\_num\_t adc\_gpio\_num, dac\_gpio\_num; //DAC引脚定义

r = adc2\_pad\_get\_io\_num( ADC2\_EXAMPLE\_CHANNEL, &adc\_gpio\_num );//ADC配置

assert( r == ESP\_OK ); //确认正确与否

r = dac\_pad\_get\_io\_num( DAC\_EXAMPLE\_CHANNEL, &dac\_gpio\_num ); //DAC配置

assert( r == ESP\_OK ); //确认正确与否

printf("ADC2 channel %d @ GPIO %d, DAC channel %d @ GPIO %d.\n", ADC2\_EXAMPLE\_CHANNEL, adc\_gpio\_num, DAC\_EXAMPLE\_CHANNEL + 1, dac\_gpio\_num ); //打印信息

dac\_output\_enable( DAC\_EXAMPLE\_CHANNEL ); //DAC使能

printf("adc2\_init...\n");

adc2\_config\_channel\_atten( ADC2\_EXAMPLE\_CHANNEL, ADC\_ATTEN\_11db );

//ADC初始化

vTaskDelay(2 \* portTICK\_PERIOD\_MS);

printf("start conversion.\n");

while(1) {

dac\_output\_voltage( DAC\_EXAMPLE\_CHANNEL, output\_data++ ); //DAC输出

r = adc2\_get\_raw( ADC2\_EXAMPLE\_CHANNEL, width, &read\_raw);//ADC输出

if ( r == ESP\_OK ) {

printf("%d: %d\n", output\_data, read\_raw );

} else if ( r == ESP\_ERR\_INVALID\_STATE ) {

printf("%s: ADC2 not initialized yet.\n", esp\_err\_to\_name(r));

} else if ( r == ESP\_ERR\_TIMEOUT ) { //使用WiFi出现的情况

printf("%s: ADC2 is in use by WiFi.\n", esp\_err\_to\_name(r));

} else {

printf("%s\n", esp\_err\_to\_name(r));

}

vTaskDelay( 2 \* portTICK\_PERIOD\_MS );

}

}

5. Arduino开发环境实现ADC-DAC变换

代码如下：

#include "driver/gpio.h"

#include "driver/adc.h"

#include "driver/dac.h"

#include "esp\_system.h"

#define DAC\_EXAMPLE\_CHANNEL DAC\_CHANNEL\_1

#define ADC2\_EXAMPLE\_CHANNEL ADC2\_CHANNEL\_7

#if CONFIG\_IDF\_TARGET\_ESP32

static const adc\_bits\_width\_t width = ADC\_WIDTH\_BIT\_12;

#elif CONFIG\_IDF\_TARGET\_ESP32S2

static const adc\_bits\_width\_t width = ADC\_WIDTH\_BIT\_13;

#endif

void setup() {

Serial.begin(115200); //设置串口监视器波特率

}

void loop()

{

r = adc2\_pad\_get\_io\_num( ADC2\_EXAMPLE\_CHANNEL, &adc\_gpio\_num );//ADC配置

assert( r == ESP\_OK ); //确认正确与否

r = dac\_pad\_get\_io\_num( DAC\_EXAMPLE\_CHANNEL, &dac\_gpio\_num ); //DAC配置

assert( r == ESP\_OK ); //确认正确与否

Serial.printf("ADC2 channel %d @ GPIO %d, DAC channel %d @ GPIO %d.\n", ADC2\_EXAMPLE\_CHANNEL, adc\_gpio\_num, DAC\_EXAMPLE\_CHANNEL + 1, dac\_gpio\_num );//打印信息

dac\_output\_enable( DAC\_EXAMPLE\_CHANNEL ); //DAC使能

Serial.printf("adc2\_init...\n");

adc2\_config\_channel\_atten( ADC2\_EXAMPLE\_CHANNEL, ADC\_ATTEN\_11db );//ADC初始化

delay(2 \* portTICK\_PERIOD\_MS);

Serial.printf("start conversion.\n");

dac\_output\_voltage( DAC\_EXAMPLE\_CHANNEL, output\_data++ ); //DAC输出

r = adc2\_get\_raw( ADC2\_EXAMPLE\_CHANNEL, width, &read\_raw);//ADC输入

if ( r == ESP\_OK ) {

printf("%d: %d\n", output\_data, read\_raw );

} else if ( r == ESP\_ERR\_INVALID\_STATE ) {

printf("%s: ADC2 not initialized yet.\n", esp\_err\_to\_name(r));

} else if ( r == ESP\_ERR\_TIMEOUT ) { //使用WiFi出现的情况

printf("%s: ADC2 is in use by Wi-Fi.\n", esp\_err\_to\_name(r));

} else {

printf("%s\n", esp\_err\_to\_name(r));

}

delay( 2 \* portTICK\_PERIOD\_MS );

}

6. MicroPython开发环境ADC-DAC变换

代码如下：

#短接GPIO26和32

from machine import ADC, DAC, Pin

from time import sleep\_ms

adc = ADC(Pin(32))

adc.atten(ADC.ATTN\_11DB)

adc.width(ADC.WIDTH\_12BIT)

dac = DAC(Pin(26, Pin.OUT), bits=12)

for i in range(256):

dac.write(i)

print(' DAC: ' + str(i) + '\tADC: ' + str(adc.read()))

sleep\_ms(10)

### 4.5.3 定时器示例程序

#include <stdio.h>

#include "esp\_types.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "soc/timer\_group\_struct.h"

#include "driver/periph\_ctrl.h"

#include "driver/timer.h"

#include "driver/gpio.h"

#define LED 2

esp\_timer\_handle\_t test\_p\_handle = 0;

void test\_timer\_periodic\_cb(void \*arg) { //回调程序

gpio\_set\_level(LED, 0); //电平为低

vTaskDelay(1000 / portTICK\_PERIOD\_MS); //延迟1S

gpio\_set\_level(LED, 1); //电平为高

vTaskDelay(1000 / portTICK\_PERIOD\_MS); //延迟1S

printf("Hello, LED\n");

}

void app\_main() //主程序入口

{

gpio\_pad\_select\_gpio(LED); //选择芯片引脚

gpio\_set\_direction(LED, GPIO\_MODE\_OUTPUT); //设置该引脚为输出模式

//定义一个周期重复运行的定时器结构体

esp\_timer\_create\_args\_t test\_periodic\_arg = {

.callback =

&test\_timer\_periodic\_cb, //设置回调函数

.arg = NULL, //不携带参数

.name = "TestPeriodicTimer" //定时器名字

};

esp\_err\_t err = esp\_timer\_create(&test\_periodic\_arg, &test\_p\_handle); //创建定时器

err = esp\_timer\_start\_periodic(test\_p\_handle, 1000 \* 1000); //开启周期定时

if (err==0) //判断并打印信息

printf("Timer Start: ESP\_OK!\n" );

}

2. Arduino开发环境实现定时器控制LED

代码如下：

#define LED 2 //定义输出引脚

hw\_timer\_t \* timer = NULL;

volatile SemaphoreHandle\_t timerSemaphore;

portMUX\_TYPE timerMux = portMUX\_INITIALIZER\_UNLOCKED;

volatile uint32\_t isrCounter = 0;

volatile uint32\_t lastIsrAt = 0;

void IRAM\_ATTR onTimer(){ //回调函数

portENTER\_CRITICAL\_ISR(&timerMux); //递增计数器并设置ISR的时间

isrCounter++;

lastIsrAt = millis();

portEXIT\_CRITICAL\_ISR(&timerMux);

xSemaphoreGiveFromISR(timerSemaphore, NULL); //给出一个可以在循环中检查的信号量

//如果想切换输出，在此处使用digitalRead/Write

digitalWrite(LED, !digitalRead(LED)); //电平翻转

Serial.printf("Hello, LED\n");

}

void setup() {

Serial.begin(115200);

pinMode(LED, OUTPUT); //设置引脚状态为输出

timerSemaphore = xSemaphoreCreateBinary(); //创建信号标，标记计时器启动

//使用第1个计时器（从零开始计数），预设80分频器（更多信息，请参阅ESP32技术参考手册）

timer = timerBegin(0, 80, true);

timerAttachInterrupt(timer, &onTimer, true); //将onTimer函数附加到计时器

//将alarm设置为每秒调用onTimer函数（以微秒为单位的值），重复报警（第三个参数）

timerAlarmWrite(timer, 1000 \* 1000, true);

timerAlarmEnable(timer); //启动alarm

}

void loop() {

if (xSemaphoreTake(timerSemaphore, 0) == pdTRUE){ //如果计时器已启动

uint32\_t isrCount = 0, isrTime = 0; //读取中断计数和时间

portENTER\_CRITICAL(&timerMux);

isrCount = isrCounter;

isrTime = lastIsrAt;

portEXIT\_CRITICAL(&timerMux);

Serial.print("onTimer no. ");

Serial.print(isrCount);

Serial.print(" at ");

Serial.print(isrTime);

Serial.println(" ms");

}

}

3. MicroPython实现定时器控制LED

代码如下：

from machine import Timer, Pin

from time import sleep\_ms

def toggle\_led(timer):

led\_pin.value(0) #电平为低

sleep\_ms(1000) #延时

led\_pin.value(1) #电平为高

sleep\_ms(1000) #延时

print('Hello, LED')

#声明引脚2作为LED的引脚

led\_pin = Pin(2, Pin.OUT)

timer = Timer(1) #创建定时器对象

timer.init(period = 1000, mode = Timer.PERIODIC, callback = toggle\_led)

4. 基于ESP IDF的VS Code开发环境定时器重启控制LED

本例通过定时器回调程序，实现50s的LED亮灭，然后重新启动，在串口打印时间和信息，代码如下：

#include <stdio.h>

#include "esp\_types.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "soc/timer\_group\_struct.h"

#include "driver/periph\_ctrl.h"

#include "driver/timer.h"

#include "driver/gpio.h"

#define LED 2

esp\_timer\_handle\_t led\_timer\_handle = 0;

void led\_timer\_cb(void \*arg)

{

int64\_t tick = esp\_timer\_get\_time();//获取时间戳

printf("timer cnt = %lld \r\n", tick);

if (tick > 50000000) //50s结束

{

esp\_timer\_stop(led\_timer\_handle); //定时器暂停、删除

esp\_timer\_delete(led\_timer\_handle);

printf("timer stop and delete!!! \r\n");

esp\_restart(); //重启

}

gpio\_set\_level(LED, 0); //设置0电平

vTaskDelay(1000/ portTICK\_PERIOD\_MS);

gpio\_set\_level(LED, 1); //设置1电平

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

void app\_main() { //主程序入口

gpio\_pad\_select\_gpio(LED); //选择I/O

gpio\_set\_direction(LED, GPIO\_MODE\_OUTPUT); //设置I/O为输出

esp\_timer\_create\_args\_t led\_timer = //定时器结构体初始化

{

.callback = &led\_timer\_cb, //回调函数

.arg = NULL, //参数

.name = "led\_timer" //定时器名称

};

esp\_err\_t err = esp\_timer\_create(&led\_timer, &led\_timer\_handle); //定时器创建、启动

err = esp\_timer\_start\_periodic(led\_timer\_handle, 1000 \* 1000); //1s回调

if(err == ESP\_OK)

{

printf("led timer cteate and start ok!\r\n");

}

}

5. Arduino开发环境实现定时器重启控制LED

代码如下：

#define LED 2 //定义输出引脚

hw\_timer\_t \* timer = NULL;

volatile SemaphoreHandle\_t timerSemaphore;

portMUX\_TYPE timerMux = portMUX\_INITIALIZER\_UNLOCKED;

volatile uint32\_t isrCounter = 0;

volatile uint32\_t lastIsrAt = 0;

void IRAM\_ATTR onTimer(){ //回调函数

if(millis() > 50000) { // millis()函数获取时间戳（单位ms），50s结束

if (timer) { //如果计时器仍在运行

timerEnd(timer); //停止并释放计时器

timer = NULL;

esp\_restart(); //重启

}

}

portENTER\_CRITICAL\_ISR(&timerMux); //递增计数器并设置ISR的时间

isrCounter++;

lastIsrAt = millis();

portEXIT\_CRITICAL\_ISR(&timerMux);

//给出一个可以在循环中检查的信号量

xSemaphoreGiveFromISR(timerSemaphore, NULL);

//如果想切换输出，在此处使用digitalRead/Write

digitalWrite(LED, !digitalRead(LED)); //电平翻转

}

void setup() {

Serial.begin(115200);

pinMode(LED, OUTPUT); //设置引脚状态为输出

//创建信号标，标记计时器启动

timerSemaphore = xSemaphoreCreateBinary();

//使用第1个计时器（从零开始计数），预设80分频器（更多信息，请参阅ESP32技术参考手册）

timer = timerBegin(0, 80, true);

timerAttachInterrupt(timer, &onTimer, true); //将onTimer函数附加到计时器

//将alarm设置为每秒调用onTimer函数（以微秒为单位的值），重复报警（第三个参数）

timerAlarmWrite(timer, 1000 \* 1000, true);

timerAlarmEnable(timer); //启动alarm

}

void loop() {

if (xSemaphoreTake(timerSemaphore, 0) == pdTRUE){ //如果计时器已启动

uint32\_t isrCount = 0, isrTime = 0;

//读取中断计数和时间

portENTER\_CRITICAL(&timerMux);

isrCount = isrCounter;

isrTime = lastIsrAt;

portEXIT\_CRITICAL(&timerMux);

Serial.print("onTimer no. ");

Serial.print(isrCount);

Serial.print(" at ");

Serial.print(isrTime);

Serial.println(" ms");

}

}

6. MicroPython开发环境实现定时器重启控制LED

代码如下：

import machine

from machine import Pin, Timer

import utime

start = utime.ticks\_ms()

def led(t):

led\_pin.value(0) #电平为低

utime.sleep\_ms(1000) #延时

led\_pin.value(1) #电平为高

utime.sleep\_ms(1000) #延时

times = utime.ticks\_diff(utime.ticks\_ms(), start)

print(times)

if times > 50000:

t.deinit() #反初始化定时器：停止定时器，并禁用定时器外设

print('timer stop and delete!!!')

machine.soft\_reset() #软复位，注意MicroPython环境进行硬复位（如按下复位键）后必须先断开串口，然后重新连接，才能再次进行烧录

led\_pin = Pin(2, Pin.OUT)

cnt = 0

t = Timer(1)

t.init(period = 2000, mode = Timer.PERIODIC, callback = led)

### 5.1.4 UART示例程序

1. 基于ESP IDF的VS Code开发环境UART0示例

代码如下：

#include <stdio.h>

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/uart.h"

#include "esp\_log.h"

static const char \*TAG = "uart\_events";

//UART0，接收缓存开，发送缓冲关，流控关闭，事件队列打开，收发为默认引脚

#define EX\_UART\_NUM UART\_NUM\_0

#define PATTERN\_CHR\_NUM (3) /\*定义UART模式\*/

#define BUF\_SIZE (1024)

#define RD\_BUF\_SIZE (BUF\_SIZE)

static QueueHandle\_t uart0\_queue;

static void uart\_event\_task(void \*pvParameters) //定义事件任务

{

uart\_event\_t event;

size\_t buffered\_size;

uint8\_t\* dtmp = (uint8\_t\*) malloc(RD\_BUF\_SIZE);

for(;;) { //等待UART事件

if(xQueueReceive(uart0\_queue, (void \* )&event, (portTickType)portMAX\_DELAY)) {

bzero(dtmp, RD\_BUF\_SIZE);

ESP\_LOGI(TAG, "uart[%d] event:", EX\_UART\_NUM);

switch(event.type) { // UART接收数据的事件

case UART\_DATA: /\*快速处理数据事件，防止队列填满 \*/

ESP\_LOGI(TAG, "[UART DATA]: %d", event.size);

uart\_read\_bytes(EX\_UART\_NUM,dtmp,event.size, portMAX\_DELAY);

ESP\_LOGI(TAG, "[DATA EVT]:");

uart\_write\_bytes(EX\_UART\_NUM, (const char\*) dtmp, event.size);

break;

case UART\_FIFO\_OVF: //HW FIFO 溢出检测事件

ESP\_LOGI(TAG, "hw fifo overflow");

//如果发生溢出，则考虑是应用程序添加流控制

// ISR已经重置了RX FIFO，

//例如，直接在此处刷新rx缓冲区以读取更多数据

uart\_flush\_input(EX\_UART\_NUM);

xQueueReset(uart0\_queue);

break;

case UART\_BUFFER\_FULL: //UART环形缓冲区满的事件

ESP\_LOGI(TAG, "ring buffer full");

//如果缓冲区已满，则应考虑增加缓冲区大小

//例如，直接在此处刷新rx缓冲区以读取更多数据

uart\_flush\_input(EX\_UART\_NUM);

xQueueReset(uart0\_queue);

break;

case UART\_BREAK: //检测到UART RX中断事件

ESP\_LOGI(TAG, "uart rx break");

break;

case UART\_PARITY\_ERR: //UART奇偶校验错误

ESP\_LOGI(TAG, "uart parity error");

break;

case UART\_FRAME\_ERR: //UART帧错误事件

ESP\_LOGI(TAG, "uart frame error");

break;

case UART\_PATTERN\_DET: //UART\_PATTERN\_DET模式检测

uart\_get\_buffered\_data\_len(EX\_UART\_NUM, &buffered\_size);

int pos = uart\_pattern\_pop\_pos(EX\_UART\_NUM);

ESP\_LOGI(TAG, "[UART PATTERN DETECTED] pos: %d, buffered size: %d", pos, buffered\_size);

if (pos == -1) {

//过去的UART\_PATTERN\_DET事件

//模式位置队列已满，无法记录位置。应该设置更大的队列

//例如，直接刷新RX缓冲区.

uart\_flush\_input(EX\_UART\_NUM);

} else {

uart\_read\_bytes(EX\_UART\_NUM, dtmp, pos, 100 / portTICK\_PERIOD\_MS);

uint8\_t pat[PATTERN\_CHR\_NUM + 1];

memset(pat, 0, sizeof(pat));

uart\_read\_bytes(EX\_UART\_NUM,pat, PATTERN\_CHR\_NUM, 100 / portTICK\_PERIOD\_MS);

ESP\_LOGI(TAG, "read data: %s", dtmp);

ESP\_LOGI(TAG, "read pat : %s", pat);

}

break;

default: //其他情况

ESP\_LOGI(TAG, "uart event type: %d", event.type);

break;

}

}

}

free(dtmp);

dtmp = NULL;

vTaskDelete(NULL);

}

void app\_main(void)

{

esp\_log\_level\_set(TAG, ESP\_LOG\_INFO);

/\*配置UART驱动程序的参数，通信引脚并安装驱动程序\*/

uart\_config\_t uart\_config = {

.baud\_rate = 115200,

.data\_bits = UART\_DATA\_8\_BITS,

.parity = UART\_PARITY\_DISABLE,

.stop\_bits = UART\_STOP\_BITS\_1,

.flow\_ctrl = UART\_HW\_FLOWCTRL\_DISABLE,

.source\_clk = UART\_SCLK\_APB,

};

//安装UART驱动程序，并获取队列

uart\_driver\_install(EX\_UART\_NUM,BUF\_SIZE \*2,BUF\_SIZE\*2,20,&uart0\_queue, 0);

uart\_param\_config(EX\_UART\_NUM, &uart\_config);

//设置UART日志级别

esp\_log\_level\_set(TAG, ESP\_LOG\_INFO);

//设置UART引脚，使用UART0默认引脚，即没有变化

uart\_set\_pin(EX\_UART\_NUM,UART\_PIN\_NO\_CHANGE, UART\_PIN\_NO\_CHANGE, UART\_PIN\_NO\_CHANGE, UART\_PIN\_NO\_CHANGE);

//设置UART模式检测功能

uart\_enable\_pattern\_det\_baud\_intr(EX\_UART\_NUM,'+',PATTERN\_CHR\_NUM,9,0, 0);

//重置模式队列长度以最多记录20个位置

uart\_pattern\_queue\_reset(EX\_UART\_NUM, 20);

//创建任务以处理来自ISR的UART事件

xTaskCreate(uart\_event\_task, "uart\_event\_task", 2048, NULL, 12, NULL);

}

2. 基于ESP IDF的VS Code开发环境UART1示例

本示例可以在任何常用的ESP32开发板上运行。使用Micro-USB电缆连接开发板到计算机，一条单线电缆，用于短接开发板的两个引脚。需要在代码中配置的“RXD\_PIN”和“TXD\_PIN”（本例为“GPIO4”和“GPIO5”）。为了接收已发送的相同数据，启动了两个FreeRTOS任务：第一项任务通过UART定期发送“Hello world”。第二项任务监听，接收并打印来自UART的数据。代码如下：

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_log.h"

#include "driver/uart.h"

#include "string.h"

#include "driver/gpio.h"

static const int RX\_BUF\_SIZE = 1024;

#define TXD\_PIN (GPIO\_NUM\_4)

#define RXD\_PIN (GPIO\_NUM\_5)

void init(void) { //初始化串口

const uart\_config\_t uart\_config = {

.baud\_rate = 115200,

.data\_bits = UART\_DATA\_8\_BITS,

.parity = UART\_PARITY\_DISABLE,

.stop\_bits = UART\_STOP\_BITS\_1,

.flow\_ctrl = UART\_HW\_FLOWCTRL\_DISABLE,

.source\_clk = UART\_SCLK\_APB,

};

//不使用缓冲区发送数据

uart\_driver\_install(UART\_NUM\_1, RX\_BUF\_SIZE \* 2, 0, 0, NULL, 0);

uart\_param\_config(UART\_NUM\_1, &uart\_config);

uart\_set\_pin(UART\_NUM\_1, TXD\_PIN, RXD\_PIN, UART\_PIN\_NO\_CHANGE, UART\_PIN\_NO\_CHANGE);

}

int sendData(const char\* logName, const char\* data) //发送数据函数

{

const int len = strlen(data);

const int txBytes = uart\_write\_bytes(UART\_NUM\_1, data, len);

ESP\_LOGI(logName, "Wrote %d bytes", txBytes);

return txBytes;

}

static void tx\_task(void \*arg) //发送任务函数

{

static const char \*TX\_TASK\_TAG = "TX\_TASK";

esp\_log\_level\_set(TX\_TASK\_TAG, ESP\_LOG\_INFO);

while (1) {

sendData(TX\_TASK\_TAG, "Hello world");

vTaskDelay(2000 / portTICK\_PERIOD\_MS);

}

}

static void rx\_task(void \*arg) //接收任务函数

{

static const char \*RX\_TASK\_TAG = "RX\_TASK";

esp\_log\_level\_set(RX\_TASK\_TAG, ESP\_LOG\_INFO);

uint8\_t\* data = (uint8\_t\*) malloc(RX\_BUF\_SIZE+1);

while (1) {

const int rxBytes = uart\_read\_bytes(UART\_NUM\_1, data, RX\_BUF\_SIZE, 1000 / portTICK\_RATE\_MS);

if (rxBytes > 0) {

data[rxBytes] = 0;

ESP\_LOGI(RX\_TASK\_TAG, "Read %d bytes: '%s'", rxBytes, data);

ESP\_LOG\_BUFFER\_HEXDUMP(RX\_TASK\_TAG,data,rxBytes, ESP\_LOG\_INFO);

}

}

free(data);

}

void app\_main(void) //主程序入口

{

init(); //初始化

xTaskCreate(rx\_task, "uart\_rx\_task", 1024\*2, NULL, configMAX\_PRIORITIES, NULL);

//创建接收任务

xTaskCreate(tx\_task, "uart\_tx\_task", 1024\*2, NULL, configMAX\_PRIORITIES-1, NULL);

//创建发送任务

}

3. Arduino开发环境实现UART操作示例

本示例通过Arduino IDE实现在串口写入数据后，串口可以读取写入的数据，并打印在串口监视器，代码如下：

void setup() {

//设定串口波特率

Serial.begin(115200);

}

void loop() {

if (Serial.available()) {

delay (100); //等待数据传输完毕

int n = Serial.available();

Serial.print("接收到 ");

Serial.print(n);

Serial.print("字节数据：");

delay (100);

for (int i = 0; i < n; ++i) {

Serial.print((char)Serial.read());

}

Serial.println();

}

}

4. MicroPython开发环境实现UART操作示例

ESP32串口通信-字符串数据自发实验，将开发板的5引脚与4引脚用杜邦线相连接。

from machine import UART,Pin

import utime

#初始化一个UART对象

uart = UART(2, baudrate=115200, rx=5,tx=4,timeout=10)

count = 1

while True:

print('\n\n===============CNT {}==============='.format(count))

#发送一条消息

print('Send: {}'.format('hello world {}\n'.format(count)))

print('Send Byte :',len('hello world {}\n')) # 发送字节数

uart.write('hello world {}\n'.format(count))

#等待1s

utime.sleep\_ms(1000)

if uart.any():

#如果有数据 读入一行数据返回数据为字节类型

#例如：b'hello 1\n'

bin\_data = uart.readline()

#将收到的信息打印在终端

print('Echo Byte: {}'.format(bin\_data))

#将字节数据转换为字符串，字节默认为UTF-8编码

print('Echo String: {}'.format(bin\_data.decode()))

#计数器+1

count += 1

print('---------------------------------------')

### 5.2.4 I2C示例程序

1. 基于ESP IDF的VS Code开发环境实现

#include <stdio.h>

#include "esp\_log.h"

#include "driver/i2c.h"

#include "sdkconfig.h"

static const char \*TAG = "i2c-example";

#define \_I2C\_NUMBER(num) I2C\_NUM\_##num

#define I2C\_NUMBER(num) \_I2C\_NUMBER(num)

#define DATA\_LENGTH 512 /\*数据缓冲区长度\*/

#define RW\_TEST\_LENGTH 128 /\*读/写测试数据长度，[0,DATA\_LENGTH] \*/

#define DELAY\_TIME\_BETWEEN\_ITEMS\_MS 1000 /\*不同测试项之间的延迟\*/

#define I2C\_SLAVE\_SCL\_IO CONFIG\_I2C\_SLAVE\_SCL /\*从设备SCL的GPIO引脚\*/

#define I2C\_SLAVE\_SDA\_IO CONFIG\_I2C\_SLAVE\_SDA /\*从设备SDA的GPIO引脚\*/

#define I2C\_SLAVE\_NUM I2C\_NUMBER(CONFIG\_I2C\_SLAVE\_PORT\_NUM) /\*I2C从设备端口号\*/

#define I2C\_SLAVE\_TX\_BUF\_LEN (2 \* DATA\_LENGTH) /\* I2C从设备发送缓冲大小\*/

#define I2C\_SLAVE\_RX\_BUF\_LEN (2 \* DATA\_LENGTH) /\* I2C从设备接收缓冲大小\*/

#define I2C\_MASTER\_SCL\_IO CONFIG\_I2C\_MASTER\_SCL /\*主设备SCL的GPIO引脚 \*/

#define I2C\_MASTER\_SDA\_IO CONFIG\_I2C\_MASTER\_SDA /\*主设备SDA的GPIO引脚\*/

#define I2C\_MASTER\_NUM I2C\_NUMBER(CONFIG\_I2C\_MASTER\_PORT\_NUM) /\*I2C主设备端口号\*/

