ESP8266 RTOS SDK

编程手册

版本 1.5

关于本手册

本文档提供 ESP8266_RTOS_SDK 的编程示例。

文档结构如下：

章节	标题	内容
第 1 章	前言	介绍 ESP8266EX。
第 2 章	概述	介绍 RTOS SDK 和编程注意事项。
第 3 章	编程示例	提供基于 ESP8266_RTOS_SDK 编程的示例。
第 4 章	附录	提供基于 ESP8266_RTOS_SDK 开发的补充说明。

发布说明

日期	版本	发布说明
2016.04	V1.4	首次发布。
2017.02	V1.5	基于 ESP8266_RTOS_SDK v1.5.0 的重大更新。
2017.05	V1.5	更新章节 2.2。

目录

1. 前言

ESP8266EX 由乐鑫公司开发，提供了一套高度集成的 Wi-Fi SoC 解决方案，其低功耗、紧凑设计和高稳定性可以满足用户的需求。

ESP8266EX 拥有完整的且自成体系的 Wi-Fi 网络功能，既能够独立应用，也可以作为从机搭载于其他主机 MCU 运行。当 ESP8266EX 独立应用时，能够直接从外接 Flash 中启动。内置的高速缓冲存储器有利于提高系统性能，并且优化存储系统。此外 ESP8266EX 只需通过 SPI/SDIO 接口或 I2C/UART 口即可作为 Wi-Fi 适配器，应用到基于任何微控制器的设计中。

ESP8266EX 集成了天线开关、射频 balun、功耗放大器、低噪放大器、过滤器和电源管理模块。这样紧凑的设计仅需极少的外部电路并且将 PCB 的尺寸降到最小。

ESP8266EX 还集成了增强版的 Tensilica's L106 钻石系列 32-bit 内核处理器，带片上 SRAM。ESP8266EX 可以通过 GPIO 外接传感器和其他设备。软件开发包 (SDK) 提供了一些应用的示例代码。

乐鑫智能互联平台 (ESCP-Espressif Systems' Smart Connectivity Platform) 表现出来的领先特征有：睡眠/唤醒模式之间的快速切换以实现节能、配合低功耗操作的自适应射频偏置、前端信号的处理功能、故障排除和射频共存机制可消除蜂窝/蓝牙/DDR/LVDS/LCD 干扰。

基于 ESP8266EX 物联网平台的 SDK 为用户提供了一个简单、快速、高效开发物联网产品的软件平台。本文旨在介绍该 SDK 的基本框架，以及相关的 API 接口。主要的阅读对象为需要在 ESP8266EX 物联网平台进行软件开发的嵌入式软件开发人员。

2. 概述

2.1. RTOS SDK 简介

SDK 为用户提供了一套数据接收、发送的函数接口，用户不必关心底层网络，如 Wi-Fi、TCP/IP 等的具体实现，只需要专注于物联网上层应用的开发，利用相应接口完成网络数据的收发即可。

ESP8266 物联网平台的所有网络功能均在库中实现，对用户不透明。用户应用的初始化功能可以在 ***user_main.c*** 中实现。

`void user_init(void)` 是上层程序的入口函数，给用户提供一个初始化接口，用户可在该函数内增加硬件初始化、网络参数设置、定时器初始化等功能。

2.2. 编程注意事项

- 建议使用定时器实现长时间的查询功能，可将定时器设置为循环调用，注意：
  - 定时器（freeRTOS timer 或 os_timer）执行函数内部请勿使用 while(1) 或其他能阻塞线程的方式延时，例如，不能在定时器回调中进行 socket send 操作，因为 send 函数会阻塞线程；
  - 定时器回调执行请勿超过 15 ms；
  - `os_timer_t` 建立的变量不能为局部变量，必须为全局变量、静态变量或 `os_malloc` 分配的指针。
- 从 ESP8266_RTOS_SDK_v1.2.0 起，无需添加宏 `ICACHE_FLASH_ATTR`，函数将默认存放在 CACHE 区，中断函数也可以存放在 CACHE 区；如需将部分频繁调用的函数定义在 RAM 中，请在函数前添加宏 `IRAM_ATTR`；
- 网络编程使用通用的 socket 编程，网络通信时，socket 请勿绑定在同一端口；
- freeRTOS 操作系统及系统自带的 API 说明请参考 *http://www.freertos.org*；
- RTOS SDK 的系统任务最高优先级为 14，创建任务的接口 `xTaskCreate` 为 freeRTOS 自带接口，使用 `xTaskCreate` 创建任务时，任务堆栈设置范围为 [176, 512]。
  - 在任务内部如需使用长度超过 60 的大数组，建议使用 `os_malloc` 和 `os_free` 的方式操作，否则，大数组将占用任务的堆空间；
  - SDK 底层已占用部分优先级：watchdog task 优先级 14，pp task 优先级 13，高精度 timer (ms) 线程优先级 12，TCP/IP task 优先级 10，freeRTOS timer 优先级 2，Wi-Fi event 优先级为 2，idle task 优先级为 0；

- 可供用户任务使用的优先级为 1 ~ 9；但注意，用户任务请勿始终占用 CPU，导致低优先级的系统任务无法执行；
- 请勿修改 ***FreeRTOSConfig.h***，此处修改头文件并不能生效，设置由 SDK 库文件决定。

3. 编程示例

3.1. RTOS SDK 目录结构

ESP8266 RTOS SDK 下载链接为：*ESP8266 RTOS SDK*。

以下为 RTOS SDK 目录结构。

- ***bin***：乐鑫官方提供的 boot 和初始化参数固件。
- ***documents***：ESP8266_RTOS_SDK 文档资料。
- ***driver_lib***：乐鑫官方提供的驱动示例代码。
- ***examples***：乐鑫提供的应用程序示例代码。
  - ***openssl_demo***：乐鑫官方提供的 OpenSSL 接口功能示例代码。
  - ***project_template***：乐鑫官方提供的工程模板示例代码。
  - ***smart_config***：乐鑫官方提供的 SmartConfig 功能示例代码。
  - ***spiffs_test***：乐鑫官方提供的 SPIFFS 文件系统功能示例代码。
  - ***websocket_demo***：乐鑫官方提供的 WebSocket 功能示例代码。
- ***include***：ESP8266_RTOS_SDK 的头文件，包含了供用户使用的软件接口和宏定义。
- ***ld***：编译时使用的链接文件，用户一般无需修改。
- ***lib***：ESP8266_RTOS_SDK 的库文件。
- ***third_party***：乐鑫开放源代码的第三方库，当前包含 freeRTOS、JSON、lwIP, mbedTLS、noPoll、OpenSSL、SPIFFS 和 SSL。
- ***tools***：工具，用户无需修改。

3.2. 基本示例

基本示例包含如下：

- 初始化
- 如何读取芯片 ID
- 如何设置 Wi-Fi 工作模式
  - ESP8266 作为 Station 模式，连接路由
  - ESP8266 作为 SoftAP 模式，可供其他 Station 连接

- Wi-Fi 连接状态的事件
- 如何读取和设置芯片 MAC 地址
- 如何扫描附近的 AP
- 如何获取 AP 的 RF 信号强度 (RSSI)
- 如何从 Flash 读写数据
- RTC 使用示例
- 非 OS SDK app 如何移植为 RTOS SDK app

3.2.1. 初始化

1. 应用程序的初始化可在 `user_main.c` 中实现。`void user_init(void)` 是上层程序的入口函数，用户可在该函数内实现初始化操作，建议打印 SDK 版本号，并设置 Wi-Fi 工作模式。

```
void user_init(void)
{
    printf("SDK version:%s\n", system_get_sdk_version());
    /* station + soft-AP mode */
    wifi_set_opmode(STATIONAP_MODE);
    ......
}
```

2. ESP8266_RTOS_SDK 默认使用 UART0 打印调试信息，默认波特率为 74880。用户可以在 `user_init` 中自定义初始化 UART，参考 `uart_init_new` 实现。

   UART 驱动示例：***\ESP8266_RTOS_SDK\driver_lib\driver\uart.c***

   以初始化 UART0 为例。定义 UART 参数：

```
    UART_ConfigTypeDef uart_config;
    uart_config.baud_rate    = BIT_RATE_74880;
    uart_config.data_bits    = UART_WordLength_8b;
    uart_config.parity       = USART_Parity_None;
    uart_config.stop_bits    = USART_StopBits_1;
    uart_config.flow_ctrl    = USART_HardwareFlowControl_None;
    uart_config.UART_RxFlowThresh = 120;
    uart_config.UART_InverseMask  = UART_None_Inverse;
    UART_ParamConfig(UART0, &uart_config);
```

