ESP32_RTOS_SDK

V3.0.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

ESP32_RTOS_SDK

- Misc APIs : Misc APIs
- Sensor APIs : Temperature sensor and Touch pad sensor APIs
- WiFi APIs : WiFi related APIs
  - Phy APIs : ESP32 phy layer related APIs
  - Regdomain APIs : ESP32 Regdomain APIs
  - SoftAP APIs : ESP32 Soft-AP APIs
  - Station APIs : ESP32 station APIs
  - Common APIs : WiFi common APIs
  - Sniffer APIs : WiFi sniffer APIs
  - Smartconfig APIs : SmartConfig APIs
- System APIs : System APIs
  - Boot APIs : Boot mode APIs
  - Hardware MAC APIs : Hardware MAC address APIs
- Software timer APIs : Software timer APIs
- Sensor APIs : Temperature Sensor and Touch pad Sensor APIs
- SSC APIs : SSC APIs
- OTA APIs : OTA APIs
- Driver APIs : Driver APIs
  - SPI Driver APIs : SPI Flash APIs
  - GPIO Driver APIs : GPIO APIs
  - I2S Driver APIs : I2S APIs
  - PWM Driver APIs : PWM APIs
  - UART Driver APIs : UART APIs

void user_init(void) is the entrance function of the application.

Attention

1. It is recommended that users set the timer to the periodic mode for periodic checks.
(1). In freeRTOS timer or os_timer, do not delay by while(1) or in the manner that will block the thread.
(2). The timer callback should not occupy CPU more than 15ms.
(3). os_timer_t should not define a local variable, it has to be global varialbe or memory got by malloc.
2. Functions are stored in CACHE by default, need not ICACHE_FLASH_ATTR any more. The interrupt functions can also be stored in CACHE. If users want to store some frequently called functions in RAM, please add IRAM_ATTR before functions' name.
3. Priority of the RTOS SDK is 15. xTaskCreate is an interface of freeRTOS. For details of the freeRTOS and APIs of the system, please visit `http://www.freertos.org`
(1). When using xTaskCreate to create a task, the task stack range is [176, 512].
(2). If an array whose length is over 60 bytes is used in a task, it is suggested that users use malloc and free rather than local variable to allocate array. Large local variables could lead to task stack overflow.
(3). The RTOS SDK takes some priorities. Priority of the pp task is 13; priority of precise timer(ms) thread is 12; priority of the TCP/IP task is 10; priority of the freeRTOS timer is 2; priority of the idle task is 0.
(4). Users can use tasks with priorities from 1 to 9.
(5). Do not revise FreeRTOSConfig.h, configurations are decided by source code inside the RTOS SDK, users can not change it.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 Misc APIs

misc APIs

Data Structures

- struct dhcps_lease

Macros

- #define **MAC2STR**(a) (a)[0], (a)[1], (a)[2], (a)[3], (a)[4], (a)[5]
- #define **MACSTR** "%02x:%02x:%02x:%02x:%02x:%02x"
- #define **IP2STR**(ipaddr)
- #define **IPSTR** "%d.%d.%d.%d"
- #define **os_delay_us** ets_delay_us
- #define **os_install_putc1** ets_install_putc1
- #define **os_putc** ets_putc
- #define **printf_call** ets_printf
- #define **os_printf**(fmt, ...)
- #define **os_printf_isr**(fmt, ...)