#define I2C\_MASTER\_FREQ\_HZ CONFIG\_I2C\_MASTER\_FREQUENCY /\*I2C主设备时钟频率\*/

#define I2C\_MASTER\_TX\_BUF\_DISABLE 0 /\* I2C主设备不用发送缓冲\*/

#define I2C\_MASTER\_RX\_BUF\_DISABLE 0 /\* I2C主设备不用接收缓冲\*/

#define BH1750\_SENSOR\_ADDR CONFIG\_BH1750\_ADDR /\*BH1750传感器地址\*/

#define BH1750\_CMD\_START CONFIG\_BH1750\_OPMODE /\* BH1750操作模式\*/

#define ESP\_SLAVE\_ADDR CONFIG\_I2C\_SLAVE\_ADDRESS /\*ESP32从设备地址\*/

#define WRITE\_BIT I2C\_MASTER\_WRITE /\*I2C主设备写入\*/

#define READ\_BIT I2C\_MASTER\_READ /\* I2C主设备读取\*/

#define ACK\_CHECK\_EN 0x1 /\*I2C主设备检查ACK\*/

#define ACK\_CHECK\_DIS 0x0 /\*I2C主设备不检查ACK\*/

#define ACK\_VAL 0x0 /\* I2C的ACK值 \*/

#define NACK\_VAL 0x1 /\*I2C的NACK的值\*/

SemaphoreHandle\_t print\_mux = NULL;

/\*主设备读取从设备数据\*/

static esp\_err\_t \_\_attribute\_\_((unused)) i2c\_master\_read\_slave(i2c\_port\_t i2c\_num, uint8\_t \*data\_rd, size\_t size)

{

if (size == 0) {

return ESP\_OK;

}

i2c\_cmd\_handle\_t cmd = i2c\_cmd\_link\_create();

i2c\_master\_start(cmd);

i2c\_master\_write\_byte(cmd, (ESP\_SLAVE\_ADDR << 1) | READ\_BIT, ACK\_CHECK\_EN);

if (size > 1) {

i2c\_master\_read(cmd, data\_rd, size - 1, ACK\_VAL);

}

i2c\_master\_read\_byte(cmd, data\_rd + size - 1, NACK\_VAL);

i2c\_master\_stop(cmd);

esp\_err\_t ret = i2c\_master\_cmd\_begin(i2c\_num, cmd, 1000 / portTICK\_RATE\_MS);

i2c\_cmd\_link\_delete(cmd);

return ret;

}

/\*主设备写数据到从设备\*/

static esp\_err\_t \_\_attribute\_\_((unused)) i2c\_master\_write\_slave(i2c\_port\_t i2c\_num, uint8\_t \*data\_wr, size\_t size)

{

i2c\_cmd\_handle\_t cmd = i2c\_cmd\_link\_create();

i2c\_master\_start(cmd);

i2c\_master\_write\_byte(cmd, (ESP\_SLAVE\_ADDR << 1) | WRITE\_BIT, ACK\_CHECK\_EN);

i2c\_master\_write(cmd, data\_wr, size, ACK\_CHECK\_EN);

i2c\_master\_stop(cmd);

esp\_err\_t ret = i2c\_master\_cmd\_begin(i2c\_num, cmd, 1000 / portTICK\_RATE\_MS);

i2c\_cmd\_link\_delete(cmd);

return ret;

}

/\*传感器操作 \*/

static esp\_err\_t i2c\_master\_sensor\_test(i2c\_port\_t i2c\_num, uint8\_t \*data\_h, uint8\_t \*data\_l)

{

int ret;

i2c\_cmd\_handle\_t cmd = i2c\_cmd\_link\_create();

i2c\_master\_start(cmd);

i2c\_master\_write\_byte(cmd, BH1750\_SENSOR\_ADDR << 1 | WRITE\_BIT, ACK\_CHECK\_EN);

i2c\_master\_write\_byte(cmd, BH1750\_CMD\_START, ACK\_CHECK\_EN);

i2c\_master\_stop(cmd);

ret = i2c\_master\_cmd\_begin(i2c\_num, cmd, 1000 / portTICK\_RATE\_MS);

i2c\_cmd\_link\_delete(cmd);

if (ret != ESP\_OK) {

return ret;

}

vTaskDelay(30 / portTICK\_RATE\_MS);

cmd = i2c\_cmd\_link\_create();

i2c\_master\_start(cmd);

i2c\_master\_write\_byte(cmd, BH1750\_SENSOR\_ADDR << 1 | READ\_BIT, ACK\_CHECK\_EN);

i2c\_master\_read\_byte(cmd, data\_h, ACK\_VAL);

i2c\_master\_read\_byte(cmd, data\_l, NACK\_VAL);

i2c\_master\_stop(cmd);

ret = i2c\_master\_cmd\_begin(i2c\_num, cmd, 1000 / portTICK\_RATE\_MS);

i2c\_cmd\_link\_delete(cmd);

return ret;

}

/\*主设备初始化\*/

static esp\_err\_t i2c\_master\_init(void)

{

int i2c\_master\_port = I2C\_MASTER\_NUM;

i2c\_config\_t conf = {

.mode = I2C\_MODE\_MASTER,

.sda\_io\_num = I2C\_MASTER\_SDA\_IO,

.sda\_pullup\_en = GPIO\_PULLUP\_ENABLE,

.scl\_io\_num = I2C\_MASTER\_SCL\_IO,

.scl\_pullup\_en = GPIO\_PULLUP\_ENABLE,

.master.clk\_speed = I2C\_MASTER\_FREQ\_HZ,

// .clk\_flags = 0, /\*可选，使用I2C\_SCLK\_SRC\_FLAG\_\*选择时钟源\*/

};

esp\_err\_t err = i2c\_param\_config(i2c\_master\_port, &conf);

if (err != ESP\_OK) {

return err;

}

return i2c\_driver\_install(i2c\_master\_port, conf.mode, I2C\_MASTER\_RX\_BUF\_DISABLE, I2C\_MASTER\_TX\_BUF\_DISABLE, 0);

}

#if !CONFIG\_IDF\_TARGET\_ESP32C3

/\*从设备初始化\*/

static esp\_err\_t i2c\_slave\_init(void)

{

int i2c\_slave\_port = I2C\_SLAVE\_NUM;

i2c\_config\_t conf\_slave = {

.sda\_io\_num = I2C\_SLAVE\_SDA\_IO,

.sda\_pullup\_en = GPIO\_PULLUP\_ENABLE,

.scl\_io\_num = I2C\_SLAVE\_SCL\_IO,

.scl\_pullup\_en = GPIO\_PULLUP\_ENABLE,

.mode = I2C\_MODE\_SLAVE,

.slave.addr\_10bit\_en = 0,

.slave.slave\_addr = ESP\_SLAVE\_ADDR,

};

esp\_err\_t err = i2c\_param\_config(i2c\_slave\_port, &conf\_slave);

if (err != ESP\_OK) {

return err;

}

return i2c\_driver\_install(i2c\_slave\_port, conf\_slave.mode, I2C\_SLAVE\_RX\_BUF\_LEN, I2C\_SLAVE\_TX\_BUF\_LEN, 0);

}

/\*显示缓冲区数据\*/

static void disp\_buf(uint8\_t \*buf, int len)

{

int i;

for (i = 0; i < len; i++) {

printf("%02x ", buf[i]);

if ((i + 1) % 16 == 0) {

printf("\n");

}

}

printf("\n");

}

#endif //!CONFIG\_IDF\_TARGET\_ESP32C3

/\*构建测试任务\*/

static void i2c\_test\_task(void \*arg)

{

int ret;

uint32\_t task\_idx = (uint32\_t)arg;

#if !CONFIG\_IDF\_TARGET\_ESP32C3

int i = 0;

uint8\_t \*data = (uint8\_t \*)malloc(DATA\_LENGTH);

uint8\_t \*data\_wr = (uint8\_t \*)malloc(DATA\_LENGTH);

uint8\_t \*data\_rd = (uint8\_t \*)malloc(DATA\_LENGTH);

#endif //!CONFIG\_IDF\_TARGET\_ESP32C3

uint8\_t sensor\_data\_h, sensor\_data\_l;

int cnt = 0;

while (1) {

ESP\_LOGI(TAG, "TASK[%d] test cnt: %d", task\_idx, cnt++);

ret = i2c\_master\_sensor\_test(I2C\_MASTER\_NUM, &sensor\_data\_h, &sensor\_data\_l);

xSemaphoreTake(print\_mux, portMAX\_DELAY);

if (ret == ESP\_ERR\_TIMEOUT) {

ESP\_LOGE(TAG, "I2C Timeout");

} else if (ret == ESP\_OK) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("TASK[%d] MASTER READ SENSOR( BH1750 )\n", task\_idx);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("data\_h: %02x\n", sensor\_data\_h);

printf("data\_l: %02x\n", sensor\_data\_l);

printf("sensor val: %.02f [Lux]\n", (sensor\_data\_h << 8 | sensor\_data\_l) / 1.2);

} else {

ESP\_LOGW(TAG, "%s: No ack, sensor not connected...skip...", esp\_err\_to\_name(ret));

}

xSemaphoreGive(print\_mux);

vTaskDelay((DELAY\_TIME\_BETWEEN\_ITEMS\_MS \* (task\_idx + 1)) / portTICK\_RATE\_MS);

//---------------------------------------------------

#if !CONFIG\_IDF\_TARGET\_ESP32C3

for (i = 0; i < DATA\_LENGTH; i++) {

data[i] = i;

}

xSemaphoreTake(print\_mux, portMAX\_DELAY);

size\_t d\_size = i2c\_slave\_write\_buffer(I2C\_SLAVE\_NUM, data, RW\_TEST\_LENGTH, 1000 / portTICK\_RATE\_MS);

if (d\_size == 0) {

ESP\_LOGW(TAG, "i2c slave tx buffer full");

ret = i2c\_master\_read\_slave(I2C\_MASTER\_NUM, data\_rd, DATA\_LENGTH);

} else {

ret = i2c\_master\_read\_slave(I2C\_MASTER\_NUM, data\_rd, RW\_TEST\_LENGTH);

}

if (ret == ESP\_ERR\_TIMEOUT) {

ESP\_LOGE(TAG, "I2C Timeout");

} else if (ret == ESP\_OK) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("TASK[%d] MASTER READ FROM SLAVE\n", task\_idx);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("====TASK[%d] Slave buffer data ====\n", task\_idx);

disp\_buf(data, d\_size);

printf("====TASK[%d] Master read ====\n", task\_idx);

disp\_buf(data\_rd, d\_size);

} else {

ESP\_LOGW(TAG, "TASK[%d] %s: Master read slave error, IO not connected...\n",

task\_idx, esp\_err\_to\_name(ret));

}

xSemaphoreGive(print\_mux);

vTaskDelay((DELAY\_TIME\_BETWEEN\_ITEMS\_MS \* (task\_idx + 1)) / portTICK\_RATE\_MS);

//---------------------------------------------------

int size;

for (i = 0; i < DATA\_LENGTH; i++) {

data\_wr[i] = i + 10;

}

xSemaphoreTake(print\_mux, portMAX\_DELAY);

//填充从设备的缓冲区，以便主设备可以读取

ret = i2c\_master\_write\_slave(I2C\_MASTER\_NUM, data\_wr, RW\_TEST\_LENGTH);

if (ret == ESP\_OK) {

size = i2c\_slave\_read\_buffer(I2C\_SLAVE\_NUM, data, RW\_TEST\_LENGTH, 1000 / portTICK\_RATE\_MS);

}

if (ret == ESP\_ERR\_TIMEOUT) {

ESP\_LOGE(TAG, "I2C Timeout");

} else if (ret == ESP\_OK) {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("TASK[%d] MASTER WRITE TO SLAVE\n", task\_idx);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("----TASK[%d] Master write ----\n", task\_idx);

disp\_buf(data\_wr, RW\_TEST\_LENGTH);

printf("----TASK[%d] Slave read: [%d] bytes ----\n", task\_idx, size);

disp\_buf(data, size);

} else {

ESP\_LOGW(TAG, "TASK[%d] %s: Master write slave error, IO not connected....\n",

task\_idx, esp\_err\_to\_name(ret));

}

xSemaphoreGive(print\_mux);

vTaskDelay((DELAY\_TIME\_BETWEEN\_ITEMS\_MS \* (task\_idx + 1)) / portTICK\_RATE\_MS);

#endif //!CONFIG\_IDF\_TARGET\_ESP32C3

}

vSemaphoreDelete(print\_mux);

vTaskDelete(NULL);

}

void app\_main(void)

{

print\_mux = xSemaphoreCreateMutex();

#if !CONFIG\_IDF\_TARGET\_ESP32C3

ESP\_ERROR\_CHECK(i2c\_slave\_init());

#endif

ESP\_ERROR\_CHECK(i2c\_master\_init());

xTaskCreate(i2c\_test\_task, "i2c\_test\_task\_0", 1024 \* 2, (void \*)0, 10, NULL);

xTaskCreate(i2c\_test\_task, "i2c\_test\_task\_1", 1024 \* 2, (void \*)1, 10, NULL);

}

2. Arduino开发环境实现

在 Arduino IDE 中使用ESP32时，默认的 I2C 引脚为GPIO21 (SDA)，GPIO22 (SCL)，本例采用GY-30光强传感器模块，读取其I2C设备地址，并打印在串口监视器上。将GY-30的SDA引脚连接GPIO21，SCL引脚连接GPIO22，VCC接3.3V，GND连接GND，代码如下：

#include "Wire.h"

void setup() {

Serial.begin(115200);

Wire.begin();

}

void loop() {

byte error, address;

int nDevices = 0;

delay(5000);

Serial.println("Scanning for I2C devices ...");

for(address = 0x01; address < 0x7f; address++){

Wire.beginTransmission(address);

error = Wire.endTransmission();

if (error == 0){

Serial.printf("I2C device found at address 0x%02X\n", address);

nDevices++;

} else if(error != 2){

Serial.printf("Error %d at address 0x%02X\n", error, address);

}

}

if (nDevices == 0){

Serial.println("No I2C devices found");

}

}

3. MicroPython开发环境实现

代码如下：

本示例读取GY-30光照强度传感器的数值，传感器SCL引脚连接开发板GPIO22引脚，传感器SDA引脚连接开发板GPIO21引脚，传感器的VCC接开发板3.3V，传感器的GND连接开发板的GND，代码如下：

import time

from machine import Pin, SoftI2C

i2c = SoftI2C(scl = Pin(22),sda = Pin(21),freq = 10000) #软件I2C

addr\_list = i2c.scan() #获取设备的地址

i2c.writeto(addr\_list[0],b'\x10') #设置分辨率模式为连续高分辨率模式

while True:

data = i2c.readfrom(35,2) # 读取测量结果

result = float(data[0]\*0xff+data[1])/1.2 #处理测量结果

print(result)

### 5.3.4 I2S示例程序

1.基于ESP IDF的VS Code开发环境实现

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/i2s.h"

#include "driver/gpio.h"

#include "esp\_system.h"

#include <math.h>

#define SAMPLE\_RATE (36000) //采样速率

#define I2S\_NUM (0) //I2S端口0

#define WAVE\_FREQ\_HZ (100) //波形频率

#define PI (3.14159265)

#define I2S\_BCK\_IO (GPIO\_NUM\_13) //时钟引脚

#define I2S\_WS\_IO (GPIO\_NUM\_15) //声道选择

#define I2S\_DO\_IO (GPIO\_NUM\_21) //输出数据

#define I2S\_DI\_IO (-1) //输入数据（未使用）

#define SAMPLE\_PER\_CYCLE (SAMPLE\_RATE/WAVE\_FREQ\_HZ)

static void setup\_triangle\_sine\_waves(int bits) //设置三角波和正弦波

{

int \*samples\_data = malloc(((bits+8)/16)\*SAMPLE\_PER\_CYCLE\*4); //开辟存储空间

unsigned int i, sample\_val;

double sin\_float, triangle\_float, triangle\_step = (double) pow(2, bits) / SAMPLE\_PER\_CYCLE;

//设置三角波步长

size\_t i2s\_bytes\_write = 0;

printf("\r\nTest bits=%d free mem=%d, written data=%d\n", bits, esp\_get\_free\_heap\_size(), ((bits+8)/16)\*SAMPLE\_PER\_CYCLE\*4); //打印信息

triangle\_float = -(pow(2, bits)/2 - 1); //初始化三角波的值

for(i = 0; i < SAMPLE\_PER\_CYCLE; i++) {

sin\_float = sin(i \* 2 \* PI / SAMPLE\_PER\_CYCLE);

if(sin\_float >= 0)

triangle\_float += triangle\_step;

else

triangle\_float -= triangle\_step;

sin\_float \*= (pow(2, bits)/2 - 1);

if (bits == 16) {

sample\_val = 0;

sample\_val += (short)triangle\_float;

sample\_val = sample\_val << 16;

sample\_val += (short) sin\_float;

samples\_data[i] = sample\_val;

} else if (bits == 24) {

samples\_data[i\*2] = ((int) triangle\_float) << 8;

samples\_data[i\*2 + 1] = ((int) sin\_float) << 8;

} else {

samples\_data[i\*2] = ((int) triangle\_float);

samples\_data[i\*2 + 1] = ((int) sin\_float);

}

}

i2s\_set\_clk(I2S\_NUM, SAMPLE\_RATE, bits, 2); //设置时钟

// for(i = 0; i < SAMPLE\_PER\_CYCLE; i++) {

// if (bits == 16)

// i2s\_push\_sample(0, &samples\_data[i], 100);

// else

// i2s\_push\_sample(0, &samples\_data[i\*2], 100);

// }

i2s\_write(I2S\_NUM, samples\_data, ((bits+8)/16)\*SAMPLE\_PER\_CYCLE\*4, &i2s\_bytes\_write, 100); //写数据

free(samples\_data); //释放采样数据

}

void app\_main(void)

{

/\*对于36KHz的采样率，创建100Hz正弦波，每个周期需要36000/100 = 360个采样（每个采样4个字节或8个字节），取决于bits\_per\_sample，使用6个缓冲区，每个缓冲区需要60个采样，如果2个通道，每个通道16位，总缓冲区为360 \* 4 = 1440字节；如果是2通道，每个通道24/32位，则总缓冲区为360×8 = 2880字节\*/

i2s\_config\_t i2s\_config = { //配置参数

.mode = I2S\_MODE\_MASTER | I2S\_MODE\_TX, //设置主机发送模式

.sample\_rate = SAMPLE\_RATE,

.bits\_per\_sample = 16,

.channel\_format = I2S\_CHANNEL\_FMT\_RIGHT\_LEFT, //2通道

.communication\_format = I2S\_COMM\_FORMAT\_STAND\_MSB,

.dma\_buf\_count = 6,

.dma\_buf\_len = 60,

.use\_apll = false,

.intr\_alloc\_flags = ESP\_INTR\_FLAG\_LEVEL1 //中断为1

};

i2s\_pin\_config\_t pin\_config = { //引脚配置

.bck\_io\_num = I2S\_BCK\_IO,

.ws\_io\_num = I2S\_WS\_IO,

.data\_out\_num = I2S\_DO\_IO,

.data\_in\_num = I2S\_DI\_IO //未使用

};

i2s\_driver\_install(I2S\_NUM, &i2s\_config, 0, NULL); //安装I2S驱动

i2s\_set\_pin(I2S\_NUM, &pin\_config); //引脚初始化

int test\_bits = 16; //设置位宽

while (1) {

setup\_triangle\_sine\_waves(test\_bits); //开启波形变换

vTaskDelay(5000/portTICK\_RATE\_MS); //延迟5s

test\_bits += 8; //改变位宽，重新变换

if(test\_bits > 32)

test\_bits = 16;

}

}

2. Arduino开发环境实现

Arduino IDE的示例程序，GPIO34输入任何音频或模拟值，串口输出是：，代码如下：

#include <I2S.h>

void setup() {

Serial.begin(115200);

while (!Serial) {

;

}

if (!I2S.begin(ADC\_DAC\_MODE, 8000, 16)) { // I2S开启采样

Serial.println("Failed to initialize I2S!");

while (1);

}

}

void loop() {

int sample = I2S.read(); // 读取并打印采样值

Serial.println(sample);

}

3. MicroPython开发环境实现

MicroPython将I2S封装为专用于连接数字音频设备，I2S类当前还处于技术预览阶段。在预览期间，基于用户的反馈，I2S类API和实现可能会更改，实例的初始化代码如下：

from machine import Pin

from machine import I2S

sck\_pin = Pin(14) #串行时钟输出

ws\_pin = Pin(13) #字时钟输出

sd\_pin = Pin(12) #串行数据输出

audio\_out = I2S(0,

sck=sck\_pin, ws=ws\_pin, sd=sd\_pin,

mode=I2S.TX,

bits=16,

format=I2S.MONO,

rate=44100,

ibuf=20000)

audio\_in = I2S(0,

sck=sck\_pin, ws=ws\_pin, sd=sd\_pin,

mode=I2S.RX,

bits=32,

format=I2S.STEREO,

rate=22050,

ibuf=20000)

samples=bytearray(1024)

wav = open('sound.pcm','wb')

print('Starting')

num\_read = audio\_in.readinto(samples)

wav.write(samples)

wav.close()

audio\_in.deinit()

print(samples)

print('Done')

### 5.4.3 SPI示例程序

1. 基于ESP IDF的VS Code开发环境

本示例使用SPI总线驱动SPI接口的LCD，可以在LCD显示信息并在串口打印LCD的信息，代码如下：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "driver/spi\_master.h"

#include "driver/gpio.h"

#include "pretty\_effect.h"

#ifdef CONFIG\_IDF\_TARGET\_ESP32

#define LCD\_HOST HSPI\_HOST

#define PIN\_NUM\_MISO 25

#define PIN\_NUM\_MOSI 23

#define PIN\_NUM\_CLK 19

#define PIN\_NUM\_CS 22

#define PIN\_NUM\_DC 21

#define PIN\_NUM\_RST 18

#define PIN\_NUM\_BCKL 5

#elif defined CONFIG\_IDF\_TARGET\_ESP32S2

#define LCD\_HOST SPI2\_HOST

#define PIN\_NUM\_MISO 37

#define PIN\_NUM\_MOSI 35

#define PIN\_NUM\_CLK 36

#define PIN\_NUM\_CS 34

#define PIN\_NUM\_DC 4

#define PIN\_NUM\_RST 5

#define PIN\_NUM\_BCKL 6

#elif defined CONFIG\_IDF\_TARGET\_ESP32C3

#define LCD\_HOST SPI2\_HOST

#define PIN\_NUM\_MISO 2

#define PIN\_NUM\_MOSI 7

#define PIN\_NUM\_CLK 6

#define PIN\_NUM\_CS 10

#define PIN\_NUM\_DC 9

#define PIN\_NUM\_RST 4

#define PIN\_NUM\_BCKL 5

#endif

#define PARALLEL\_LINES 16

/\*LCD初始化定义\*/

typedef struct {

uint8\_t cmd;

uint8\_t data[16];

uint8\_t databytes; //No of data in data; bit 7 = delay after set; 0xFF = end of cmds.

} lcd\_init\_cmd\_t;

typedef enum {

LCD\_TYPE\_ILI = 1,

LCD\_TYPE\_ST,

LCD\_TYPE\_MAX,

} type\_lcd\_t;

//将数据放入DRAM，默认情况下，常量数据被放入DRRM，DMA无法访问

DRAM\_ATTR static const lcd\_init\_cmd\_t st\_init\_cmds[]={

{0x36, {(1<<5)|(1<<6)}, 1},

{0x3A, {0x55}, 1},

{0xB2, {0x0c, 0x0c, 0x00, 0x33, 0x33}, 5},

{0xB7, {0x45}, 1},

{0xBB, {0x2B}, 1},

{0xC0, {0x2C}, 1},

{0xC2, {0x01, 0xff}, 2},

{0xC3, {0x11}, 1},

{0xC4, {0x20}, 1},

{0xC6, {0x0f}, 1},

{0xD0, {0xA4, 0xA1}, 1},

{0xE0, {0xD0, 0x00, 0x05, 0x0E, 0x15, 0x0D, 0x37, 0x43, 0x47, 0x09, 0x15, 0x12, 0x16, 0x19}, 14},

{0xE1, {0xD0, 0x00, 0x05, 0x0D, 0x0C, 0x06, 0x2D, 0x44, 0x40, 0x0E, 0x1C, 0x18, 0x16, 0x19}, 14},

{0x11, {0}, 0x80},

{0x29, {0}, 0x80},

{0, {0}, 0xff}

};

DRAM\_ATTR static const lcd\_init\_cmd\_t ili\_init\_cmds[]={ //初始化命令

{0xCF, {0x00, 0x83, 0X30}, 3},

{0xED, {0x64, 0x03, 0X12, 0X81}, 4},

{0xE8, {0x85, 0x01, 0x79}, 3},

{0xCB, {0x39, 0x2C, 0x00, 0x34, 0x02}, 5},

{0xF7, {0x20}, 1},

{0xEA, {0x00, 0x00}, 2},

{0xC0, {0x26}, 1},

{0xC1, {0x11}, 1},

{0xC5, {0x35, 0x3E}, 2},

{0xC7, {0xBE}, 1},

{0x36, {0x28}, 1},

{0x3A, {0x55}, 1},

{0xB1, {0x00, 0x1B}, 2},

{0xF2, {0x08}, 1},

{0x26, {0x01}, 1},

{0xE0, {0x1F, 0x1A, 0x18, 0x0A, 0x0F, 0x06, 0x45, 0X87, 0x32, 0x0A, 0x07, 0x02, 0x07, 0x05, 0x00}, 15},

{0XE1, {0x00, 0x25, 0x27, 0x05, 0x10, 0x09, 0x3A, 0x78, 0x4D, 0x05, 0x18, 0x0D, 0x38, 0x3A, 0x1F}, 15},

{0x2A, {0x00, 0x00, 0x00, 0xEF}, 4},

{0x2B, {0x00, 0x00, 0x01, 0x3f}, 4},

{0x2C, {0}, 0},

{0xB7, {0x07}, 1},

{0xB6, {0x0A, 0x82, 0x27, 0x00}, 4},

{0x11, {0}, 0x80},

{0x29, {0}, 0x80},

{0, {0}, 0xff},

};

void lcd\_cmd(spi\_device\_handle\_t spi, const uint8\_t cmd) //发送命令

{

esp\_err\_t ret;

spi\_transaction\_t t;

memset(&t, 0, sizeof(t)); //清零

t.length=8; //8位命令

t.tx\_buffer=&cmd; //数据即命令本身

t.user=(void\*)0; //D/C需要置零

ret=spi\_device\_polling\_transmit(spi, &t); //发送命令

assert(ret==ESP\_OK); //

}

void lcd\_data(spi\_device\_handle\_t spi, const uint8\_t \*data, int len) //发送数据

{

esp\_err\_t ret;

spi\_transaction\_t t;

if (len==0) return; //不需要发送任何数据

memset(&t, 0, sizeof(t)); //清零

t.length=len\*8; //数据长度为字节，交易的长度为比特

t.tx\_buffer=data; //数据

t.user=(void\*)1; //D/C 需要置1

ret=spi\_device\_polling\_transmit(spi, &t); //发送

assert(ret==ESP\_OK);

}

//函数在传输开始之前被调用（在IRQ上下文中），它会将 D/C设置为用户字段中指示的值

void lcd\_spi\_pre\_transfer\_callback(spi\_transaction\_t \*t)

{

int dc=(int)t->user;

gpio\_set\_level(PIN\_NUM\_DC, dc);

}

uint32\_t lcd\_get\_id(spi\_device\_handle\_t spi) //获取设备ID值

{

lcd\_cmd(spi, 0x04);

spi\_transaction\_t t;

memset(&t, 0, sizeof(t));

t.length=8\*3;

t.flags = SPI\_TRANS\_USE\_RXDATA;

t.user = (void\*)1;

esp\_err\_t ret = spi\_device\_polling\_transmit(spi, &t);

assert( ret == ESP\_OK );

return \*(uint32\_t\*)t.rx\_data;