   注册 UART 中断处理函数，使能 UART 中断：

```
    UART_IntrConfTypeDef uart_intr;
```

```
    uart_intr.UART_IntrEnMask = UART_RXFIFO_TOUT_INT_ENA | UART_FRM_ERR_INT_ENA |
UART_RXFIFO_FULL_INT_ENA | UART_TXFIFO_EMPTY_INT_ENA;
    uart_intr.UART_RX_FifoFullIntrThresh = 10;
    uart_intr.UART_RX_TimeOutIntrThresh = 2;
    uart_intr.UART_TX_FifoEmptyIntrThresh = 20;
    UART_IntrConfig(UART0, &uart_intr);

    UART_SetPrintPort(UART0);
    UART_intr_handler_register(uart0_rx_intr_handler);
    ETS_UART_INTR_ENABLE();
```

3. ESP8266_RTOS_SDK 支持多线程，可以建立多个任务。创建任务的接口 xTaskCreate 为 freeRTOS 自带接口，使用 xTaskCreate 创建任务时，任务堆栈设置范围为 [176, 512]。

```
    xTaskCreate(task2, "tsk2", 256, NULL, 2, NULL);
    xTaskCreate(task3, "tsk3", 256, NULL, 2, NULL);
```

注册任务执行函数，以 task2 为例：

```
void task2(void *pvParameters)
{
    printf("Hello, welcome to task2!\r\n");
    while (1) {
        ......
    }
    vTaskDelete(NULL);
}
```

4. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

5. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

```
SDK version:1.0.3(601f5cd)
mode : sta(18:fe:34:97:f7:40) + softAP(1a:fe:34:97:f7:40)
Hello, welcome to task2!
Hello, welcome to task3!
```

3.2.2. 如何读取芯片 ID

1. 软件接口介绍：

system_get_chip_id 返回硬件模组的 ESP8266 芯片 ID 值。ESP8266 芯片 ID 具有唯一性。

```
    printf("ESP8266 chip ID:0x%x\n", system_get_chip_id());
```

2. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

3. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

```
ESP8266 chip ID:0x97f740
```

3.2.3. ESP8266 作为 Station 连接路由

1. 设置 ESP8266 为 Station 模式，或者 Station+SoftAP 共存模式。

```
wifi_set_opmode(STATION_MODE);
```

2. 设置连接 AP 的信息。

```
#define DEMO_AP_SSID      "DEMO_AP"
#define DEMO_AP_PASSWORD  "12345678"
```

`wifi_station_set_config` 设置 ESP8266 Station 连接 AP 的信息。请注意将 `station_config` 中的 `bssid_set` 初始化为 0，除非需要指定 AP MAC 地址的情况才设置为 1。

`wifi_station_connect` 设置连接路由。

```
struct station_config * config = (struct station_config *)zalloc(sizeof(struct
station_config));
sprintf(config->ssid, DEMO_AP_SSID);
sprintf(config->password, DEMO_AP_PASSWORD);


wifi_station_set_config(config);
free(config);
wifi_station_connect();
```

3. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

4. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

```
connected with DEMO_AP, channel 11
dhcp client start...
ip:192.168.1.103,mask:255.255.255.0,gw:192.168.1.1
```

3.2.4. ESP8266 作为 SoftAP 模式

1. 必须先设置 ESP8266 为 SoftAP 模式，或者 Station+SoftAP 共存模式

```
wifi_set_opmode(SOFTAP_MODE);
```

2. 设置 ESP8266 SoftAP 的配置

```
    #define DEMO_AP_SSID      "DEMO_AP"
    #define DEMO_AP_PASSWORD  "12345678"
    struct softap_config *config = (struct softap_config *)zalloc(sizeof(struct
softap_config));                    // initialization

    wifi_softap_get_config(config);   // Get soft-AP config first.

    sprintf(config->ssid, DEMO_AP_SSID);
    sprintf(config->password, DEMO_AP_PASSWORD);

    config->authmode = AUTH_WPA_WPA2_PSK;
    config->ssid_len = 0;             // or its actual SSID length
    config->max_connection = 4;

    wifi_softap_set_config(config);   // Set ESP8266 soft-AP config
    free(config);
```

3. 查询连接到 ESP8266 SoftAP 的 Station 的信息

```
struct station_info * station = wifi_softap_get_station_info();
while(station){
    printf(bssid : MACSTR, ip : IPSTR/n,
                MAC2STR(station->bssid), IP2STR(&station->ip));
    station = STAILQ_NEXT(station, next);
}
wifi_softap_free_station_info();    // Free it by calling functions
```

4. ESP8266 SoftAP 的默认 IP 地址为 192.168.4.1，开发者可以更改 ESP8266 SoftAP 的 IP 地址，请注意，先关闭 DHCP server 功能。例如，设置 ESP8266 SoftAP 的 IP 地址为 192.168.5.1。

```
    wifi_softap_dhcps_stop();                    // disable soft-AP DHCP server

    struct ip_info info;
    IP4_ADDR(&info.ip, 192, 168, 5, 1);          // set IP
    IP4_ADDR(&info.gw, 192, 168, 5, 1);          // set gateway
    IP4_ADDR(&info.netmask, 255, 255, 255, 0);   // set netmask
    wifi_set_ip_info(SOFTAP_IF, &info);
```

5. 开发者可以设置 ESP8266 SoftAP 分配给连入 Station 的 IP 地址范围。例如，设置 IP 池范围为：192.168.5.100 到 192.168.5.105。请注意，设置完成后，再打开 DHCP server 功能。

```
    struct dhcps_lease dhcp_lease;
    IP4_ADDR(&dhcp_lease.start_ip, 192, 168, 5, 100);
```

```
    IP4_ADDR(&dhcp_lease.end_ip, 192, 168, 5, 105);
    wifi_softap_set_dhcps_lease(&dhcp_lease);

    wifi_softap_dhcps_start();    // enable soft-AP DHCP server
```

6. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

7. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。使用 PC 或者其他 Station 连接 ESP8266 SoftAP。

运行结果：

ESP8266 作为 SoftAP 时，如有 Station 连入，则打印如下信息：

```
station: c8:3a:35:cc:14:94 join, AID = 1
```

3.2.5. Wi-Fi 连接状态的事件

1. 当 ESP8266 作为 Station 连接路由，或者 ESP8266 作为 SoftAP 时，可由 `wifi_set_event_handler_cb` 注册 Wi-Fi 事件的回调。

2. 示例代码：

```
void wifi_handle_event_cb(System_Event_t *evt)
{
    printf("event %x\n", evt->event_id);
    switch (evt->event_id) {
    case EVENT_STAMODE_CONNECTED:
        printf("connect to ssid %s, channel %d\n",
                    evt->event_info.connected.ssid,
                    evt->event_info.connected.channel);
        break;
    case EVENT_STAMODE_DISCONNECTED:
        printf("disconnect from ssid %s, reason %d\n",
                    evt->event_info.disconnected.ssid,
                    evt->event_info.disconnected.reason);
        break;
    case EVENT_STAMODE_AUTHMODE_CHANGE:
        printf("mode: %d -> %d\n",
                    evt->event_info.auth_change.old_mode,
                    evt->event_info.auth_change.new_mode);
```

```
            break;
        case EVENT_STAMODE_GOT_IP:
            printf("ip:" IPSTR ",mask:" IPSTR ",gw:" IPSTR,
                        IP2STR(&evt->event_info.got_ip.ip),
                        IP2STR(&evt->event_info.got_ip.mask),
                        IP2STR(&evt->event_info.got_ip.gw));
            printf("\n");
            break;
        case EVENT_SOFTAPMODE_STACONNECTED:
            printf("station: " MACSTR "join, AID = %d\n",
                        MAC2STR(evt->event_info.sta_connected.mac),
                        evt->event_info.sta_connected.aid);
            break;
        case EVENT_SOFTAPMODE_STADISCONNECTED:
            printf("station: " MACSTR "leave, AID = %d\n",
                        MAC2STR(evt->event_info.sta_disconnected.mac),
                        evt->event_info.sta_disconnected.aid);
            break;
        default:
            break;
        }
}
void user_init(void)
{
    // TODO: add user's own code here....
    wifi_set_event_handler_cb(wifi_handle_event_cb);
}
```

3. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

4. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

例如，ESP8266 Station 连入路由器的过程如下：

```
wifi_handle_event_cb : event 1
connect to ssid Demo_AP, channel 1
wifi_handle_event_cb : event 4
IP:192.168.1.126,mask:255.255.255.0,gw:192.168.1.1
wifi_handle_event_cb : event 2
disconnect from ssid Demo_AP, reason 8
```

3.2.6. 读取和设置 ESP8266 MAC 地址

1. ESP8266 可以工作在 Station+SoftAP 共存模式，Station 接口和 SoftAP 接口的 MAC 地址不同。乐鑫保证芯片出厂的默认 MAC 地址的唯一性，用户如果重新设置 MAC 地址，则需要自行确保 MAC 地址的唯一性。
2. 设置 ESP8266 为 Station+SoftAP 共存模式。

```
wifi_set_opmode(STATIONAP_MODE);
```

3. 分别读取 Station 接口和 SoftAP 接口的 MAC 地址。

```
wifi_get_macaddr(SOFTAP_IF, sofap_mac);
wifi_get_macaddr(STATION_IF, sta_mac);
```

4. 分别设置 Station 接口和 SoftAP 接口的 MAC 地址。MAC 地址的设置不保存到 Flash 中，先使能对应接口，才能设置该接口的 MAC 地址。

```
char sofap_mac[6] = {0x16, 0x34, 0x56, 0x78, 0x90, 0xab};
char sta_mac[6] = {0x12, 0x34, 0x56, 0x78, 0x90, 0xab};

wifi_set_macaddr(SOFTAP_IF, sofap_mac);
wifi_set_macaddr(STATION_IF, sta_mac);
```

5. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。
6. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

> ⚠ **注意：**
>
> - *ESP8266 SoftAP* 和 *Station* 的 *MAC* 地址并不相同，请勿设置为同一 *MAC* 地址。
> - *ESP8266 MAC* 地址第一个字节的 *bit 0* 不能为 *1*。例如，*MAC* 地址可设置为 *"1a:fe:36:97:d5:7b"*，但不能设置为 *"15:fe:36:97:d5:7b"*。

运行结果：

```
ESP8266 station MAC :18:fe:34:97:f7:40
ESP8266 soft-AP MAC :1a:fe:34:97:f7:40
ESP8266 station new MAC :12:34:56:78:90:ab
ESP8266 soft-AP new MAC :16:34:56:78:90:ab
```

3.2.7. 扫描附近 AP

1. 设置 ESP8266 为 Station 模式或者 Station+SoftAP 共存模式。

```
wifi_set_opmode(STATIONAP_MODE);
```

2. 扫描附近 AP。

wifi_station_scan 如果第一个参数为 NULL，则扫描附近所有 AP；如果第一个参数设置了特定 SSID、channel 等信息，则返回特定的 AP。

```
wifi_station_scan(NULL,scan_done);
```

扫描 AP 完成的回调函数。

```
void scan_done(void *arg, STATUS status)
{
    uint8 ssid[33];
    char temp[128];

    if (status == OK) {
        struct bss_info *bss_link = (struct bss_info *)arg;

        while (bss_link != NULL) {
            memset(ssid, 0, 33);
            if (strlen(bss_link->ssid) <= 32)
                memcpy(ssid, bss_link->ssid, strlen(bss_link->ssid));
            else
                memcpy(ssid, bss_link->ssid, 32);

            printf("(%d,\"%s\",%d,\""MACSTR"\",%d)\r\n",
                bss_link->authmode, ssid, bss_link->rssi,
                MAC2STR(bss_link->bssid),bss_link->channel);
            bss_link = bss_link->next.stqe_next;
        }
    } else {
        printf("scan fail !!!\r\n");
    }
}
```

3. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

4. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

```
Hello, welcome to scan-task!
scandone
(0,"ESP_A13319",-41,"1a:fe:34:a1:33:19",1)
(4,"sscgov217",-75,"80:89:17:79:63:cc",1)
(0,"ESP_97F0B1",-46,"1a:fe:34:97:f0:b1",1)
(0,"ESP_A1327E",-36,"1a:fe:34:a1:32:7e",1)
```

3.2.8. 获取 AP 的 射频信号强度 (RSSI)

1. 如果 ESP8266 Station 未连接路由，可以使用指定 SSID 扫描 AP 的方式，在扫描返回的信息中获得射频信号强度。

   指定目标 AP 的 SSID：

```
#define DEMO_AP_SSID      “DEMO_AP"
```

   指定 SSID 扫描特定 AP，扫描完成回调 scan_done 可参考前例。

```
struct scan_config config;

memset(&config, 0, sizeof(config));
config.ssid = DEMO_AP_SSID;

wifi_station_scan(&config,scan_done);
```

2. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。
3. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

```
Hello, welcome to scan-task!
scandone
(3,"DEMO_AP",-49,"aa:5b:78:30:46:0a",11)
```

3.2.9. 从 Flash 读取数据

1. Flash 读写数据，要求必须 4 字节对齐。例如，从 Flash 读取数据：

```
#define SPI_FLASH_SEC_SIZE        4096
uint32 value;
uint8 *addr = (uint8 *)&value;
spi_flash_read(0x3E * SPI_FLASH_SEC_SIZE, (uint32 *)addr, 4);
printf("0x3E sec:%02x%02x%02x%02x\r\n", addr[0], addr[1], addr[2], addr[3]);
```

2. 向 Flash 写入数据同理，要求 4 字节对齐。先使用 spi_flash_erase_sector 擦除待写入区域，再使用接口 spi_flash_write 写入即可。例如，

```
uint32 data[M];
// TODO: fit in the data
spi_flash_erase_sector(N);
spi_flash_write(N*4*1024, data, M*4);
```

3. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。
4. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

```
read data from 0x3E000 : 05 00 04 02
```

3.2.10. RTC 使用示例

1. 软件复位 (system_restart) 时，系统时间归零，但是 RTC 时间仍然继续。但是如果外部硬件通过 EXT_RST 脚或者 CHIP_EN 脚，将芯片复位后（包括 Deep-sleep 定时唤醒的情况），RTC 时钟会复位。具体如下：

   - 外部复位 (EXT_RST)：RTC memory 不变，RTC timer 寄存器从零计数
   - watchdog reset：RTC memory 不变，RTC timer 寄存器不变
   - system_restart：RTC memory 不变，RTC timer 寄存器不变
   - 电源上电：RTC memory 随机值，RTC timer 寄存器从零计数
   - CHIP_EN 复位：RTC memory 随机值，RTC timer 寄存器从零计数

   例如，system_get_rtc_time 返回 10（表示 10 个 RTC 周期），system_rtc_clock_cali_proc 返回 5.75（表示 1 个 RTC 周期为 5.75 μs），则实际时间为 10 x 5.75 = 57.5 μs。

```
rtc_t = system_get_rtc_time();
cal = system_rtc_clock_cali_proc();
os_printf("cal: %d.%d \r\n", ((cal*1000)>>12)/1000, ((cal*1000)>>12)%1000 );
```

   读写 RTC memory，请注意读写 RTC memory 只能 4 字节对齐。

```
 typedef struct {
     uint64 time_acc;
     uint32 magic ;
     uint32 time_base;
} RTC_TIMER_DEMO;
system_rtc_mem_read(64, &rtc_time, sizeof(rtc_time));
```

2. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。
3. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

```
rtc_time: 1613921
cal: 6.406
```

3.2.11. 非 OS SDK app 如何移植为 RTOS SDK app

1. 定时器 (timer) 的程序可通用，无需改动。
2. 回调函数 (callback) 的程序可通用，无需改动。

3. 创建任务的方式，需要改动。RTOS SDK 创建任务使用的是 freeRTOS 自带的软件接口：`xTaskCreate`。

非 OS SDK 创建任务：

```
#define Q_NUM   (10)
ETSEvent test_q[Q_NUM];

void test_task(ETSEvent *e)
{
    switch(e->sig)
    {
        case 1:
            func1(e->par);
            break;
        case 2:
            func2();
            break;
        case 3:
            func3();
            break;
        default:
            break;
    }
}

void func_send_Sig(void)
{
    ETSSignal sig = 2;
    system_os_post(2,sig,0);
}

void task_ini(void)
{
    system_os_task(test_task, 2, test_q, Q_NUM);
    // test_q is the corresponding array of test_task.
    // (2) is the priority of test_task.
    // Q_NUM is the queue length of test_task.
}
```

