Espressif IoT SDK: Programming Guide

Status	Released
Current version	v1.0.0
Author	Fei Yu
Completion Date	2015.03.14
Reviewer	JG Wu
Completion Date	2015.03.14

[√] CONFIDENTIAL

[ ] INTERNAL

[ ] PUBLIC

Version Info

Date	Version	Author	Comments/Changes
2013.12.25 ~2014.7.10	0.1~0.8	JG Wu / Han Liu/ Fei Yu	Draft and changes
2014.8.13	0.9	Fei Yu	1.Revise espconn APIs； 2.Add sniffer APIs； 3.Add system_get_chip_id； 4.Add APIs to read/ write mac&ip；
2014.9.23	0.9.1	Fei Yu	1、Add system_deep_sleep; 2、Revise APIs to read/write flash; 3、Add APIs for AP_CHE 4、Revise UDP APIs
2014.11.20	0.9.3	Fei Yu	Introduce esp_iot_sdk_v0.9.3 1、Add APIs for auto connecting to router when power on or not; 2、Add APIs for UART swap; 3、Add APIs for DHCP; 4、Add APIs for RTC
2014.12.19	0.9.4	Fei Yu	Introduce esp_iot_sdk_v0.9.4 1、Add APIs for sleep type; 2、Add APIs for igmp
2015.01.22	0.9.5	Fei Yu	Introduce esp_iot_sdk_v0.9.5 1、Revised upgrade APIs： 2、Add more DHCP APIs： 3、Add API to get recorded AP info 4、Add smart config APIs； 5、Add API to block TCP receiving data 6、Add API for AT commands
2015.03.14	1.0.0	FeiYu	Introduce esp_iot_sdk_v1.0.0 1、Add APIs about CPU frequency; 2、Add mac filter API for sniffer; 3、Add API to set broadcast interface; 4、Update smart config APIs； 5、Add API of user-define AT commands 6、Add APIs of AT commands

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member Logo is a trademark of the Wi-Fi Alliance.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Table of Contents

1. Foreword

The SDK based on ESP8266 IoT platform offers users an easy, fast and efficient way to develop IoT devices.

The programming guide provides overview of the SDK as well as details on the API. It is written for embedded software developers to help them program on ESP8266 IoT platform.

2. Overview

The SDK provides a set of interfaces for data receive and transmit functions over the Wi-Fi and TCP/IP layer so programmers can focus on application development on the high level. Users can easily make use of the corresponding interfaces to realize data receive and transmit.

All networking functions on the ESP8266 IoT platform are realized in the library, and are not transparent to users. Instead, users can initialize the interface in user_main.c

void usre_init(void) is the default method provided. Users can add functions like firmware initialization, network parameters setting, and timer initialization in the interface.

The SDK provides an API to handle json, and users can also use self-defined data types to handle the them.

3. Application Programming Interface (APIs)

3.1. Timer

Locate in “\esp_iot_sdk\include\osapi.h”

3.1.1. os_timer_arm

Function: arm timer

Prototype:

void os_timer_arm(ETSTimer *ptimer, uint32_t milliseconds, boolrepeat_flag)

Input parameters:

ETSTimer*ptimer——Timer structure

uint32_t milliseconds——Timing, Unit: milisecond

boolrepeat_flag——Whether to repeat the timing

Return:

null

3.1.2. os_timer_disarm

Function: Disarm timer

Prototype:

void os_timer_disarm (ETSTimer *ptimer)

Input parameters:

ETSTimer*ptimer——Timer structure

Return:

null

3.1.3. os_timer_setfn

Function: Set timer callback function

Prototype:

void os_timer_setfn (ETSTimer *ptimer, ETSTimerFunc *pfunction, void *parg)

Input parameters:

ETSTimer*ptimer——Timer structure

TESTimerFunc*pfunction——timer callback function

void*parg——callback function parameter

Return:

null

3.2. System APIs

3.2.1. system_restore

Function: Reset to default settings

Prototype:

void system_restore(void)

Input parameters:

null

Return:

null

3.2.2. system_restart

Function: Restart

Prototype:

void system_restart(void)

Input parameters:

null

Return:

null

3.2.3. system_timer_reinit

Function: Reinitiate the timer when you need to use microsecond timer

Not es: 1. Define USE_US_TIMER;

2. Put system_timer_reinit at the beginning and user_init in the first sentence.

Function definition:

void system_timer_reinit (void)

Input parameters:

null

Return:

Null

3.2.4. system_init_done_cb

Function：call this API in user_init to register a system-init-done callback.

Note：wifi_station_scan need to be called after system init done and station enable.

Prototype：

void system_init_done_cb(init_done_cb_t cb)

Parameter：

init_done_cb_t cb - system-init-done callback

Return：

NULL

Example：

```
void to_scan(void)
{
   wifi_station_scan(NULL,scan_done);
}

void user_init(void)
{
wifi_set_opmode(STATION_MODE);
system_init_done_cb(to_scan);
}
```

3.2.5. system_get_chip_id

Function: Get chip id

Prototype:

uint32 system_get_chip_id (void)

Input parameters:

null

Return:

Chip id

3.2.6. system_deep_sleep

Function: Set for deep-sleep mode. Device in deep-sleep mode automatically, every X us wake up once. Everytime device wakes up, it starts from user_init.

Prototype：

void system_deep_sleep(uint32 time_in_us)

parameters：

uint32 time_in_us – during the time (us) device is in deep-sleep

Return：

NULL

Note:

Hardware has to support deep-sleep wake up (XPD_DCDC connects to EXT_RSTB with 0R).

system_deep_sleep(0) ，set no wake up timer，connect a GPIO to pin RST, the chip will wake up by a falling-edge on pin RST

3.2.7. system_deep_sleep_set_option

Function: Call this API before system_deep_sleep to set what the chip will do when deep-sleep wake up.

Note: following “init data” means esp_init_data_default.bin.

Prototype：

bool system_deep_sleep_set_option(uint8 option)

Parameter：

uint8 option - option=0, init data byte 108 is valuable;
option>0, init data byte 108 is valueless.

More details as follows：

deep_sleep_set_option(0), RF_CAL or not after deep-sleep wake up, depends on init data byte 108.

deep_sleep_set_option(1), RF_CAL after deep-sleep wake up, there will be large current.

deep_sleep_set_option(2) , no RF_CAL after deep-sleep wake up, there will only be small current.

deep_sleep_set_option(4) , disable RF after deep-sleep wake up, just like modem sleep, there will be the smallest current.

Return:

True - succeed；

False - fail.

3.2.8. system_set_os_print

Function: Turn on/off print logFunction

Function definition:

void system_set_os_print (uint8onoff)

Input parameters:

uint8 onoff — turn on/off print function;

0x00 : print function off

0x01: print function on

Defaut: print function on

Return:

null

3.2.9. system_print_meminfo

Function: Print memory information, including data/rodata/bss/heap

Function definition:

void system_print_meminfo (void)

Input parameters:

null

Return:

null

3.2.10. system_get_free_heap_size

Function: Get free heap size

Function definition:

uint32 system_get_free_heap_size(void)

Input parameters:

null

Return:

uint32 ——available heap size

3.2.11. system_os_task

Function: Set up tasks

Function definition:

bool system_os_task(os_task_t task, uint8 prio, os_event_t *queue, uint8 qlen)

Input parameters:

os_task_t task——task function

uint8 prio——task priority. 3 priorities are supported, 0/1/2, 0 is the lowest priority.

os_event_t *queue——message queue pointer

uint8 qlen——message queue depth

Return：

True - succeed；

False - fail.

Example:

```
#define SIG_RX    0
#define TEST_QUEUE_LEN    4
os_event_t *testQueue;
void test_task (os_event_t *e)
{
    switch (e->sig) {
case SIG_RX:
    os_printf("sig_rx %c\n",    (char)e->par);
    break;
default:
break;
    }
```

```
}
void task_init(void)
{
testQueue=(os_event_t *)os_malloc(sizeof(os_event_t)*TEST_QUEUE_LEN);
system_os_task(test_task,USER_TASK_PRIO_0,testQueue,TEST_QUEUE_
LEN);
}
```

3.2.12. system_os_post

Function: send message to task

Function definition:

bool system_os_post (uint8 prio, os_signal_t sig, os_param_t par)

Input parameters:

uint8 prio——task priority, corresponding to that you set up

os_signal_t sig——message type

os_param_t par——message parameters

Return：

True - succeed;

False - fail.

Refer to the example above:

```
void task_post(void)
{
system_os_post(USER_TASK_PRIO_0, SIG_RX, 'a');
}
```

Printout: sig_rx a

3.2.13. system_get_time

Function：Get system time (us).

Prototype：

uint32 system_get_time(void)

Parameter：

Null

Return：

System time in us.

3.2.14. system_get_rtc_time

Function：Get RTC time , count by RTC clock period.

Example: If system_get_rtc_time returns 10 (means 10 RTC cycles), system_rtc_clock_cali_proc returns 5 (means 5us per RTC cycle), then real time is 10 x 5 = 50 us.

Note: System time will return to zero because of deep sleep or system_restart, but RTC still goes on.

Prototype：

uint32 system_get_rtc_time(void)

Parameter：

Null

Return：

RTC time.

3.2.15. system_rtc_clock_cali_proc

Function: Get RTC clock period.

Prototype：

uint32 system_rtc_clock_cali_proc(void)

Parameter：

Null

Return：

RTC clock period (in us), bit11~ bit0 are decimal.

Note: RTC demo in Appendix.