}

void lcd\_init(spi\_device\_handle\_t spi) //初始化显示

{

int cmd=0;

const lcd\_init\_cmd\_t\* lcd\_init\_cmds;

//初始化非SPI GPIO引脚

gpio\_set\_direction(PIN\_NUM\_DC, GPIO\_MODE\_OUTPUT);

gpio\_set\_direction(PIN\_NUM\_RST, GPIO\_MODE\_OUTPUT);

gpio\_set\_direction(PIN\_NUM\_BCKL, GPIO\_MODE\_OUTPUT);

//重置显示

gpio\_set\_level(PIN\_NUM\_RST, 0);

vTaskDelay(100 / portTICK\_RATE\_MS);

gpio\_set\_level(PIN\_NUM\_RST, 1);

vTaskDelay(100 / portTICK\_RATE\_MS);

//检测LCD类型

uint32\_t lcd\_id = lcd\_get\_id(spi);

int lcd\_detected\_type = 0;

int lcd\_type;

printf("LCD ID: %08X\n", lcd\_id);

if ( lcd\_id == 0 ) {

lcd\_detected\_type = LCD\_TYPE\_ILI; //为零

printf("ILI9341 detected.\n");

} else {

lcd\_detected\_type = LCD\_TYPE\_ST; //非零

printf("ST7789V detected.\n");

}

#ifdef CONFIG\_LCD\_TYPE\_AUTO

lcd\_type = lcd\_detected\_type;

#elif defined( CONFIG\_LCD\_TYPE\_ST7789V )

printf("kconfig: force CONFIG\_LCD\_TYPE\_ST7789V.\n");

lcd\_type = LCD\_TYPE\_ST;

#elif defined( CONFIG\_LCD\_TYPE\_ILI9341 )

printf("kconfig: force CONFIG\_LCD\_TYPE\_ILI9341.\n");

lcd\_type = LCD\_TYPE\_ILI;

#endif

if ( lcd\_type == LCD\_TYPE\_ST ) {

printf("LCD ST7789V initialization.\n");

lcd\_init\_cmds = st\_init\_cmds;

} else {

printf("LCD ILI9341 initialization.\n");

lcd\_init\_cmds = ili\_init\_cmds;

}

//发送所有命令

while (lcd\_init\_cmds[cmd].databytes!=0xff) {

lcd\_cmd(spi, lcd\_init\_cmds[cmd].cmd);

lcd\_data(spi, lcd\_init\_cmds[cmd].data, lcd\_init\_cmds[cmd].databytes&0x1F);

if (lcd\_init\_cmds[cmd].databytes&0x80) {

vTaskDelay(100 / portTICK\_RATE\_MS);

}

cmd++;

}

//开启背光

gpio\_set\_level(PIN\_NUM\_BCKL, 0);

}

static void send\_lines(spi\_device\_handle\_t spi, int ypos, uint16\_t \*linedata) //按行发送数据

{

esp\_err\_t ret;

int x;

static spi\_transaction\_t trans[6];

for (x=0; x<6; x++) {

memset(&trans[x], 0, sizeof(spi\_transaction\_t));

if ((x&1)==0) {

//偶数传输是命令

trans[x].length=8;

trans[x].user=(void\*)0;

} else {

//奇数传输是数据

trans[x].length=8\*4;

trans[x].user=(void\*)1;

}

trans[x].flags=SPI\_TRANS\_USE\_TXDATA;

}

trans[0].tx\_data[0]=0x2A; //列地址设置

trans[1].tx\_data[0]=0; //起始点位置

trans[1].tx\_data[1]=0;

trans[1].tx\_data[2]=(320)>>8; //终止点位置

trans[1].tx\_data[3]=(320)&0xff;

trans[2].tx\_data[0]=0x2B; //页面地址设置

trans[3].tx\_data[0]=ypos>>8; //起始点位置

trans[3].tx\_data[1]=ypos&0xff;

trans[3].tx\_data[2]=(ypos+PARALLEL\_LINES)>>8; //终止点位置

trans[3].tx\_data[3]=(ypos+PARALLEL\_LINES)&0xff;

trans[4].tx\_data[0]=0x2C; //写入存储器

trans[5].tx\_buffer=linedata; //发送数据

trans[5].length=320\*2\*8\*PARALLEL\_LINES; //数据按照位长度

trans[5].flags=0; //undo SPI\_TRANS\_USE\_TXDATA flag

//所有交易排队

for (x=0; x<6; x++) {

ret=spi\_device\_queue\_trans(spi, &trans[x], portMAX\_DELAY);

assert(ret==ESP\_OK);

}

}

static void send\_line\_finish(spi\_device\_handle\_t spi) //发送数据结束

{

spi\_transaction\_t \*rtrans;

esp\_err\_t ret;

for (int x=0; x<6; x++) {

ret=spi\_device\_get\_trans\_result(spi, &rtrans, portMAX\_DELAY);

assert(ret==ESP\_OK);

}

}

static void display\_pretty\_colors(spi\_device\_handle\_t spi) //显示图形

{

uint16\_t \*lines[2];

for (int i=0; i<2; i++) {

lines[i]=heap\_caps\_malloc(320\*PARALLEL\_LINES\*sizeof(uint16\_t), MALLOC\_CAP\_DMA);

assert(lines[i]!=NULL);

}

int frame=0;

int sending\_line=-1;

int calc\_line=0;

while(1) {

frame++;

for (int y=0; y<240; y+=PARALLEL\_LINES) {

pretty\_effect\_calc\_lines(lines[calc\_line], y, frame, PARALLEL\_LINES);

if (sending\_line!=-1) send\_line\_finish(spi);

sending\_line=calc\_line;

calc\_line=(calc\_line==1)?0:1;

send\_lines(spi, y, lines[sending\_line]);

}

}

}

void app\_main(void)

{

esp\_err\_t ret;

spi\_device\_handle\_t spi;

spi\_bus\_config\_t buscfg={

.miso\_io\_num=PIN\_NUM\_MISO,

.mosi\_io\_num=PIN\_NUM\_MOSI,

.sclk\_io\_num=PIN\_NUM\_CLK,

.quadwp\_io\_num=-1,

.quadhd\_io\_num=-1,

.max\_transfer\_sz=PARALLEL\_LINES\*320\*2+8

};

spi\_device\_interface\_config\_t devcfg={

#ifdef CONFIG\_LCD\_OVERCLOCK

.clock\_speed\_hz=26\*1000\*1000, //时钟为26MHz

#else

.clock\_speed\_hz=10\*1000\*1000, //时钟为10MHz

#endif

.mode=0, //SPI模式0

.spics\_io\_num=PIN\_NUM\_CS, //CS引脚

.queue\_size=7, //队列为7

.pre\_cb=lcd\_spi\_pre\_transfer\_callback,

};

ret=spi\_bus\_initialize(LCD\_HOST, &buscfg, SPI\_DMA\_CH\_AUTO); //初始化SPI

ESP\_ERROR\_CHECK(ret);

ret=spi\_bus\_add\_device(LCD\_HOST, &devcfg, &spi); //附着LCD到SPI总线

ESP\_ERROR\_CHECK(ret);

lcd\_init(spi); //初始化LCD

ret=pretty\_effect\_init(); //初始化显示效果

ESP\_ERROR\_CHECK(ret);

display\_pretty\_colors(spi);

}

2. Arduino开发环境

本示例实现ESP32开发板与SD进行通信、SD卡的信息显示与相关操作，连接引脚如表5-7所示。

表5-7 ESP32开发板与SD卡引脚连线

|  |  |
| --- | --- |
| ESP32开发板 | SD卡 |
| 5V/3.3V | VCC |
| GPIO 5 | CS |
| GPIO18 | SCK |
| GPIO19 | MISO |
| GPIO23 | MOSI |
| GND | GND |

代码如下：

#include "FS.h"

#include "SD.h"

#include "SPI.h"

void listDir(fs::FS &fs, const char \* dirname, uint8\_t levels){ //列出文件夹

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();

}

}

void createDir(fs::FS &fs, const char \* path){ //新建文件夹

Serial.printf("Creating Dir: %s\n", path);

if(fs.mkdir(path)){

Serial.println("Dir created");

} else {

Serial.println("mkdir failed");

}

}

void removeDir(fs::FS &fs, const char \* path){ //删除文件夹

Serial.printf("Removing Dir: %s\n", path);

if(fs.rmdir(path)){

Serial.println("Dir removed");

} else {

Serial.println("rmdir failed");

}

}

void readFile(fs::FS &fs, const char \* path){ //读取文件

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()){

Serial.write(file.read());

}

file.close();

}

void writeFile(fs::FS &fs, const char \* path, const char \* message){ //写入文件

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();

}

void appendFile(fs::FS &fs, const char \* path, const char \* message){ //增加写入文件

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();

}

void renameFile(fs::FS &fs, const char \* path1, const char \* path2){ //重命名文件

Serial.printf("Renaming file %s to %s\n", path1, path2);

if (fs.rename(path1, path2)) {

Serial.println("File renamed");

} else {

Serial.println("Rename failed");

}

}

void deleteFile(fs::FS &fs, const char \* path){ //删除文件

Serial.printf("Deleting file: %s\n", path);

if(fs.remove(path)){

Serial.println("File deleted");

} else {

Serial.println("Delete failed");

}

}

void testFileIO(fs::FS &fs, const char \* path){ //测试SD卡功能

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()){

Serial.println("Card Mount Failed");

return;

}

uint8\_t cardType = SD.cardType();

if(cardType == CARD\_NONE){

Serial.println("No SD card attached");

return;

}

Serial.print("SD Card Type: "); //打印SD卡类型

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");

}

uint64\_t cardSize = SD.cardSize() / (1024 \* 1024);

Serial.printf("SD Card Size: %lluMB\n", cardSize); //打印SD卡容量

listDir(SD, "/", 0);

createDir(SD, "/mydir");

listDir(SD, "/", 0);

removeDir(SD, "/mydir");

listDir(SD, "/", 2);

writeFile(SD, "/hello.txt", "Hello ");

appendFile(SD, "/hello.txt", "World!\n");

readFile(SD, "/hello.txt");

deleteFile(SD, "/foo.txt");

renameFile(SD, "/hello.txt", "/foo.txt");

readFile(SD, "/foo.txt");

testFileIO(SD, "/test.txt");

Serial.printf("Total space: %lluMB\n", SD.totalBytes() / (1024 \* 1024));

Serial.printf("Used space: %lluMB\n", SD.usedBytes() / (1024 \* 1024));

}

void loop() {}

3. MicroPython开发环境

本示例实现ESP32开发板与OLED进行通信、屏幕信息显示，连接引脚如表5-8所示。代码如下：

表5-7 ESP32开发板与SD卡引脚连线

|  |  |
| --- | --- |
| ESP32开发板 | OLED |
| 3.3V | VCC |
| GPIO22 | CS |
| GPIO19 | SCK/D0 |
| GPIO5 | MISO/未使用 |
| GPIO23 | MOSI/D1 |
| GPIO21 | DC |
| GPIO18 | RES |
| GND | GND |

ssd1306.py

# MicroPython SSD1306 OLED驱动, SPI接口

from micropython import const

import framebuf

#寄存器定义

SET\_CONTRAST = const(0x81)

SET\_ENTIRE\_ON = const(0xA4)

SET\_NORM\_INV = const(0xA6)

SET\_DISP = const(0xAE)

SET\_MEM\_ADDR = const(0x20)

SET\_COL\_ADDR = const(0x21)

SET\_PAGE\_ADDR = const(0x22)

SET\_DISP\_START\_LINE = const(0x40)

SET\_SEG\_REMAP = const(0xA0)

SET\_MUX\_RATIO = const(0xA8)

SET\_IREF\_SELECT = const(0xAD)

SET\_COM\_OUT\_DIR = const(0xC0)

SET\_DISP\_OFFSET = const(0xD3)

SET\_COM\_PIN\_CFG = const(0xDA)

SET\_DISP\_CLK\_DIV = const(0xD5)

SET\_PRECHARGE = const(0xD9)

SET\_VCOM\_DESEL = const(0xDB)

SET\_CHARGE\_PUMP = const(0x8D)

#FrameBuffer提供对图形基元的支持

# http://docs.micropython.org/en/latest/pyboard/library/framebuf.html

class SSD1306(framebuf.FrameBuffer):

def \_\_init\_\_(self, width, height, external\_vcc):

self.width = width

self.height = height

self.external\_vcc = external\_vcc

self.pages = self.height // 8

self.buffer = bytearray(self.pages \* self.width)

super().\_\_init\_\_(self.buffer, self.width, self.height, framebuf.MONO\_VLSB)

self.init\_display()

def init\_display(self):

for cmd in (

SET\_DISP, # display off

#设置地址

SET\_MEM\_ADDR,

0x00, # 水平

#分辨率和布局

SET\_DISP\_START\_LINE, #从零行开始

SET\_SEG\_REMAP | 0x01, #列地址127映射为SEG0

SET\_MUX\_RATIO,

self.height - 1,

SET\_COM\_OUT\_DIR | 0x08, #从COM[N]到COM0扫描

SET\_DISP\_OFFSET,

0x00,

SET\_COM\_PIN\_CFG,

0x02 if self.width > 2 \* self.height else 0x12,

#定时和驱动方案

SET\_DISP\_CLK\_DIV,

0x80,

SET\_PRECHARGE,

0x22 if self.external\_vcc else 0xF1,

SET\_VCOM\_DESEL,

0x30, # 0.83\*Vcc

#显示

SET\_CONTRAST,

0xFF, # maximum

SET\_ENTIRE\_ON, #跟随 RAM 内容输出

SET\_NORM\_INV, #不翻转

SET\_IREF\_SELECT,

0x30, #在显示期间启用内部 IREF

SET\_CHARGE\_PUMP,

0x10 if self.external\_vcc else 0x14,

SET\_DISP | 0x01, #开启显示

):

self.write\_cmd(cmd)

self.fill(0)

self.show()

def poweroff(self):

self.write\_cmd(SET\_DISP)

def poweron(self):

self.write\_cmd(SET\_DISP | 0x01)

def contrast(self, contrast):

self.write\_cmd(SET\_CONTRAST)

self.write\_cmd(contrast)

def invert(self, invert):

self.write\_cmd(SET\_NORM\_INV | (invert & 1))

def rotate(self, rotate):

self.write\_cmd(SET\_COM\_OUT\_DIR | ((rotate & 1) << 3))

self.write\_cmd(SET\_SEG\_REMAP | (rotate & 1))

def show(self):

x0 = 0

x1 = self.width - 1

if self.width != 128:

# narrow displays use centred columns

col\_offset = (128 - self.width) // 2

x0 += col\_offset

x1 += col\_offset

self.write\_cmd(SET\_COL\_ADDR)

self.write\_cmd(x0)

self.write\_cmd(x1)

self.write\_cmd(SET\_PAGE\_ADDR)

self.write\_cmd(0)

self.write\_cmd(self.pages - 1)

self.write\_data(self.buffer)

def disp(self, s: str, x: int, y: int, c: int = 1):

print('jjj')

self.text('\*'+s, x, y, c)

class SSD1306\_I2C(SSD1306):

def \_\_init\_\_(self, width, height, i2c, addr=0x3C, external\_vcc=False):

self.i2c = i2c

self.addr = addr

self.temp = bytearray(2)

self.write\_list = [b"\x40", None] # Co=0, D/C#=1

super().\_\_init\_\_(width, height, external\_vcc)

def write\_cmd(self, cmd):

self.temp[0] = 0x80 # Co=1, D/C#=0

self.temp[1] = cmd

self.i2c.writeto(self.addr, self.temp)

def write\_data(self, buf):

self.write\_list[1] = buf

self.i2c.writevto(self.addr, self.write\_list)

class SSD1306\_SPI(SSD1306):

def \_\_init\_\_(self, width, height, spi, dc, res, cs, external\_vcc=False):

self.rate = 10 \* 1024 \* 1024

dc.init(dc.OUT, value=0)

res.init(res.OUT, value=0)

cs.init(cs.OUT, value=1)

self.spi = spi

self.dc = dc

self.res = res

self.cs = cs

import time

self.res(1)

time.sleep\_ms(1)

self.res(0)

time.sleep\_ms(10)

self.res(1)

super().\_\_init\_\_(width, height, external\_vcc)

def write\_cmd(self, cmd):

self.spi.init(baudrate=self.rate, polarity=0, phase=0)

self.cs(1)

self.dc(0)

self.cs(0)

self.spi.write(bytearray([cmd]))

self.cs(1)

def write\_data(self, buf):

self.spi.init(baudrate=self.rate, polarity=0, phase=0)

self.cs(1)

self.dc(1)

self.cs(0)

self.spi.write(buf)

self.cs(1)

main.py文件代码如下：

import machine

from machine import Pin,SoftSPI

import time

from ssd1306 import SSD1306\_SPI

spi = SoftSPI(baudrate=80000000, polarity=0, phase=0, sck=Pin(19,Pin.OUT), mosi=Pin(23,Pin.OUT), miso=Pin(5)) #sck(D0)=19 mosi(D1)=23 miso=unused

oled = SSD1306\_SPI(128, 64, spi, Pin(21),Pin(18), Pin(22)) #21=dc 18=res 22=cs

oled.text('Hello, World 1!', 0, 0)

oled.text('Hello, World 2!', 0, 10)

oled.text('Hello, World 3!', 0, 20)

oled.show()

### 6.2.3 设置WiFi的STA模式示例

1. 基于ESP IDF的VS Code开发环境实现

代码如下：

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/event\_groups.h"

#include "esp\_system.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "lwip/err.h"

#include "lwip/sys.h"

/\*ESP32的配置，可以修改\*/

#define EXAMPLE\_ESP\_WIFI\_SSID CONFIG\_ESP\_WIFI\_SSID //SSID

#define EXAMPLE\_ESP\_WIFI\_PASS CONFIG\_ESP\_WIFI\_PASSWORD //密码

#define EXAMPLE\_ESP\_MAXIMUM\_RETRY CONFIG\_ESP\_MAXIMUM\_RETRY //最大重试数

/\*FreeRTOS事件组在连接时发出信号，事件组允许每个事件使用多个位，但是本程序只关心两个事件：已通过IP连接到AP；重试次数达到最多后，无法连接\*/

#define WIFI\_CONNECTED\_BIT BIT0

#define WIFI\_FAIL\_BIT BIT1

static const char \*TAG = "wifi station"; //字符串

static int s\_retry\_num = 0; //连接次数变量

static void event\_handler(void\* arg, esp\_event\_base\_t event\_base, int32\_t event\_id, void\* event\_data)

//WiFi事件句柄

{

if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_START) {

esp\_wifi\_connect();

} else if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_DISCONNECTED) {

if (s\_retry\_num < EXAMPLE\_ESP\_MAXIMUM\_RETRY) {

esp\_wifi\_connect(); //开始连接

s\_retry\_num++; //次数累加

ESP\_LOGI(TAG, "retry to connect to the AP"); //打印重连信息

} else {

xEventGroupSetBits(s\_wifi\_event\_group, WIFI\_FAIL\_BIT);

}

ESP\_LOGI(TAG,"connect to the AP fail"); //连接失败信息

} else if (event\_base == IP\_EVENT && event\_id == IP\_EVENT\_STA\_GOT\_IP) {

ip\_event\_got\_ip\_t\* event = (ip\_event\_got\_ip\_t\*) event\_data;

ESP\_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip\_info.ip)); //IP信息

s\_retry\_num = 0;

xEventGroupSetBits(s\_wifi\_event\_group, WIFI\_CONNECTED\_BIT);

}

}

void wifi\_init\_sta(void) //WiFi站点初始化

{

s\_wifi\_event\_group = xEventGroupCreate();

ESP\_ERROR\_CHECK(esp\_netif\_init());

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());

esp\_netif\_create\_default\_wifi\_sta();

wifi\_init\_config\_t cfg = WIFI\_INIT\_CONFIG\_DEFAULT();

ESP\_ERROR\_CHECK(esp\_wifi\_init(&cfg));

esp\_event\_handler\_instance\_t instance\_any\_id;

esp\_event\_handler\_instance\_t instance\_got\_ip;

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_register(WIFI\_EVENT, ESP\_EVENT\_ANY\_ID, &event\_handler, NULL, &instance\_any\_id));

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_register(IP\_EVENT, IP\_EVENT\_STA\_GOT\_IP, &event\_handler, NULL, &instance\_got\_ip));

wifi\_config\_t wifi\_config = { //WiFi配置

.sta = {

.ssid = EXAMPLE\_ESP\_WIFI\_SSID,

.password = EXAMPLE\_ESP\_WIFI\_PASS,

.threshold.authmode = WIFI\_AUTH\_WPA2\_PSK,

.pmf\_cfg = {

.capable = true,

.required = false

},

},

};

ESP\_ERROR\_CHECK(esp\_wifi\_set\_mode(WIFI\_MODE\_STA) ); //站点模式

ESP\_ERROR\_CHECK(esp\_wifi\_set\_config(ESP\_IF\_WIFI\_STA, &wifi\_config) );

ESP\_ERROR\_CHECK(esp\_wifi\_start() );

ESP\_LOGI(TAG, "wifi\_init\_sta finished.");

/\*等待连接成功或失败事件\*/

EventBits\_t bits = xEventGroupWaitBits(s\_wifi\_event\_group,

WIFI\_CONNECTED\_BIT | WIFI\_FAIL\_BIT,

pdFALSE,

pdFALSE,

portMAX\_DELAY);

/\*xEventGroupWaitBits()返回调用之前的比特，可以测试实际发生的事件\*/

if (bits & WIFI\_CONNECTED\_BIT) { //成功打印信息

ESP\_LOGI(TAG, "connected to ap SSID:%s password:%s",

EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS);

} else if (bits & WIFI\_FAIL\_BIT) { //失败打印信息

ESP\_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",

EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS);

} else {

ESP\_LOGE(TAG, "UNEXPECTED EVENT");

}

/\*注销后将不处理该事件\*/

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_unregister(IP\_EVENT, IP\_EVENT\_STA\_GOT\_IP, instance\_got\_ip));

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_unregister(WIFI\_EVENT, ESP\_EVENT\_ANY\_ID, instance\_any\_id));

vEventGroupDelete(s\_wifi\_event\_group);

}

void app\_main(void) //主程序入口

{

esp\_err\_t ret = nvs\_flash\_init(); //初始化

if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND) {

ESP\_ERROR\_CHECK(nvs\_flash\_erase());

ret = nvs\_flash\_init();

}

ESP\_ERROR\_CHECK(ret);

ESP\_LOGI(TAG, "ESP\_WIFI\_MODE\_STA");

wifi\_init\_sta();

}

2. Arduino开发环境实现

代码如下：

#include <Arduino.h>

#include "WiFi.h"

void setup()

{

Serial.begin(115200);

WiFi.begin("myssid", "mypassword");

WiFi.setAutoReconnect(true);

}

void loop()

{

Serial.print("是否连接:");

Serial.println(WiFi.isConnected());

Serial.print("本地IP:");

Serial.println(WiFi.localIP());

Serial.print("本地IPv6:");

Serial.println(WiFi.localIPv6());

Serial.print("mac地址:");

Serial.println(WiFi.macAddress());

Serial.print("休息:");

Serial.println(WiFi.getSleep());

Serial.print("获取状态码:");

Serial.println(WiFi.getStatusBits());

Serial.print("getTxPower:");

Serial.println(WiFi.getTxPower());

Serial.print("是否自动连接:");

Serial.println(WiFi.getAutoConnect());

Serial.print("是否自动重连:");

Serial.println(WiFi.getAutoReconnect());

Serial.print("获取模式:");

Serial.println(WiFi.getMode());

Serial.print("获取主机名:");

Serial.println(WiFi.getHostname());

Serial.print("获取网关IP:");

Serial.println(WiFi.gatewayIP());

Serial.print("dnsIP:");

Serial.println(WiFi.dnsIP());

Serial.print("状态:");

Serial.println(WiFi.status());

delay(1000);

}

3. MicroPython开发环境实现

作为站点STA连接Wi-Fi基站，代码如下：

import network

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

if not wlan.isconnected():

print('connecting to network...')

wlan.connect('testAP', '12345678') #连接到AP

#'SSID'： WiFi账号名

#'PASSWORD'：WiFi密码

while not wlan.isconnected():

pass

print('network config:', wlan.ifconfig())

### 6.2.4 扫描AP示例

1. 基于ESP IDF的VS Code开发环境实现

代码如下：

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/event\_groups.h"

#include "esp\_wifi.h"

#include "esp\_log.h"

#include "esp\_event.h"

#include "nvs\_flash.h"

#define DEFAULT\_SCAN\_LIST\_SIZE CONFIG\_EXAMPLE\_SCAN\_LIST\_SIZE //10个

static const char \*TAG = "scan";

static void print\_auth\_mode(int authmode) //打印不同的认证模式

{

switch (authmode) {

case WIFI\_AUTH\_OPEN:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_OPEN");

break;

case WIFI\_AUTH\_WEP:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_WEP");

break;

case WIFI\_AUTH\_WPA\_PSK:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_WPA\_PSK");

break;

case WIFI\_AUTH\_WPA2\_PSK:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_WPA2\_PSK");

break;

case WIFI\_AUTH\_WPA\_WPA2\_PSK:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_WPA\_WPA2\_PSK");

break;

case WIFI\_AUTH\_WPA2\_ENTERPRISE:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_WPA2\_ENTERPRISE");

break;

case WIFI\_AUTH\_WPA3\_PSK:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_WPA3\_PSK");

break;

case WIFI\_AUTH\_WPA2\_WPA3\_PSK:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_WPA2\_WPA3\_PSK");

break;

default:

ESP\_LOGI(TAG, "Authmode \tWIFI\_AUTH\_UNKNOWN");

break;

}

}

static void print\_cipher\_type(int pairwise\_cipher, int group\_cipher) //打印不同的加密方式

{

switch (pairwise\_cipher) {

case WIFI\_CIPHER\_TYPE\_NONE:

ESP\_LOGI(TAG, "Pairwise Cipher \tWIFI\_CIPHER\_TYPE\_NONE");

break;

case WIFI\_CIPHER\_TYPE\_WEP40:

ESP\_LOGI(TAG, "Pairwise Cipher \tWIFI\_CIPHER\_TYPE\_WEP40");

break;

case WIFI\_CIPHER\_TYPE\_WEP104:

ESP\_LOGI(TAG, "Pairwise Cipher \tWIFI\_CIPHER\_TYPE\_WEP104");

break;

case WIFI\_CIPHER\_TYPE\_TKIP:

ESP\_LOGI(TAG, "Pairwise Cipher \tWIFI\_CIPHER\_TYPE\_TKIP");

break;

case WIFI\_CIPHER\_TYPE\_CCMP:

ESP\_LOGI(TAG, "Pairwise Cipher \tWIFI\_CIPHER\_TYPE\_CCMP");

break;

case WIFI\_CIPHER\_TYPE\_TKIP\_CCMP:

ESP\_LOGI(TAG, "Pairwise Cipher \tWIFI\_CIPHER\_TYPE\_TKIP\_CCMP");

break;

default:

ESP\_LOGI(TAG, "Pairwise Cipher \tWIFI\_CIPHER\_TYPE\_UNKNOWN");

break;

}

switch (group\_cipher) {

case WIFI\_CIPHER\_TYPE\_NONE:

ESP\_LOGI(TAG, "Group Cipher \tWIFI\_CIPHER\_TYPE\_NONE");

break;

case WIFI\_CIPHER\_TYPE\_WEP40:

ESP\_LOGI(TAG, "Group Cipher \tWIFI\_CIPHER\_TYPE\_WEP40");

break;

case WIFI\_CIPHER\_TYPE\_WEP104:

ESP\_LOGI(TAG, "Group Cipher \tWIFI\_CIPHER\_TYPE\_WEP104");

break;

case WIFI\_CIPHER\_TYPE\_TKIP:

ESP\_LOGI(TAG, "Group Cipher \tWIFI\_CIPHER\_TYPE\_TKIP");

break;

case WIFI\_CIPHER\_TYPE\_CCMP:

ESP\_LOGI(TAG, "Group Cipher \tWIFI\_CIPHER\_TYPE\_CCMP");

break;

case WIFI\_CIPHER\_TYPE\_TKIP\_CCMP:

ESP\_LOGI(TAG, "Group Cipher \tWIFI\_CIPHER\_TYPE\_TKIP\_CCMP");

break;

default:

ESP\_LOGI(TAG, "Group Cipher \tWIFI\_CIPHER\_TYPE\_UNKNOWN");

break;

}

}

static void wifi\_scan(void) //初始化WiFi作为站点STA并指定扫描方法

{

ESP\_ERROR\_CHECK(esp\_netif\_init()); //网络接口初始化

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default()); //创建事件

esp\_netif\_t \*sta\_netif = esp\_netif\_create\_default\_wifi\_sta(); //创建WiFi站点

assert(sta\_netif); //初始化站点

wifi\_init\_config\_t cfg = WIFI\_INIT\_CONFIG\_DEFAULT(); //WiFi初始化配置

ESP\_ERROR\_CHECK(esp\_wifi\_init(&cfg));

uint16\_t number = DEFAULT\_SCAN\_LIST\_SIZE; //最大扫描数量

wifi\_ap\_record\_t ap\_info[DEFAULT\_SCAN\_LIST\_SIZE];

uint16\_t ap\_count = 0;

memset(ap\_info, 0, sizeof(ap\_info)); //存储信息

ESP\_ERROR\_CHECK(esp\_wifi\_set\_mode(WIFI\_MODE\_STA)); //WiFi模式设定

ESP\_ERROR\_CHECK(esp\_wifi\_start()); //启动WiFi

ESP\_ERROR\_CHECK(esp\_wifi\_scan\_start(NULL, true)); //开始扫描

ESP\_ERROR\_CHECK(esp\_wifi\_scan\_get\_ap\_records(&number, ap\_info)); //记录

ESP\_ERROR\_CHECK(esp\_wifi\_scan\_get\_ap\_num(&ap\_count)); //数量累计

ESP\_LOGI(TAG, "Total APs scanned = %u", ap\_count); //打印信息

for (int i = 0; (i < DEFAULT\_SCAN\_LIST\_SIZE) && (i < ap\_count); i++) {

ESP\_LOGI(TAG, "SSID \t\t%s", ap\_info[i].ssid); //打印10个AP信息

ESP\_LOGI(TAG, "RSSI \t\t%d", ap\_info[i].rssi);

print\_auth\_mode(ap\_info[i].authmode);

if (ap\_info[i].authmode != WIFI\_AUTH\_WEP) {

print\_cipher\_type(ap\_info[i].pairwise\_cipher, ap\_info[i].group\_cipher);

}

ESP\_LOGI(TAG, "Channel \t\t%d\n", ap\_info[i].primary);

}

}

void app\_main(void) //主程序入口

{

esp\_err\_t ret = nvs\_flash\_init(); //初始化闪存NVS

if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND) {

ESP\_ERROR\_CHECK(nvs\_flash\_erase());

ret = nvs\_flash\_init();

}

ESP\_ERROR\_CHECK( ret );

wifi\_scan(); //开启扫描

}

2. Arduino开发环境实现

代码如下：

#include "WiFi.h"

void setup()

{

Serial.begin(115200);

//设置Wi-Fi站点模式，并断开连接的AP

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

Serial.println("Setup done");

}

void loop()

{

Serial.println("scan start");

//返回发现的Wi-Fi数量

int n = WiFi.scanNetworks();

Serial.println("scan done");

if (n == 0) {

Serial.println("no networks found");

} else {

Serial.print(n);

Serial.println(" networks found");

for (int i = 0; i < n; ++i) {

//打印SSID和RSSI

Serial.print(i + 1);

Serial.print(": ");

Serial.print(WiFi.SSID(i));

Serial.print(" (");

Serial.print(WiFi.RSSI(i));

Serial.print(")");

Serial.println((WiFi.encryptionType(i) == WIFI\_AUTH\_OPEN)?" ":"\*");

delay(10);

}

}

Serial.println("");

delay(5000); //5s后重新扫描

}

3. MicroPython开发环境实现

代码如下：

import network, ubinascii

from utime import sleep\_ms

wlan = network.WLAN(network.STA\_IF) #创建站点接口

wlan.active(True) #激活接口

wlan\_list = wlan.scan() #扫描接入点

print('ssid\t\t通道 RSSI authmode 是否隐藏 bssid')

for i in wlan\_list:

sleep\_ms(10)

print(i[0].decode() + ('' if len(i[0].decode()) > 6 else '\t') + '\t' + str(i[2]) + '\t' + str(i[3]) + '\t' + str(i[4]) + '\t' + str(i[5]) + '\t' + ubinascii.hexlify(i[1]).decode())

'''

返回包含WiFi接入点信息的元组：(ssid, bssid, 通道, RSSI, authmode, 是否隐藏)

bssid是访问点的硬件地址，以二进制形式，返回字节对象。可以使用ubinascii.hexlify()将其转换为ASCII格式，authmode有5个值：0–开放网络; 1–WEP; 2–WPA-PSK; 3–WPA2-PSK; 4–WPA/WPA2-PSK

'''

### 6.3.2 智能配置WiFi示例

1. 基于ESP IDF的VS Code开发环境实现

开发板的程序代码如下：

#include <string.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/event\_groups.h"

#include "esp\_wifi.h"

#include "esp\_wpa2.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "esp\_system.h"

#include "nvs\_flash.h"

#include "esp\_netif.h"

#include "esp\_smartconfig.h"

/\*FreeRTOS事件组在连接并准备发出请求时发出信号\*/

static EventGroupHandle\_t s\_wifi\_event\_group;

/\*事件组允许每个事件使用多个比特位\*/

static const int CONNECTED\_BIT = BIT0; //连接比特位

static const int ESPTOUCH\_DONE\_BIT = BIT1; //ESPTOUCH比特位

static const char \*TAG = "smartconfig\_example"; //定义字符串

static void smartconfig\_example\_task(void \* parm); //定义任务

static void event\_handler(void\* arg, esp\_event\_base\_t event\_base, int32\_t event\_id, void\* event\_data)

//事件句柄

{

if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_START) {

xTaskCreate(smartconfig\_example\_task, "smartconfig\_example\_task", 4096, NULL, 3, NULL);

//创建任务

} else if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_DISCONNECTED) {

esp\_wifi\_connect(); //WiFi连接

xEventGroupClearBits(s\_wifi\_event\_group, CONNECTED\_BIT); //清除比特位

} else if (event\_base == IP\_EVENT && event\_id == IP\_EVENT\_STA\_GOT\_IP) {

xEventGroupSetBits(s\_wifi\_event\_group, CONNECTED\_BIT); //设置比特位

} else if (event\_base == SC\_EVENT && event\_id == SC\_EVENT\_SCAN\_DONE) {

ESP\_LOGI(TAG, "Scan done"); //打印扫描完毕信息

} else if (event\_base == SC\_EVENT && event\_id == SC\_EVENT\_FOUND\_CHANNEL) {

ESP\_LOGI(TAG, "Found channel"); //打印发现信道

} else if (event\_base == SC\_EVENT && event\_id == SC\_EVENT\_GOT\_SSID\_PSWD) {

ESP\_LOGI(TAG, "Got SSID and password"); //获取SSID和密码

smartconfig\_event\_got\_ssid\_pswd\_t \*evt = (smartconfig\_event\_got\_ssid\_pswd\_t \*)event\_data; //获取SSID和密码的类型配置指针

wifi\_config\_t wifi\_config; //WiFi配置

uint8\_t ssid[33] = { 0 };

uint8\_t password[65] = { 0 };

bzero(&wifi\_config, sizeof(wifi\_config\_t));

memcpy(wifi\_config.sta.ssid, evt->ssid, sizeof(wifi\_config.sta.ssid)); //拷贝SSID

memcpy(wifi\_config.sta.password, evt->password, sizeof(wifi\_config.sta.password)); //拷贝密码

wifi\_config.sta.bssid\_set = evt->bssid\_set; //设置SSID

if (wifi\_config.sta.bssid\_set == true) {

memcpy(wifi\_config.sta.bssid, evt->bssid, sizeof(wifi\_config.sta.bssid));

}

memcpy(ssid, evt->ssid, sizeof(evt->ssid)); //拷贝SSID

memcpy(password, evt->password, sizeof(evt->password)); //拷贝密码

ESP\_LOGI(TAG, "SSID:%s", ssid); //打印信息

ESP\_LOGI(TAG, "PASSWORD:%s", password);

ESP\_ERROR\_CHECK( esp\_wifi\_disconnect() );

ESP\_ERROR\_CHECK(esp\_wifi\_set\_config(ESP\_IF\_WIFI\_STA, &WiFi\_config) );

ESP\_ERROR\_CHECK( esp\_wifi\_connect() ); //检测连接

} else if (event\_base == SC\_EVENT && event\_id == SC\_EVENT\_SEND\_ACK\_DONE) {

xEventGroupSetBits(s\_wifi\_event\_group, ESPTOUCH\_DONE\_BIT);//比特置位

}

}

static void initialise\_wifi(void) //初始化WiFi

{

ESP\_ERROR\_CHECK(esp\_netif\_init()); //网络接口初始化

s\_wifi\_event\_group = xEventGroupCreate(); //创建事件组

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default()); //默认方式创建

esp\_netif\_t \*sta\_netif = esp\_netif\_create\_default\_wifi\_sta(); //创建WiFi站点

assert(sta\_netif); //确认站点网络接口

wifi\_init\_config\_t cfg = WIFI\_INIT\_CONFIG\_DEFAULT(); //以默认值初始化

ESP\_ERROR\_CHECK( esp\_wifi\_init(&cfg) ); //确认WiFi初始化配置

ESP\_ERROR\_CHECK( esp\_event\_handler\_register(WIFI\_EVENT, ESP\_EVENT\_ANY\_ID, &event\_handler, NULL) ); //WiFi事件句柄注册

ESP\_ERROR\_CHECK( esp\_event\_handler\_register(IP\_EVENT, IP\_EVENT\_STA\_GOT\_IP, &event\_handler, NULL) ); //IP事件句柄注册

ESP\_ERROR\_CHECK( esp\_event\_handler\_register(SC\_EVENT, ESP\_EVENT\_ANY\_ID, &event\_handler, NULL) ); //智能配置事件句柄注册

ESP\_ERROR\_CHECK( esp\_wifi\_set\_mode(WIFI\_MODE\_STA) ); //设置WiFi模式

ESP\_ERROR\_CHECK( esp\_wifi\_start() ); //启动WiFi

}

static void smartconfig\_example\_task(void \* parm) //创建智能配置任务

{

EventBits\_t uxBits;

ESP\_ERROR\_CHECK( esp\_smartconfig\_set\_type(SC\_TYPE\_ESPTOUCH) ); //设置类型

smartconfig\_start\_config\_t cfg = SMARTCONFIG\_START\_CONFIG\_DEFAULT();

ESP\_ERROR\_CHECK( esp\_smartconfig\_start(&cfg) ); //启动配置

while (1) {

uxBits = xEventGroupWaitBits(s\_wifi\_event\_group, CONNECTED\_BIT | ESPTOUCH\_DONE\_BIT, true, false, portMAX\_DELAY); //事件组比特置位获取

if(uxBits & CONNECTED\_BIT) {

ESP\_LOGI(TAG, "wifi Connected to ap"); //连接成功

}

if(uxBits & ESPTOUCH\_DONE\_BIT) { //ESPTOUCH比特置位

ESP\_LOGI(TAG, "smartconfig over"); //打印配置完成

esp\_smartconfig\_stop(); //停止配置

vTaskDelete(NULL); //删除任务

}

}

}

void app\_main(void) //主程序入口

{

ESP\_ERROR\_CHECK( nvs\_flash\_init() ); //闪存初始化

initialise\_wifi(); //初始化WiFi

}

2. Arduino开发环境实现

代码如下：

#include "WiFi.h"

void setup() {

Serial.begin(115200);

//初始化WiFi为STA模式，开始智能配置

WiFi.mode(WIFI\_AP\_STA);

WiFi.beginSmartConfig();

//等待智能配置

Serial.println("Waiting for SmartConfig.");

while (!WiFi.smartConfigDone()) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("SmartConfig received.");

//等待WiFi连接AP

Serial.println("Waiting for WiFi");

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("WiFi Connected.");

Serial.print("IP Address: ");

Serial.println(WiFi.localIP()); //打印IP地址

}

void loop() {}

3. MicroPython开发环境实现

import network

import socket

import ure

import time

NETWORK\_PROFILES = 'wifi.dat'

wlan\_ap = network.WLAN(network.AP\_IF)

wlan\_sta = network.WLAN(network.STA\_IF)

server\_socket = None

def send\_header(conn, status\_code=200, content\_length=None ):

conn.sendall("HTTP/1.0 {} OK\r\n".format(status\_code))

conn.sendall("Content-Type: text/html\r\n")

if content\_length is not None:

conn.sendall("Content-Length: {}\r\n".format(content\_length))

conn.sendall("\r\n")

def send\_response(conn, payload, status\_code=200):

content\_length = len(payload)

send\_header(conn, status\_code, content\_length)

if content\_length > 0:

conn.sendall(payload)

conn.close()

def config\_page():

return b"""<html>

<head>

<title>MYESP32 AP Test</title>

<meta charset="UTF-8">

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

</head>

<body>

<h1>Wifi 配网</h1>

<form action="configure" method="post">

<div>

<label>SSID</label>

<input type="text" name="ssid">

</div>

<div>

<label>PASSWORD</label>

<input type="password" name="password">

</div>

<input type="submit" value="连接">

<form>

</body>

</html>"""

def wifi\_conf\_page(ssid, passwd):

return b"""<html>

<head>

<title>Wifi Conf Info</title>

<meta charset="UTF-8">

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

</head>

<body>

<h1>Post data:</h1>

<p>SSID: %s</p>

<p>PASSWD: %s</p>

<a href="/">Return Configure Page</a>

</body>

</html>""" % (ssid, passwd)

def connect\_sucess(new\_ip):

return b"""<html>

<head>

<title>Connect Sucess!</title>

<meta charset="UTF-8">

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

</head>

<body>

<p>Wifi Connect Sucess</p>

<p>IP Address: %s</p>

<a href="http://%s">Home</a>

<a href="/disconnect">Disconnect</a>

</body>

</html>""" % (new\_ip, new\_ip)

def get\_wifi\_conf(request):

match = ure.search("ssid=([^&]\*)&password=(.\*)", request)

if match is None:

return False

try:

ssid = match.group(1).decode("utf-8").replace("%3F", "?").replace("%21", "!")

password = match.group(2).decode("utf-8").replace("%3F", "?").replace("%21", "!")

except Exception:

ssid = match.group(1).replace("%3F", "?").replace("%21", "!")

password = match.group(2).replace("%3F", "?").replace("%21", "!")

if len(ssid) == 0:

return False

return (ssid, password)

def handle\_wifi\_configure(ssid, password):

if do\_connect(ssid, password):

new\_ip = wlan\_sta.ifconfig()[0]

return new\_ip

else:

print('connect fail')

return False

def check\_wlan\_connected():

if wlan\_sta.isconnected():

return True

else:

return False

def do\_connect(ssid, password):

wlan\_sta.active(True)

if wlan\_sta.isconnected():

return None

print('Connect to %s' % ssid)

wlan\_sta.connect(ssid, password)

for retry in range(100):

connected = wlan\_sta.isconnected()

if connected:

break

time.sleep(0.1)

print('.', end='')

if connected:

print('\nConnected : ', wlan\_sta.ifconfig())

else:

print('\nFailed. Not Connected to: ' + ssid)

return connected

def read\_profiles():

with open(NETWORK\_PROFILES) as f:

lines = f.readlines()

profiles = {}

for line in lines:

ssid, password = line.strip("\n").split(";")

profiles[ssid] = password

return profiles

def write\_profiles(profiles):

lines = []

for ssid, password in profiles.items():

lines.append("%s;%s\n" % (ssid, password))

with open(NETWORK\_PROFILES, "w") as f:

f.write(''.join(lines))

def stop():

global server\_socket

if server\_socket:

server\_socket.close()

server\_socket = None

def startAP():

global server\_socket

stop()

wlan\_ap.active(True)

wlan\_ap.config(essid='ESP32',authmode=0)

server\_socket = socket.socket()

server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

server\_socket.bind(('0.0.0.0', 80))

server\_socket.listen(3)

while not wlan\_sta.isconnected():

conn, addr = server\_socket.accept()

print('Connection: %s ' % str(addr))

try:

conn.settimeout(3)

request = b""

try:

while "\r\n\r\n" not in request:

request += conn.recv(512)

except OSError:

pass

# url process

try:

url = ure.search("(?:GET|POST) /(.\*?)(?:\\?.\*?)? HTTP", request).group(1).decode("utf-8").rstrip("/")

except Exception:

url = ure.search("(?:GET|POST) /(.\*?)(?:\\?.\*?)? HTTP", request).group(1).rstrip("/")

print("URL is {}".format(url))

if url == "":

response = config\_page()

send\_response(conn, response)

elif url == "configure":

ret = get\_wifi\_conf(request)

ret = handle\_wifi\_configure(ret[0], ret[1])

if ret is not None:

response = connect\_sucess(ret)

send\_response(conn, response)

print('connect sucess')

elif url == "disconnect":

wlan\_sta.disconnect()

finally:

conn.close()

wlan\_ap.active(False)

print('ap exit')

def home():

global server\_socket

stop()

wlan\_sta.active(True)

ip\_addr = wlan\_sta.ifconfig()[0]

print('wifi connected')

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

server\_socket.bind(('0.0.0.0', 80))

server\_socket.listen(3)

while check\_wlan\_connected():

conn, addr = server\_socket.accept()

try:

conn.settimeout(3)

request = b""

try:

while "\r\n\r\n" not in request:

request += conn.recv(512)

except OSError:

pass

# url process

try:

url = ure.search("(?:GET|POST) /(.\*?)(?:\\?.\*?)? HTTP", request).group(1).decode("utf-8").rstrip("/")

except Exception:

url = ure.search("(?:GET|POST) /(.\*?)(?:\\?.\*?)? HTTP", request).group(1).rstrip("/")

if url == "":

response = connect\_sucess(ip\_addr)

send\_response(conn, response)

elif url == "disconnect":

wlan\_sta.disconnect()

finally:

conn.close()

wlan\_sta.active(False)

print('sta exit')

def main():

while True:

if not check\_wlan\_connected():

startAP()

else:

home()

main()

### 6.4.3 基于TCP的Socket通信示例

1.基于ESP IDF的VS Code开发环境客户端

代码如下：

#include <string.h>

#include <sys/param.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/event\_groups.h"

#include "esp\_system.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "esp\_netif.h"

#include "protocol\_examples\_common.h"

#include "addr\_from\_stdin.h"

#include "lwip/err.h"

#include "lwip/sockets.h"

#if defined(CONFIG\_EXAMPLE\_IPV4) //IPV4地址，本例使用

#define HOST\_IP\_ADDR CONFIG\_EXAMPLE\_IPV4\_ADDR

#elif defined(CONFIG\_EXAMPLE\_IPV6) //IPV6地址

#define HOST\_IP\_ADDR CONFIG\_EXAMPLE\_IPV6\_ADDR

#else

#define HOST\_IP\_ADDR " " //其他情况地址

#endif

#define PORT CONFIG\_EXAMPLE\_PORT //端口

static const char \*TAG = "example";

static const char \*payload = "Message from ESP32 "; //连接成功发送的信息

static void tcp\_client\_task(void \*pvParameters) //任务定义

{

char rx\_buffer[128]; //接收字符缓存

char host\_ip[] = HOST\_IP\_ADDR; //存放主机IP地址

int addr\_family = 0; //地址组

int ip\_protocol = 0; //IP协议

while (1) {

#if defined(CONFIG\_EXAMPLE\_IPV4) //IPV4情况

struct sockaddr\_in dest\_addr; //Socket地址结构体

dest\_addr.sin\_addr.s\_addr = inet\_addr(host\_ip); //IP地址

dest\_addr.sin\_family = AF\_INET; //地址组

dest\_addr.sin\_port = htons(PORT); //端口

addr\_family = AF\_INET; //地址组

ip\_protocol = IPPROTO\_IP; //IP协议

#elif defined(CONFIG\_EXAMPLE\_IPV6) //IPV6情况

struct sockaddr\_in6 dest\_addr = { 0 };

inet6\_aton(host\_ip, &dest\_addr.sin6\_addr);

dest\_addr.sin6\_family = AF\_INET6;

dest\_addr.sin6\_port = htons(PORT);

dest\_addr.sin6\_scope\_id = esp\_netif\_get\_netif\_impl\_index(EXAMPLE\_INTERFACE);

addr\_family = AF\_INET6;

ip\_protocol = IPPROTO\_IPV6;

#elif defined(CONFIG\_EXAMPLE\_SOCKET\_IP\_INPUT\_STDIN) //其他情况

struct sockaddr\_in6 dest\_addr = { 0 };

ESP\_ERROR\_CHECK(get\_addr\_from\_stdin(PORT, SOCK\_STREAM, &ip\_protocol, &addr\_family, &dest\_addr));

#endif

int sock = socket(addr\_family, SOCK\_STREAM, ip\_protocol); //创建Socket

if (sock < 0) { //不能创建输出信息

ESP\_LOGE(TAG, "Unable to create socket: errno %d", errno);

break;

}

ESP\_LOGI(TAG, "Socket created, connecting to %s:%d", host\_ip, PORT); //成功

int err = connect(sock, (struct sockaddr \*)&dest\_addr, sizeof(struct sockaddr\_in6));

if (err != 0) { //连接失败信息

ESP\_LOGE(TAG, "Socket unable to connect: errno %d", errno);

break;

}

ESP\_LOGI(TAG, "Successfully connected"); //成功

while (1) {

int err = send(sock, payload, strlen(payload), 0); //发送消息

if (err < 0) { //发送失败

ESP\_LOGE(TAG, "Error occurred during sending: errno %d", errno);

break;

}

int len = recv(sock, rx\_buffer, sizeof(rx\_buffer) - 1, 0); //接收消息

if (len < 0) { //错误情况

ESP\_LOGE(TAG, "recv failed: errno %d", errno);

break;

}

// Data received

else {

rx\_buffer[len] = 0; //接收字符串

ESP\_LOGI(TAG, "Received %d bytes from %s:", len, host\_ip);

ESP\_LOGI(TAG, "%s", rx\_buffer); //输出字符串

}

vTaskDelay(2000 / portTICK\_PERIOD\_MS); //延迟2S

}

if (sock != -1) { //关闭Socket

ESP\_LOGE(TAG, "Shutting down socket and restarting...");

shutdown(sock, 0);

close(sock);

}

}

vTaskDelete(NULL); //删除任务

}

void app\_main(void) //主函数入口

{

ESP\_ERROR\_CHECK(nvs\_flash\_init()); //闪存初始化

ESP\_ERROR\_CHECK(esp\_netif\_init()); //网络接口初始化

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default()); //创建默认循环任务

ESP\_ERROR\_CHECK(example\_connect()); //开始连接

xTaskCreate(tcp\_client\_task, "tcp\_client", 4096, NULL, 5, NULL); //创建任务

}

2. Arduino开发环境客户端实现

代码如下：

#include <WiFi.h>

const char\* ssid = "your-ssid"; //连接的WiFi名称

const char\* password = "your-password"; //连接的WiFi密码

const char\* host = " your-host"; //连接的主机服务器地址

const char\* streamId = "....................";

const char\* privateKey = "....................";

void setup()

{

Serial.begin(115200);

delay(10);

//连接WiFi

Serial.println();

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password); //连接WiFi

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected"); //输出WiFi连接信息

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

}

int value = 0;

void loop()

{

delay(5000);

++value;

Serial.print("connecting to ");

Serial.println(host);

//使用WiFiClient类创建TCP连接

WiFiClient client;

const int httpPort = 80; //使用主机的80端口

if (!client.connect(host, httpPort)) {

Serial.println("connection failed");

return;

}

//为请求创建一个URI

String url = "/input/";

url += streamId;

url += "?private\_key=";

url += privateKey;

url += "&value=";

url += value;

Serial.print("Requesting URL: ");

Serial.println(url);

//向服务器端发送请求

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

unsigned long timeout = millis();

while (client.available() == 0) {

if (millis() - timeout > 5000) {

Serial.println(">>> Client Timeout !");

client.stop();

return;

}

}

//从服务器端读取回复的所有行，并打印到串口

while(client.available()) {

String line = client.readStringUntil('\r');

Serial.print(line);

}

Serial.println();

Serial.println("closing connection");

}

3.MicroPython开发环境客户端实现

代码如下：

import network

import socket

import time

SSID="yourSSID" #修改为使用的WiFi名称

PASSWORD="yourPASSWD" #修改为使用WiFi的密码

host="yourHOST" #修改为主机服务器地址

port=100 #主机服务器端口定义

wlan=None

s=None

def connectWifi(ssid,passwd):

global wlan

wlan=network.WLAN(network.STA\_IF) #实例化WLAN

wlan.active(True) #激活网络接口

wlan.disconnect() #断开上一连接

wlan.connect(ssid,passwd) #连接WiFi

while(wlan.ifconfig()[0]=='0.0.0.0'):

time.sleep(1)

return True

#捕获异常，如果在“try”中意外中断，则停止程序

try:

connectWifi(SSID,PASSWORD)

ip=wlan.ifconfig()[0] #获取IP地址

s = socket.socket() #创建Socket

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

#设置给定Socket选项的值

s.connect((host,port)) #发送连接请求

s.send("hello, I am TCP Client") #发送数据

while True:

data = s.recv(1024) #从Socket接收1024字节的数据

if(len(data) == 0): #如果无数据，关闭

print("close socket")

s.close()

break

print(data) #输出数据

ret = s.send(data)

except: #异常处理

if (s):

s.close()

wlan.disconnect()

wlan.active(False)

4.基于ESP IDF的VS Code开发环境服务器端实现

代码如下：

#include <string.h>

#include <sys/param.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "esp\_netif.h"

#include "protocol\_examples\_common.h"

#include "lwip/err.h"

#include "lwip/sockets.h"

#include "lwip/sys.h"

#include <lwip/netdb.h>

#define PORT CONFIG\_EXAMPLE\_PORT //TCP服务器端口

static const char \*TAG = "example";

static void do\_retransmit(const int sock) //重传字符串函数定义

{

int len;

char rx\_buffer[128];

do {

len = recv(sock, rx\_buffer, sizeof(rx\_buffer) - 1, 0);

if (len < 0) { //字符串发生错误打印信息

ESP\_LOGE(TAG, "Error occurred during receiving: errno %d", errno);

} else if (len == 0) { //连接关闭

ESP\_LOGW(TAG, "Connection closed");

} else { //打印输出的字符串

rx\_buffer[len] = 0;

ESP\_LOGI(TAG, "Received %d bytes: %s", len, rx\_buffer);

int to\_write = len;

while (to\_write > 0) { //发送收到的字符串

int written = send(sock, rx\_buffer + (len - to\_write), to\_write, 0);

if (written < 0) {

ESP\_LOGE(TAG, "Error occurred during sending: errno %d", errno);