RTOS SDK 创建任务：

```
#define Q_NUM   (10)
xQueueHandle test_q;
xTaskHandle test_task_hdl;
```

```
void test_task(void *pvParameters)
{
    int *sig;
    for(;;) {
        if(pdTRUE == xQueueReceive(test_q, &sig, (portTickType)portMAX_DELAY) ){
            vTaskSuspendAll();
            switch(*sig)
            {
                case 1:
                    func1();
                    break;
                case 2:
                    func2();
                    break;
                default:
                    break;
            }
            free(sig);
            xTaskResumeAll();
        }
    }
}

void func_send_Sig(void)
{
    int *evt = (int *)malloc(sizeif(int));
    *evt = 2;
    if(xQueueSend(test_q,&evt,10/portTick_RATE_MS)!=pdTRUE){
        os_printf("test_q is full\n");
    }
// It is the address of parameter that stored in test_q, so int *evt and int *sig can be
other types.
}

void task_ini(void)
{
    test_q = xQueueCreate(Q_NUM,sizeof(void *));
    xTaskCreate(test_task,(signed portCHAR *)"test_task", 512, NULL, (1), &test_task_hdl );
// 512 means the heap size of this task, 512 * 4 byte.
// NULL is a pointer of parameter to test_task.
// (1) is the priority of test_task.
// test_task_hdl is the pointer of the task of test_task.
}
```

3.3. 网络协议示例

ESP8266_RTOS_SDK 的网络协议即 socket 编程，示例包含如下：

- UDP 传输示例
- TCP 连接示例
  - ESP8266 作为 TCP client
  - ESP8266 作为 TCP server

3.3.1. UDP 传输

1. 设定 UDP 本地端口号，例如，端口号为 1200。

```
#define UDP_LOCAL_PORT  1200
```

2. 创建 socket。

```
LOCAL int32 sock_fd;
struct sockaddr_in server_addr;

memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(UDP_LOCAL_PORT);
server_addr.sin_len = sizeof(server_addr);

do{
     sock_fd = socket(AF_INET, SOCK_DGRAM, 0);
     if (sock_fd == -1) {
          printf("ESP8266 UDP task > failed to create sock!\n");
          vTaskDelay(1000/portTICK_RATE_MS);
     }
}while(sock_fd == -1);

printf("ESP8266 UDP task > socket OK!\n");
```

3. 绑定本地端口。

```
do{
     ret = bind(sock_fd, (struct sockaddr *)&server_addr, sizeof(server_addr));
     if (ret != 0) {
           printf("ESP8266 UDP task > captdns_task failed to bind sock!\n");
           vTaskDelay(1000/portTICK_RATE_MS);
```

```
        }
    }while(ret != 0);


    printf("ESP8266 UDP task > bind OK!\n");
```

4. 收发 UDP 数据。

```
while(1){
    memset(udp_msg, 0, UDP_DATA_LEN);
    memset(&from, 0, sizeof(from));


    setsockopt(sock_fd, SOL_SOCKET, SO_RCVTIMEO, (char *)&nNetTimeout, sizeof(int));
    fromlen = sizeof(struct sockaddr_in);
    ret = recvfrom(sock_fd, (uint8 *)udp_msg, UDP_DATA_LEN, 0,(struct sockaddr *)&from,
(socklen_t *)&fromlen);
    if (ret > 0) {
        printf("ESP8266 UDP task > recv %d Bytes from %s, Port %d\n" ,ret,
inet_ntoa(from.sin_addr), ntohs(from.sin_port));


        sendto(sock_fd,(uint8*)udp_msg, ret, 0, (struct sockaddr *)&from, fromlen);
       }
}


if(udp_msg){
    free(udp_msg);
    udp_msg = NULL;
}
close(sock_fd);
```

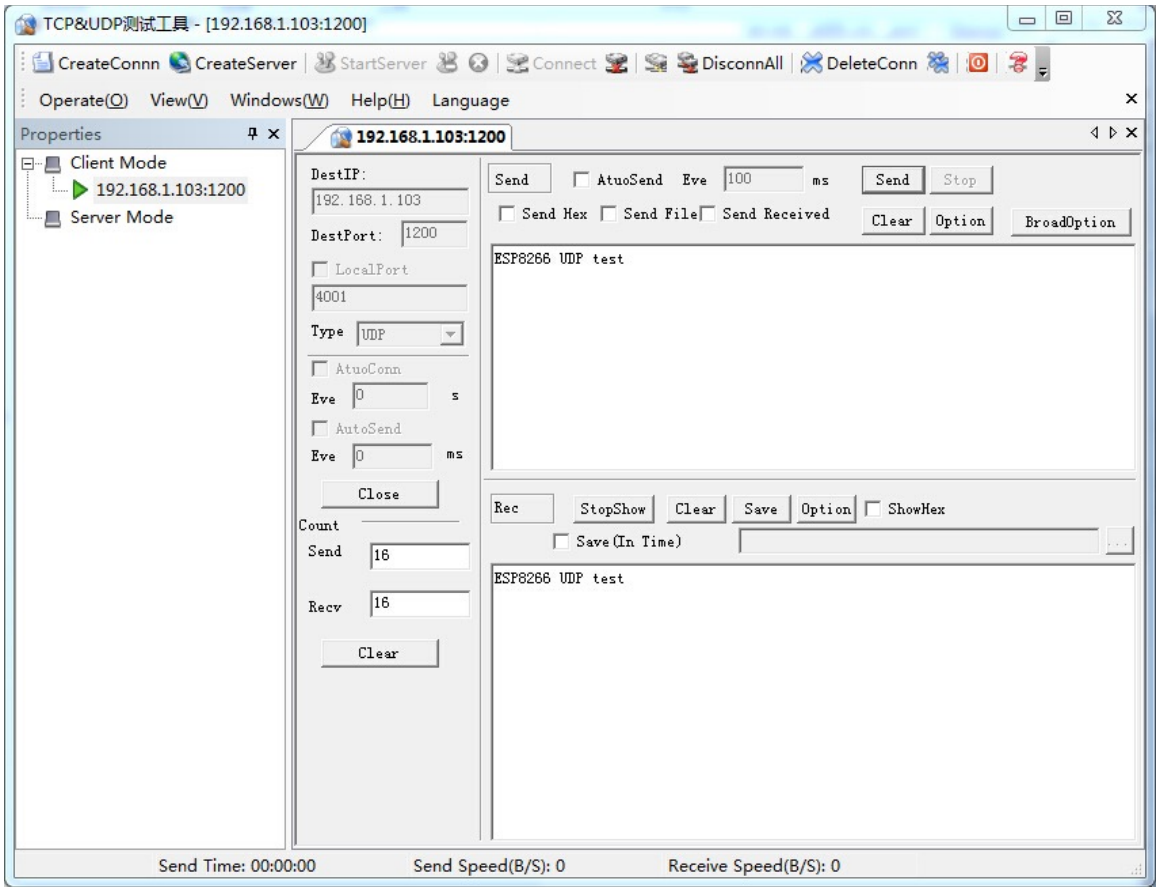
5. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

6. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

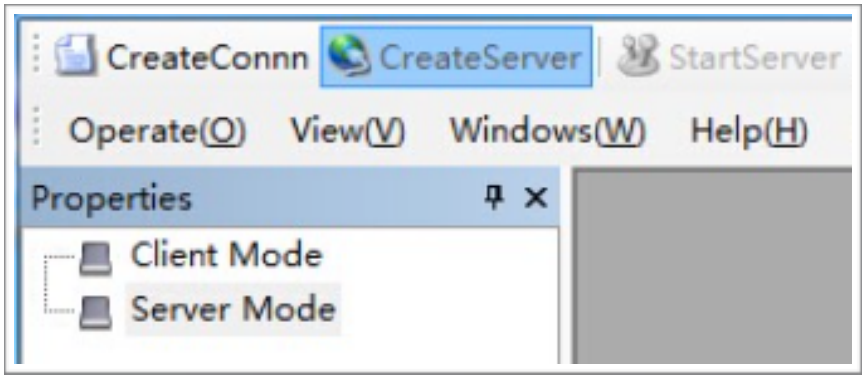
```
ip:192.168.1.103,mask:255.255.255.0,gw:192.168.1.1
ESP8266 UDP task > socket ok!
ESP8266 UDP task > bind ok!
ESP8266 UDP task > recv data 16 Bytes from 192.168.1.112, Port 57233
```