3.2.16. system_rtc_mem_write

Function: During deep sleep, only RTC still working, so maybe we need to save some user data in RTC memory. Only “user data” area can be used by user.

|_ _ _ _ _system data _ _ _ _ _|_ _ _ _ _ _ _ _ _ _ user data _ _ _ _ _ _ _ _ _|
| 256 bytes | 512 bytes |

Note: RTC memory is 4 bytes aligned for read and write operations. Parameter “des_addr” means block number(4 bytes per block). So, if we want to save some data at the beginning of user data area, “des_addr” will be 256/4 = 64, “save_size” will be data length.

Prototype：

bool system_rtc_mem_write (uint32 des_addr, void * src_addr, uint32 save_size)

Parameter：

uint32 des_addr —— destination address (block number) in RTC memory, des_addr >=64

void * src_addr —— data pointer.

uint32 save_size —— data length (byte)

Return：

True，succeed；False，fail.

3.2.17. system_rtc_mem_read

Function: Read user data from RTC memory. Only “user data” area here can be used by user.

|_ _ _ _ _system data _ _ _ _ _|_ _ _ _ _ _ _ _ _ _ user data _ _ _ _ _ _ _ _ _|
| 256 bytes | 512 bytes |

Note: RTC memory is 4 bytes aligned for read and write operations. Parameter "src_addr" means block number(4 bytes per block). So, if we want to read some data from the beginning of user data area, "src_addr" will be 256/4 = 64, "save_size" will be data length.

Prototype：

bool system_rtc_mem_read (uint32 src_addr, void * des_addr, uint32 save_size)

Parameter：

uint32 src_addr —— source address (block number) in rtc memory，src_addr >=64

void * des_addr —— data pointer

uint32 save_size —— data length，byte

Return：

True， succeed；False，fail.

3.2.18. system_uart_swap

Function: UART0 swap. Use MTCK as UART0 RX，MTDO as UART0 TX, so ROM log won't output from this new UART0. We also need to use MTDO(U0CTS) and MTCK(U0RTS) as UART0 in hardware.

Prototype：

void system_uart_swap (void)

Parameter：

NULL

Return：

NULL

3.2.19. system_get_boot_version

Function：Get version info of boot

Prototype：

Uint8 system_get_boot_version (void)

Parameter：

NULL

Return：

Version info of boot .

Note：

If boot version >= 3 , you could enable boot enhance mode（refer to system_restart_enhance）

3.2.20. system_get_userbin_addr

Function：Get address of the current running user bin（user1.bin or user2.bin）.

Prototype：

uint32 system_get_userbin_addr (void)

Parameter：

NULL

Return：

Start address info of the current running user bin.

3.2.21. system_get_boot_mode

Function：Get boot mode.

Prototype：

Uint8 system_get_boot_mode (void)

Parameter：

NULL

Return：

#define SYS_BOOT_ENHANCE_MODE 0

#define SYS_BOOT_NORMAL_MODE 1

Note：

Enhance boot mode：can load and run FW at any address;

Normal boot mode：can only load and run normal user1.bin（or user2.bin）.

3.2.22. system_restart_enhance

Function：restart system, and enter enhance boot mode.

Prototype：

bool system_restart_enhance(uint8 bin_type, uint32 bin_addr)

Parameter：

uint8 bin_type – type of bin

#define SYS_BOOT_NORMAL_BIN 0 // user1.bin or user2.bin

#define SYS_BOOT_TEST_BIN 1 // can only be Espressif test bin

uint32 bin_addr – start address of bin file

Return：

TRUE，succeed；FALSE，fail。

Note：

SYS_BOOT_TEST_BIN is for factory test，you can apply for the test bin from Espressif.

3.2.23. system_update_cpu_freq

Function：Set CPU frequency。Default is 80MHz。

Prototype：

bool system_update_cpu_freq(uint8 freq)

Parameter：

uint8 freq – CPU frequency

#define SYS_CPU_80MHz 80

#define SYS_CPU_160MHz 160

Return：

TRUE，succeed；FALSE，fail。

3.2.24. system_get_cpu_freq

Function：Get CPU frequency。

Prototype：

Uint8 system_get_cpu_freq(void)

Parameter：

NULL

Return：

CPU frequency，unit ：MHz。

3.3. SPI Flash Related APIs

3.3.1. spi_flash_get_id

Function: Get id info of spi flash

Prototype：

Uint32 spi_flash_get_id (void)

Parameters：

Null

Return：

SPI Flash id

3.3.2. spi_flash_erase_sector

Function: erase sector in flash

Note: More details in document “Espressif IOT Flash RW Operation”

Prototype：

SpiFlashOpResult spi_flash_erase_sector (uint16 sec)

Parameters：

uint16 sec - Sector number, the count starts at sector 0, 4KB per sector.

Return：

Typedef enum{

SPI_FLASH_RESULT_OK,

SPI_FLASH_RESULT_ERR,

SPI_FLASH_RESULT_TIMEOUT

}SpiFlashOpResult；

3.3.3. spi_flash_write

Function：Write data to flash。

Note: More details in document “”

Prototype：

SpiFlashOpResult spi_flash_write (uint32 des_addr, uint32 *src_addr, uint32 size)

Parameters：

uint32 des_addr - destination address in flash.

uint32 *src_addr – source address of the data.

Uint32 size - length of data

Return：

Typedef enum{

SPI_FLASH_RESULT_OK,

SPI_FLASH_RESULT_ERR,

SPI_FLASH_RESULT_TIMEOUT

}SpiFlashOpResult；

3.3.4. spi_flash_read

Function： Read data from flash.

Note: More details in document “”

Prototype：

SpiFlashOpResult spi_flash_read(uint32 src_addr, uint32 * des_addr, uint32 size)

Parameters：

uint32 src_addr- source address in flash

uint32 * des_addr – destination address to keep data.

Uint32 size - length of data

Return：

```
Typedef enum{
    SPI_FLASH_RESULT_OK,
    SPI_FLASH_RESULT_ERR,
    SPI_FLASH_RESULT_TIMEOUT
}SpiFlashOpResult;
```

3.4. WIFI Related APIs

3.4.1. wifi_get_opmode

Function: get wifi working mode

Function definition:

uint8 wifi_get_opmode (void)

Input parameters:

null

Return:

Wifi working modes:

0x01 means STATION_MODE,

0x02 means SOFTAP_MODE,

0x03 means STATIONAP_MODE.

3.4.2. wifi_set_opmode

Function: set wifi working mode as STATION, SOFTAP or STATION+SOFTAP

Note:

Versions before esp_iot_sdk_v0.9.2, need to call system_restart() after this api;

after esp_iot_sdk_v0.9.2, need not to restart.

Function definition:

bool wifi_set_opmode (uint8 opmode)

Input parameters:

uint8 opmode——Wifi working modes: 0x01 means STATION_MODE, 0x02

means SOFTAP_MODE, 0x03 means STATIONAP_MODE.

Return:

True - succeed;

False - fail.

3.4.3. wifi_station_get_config

Function: get wifi station configuration

Function definition:

bool wifi_station_get_config (struct station_config *config)

Input parameters:

struct station_config *config——wifi station configuration pointer

Return:

True - succeed；

False - fail.

3.4.4. wifi_station_set_config

Function: Set wifi station configuration

Note: If wifi_station_set_config is called in user_init , there is no need to call wifi_station_connect after that, ESP8266 will connect to router automatically; otherwise, need wifi_station_connect to connect.

In general, station_config.bssid_set need to be 0, otherwise it will check bssid which is the mac address of AP.

Function definition:

bool wifi_station_set_config (struct station_config *config)

Input parameters:

struct station_config *config——wifi station configuration pointer

Return:

True - succeed；

False - fail.

3.4.5. wifi_station_connect

Function: wifi station connected AP

Note：if ESP8266 has already connected to a router，it's necessary to call wifi_station_disconnect first，then call wifi_station_connect to connect.

Function definition:

bool wifi_station_connect (void)

Input parameters:

null

Return:

True - succeed；

False - fail.

3.4.6. wifi_station_disconnect

Function: wifi station disconnected AP

Function definition:

bool wifi_station_disconnect (void)

Input parameters:

null

Return:

True - succeed；

False - fail.

3.4.7. wifi_station_get_connect_status

Function: get the connection status between wifi station and AP

Function definition:

uint8 wifi_station_get_connect_status (void)

Input parameters:

null

Return:

```
enum{
    STATION_IDLE = 0,
    STATION_CONNECTING,
    STATION_WRONG_PASSWORD,
    STATION_NO_AP_FOUND,
    STATION_CONNECT_FAIL,
    STATION_GOT_IP
};
```

3.4.8. wifi_station_scan

Function: Scan AP

Note: Do not call this API in user_init. This API need to be called after system initialize done and station enable.

Function definition:

```
bool wifi_station_scan (struct scan_config *config, scan_done_cb_t cb);
```

Structure：

```
struct scan_config{
    uint8 *ssid;          // AP's ssid
    uint8 *bssid;         // AP's bssid
    uint8 channel;        //scan a specific channel
    uint8 show_hidden;    //scan APs of which ssid is hidden.
};
```

Parameters:

struct scan_config *config – AP config for scan

if config = Null， scan all APs

if config.ssid、config.bssid are null，config.channel isn't null, ESP8266

will scan the specific channel.

scan_done_cb_t cb - callback function after scan

Return:

True - succeed；

False - fail.