Enumerations

- enum dhcp_status { DHCP_STOPPED, DHCP_STARTED }
- enum dhcps_offer_option { OFFER_START = 0x00, OFFER_ROUTER = 0x01, OFFER_END }

Functions

- void **ets_delay_us** (uint16 us)
- void **ets_install_putc1** (void(∗p)(char c))
- void **ets_putc** (char c)
- void **ets_printf** (const char ∗fmt,...)
- unsigned long **os_random** (void)
- int **os_get_random** (uint8 ∗buf, size_t len)
- void ∗ **zalloc** (size_t len)

Variables

- SemaphoreHandle_t **stdio_mutex_tx**

4.1.1 Detailed Description

misc APIs

4.1.2 Macro Definition Documentation

4.1.2.1 #define IP2STR(*ipaddr*)

Value:

```
ip4_addr1_16(ipaddr), \
    ip4_addr2_16(ipaddr), \
    ip4_addr3_16(ipaddr), \
    ip4_addr4_16(ipaddr)
```

4.1.2.2 #define os_printf(*fmt, ...*)

Value:

```
do {    \
        xSemaphoreTake(stdio_mutex_tx, portMAX_DELAY);  \
        printf_call(fmt, ##__VA_ARGS__); \
        xSemaphoreGive(stdio_mutex_tx); \
    } while(0)
```

4.1.2.3 #define os_printf_isr(*fmt, ...*)

Value:

```
do {    \
        static const char ram_str[] DRAM_ATTR STORE_ATTR = fmt;  \
        ets_printf(ram_str, ##__VA_ARGS__);   \
    } while(0)
```

4.1.3 Enumeration Type Documentation

4.1.3.1 enum **dhcp_status**

Enumerator

DHCP_STOPPED	disable DHCP
DHCP_STARTED	enable DHCP

4.1.3.2 enum **dhcps_offer_option**

Enumerator

OFFER_START	DHCP offer option start
OFFER_ROUTER	DHCP offer router, only support this option now
OFFER_END	DHCP offer option start

4.2 WiFi Related APIs

WiFi APIs.

Modules

- Phy APIs

  ESP32 phy layer related APIs.
- Regdomain APIs

  ESP32 Regdomain APIs.
- SoftAP APIs

  ESP32 Soft-AP APIs.
- Station APIs

  ESP32 station APIs.
- Common APIs

  WiFi common APIs.
- Sniffer APIs

  WiFi sniffer APIs.
- WPS APIs

  ESP32 WPS APIs.
- Smartconfig APIs

  SmartConfig APIs.

4.2.1 Detailed Description

WiFi APIs.

4.3 Phy APIs

ESP32 phy layer related APIs.

Data Structures

- struct phy_config

Macros

- #define **CFG_MASK_PHY_MODE** (1<<1)
- #define **CFG_MASK_PHY_BW** (1<<2)
- #define **CFG_MASK_PHY_SND_CHAN** (1<<3)

Enumerations

- enum phy_mode { PHY_MODE_11B = 1, PHY_MODE_11G = 2, PHY_MODE_11N = 3 }
- enum phy_bw { PHY_BW_HT20 = 1, PHY_BW_HT40 = 2, PHY_BW_HT2040 = 3 }
- enum phy_2nd_chan { PHY_HT40U = 1, PHY_HT40D = 2 }

Functions

- bool wifi_set_phy (WIFI_INTERFACE ifx, struct phy_config ∗config)

  Set the phy configuration of ESP32 station/softap and save it to the Flash.
- bool wifi_get_phy (WIFI_INTERFACE ifx, struct phy_config ∗config)

  Get the phy configuration of ESP32 station/softap and save it to the Flash.
- bool wifi_set_phy_mode (WIFI_INTERFACE ifx, enum phy_mode phy_mode)

  Set the phy mode of ESP32 station/softap and save it to the flash.
- bool wifi_set_phy_bw (WIFI_INTERFACE ifx, enum phy_bw phy_bw)

  Set the phy bandwidth of ESP32 station/softap and save it to the flash.
- bool wifi_set_phy_2nd_chan (enum phy_2nd_chan chan)

  Set the phy second channel of ESP32 softap and save it to the flash.

4.3.1 Detailed Description

ESP32 phy layer related APIs.