在 PC 端使用网络调试工具，建立 UDP 通信，向 ESP8266 UDP 端口发送数据 `ESP8266 UDP test`，ESP8266 收到 UDP 数据后，回复同样的消息给 PC。



3.3.2. TCP Client

1. 设置 ESP8266 Station 连接路由，示例可参考前例。
2. 使用网络调试工具建立一个 TCP server




```
#define SERVER_IP        "192.168.1.124"
#define SERVER_PORT        1001
```

3. socket 编程实现 TCP 通信。

   建立 socket：

```
sta_socket = socket(PF_INET, SOCK_STREAM, 0);
if (-1 == sta_socket) {
    close(sta_socket);
    vTaskDelay(1000 / portTICK_RATE_MS);
    printf("ESP8266 TCP client task > socket fail!\n");
    continue;
}
printf("ESP8266 TCP client task > socket ok!\n");
```

建立 TCP 连接：

```
bzero(&remote_ip, sizeof(struct sockaddr_in));
remote_ip.sin_family = AF_INET;
remote_ip.sin_addr.s_addr = inet_addr(SERVER_IP);
remote_ip.sin_port = htons(SERVER_PORT);

if (0 != connect(sta_socket, (struct sockaddr *)(&remote_ip), sizeof(struct sockaddr)))
{
    close(sta_socket);
    vTaskDelay(1000 / portTICK_RATE_MS);
    printf("ESP8266 TCP client task > connect fail!\n");
    continue;
}
printf("ESP8266 TCP client task > connect ok!\n");
```

TCP 通信，发送数据包：

```
if (write(sta_socket, pbuf, strlen(pbuf) + 1) < 0){
    close(sta_socket);
    vTaskDelay(1000 / portTICK_RATE_MS);
    printf("ESP8266 TCP client task > send fail\n");
    continue;
}
printf("ESP8266 TCP client task > send success\n");
free(pbuf);
```

TCP 通信，接收数据包：

```
char *recv_buf = (char *)zalloc(128);
while ((recbytes = read(sta_socket , recv_buf, 128)) > 0) {
    recv_buf[recbytes] = 0;
    printf("ESP8266 TCP client task > recv data %d bytes!\nESP8266 TCP client task >
%s\n", recbytes, recv_buf);
}
free(recv_buf);
```

```
        if (recbytes <= 0) {
            close(sta_socket);
            printf("ESP8266 TCP client task > read data fail!\n");
        }
```

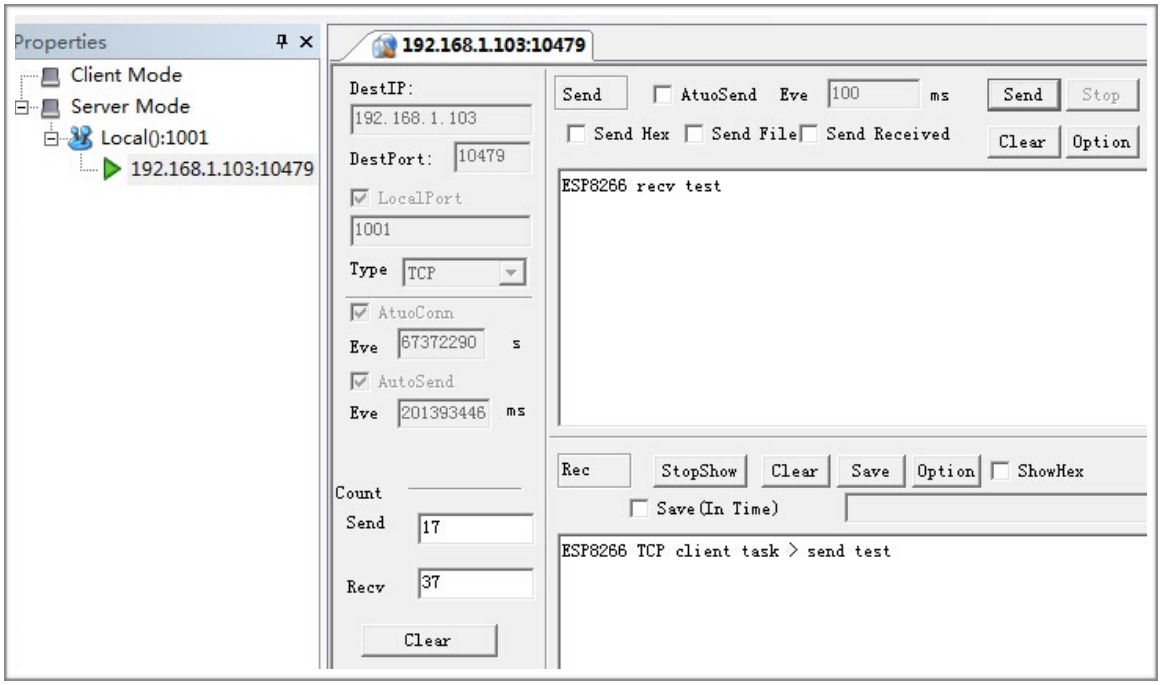
4. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

5. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

```
ESP8266 TCP client task > socket ok!
ESP8266 TCP client task > connect ok!
ESP8266 TCP client task > send success
ESP8266 TCP client task > recv data 17 bytes!
ESP8266 TCP client task > ESP8266 recv test
```

网络调试工具端建立的 TCP server 与 ESP8266 成功 TCP 通信。



3.3.3. TCP Server

1. 建立 TCP server，绑定本地端口。

```
        #define SERVER_PORT 1002
        int32 listenfd;
        int32 ret;
        struct sockaddr_in server_addr,remote_addr;
        int stack_counter=0;
```

```
/* Construct local address structure */
memset(&server_addr, 0, sizeof(server_addr)); /* Zero out structure */
server_addr.sin_family = AF_INET;            /* Internet address family */
server_addr.sin_addr.s_addr = INADDR_ANY;   /* Any incoming interface */
server_addr.sin_len = sizeof(server_addr);
server_addr.sin_port = htons(httpd_server_port); /* Local port */

/* Create socket for incoming connections */
do{
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    if (listenfd == -1) {
        printf("ESP8266 TCP server task > socket error\n" );
        vTaskDelay(1000/portTICK_RATE_MS);
    }
}while(listenfd == -1);

printf("ESP8266 TCP server task > create socket: %d\n", server_sock);

/* Bind to the local port */
do{
    ret = bind(listenfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
    if (ret != 0) {
        printf("ESP8266 TCP server task > bind fail\n" );
        vTaskDelay(1000/portTICK_RATE_MS);
    }
}while(ret != 0);

printf("ESP8266 TCP server task > port:%d\n" ,ntohs(server_addr.sin_port));
```

建立 TCP server 侦听：

```
do{
    /* Listen to the local connection */
    ret = listen(listenfd, MAX_CONN);
    if (ret != 0) {
        printf("ESP8266 TCP server task > failed to set listen queue!\n");
        vTaskDelay(1000/portTICK_RATE_MS);
    }
}while(ret != 0);

printf("ESP8266 TCP server task > listen ok\n" );
```