3.4.9. scan_done_cb_t

Function: scan callback function

Function definition:

void scan_done_cb_t (void *arg, STATUS status);

Input parameters:

void *arg——information of APs that be found, refer to struct bss_info

STATUS status——get status

Return:

NULL

Example:

```
wifi_station_scan(&config, scan_done);
static void ICACHE_FLASH_ATTR
scan_done(void *arg, STATUS status)
{
  if (status == OK)
  {
    struct bss_info *bss_link = (struct bss_info *)arg;
    bss_link = bss_link->next.stqe_next;//ignore first
    ……
    }
}
```

3.4.10. wifi_station_ap_number_set

Function: Set the number of APs that can be recorded for ESP8266 station. When ESP8266 station connects to an AP, ESP8266 keeps a record of this AP. Record id starts counting from 0.

Prototype：

bool wifi_station_ap_number_set (uint8 ap_number);

Parameters：

uint8 ap_number—— how many APs can be recorded（MAX: 5）

eg: if ap_number is 5, record id : 0 ~ 4

Return：

True - succeed；

False - fail.

3.4.11. wifi_station_get_ap_info

Function: Get information of APs recorded by ESP8266 station.

Prototype：

uint8 wifi_station_get_ap_info(struct station_config config[])

Parameters：

struct station_config config[] — information of APs, array size has to be 5.

Return：

How many APs that is actually recorded.

Example：

```
struct station_config config[5];
int i = wifi_station_get_ap_info(config);
```

3.4.12. wifi_station_ap_change

Function: ESP8266 station change to connect to the AP which is recorded in specific id.

Prototype：

bool wifi_station_ap_change (uint8 current_ap_id);

Parameters：

uint8 current_ap_id — AP's record id, start counting from 0.

Return：

True - succeed；

False - fail.

3.4.13. wifi_station_get_current_ap_id

Function: Get the current record id of AP.

Prototype：

Uint8 wifi_station_get_current_ap_id ();

Parameter：

Null

Return：

The record id of the AP which ESP8266 is connected with right now.

3.4.14. wifi_station_get_auto_connect

Function: Check whether ESP8266 station will connect to AP (which is recorded) automatically or not when power on.

Prototype：

uint8 wifi_station_get_auto_connect(void)

Parameter：

Null

Return：

0 , won't connect to AP automatically;

Non-0 , will connect to AP automatically.

3.4.15. wifi_station_set_auto_connect

Function: Set whether ESP8266 station will connect to AP (which is recorded) automatically or not when power on.

Note: Call this API in user_init , it is effective in this current power on; call it in other place, it will be effective in next power on.

Prototype:

bool wifi_station_set_auto_connect(uint8 set)

Parameter:

uint8 set — Automatically connect or not;

0, won't connect automatically; 1, will connect automatically.

Return:

True , succeed; False , fail.

3.4.16. wifi_station_dhcpc_start

Function: Enable ESP8266 station dhcp client.

Note: DHCP default enable。

Prototype:

bool wifi_station_dhcpc_start(void)

Parameter:

Null

Return:

True , succeed; False , fail.

3.4.17. wifi_station_dhcpc_stop

Function: Disable ESP8266 station dhcp client.

Note: DHCP default enable.

Prototype：

bool wifi_station_dhcpc_stop(void)

Parameter：

Null

Return：

True ，succeed；False ，fail.

3.4.18. wifi_station_dhcpc_status

Function：Get ESP8266 station dhcp client status.

Prototype：

enum dhcp_status wifi_station_dhcpc_status(void)

Parameter：

Null

Return：

```
enum dhcp_status {
    DHCP_STOPPED,
    DHCP_STARTED
};
```

3.4.19. wifi_softap_get_config

Function: set wifi softap configuration

Function definition:

bool wifi_softap_get_config(struct softap_config *config)

Parameter：

struct softap_config *config——ESP8266 softap config

Return：

True - succeed；

False - fail.

3.4.20. wifi_softap_set_config

Function: set wifi softap configuration

Function definition:

bool wifi_softap_set_config (struct softap_config *config)

Parameter：

struct softap_config *config—— wifi softap configuration pointer

Return：

True - succeed；

False - fail.

3.4.21. wifi_softap_get_station_info

Function: get connected station devices under softap mode, including mac and ip

Function definition：

struct station_info * wifi_softap_get_station_info(void)

Input parameters:

null

Return:

struct station_info*——station information structure

3.4.22. wifi_softap_free_station_info

Function: free the struct station_info by calling the wifi_softap_get_station_info function

Function definition:

void wifi_softap_free_station_info(void)

Input parameters:

null

Return:

null

Examples of getting mac and ip information:

Method 1:

```
struct station_info * station = wifi_softap_get_station_info();
struct station_info * next_station;
while(station){
        os_printf("bssid : "MACSTR", ip : "IPSTR"\n",
MAC2STR(station->bssid), IP2STR(&station->ip));
        next_station = STAILQ_NEXT(station, next);
        os_free(station);    // Free it directly
        station = next_station;
}
```

Method 2：

```
struct station_info * station = wifi_softap_get_station_info();
while(station){
   os_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
```

3.4.23. wifi_softap_dhcps_start

Function：Enable ESP8266 softAP dhcp server.

Note：DHCP default enable.

Prototype：

```
bool wifi_softap_dhcps_start(void)
```

Parameter：

Null

Return：

True - succeed;

False - fail.

3.4.24. wifi_softap_dhcps_stop

Function：Disable ESP8266 softAP dhcp server.

Note：DHCP default enable.

Prototype：

bool wifi_softap_dhcps_stop(void)

Parameter：

Null

Return：

True - succeed;

False - fail.

3.4.25. wifi_softap_set_dhcps_lease

Function：Set the IP range that can be got from ESP8266 softAP dhcp server.

Note：This API need to be called during DHCP server disable.

Prototype：

bool wifi_softap_set_dhcps_lease(struct dhcps_lease *please)

Parameter：

struct dhcps_lease {

uint32 start_ip;

uint32 end_ip;

};

Return：

True - succeed;

False - fail.

3.4.26. wifi_softap_dhcps_status

Function：Get ESP8266 softAP dhcp server status.

Prototype：

```
enum dhcp_status wifi_softap_dhcps_status(void)
```

Parameter：

NULL

Return：

```
enum dhcp_status {
    DHCP_STOPPED,
    DHCP_STARTED
};
```

3.4.27. wifi_set_phy_mode

Fuction：Set ESP8266 physical mode（802.11b/g/n）.

Note：ESP8266 softAP only support bg.

Prototype：

```
bool wifi_set_phy_mode(enum phy_mode mode)
```

Parameter：

enum phy_mode mode – physical mode

```
enum phy_mode{
    PHY_MODE_11B = 1,
    PHY_MODE_11G = 2,
    PHY_MODE_11N = 3
};
```

Return：

True - succeed；

False - fail.

3.4.28. wifi_get_phy_mode

Function：Get ESP8266 physical mode（802.11b/g/n）

Prototype：

Enum phy_mode wifi_get_phy_mode(void)

Parameter：

Null

Return：

```
enum phy_mode{
    PHY_MODE_11B = 1,
    PHY_MODE_11G = 2,
    PHY_MODE_11N = 3
};
```

3.4.29. wifi_get_ip_info

Function: Get ip info of wifi station or softap interface

Function definition:

bool wifi_get_ip_info(uint8 if_index, struct ip_info *info)

Parameters:

uint8 if_index——the interface to get ip info: 0x00 for STATION_IF, 0x01 for SOFTAP_IF.

struct ip_info *info——pointer to get ip info of a certain interface

Return：

True - succeed;

False - fail.

3.4.30. wifi_set_ip_info

Function: Set ip address of ESP8266 station or softAP

Note: only can be used in user_init.

Function definition:

bool wifi_set_ip_info(uint8 if_index, struct ip_info *info)

Prototype：

uint8 if_index – set station ip or softAP ip

#define STATION_IF 0x00

#define SOFTAP_IF 0x01

struct ip_info *info – ip information

Example：

```
struct ip_info info;

IP4_ADDR(&info.ip, 192, 168, 3, 200);
IP4_ADDR(&info.gw, 192, 168, 3, 1);
IP4_ADDR(&info.netmask, 255, 255, 255, 0);
wifi_set_ip_info(STATION_IF, &info);

IP4_ADDR(&info.ip, 10, 10, 10, 1);
IP4_ADDR(&info.gw, 10, 10, 10, 1);
IP4_ADDR(&info.netmask, 255, 255, 255, 0);
wifi_set_ip_info(SOFTAP_IF, &info);
```

Return：

True - succeed；

False - fail.

3.4.31. wifi_set_macaddr

Function: set mac address

Note: only can be used in user_init

Function definition:

bool wifi_set_macaddr(uint8 if_index, uint8 *macaddr)

Parameter：

uint8 if_index – set station mac or softAP mac

#define STATION_IF 0x00

#define SOFTAP_IF 0x01

uint8 *macaddr – mac address

Example：

```
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);
```

Return：

True - succeed;

False - fail.

3.4.32. wifi_get_macaddr

Function: get mac address

Function definition:

Bool wifi_get_macaddr(uint8 if_index , uint8 *macaddr)

Parameter：

uint8 if_index —— set station mac or softAP mac

#define STATION_IF 0x00

#define SOFTAP_IF 0x01

uint8 *macaddr—— mac address

Return：

True - succeed；

False - fail.

3.4.33. wifi_set_sleep_type

Function：Set sleep type for power saving. Set NONE_SLEEP_T to disable power saving

Note: Default to be Modem sleep.

Prototype：

Bool wifi_set_sleep_type(enum sleep_type type)

Parameters：

enum sleep_type type —— sleep type

Return：

True ，succeed；False ，fail.

3.4.34. wifi_get_sleep_type

Function: Get sleep type.