4.3.2 Enumeration Type Documentation

4.3.2.1 enum **phy_2nd_chan**

Enumerator

PHY_HT40U	the second channel is above the primary channel
PHY_HT40D	the second channel is below the primary channel

4.3.2.2 enum phy_bw

Enumerator

PHY_BW_HT20	phy bandwidth 20M only
PHY_BW_HT40	phy bandwidth 40M only
PHY_BW_HT2040	phy bandwidth 2040M coexist, currently not supported

4.3.2.3 enum phy_mode

Enumerator

PHY_MODE_11B	802.11b mode
PHY_MODE_11G	802.11g mode
PHY_MODE_11N	802.11n mode

4.3.3 Function Documentation

4.3.3.1 bool wifi_get_phy (WIFI_INTERFACE *ifx,* struct phy_config ∗ *config*)

Get the phy configuration of ESP32 station/softap and save it to the Flash.

Parameters

WIFI_INTERFACE	: station/softap interface
struct	phy_config ∗config : phy configuration

Returns

true : success
false : failure

4.3.3.2 bool wifi_set_phy (WIFI_INTERFACE *ifx,* struct phy_config ∗ *config*)

Set the phy configuration of ESP32 station/softap and save it to the Flash.

Attention

1. This API can be called only when the opmode is correctly configured, e.g. if you try to configure softap phy while the opmode is station, it returns false.
2. If call this API when station/softap is started(connected), it will firstly stop/disconnect the station/softap, configure phy, then start/connect station/softap.
3. If you change the phy mode from 11N HT40 to 11b/g, the bandwith will be switched to HT20 since the HT40 is for 11N only.
4. You can configure phy mode/bw/2nd_chan by mask the cfg_mask correctly, e.g. if (cfg_mask & CFG_M←ASK_PHY_MODE) is true, the phy mode will be configued, otherwise, it has no impact on the phy mode.
5. Since wifi_set_phy_mode/wifi_set_phy_bw/wifi_set_2nd_chan will stop station/softap and then start station/softap. So if you don't want station/softap stop/start several times, please use wifi_set_phy since it only stop/start once. Of course, if you only want to configure phy mode, you can use the one you prefer.
6. The default sdk phy configuration: 11N/HT40

Parameters

WIFI_INTERFACE	: interface to be configured
struct	phy_config ∗config : phy configuration

Returns

true : success
false : failure

4.3.3.3 bool wifi_set_phy_2nd_chan (enum **phy_2nd_chan** *chan*)

Set the phy second channel of ESP32 softap and save it to the flash.

Attention

Refer to wifi_set_phy()

Parameters

enum	phy_2nd_chan : phy configuration

Returns

true : success
false : failure

4.3.3.4 bool wifi_set_phy_bw (**WIFI_INTERFACE** *ifx,* enum **phy_bw** *phy_bw*)

Set the phy bandwidth of ESP32 station/softap and save it to the flash.

Attention

See wifi_set_phy()

Parameters

WIFI_INTERFACE	: interface to be configured
enum	phy_bw phy_bw : phy bandwidth

Returns

true : success
false : failure

4.3.3.5 bool wifi_set_phy_mode (**WIFI_INTERFACE** *ifx,* enum **phy_mode** *phy_mode*)

Set the phy mode of ESP32 station/softap and save it to the flash.

Attention

Refer to wifi_set_phy()

Parameters

WIFI_INTERFACE	: interface to be configured
enum	phy_mode phy_mode : phy mode

Returns

true : success
false : failure

4.4 Regdomain APIs

ESP32 Regdomain APIs.

Data Structures

- struct regdomain_info

Macros

- #define **REGDOMAIN_CHANGROUP_MAX** 1

Functions

- bool wifi_get_regdomain (WIFI_INTERFACE ifx, struct regdomain_info ∗rd)

  Get the regdomain info of station/softap.
- bool wifi_set_regdomain (WIFI_INTERFACE ifx, struct regdomain_info ∗rd)

  Set the regdomain info of station/softap.

4.4.1 Detailed Description

ESP32 Regdomain APIs.