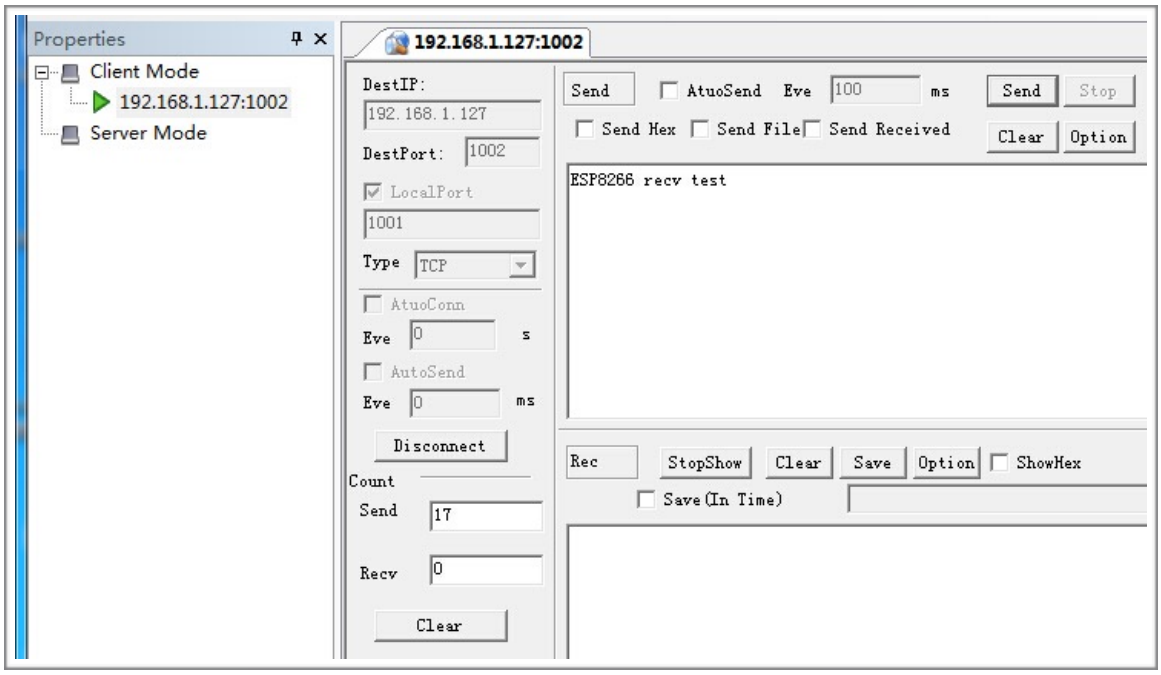
等待 TCP client 连入，建立 TCP 通信，接收数据包：

```
    int32 client_sock;
    int32 len = sizeof(struct sockaddr_in);


    for (;;) {
        printf("ESP8266 TCP server task > wait client\n" );
        /*block here waiting remote connect request*/
        if ((client_sock = accept(listenfd, (struct sockaddr *)&remote_addr, (socklen_t
*)&len)) < 0) {
            printf("ESP8266 TCP server task > accept fail\n");
            continue;
        }
        printf("ESP8266 TCP server task > Client from %s %d\n",
inet_ntoa(remote_addr.sin_addr), htons(remote_addr.sin_port));


        char *recv_buf = (char *)zalloc(128);
        while ((recbytes = read(client_sock , recv_buf, 128)) > 0) {
            recv_buf[recbytes] = 0;
            printf("ESP8266 TCP server task > read data success %d!\nESP8266 TCP server task
> %s\n", recbytes, recv_buf);
        }
        free(recv_buf);


        if (recbytes <= 0) {
            printf("ESP8266 TCP server task > read data fail!\n");
            close(client_sock);
        }
    }
```

2. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

3. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

4. 使用网络调试工具建立一个 TCP client，连接到 ESP8266 TCP server，并发送数据。



运行结果：

```
ip:192.168.1.127,mask:255.255.255.0,gw:192.168.1.1
got ip !!!
Hello, welcome to ESP8266 TCP server task!
ESP8266 TCP server task > create socket: 0
ESP8266 TCP server task > bind port: 1002
ESP8266 TCP server task > listen ok
ESP8266 TCP server task > wait client
ESP8266 TCP server task > Client from 192.168.1.108 1001
ESP8266 TCP server task > read data success 17!
ESP8266 TCP server task > ESP8266 recv test
```

3.4. 高级应用示例

ESP8266_RTOS_SDK 的高级示例包含如下：

- OTA 固件升级
- 强制休眠示例
- SPIFFS 文件系统
- SSL 应用示例

3.4.1. OTA 固件升级

OTA 固件升级，是指 ESP8266 硬件模块通过 Wi-Fi 无线网络从服务器下载新版本固件，实现固件升级。

> ⚠ **注意：**
>
> 由于擦除 *Flash* 的过程较慢，边下载边擦写 *Flash* 可能占用较长时间，影响网络传输的稳定性。因此，请先调用 `spi_flash_erase_sector` 将 *Flash* 待升级区域擦除，再建立网络连接，从 *OTA server* 下载新固件，调用 `spi_flash_write` 写入 *Flash*。

1. 搭建用户自己的云端服务器，或者使用乐鑫的云端服务器。
2. 将新版本固件上传到云端服务器。
3. 代码说明如下：

   设置 ESP8266 模块连接到路由器，示例可参考前述。在 `upgrade_task` 中查询 ESP8266 Station 是否获取到 IP 地址。

```
wifi_get_ip_info(STATION_IF, &ipconfig);

/* check the IP address or net connection state*/
while (ipconfig.ip.addr == 0) {
    vTaskDelay(1000 / portTICK_RATE_MS);
    wifi_get_ip_info(STATION_IF, &ipconfig);
}
```

- ESP8266 获取到 IP 地址后，与云端服务器建立连接，可参考前例 socket 编程。
- `system_upgrade_flag_set` 设置升级状态标志：
  - `UPGRADE_FLAG_IDLE`：空闲状态。
  - `UPGRADE_FLAG_START`：开始升级。
  - `UPGRADE_FLAG_FINISH`：从服务器下载新版本固件完成。
- `system_upgrade_userbin_check` 查询当前正在运行的是 ***user1.bin*** 还是 ***user2.bin***，若正在运行 ***user1.bin*** 则下载 ***user2.bin***，否则下载 ***user1.bin***。

```
system_upgrade_init();
system_upgrade_flag_set(UPGRADE_FLAG_START);
```

- 向服务器发送下载请求，从服务器接收新固件数据，并写入 Flash。

```
if(write(sta_socket,server->url,strlen(server->url)+1) < 0) {
    ......
}
```

```
while((recbytes = read(sta_socket ,precv_buf,UPGRADE_DATA_SEG_LEN)) > 0) {
    // write the new firmware into flash by spi_flash_write
}
```

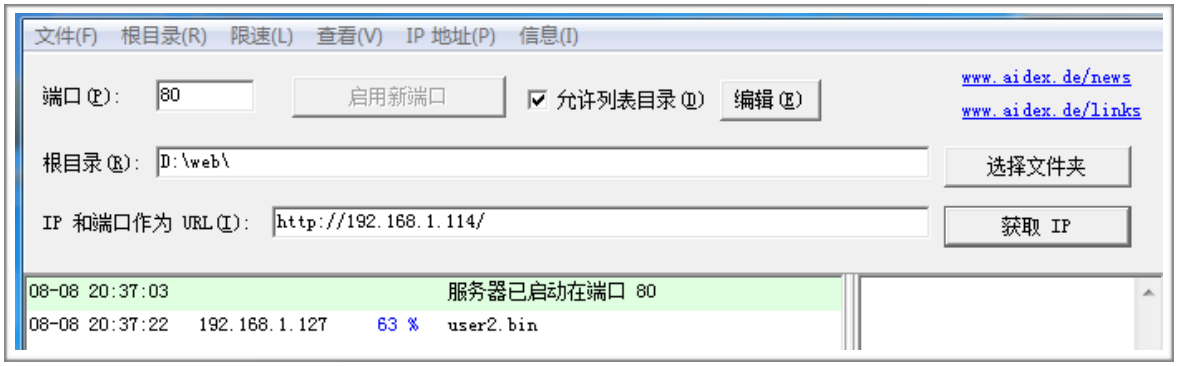
- 设置一个软件定时器检查升级状态，如果定时器超时，仍未完成从服务器下载固件，则判断升级失败，将升级状态置回空闲，释放相关资源，退出本次升级。
- 若从服务器下载固件成功，升级状态设置为 `UPGRADE_FLAG_FINISH`，在此状态下，调用软件接口 `system_upgrade_reboot`，可控制 ESP8266 重新启动，运行新版本固件。