}

to\_write -= written;

}

}

} while (len > 0);

}

static void tcp\_server\_task(void \*pvParameters) //创建TCP服务器端任务

{

char addr\_str[128];

int addr\_family = (int)pvParameters;

int ip\_protocol = 0;

struct sockaddr\_in6 dest\_addr; //构建Socket结构体

if (addr\_family == AF\_INET) { //IPV4地址

struct sockaddr\_in \*dest\_addr\_ip4 = (struct sockaddr\_in \*)&dest\_addr;

dest\_addr\_ip4->sin\_addr.s\_addr = htonl(INADDR\_ANY);

dest\_addr\_ip4->sin\_family = AF\_INET;

dest\_addr\_ip4->sin\_port = htons(PORT);

ip\_protocol = IPPROTO\_IP;

} else if (addr\_family == AF\_INET6) { //IPV6地址

bzero(&dest\_addr.sin6\_addr.un, sizeof(dest\_addr.sin6\_addr.un));

dest\_addr.sin6\_family = AF\_INET6;

dest\_addr.sin6\_port = htons(PORT);

ip\_protocol = IPPROTO\_IPV6;

}

int listen\_sock = socket(addr\_family, SOCK\_STREAM, ip\_protocol); //创建Socket

if (listen\_sock < 0) {

ESP\_LOGE(TAG, "Unable to create socket: errno %d", errno); //未成功

vTaskDelete(NULL);

return;

}

#if defined(CONFIG\_EXAMPLE\_IPV4) && defined(CONFIG\_EXAMPLE\_IPV6)

//注意，默认情况下，IPV6绑定到这两个协议

int opt = 1;

setsockopt(listen\_sock, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt));

setsockopt(listen\_sock, IPPROTO\_IPV6, IPV6\_V6ONLY, &opt, sizeof(opt));

#endif

ESP\_LOGI(TAG, "Socket created"); //Socket创建成功

int err = bind(listen\_sock, (struct sockaddr \*)&dest\_addr, sizeof(dest\_addr)); //绑定IP

if (err != 0) {

ESP\_LOGE(TAG, "Socket unable to bind: errno %d", errno); //绑定未成功

ESP\_LOGE(TAG, "IPPROTO: %d", addr\_family);

goto CLEAN\_UP; //跳转

}

ESP\_LOGI(TAG, "Socket bound, port %d", PORT); //绑定，打印端口

err = listen(listen\_sock, 1);

if (err != 0) {

ESP\_LOGE(TAG, "Error occurred during listen: errno %d", errno); //监听出错

goto CLEAN\_UP; //跳转

}

while (1) {

ESP\_LOGI(TAG, "Socket listening"); //一直监听，等待客户端连接

struct sockaddr\_in6 source\_addr; // Socket的IP地址，足够大IPv4、IPv6

uint addr\_len = sizeof(source\_addr);

int sock = accept(listen\_sock, (struct sockaddr \*)&source\_addr, &addr\_len);//接收接入

if (sock < 0) { //不能接入，打印报错信息

ESP\_LOGE(TAG, "Unable to accept connection: errno %d", errno);

break;

}

//将IP地址转换为字符串

if (source\_addr.sin6\_family == PF\_INET) { //IPV4

inet\_ntoa\_r(((struct sockaddr\_in \*)&source\_addr)->sin\_addr.s\_addr, addr\_str, sizeof(addr\_str) - 1);

} else if (source\_addr.sin6\_family == PF\_INET6) { //IPV6

inet6\_ntoa\_r(source\_addr.sin6\_addr, addr\_str, sizeof(addr\_str) - 1);

}

ESP\_LOGI(TAG, "Socket accepted ip address: %s", addr\_str); //打印客户端IP地址

do\_retransmit(sock); //重传字符串

shutdown(sock, 0);

close(sock);

}

CLEAN\_UP: //跳转点

close(listen\_sock); //关闭监听，删除任务

vTaskDelete(NULL);

}

void app\_main(void) //主函数入口

{

ESP\_ERROR\_CHECK(nvs\_flash\_init()); //初始化闪存

ESP\_ERROR\_CHECK(esp\_netif\_init()); //初始化网络接口

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default()); //创建循环默认事件

ESP\_ERROR\_CHECK(example\_connect()); //开启连接

#ifdef CONFIG\_EXAMPLE\_IPV4 //IPV4任务创建

xTaskCreate(tcp\_server\_task, "tcp\_server", 4096, (void\*)AF\_INET, 5, NULL);

#endif

#ifdef CONFIG\_EXAMPLE\_IPV6 //IPV6任务创建

xTaskCreate(tcp\_server\_task, "tcp\_server", 4096, (void\*)AF\_INET6, 5, NULL);

#endif

}

5.Arduino开发环境实现服务器

#include <WiFi.h>

const char\* ssid = "yourssid"; //连接的WiFi名称

const char\* password = "yourpasswd"; //连接的WiFi密码

WiFiServer server(80); //服务器定义

void setup()

{

Serial.begin(115200);

pinMode(5, OUTPUT); //设置LED引脚

delay(10);

//连接WiFi

Serial.println();

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected.");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

server.begin();

}

int value = 0;

void loop(){

WiFiClient client = server.available(); //监听传入的客户端

if (client) { //如果监听到一个客户端

Serial.println("New Client."); //从串口监视器打印一条消息

String currentLine = ""; //创建一个字符串以保存来自客户端的传入数据

while (client.connected()) { //连接到客户端时进行循环

if (client.available()) { //如果要从客户端读取数据

char c = client.read(); //读取1个字节

Serial.write(c); //在串口监视器中打印

if (c == '\n') { //如果读取的字节是换行符

//如果当前行为空，则一行中有两个换行符

//客户端HTTP请求到此结束，发送响应

if (currentLine.length() == 0) {

//HTTP头总是以响应代码（例如HTTP/1.1 200 OK）和内容类型开头，以便客户端知道接下来会发生什么，然后是一个空行：

client.println("HTTP/1.1 200 OK");

client.println("Content-type:text/html");

client.println();

//HTTP响应的内容如下：

client.print("Click <a href=\"/H\">here</a> to turn the LED on pin 5 on.<br>");

client.print("Click <a href=\"/L\">here</a> to turn the LED on pin 5 off.<br>");

//HTTP响应以另一个空行结束：

client.println();

//跳出循环

break;

} else { //如果有换行符，则清除currentLine：

currentLine = "";

}

} else if (c != '\r') { //如果没有回车符

currentLine += c; //将其添加到当前行的末尾

}

//检查客户端请求是“GET/H”还是“GET/L”：

if (currentLine.endsWith("GET /H")) {

digitalWrite(5, HIGH); //请求为“GET/H”，打开LED

}

if (currentLine.endsWith("GET /L")) {

digitalWrite(5, LOW); //请求为“GET/L”，关闭LED

}

}

}

//停止连接

client.stop();

Serial.println("Client Disconnected.");

}

}

6. MicroPython开发环境实现服务器

通过连接目标WiFi，为ESP32开发板分配IP地址，打印该IP以便客户端能够连接，代码如下：

import network

import socket

import time

SSID=" " #连接的目标WiFi名称

PASSWORD=" " #连接的目标WiFi密码

port=10000 #使用的端口

wlan=None

listenSocket=None

def connectWifi(ssid,passwd): #定义WiFi连接函数

global wlan

wlan=network.WLAN(network.STA\_IF) #实例化WLAN

wlan.active(True) #激活网络接口

wlan.disconnect() #断开上一连接

wlan.connect(ssid,passwd) #连接WiFi

while(wlan.ifconfig()[0]=='0.0.0.0'):

time.sleep(1)

return True

#捕获异常，如果在“try”中意外中断，则停止程序

try:

connectWifi(SSID,PASSWORD)

ip=wlan.ifconfig()[0] #获取IP地址

listenSocket = socket.socket() #创建Socket

listenSocket.bind((ip,port)) #绑定IP和端口

listenSocket.listen(1) #监听消息

listenSocket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

#设置给定Socket选项的值

print(ip) #打印分配的IP，以便客户端连接

print ('tcp waiting...')

while True:

print("accepting.....")

conn,addr = listenSocket.accept() #接收连接，conn是一个新的Socket对象

print(addr,"connected")

while True:

data = conn.recv(1024) #从Socket接收1024字节的数据

if(len(data) == 0):

print("close socket")

conn.close() #如果没有数据，关闭

break

print(data)

ret = conn.send(data) #发送数据

except:

if(listenSocket):

listenSocket.close()

wlan.disconnect()

wlan.active(False)

### 6.4.4 基于UDP的Socket通信示例

代码如下：

#include <string.h>

#include <sys/param.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/event\_groups.h"

#include "esp\_system.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "esp\_netif.h"

#include "protocol\_examples\_common.h"

#include "lwip/err.h"

#include "lwip/sockets.h"

#include "lwip/sys.h"

#include <lwip/netdb.h>

#include "addr\_from\_stdin.h"

#if defined(CONFIG\_EXAMPLE\_IPV4)

#define HOST\_IP\_ADDR CONFIG\_EXAMPLE\_IPV4\_ADDR //IPV4

#elif defined(CONFIG\_EXAMPLE\_IPV6)

#define HOST\_IP\_ADDR CONFIG\_EXAMPLE\_IPV6\_ADDR //IPV6

#else

#define HOST\_IP\_ADDR " " //其他

#endif

#define PORT CONFIG\_EXAMPLE\_PORT //端口定义

static const char \*TAG = "example";

static const char \*payload = "Message from ESP32 "; //连接后发送信息

static void udp\_client\_task(void \*pvParameters) //构建UDP客户端任务

{

char rx\_buffer[128];

char host\_ip[] = HOST\_IP\_ADDR;

int addr\_family = 0;

int ip\_protocol = 0;

while (1) {

#if defined(CONFIG\_EXAMPLE\_IPV4) //IPV4

struct sockaddr\_in dest\_addr; //Socket结构体定义

dest\_addr.sin\_addr.s\_addr = inet\_addr(HOST\_IP\_ADDR);

dest\_addr.sin\_family = AF\_INET; //地址组

dest\_addr.sin\_port = htons(PORT); //端口

addr\_family = AF\_INET; //地址组

ip\_protocol = IPPROTO\_IP; //IP地址

#elif defined(CONFIG\_EXAMPLE\_IPV6) //IPV6

struct sockaddr\_in6 dest\_addr = { 0 };

inet6\_aton(HOST\_IP\_ADDR, &dest\_addr.sin6\_addr);

dest\_addr.sin6\_family = AF\_INET6;

dest\_addr.sin6\_port = htons(PORT);

dest\_addr.sin6\_scope\_id = esp\_netif\_get\_netif\_impl\_index(EXAMPLE\_INTERFACE);

addr\_family = AF\_INET6;

ip\_protocol = IPPROTO\_IPV6;

#elif defined(CONFIG\_EXAMPLE\_SOCKET\_IP\_INPUT\_STDIN) //输入流

struct sockaddr\_in6 dest\_addr = { 0 };

ESP\_ERROR\_CHECK(get\_addr\_from\_stdin(PORT, SOCK\_DGRAM, &ip\_protocol, &addr\_family, &dest\_addr)); //初始化

#endif

int sock = socket(addr\_family, SOCK\_DGRAM, ip\_protocol); //创建Socket

if (sock < 0) { //如果错误，打印信息

ESP\_LOGE(TAG, "Unable to create socket: errno %d", errno);

break;

}

ESP\_LOGI(TAG, "Socket created, sending to %s:%d", HOST\_IP\_ADDR, PORT);

while (1) {

int err = sendto(sock, payload, strlen(payload), 0, (struct sockaddr \*)&dest\_addr, sizeof(dest\_addr)); //发送定义的信息

if (err < 0) { //如果错误，打印信息

ESP\_LOGE(TAG, "Error occurred during sending: errno %d", errno);

break;

}

ESP\_LOGI(TAG, "Message sent"); //成功

struct sockaddr\_in source\_addr; //Socket的IPV4或V6地址

socklen\_t socklen = sizeof(source\_addr); //获取IP地址长度

int len = recvfrom(sock, rx\_buffer, sizeof(rx\_buffer) - 1, 0, (struct sockaddr \*)&source\_addr, &socklen); //从已连接Socket上接收数据，并捕获数据发送源的地址

if (len < 0) { //如果错误，打印信息

ESP\_LOGE(TAG, "recvfrom failed: errno %d", errno);

break;

}

else { //接收数据

rx\_buffer[len] = 0; //字符串接收缓冲区

ESP\_LOGI(TAG, "Received %d bytes from %s:", len, host\_ip); //打印信息

ESP\_LOGI(TAG, "%s", rx\_buffer);

if (strncmp(rx\_buffer, "OK: ", 4) == 0) { //如果输入“OK: ”，重新连接

ESP\_LOGI(TAG, "Received expected message, reconnecting");

break;

}

}

vTaskDelay(2000 / portTICK\_PERIOD\_MS);

}

if (sock != -1) { //关闭并重启

ESP\_LOGE(TAG, "Shutting down socket and restarting...");

shutdown(sock, 0);

close(sock);

}

}

vTaskDelete(NULL); //删除任务

}

void app\_main(void) //主程序入口

{

ESP\_ERROR\_CHECK(nvs\_flash\_init()); //闪存初始化

ESP\_ERROR\_CHECK(esp\_netif\_init()); //网络接口初始化

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default()); //创建默认循环任务

ESP\_ERROR\_CHECK(example\_connect()); //连接

xTaskCreate(udp\_client\_task, "udp\_client", 4096, NULL, 5, NULL); //创建任务

}

2. Arduino开发环境实现客户端

代码如下：

#include <WiFi.h>

#include <WiFiUdp.h>

//使用的Wi-Fi名称和密码，服务器IP地址和端口号

const char \* networkName = "your-ssid";

const char \* networkPswd = "your-password";

const char \* udpAddress = "192.168.0.255";

const int udpPort = 3333;

boolean connected = false; //当前是否连接

WiFiUDP udp; //UDP实例

void setup(){

Serial.begin(115200); //初始化串口

connectToWiFi(networkName, networkPswd); //连接到WiFi路由器

}

void loop(){

//连接成功发送数据

if(connected){

udp.beginPacket(udpAddress,udpPort);

udp.printf("Seconds since boot: %lu", millis()/1000);

udp.endPacket();

}

delay(1000);

}

void connectToWiFi(const char \* ssid, const char \* pwd){

Serial.println("Connecting to WiFi network: " + String(ssid)); //删除旧的配置

WiFi.disconnect(true);

WiFi.onEvent(WiFiEvent); //注册事件句柄

WiFi.begin(ssid, pwd); //初始化连接

Serial.println("Waiting for WIFI connection...");

}

void WiFiEvent(WiFiEvent\_t event){ //Wi-Fi事件句柄

switch(event) {

case ARDUINO\_EVENT\_WIFI\_STA\_GOT\_IP: //连接完成

Serial.print("WiFi connected! IP address: ");

Serial.println(WiFi.localIP());

udp.begin(WiFi.localIP(),udpPort); //初始化UDP和传输缓冲区

connected = true;

break;

case ARDUINO\_EVENT\_WIFI\_STA\_DISCONNECTED:

Serial.println("WiFi lost connection");

connected = false;

break;

default: break;

}

}

3. MicroPython开发环境实现客户端

import socket

import network

import time

host=' yourHOST ' #目标主机服务器端地址

port = 80 #目标主机服务器端口

SSID="yourSSID" #目标WiFi名称

PASSWORD="yourPASSWD" #目标WiFi密码

wlan=None

s=None

def connectWifi(ssid,passwd):

global wlan

wlan=network.WLAN(network.STA\_IF) #创建WLAN对象

wlan.active(True) #激活网络接口

wlan.disconnect() #断开上次连接的WiFi

wlan.connect(ssid,passwd) #连接WiFi

while(wlan.ifconfig()[0]=='0.0.0.0'):

time.sleep(1)

return True

try:

if(connectWifi(SSID,PASSWORD) == True): #判断是否连接WiFi

s=socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) #创建Socket

s.setsockopt(socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1)

#设置给定Socket选项的值

ip=wlan.ifconfig()[0] #获取IP地址

while True:

s.sendto(b'hello\r\n',(host,port)) #连续发送数据hello

time.sleep(1)

except:

if (s):

s.close()

wlan.disconnect()

wlan.active(False)

4. 基于ESP IDF的VS Code开发环境服务器端实现

代码如下：

#include <string.h>

#include <sys/param.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_WiFi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "esp\_netif.h"

#include "protocol\_examples\_common.h"

#include "lwip/err.h"

#include "lwip/sockets.h"

#include "lwip/sys.h"

#include <lwip/netdb.h>

#define PORT CONFIG\_EXAMPLE\_PORT //端口定义

static const char \*TAG = "example";

static void udp\_server\_task(void \*pvParameters) //UDP服务器任务定义

{

char rx\_buffer[128]; //接收缓冲区

char addr\_str[128]; //地址字符串

int addr\_family = (int)pvParameters; //地址组

int ip\_protocol = 0; //IP协议

struct sockaddr\_in6 dest\_addr; //目的地址

while (1) {

if (addr\_family == AF\_INET) { //IPV4协议

struct sockaddr\_in \*dest\_addr\_ip4 = (struct sockaddr\_in \*)&dest\_addr;

dest\_addr\_ip4->sin\_addr.s\_addr = htonl(INADDR\_ANY);

dest\_addr\_ip4->sin\_family = AF\_INET;

dest\_addr\_ip4->sin\_port = htons(PORT);

ip\_protocol = IPPROTO\_IP;

} else if (addr\_family == AF\_INET6) { //IPV6协议

bzero(&dest\_addr.sin6\_addr.un, sizeof(dest\_addr.sin6\_addr.un));

dest\_addr.sin6\_family = AF\_INET6;

dest\_addr.sin6\_port = htons(PORT);

ip\_protocol = IPPROTO\_IPV6;

}

int sock = socket(addr\_family, SOCK\_DGRAM, ip\_protocol); //构建Socket

if (sock < 0) { //如果错误，打印信息

ESP\_LOGE(TAG, "Unable to create socket: errno %d", errno);

break;

}

ESP\_LOGI(TAG, "Socket created"); //Socket创建成功

#if defined(CONFIG\_EXAMPLE\_IPV4) && defined(CONFIG\_EXAMPLE\_IPV6) //IPV6协议

if (addr\_family == AF\_INET6) {

//请注意，默认情况下，IPV6绑定到这两个协议

int opt = 1;

setsockopt(sock, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt));

setsockopt(sock, IPPROTO\_IPV6, IPV6\_V6ONLY, &opt, sizeof(opt));

}

#endif

int err = bind(sock, (struct sockaddr \*)&dest\_addr, sizeof(dest\_addr)); //绑定地址

if (err < 0) { //如果错误，打印信息

ESP\_LOGE(TAG, "Socket unable to bind: errno %d", errno);

}

ESP\_LOGI(TAG, "Socket bound, port %d", PORT); //成功绑定

while (1) {

ESP\_LOGI(TAG, "Waiting for data"); //作为服务器等待客户端消息

struct sockaddr\_in6 source\_addr; //源地址

socklen\_t socklen = sizeof(source\_addr); //地址长度

int len = recvfrom(sock, rx\_buffer, sizeof(rx\_buffer) - 1, 0, (struct sockaddr \*)&source\_addr, &socklen); //接收数据

if (len < 0) { //如果错误，打印信息

ESP\_LOGE(TAG, "recvfrom failed: errno %d", errno);

break;

}

else { //接收数据，获取客户端的IP作为字符串

if (source\_addr.sin6\_family == PF\_INET) {

inet\_ntoa\_r(((struct sockaddr\_in \*)&source\_addr)->sin\_addr.s\_addr, addr\_str, sizeof(addr\_str) - 1);

} else if (source\_addr.sin6\_family == PF\_INET6) { // IPV6协议

inet6\_ntoa\_r(source\_addr.sin6\_addr, addr\_str, sizeof(addr\_str) - 1);

}

rx\_buffer[len] = 0; //作为字符串接收，打印信息

ESP\_LOGI(TAG, "Received %d bytes from %s:", len, addr\_str);

ESP\_LOGI(TAG, "%s", rx\_buffer);

int err = sendto(sock, rx\_buffer, len, 0, (struct sockaddr \*)&source\_addr, sizeof(source\_addr)); //接收到的字符串发回客户端

if (err < 0) {

ESP\_LOGE(TAG, "Error occurred during sending: errno %d", errno);

break;

}

}

}

if (sock != -1) { //错误则关闭重启

ESP\_LOGE(TAG, "Shutting down socket and restarting...");

shutdown(sock, 0);

close(sock);

}

}

vTaskDelete(NULL); //删除任务

}

void app\_main(void) //主函数入口

{

ESP\_ERROR\_CHECK(nvs\_flash\_init()); //闪存初始化

ESP\_ERROR\_CHECK(esp\_netif\_init()); //网络接口初始化

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default()); //创建默认任务

ESP\_ERROR\_CHECK(example\_connect()); //连接

#ifdef CONFIG\_EXAMPLE\_IPV4 //IPV4

xTaskCreate(udp\_server\_task, "udp\_server", 4096, (void\*)AF\_INET, 5, NULL);

#endif

#ifdef CONFIG\_EXAMPLE\_IPV6 //IPV6

xTaskCreate(udp\_server\_task, "udp\_server", 4096, (void\*)AF\_INET6, 5, NULL);

#endif

}

5. Arduino开发环境服务器端实现

代码如下：

#include <WiFi.h>

#include <WiFiUdp.h> //引用以使用UDP

const char \*ssid = "\*\*\*\*\*\*\*\*";

const char \*password = "\*\*\*\*\*\*\*\*";

WiFiUDP Udp; //建立UDP对象

unsigned int localUdpPort = 2333; //本地端口号

void setup()

{

Serial.begin(115200);

Serial.println();

WiFi.mode(WIFI\_STA);

WiFi.begin(ssid, password);

while (!WiFi.isConnected())

{

delay(500);

Serial.print(".");

}

Serial.println("Connected");

Serial.print("IP Address:");

Serial.println(WiFi.localIP());

Udp.begin(localUdpPort); //启用UDP监听以接收数据

}

void loop()

{

int packetSize = Udp.parsePacket(); //获取当前队首数据包长度

if (packetSize) //若是有数据可用

{

char buf[packetSize];

Udp.read(buf, packetSize); //读取当前包数据

Serial.println();

Serial.print("Received: ");

Serial.println(buf);

Serial.print("From IP: ");

Serial.println(Udp.remoteIP());

Serial.print("From Port: ");

Serial.println(Udp.remotePort());

Udp.beginPacket(Udp.remoteIP(), Udp.remotePort()); //准备发送数据

Udp.print("Received: "); //复制数据到发送缓存

Udp.write((const uint8\_t\*)buf, packetSize); //复制数据到发送缓存

Udp.endPacket(); //发送数据

}

}

6. MicroPython开发环境服务器端实现

代码如下：

import socket

import network

import time

port = 80 #设置参数

SSID="yourSSID"

PASSWORD="yourPASSWD"

wlan=None

s=None

def connectWifi(ssid,passwd):

global wlan

wlan=network.WLAN(network.STA\_IF) #创建WLAN对象

wlan.active(True) #激活网络接口

wlan.disconnect() #断开上次连接的WiFi

wlan.connect(ssid,passwd) #连接WiFi

while(wlan.ifconfig()[0]=='0.0.0.0'):

time.sleep(1)

return True

#捕获异常，如果在“尝试”中意外中断，则停止程序

try:

if(connectWifi(SSID, PASSWORD) == True): #判断是否连接WiFi

s=socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) #创建Socket

s.setsockopt(socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1) #设置给定Socket选项的值

ip=wlan.ifconfig()[0] #获取IP地址

s.bind((ip,port)) #绑定IP和端口

print(ip)

print('waiting...')

while True:

data,addr=s.recvfrom(1024) #从Socket 接收1024字节的数据

print('received:',data,'from',addr) #打印接收到的数据

s.sendto(data,addr) #向addr地址发送数据

except:

if (s):

s.close()

wlan.disconnect() #断开WiFi

wlan.active(False)







### 7.1.2 HTTP服务器端程序示例

1. 基于ESP IDF的VS Code开发环境实现

CONFIG\_EXAMPLE\_WIFI\_SSID="myssid" //设置WiFi名字

CONFIG\_EXAMPLE\_WIFI\_PASSWORD="mypassword" //设置WiFi密码

代码如下：

#include <esp\_wifi.h>

#include <esp\_event.h>

#include <esp\_log.h>

#include <esp\_system.h>

#include <nvs\_flash.h>

#include <sys/param.h>

#include "nvs\_flash.h"

#include "esp\_netif.h"

#include "esp\_eth.h"

#include "protocol\_examples\_common.h"

#include <esp\_http\_server.h>

static const char \*TAG = "example";

static esp\_err\_t hello\_get\_handler(httpd\_req\_t \*req) //HTTP GET句柄

{

char\* buf;

size\_t buf\_len; /\*获取标头长度，并分配长度+1的内存，为终止符添加额外字节\*/

buf\_len = httpd\_req\_get\_hdr\_value\_len(req, "Host") + 1;

if (buf\_len > 1) {

buf = malloc(buf\_len);

/\*将终止符结尾的字符串复制到缓冲区\*/

if (httpd\_req\_get\_hdr\_value\_str(req, "Host", buf, buf\_len) == ESP\_OK) {

ESP\_LOGI(TAG, "Found header => Host: %s", buf); //日志信息输出

}

free(buf);

}

buf\_len = httpd\_req\_get\_hdr\_value\_len(req, "Test-Header-2") + 1; //获取字符串长度

if (buf\_len > 1) {

buf = malloc(buf\_len); /\*将终止符结尾的字符串复制到缓冲区\*/

if (httpd\_req\_get\_hdr\_value\_str(req, "Test-Header-2", buf, buf\_len) == ESP\_OK) {

ESP\_LOGI(TAG, "Found header => Test-Header-2: %s", buf);

}

free(buf);

}

buf\_len = httpd\_req\_get\_hdr\_value\_len(req, "Test-Header-1") + 1; //获取字符串长度

if (buf\_len > 1) {

buf = malloc(buf\_len); /\*将终止符结尾的字符串复制到缓冲区\*/

if (httpd\_req\_get\_hdr\_value\_str(req, "Test-Header-1", buf, buf\_len) == ESP\_OK) {

ESP\_LOGI(TAG, "Found header => Test-Header-1: %s", buf);

}

free(buf);

}

/\*读取URL查询字符串的长度，并分配长度+1的内存，为终止符添加额外字节\*/

buf\_len = httpd\_req\_get\_url\_query\_len(req) + 1;

if (buf\_len > 1) {

buf = malloc(buf\_len); /\*将终止符结尾的字符串复制到缓冲区\*/

if (httpd\_req\_get\_url\_query\_str(req, buf, buf\_len) == ESP\_OK) {

ESP\_LOGI(TAG, "Found URL query => %s", buf);

char param[32];

/\*从查询字符串获取期望键值\*/

if (httpd\_query\_key\_value(buf, "query1", param, sizeof(param)) == ESP\_OK) {

ESP\_LOGI(TAG, "Found URL query parameter => query1=%s", param);

}

if (httpd\_query\_key\_value(buf, "query3", param, sizeof(param)) == ESP\_OK) {

ESP\_LOGI(TAG, "Found URL query parameter => query3=%s", param);

}

if (httpd\_query\_key\_value(buf, "query2", param, sizeof(param)) == ESP\_OK) {

ESP\_LOGI(TAG, "Found URL query parameter => query2=%s", param);

}

}

free(buf);

}

/\*设置定制的标头\*/

httpd\_resp\_set\_hdr(req, "Custom-Header-1", "Custom-Value-1");

httpd\_resp\_set\_hdr(req, "Custom-Header-2", "Custom-Value-2");

/\*带有自定义标头和正文设置为字符串，在用户上下文中传递的发送响应\*/

const char\* resp\_str = (const char\*) req->user\_ctx;

httpd\_resp\_send(req, resp\_str, strlen(resp\_str));

/\*发送响应后，旧的HTTP请求标头将丢失，检查是否可以读取HTTP请求标头\*/

if (httpd\_req\_get\_hdr\_value\_len(req, "Host") == 0) {

ESP\_LOGI(TAG, "Request headers lost");

}

return ESP\_OK;

}

static const httpd\_uri\_t hello = { //针对hello的处理

.uri = "/hello",

.method = HTTP\_GET,

.handler = hello\_get\_handler,

/\*Let's pass response string in user

\*context to demonstrate it's usage\*/

.user\_ctx = "Hello World!"