4.4.2 Function Documentation

4.4.2.1 bool wifi_get_regdomain (WIFI_INTERFACE *ifx*, struct regdomain_info ∗ *rd*)

Get the regdomain info of station/softap.

Parameters

WIFI_INTERFACE	ifx : station/softap interface
struct	regdomain_info ∗rd : For station, only field rd_sta_enable is valid, please ignore all other fields. For softap, if rd_ap_enable is 1, then fields from regdomain to chan are valid, otherwise, all other fields are invalid.

Returns

true : success
false : failure

4.4.2.2 bool wifi_set_regdomain (WIFI_INTERFACE *ifx*, struct regdomain_info ∗ *rd*)

Set the regdomain info of station/softap.

Attention

1. If the regodmain of station is enable, the station will use passive scan to scan channel 1∼14, NO PROBE REQUEST will send out, this is also the behavior specified in 80211-2012 standard. However, this may impact the scan performance, e.g. it can't find the hidden AP etc. If you think this not what you expected, pls contact espressif.
2. For softap, if regdomain is enable, the regodmain info will be added into beacon.
3. If current channel of softap is not in [schan, schan+nchan-1], then softap will switch to the channel schan automatically
4. In stationap mode, if the regdomain information of softap is conflicted with the regodmain information of the external AP that sta is conneted, then the external AP's regodmain information is prefered.

Parameters

WIFI_INTERFACE	ifx : station/softap interface
struct	regdomain_info ∗rd : For station, only field rd_sta_enable is valid, please ignore all other field. For softap, if rd_ap_enable is 1, then fields from regdomain to chan are valid, otherwise, all other fields are invalid.

Returns

true : success
false : failure

4.5 Sensor APIs

Temperature Sensor and Touch pad Sensor APIs.

Enumerations

- enum **touch_sensor_pad** {
  TOUCH_SENSOR_PAD0 = BIT0, **TOUCH_SENSOR_PAD1** = BIT1, **TOUCH_SENSOR_PAD2** = BIT2, **T**↩
  OUCH_SENSOR_PAD3 = BIT3,
  TOUCH_SENSOR_PAD4 = BIT4, **TOUCH_SENSOR_PAD5** = BIT5, **TOUCH_SENSOR_PAD6** = BIT6, **T**↩
  OUCH_SENSOR_PAD7 = BIT7,
  TOUCH_SENSOR_PAD8 = BIT8, **TOUCH_SENSOR_PAD9** = BIT9 }

Functions

- uint8 temperature_sensor_read (void)

  Read value from temperature sensor.
- void touch_sensor_init (touch_sensor_pad pad)

  Initialize touch pad sensor.
- void touch_sensor_read (uint16 ∗pad_out, uint16 sample_num)

  Read value from touch pad sensor.

4.5.1 Detailed Description

Temperature Sensor and Touch pad Sensor APIs.

4.5.2 Function Documentation

4.5.2.1 uint8 temperature_sensor_read (void)

Read value from temperature sensor.

Parameters

null	

Returns

range [0, 255]

4.5.2.2 void touch_sensor_init (touch_sensor_pad *pad*)

Initialize touch pad sensor.

Parameters

touch_sensor_pad	pad : enable the corresponding touch_pad[9:0]

Returns

null

4.5.2.3 void touch_sensor_read (uint16 ∗ *pad_out,* uint16 *sample_num*)

Read value from touch pad sensor.

Example:

```
uint16 pad_out[10];
uint16 sample_num = 10000;
rtc_touch_read(pad_out, sample_num);
```

Parameters

uint16	∗pad_out : pointer of the start address of uint16 pad_out[10], to get the value from touch pad sensor (touch_pad[9:0]).
uint16	sample_num : range [0, 65535], meaturing time of touch pad = sample_num∗(1/RTC_CLK)

Returns

null

4.6 SoftAP APIs

ESP32 Soft-AP APIs.