4. 编译应用程序，生成固件烧录到 ESP8266 硬件模组中。

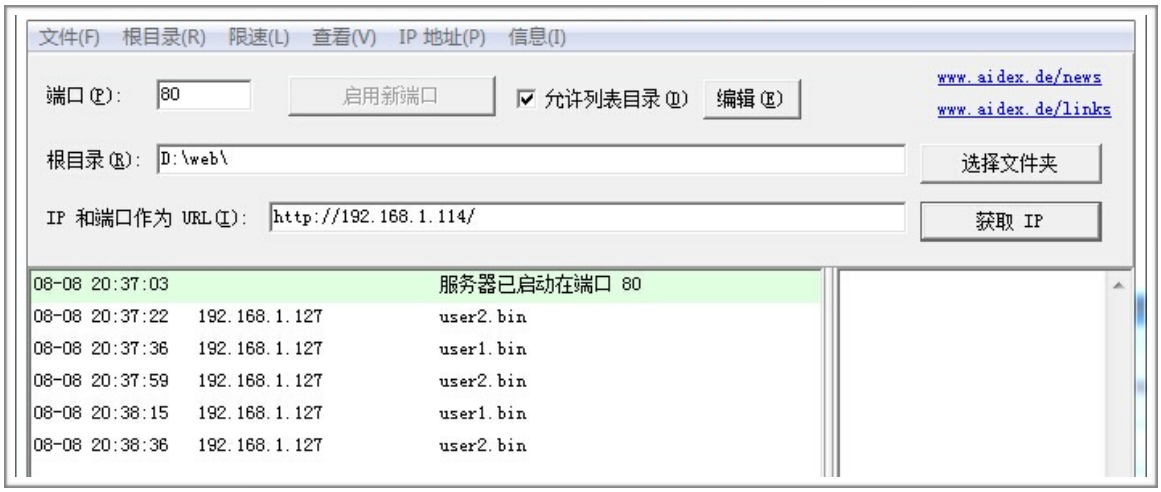
5. 将硬件模组断电，切换到运行模式，重新上电，运行应用程序。

运行结果：

- 在 PC 端使用 webserver 工具，建立一个服务器，并上传 ***user1.bin*** 和 ***user2.bin***，ESP8266 烧录固件后，默认先运行 ***user1.bin***，从服务器下载 ***user2.bin***。



- 下载 ***user2.bin*** 成功后，ESP8266 模块自动重启，运行新固件 ***user2.bin***，***user1.bin*** 直到下次 FOTA 升级前都不会再被运行。当服务器上出现新的固件，且用户需要再次更新时，用户发送升级请求，下载 ***user1.bin*** 的新固件，模块自动重启后，运行 ***user1.bin***。如此循环。



- ESP8266 升级流程的打印信息：

```
connected with Demo_AP, channel 6

dhcp client start...

ip:192.168.1.127,mask:255.255.255.0,gw:192.168.1.1

Hello, welcome to client!

socket ok!

connect ok!

GET /user2.bin HTTP/1.0

Host: "192.168.1.114":80

Connection: keep-alive

Cache-Control: no-cache

User-Agent: Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
30.0.1599.101 Safari/537.36

Accept: */*

Accept-Encoding: gzip,deflate,sdch

Accept-Language: zh-CN,zh;q=0.8


send success

read data success!

upgrade file download start.

read data success!

totallen = 1460

read data success!

totallen = 2920

read data success!

... ...
```

3.4.2. 强制休眠示例

强制休眠接口，在需要的情况下强制关闭 RF 电路以降低功耗。

> ⚠ **注意：**
>
> - 强制休眠接口调用后，并不会立即休眠，而是等到系统 *idle task* 执行时才进入休眠。
> - 请参考下述示例使用。

示例一：Modem-sleep 模式（关闭射频）

```
    #define FPM_SLEEP_MAX_TIME          0xFFFFFFF
void fpm_wakup_cb_func1(void)
{
    wifi_fpm_close();                   // disable force sleep function
    wifi_set_opmode(STATION_MODE);      // set station mode
    wifi_station_connect();                         // connect to AP
}
void user_func(…)
{
    …
    wifi_station_disconnect();
    wifi_set_opmode(NULL_MODE);                     // set WiFi mode to null mode.
    wifi_fpm_set_sleep_type(MODEM_SLEEP_T);         // modem sleep
    wifi_fpm_open();                                // enable force sleep
#ifdef SLEEP_MAX
/* For modem sleep, FPM_SLEEP_MAX_TIME can only be wakened by calling wifi_fpm_do_wakeup. */
    wifi_fpm_do_sleep(FPM_SLEEP_MAX_TIME);
#else
// wakeup automatically when timeout.
    wifi_fpm_set_wakeup_cb(fpm_wakup_cb_func1) ;    // Set wakeup callback
     wifi_fpm_do_sleep(50*1000);
#endif
    …
}

#ifdef SLEEP_MAX
void func1(void)
{
    wifi_fpm_do_wakeup();
    wifi_fpm_close();                               // disable force sleep function
    wifi_set_opmode(STATION_MODE);                  // set station mode
    wifi_station_connect();                         // connect to AP
}
#endif
```

示例二：Light-sleep 模式（关闭射频和 CPU）

强制进入 Light-sleep 模式，即强制关闭射频和 CPU，需要设置一个回调函数，以便唤醒后程序继续运行。

```
void fpm_wakup_cb_func1(void)
{
    wifi_fpm_close();                 // disable force sleep function
    wifi_set_opmode(STATION_MODE);    // set station mode
    wifi_station_connect();           // connect to AP
}

#ifndef SLEEP_MAX
// Wakeup till time out.
void user_func(…)
{
    wifi_station_disconnect();

    wifi_set_opmode(NULL_MODE);                     // set WiFi mode to null mode.
    wifi_fpm_set_sleep_type(LIGHT_SLEEP_T);         // light sleep
    wifi_fpm_open();                                // enable force sleep
    wifi_fpm_set_wakeup_cb(fpm_wakup_cb_func1);     // Set wakeup callback
    wifi_fpm_do_sleep(50*1000);
}
#else
// Or wakeup by GPIO
void user_func(…)
{
     wifi_station_disconnect();
    wifi_set_opmode(NULL_MODE);                     // set WiFi mode to null mode.
    wifi_fpm_set_sleep_type(LIGHT_SLEEP_T);         // light sleep
    wifi_fpm_open();                                // enable force sleep
    PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTCK_U,3);
    gpio_pin_wakeup_enable(13, GPIO_PIN_INTR_LOLEVEL);

    wifi_fpm_set_wakeup_cb(fpm_wakup_cb_func1);     // Set wakeup callback
    wifi_fpm_do_sleep(0xFFFFFFF);
}
#endif
```

3.4.3. SPIFFS 文件系统应用

1. 调用软件接口 esp_spiffs_init，初始化 SPIFFS 文件系统。

```
void spiffs_fs1_init(void)
{
    struct esp_spiffs_config config;

    config.phys_size = FS1_FLASH_SIZE;
    config.phys_addr = FS1_FLASH_ADDR;
    config.phys_erase_block = SECTOR_SIZE;
    config.log_block_size = LOG_BLOCK;
    config.log_page_size = LOG_PAGE;
    config.fd_buf_size = FD_BUF_SIZE * 2;
    config.cache_buf_size = CACHE_BUF_SIZE;

    esp_spiffs_init(&config);
}
```

2. 打开并创建一个文件，写入数据。

```
char *buf="hello world";
char out[20] = {0};

int pfd = open("myfile", O_TRUNC | O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
if(pfd <= 3) {
    printf("open file error \n");
}
int write_byte = write(pfd, buf, strlen(buf));
if (write_byte <= 0)
{
    printf("write file error \n");
}
close(pfd);
```

3. 通过文件系统读取数据。

```
open("myfile",O_RDWR);
if (read(pfd, out, 20) < 0)
    printf("read errno \n");
close(pfd);
printf("--> %s <--\n", out);
```

3.4.4. SSL 应用示例

1. 定义将连接的 SSL server IP 和端口。

```
#define SSL_SERVER_IP     “115.29.202.58”
#define SSL_SERVER_PORT   443

esp_test *pTestParamer = (esp_test *)zalloc(sizeof(esp_test));

pTestParamer->ip.addr = ipaddr_addr(SSL_SERVER_IP);
pTestParamer->port = server_port;
```

2. 创建 SSL client 的任务。

```
xTaskCreate(esp_client, "esp_client", 1024, (void*)pTestParamer, 4, NULL);
```

3. 参考前文示例，设置 ESP8266 Station 连接路由。在 SSL client 的任务中，先检查 ESP8266 Station 获得了 IP 地址，再建立 SSL 连接。

```
struct ip_info ipconfig;
wifi_get_ip_info(STATION_IF, &ipconfig);

while (ipconfig.ip.addr == 0) {
    vTaskDelay(1000 / portTICK_RATE_MS);
    wifi_get_ip_info(STATION_IF, &ipconfig);
}
```