Prototype：

Enum sleep_type wifi_get_sleep_type(void)

Parameters：

NULL

Return：

```
Enum sleep_type{
    NONE_SLEEP_T = 0;
    LIGHT_SLEEP_T,
    MODEM_SLEEP_T
};
```

3.4.35. wifi_status_led_install

Function: Install wifi status LED

Function definition:

Void wifi_status_led_install (uint8 gpio_id, uint32 gpio_name, uint8 gpio_func)

Parameter：

uint8 gpio_id——gpio number

uint8 gpio_name——gpio mux name

uint8 gpio_func——gpio function

Return：

NULL

Example：

Use GPIO0 as wifi status LED

```
#define HUMITURE_WIFI_LED_IO_MUX      PERIPHS_IO_MUX_GPIO0_U
#define HUMITURE_WIFI_LED_IO_NUM      0
#define HUMITURE_WIFI_LED_IO_FUNC     FUNC_GPIO0
wifi_status_led_install(HUMITURE_WIFI_LED_IO_NUM,
HUMITURE_WIFI_LED_IO_MUX, HUMITURE_WIFI_LED_IO_FUNC)
```

3.4.36. wifi_status_led_uninstall

Function: Uninstall wifi status LED

Function definition:

Void wifi_status_led_uninstall ()

Parameter：

NULL

Return：

NULL

3.4.37. wifi_set_broadcast_if

Function：Set ESP8266 send UDP broadcast from station interface or softAP interface. Default to be softAP.

Prototype：

Bool wifi_set_broadcast_if (uint8 interface)

Parameter：

uint8 interface —— 1，station
2，softAP
3，both station and softAP

Return：

True ，succeed；False ，fail

3.4.38. wifi_get_broadcast_if

Function：Get interface which ESP8266 sent UDP broadcast from.

Prototype：

Uint8 wifi_get_broadcast_if (void)

Parameter：

NULL

Return：

1，station
2，softAP
3，both station and softAP

3.5. Upgrade(FOTA) APIs

3.5.1. system_upgrade_userbin_check

Function: Check userbin

Function definition:

uint8 system_upgrade_userbin_check()

Input parameters:

null

Return:

0x00 : UPGRADE_FW_BIN1 , i.e., user1.bin

0x01 : UPGRADE_FW_BIN2 , i.e., user2.bin

3.5.2. system_upgrade_flag_set

Function: Set upgrade status flag.

Note:

If you using system_upgrade_start to upgrade, this API need not be called;

If you using spi_flash_write to upgrade firmware yourself, this flag need to be set to UPGRADE_FLAG_FINISH, then call system_upgrade_reboot to reboot to run new firmware.

Prototype:

void system_upgrade_flag_set(uint8 flag)

Parameter：

uint8 flag – #define UPGRADE_FLAG_IDLE 0x00

#define UPGRADE_FLAG_START 0x01

#define UPGRADE_FLAG_FINISH 0x02

Return：

NULL

3.5.3. system_upgrade_flag_check

Function: Get upgrade status flag.

Prototype：

uint8 system_upgrade_flag_check()

Parameter：

NULL

Return：

#define UPGRADE_FLAG_IDLE 0x00

#define UPGRADE_FLAG_START 0x01

#define UPGRADE_FLAG_FINISH 0x02

3.5.4. system_upgrade_start

Function: Configure parameters and start upgrade

Function definition:

bool system_upgrade_start (struct upgrade_server_info *server)

Parameters:

struct upgrade_server_info *server – server related parameters

Return

true: start upgrade

false: upgrade can’t be started.

3.5.5. system_upgrade_reboot

Function: reboot system and use new version

Function definition:

void system_upgrade_reboot (void)

Input parameters:

null

Return:

Null

3.6. Sniffer Related APIs

3.6.1. wifi_promiscuous_enable

Function: Enable promiscuous mode for sniffer

Function definition:

Void wifi_promiscuous_enable(uint8 promiscuous)

Parameter：

uint8 promiscuous —— 0, disable promiscuous

1, enable promiscuous

Return:

null

Example: apply for a demo of sniffer function from Espressif

3.6.2. wifi_promiscuous_set_mac

Function：Set mac address filter for sniffer.

Note: This filter only available in the current sniffer phase，if you disable sniffer and then enable sniffer，you need to set filter again if you need it.

Prototype：

Void wifi_promiscuous_set_mac(const uint8_t *address)

Parameter：

const uint8_t *address —— MAC address

Return：

NULL

3.6.3. wifi_set_promiscuous_rx_cb

Function: register a rx callback function in promiscuous mode, which will be called when data packet is received.

Function definition:

Void wifi_set_promiscuous_rx_cb(wifi_promiscuous_cb_t cb)

Parameter：

wifi_promiscuous_cb_t cb—— callback

Return:

null

3.6.4. wifi_get_channel

Function: get channel number, for sniffer

Function definition:

uint8 wifi_get_channel(void)

parameters:

null

Return:

Channel number

3.6.5. wifi_set_channel

Function: set channel number, for sniffer

Function definition:

bool wifi_set_channel (uint8 channel)

Parameters：

uint8 channel—— channel number

Return：

True - succeed；

False - fail.

3.7. smart config APIs

3.7.1. smartconfig_start

Function： Make ESP8266 station connect to AP

Note：In this API，ESP8266 will be set to station mode. Run phone APP to make device listen to the SSID and password of targeting AP. Can not call smartconfig_start twice before it finish.

Prototype：

bool smartconfig_start(sc_type type, sc_callback_t cb, …)

Parameter：

sc_type type —— smart config protocol type：AirKiss or ESP-TOUCH。

sc_callback_t cb—— smart config callback，go into it when ESP8266 got SSID and password of targeting AP，parameter of this callback refer to the example，a pointer of struct station_config

… -- 1, UART output logs; otherwise, UART only output the result.

Return：

True - succeed；

False - fail.

Example：

```
void ICACHE_FLASH_ATTR
smartconfig_done(void *data)
{
    struct station_config *sta_conf = data;

    wifi_station_set_config(sta_conf);
    wifi_station_disconnect();
    wifi_station_connect();
    user_devicefind_init();
    user_esp_platform_init();
}
smartconfig_start(SC_TYPE_ESPTOUCH,smartconfig_done);
```

3.7.2. smartconfig_stop

Function：stop smart config，free the buffer taken by smartconfig_start.

Note: When connect to AP succeed, this API should be called to free memory taken by smartconfig_start.

Prototype：

bool smartconfig_stop(void)

Parameter：

NULL

Return：

True - succeed；

False - fail.

3.7.3. get_smartconfig_status

Function：get smart config status

Note：Can not call this API after smartconfig_stop，because smartconfig_stop will free memory which contains this smart config status。

Prototype：

sc_status get_smartconfig_status(void)

Parameter：

NULL

Return：

```
typedef enum  {
 SC_STATUS_WAIT = 0,
 SC_STATUS_FIND_CHANNEL,
 SC_STATUS_GETTING_SSID_PSWD,
 SC_STATUS_GOT_SSID_PSWD,
 SC_STATUS_LINK,
 SC_STATUS_LINK_OVER,
```

```
    } sc_status;
```

NOTE:

Use APP to start connection when get_smartconfig_status to be SC_STATUS_FIND_CHANNEL

3.8. Network APIs

Locate in “esp_iot_sdk\include\espconn.h”

General APIs：APIs can be used for both TCP and UDP .

TCP APIs：APIs that are only used for TCP.

UDP APIs：APIs that are only used for UDP.

3.8.1. General APIs

3.8.1.1. espconn_delete

Function: Delete a transmission.

Note: Correspondence create api : TCP espconn_accept, UDP espconn_create

Prototype：

Sin8 espconn_delete(struct espconn *espconn)

Parameter：

struct espconn *espconn —— corresponding connected control block structure

Return：

0 - succeed，#define ESPCONN_OK 0

Not 0 - Erro，pls refer to espconn.h

3.8.1.2. espconn_gethostbyname

Function: DNS

Function definition:

Err_t espconn_gethostbyname(struct espconn *pespconn, const char *hostname, ip_addr_t *addr, dns_found_callback found)

Parameters:

struct espconn *espconn——corresponding connected control block

structure

const char *hostname——domain name string pointer

ip_addr_t *addr——ip address

dns_found_callback found——callback

Return:

Err_t——ESPCONN_OK

ESPCONN_INPROGRESS

ESPCONN_ARG

Example as follows. Pls refer to source code of IoT_Demo：

```
ip_addr_t esp_server_ip;
LOCAL void ICACHE_FLASH_ATTR
user_esp_platform_dns_found(const char *name, ip_addr_t *ipaddr, void *arg)
{
    struct espconn *pespconn = (struct espconn *)arg;

    os_printf("user_esp_platform_dns_found %d.%d.%d.%d\n",
              *((uint8 *)&ipaddr->addr), *((uint8 *)&ipaddr->addr + 1),
              *((uint8 *)&ipaddr->addr + 2), *((uint8 *)&ipaddr->addr + 3));
}
Void dns_test(void)
{
espconn_gethostbyname(pespconn,”iot.espressif.cn”,&esp_server_ip,user_esp_platform_dns_found);
}
```

3.8.1.3. espconn_port

Function: get void ports

Function definition:

uint32 espconn_port(void);

Input parameters:

null

Return:

uint32——id of the port you get

3.8.1.4. espconn_regist_sentcb

Function: register data sent function which will be called back when data are successfully sent.