};

static esp\_err\_t echo\_post\_handler(httpd\_req\_t \*req) /\*HTTP POST句柄\*/

{

char buf[100];

int ret, remaining = req->content\_len;

while (remaining > 0) { /\*读取请求数据\*/

if ((ret = httpd\_req\_recv(req, buf,

MIN(remaining, sizeof(buf)))) <= 0) {

if (ret == HTTPD\_SOCK\_ERR\_TIMEOUT) { /\*超时后重试\*/

continue;

}

return ESP\_FAIL;

}

httpd\_resp\_send\_chunk(req, buf, ret); /\*返回同样的数据\*/

remaining -= ret;

/\*接收日志数据\*/

ESP\_LOGI(TAG, "=========== RECEIVED DATA ==========");

ESP\_LOGI(TAG, "%.\*s", ret, buf);

ESP\_LOGI(TAG, "====================================");

}

httpd\_resp\_send\_chunk(req, NULL, 0); //响应结束

return ESP\_OK;

}

static const httpd\_uri\_t echo = { //echo处理

.uri = "/echo",

.method = HTTP\_POST,

.handler = echo\_post\_handler,

.user\_ctx = NULL

};

esp\_err\_t http\_404\_error\_handler(httpd\_req\_t \*req, httpd\_err\_code\_t err) //错误处理

{

if (strcmp("/hello", req->uri) == 0) { //hello处理

httpd\_resp\_send\_err(req, HTTPD\_404\_NOT\_FOUND, "/hello URI is not available");

/\*返回ESP\_OK以保持底层套接字打开\*/

return ESP\_OK;

} else if (strcmp("/echo", req->uri) == 0) { //echo处理

httpd\_resp\_send\_err(req, HTTPD\_404\_NOT\_FOUND, "/echo URI is not available");

/\*返回ESP\_FAIL以关闭底层套接字\*/

return ESP\_FAIL;

}

/\*对于其他任何URI，发送404并关闭套接字\*/

httpd\_resp\_send\_err(req, HTTPD\_404\_NOT\_FOUND, "Some 404 error message");

return ESP\_FAIL;

}

/\*HTTP PUT处理程序，URI处理程序的实时注册和注销\*/

static esp\_err\_t ctrl\_put\_handler(httpd\_req\_t \*req)

{

char buf;

int ret;

if ((ret = httpd\_req\_recv(req, &buf, 1)) <= 0) {

if (ret == HTTPD\_SOCK\_ERR\_TIMEOUT) {

httpd\_resp\_send\_408(req);

}

return ESP\_FAIL;

}

if (buf == '0') {

/\*可以使用URI字符串注销URI处理程序\*/

ESP\_LOGI(TAG, "Unregistering /hello and /echo URIs");

httpd\_unregister\_uri(req->handle, "/hello");

httpd\_unregister\_uri(req->handle, "/echo");

/\*注册自定义错误处理程序\*/

httpd\_register\_err\_handler(req->handle, HTTPD\_404\_NOT\_FOUND, http\_404\_error\_handler);

}

else { //注册hello和echo的URI

ESP\_LOGI(TAG, "Registering /hello and /echo URIs");

httpd\_register\_uri\_handler(req->handle, &hello);

httpd\_register\_uri\_handler(req->handle, &echo);

/\*取消注册自定义错误处理程序\*/

httpd\_register\_err\_handler(req->handle, HTTPD\_404\_NOT\_FOUND, NULL);

}

httpd\_resp\_send(req, NULL, 0); /\*以空体回应\*/

return ESP\_OK;

}

static const httpd\_uri\_t ctrl = { //ctrl处理

.uri = "/ctrl",

.method = HTTP\_PUT,

.handler = ctrl\_put\_handler,

.user\_ctx = NULL

};

static httpd\_handle\_t start\_webserver(void) //开启网络服务器

{

httpd\_handle\_t server = NULL;

httpd\_config\_t config = HTTPD\_DEFAULT\_CONFIG();

ESP\_LOGI(TAG, "Starting server on port: '%d'", config.server\_port);

if (httpd\_start(&server, &config) == ESP\_OK) { //开启httpd服务器

ESP\_LOGI(TAG, "Registering URI handlers"); //设置URI句柄

httpd\_register\_uri\_handler(server, &hello);

httpd\_register\_uri\_handler(server, &echo);

httpd\_register\_uri\_handler(server, &ctrl);

return server;

}

ESP\_LOGI(TAG, "Error starting server!");

return NULL;

}

static void stop\_webserver(httpd\_handle\_t server) //停止网络服务器端

{

httpd\_stop(server); //停止httpd服务器端

}

static void disconnect\_handler(void\* arg, esp\_event\_base\_t event\_base,

int32\_t event\_id, void\* event\_data)

{ //断开句柄

httpd\_handle\_t\* server = (httpd\_handle\_t\*) arg;

if (\*server) {

ESP\_LOGI(TAG, "Stopping webserver");

stop\_webserver(\*server);

\*server = NULL;

}

}

static void connect\_handler(void\* arg, esp\_event\_base\_t event\_base,

int32\_t event\_id, void\* event\_data)

{ //连接句柄

httpd\_handle\_t\* server = (httpd\_handle\_t\*) arg;

if (\*server == NULL) {

ESP\_LOGI(TAG, "Starting webserver");

\*server = start\_webserver();

}

}

void app\_main(void) //主函数入口

{

static httpd\_handle\_t server = NULL;

ESP\_ERROR\_CHECK(nvs\_flash\_init());

ESP\_ERROR\_CHECK(esp\_netif\_init());

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());

ESP\_ERROR\_CHECK(example\_connect());

/\*注册事件处理程序断开WiFi或以太网时停止服务器端，并在连接后重新启动它\*/

#ifdef CONFIG\_EXAMPLE\_CONNECT\_WIFI //WiFi处理

ESP\_ERROR\_CHECK(esp\_event\_handler\_register(IP\_EVENT, IP\_EVENT\_STA\_GOT\_IP, &connect\_handler, &server));

ESP\_ERROR\_CHECK(esp\_event\_handler\_register(WIFI\_EVENT, WIFI\_EVENT\_STA\_DISCONNECTED, &disconnect\_handler, &server));

#endif // CONFIG\_EXAMPLE\_CONNECT\_WIFI

#ifdef CONFIG\_EXAMPLE\_CONNECT\_ETHERNET //以太网处理

ESP\_ERROR\_CHECK(esp\_event\_handler\_register(IP\_EVENT, IP\_EVENT\_ETH\_GOT\_IP, &connect\_handler, &server));

ESP\_ERROR\_CHECK(esp\_event\_handler\_register(ETH\_EVENT, ETHERNET\_EVENT\_DISCONNECTED, &disconnect\_handler, &server));

#endif // CONFIG\_EXAMPLE\_CONNECT\_ETHERNET

server = start\_webserver(); /\*开启服务器端\*/

}

2. Arduino开发环境实现服务器

一个简单的 Web 服务器，可以通过浏览器控制LED。此程序打印服务器的IP地址到串行监视器，可以在浏览器中访问该地址，打开和关闭引脚5上的LED，代码如下：

#include <WiFi.h>

const char\* ssid = "yourssid";

const char\* password = "yourpasswd";

WiFiServer server(80);

void setup()

{

Serial.begin(115200);

pinMode(5, OUTPUT); //设置LED引脚模式

delay(10);

//开始连接Wi-Fi

Serial.println();

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected.");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

server.begin();

}

int value = 0;

void loop(){

WiFiClient client = server.available(); //监听到来的客户端请求

if (client) { //如果有客户端请求

Serial.println("New Client."); //打印消息到串口监视器

String currentLine = ""; //接收来自客户端的数据

while (client.connected()) { //客户端连接时，循环扫描

if (client.available()) { //如果客户端有数据

char c = client.read(); //读取一个字节然后打印在串口监视器

Serial.write(c);

if (c == '\n') { //如果字节是一个新行的字符

//如果当前行为空白，则连续有两个换行符。 客户端HTTP请求结束，所以发送一个响应

if (currentLine.length() == 0) {

// HTTP头一般开始于响应码 (例如，HTTP/1.1 200 OK)

//和一个内容类型，以便客户端知道接下来会发生什么，然后是一个空行

client.println("HTTP/1.1 200 OK");

client.println("Content-type:text/html");

client.println();

//HTTP 响应的内容跟在头部之后

client.print("Click <a href=\"/H\">here</a> to turn the LED on pin 5 on.<br>");

client.print("Click <a href=\"/L\">here</a> to turn the LED on pin 5 off.<br>");

// HTTP 响应以另一个空行结束

client.println();

//跳出while循环

break;

} else { //如果有换行符，则清除当前行

currentLine = "";

}

} else if (c != '\r') { //如果除了回车符之外还有其他内容，请将其添加到当前行的末尾

currentLine += c;

}

//检查客户端请求是否为"GET /H" or "GET /L"

if (currentLine.endsWith("GET /H")) {

digitalWrite(5, HIGH); // GET /H 则打开LED

}

if (currentLine.endsWith("GET /L")) {

digitalWrite(5, LOW); // GET /L则关闭LED

}

}

}

client.stop(); //关闭连接

Serial.println("Client Disconnected.");

}

}

3. MicroPython开发环境实现服务器

代码如下：

import network

import usocket as socket #引用Socket模块

#响应头

responseHeaders = b'''

HTTP/1.1 200 OK

Content-Type: text/html

Connection: close

'''

#响应头网页正文内容

content = b'''

Hello World!

'''

ap = network.WLAN(network.AP\_IF) #ESP32为AP模式

ap.config(essid='ESP32' , authmode=4 , password='12345678') #设置AP的名称和密码

ap.active(True) # 开启无线热点

def main(): #定义主函数

s = socket.socket()

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1) # (重要)设置端口释放后立即就可以被再次使用

s.bind(socket.getaddrinfo("0.0.0.0", 80)[0][-1]) # 绑定地址

s.listen(5) # 开启监听（最大连接数5）

print('接入热点后可从浏览器访问下面地址：')

print(ap.ifconfig()[0])

print('')

while True: # mian()函数中进行死循环，在这里保持监听浏览器请求与对应处理

client\_sock, client\_addr = s.accept() # 接收来自客户端的请求与客户端地址

print('Client address:', client\_addr)

while True:

h = client\_sock.readline() # 按行读取来自客户端的请求内容

print(h.decode('utf8'), end='')

if h == b'' or h == b'\r\n': #当读取到空行的时候表示接收到完整的请求头

break

client\_sock.write(responseHeaders) # 向客户端发送响应头

client\_sock.write(content) # 向客户端发送网页内容

client\_sock.close()

main() #运行main()函数

### 7.1.4 HTTP客户端请求示例

1. 基于ESP IDF的VS Code开发环境实现

CONFIG\_EXAMPLE\_WIFI\_SSID="myssid" //更换WiFi名字

CONFIG\_EXAMPLE\_WIFI\_PASSWORD="mypassword" //更换WiFi密码

代码如下：

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "protocol\_examples\_common.h"

#include "lwip/err.h"

#include "lwip/sockets.h"

#include "lwip/sys.h"

#include "lwip/netdb.h"

#include "lwip/dns.h"

/\*定义常量\*/

#define WEB\_SERVER "example.com"

#define WEB\_PORT "80"

#define WEB\_PATH "/"

static const char \*TAG = "example";

static const char \*REQUEST = "GET " WEB\_PATH " HTTP/1.0\r\n"

"Host: "WEB\_SERVER":"WEB\_PORT"\r\n"

"User-Agent: esp-idf/1.0 esp32\r\n"

"\r\n";

static void http\_get\_task(void \*pvParameters) //GET方法任务

{

const struct addrinfo hints = {

.ai\_family = AF\_INET,

.ai\_socktype = SOCK\_STREAM,

};

struct addrinfo \*res;

struct in\_addr \*addr;

int s, r;

char recv\_buf[64];

while(1) {

int err = getaddrinfo(WEB\_SERVER, WEB\_PORT, &hints, &res);

if(err != 0 || res == NULL) {

ESP\_LOGE(TAG, "DNS lookup failed err=%d res=%p", err, res);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

continue;

}

/\*打印IP地址\*/

addr = &((struct sockaddr\_in \*)res->ai\_addr)->sin\_addr;

ESP\_LOGI(TAG, "DNS lookup succeeded. IP=%s", inet\_ntoa(\*addr));

s = socket(res->ai\_family, res->ai\_socktype, 0);

if(s < 0) {

ESP\_LOGE(TAG, "... Failed to allocate socket.");

freeaddrinfo(res);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

continue;

}

ESP\_LOGI(TAG, "... allocated socket");

if(connect(s, res->ai\_addr, res->ai\_addrlen) != 0) {

ESP\_LOGE(TAG, "... socket connect failed errno=%d", errno);

close(s);

freeaddrinfo(res);

vTaskDelay(4000 / portTICK\_PERIOD\_MS);

continue;

}

ESP\_LOGI(TAG, "... connected");

freeaddrinfo(res);

if (write(s, REQUEST, strlen(REQUEST)) < 0) {

ESP\_LOGE(TAG, "... socket send failed");

close(s);

vTaskDelay(4000 / portTICK\_PERIOD\_MS);

continue;

}

ESP\_LOGI(TAG, "... socket send success");

struct timeval receiving\_timeout;

receiving\_timeout.tv\_sec = 5;

receiving\_timeout.tv\_usec = 0;

if (setsockopt(s, SOL\_SOCKET, SO\_RCVTIMEO, &receiving\_timeout,

sizeof(receiving\_timeout)) < 0) {

ESP\_LOGE(TAG, "... failed to set socket receiving timeout");

close(s);

vTaskDelay(4000 / portTICK\_PERIOD\_MS);

continue;

}

ESP\_LOGI(TAG, "... set socket receiving timeout success");

/\*读取HTTP响应\*/

do {

bzero(recv\_buf, sizeof(recv\_buf));

r = read(s, recv\_buf, sizeof(recv\_buf)-1);

for(int i = 0; i < r; i++) {

putchar(recv\_buf[i]);

}

} while(r > 0);

ESP\_LOGI(TAG, "... done reading from socket. Last read return=%d errno=%d.", r, errno);

close(s);

for(int countdown = 10; countdown >= 0; countdown--) {

ESP\_LOGI(TAG, "%d... ", countdown);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

ESP\_LOGI(TAG, "Starting again!");

}

}

void app\_main(void)

{

ESP\_ERROR\_CHECK( nvs\_flash\_init() );

ESP\_ERROR\_CHECK(esp\_netif\_init());

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());

ESP\_ERROR\_CHECK(example\_connect());

xTaskCreate(&http\_get\_task, "http\_get\_task", 4096, NULL, 5, NULL);

}

2. Arduino开发环境实现

通过连接目标WiFi，实现远程URL的访问，并打印结果，代码如下：

#include <WiFi.h>

const char\* ssid = " "; //Wi-Fi名称

const char\* password = " "; //Wi-Fi密码

const char\* host = "example.com"; //访问服务器的地址

const char\* streamId = "....................";

const char\* privateKey = "....................";

void setup()

{

Serial.begin(115200);

delay(10);

Serial.println();

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password); //连接Wi-Fi网络并打印IP地址

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

}

int value = 0;

void loop()

{

delay(5000);

++value;

Serial.print("connecting to ");

Serial.println(host);

WiFiClient client; //使用WiFiClient类创建TCP连接

const int httpPort = 80;

if (!client.connect(host, httpPort)) {

Serial.println("connection failed");

return;

}

//创建URI请求

String url = "/input/";

url += streamId;

url += "?private\_key=";

url += privateKey;

url += "&value=";

url += value;

Serial.print("Requesting URL: ");

Serial.println(url);

//向服务器发送请求

client.print(String("GET ") + url + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n\r\n");

unsigned long timeout = millis();

while (client.available() == 0) {

if (millis() - timeout > 5000) {

Serial.println(">>> Client Timeout !");

client.stop();

return;

}

}

//读取服务器返回的所有行数据并在串口打印

while(client.available()) {

String line = client.readStringUntil('\r');

Serial.print(line);

}

Serial.println();

Serial.println("closing connection");

}

3. MicroPython开发环境实现

代码如下：

import network, urequests, ujson

from utime import sleep\_ms

ssid = ' ' #连接WiFi名称

password = ' ' #连接WiFi密码

url = 'http://example.com' #测试用例

wlan = network.WLAN(network.STA\_IF) #创建站点接口

wlan.active(True) #激活接口

wlan.connect(ssid, password) #连接WiFi

while not wlan.isconnected():

print('.')

sleep\_ms(500)

print('WiFi connected')

response = urequests.get(url)

if response.status\_code == 200: #返回值200为请求成功

for i in response.text.split('\n'): #输出URL的请求响应

print(i)

sleep\_ms(10)

else:

print('Requests failed, status\_code:')

print(response.status\_code)

### 7.2.2 WebSocket程序示例

from SimpleWebSocketServer import SimpleWebSocketServer, WebSocket

class SimpleEcho(WebSocket):

def handleMessage(self):

# echo message back to client

self.sendMessage(self.data)

def handleConnected(self):

print(self.address, 'connected')

def handleClose(self):

print(self.address, 'closed')

print('waiting for the incoming client...')

server = SimpleWebSocketServer('YourIP', 8000, SimpleEcho) #客户端和服务端一致

server.serveforever()

1. 基于ESP IDF的VS Code开发环境实现

CONFIG\_EXAMPLE\_WIFI\_SSID="myssid" //更换WiFi名字

CONFIG\_EXAMPLE\_WIFI\_PASSWORD="mypassword" //更换WiFi密码

程序源码如下：

#include <stdio.h>

#include "esp\_wifi.h"

#include "esp\_system.h"

#include "nvs\_flash.h"

#include "esp\_event.h"

#include "protocol\_examples\_common.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/semphr.h"

#include "freertos/event\_groups.h"

#include "esp\_log.h"

#include "esp\_websocket\_client.h"

#include "esp\_event.h"

#define NO\_DATA\_TIMEOUT\_SEC 10 //10s超时定义

static const char \*TAG = "WEBSOCKET";

static TimerHandle\_t shutdown\_signal\_timer;

static SemaphoreHandle\_t shutdown\_sema;

static void shutdown\_signaler(TimerHandle\_t xTimer) //超时关闭信号定时器

{

ESP\_LOGI(TAG, "No data received for %d seconds, signaling shutdown", NO\_DATA\_TIMEOUT\_SEC);

xSemaphoreGive(shutdown\_sema);

}

#if CONFIG\_WEBSOCKET\_URI\_FROM\_STDIN //URI从标准输入获取

static void get\_string(char \*line, size\_t size)

{

int count = 0;

while (count < size) {

int c = fgetc(stdin);

if (c == '\n') {

line[count] = '\0';

break;

} else if (c > 0 && c < 127) {

line[count] = c;

++count;

}

vTaskDelay(10 / portTICK\_PERIOD\_MS);

}

}

#endif /\* CONFIG\_WEBSOCKET\_URI\_FROM\_STDIN \*/

static void websocket\_event\_handler(void \*handler\_args, esp\_event\_base\_t base, int32\_t event\_id, void \*event\_data) //WebSocket事件句柄

{

esp\_websocket\_event\_data\_t \*data = (esp\_websocket\_event\_data\_t \*)event\_data; //数据获取

switch (event\_id) { //事件ID处理

case WEBSOCKET\_EVENT\_CONNECTED: //连接

ESP\_LOGI(TAG, "WEBSOCKET\_EVENT\_CONNECTED");

break;

case WEBSOCKET\_EVENT\_DISCONNECTED: //断开

ESP\_LOGI(TAG, "WEBSOCKET\_EVENT\_DISCONNECTED");

break;

case WEBSOCKET\_EVENT\_DATA: //接收数据

ESP\_LOGI(TAG, "WEBSOCKET\_EVENT\_DATA");

ESP\_LOGI(TAG, "Received opcode=%d", data->op\_code);

ESP\_LOGW(TAG, "Received=%.\*s", data->data\_len, (char \*)data->data\_ptr);

ESP\_LOGW(TAG, "Total payload length=%d, data\_len=%d, current payload offset=%d\r\n", data->payload\_len, data->data\_len, data->payload\_offset);

xTimerReset(shutdown\_signal\_timer, portMAX\_DELAY);

break;

case WEBSOCKET\_EVENT\_ERROR: //错误

ESP\_LOGI(TAG, "WEBSOCKET\_EVENT\_ERROR");

break;

}

}

static void websocket\_app\_start(void) //启动WebSocket应用

{

esp\_websocket\_client\_config\_t websocket\_cfg = {}; //WebSocket配置变量

shutdown\_signal\_timer = xTimerCreate("Websocket shutdown timer", NO\_DATA\_TIMEOUT\_SEC \* 1000 / portTICK\_PERIOD\_MS, pdFALSE, NULL, shutdown\_signaler); //创建定时器

shutdown\_sema = xSemaphoreCreateBinary(); //创建二值信号量

#if CONFIG\_WEBSOCKET\_URI\_FROM\_STDIN //URI从标准输入获取

char line[128];

ESP\_LOGI(TAG, "Please enter uri of websocket endpoint");

get\_string(line, sizeof(line));

websocket\_cfg.uri = line;

ESP\_LOGI(TAG, "Endpoint uri: %s\n", line);

#else

websocket\_cfg.uri = CONFIG\_WEBSOCKET\_URI; //URI从配置中获取

#endif /\* CONFIG\_WEBSOCKET\_URI\_FROM\_STDIN \*/

ESP\_LOGI(TAG, "Connecting to %s...", websocket\_cfg.uri); //连接并初始化

esp\_websocket\_client\_handle\_t client = esp\_websocket\_client\_init(&websocket\_cfg);

esp\_websocket\_register\_events(client, WEBSOCKET\_EVENT\_ANY, websocket\_event\_handler, (void \*)client); //注册事件

esp\_websocket\_client\_start(client); //开启客户端

xTimerStart(shutdown\_signal\_timer, portMAX\_DELAY); //开启定时器

char data[32];

int i = 0;

while (i < 10) { //小于10次，开始获取数据

if (esp\_websocket\_client\_is\_connected(client)) {

int len = sprintf(data, "hello %04d", i++);

ESP\_LOGI(TAG, "Sending %s", data);

esp\_websocket\_client\_send\_text(client, data, len, portMAX\_DELAY);//发送文本

}

vTaskDelay(1000 / portTICK\_RATE\_MS); //每次延迟1s

}

xSemaphoreTake(shutdown\_sema, portMAX\_DELAY); //FreeRTOS获取信号量

esp\_websocket\_client\_stop(client); //停止客户端

ESP\_LOGI(TAG, "Websocket Stopped");

esp\_websocket\_client\_destroy(client); //释放资源

}

void app\_main(void) //主函数入口

{

ESP\_LOGI(TAG, "[APP] Startup..");

ESP\_LOGI(TAG, "[APP] Free memory: %d bytes", esp\_get\_free\_heap\_size());

ESP\_LOGI(TAG, "[APP] IDF version: %s", esp\_get\_idf\_version());

esp\_log\_level\_set("\*", ESP\_LOG\_INFO); //日志级别设置

esp\_log\_level\_set("WEBSOCKET\_CLIENT", ESP\_LOG\_DEBUG);

esp\_log\_level\_set("TRANS\_TCP", ESP\_LOG\_DEBUG);

ESP\_ERROR\_CHECK(nvs\_flash\_init()); //闪存初始化

ESP\_ERROR\_CHECK(esp\_netif\_init()); //网络接口初始化

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default()); //创建默认循环事件

ESP\_ERROR\_CHECK(example\_connect()); //连接网络

websocket\_app\_start(); //开启WebSocket应用程序

}

2.Arduino开发环境实现

代码如下：

#include <WiFi.h>

#include <WebSocketClient.h>

const char\* ssid = ""; //WiFi名称

const char\* password = ""; //WiFi密码

char path[] = "/"; //路径

char host[] = "echo.websocket.org"; //主机地址

//实例化WebSocketClient

WebSocketClient webSocketClient;

//使用WiFiClient类创建TCP连接

WiFiClient client;

void setup() {

Serial.begin(115200);

delay(10);

//连接WiFi

Serial.println();

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

delay(5000);

//连接WebSocket服务器端

if (client.connect("echo.websocket.org", 80)) {

Serial.println("Connected");

} else {

Serial.println("Connection failed.");

while(1) {

//等待

}

}

//与服务器端握手

webSocketClient.path = path;

webSocketClient.host = host;

if (webSocketClient.handshake(client)) {

Serial.println("Handshake successful");

} else {

Serial.println("Handshake failed.");

while(1) {

//等待

}

}

}

void loop() {

String data;

if (client.connected()) {

//接收数据

webSocketClient.getData(data); //获取数据并串口打印

if (data.length() > 0) {

Serial.print("Received data: ");

Serial.println(data);

}

pinMode(32, INPUT); //第32引脚设置为输入

data = String(analogRead(32)); //读取模拟值

Serial.print("Sent data: ");

Serial.println(data);

//发送数据

webSocketClient.sendData(data);

} else {

Serial.println("Client disconnected.");

}

delay(3000);

}

3. MicroPython开发环境实现

代码如下：

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

import sys

try:

import ubinascii as binascii

except:

import binascii

try:

import uhashlib as hashlib

except:

import hashlib

DEBUG = 0

def server\_handshake(sock):

clr = sock.makefile("rwb", 0)

l = clr.readline()