4. 建立 socket 连接。

```
client_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (client_fd < 0){
    printf("create with the socket err\n");
}
memset(&client_addr, 0, sizeof(client_addr));
client_addr.sin_family = AF_INET;
client_addr.sin_port = htons(port);
client_addr.sin_addr.s_addr = sin_addr;

if(connect(client_fd, (struct sockaddr *)&client_addr, sizeof(client_addr))< 0)
    printf("connect with the host err\n");
```

5. 创建 SSL 上下文内容 (context)。SSL 需要较大内存，建议调用 `system_get_free_heap_size` 检查可用内存容量。

```
uint32 options = SSL_SERVER_VERIFY_LATER|SSL_DISPLAY_CERTS|SSL_NO_DEFAULT_KEY;
if ((ssl_ctx = ssl_ctx_new(options, SSL_DEFAULT_CLNT_SESS)) == NULL){
```

```
    printf("Error: Client context is invalid\n");
}
printf("heap_size %d\n" ,system_get_free_heap_size());
```

6. 如果开发者需要认证功能，如果不使用 SPIFFS 文件系统，请先运行 ***esp_iot_sdk_freertos\tools\ make_cert.py*** 脚本，生成 ***esp_ca_cert.bin*** 文件，烧录到 Flash 自定义地址。

   以下代码示例从 Flash 读取 SSL 密钥和证书信息。

```
uint8 flash_offset = 0x78;   // Example : Flash address 0x78000

if (ssl_obj_option_load(ssl_ctx, SSL_OBJ_RSA_KEY, "XX.key" , password, flash_offset)){
    printf("Error: the Private key is undefined.\n");
}

if (ssl_obj_option_load(ssl_ctx, SSL_OBJ_X509_CERT, "XX.cer" , NULL, flash_offset)){
    printf("Error: the Certificate is undefined.\n");
}
```

   如果使用 SPIFFS 文件系统，请运行工具 spiffy（*https://github.com/xlfe/spiffy*，注意，此工具内的 ***spiffs_config.h*** 文件必须修改成与 RTOS SDK 中的 ***spiffs_config.h*** 一致），生成符合 SPIFFS 格式的 ***spiffs_rom.bin*** 文件，烧录到 Flash SPIFFS 配置的地址，可参考前例 `esp_spiffs_init`。

   以下代码示例使用 SPIFFS 文件系统的情况下，读取 SSL 密钥和证书信息。

```
if (ssl_obj_load(ssl_ctx, SSL_OBJ_RSA_KEY, "XX.key" , password)){
    printf("Error: the Private key is undefined.\n");
}

if (ssl_obj_load(ssl_ctx, SSL_OBJ_X509_CERT, "XX.cer" , NULL)){
    printf("Error: the Certificate is undefined.\n");
}
```

7. 开始 SSL client 握手。

```
ssl = ssl_client_new(ssl_ctx, client_fd, NULL, 0);
if (ssl != NULL){
    printf("client handshake start\n");
}
```

8. 检查 SSL 连接状态。

```
if ((res = ssl_handshake_status(ssl)) == SSL_OK){
    ... ...
```

```
}
```

9. 如果 SSL 握手成功，则可以释放证书，节省内存空间。

```
const char *common_name = ssl_get_cert_dn(ssl,SSL_X509_CERT_COMMON_NAME);
if (common_name){
    printf("Common Name:\t\t\t%s\n", common_name);
}
display_session_id(ssl);
display_cipher(ssl);
quiet = true;
os_printf("client handshake ok! heapsize %d\n",system_get_free_heap_size());
x509_free(ssl->x509_ctx);
ssl->x509_ctx=NULL;
os_printf("certificate free ok! heapsize %d\n" ,system_get_free_heap_size());
```

10.发送 SSL 数据。

```
uint8 buf[512];
bzero(buf, sizeof(buf));
sprintf(buf,httphead,"/", "iot.espressif.cn",port);
os_printf("%s\n", buf);
if(ssl_write(ssl, buf, strlen(buf)+1) < 0) {
    ssl_free(ssl);
    ssl_ctx_free(ssl_ctx);
    close(client_fd);
    vTaskDelay(1000 / portTICK_RATE_MS);
    os_printf("send fail\n");
    continue;
}
```

11.接收 SSL 数据。

```
while((recbytes = ssl_read(ssl, &read_buf)) >= 0) {
    if(recbytes == 0){
        vTaskDelay(500 / portTICK_RATE_MS);
        continue;
    }
    os_printf("%s\n", read_buf);
}

free(read_buf);
if(recbytes < 0) {
    os_printf("ERROR:read data fail! recbytes %d\r\n",recbytes);
```

```
        ssl_free(ssl);
        ssl_ctx_free(ssl_ctx);

        close(client_fd);
        vTaskDelay(1000 / portTICK_RATE_MS);
    }
```

运行结果：

```
ip:192.168.1.127,mask:255.255.255.0,gw:192.168.1.1
-----BEGIN SSL SESSION PARAMETERS-----
4ae116a6a0445b369f010e0ea5420971497e92179a6602c8b5968c1f35b60483
-----END SSL SESSION PARAMETERS-----
CIPHER is AES128-SHA
client handshake ok! heapsize 38144
certificate free ok! heapsize 38144
GET / HTTP/1.1
Host: iot.espressif.cn:443
Connection: keep-alive
......
```

A. 附录

A.1. Sniffer 说明

关于 sniffer 的详细说明，请参考《*ESP8266 技术参考*》。

A.2. ESP8266 SoftAP 和 Station 信道定义

虽然 ESP8266 支持 SoftAP+Station 共存模式，但是 ESP8266 实际只有一个硬件信道。因此在 SoftAP+Station 模式时，ESP8266 SoftAP 会动态调整信道值与 ESP8266 Station 一致。

这个限制会导致 ESP8266 SoftAP+Station 模式时一些行为上的不便，用户请注意。例如：

情况一

1. 如果 ESP8266 Station 连接到一个路由（假设路由信道号为 6）
2. 通过接口 `wifi_softap_set_config` 设置 ESP8266 SoftAP
3. 若设置值合法有效，该 API 将返回 `true`，但信道号仍然会自动调节成与 ESP8266 Station 接口一致，在这个例子里也就是信道号为 6。因为 ESP8266 在硬件上只有一个信道，由 ESP8266 Station 与 SoftAP 接口共用。

情况二

1. 调用接口 `wifi_softap_set_config` 设置 ESP8266 SoftAP（例如信道号为 5）
2. 其他 Station 连接到 ESP8266 SoftAP
3. 将 ESP8266 Station 连接到路由（假设路由信道号为 6）
4. ESP8266 SoftAP 将自动调整信道号与 ESP8266 Station 一致（信道 6）
5. 由于信道改变，之前连接到 ESP8266 SoftAP 的 Station 的 Wi-Fi 连接断开。

情况三

1. 其他 Station 与 ESP8266 SoftAP 建立连接
2. 如果 ESP8266 Station 一直尝试扫描或连接某路由，可能导致 ESP8266 SoftAP 端的连接断开。

3. 因为 ESP8266 Station 会遍历各个信道查找目标路由，意味着 ESP8266 其实在不停切换信道，ESP8266 SoftAP 的信道也因此在不停更改。这可能导致 ESP8266 SoftAP 端的原有连接断开。
4. 这种情况，用户可以通过设置定时器，超时后调用 `wifi_station_disconnect` 停止 ESP8266 Station 不断连接路由的尝试；或者在初始配置时，调用 `wifi_station_set_reconnect_policy` 和 `wifi_station_set_auto_connect` 禁止 ESP8266 Station 尝试重连路由。

A.3. ESP8266 启动信息说明

ESP8266 启动时，将从 UART0 以波特率 74880 打印如下启动信息：

```
 ets Jan  8 2013,rst cause:2, boot mode:(3,6)



load 0x4010f000, len 1264, room 16

tail 0

chksum 0x42

csum 0x42
```

其中可供用户参考的启动信息说明如下：

启动信息	说明
rst cause	1：上电
	2：外部复位
	4：硬件看门狗复位
boot mode 第一个参数	1：ESP8266 处于 UART-down 模式，可通过 UART 下载固件
	3：ESP8266 处于 Flash-boot 模式，从 Flash 启动运行
chksum	chksum 与 csum 值相等，表示启动过程中 Flash 读取正确

乐鑫 IOT 团队

www.espressif.com