Function definition:

Sint8 espconn_regist_sentcb(struct espconn *espconn, espconn_sent_callback sent_cb)

Parameters：

struct espconn *espconn——corresponding connected control block structure

espconn_sent_callback sent_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.1.5. espconn_regist_recvcb

Function: register data receive function which will be called back when data are received

Function definition:

Sint8 espconn_regist_recvcb(struct espconn *espconn, espconn_recv_callback recv_cb)

Input parameters:

struct espconn *espconn——corresponding connected control block

structure

espconn_connect_callback connect_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.1.6. espconn_sent_callback

Function: callback after the data are sent

Function definition:

void espconn_sent_callback (void *arg)

Input parameters:

void *arg——call back function parameters

Return:

null

3.8.1.7. espconn_recv_callback

Function: callback after data are received

Function definition:

void espconn_recv_callback (void *arg, char *pdata, unsigned short len)

Input parameters:

void *arg——callback function parameters

char *pdata——received data entry parameters

unsigned short len——received data length

Return:

null

3.8.1.8. espconn_sent

Function: send data through wifi

Note: Please call espconn_sent after espconn_sent_callback of the pre-packet.

Function definition:

sint8 espconn_sent(struct espconn *espconn, uint8 *psent, uint16 length)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

uint8 *psent——sent data pointer

uint16 length——sent data length

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2. TCP APIs

3.8.2.1. espconn_accept

Function: listening connection. This function is used when create a TCP server.

Function definition:

sint8 espconn_accept(struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.2. espconn_secure_accept

Function: encrypted listening connection. This function is used when create a TCP server which support SSL.

Function definition:

sint8 espconn_secure_accept(struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.3. espconn_regist_time

Function: register timeout interval when ESP8266 is TCP server, ESP8266 will never disconnect to TCP client which didn't communication with it .

NOTE:

If you need to set timeout please call this API after espconn_accept.

Set timeout to be 0, ESP8266 TCP server will never disconnect to TCP client which didn't communication with ESP8266. We don't recommend this.

Function definition:

sint8 espconn_regist_time(struct espconn *espconn, uint32 interval, uint8 type_flag)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

uint32 interval ——timeout interval, unit: second, maximum: 7200 seconds

uint8 type_flag ——0, set all connections; 1, set a single connection

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.4. espconn_get_connection_info

Function: get a connection's info in TCP multi-connection case

Function definition:

sint8 espconn_get_connection_info(struct espconn *espconn, remot_info **pcon_info, uint8 typeflags)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

remot_info **pcon_info——connect to client info

uint8 typeflags —— 0, regular server;1, ssl server

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.5. espconn_connect

Function: connect to a TCP server, and ESP8266 is the TCP client.

Function definition:

sint8 espconn_connect(struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0	-	error, pls refer to espconn.h

3.8.2.6. espconn_connect_callback

Function: successful listening(ESP8266 as TCP server) or connection(ESP8266 as TCP client) callback

Function definition:

void espconn_connect_callback (void *arg)

Input parameters:

void *arg——callback function parameters

Return:

null

3.8.2.7. espconn_set_opt

Function: Set option of TCP connection

Prototype：

sint8 espconn_set_opt(struct espconn *espconn, uint8 opt)

Parameter：

struct espconn *espconn——corresponding connected control structure

uint8 opt – Option of TCP connection

bit1, 1, free memory after TCP disconnection happen need not wait 2 minutes；

bit2, 1，disable nalgo algorithm during TCP data transmission, quiken the data transmission.

Bit3, 1, use 2920 bytes write buffer for the data espconn_sent sending.

Return：

0	-	succeed，#define ESPCONN_OK 0
Not 0	-	error, pls refer to espconn.h

Note：

In general, we need not call this API；

If call espconn_set_opt(espconn, 0)，please call it in connected callback；

If call espconn_set_opt(espconn, 1)，please call it before disconnect

3.8.2.8. espconn_disconnect

Function: disconnect a TCP connection

Function definition:

sint8 espconn_disconnect(struct espconn *espconn);

Input parameters:

struct espconn *espconn——corresponding connected control structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.9. espconn_regist_connectcb

Function: register connection function which will be called back under successful TCP connection

Function definition:

Sint8 espconn_regist_connectcb(struct espconn *espconn, espconn_connect_callback connect_cb)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

espconn_connect_callback connect_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.10. espconn_regist_reconcb

Function: register reconnect callback

Note: Reconnect callback is more like a network error handler, no matter error occurred in any phase, it will go into reconnect callback. For example, if espconn_sent fail, it will go into reconnect callback as network is broken.

Function definition:

sint8 espconn_regist_reconcb(struct espconn *espconn, espconn_connect_callback recon_cb)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

espconn_connect_callback connect_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.11. espconn_regist_disconcb

Function: register disconnection function which will be called back under successful TCP disconnection

Function definition:

Sint8 espconn_regist_disconcb(struct espconn *espconn, espconn_connect_callback discon_cb)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

espconn_connect_callback connect_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.12. espconn_regist_write_finish

Function: register a callback which will be called when all sending data is completely write into write buffer or sent.

Function definition:

Sint8 espconn_regist_write_finish (struct espconn *espconn, espconn_connect_callback write_finish_fn)

Input parameters:

struct espconn *espconn —— corresponding connected control block structure

espconn_connect_callback write_finish_fn —— registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.13. espconn_secure_connect

Function: Secure connect(SSL) to a TCP server, and ESP8266 is the TCP client.

Function definition:

Sint8 espconn_secure_connect (struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block

structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.14. espconn_secure_sent

Function: send encrypted data（SSL）

Function definition:

Sint8 espconn_secure_sent (struct espconn *espconn, uint8 *psent, uint16 length)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

uint8 *psent——sent data pointer

uint16 length——sent data length

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.15. espconn_secure_disconnect

Function: secure TCP disconnection(SSL)

Function definition:

Sint8 espconn_secure_disconnect(struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.16. espconn_tcp_get_max_con

Function：Get maximum number of how many TCP connection is allowed.

Prototype：

uint8 espconn_tcp_get_max_con(void)

Parameter：

NULL

Return：

Maximum number of how many TCP connection is allowed.

3.8.2.17. espconn_tcp_set_max_con

Function：Set the maximum number of how many TCP connection is allowed.

Prototype：

Sint8 espconn_tcp_set_max_con(uint8 num)

Parameter：

uint8 num—— Maximum number of how many TCP connection is allowed.

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.18. espconn_tcp_get_max_con_allow

Function: Get the maximum number of TCP clients which are allowed to connect to ESP8266 TCP server.

Prototype：

Sint8 espconn_tcp_get_max_con_allow(struct espconn *espconn)

Parameter:

struct espconn *espconn——corresponding connected control structure

Return:

Maximum number of TCP clients which are allowed.

3.8.2.19. espconn_tcp_set_max_con_allow

Function: Set the maximum number of TCP clients which are allowed to connect to ESP8266 TCP server.

Prototype：

Sint8 espconn_tcp_set_max_con_allow(struct espconn *espconn, uint8 num)

Parameter:

struct espconn *espconn——corresponding connected control structure

uint8 num -- Maximum number of TCP clients which are allowed.

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

3.8.2.20. espconn_recv_hold

Function：Block TCP receiving data

Note：This API block TCP receiving data eventually，not immediately，so we recommended to call it while reserving 1460*5 Bytes memory. This API can be called more than once.

Prototype：

Sint8 espconn_recv_hold(struct espconn *espconn)

Parameter:

struct espconn *espconn——corresponding connected control structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h，ESPCONN_ARG means could not find the TCP connection according to parameter “espconn”

3.8.2.21. espconn_recv_unhold

Function：Stop blocking TCP receiving data。

Note：This API take effect immediately

Prototype：

Sint8 espconn_recv_unhold(struct espconn *espconn)

Parameter:

struct espconn *espconn——corresponding connected control structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h，ESPCONN_ARG means could not find the TCP connection according to parameter “espconn”

3.8.3. UDP APIs

3.8.3.1. espconn_create

Function：create UDP transmission.

Prototype：

Sin8 espconn_create(struct espconn *espconn)

Parameter：

struct espconn *espconn——corresponding connected control block structure

Return：

0 - succeed，#define ESPCONN_OK 0

Not 0 - Erro，pls refer to espconn.h

3.8.3.2. espconn_igmp_join

Function：Join a multicast group

Prototype：

Sin8 espconn_igmp_join(ip_addr_t *host_ip, ip_addr_t *multicast_ip)

Parameters：

ip_addr_t *host_ip —— ip of host

ip_addr_t *multicast_ip – ip of multicast group

Return：

0 - succeed，#define ESPCONN_OK 0

Not 0 - Erro，pls refer to espconn.h

3.8.3.3. espconn_igmp_leave

Function：Quit a multicast group

Prototype：

Sin8 espconn_igmp_leave(ip_addr_t *host_ip, ip_addr_t *multicast_ip)

Parameters：

ip_addr_t *host_ip —— ip of host

ip_addr_t *multicast_ip –ip of multicast group

Return：

0 - succeed，#define ESPCONN_OK 0

Not 0 - Erro，pls refer to espconn.h

3.9. AT APIs

AT APIs example refer to esp_iot_sdk/examples/5.at/user/user_main.c

3.9.1. at_response_ok

Function：output “OK” to AT Port (UART0)

Prototype：

void at_response_ok(void)

Parameter：

NULL

Return：

NULL

3.9.2. at_response_error

Function：output “ERROR” to AT Port (UART0)

Prototype：

void at_response_error(void)

Parameter：

NULL

Return：

NULL

3.9.3. at_cmd_array_regist

Function：register user-define AT commands

Prototype：

void at_cmd_array_regist (at_funcation * custom_at_cmd_arrar, uint32 cmd_num)

Parameter：

at_funcation * custom_at_cmd_arrar – Array of user-define AT commands

uint32 cmd_num – Number counts of user-define AT commands

Return：

NULL

Example：refer to esp_iot_sdk/examples/5.at/user/user_main.c

3.9.4. at_get_next_int_dec

Function：parse int from AT command

Prototype：

bool at_get_next_int_dec (char **p_src,int* result,int* err)