#sys.stdout.write(repr(l))

webkey = None

while 1:

l = clr.readline()

if not l:

raise OSError("EOF in headers")

if l == b"\r\n":

break

h, v = [x.strip() for x in l.split(b":", 1)]

if DEBUG:

print((h, v))

if h == b'Sec-WebSocket-Key':

webkey = v

if not webkey:

raise OSError("Not a websocket request")

if DEBUG:

print("Sec-WebSocket-Key:", webkey, len(webkey))

respkey = webkey + b"258EAFA5-E914-47DA-95CA-C5AB0DC85B11"

respkey = hashlib.sha1(respkey).digest()

respkey = binascii.b2a\_base64(respkey)[:-1]

resp = b"""\ HTTP/1.1 101 Switching Protocols\r Upgrade: websocket\r Connection: Upgrade\r Sec-WebSocket-Accept: %s\r \r """ % respkey

if DEBUG:

print(resp)

sock.send(resp)

def client\_handshake(sock):

cl = sock.makefile("rwb", 0)

cl.write(b"""\ GET / HTTP/1.1\r Host: echo.websocket.org\r Connection: Upgrade\r Upgrade: websocket\r Sec-WebSocket-Key: foo\r \r """)

l = cl.readline()

while 1:

l = cl.readline()

if l == b"\r\n":

break

执行的主函数文件websocket.py，代码如下：

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

import socket

import websocket\_helper

try:

import network

except:

pass

import sys

import os

try:

import ustruct as struct

except:

import struct

DEBUG = False

class websocket:

def \_\_init\_\_(self, s):

self.s = s

def write(self, data):

l = len(data)

if l < 126:

hdr = struct.pack(">BB", 0x82, l)

else:

hdr = struct.pack(">BBH", 0x82, 126, l)

self.s.send(hdr)

self.s.send(data)

def recvexactly(self, sz):

res = b""

while sz:

data = self.s.recv(sz)

if not data:

break

res += data

sz -= len(data)

return res

def read(self):

while True:

hdr = self.recvexactly(2)

assert len(hdr) == 2

firstbyte, secondbyte = struct.unpack(">BB", hdr)

mskenable = True if secondbyte & 0x80 else False

length = secondbyte & 0x7f

if DEBUG:

print('test length=%d' % length)

print('mskenable=' + str(mskenable))

if length == 126:

hdr = self.recvexactly(2)

assert len(hdr) == 2

(length,) = struct.unpack(">H", hdr)

if length == 127:

hdr = self.recvexactly(8)

assert len(hdr) == 8

(length,) = struct.unpack(">Q", hdr)

if DEBUG:

print('length=%d' % length)

opcode = firstbyte & 0x0f

if opcode == 8:

self.s.close()

return ''

fin = True if firstbyte&0x80 else False

if DEBUG:

print('fin='+str(fin))

print('opcode=%d'%opcode)

if mskenable:

hdr = self.recvexactly(4)

assert len(hdr) == 4

(msk1,msk2,msk3,msk4) = struct.unpack(">BBBB", hdr)

msk = [msk1,msk2,msk3,msk4]

data = []

while length:

skip = self.s.recv(length)

# debugmsg("Skip data: %s" % skip)

length -= len(skip)

data.extend(skip)

newdata = []

for i,item in enumerate(data): #解码数据

j = i % 4

newdata.append(chr(data[i] ^ msk[j]))

res = ''.join(newdata)

return res

print('my server start...')

try:

sta\_if = network.WLAN(network.STA\_IF)

sta\_if.active(True)

sta\_if.connect('Redmi\_77DE','12345678')

while True:

if sta\_if.ifconfig()[0] != '0.0.0.0':

break

print('succ connect wifi ap,get ipaddr:')

print(sta\_if.ifconfig())

except:

pass

sock = socket.socket()

sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

sock.bind(('0.0.0.0', 8000))

sock.listen(5)

print('websocket listen at 8000...')

while True:

conn, address = sock.accept() # 接收到socket

print('client connect...:')

print(address)

websocket\_helper.server\_handshake(conn)

ws = websocket(conn)

print('websocket connect succ')

while True:

text = ws.read()

if text =='':

break

print(text)

### 7.3.2 MQTT程序示例

本示例将开发框架的例子进行修改，官方的源码路径为：esp-idf\examples\protocols\ mqtt\tcp。打开VS Code，在sdkconfig文件中更新WiFi的用户名和密码并保存，找到如下语句：

CONFIG\_BROKER\_URL="mqtt://test.mosquitto.org"

CONFIG\_EXAMPLE\_WIFI\_SSID="myssid" //更换WiFi名字

CONFIG\_EXAMPLE\_WIFI\_PASSWORD="mypassword" //更换WiFi密码

代码如下：

#include <stdio.h>

#include <stdint.h>

#include <stddef.h>

#include <string.h>

#include "esp\_wifi.h"

#include "esp\_system.h"

#include "nvs\_flash.h"

#include "esp\_event.h"

#include "esp\_netif.h"

#include "protocol\_examples\_common.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/semphr.h"

#include "freertos/queue.h"

#include "lwip/sockets.h"

#include "lwip/dns.h"

#include "lwip/netdb.h"

#include "esp\_log.h"

#include "mqtt\_client.h"

//#define CONFIG\_BROKER\_URL\_FROM\_STDIN 1 如果定义，运行时输入链接地址

static const char \*TAG = "MQTT\_EXAMPLE";

static esp\_err\_t mqtt\_event\_handler\_cb(esp\_mqtt\_event\_handle\_t event) //事件句柄回调程序

{

esp\_mqtt\_client\_handle\_t client = event->client;

int msg\_id;

// your\_context\_t \*context = event->context;

switch (event->event\_id) { //根据事件ID，执行相关分支

case MQTT\_EVENT\_CONNECTED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_CONNECTED");

msg\_id = esp\_mqtt\_client\_publish(client, "/topic/qos1", "data\_3", 0, 1, 0);

ESP\_LOGI(TAG, "sent publish successful, msg\_id=%d", msg\_id);

msg\_id = esp\_mqtt\_client\_subscribe(client, "/topic/qos0", 0);

ESP\_LOGI(TAG, "sent subscribe successful, msg\_id=%d", msg\_id);

msg\_id = esp\_mqtt\_client\_subscribe(client, "/topic/qos1", 1);

ESP\_LOGI(TAG, "sent subscribe successful, msg\_id=%d", msg\_id);

msg\_id = esp\_mqtt\_client\_unsubscribe(client, "/topic/qos1");

ESP\_LOGI(TAG, "sent unsubscribe successful, msg\_id=%d", msg\_id);

break;

case MQTT\_EVENT\_DISCONNECTED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_DISCONNECTED");

break;

case MQTT\_EVENT\_SUBSCRIBED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_SUBSCRIBED, msg\_id=%d", event->msg\_id);

msg\_id = esp\_mqtt\_client\_publish(client, "/topic/qos0", "data", 0, 0, 0);

ESP\_LOGI(TAG, "sent publish successful, msg\_id=%d", msg\_id);

break;

case MQTT\_EVENT\_UNSUBSCRIBED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_UNSUBSCRIBED, msg\_id=%d", event->msg\_id);

break;

case MQTT\_EVENT\_PUBLISHED:

ESP\_LOGI(TAG, "MQTT\_EVENT\_PUBLISHED, msg\_id=%d", event->msg\_id);

break;

case MQTT\_EVENT\_DATA:

ESP\_LOGI(TAG, "MQTT\_EVENT\_DATA");

printf("TOPIC=%.\*s\r\n", event->topic\_len, event->topic);

printf("DATA=%.\*s\r\n", event->data\_len, event->data);

break;

case MQTT\_EVENT\_ERROR:

ESP\_LOGI(TAG, "MQTT\_EVENT\_ERROR");

break;

default:

ESP\_LOGI(TAG, "Other event id:%d", event->event\_id);

break;

}

return ESP\_OK;

}

static void mqtt\_event\_handler(void \*handler\_args, esp\_event\_base\_t base, int32\_t event\_id, void \*event\_data) { //事件句柄处理程序

ESP\_LOGD(TAG, "Event dispatched from event loop base=%s, event\_id=%d", base, event\_id);

mqtt\_event\_handler\_cb(event\_data);

}

static void mqtt\_app\_start(void) //应用开启

{

esp\_mqtt\_client\_config\_t mqtt\_cfg = {

.uri = CONFIG\_BROKER\_URL, //使用sdkconfig设置的链接地址

};

#if CONFIG\_BROKER\_URL\_FROM\_STDIN //如果定义了键盘输入

char line[128];

if (strcmp(mqtt\_cfg.uri, "FROM\_STDIN") == 0) {

int count = 0;

printf("Please enter url of mqtt broker\n");

while (count < 128) {

int c = fgetc(stdin);

if (c == '\n') {

line[count] = '\0';

break;

} else if (c > 0 && c < 127) {

line[count] = c;

++count;

}

vTaskDelay(10 / portTICK\_PERIOD\_MS);

}

mqtt\_cfg.uri = line;

printf("Broker url: %s\n", line);

} else {

ESP\_LOGE(TAG, "Configuration mismatch: wrong broker url");

abort();

}

#endif /\* CONFIG\_BROKER\_URL\_FROM\_STDIN \*/

esp\_mqtt\_client\_handle\_t client = esp\_mqtt\_client\_init(&mqtt\_cfg); //定义客户端并注册

esp\_mqtt\_client\_register\_event(client, ESP\_EVENT\_ANY\_ID, mqtt\_event\_handler, client);

esp\_mqtt\_client\_start(client); //开启客户端

}

void app\_main(void) //主程序入口

{

ESP\_LOGI(TAG, "[APP] Startup.."); //信息日志输出设置

ESP\_LOGI(TAG, "[APP] Free memory: %d bytes", esp\_get\_free\_heap\_size());

ESP\_LOGI(TAG, "[APP] IDF version: %s", esp\_get\_idf\_version());

esp\_log\_level\_set("\*", ESP\_LOG\_INFO);

esp\_log\_level\_set("MQTT\_CLIENT", ESP\_LOG\_VERBOSE);

esp\_log\_level\_set("MQTT\_EXAMPLE", ESP\_LOG\_VERBOSE);

esp\_log\_level\_set("TRANSPORT\_TCP", ESP\_LOG\_VERBOSE);

esp\_log\_level\_set("TRANSPORT\_SSL", ESP\_LOG\_VERBOSE);

esp\_log\_level\_set("TRANSPORT", ESP\_LOG\_VERBOSE);

esp\_log\_level\_set("OUTBOX", ESP\_LOG\_VERBOSE);

ESP\_ERROR\_CHECK(nvs\_flash\_init()); //初始化闪存

ESP\_ERROR\_CHECK(esp\_netif\_init()); //初始化网络接口

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default()); //创建默认循环任务

ESP\_ERROR\_CHECK(example\_connect()); //连接网络

mqtt\_app\_start(); //开启应用

}

2. Arduino开发环境实现

#include<WiFi.h>

#include<PubSubClient.h> //在管理库中下载

const char\*ssid =""; //ESP32连接的Wi-Fi账号

const char\*password = ""; //Wi-Fi密码

const char\*mqttServer = ""; //要连接到的服务器IP

const int mqttPort =1883; //要连接到的服务器端口号

const char\*mqttUser = "admin"; //MQTT服务器账号

const char\*mqttPassword = "public"; //MQTT服务器密码

WiFiClient espClient; //定义wifiClient实例

PubSubClient client(espClient); //定义PubSubClient的实例

void callback(char\*topic, byte\* payload, unsigned int length)

{

Serial.print("来自订阅的主题:"); //串口打印：来自订阅的主题:

Serial.println(topic); //串口打印订阅的主题

Serial.print("信息："); //串口打印：信息：

for (int i = 0; i< length; i++) //使用循环打印接收到的信息

{

Serial.print((char)payload[i]);

}

Serial.println();

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

}

void setup()

{

Serial.begin(115200); //串口函数，波特率设置

while (WiFi.status() != WL\_CONNECTED) //若Wi-Fi接入成功

{

Serial.println("连接wifi中"); //串口输出：连接Wi-Fi中

WiFi.begin(ssid,password); //接入WiFi函数（Wi-Fi名称，密码）

delay(2000); //若尚未连接Wi-Fi，则进行重连Wi-Fi的循环

}

Serial.println("wifi连接成功"); //连接Wi-Fi成功之后会跳出循环，串口并输出连接成功

client.setServer(mqttServer,mqttPort); //MQTT服务器连接函数（服务器IP，端口号）

client.setCallback(callback); //设定回调方式，当ESP32收到订阅消息时会调用此方法

while (!client.connected()) //是否连接上MQTT服务器

{

Serial.println("连接服务器中"); //串口打印，连接服务器中

if (client.connect("ESP32Client",mqttUser, mqttPassword )) //如果服务器连接成功

{

Serial.println("服务器连接成功"); //串口打印，服务器连接成功

}

else

{

Serial.print("连接服务器失败"); //串口打印，连接服务器失败

Serial.print(client.state()); //重新连接函数

delay(2000);

}

}

client.subscribe("ESP32"); //连接MQTT服务器后订阅主题

Serial.print("已订阅主题，等待主题消息...."); //串口打印，已订阅主题，等待主题消息

client.publish("/World","Hello from ESP32"); //向服务器端发送的信息(主题，内容)

}

void loop()

{

client.loop(); //回旋接收函数 等待服务器端返回的数据

}

3. MicroPython开发环境实现

代码如下：

from umqtt.simple import MQTTClient

from machine import Pin

from utime import sleep\_ms

import network

SSID = ''

PASSWORD = ''

led = Pin(2, Pin.OUT, value = 0)

SERVER = 'yourMQTTSever' #MQTT服务器端地址

PORT = 'yourMQTTSeverPort' #MQTT服务器端口

CLIENT\_ID = 'yourClientID' #MQTT客户端ID

TOPIC = b"yourTOPIC" #订阅信息主题

username='yourIotUserName' #可选，账户名

password='yourIotPassword' #可选，账户密码

state = 0

c = None

def sub\_cb(topic, msg):

global state

print((topic, msg))

if msg == b"on": #点亮

state = 1

elif msg == b"off": #关闭

state = 0

elif msg == b"toggle": #翻转

state = 1 - state

else:

return

led.value(state)

print(state)

def connectWifi(ssid,passwd):

global wlan

wlan=network.WLAN(network.STA\_IF) #实例化WLAN

wlan.active(True) #激活网络接口

wlan.disconnect() #断开现有连接

wlan.connect(ssid,passwd) #连接WiFi

while(wlan.ifconfig()[0] == '0.0.0.0'):

print('.')

sleep\_ms(500)

print('WiFi connected')

connectWifi(SSID, PASSWORD) #连接WiFi

#捕获异常，如果在“try”中意外中断，则停止程序

try:

c = MQTTClient(CLIENT\_ID, SERVER, PORT) #实例化MQTT客户端

#c = MQTTClient(CLIENT\_ID, SERVER, PORT, username, password)

c.set\_callback(sub\_cb) #设置回调函数

c.connect() #连接MQTT服务器

c.subscribe(TOPIC) #客户端订阅一个主题

print("Connected to %s, subscribed to %s topic" % (SERVER, TOPIC))

while True:

c.wait\_msg() #等待消息

finally:

if(c is not None):

c.disconnect()

wlan.disconnect()

wlan.active(False)

### 8.4.1 基于ESP IDF开发实现

修改sdkconfig文件中的CONFIG\_BLINK\_GPIO=2，通过VS Code编辑，代码如下：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/event\_groups.h"

#include "esp\_system.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "esp\_bt.h"

#include "esp\_gap\_ble\_api.h"

#include "esp\_gatts\_api.h"

#include "esp\_bt\_defs.h"

#include "esp\_bt\_main.h"

#include "esp\_gatt\_common\_api.h"

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "sdkconfig.h"

#define BLINK\_GPIO CONFIG\_BLINK\_GPIO

void turn\_on\_led(){ /\*打开LED \*/

printf("Turning on the LED\n");

gpio\_set\_level(BLINK\_GPIO, 1);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

void turn\_off\_led(){ /\*关闭LED\*/

printf("Turning off the LED\n");

gpio\_set\_level(BLINK\_GPIO, 0);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

#define GATTS\_TAG "GATTS\_DEMO"

static void gatts\_profile\_a\_event\_handler(esp\_gatts\_cb\_event\_t event, esp\_gatt\_if\_t gatts\_if, esp\_ble\_gatts\_cb\_param\_t \*param); //声明静态函数

#define GATTS\_SERVICE\_UUID\_TEST\_A 0x00FF //测试服务UUID

#define GATTS\_CHAR\_UUID\_TEST\_A 0xFF01

#define GATTS\_DESCR\_UUID\_TEST\_A 0x3333

#define GATTS\_NUM\_HANDLE\_TEST\_A 4

#define GATTS\_SERVICE\_UUID\_TEST\_B 0x00EE

#define GATTS\_CHAR\_UUID\_TEST\_B 0xEE01

#define GATTS\_DESCR\_UUID\_TEST\_B 0x2222

#define GATTS\_NUM\_HANDLE\_TEST\_B 4

#define TEST\_DEVICE\_NAME "ESP\_GATTS\_DEMO" //广播设备名称

#define TEST\_MANUFACTURER\_DATA\_LEN 17

#define GATTS\_DEMO\_CHAR\_VAL\_LEN\_MAX 0x40

#define PREPARE\_BUF\_MAX\_SIZE 1024

static uint8\_t char1\_str[] = {0x11,0x22,0x33};

static esp\_gatt\_char\_prop\_t a\_property = 0;

static esp\_gatt\_char\_prop\_t b\_property = 0;

static esp\_attr\_value\_t gatts\_demo\_char1\_val = //属性值

{

.attr\_max\_len = GATTS\_DEMO\_CHAR\_VAL\_LEN\_MAX,

.attr\_len = sizeof(char1\_str),

.attr\_value = char1\_str,

};

static uint8\_t adv\_config\_done = 0;

#define adv\_config\_flag (1 << 0)

#define scan\_rsp\_config\_flag (1 << 1)

#ifdef CONFIG\_SET\_RAW\_ADV\_DATA //定义广播数据

static uint8\_t raw\_adv\_data[] = { //广播数据

0x02, 0x01, 0x06,

0x02, 0x0a, 0xeb, 0x03, 0x03, 0xab, 0xcd

};

static uint8\_t raw\_scan\_rsp\_data[] = { //扫描响应数据

0x0f, 0x09, 0x45, 0x53, 0x50, 0x5f, 0x47, 0x41, 0x54, 0x54, 0x53, 0x5f, 0x44,

0x45, 0x4d, 0x4f

};

#else

static uint8\_t adv\_service\_uuid128[32] = { 广播服务UUID

/\*LSB <-> MSB \*/

//第一个UUID, 16比特, [12],[13]位是值

0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0xEE, 0x00, 0x00, 0x00,

//第二个UUID, 32比特, [12], [13], [14], [15]位是值

0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00,

};

//广播数据小于31字节

//static uint8\_t test\_manufacturer[TEST\_MANUFACTURER\_DATA\_LEN] = {0x12, 0x23, 0x45, 0x56}; 测试用

static esp\_ble\_adv\_data\_t adv\_data = {//广播数据

.set\_scan\_rsp = false,

.include\_name = true,

.include\_txpower = false,

.min\_interval = 0x0006, //slave connection min interval, Time = min\_interval \* 1.25 msec

.max\_interval = 0x0010, //slave connection max interval, Time = max\_interval \* 1.25 msec

.appearance = 0x00,

.manufacturer\_len = 0, //TEST\_MANUFACTURER\_DATA\_LEN,

.p\_manufacturer\_data = NULL, //&test\_manufacturer[0],

.service\_data\_len = 0,

.p\_service\_data = NULL,

.service\_uuid\_len = sizeof(adv\_service\_uuid128),

.p\_service\_uuid = adv\_service\_uuid128,

.flag = (ESP\_BLE\_ADV\_FLAG\_GEN\_DISC | ESP\_BLE\_ADV\_FLAG\_BREDR\_NOT\_SPT),

};

static esp\_ble\_adv\_data\_t scan\_rsp\_data = {//扫描响应数据

.set\_scan\_rsp = true,

.include\_name = true,

.include\_txpower = true,

//.min\_interval = 0x0006,

//.max\_interval = 0x0010,

.appearance = 0x00,

.manufacturer\_len = 0, //TEST\_MANUFACTURER\_DATA\_LEN,

.p\_manufacturer\_data = NULL, //&test\_manufacturer[0],

.service\_data\_len = 0,

.p\_service\_data = NULL,

.service\_uuid\_len = sizeof(adv\_service\_uuid128),

.p\_service\_uuid = adv\_service\_uuid128,

.flag = (ESP\_BLE\_ADV\_FLAG\_GEN\_DISC | ESP\_BLE\_ADV\_FLAG\_BREDR\_NOT\_SPT),

};

#endif /\* CONFIG\_SET\_RAW\_ADV\_DATA \*/

static esp\_ble\_adv\_params\_t adv\_params = { //广播参数

.adv\_int\_min = 0x20,

.adv\_int\_max = 0x40,

.adv\_type = ADV\_TYPE\_IND,

.own\_addr\_type = BLE\_ADDR\_TYPE\_PUBLIC,

//.peer\_addr =

//.peer\_addr\_type =

.channel\_map = ADV\_CHNL\_ALL,

.adv\_filter\_policy = ADV\_FILTER\_ALLOW\_SCAN\_ANY\_CON\_ANY,

};

#define PROFILE\_NUM 2

#define PROFILE\_A\_APP\_ID 0

#define PROFILE\_B\_APP\_ID 1

struct gatts\_profile\_inst { //GATT配置参数

esp\_gatts\_cb\_t gatts\_cb;

uint16\_t gatts\_if;

uint16\_t app\_id;

uint16\_t conn\_id;

uint16\_t service\_handle;

esp\_gatt\_srvc\_id\_t service\_id;

uint16\_t char\_handle;

esp\_bt\_uuid\_t char\_uuid;

esp\_gatt\_perm\_t perm;

esp\_gatt\_char\_prop\_t property;

uint16\_t descr\_handle;

esp\_bt\_uuid\_t descr\_uuid;

};

/\*配置文件，app\_id 和gatts\_if，这个数组将存储 ESP\_GATTS\_REG\_EVT返回的gatts\_if \*/

static struct gatts\_profile\_inst gl\_profile\_tab[PROFILE\_NUM] = {

[PROFILE\_A\_APP\_ID] = {

.gatts\_cb = gatts\_profile\_a\_event\_handler, //回调句柄

.gatts\_if = ESP\_GATT\_IF\_NONE, //没有获得gatt\_if

},

};

typedef struct {

uint8\_t \*prepare\_buf;

int prepare\_len;

} prepare\_type\_env\_t; //定义准备结构体

static prepare\_type\_env\_t a\_prepare\_write\_env; //准备写入

void example\_write\_event\_env(esp\_gatt\_if\_t gatts\_if, prepare\_type\_env\_t \*prepare\_write\_env, esp\_ble\_gatts\_cb\_param\_t \*param);

void example\_exec\_write\_event\_env(prepare\_type\_env\_t \*prepare\_write\_env, esp\_ble\_gatts\_cb\_param\_t \*param);

static void gap\_event\_handler(esp\_gap\_ble\_cb\_event\_t event, esp\_ble\_gap\_cb\_param\_t \*param)

{ //GAP事件句柄

switch (event) {

#ifdef CONFIG\_SET\_RAW\_ADV\_DATA

case ESP\_GAP\_BLE\_ADV\_DATA\_RAW\_SET\_COMPLETE\_EVT: //原始数据设置完成

adv\_config\_done &= (~adv\_config\_flag);

if (adv\_config\_done==0){

esp\_ble\_gap\_start\_advertising(&adv\_params);

}

break;

case ESP\_GAP\_BLE\_SCAN\_RSP\_DATA\_RAW\_SET\_COMPLETE\_EVT: //响应设置完成

adv\_config\_done &= (~scan\_rsp\_config\_flag);

if (adv\_config\_done==0){

esp\_ble\_gap\_start\_advertising(&adv\_params);

}

break;

#else

case ESP\_GAP\_BLE\_ADV\_DATA\_SET\_COMPLETE\_EVT: //广播数据设置完成

adv\_config\_done &= (~adv\_config\_flag);

if (adv\_config\_done == 0){

esp\_ble\_gap\_start\_advertising(&adv\_params);

}

break;

case ESP\_GAP\_BLE\_SCAN\_RSP\_DATA\_SET\_COMPLETE\_EVT: //扫描响应设置完成

adv\_config\_done &= (~scan\_rsp\_config\_flag);

if (adv\_config\_done == 0){

esp\_ble\_gap\_start\_advertising(&adv\_params);

}

break;

#endif

case ESP\_GAP\_BLE\_ADV\_START\_COMPLETE\_EVT: //开启广播完成

//advertising start complete event to indicate advertising start successfully or failed

if (param->adv\_start\_cmpl.status != ESP\_BT\_STATUS\_SUCCESS) {

ESP\_LOGE(GATTS\_TAG, "Advertising start failed\n");

}

break;

case ESP\_GAP\_BLE\_ADV\_STOP\_COMPLETE\_EVT: //停止广播完成

if (param->adv\_stop\_cmpl.status != ESP\_BT\_STATUS\_SUCCESS) {

ESP\_LOGE(GATTS\_TAG, "Advertising stop failed\n");

} else {

ESP\_LOGI(GATTS\_TAG, "Stop adv successfully\n");

}

break;

case ESP\_GAP\_BLE\_UPDATE\_CONN\_PARAMS\_EVT: //更新参数

ESP\_LOGI(GATTS\_TAG, "update connection params status = %d, min\_int = %d, max\_int = %d,conn\_int = %d,latency = %d, timeout = %d",

param->update\_conn\_params.status,

param->update\_conn\_params.min\_int,

param->update\_conn\_params.max\_int,

param->update\_conn\_params.conn\_int,

param->update\_conn\_params.latency,

param->update\_conn\_params.timeout);

break;

default:

break;

}

}

void example\_write\_event\_env(esp\_gatt\_if\_t gatts\_if, prepare\_type\_env\_t \*prepare\_write\_env, esp\_ble\_gatts\_cb\_param\_t \*param){ //写入事件

esp\_gatt\_status\_t status = ESP\_GATT\_OK;

if (param->write.need\_rsp){

if (param->write.is\_prep){

if (prepare\_write\_env->prepare\_buf == NULL) {

prepare\_write\_env->prepare\_buf = (uint8\_t \*)malloc(PREPARE\_BUF\_MAX\_SIZE\*sizeof(uint8\_t)); //分配空间

prepare\_write\_env->prepare\_len = 0;

if (prepare\_write\_env->prepare\_buf == NULL) {

ESP\_LOGE(GATTS\_TAG, "Gatt\_server prep no mem\n");

status = ESP\_GATT\_NO\_RESOURCES;

}

} else {

if(param->write.offset > PREPARE\_BUF\_MAX\_SIZE) { //准备缓存空间

status = ESP\_GATT\_INVALID\_OFFSET;

} else if ((param->write.offset + param->write.len) > PREPARE\_BUF\_MAX\_SIZE) {

//异常处理

status = ESP\_GATT\_INVALID\_ATTR\_LEN;

}

}

esp\_gatt\_rsp\_t \*gatt\_rsp = (esp\_gatt\_rsp\_t \*)malloc(sizeof(esp\_gatt\_rsp\_t));

gatt\_rsp->attr\_value.len = param->write.len;

gatt\_rsp->attr\_value.handle = param->write.handle;

gatt\_rsp->attr\_value.offset = param->write.offset;

gatt\_rsp->attr\_value.auth\_req = ESP\_GATT\_AUTH\_REQ\_NONE;

memcpy(gatt\_rsp->attr\_value.value, param->write.value, param->write.len);

esp\_err\_t response\_err = esp\_ble\_gatts\_send\_response(gatts\_if, param->write.conn\_id, param->write.trans\_id, status, gatt\_rsp); //响应处理

if (response\_err != ESP\_OK){ //故障处理

ESP\_LOGE(GATTS\_TAG, "Send response error\n");

}

free(gatt\_rsp);

if (status != ESP\_GATT\_OK){

return;

}

memcpy(prepare\_write\_env->prepare\_buf + param->write.offset,

param->write.value,

param->write.len);

prepare\_write\_env->prepare\_len += param->write.len;

}else{

esp\_ble\_gatts\_send\_response(gatts\_if, param->write.conn\_id, param->write.trans\_id, status, NULL);

}

}

}

void example\_exec\_write\_event\_env(prepare\_type\_env\_t \*prepare\_write\_env, esp\_ble\_gatts\_cb\_param\_t \*param){ //执行写入事件

if (param->exec\_write.exec\_write\_flag == ESP\_GATT\_PREP\_WRITE\_EXEC){

esp\_log\_buffer\_hex(GATTS\_TAG, prepare\_write\_env->prepare\_buf, prepare\_write\_env->prepare\_len); //写入