Parameter：

char **p_src - *p_src is the AT command that need to be parsed

int* result – int number parsed from the AT command

int* err - error code

1：int number is omit，return error code 1

3：only ’ -’ be found, return error code 3

Return：

TRUE，parser succeed (if int number default omit，it will return True，but error code to be 1)

FALSE，parser error with error code, probable cause：int number more than 10 bytes、contains termination characters‘\r‘ 、only contains ’ -’

Example: refer to esp_iot_sdk/examples/5.at/user/user_main.c

3.9.5. at_data_str_copy

Function：parse string from AT command

Prototype：

Int32 at_data_str_copy (char * p_dest, char ** p_src,int32 max_len)

Parameter：

char * p_dest - string parsed from the AT command

char ** p_src - *p_src is the AT command that need to be parsed

int32 max_len – max string length that allowed

Return：

If succeed，returns the length of the string；

If fail，returns -1

Example: refer to esp_iot_sdk/examples/5.at/user/user_main.c

3.9.6. at_init

Function: AT initialize

Prototype：

void at_init (void)

Parameter：

NULL

Return：

NULL

Example: refer to esp_iot_sdk/examples/5.at/user/user_main.c

3.9.7. at_port_print

Funtion: output string to AT PORT(UART0)

Prototype：

void at_port_print(const char *str)

Parameter：

const char *str – string that need to output

Return：

NULL

Example: refer to esp_iot_sdk/examples/5.at/user/user_main.c

3.9.8. at_set_custom_info

Function：User-define version info of AT which can be got by AT+GMR.

Prototype：

void at_set_custom_info (char *info)

Parameter：

char *info – version info

Return：

NULL

3.9.9. at_enter_special_state

Function：Enter processing state. In processing state, AT core will return “busy” for any further AT commands.

Prototype：

void at_enter_special_state (void)

Parameter：

NULL

Return：

NULL

3.9.10. at_leave_special_state

Function：Exit from AT processing state.

Prototype：

void at_leave_special_state (void)

Parameter：

NULL

Return：

NULL

3.9.11. at_get_version

Function：Get Espressif AT lib version。

Prototype：

uint32 at_get_version (void)

Parameter：

NULL

Return：

Espressif AT lib version

3.10. json APIs

Locate in : esp_iot_sdk\include\json\jsonparse.h & jsontree.h

3.10.1. jsonparse_setup

Function:json initialize parsing

Function definition:

void jsonparse_setup(struct jsonparse_state *state, const char *json, int len)

Input parameters:

struct jsonparse_state *state——json parsing pointer

const char *json——json parsing character string

int len——character string length

Return:

null

3.10.2. jsonparse_next

Function: jsonparse next object

Function definition:

int jsonparse_next(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

int——parsing result

3.10.3. jsonparse_copy_value

Function: copy current parsing character string to a certain buffer

Function definition:

int jsonparse_copy_value(struct jsonparse_state *state, char *str, int

size)

Input parameters:

struct jsonparse_state *state——json parsing pointer

char *str——buffer pointer

int size——buffer size

Return:

int——copy result

3.10.4. jsonparse_get_value_as_int

Function: parse json to get integer

Function definition:

int jsonparse_get_value_as_int(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

int——parsing result

3.10.5. jsonparse_get_value_as_long

Function: parse json to get long integer

Function definition:

long jsonparse_get_value_as_long(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

long——parsing result

3.10.6. jsonparse_get_len

Function: get parsed json length

Function definition:

int jsonparse_get_value_len(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

int——parsed jason length

3.10.7. jsonparse_get_value_as_type

Function: parsed json data type

Function definition:

int jsonparse_get_value_as_type(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

int——parsed json data type

3.10.8. jsonparse_strcmp_value

Function: compare parsed json and certain character string

Function definition:

int jsonparse_strcmp_value(struct jsonparse_state *state, const char *str)

Input parameters:

struct jsonparse_state *state——json parsing pointer

const char *str——character buffer

Return:

int——comparison result

3.10.9. jsontree_set_up

Function: create json data tree

Function definition:

void jsontree_setup(struct jsontree_context *js_ctx,

struct jsontree_value *root, int (* putchar)(int))

Input parameters:

struct jsontree_context *js_ctx——json tree element pointer

struct jsontree_value *root——root element pointer

int (* putchar)(int)——input function

Return:

null

3.10.10. jsontree_reset

Function: reset json tree

Function definition:

void jsontree_reset(struct jsontree_context *js_ctx)

Input parameters:

struct jsontree_context *js_ctx——json data tree pointer

Return:

null

3.10.11. jsontree_path_name

Function: get json tree parameters

Function definition:

const char *jsontree_path_name(const struct jsontree_cotext *js_ctx, int depth)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

int depth——json tree depth

Return:

char*——parameter pointer

3.10.12. jsontree_write_int

Function: write integer to joson tree

Function definition:

void jsontree_write_int(const struct jsontree_context *js_ctx, int value)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

int value——integer value

Return:

null

3.10.13. jsontree_write_int_array

Function: write integer array to json tree

Function definition:

void jsontree_write_int_array(const struct jsontree_context *js_ctx, const int *text, uint32 length)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

int *text——array entry address

uint32 length——array length

Return:

null

3.10.14. jsontree_write_string

Function: write string to json tree

Function definition:

void jsontree_write_string(const struct jsontree_context *js_ctx, const char *text)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

const char* text——character string pointer

Return:

null

3.10.15. jsontree_print_next

Function: json tree depth

Function definition:

int jsontree_print_next(struct jsontree_context *js_ctx)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

Return:

int——json tree depth

3.10.16. jsontree_find_next

Function: find json tree element

Function definition:

struct jsontree_value *jsontree_find_next(struct jsontree_context *js_ctx, int type)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

int——type

Return:

struct jsontree_value *——json tree element pointer

4. Structure definition

4.1. Timer

```
typedef void ETSTimerFunc(void *timer_arg);

typedef struct _ETSTIMER_ {
struct _ETSTIMER_       *timer_next;
      uint32_t                    timer_expire;
      uint32_t                    timer_period;
     ETSTimerFunc              *timer_func;
     void                        *timer_arg;
} ETSTimer;
```

4.2. Wifi related structure

4.2.1. station related

```
struct station_config {
uint8 ssid[32];
uint8 password[64];
uint8 bssid_set;
uint8 bssid[6];
};
```

Note:

Bssid as mac address of AP, will be used when serveral APs have the same ssid.

If station_config.bssid_set == 1 , station_config.bssid has to be set, or connection will fail.

So in general, station_config.bssid_set need to be 0.

4.2.2. softap related

```
typedef enum _auth_mode {
    AUTH_OPEN                = 0,
    AUTH_WEP,
    AUTH_WPA_PSK,
    AUTH_WPA2_PSK,
    AUTH_WPA_WPA2_PSK
} AUTH_MODE;
struct softap_config {
    uint8 ssid[32];
    uint8 password[64];
    uint8 ssid_len;
    uint8 channel;
    uint8 authmode;
    uint8 ssid_hidden;
    uint8 max_connection;
    uint8 beacon_interval;    // 100 ~ 60000 ms, default 100
};
```

Note:

If softap_config.ssid_len == 0, check ssid till find a termination characters;

otherwise it depends on softap_config.ssid_len.

4.2.3. scan related

```
struct scan_config {
    uint8 *ssid;
    uint8 *bssid;
    uint8 channel;
```

```
    uint8 show_hidden; // Scan APs which are hiding their ssid or not.
};

struct bss_info {
    STAILQ_ENTRY(bss_info)      next;
    u8 bssid[6];
    u8 ssid[32];
    u8 channel;
    s8 rssi;
    u8 authmode;
    uint8 is_hidden; // SSID of current AP is hidden or not.
};
typedef void (* scan_done_cb_t)(void *arg, STATUS status);
```

4.3. smart config structure

```
typedef enum    {
    SC_STATUS_FIND_CHANNEL = 0,
    SC_STATUS_GETTING_SSID_PSWD,
    SC_STATUS_GOT_SSID_PSWD,
    SC_STATUS_LINK,
} sc_status;

typedef enum    {
    SC_TYPE_ESPTOUCH = 0,
    SC_TYPE_AIRKISS,
} sc_type;
```

4.4. json related structure

4.3.1. json structure

```
struct jsontree_value {
uint8_t type;
};

struct jsontree_pair {
const char *name;
struct jsontree_value *value;
};

struct jsontree_context {
struct jsontree_value *values[JSONTREE_MAX_DEPTH];
uint16_t index[JSONTREE_MAX_DEPTH];
int (* putchar)(int);
uint8_t depth;
uint8_t path;
int callback_state;
};

struct jsontree_callback {
uint8_t type;
int (* output)(struct jsontree_context *js_ctx);
int (* set)(struct jsontree_context *js_ctx, struct jsonparse_state *parser);
};

struct jsontree_object {
```

```
uint8_t type;
uint8_t count;
struct jsontree_pair *pairs;
};

struct jsontree_array {
uint8_t type;
uint8_t count;
struct jsontree_value **values;
};

struct jsonparse_state {
const char *json;
int pos;
int len;
int depth;
int vstart;
int vlen;
char vtype;
char error;
char stack[JSONPARSE_MAX_DEPTH];
};
```