}else{

ESP\_LOGI(GATTS\_TAG,"ESP\_GATT\_PREP\_WRITE\_CANCEL"); //写入取消

}

if (prepare\_write\_env->prepare\_buf) { //写入后续处理

free(prepare\_write\_env->prepare\_buf);

prepare\_write\_env->prepare\_buf = NULL;

}

prepare\_write\_env->prepare\_len = 0;

}

static void gatts\_profile\_a\_event\_handler(esp\_gatts\_cb\_event\_t event, esp\_gatt\_if\_t gatts\_if, esp\_ble\_gatts\_cb\_param\_t \*param) { //配置事件句柄

switch (event) {

case ESP\_GATTS\_REG\_EVT: //注册事件

ESP\_LOGI(GATTS\_TAG, "REGISTER\_APP\_EVT, status %d, app\_id %d\n", param->reg.status, param->reg.app\_id);

gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_id.is\_primary = true;

gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_id.id.inst\_id = 0x00;

gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_id.id.uuid.len = ESP\_UUID\_LEN\_16;

gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_id.id.uuid.uuid.uuid16 = GATTS\_SERVICE\_UUID\_TEST\_A;

esp\_err\_t set\_dev\_name\_ret = esp\_ble\_gap\_set\_device\_name(TEST\_DEVICE\_NAME);

if (set\_dev\_name\_ret){

ESP\_LOGE(GATTS\_TAG, "set device name failed, error code = %x", set\_dev\_name\_ret);

}

#ifdef CONFIG\_SET\_RAW\_ADV\_DATA

esp\_err\_t raw\_adv\_ret = esp\_ble\_gap\_config\_adv\_data\_raw(raw\_adv\_data, sizeof(raw\_adv\_data));

if (raw\_adv\_ret){

ESP\_LOGE(GATTS\_TAG, "config raw adv data failed, error code = %x ", raw\_adv\_ret);

}

adv\_config\_done |= adv\_config\_flag;

esp\_err\_t raw\_scan\_ret = esp\_ble\_gap\_config\_scan\_rsp\_data\_raw(raw\_scan\_rsp\_data, sizeof(raw\_scan\_rsp\_data));

if (raw\_scan\_ret){

ESP\_LOGE(GATTS\_TAG, "config raw scan rsp data failed, error code = %x", raw\_scan\_ret);

}

adv\_config\_done |= scan\_rsp\_config\_flag;

#else

//config adv data

esp\_err\_t ret = esp\_ble\_gap\_config\_adv\_data(&adv\_data);

if (ret){

ESP\_LOGE(GATTS\_TAG, "config adv data failed, error code = %x", ret);

}

adv\_config\_done |= adv\_config\_flag;

//config scan response data

ret = esp\_ble\_gap\_config\_adv\_data(&scan\_rsp\_data);

if (ret){

ESP\_LOGE(GATTS\_TAG, "config scan response data failed, error code = %x", ret);

}

adv\_config\_done |= scan\_rsp\_config\_flag;

#endif

esp\_ble\_gatts\_create\_service(gatts\_if, &gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_id, GATTS\_NUM\_HANDLE\_TEST\_A);

break;

case ESP\_GATTS\_READ\_EVT: { //读取事件

ESP\_LOGI(GATTS\_TAG, "GATT\_READ\_EVT, conn\_id %d, trans\_id %d, handle %d\n", param->read.conn\_id, param->read.trans\_id, param->read.handle);

esp\_gatt\_rsp\_t rsp;

memset(&rsp, 0, sizeof(esp\_gatt\_rsp\_t));

rsp.attr\_value.handle = param->read.handle;

rsp.attr\_value.len = 4;

rsp.attr\_value.value[0] = 0xde;

rsp.attr\_value.value[1] = 0xed;

rsp.attr\_value.value[2] = 0xbe;

rsp.attr\_value.value[3] = 0xef;

esp\_ble\_gatts\_send\_response(gatts\_if, param->read.conn\_id, param->read.trans\_id, ESP\_GATT\_OK, &rsp);

break;

}

case ESP\_GATTS\_WRITE\_EVT: { //写入事件

ESP\_LOGI(GATTS\_TAG, "GATT\_WRITE\_EVT, conn\_id %d, trans\_id %d, handle %d", param->write.conn\_id, param->write.trans\_id, param->write.handle);

if (!param->write.is\_prep){

ESP\_LOGI(GATTS\_TAG, "GATT\_WRITE\_EVT, value len %d, value :", param->write.len);

esp\_log\_buffer\_hex(GATTS\_TAG, param->write.value, param->write.len);

ESP\_LOGI(GATTS\_TAG, "[0]2 %d", param->write.value[0]);

ESP\_LOGI(GATTS\_TAG, "[1]2 %d", param->write.value[1]);

if(param->write.len == 1){ //写入长度为1

if(param->write.value[0]=='0' ){ //如果手机端写入的值为0，打开LED

ESP\_LOGI(GATTS\_TAG, "LED ON");

turn\_on\_led();

}

if(param->write.value[0]=='1' ){ //如果手机端写入的值为1，关闭LED

ESP\_LOGI(GATTS\_TAG, "LED OFF");

turn\_off\_led();

}

}

if (gl\_profile\_tab[PROFILE\_A\_APP\_ID].descr\_handle == param->write.handle && param->write.len == 2){

uint16\_t descr\_value = param->write.value[1]<<8 | param->write.value[0];

if (descr\_value == 0x0001){ //通知处理

if (a\_property & ESP\_GATT\_CHAR\_PROP\_BIT\_NOTIFY){

ESP\_LOGI(GATTS\_TAG, "notify enable");

uint8\_t notify\_data[15];

for (int i = 0; i < sizeof(notify\_data); ++i)

{

notify\_data[i] = i%0xff;

} //通知数据小于最大传输单元

esp\_ble\_gatts\_send\_indicate(gatts\_if, param->write.conn\_id, gl\_profile\_tab[PROFILE\_A\_APP\_ID].char\_handle, sizeof(notify\_data), notify\_data, false);

}

}else if (descr\_value == 0x0002){

if (a\_property & ESP\_GATT\_CHAR\_PROP\_BIT\_INDICATE){

ESP\_LOGI(GATTS\_TAG, "indicate enable");

uint8\_t indicate\_data[15];

for (int i = 0; i < sizeof(indicate\_data); ++i)

{

indicate\_data[i] = i%0xff;

}

//指示数据小于MTU

esp\_ble\_gatts\_send\_indicate(gatts\_if, param->write.conn\_id, gl\_profile\_tab[PROFILE\_A\_APP\_ID].char\_handle, sizeof(indicate\_data), indicate\_data, true);

}

}

else if (descr\_value == 0x0000){

ESP\_LOGI(GATTS\_TAG, "notify/indicate disable ");

}else{

ESP\_LOGE(GATTS\_TAG, "unknown descr value");

esp\_log\_buffer\_hex(GATTS\_TAG, param->write.value, param->write.len);

}

}

}

example\_write\_event\_env(gatts\_if, &a\_prepare\_write\_env, param);

break;

}

case ESP\_GATTS\_EXEC\_WRITE\_EVT: //执行写入事件

ESP\_LOGI(GATTS\_TAG,"ESP\_GATTS\_EXEC\_WRITE\_EVT");

esp\_ble\_gatts\_send\_response(gatts\_if, param->write.conn\_id, param->write.trans\_id, ESP\_GATT\_OK, NULL);

example\_exec\_write\_event\_env(&a\_prepare\_write\_env, param);

break;

case ESP\_GATTS\_MTU\_EVT: //最大传输单元事件

ESP\_LOGI(GATTS\_TAG, "ESP\_GATTS\_MTU\_EVT, MTU %d", param->mtu.mtu);

break;

case ESP\_GATTS\_UNREG\_EVT: //解除注册事件

break;

case ESP\_GATTS\_CREATE\_EVT: //创建事件

ESP\_LOGI(GATTS\_TAG, "CREATE\_SERVICE\_EVT, status %d, service\_handle %d\n", param->create.status, param->create.service\_handle);

gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_handle = param->create.service\_handle;

gl\_profile\_tab[PROFILE\_A\_APP\_ID].char\_uuid.len = ESP\_UUID\_LEN\_16;

gl\_profile\_tab[PROFILE\_A\_APP\_ID].char\_uuid.uuid.uuid16 = GATTS\_CHAR\_UUID\_TEST\_A;

esp\_ble\_gatts\_start\_service(gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_handle);

a\_property = ESP\_GATT\_CHAR\_PROP\_BIT\_READ | ESP\_GATT\_CHAR\_PROP\_BIT\_WRITE | ESP\_GATT\_CHAR\_PROP\_BIT\_NOTIFY;

esp\_err\_t add\_char\_ret = esp\_ble\_gatts\_add\_char(gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_handle, &gl\_profile\_tab[PROFILE\_A\_APP\_ID].char\_uuid, ESP\_GATT\_PERM\_READ | ESP\_GATT\_PERM\_WRITE, a\_property, &gatts\_demo\_char1\_val, NULL);

if (add\_char\_ret){

ESP\_LOGE(GATTS\_TAG, "add char failed, error code =%x",add\_char\_ret);

}

break;

case ESP\_GATTS\_ADD\_INCL\_SRVC\_EVT: //增加服务事件

break;

case ESP\_GATTS\_ADD\_CHAR\_EVT: { //增加特性事件

uint16\_t length = 0;

const uint8\_t \*prf\_char;

ESP\_LOGI(GATTS\_TAG, "ADD\_CHAR\_EVT, status %d, attr\_handle %d, service\_handle %d\n",

param->add\_char.status, param->add\_char.attr\_handle, param->add\_char.service\_handle);

gl\_profile\_tab[PROFILE\_A\_APP\_ID].char\_handle = param->add\_char.attr\_handle;

gl\_profile\_tab[PROFILE\_A\_APP\_ID].descr\_uuid.len = ESP\_UUID\_LEN\_16;

gl\_profile\_tab[PROFILE\_A\_APP\_ID].descr\_uuid.uuid.uuid16 = ESP\_GATT\_UUID\_CHAR\_CLIENT\_CONFIG;

esp\_err\_t get\_attr\_ret = esp\_ble\_gatts\_get\_attr\_value(param->add\_char.attr\_handle, &length, &prf\_char);

if (get\_attr\_ret == ESP\_FAIL){

ESP\_LOGE(GATTS\_TAG, "ILLEGAL HANDLE");

}

ESP\_LOGI(GATTS\_TAG, "the gatts demo char length = %x\n", length);

for(int i = 0; i < length; i++){

ESP\_LOGI(GATTS\_TAG, "prf\_char[%x] =%x\n",i,prf\_char[i]);

}

esp\_err\_t add\_descr\_ret = esp\_ble\_gatts\_add\_char\_descr(gl\_profile\_tab[PROFILE\_A\_APP\_ID].service\_handle, &gl\_profile\_tab[PROFILE\_A\_APP\_ID].descr\_uuid, ESP\_GATT\_PERM\_READ | ESP\_GATT\_PERM\_WRITE, NULL, NULL);

if (add\_descr\_ret){

ESP\_LOGE(GATTS\_TAG, "add char descr failed, error code =%x", add\_descr\_ret);

}

break;

}

case ESP\_GATTS\_ADD\_CHAR\_DESCR\_EVT: //增加特征描述事件

gl\_profile\_tab[PROFILE\_A\_APP\_ID].descr\_handle = param->add\_char\_descr.attr\_handle;

ESP\_LOGI(GATTS\_TAG, "ADD\_DESCR\_EVT, status %d, attr\_handle %d, service\_handle %d\n", param->add\_char\_descr.status, param->add\_char\_descr.attr\_handle, param->add\_char\_descr.service\_handle);

break;

case ESP\_GATTS\_DELETE\_EVT: //删除事件

break;

case ESP\_GATTS\_START\_EVT: //开启事件

ESP\_LOGI(GATTS\_TAG, "SERVICE\_START\_EVT, status %d, service\_handle %d\n",

param->start.status, param->start.service\_handle);

break;

case ESP\_GATTS\_STOP\_EVT: //停止事件

break;

case ESP\_GATTS\_CONNECT\_EVT: { //连接事件

esp\_ble\_conn\_update\_params\_t conn\_params = {0};

memcpy(conn\_params.bda, param->connect.remote\_bda, sizeof(esp\_bd\_addr\_t));

/\*对于iOS系统，BLE连接参数限制请参考苹果官方文档\*/

conn\_params.latency = 0;

conn\_params.max\_int = 0x20; //max\_int = 0x20\*1.25ms = 40ms

conn\_params.min\_int = 0x10; //min\_int = 0x10\*1.25ms = 20ms

conn\_params.timeout = 400; //timeout = 400\*10ms = 4000ms

ESP\_LOGI(GATTS\_TAG, "ESP\_GATTS\_CONNECT\_EVT, conn\_id %d, remote %02x:%02x:%02x:%02x:%02x:%02x:", param->connect.conn\_id, param->connect.remote\_bda[0], param->connect.remote\_bda[1], param->connect.remote\_bda[2], param->connect.remote\_bda[3], param->connect.remote\_bda[4], param->connect.remote\_bda[5]);

gl\_profile\_tab[PROFILE\_A\_APP\_ID].conn\_id = param->connect.conn\_id;

//开始向对等设备发送更新连接参数。

esp\_ble\_gap\_update\_conn\_params(&conn\_params);

break;

}

case ESP\_GATTS\_DISCONNECT\_EVT: //断开事件

ESP\_LOGI(GATTS\_TAG, "ESP\_GATTS\_DISCONNECT\_EVT, disconnect reason 0x%x", param->disconnect.reason);

esp\_ble\_gap\_start\_advertising(&adv\_params);

break;

case ESP\_GATTS\_CONF\_EVT:

ESP\_LOGI(GATTS\_TAG, "ESP\_GATTS\_CONF\_EVT, status %d attr\_handle %d", param->conf.status, param->conf.handle);

if (param->conf.status != ESP\_GATT\_OK){

esp\_log\_buffer\_hex(GATTS\_TAG, param->conf.value, param->conf.len);

}

break;

case ESP\_GATTS\_OPEN\_EVT:

case ESP\_GATTS\_CANCEL\_OPEN\_EVT:

case ESP\_GATTS\_CLOSE\_EVT:

case ESP\_GATTS\_LISTEN\_EVT:

case ESP\_GATTS\_CONGEST\_EVT:

default:

break;

}

}

static void gatts\_event\_handler(esp\_gatts\_cb\_event\_t event, esp\_gatt\_if\_t gatts\_if, esp\_ble\_gatts\_cb\_param\_t \*param) //GATT事件句柄

{

/\*如果事件是注册事件，则为每个配置文件存储 gatts\_if \*/

if (event == ESP\_GATTS\_REG\_EVT) {

if (param->reg.status == ESP\_GATT\_OK) {

gl\_profile\_tab[param->reg.app\_id].gatts\_if = gatts\_if;

} else {

ESP\_LOGI(GATTS\_TAG, "Reg app failed, app\_id %04x, status %d\n",

param->reg.app\_id,

param->reg.status);

return;

}

}

/\*如果 gatts\_if 等于 profile A，则调用 profile A 回调处理程序\*/

do {

int idx;

for (idx = 0; idx < PROFILE\_NUM; idx++) {

if (gatts\_if == ESP\_GATT\_IF\_NONE || /\* ESP\_GATT\_IF\_NONE, not specify a certain gatt\_if, need to call every profile cb function \*/

gatts\_if == gl\_profile\_tab[idx].gatts\_if) {

if (gl\_profile\_tab[idx].gatts\_cb) {

gl\_profile\_tab[idx].gatts\_cb(event, gatts\_if, param);

}

}

}

} while (0);

}

void app\_main(void) //主函数

{

esp\_err\_t ret;

gpio\_pad\_select\_gpio(BLINK\_GPIO); //GPIO选择引脚

gpio\_set\_direction(BLINK\_GPIO, GPIO\_MODE\_OUTPUT); //设置输出模式

ret = nvs\_flash\_init(); //初始化存储系统

if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND) { //异常处理

ESP\_ERROR\_CHECK(nvs\_flash\_erase());

ret = nvs\_flash\_init();

}

ESP\_ERROR\_CHECK( ret );

ESP\_ERROR\_CHECK(esp\_bt\_controller\_mem\_release(ESP\_BT\_MODE\_CLASSIC\_BT));

esp\_bt\_controller\_config\_t bt\_cfg = BT\_CONTROLLER\_INIT\_CONFIG\_DEFAULT();

ret = esp\_bt\_controller\_init(&bt\_cfg); //蓝牙控制器初始化

if (ret) {

ESP\_LOGE(GATTS\_TAG, "%s initialize controller failed: %s\n", \_\_func\_\_, esp\_err\_to\_name(ret));

return;

}

ret = esp\_bt\_controller\_enable(ESP\_BT\_MODE\_BLE); //控制器启动

if (ret) {

ESP\_LOGE(GATTS\_TAG, "%s enable controller failed: %s\n", \_\_func\_\_, esp\_err\_to\_name(ret));

return;

}

ret = esp\_bluedroid\_init(); //bluedroid初始化

if (ret) {

ESP\_LOGE(GATTS\_TAG, "%s init bluetooth failed: %s\n", \_\_func\_\_, esp\_err\_to\_name(ret));

return;

}

ret = esp\_bluedroid\_enable(); //bluedroid启动

if (ret) {

ESP\_LOGE(GATTS\_TAG, "%s enable bluetooth failed: %s\n", \_\_func\_\_, esp\_err\_to\_name(ret));

return;

}

ret = esp\_ble\_gatts\_register\_callback(gatts\_event\_handler); //GATT注册回调函数

if (ret){

ESP\_LOGE(GATTS\_TAG, "gatts register error, error code = %x", ret);

return;

}

ret = esp\_ble\_gap\_register\_callback(gap\_event\_handler); //GAP注册回调

if (ret){

ESP\_LOGE(GATTS\_TAG, "gap register error, error code = %x", ret);

return;

}

ret = esp\_ble\_gatts\_app\_register(PROFILE\_A\_APP\_ID); //GATT应用注册

if (ret){

ESP\_LOGE(GATTS\_TAG, "gatts app register error, error code = %x", ret);

return;

}

esp\_err\_t local\_mtu\_ret = esp\_ble\_gatt\_set\_local\_mtu(500); //设置MTU

if (local\_mtu\_ret){

ESP\_LOGE(GATTS\_TAG, "set local MTU failed, error code = %x", local\_mtu\_ret);

}

return;

}

### 8.4.2 基于microPython开发应用

示例程序如下：

import bluetooth

import struct

from micropython import const

from machine import Pin

LED=Pin(2, Pin.OUT) #LED正极接GPIO2

LED.value(1) #初始化为点亮LED

\_IRQ\_CENTRAL\_CONNECT = const(1) #定义连接、断开和写入常量

\_IRQ\_CENTRAL\_DISCONNECT = const(2)

\_IRQ\_GATTS\_WRITE = const(3)

\_FLAG\_READ = const(0x0002) #定义读写、通知标志常量

\_FLAG\_WRITE\_NO\_RESPONSE = const(0x0004)

\_FLAG\_WRITE = const(0x0008)

\_FLAG\_NOTIFY = const(0x0010)

#串口UUID为6E400001-B5A3-F393-E0A9-E50E24DCCA9E

\_UART\_UUID = bluetooth.UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")

\_UART\_TX = (bluetooth.UUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"), \_FLAG\_READ | \_FLAG\_NOTIFY,) #发送数据

\_UART\_RX = (bluetooth.UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"), \_FLAG\_WRITE | \_FLAG\_WRITE\_NO\_RESPONSE,) #接收数据

\_UART\_SERVICE = (\_UART\_UUID,(\_UART\_TX, \_UART\_RX),) #服务定义

#广播数据定义

\_ADV\_TYPE\_FLAGS = const(0x01)

\_ADV\_TYPE\_NAME = const(0x09)

\_ADV\_TYPE\_UUID16\_COMPLETE = const(0x3)

\_ADV\_TYPE\_UUID32\_COMPLETE = const(0x5)

\_ADV\_TYPE\_UUID128\_COMPLETE = const(0x7)

\_ADV\_TYPE\_UUID16\_MORE = const(0x2)

\_ADV\_TYPE\_UUID32\_MORE = const(0x4)

\_ADV\_TYPE\_UUID128\_MORE = const(0x6)

\_ADV\_TYPE\_APPEARANCE = const(0x19)

#产生广播数据给gap\_advertise(adv\_data=...).

def advertising\_payload(limited\_disc=False, br\_edr=False, name=None, services=None, appearance=0):

payload = bytearray()

def \_append(adv\_type, value):

nonlocal payload

payload += struct.pack("BB", len(value) + 1, adv\_type) + value

\_append(

\_ADV\_TYPE\_FLAGS,

struct.pack("B", (0x01 if limited\_disc else 0x02) + (0x18 if br\_edr else 0x04)),

)

if name: #广播类型名称

\_append(\_ADV\_TYPE\_NAME, name)

if services: #服务开启

for uuid in services:

b = bytes(uuid)

if len(b) == 2:

\_append(\_ADV\_TYPE\_UUID16\_COMPLETE, b)

elif len(b) == 4:

\_append(\_ADV\_TYPE\_UUID32\_COMPLETE, b)

elif len(b) == 16:

\_append(\_ADV\_TYPE\_UUID128\_COMPLETE, b)

if appearance:

\_append(\_ADV\_TYPE\_APPEARANCE, struct.pack("<h", appearance))

return payload

class BLESimplePeripheral: #定义BLE蓝牙外设

def \_\_init\_\_(self, ble, name="ESP32"): #初始化蓝牙名称及服务

self.\_ble = ble

self.\_ble.active(True)

self.\_ble.irq(self.\_irq)

((self.\_handle\_tx, self.\_handle\_rx),) = self.\_ble.gatts\_register\_services((\_UART\_SERVICE,))

self.\_connections = set()

self.\_write\_callback = None

self.\_payload = advertising\_payload(name=name, services=[\_UART\_UUID])

self.\_advertise()

def \_irq(self, event, data):

if event == \_IRQ\_CENTRAL\_CONNECT: #连接状态

conn\_handle, \_, \_ = data

print("New connection", conn\_handle)

self.\_connections.add(conn\_handle)

elif event == \_IRQ\_CENTRAL\_DISCONNECT: #断开状态

conn\_handle, \_, \_ = data

print("Disconnected", conn\_handle)

self.\_connections.remove(conn\_handle)

self.\_advertise() #重新开启广播

elif event == \_IRQ\_GATTS\_WRITE: #写入服务

conn\_handle, value\_handle = data

value = self.\_ble.gatts\_read(value\_handle)

if value\_handle == self.\_handle\_rx and self.\_write\_callback:

self.\_write\_callback(value)

def send(self, data): #发送通知

for conn\_handle in self.\_connections:

self.\_ble.gatts\_notify(conn\_handle, self.\_handle\_tx, data)

def is\_connected(self): #连接返回

return len(self.\_connections) > 0

def \_advertise(self, interval\_us=500000): #输出广播状态

print("Starting advertising")

self.\_ble.gap\_advertise(interval\_us, adv\_data=self.\_payload)

def on\_write(self, callback): #写入回调函数

self.\_write\_callback = callback

def demo(): #定义实例

ble = bluetooth.BLE()

p = BLESimplePeripheral(ble)

def on\_rx(v):

print("RX", v)

if v==b'0': #输入0，则点亮LED

print("LED ON")

LED.value(1)

elif v==b'1': #输入1，则关闭LED

print("LED OFF")

LED.value(0)

p.on\_write(on\_rx)

if \_\_name\_\_ == "\_\_main\_\_": #运行主函数

demo()

### 8.4.3 基于Arduino开发应用

1.低功耗蓝牙

示例程序如下：

#include <Arduino.h>

#include <BLEDevice.h>

#include <BLEServer.h>

#include <BLEUtils.h>

#include <BLE2902.h>

#include <String.h>

BLECharacteristic \*pCharacteristic; //创建一个BLE特性pCharacteristic

bool deviceConnected = false; //连接否标志位

uint8\_t txValue = 0; //TX的值

String rxload = " "; //RX的值

#define SERVICE\_UUID "6E400001-B5A3-F393-E0A9-E50E24DCCA9E" //UART服务UUID

#define CHARACTERISTIC\_UUID\_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"

#define CHARACTERISTIC\_UUID\_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

class MyServerCallbacks : public BLEServerCallbacks //服务器回调

{

void onConnect(BLEServer \*pServer)

{

deviceConnected = true; //设备连接成功

};

void onDisconnect(BLEServer \*pServer)

{

deviceConnected = false; //设备连接失败

}

};

class MyCallbacks : public BLECharacteristicCallbacks //特性回调

{

void onWrite(BLECharacteristic \*pCharacteristic)

{

std::string rxValue = pCharacteristic->getValue(); //读取调试助手输入的值

if (rxValue.length() > 0)

{

rxload = "";

for (int i = 0; i < rxValue.length(); i++) //读取输入值，并放入存储变量

{

rxload += (char)rxValue[i];

Serial.print(rxValue[i]); //打印输入的值

}

Serial.println("");

}

}

};

void setupBLE(String BLEName)

{

const char \*ble\_name = BLEName.c\_str(); //将传入的BLE名字转换为指针

BLEDevice::init(ble\_name); //初始化一个蓝牙设备

BLEServer \*pServer = BLEDevice::createServer(); // 创建一个蓝牙服务器端

pServer->setCallbacks(new MyServerCallbacks()); //服务器端回调函数为MyServerCallbacks

BLEService \*pService = pServer->createService(SERVICE\_UUID); //创建一个BLE服务

pCharacteristic = pService->createCharacteristic(CHARACTERISTIC\_UUID\_TX, BLECharacteristic::PROPERTY\_NOTIFY); //创建一个(读)特征值，类型是通知

pCharacteristic->addDescriptor(new BLE2902()); //为特征添加一个描述

BLECharacteristic \*pCharacteristic = pService->createCharacteristic(CHARACTERISTIC\_UUID\_RX, BLECharacteristic::PROPERTY\_WRITE); //创建一个(写)特征，类型是写入

pCharacteristic->setCallbacks(new MyCallbacks()); //为特征添加一个回调

pService->start(); //开启服务

pServer->getAdvertising()->start(); //服务器端开始广播

Serial.println("Waiting a client connection to notify...");

}

//String val; //存储读取的值

int ledpin=2; //LED连接的引脚2

void setup()

{

Serial.begin(115200);

setupBLE("ESP32-BLE"); //设置蓝牙名称

pinMode(ledpin,OUTPUT);

}

void loop()

{

if(rxload=="0") //判断为0，点亮LED

{

digitalWrite(ledpin,HIGH);

Serial.println("LED ON!");

delay(1000);

}

else if(rxload=="1") //判断为1，熄灭LED

{

digitalWrite(ledpin,LOW);

Serial.println("LED OFF!");

delay(1000);

}

}

2.传统蓝牙

示例程序如下：

#include "BluetoothSerial.h"

BluetoothSerial SerialBT; //定义蓝牙对象

char val; //定义变量存储输入值

int ledpin=2; //LED引脚连接

void setup() {

Serial.begin(115200); //串口波特率

SerialBT.begin("ESP32-BT"); //蓝牙设备名称

Serial.println("The device started, now you can pair it with bluetooth!");

pinMode(ledpin, OUTPUT);//定义LED为输出

}

void loop() {

if (SerialBT.available()) {

SerialBT.write(Serial.read());

val=SerialBT.read();

}

if(val=='0')

{

digitalWrite(ledpin,HIGH);

Serial.println("LED ON!");

delay(1000);

}

else if(val=='1')

{

digitalWrite(ledpin,LOW);

Serial.println("LED OFF!");

delay(1000);

}

}