4.3.2. json macro definition

```
#define JSONTREE_OBJECT(name, ...)
\
static struct jsontree_pair jsontree_pair_##name[] = {__VA_ARGS__};  \
static struct jsontree_object name = {  \
```

```
    JSON_TYPE_OBJECT,                                   \
sizeof(jsontree_pair_##name)/sizeof(struct jsontree_pair),          \
    jsontree_pair_##name }

#define JSONTREE_PAIR_ARRAY(value) (struct jsontree_value *)(value)
#define JSONTREE_ARRAY(name, ...)
\
static struct jsontree_value* jsontree_value_##name[] = {__VA_ARGS__};   \
static struct jsontree_array name = {                                    \
    JSON_TYPE_ARRAY,                                    \
sizeof(jsontree_value_##name)/sizeof(struct jsontree_value*),       \
    jsontree_value_##name }
```

4.5. espconn parameters

4.4.1 callback function

```
/** callback prototype to inform about events for a espconn */
typedef void (* espconn_recv_callback)(void *arg, char *pdata, unsigned short
len);
typedef void (* espconn_callback)(void *arg, char *pdata, unsigned short len);
typedef void (* espconn_connect_callback)(void *arg);
```

4.4.2 espconn

```
    typedef void* espconn_handle;
    typedef struct _esp_tcp {
        int client_port;
        int server_port;
        char ipaddr[4];
        espconn_connect_callback connect_callback;
```

```
        espconn_connect_callback reconnect_callback;
        espconn_connect_callback disconnect_callback;
    } esp_tcp;

    typedef struct _esp_udp {
        int _port;
        char ipaddr[4];
    } esp_udp;

    /** Protocol family and type of the espconn */
    enum espconn_type {
        ESPCONN_INVALID      = 0,
        /* ESPCONN_TCP Group */
        ESPCONN_TCP          = 0x10,
        /* ESPCONN_UDP Group */
        ESPCONN_UDP          = 0x20,
    };

    /** Current state of the espconn. Non-TCP espconn are always in state ESPCONN_NONE! */
    enum espconn_state {
        ESPCONN_NONE,
        ESPCONN_WAIT,
        ESPCONN_LISTEN,
        ESPCONN_CONNECT,
        ESPCONN_WRITE,
        ESPCONN_READ,
        ESPCONN_CLOSE
    };
```

```
/** A espconn descriptor */
struct espconn {
    /** type of the espconn (TCP, UDP) */
    enum espconn_type type;
    /** current state of the espconn */
    enum espconn_state state;
    union {
        esp_tcp *tcp;
        esp_udp *udp;
    } proto;
    /** A callback function that is informed about events for this espconn */
    espconn_recv_callback recv_callback;
    espconn_sent_callback sent_callback;
    espconn_handle esp_pcb;
    uint8 *ptrbuf;
    uint16 cntr;
};
```

5. Driver

5.1. GPIO APIs

Please refer to \user\ user_plug.c。

5.1.1. PIN setting macro

- ✓ PIN_PULLUP_DIS(PIN_NAME)

  Disable pin pull up
- ✓ PIN_PULLUP_EN(PIN_NAME)

  Enable pin pull up
- ✓ PIN_PULLDWN_DIS(PIN_NAME)

  Disable pin pull down
- ✓ PIN_PULLDWN_EN(PIN_NAME)

  Enable pin pull down
- ✓ PIN_FUNC_SELECT(PIN_NAME, FUNC)

  Select pin function

  Example ： PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U, FUNC_GPIO12);

  Use MTDI pin as GPIO12。

5.1.2. gpio_output_set

Function: set gpio property

Function definition:

void gpio_output_set(uint32 set_mask, uint32 clear_mask, uint32 enable_mask, uint32 disable_mask)

Input parameters:

uint32 set_mask——set high output: 1 means high output; 0 means no status change

uint32 clear_mask——set low output: 1 means low output; 0 means no status change

uint32 enable_mask——enable outpout bit

uint32 disable_mask——enable input bit

Return:

Null

Example：

- ✓ Set GPIO12 as high-level output：gpio_output_set(BIT12, 0, BIT12, 0);
- ✓ Set GPIO12 as low-level output：gpio_output_set(0, BIT12, BIT12, 0);
- ✓ Set GPIO12 as high-level output，GPIO13 as low-level output，则：
  gpio_output_set(BIT12, BIT13, BIT12|BIT13, 0);
- ✓ Set GPIO12 as input : gpio_output_set(0, 0, 0, BIT12);

5.1.3. GPIO input and output macro

- ✓ GPIO_OUTPUT_SET(gpio_no, bit_value)
  Set gpio_no as output bit_value，the same as the output example in 5.1.2
- ✓ GPIO_DIS_OUTPUT(gpio_no)
  Set gpio_no as input, the same as the input example in 5.1.2.
- ✓ GPIO_INPUT_GET(gpio_no)
  Get the level status of gpio_no.

5.1.4. GPIO interrupt

- ✓ ETS_GPIO_INTR_ATTACH(func, arg)
  Register GPIO interrupt control function
- ✓ ETS_GPIO_INTR_DISABLE()
  Disable GPIO interrupt
- ✓ ETS_GPIO_INTR_ENABLE()
  Enable GPIO interrupt

5.1.5. gpio_pin_intr_state_set

Function: set gpio interrupt state

Function definition:

```
void gpio_pin_intr_state_set(uint32 i,  GPIO_INT_TYPE intr_state)
```

Input parameters:

uint32 i——GPIO pin ID, if you want to set GPIO14, pls use GPIO_ID_PIN(14);

GPIO_INT_TYPE intr_state——interrupt type

as：

```
typedef enum{
    GPIO_PIN_INTR_DISABLE = 0,
    GPIO_PIN_INTR_POSEDGE= 1,
    GPIO_PIN_INTR_NEGEDGE= 2,
    GPIO_PIN_INTR_ANYEGDE=3,
    GPIO_PIN_INTR_LOLEVEL= 4,
    GPIO_PIN_INTR_HILEVEL = 5
}GPIO_INT_TYPE;
```

Return：

NULL

5.1.6. GPIO interrupt handler

Follow below steps to clear interrupt status in GPIO interrupt processing function：

```
uint32 gpio_status;
gpio_status = GPIO_REG_READ(GPIO_STATUS_ADDRESS);
//clear interrupt status
GPIO_REG_WRITE(GPIO_STATUS_W1TC_ADDRESS, gpio_status);
```

5.2. UART APIs

By default, UART0 is debug output interface. In the case of dual Uart,

UART0 works as data receive and transmit interface, and UART1as debug output interface.

Please make sure all hardware are correctly connected.

5.2.1. uart_init

Function: initialize baud rates of the two uarts

Function definition:

void uart_init(UartBautRate uart0_br, UartBautRate uart1_br)

Parameters：

UartBautRate uart0_br——uart0 baud rate

UartBautRate uart1_br——uart1 baud rate

As：

```
typedef enum {
    BIT_RATE_9600     = 9600,
    BIT_RATE_19200    = 19200,
    BIT_RATE_38400    = 38400,
    BIT_RATE_57600    = 57600,
    BIT_RATE_74880    = 74880,
    BIT_RATE_115200 = 115200,
    BIT_RATE_230400 = 230400,
    BIT_RATE_460800 = 460800,
    BIT_RATE_921600 = 921600
} UartBautRate;
```

Return:

NULL

5.2.2. uart0_tx_buffer

Function: send user-defined data through UART0

Function definition:

Void uart0_tx_buffer(uint8 *buf, uint16 len)

Parameter:

Uint8 *buf——data to send later

Uint16 len——the length of data to send later

Return:

NULL

5.2.3. uart0_rx_intr_handler

Function: UART0 interrupt processing function. Users can add the processing of received data in this function. (Receive buffer size: 0x100; if the received data are more than 0x100, pls handle them yourselves.）

Function definition:

Void uart0_rx_intr_handler(void *para)

Parameter:

Void*para——the pointer pointing to RcvMsgBuff structure

Return:

NULL

5.3. i2c master APIs

5.3.1. i2c_master_gpio_init

Function: set GPIO in i2c master mode

Function definition:

void i2c_master_gpio_init (void)

Input parameters:

null

Return:

null

5.3.2. i2c_master_init

Function: initialize i2c

Function definition:

void i2c_master_init(void)

Input parameters:

null

Return:

null

5.3.3. i2c_master_start

Function: set i2c to start data delivery

Function definition:

void i2c_master_start(void)

Input parameters:

null

Return:

null

5.3.4. i2c_master_stop

Function: set i2c to stop data delivery

Function definition:

Void i2c_master_stop(void)

Input parameters:

null

Return:

null

5.3.5. i2c_master_send_ack

Function: send i2c ACK

Function definition:

void i2c_master_send_ack (void)

Input parameters:

NULL

Return:

NULL

5.3.6. i2c_master_send_nack

Function: send i2c NACK

Function definition:

void i2c_master_send_nack (void)

Input parameters:

NULL

Return:

NULL

5.3.7. i2c_master_checkAck

Function: check ACK from slave

Function definition:

bool i2c_master_checkAck (void)

Input parameters:

NULL

Return:

TRUE, get i2c slave ACK

FALSE, get i2c slave NACK

5.3.8. i2c_master_readByte

Function: read a byte from slave

Function definition:

uint8 i2c_master_readByte (void)

Input parameters:

null

Return:

uint8——the value you read

5.3.9. i2c_master_writeByte

Function: write a byte to slave

Function definition:

void i2c_master_writeByte (uint8 wrdata)

Input parameters:

uint8 wrdata——data to write

Return:

null

5.4. pwm

4 PWM outputs are supported, more details in pwm.h.

5.4.1. pwm_init

Function: initialize pwm function, including gpio, frequency, and duty cycle

Function definition:

void pwm_init(uint16 freq, uint8 *duty)

Input parameters:

uint16 freq——pwm's frequency;

uint8 *duty——duty cycle of each output

Return:

null

5.4.2. pwm_start

Function: start PWM. This function need to be called after every pwm config changing.

Prototype:

Void pwm_start (void)

Parameter：

null

Return：

null

5.4.3. pwm_set_duty

Function: set duty cycle of an output

Function definition:

void pwm_set_duty(uint8 duty, uint8 channel)

Input parameters:

uint8 duty——duty cycle

uint8 channel——an output

Return:

null

5.4.4. pwm_set_freq

Function: set pwm frequency

Function definition:

void pwm_set_freq(uint16 freq)

Input parameters:

uint16 freq——pwm frequency

Return:

null

5.4.5. pwm_get_duty

Function: get duty cycle of an output

Function definition:

uint8 pwm_get_duty(uint8 channel)

Input parameters:

uint8 channel——channel of which to get duty cycle

Return:

uint8——duty cycle

5.4.6. pwm_get_freq

Function: get pwm frequency

Function definition:

uint16 pwm_get_freq(void)

Input parameters:

null

Return:

uint16——frequency

6. Appendix

A. ESPCONN Programming

Programming guide for ESP8266 running as TCP client and TCP server.

A.1. TCP Client Mode

A.1.1. Instructions

ESP8266, working in Station mode, will start client connection when given an IP address.

ESP8266, working in softap mode, will start client connection when the devices which are connected to ESP8266 are given an IP address.

A.1.2. Steps

1) Initialize espconn parameters according to protocols.
2) Register connect callback function, and register reconnect callback function.
   (Call espconn_regist_connectcb and espconn_regist_reconcb)
3) Call espconn_connect function and set up the connection with TCP Server.
4) Registered connected callback function will be called after successful connection, which will register the corresponding callback function.
   Recommend to register disconnect callback function.
   (Call espconn_regist_recvcb , espconn_regist_sentcb and espconn_regist_disconcb in connected callback)
5) When using receive callback function or sent callback function to run disconnect, it is recommended to set a time delay to make sure that the all the firmware functions are completed.

A.2. TCP Server Mode

A.2.1. Instructions

ESP8266, working in Station mode, will start server listening when given an IP address.

ESP8266, working in softAP mode, will start server listening.

A.2.2. Steps

(1) Initialize espconn parameters according to protocols.

(2) Register connect callback and reconnect callback function.
(Call espconn_regist_connectcb and espconn_regist_reconcb)

(3) Call espconn_accept function to listen to the connection with host.

(4) Registered connect function will be called after successful connection, which will register corresponding callback function.
(Call espconn_regist_recvcb , espconn_regist_sentcb and espconn_regist_disconcb in connected callback)

B. RTC APIs Example

Demo code below shows how to get RTC time and read/write RTC memory.

```
void user_init(void)
{

os_printf("clk cal : %d \n\r",system_rtc_clock_cali_proc()>>12);
uint32 rtc_time = 0, rtc_reg_val = 0,stime = 0,rtc_time2 = 0,stime2 = 0;
rtc_time =    system_get_rtc_time();
stime = system_get_time();

os_printf("rtc time : %d \n\r",rtc_time);
os_printf("system time : %d \n\r",stime);

if( system_rtc_mem_read(0, &rtc_reg_val, 4) ){
      os_printf("rtc mem val : 0x%08x\n\r",rtc_reg_val);
}else{
      os_printf("rtc mem val error\n\r");
}
      rtc_reg_val++;
      os_printf("rtc mem val write\n\r");
      system_rtc_mem_write(0, &rtc_reg_val, 4) ;

      if( system_rtc_mem_read(0, &rtc_reg_val, 4) ){
            os_printf("rtc mem val : 0x%08x\n\r",rtc_reg_val);
      }else{
            os_printf("rtc mem val error\n\r");
      }
```

```
rtc_time2 =    system_get_rtc_time();
stime2 = system_get_time();

os_printf("rtc time : %d \n\r",rtc_time2);
os_printf("system time : %d \n\r",stime2);

os_printf("delta time rtc: %d \n\r",rtc_time2-rtc_time);
os_printf("delta system    time rtc: %d \n\r",stime2-stime);

os_printf("clk cal : %d \n\r",system_rtc_clock_cali_proc()>>12);

os_delay_us(500000);
system_restart();

}
```

C. Sniffer Structure Introduction

ESP8266 can enter promiscuous mode (sniffer) and capture IEEE 802.11 packets in the air. ESP8266 can capture HT20 packet of 11b、11g、11n（support MCS0-7，not support MCS7-76，not support HT40，not support LDPC，but support AMPDU）. Although ESP8266 can not capture some kinds of ieee80211 packets completely, but it can get the length of these special packets.

So there are two kinds of packets:

(1) The packets ESP8266 can capture completely, they contain a certain length of the head of MAC packet (MAC address of the both side of communication and encryption type) and the length of entire packet.

(2) The packets ESP8266 can not capture completely, they will be only got the length of packet.

Structure RxControl and sniffer_buf are used to represent these two kinds of packets. Structure sniffer_buf contains structure RxControl.

```
struct RxControl {
    signed rssi:8;      // signal intensity of packet
    unsigned rate:4;
    unsigned is_group:1;
    unsigned:1;
    unsigned sig_mode:2;    //0: it's a 11n packet; 1: it's not a 11n packet;
    unsigned legacy_length:12; //if it's not a 11n packet, this shows length of packet.
    unsigned damatch0:1;
    unsigned damatch1:1;
    unsigned bssidmatch0:1;
    unsigned bssidmatch1:1;
    unsigned MCS:7; // if it's a 11n packet, this shows the modulation coding sequence which range 0-76
    unsigned CWB:1; // if it's a 11n packet, this shows it's a HT40 packet or not
    unsigned HT_length:16;// if it's a 11n packet, this shows length of packet.
    unsigned Smoothing:1;
    unsigned Not_Sounding:1;
```

```
    unsigned:1;
    unsigned Aggregation:1;
    unsigned STBC:2;
    unsigned FEC_CODING:1; // if it's a 11n packet, this shows it's a LDPC packet or not.
    unsigned SGI:1;
    unsigned rxend_state:8;
    unsigned ampdu_cnt:8;
    unsigned channel:4; //which channel this packet in.
    unsigned:12;
};

struct LenSeq{
    u16 len; // length of packet
    u16 seq; //serial number of packet，the high 12bits are serial number，low 14bits are Fragment number （usually be 0）
    u8 addr3[6];//the third address in packet
};

struct sniffer_buf{
    struct RxControl rx_ctrl;
    u8 buf[36 ]; // head of ieee80211 packet
    u16 cnt; // number count of packet
    u16 len[1]; //length of packet
};
```

Callback wifi_promiscuous_rx has two parameters (buf and len). "len" means the length of "buf", "len" = 12 or "len" ≥ 60

(1) len ≥ 60, "buf" contains structure "sniffer_buf", this structure is reliable, data packets represented by it has been verified by CRC.

"sniffer_buf.cnt" means the count of packets in "buf". The value of "len"

depends on “sniffer_buf.cnt”.

sniffer_buf.cnt == 0, invalid “buf”; otherwise, len = 50 + cnt * 10;

“sniffer_buf.buf” contains the first 36 bytes of ieee80211 packet. Start from sniffer_buf.len[0], every 2 bytes represent a length information of next packet, len[0] represents the length of first packet, if there is two packet (sniffer_buf.cnt == 2), len[1] represents the length of second packet.

If sniffer_buf.cnt > 1, means it’s a AMPDU packet, head of each MPDU packets are similar, so we only provide the length of each packet (from head of MAC packet to FCS)

This structure contains: length of packet, MAC address of both sides of communication, length of the head of packet.

(2) len = 12, “buf” contains structure “RxControl”, this structure is not reliable, we can not get neither MAC address of both sides of communication nor length of the head of packet, for AMPDU packet, we can not get the count of packets or the length of packet.

This structure contains: length of packet, rssi (used to estimate if packets are sent from same device), FEC_CODING (used to estimate if packets are sent from same device) .

Notice: we should not take a long time to deal with current packets, otherwise, the following packets maybe loss.

Picture below shows the format of a ieee80211 packet：

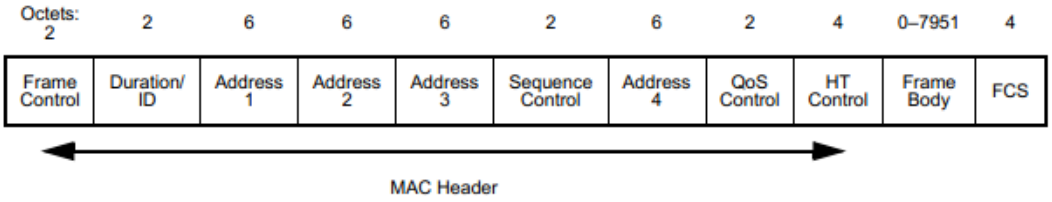

Figure 8-30—Data frame

The first 24 bytes of Mac Header of data packet has to be got; Address4 got or not depends on FromDS and ToDS which in Frame Control; QoS Control got or not depends on Subtype which in Frame Control; HT Control got or not depends on Order Field which in Frame Control; more details in IEEE Std 80211-2012.

For WEP packets, MAC Header is followed by 4 bytes IV, and before FCS there are 4 bytes ICV.

For TKIP packet, MAC Header is followed by 4 bytes IV and 4 bytes EIV, and before FCS there are 8 bytes MIC and 4 bytes ICV.

For CCMP packet, MAC Header is followed by 8 bytes CCMP header, and before FCS there are 8 bytes MIC.