Data Structures

- struct softap_config
- struct station_info

Functions

- bool wifi_softap_get_config (struct softap_config ∗config)

  Get the current configuration of the ESP32 WiFi soft-AP.
- bool wifi_softap_get_config_default (struct softap_config ∗config)

  Get the configuration of the ESP32 WiFi soft-AP saved in the flash.
- bool wifi_softap_set_config (struct softap_config ∗config)

  Set the configuration of the WiFi soft-AP and save it to the Flash.
- bool wifi_softap_set_config_current (struct softap_config ∗config)

  Set the configuration of the WiFi soft-AP; the configuration will not be saved to the Flash.
- uint8 wifi_softap_get_station_num (void)

  Get the number of stations connected to the ESP32 soft-AP.
- struct station_info ∗ wifi_softap_get_station_info (void)

  Get the information of stations connected to the ESP32 soft-AP, including MAC and IP.
- void wifi_softap_free_station_info (void)

  Free the space occupied by station_info when wifi_softap_get_station_info is called.
- bool wifi_softap_dhcps_start (void)

  Enable the ESP32 soft-AP DHCP server.
- bool wifi_softap_dhcps_stop (void)

  Disable the ESP32 soft-AP DHCP server. The DHCP is enabled by default.
- enum dhcp_status wifi_softap_dhcps_status (void)

  Get the ESP32 soft-AP DHCP server status.
- bool wifi_softap_get_dhcps_lease (struct dhcps_lease ∗please)

  Query the IP range that can be got from the ESP32 soft-AP DHCP server.
- bool wifi_softap_set_dhcps_lease (struct dhcps_lease ∗please)

  Set the IP range of the ESP32 soft-AP DHCP server.
- uint32 wifi_softap_get_dhcps_lease_time (void)

  Get ESP32 soft-AP DHCP server lease time.
- bool wifi_softap_set_dhcps_lease_time (uint32 minute)

  Set ESP32 soft-AP DHCP server lease time, default is 120 minutes.
- bool wifi_softap_reset_dhcps_lease_time (void)

  Reset ESP32 soft-AP DHCP server lease time which is 120 minutes by default.
- bool wifi_softap_set_dhcps_offer_option (uint8 level, void ∗optarg)

  Set the ESP32 soft-AP DHCP server option.

4.6.1 Detailed Description

ESP32 Soft-AP APIs.

Attention

To call APIs related to ESP32 soft-AP has to enable soft-AP mode first (wifi_set_opmode)

4.6.2 Function Documentation

4.6.2.1 bool wifi_softap_dhcps_start (void)

Enable the ESP32 soft-AP DHCP server.

Attention

1. The DHCP is enabled by default.
2. The DHCP and the static IP related API (wifi_set_ip_info) influence each other, if the DHCP is enabled, the static IP will be disabled; if the static IP is enabled, the DHCP will be disabled. It depends on the latest configuration.

Parameters

null	

Returns

true : succeed
false : fail

4.6.2.2 enum dhcp_status wifi_softap_dhcps_status (void)

Get the ESP32 soft-AP DHCP server status.

Parameters

null	

Returns

enum dhcp_status

4.6.2.3 bool wifi_softap_dhcps_stop (void)

Disable the ESP32 soft-AP DHCP server. The DHCP is enabled by default.

Parameters

null	

Returns

true : succeed
false : fail

4.6.2.4 void wifi_softap_free_station_info (void)

Free the space occupied by station_info when wifi_softap_get_station_info is called.

Attention

The ESP32 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP32 station.

Parameters

null	

Returns

null

4.6.2.5 bool wifi_softap_get_config (struct softap_config ∗ *config*)

Get the current configuration of the ESP32 WiFi soft-AP.

Parameters

struct	softap_config ∗config : ESP32 soft-AP configuration

Returns

true : succeed
false : fail

4.6.2.6 bool wifi_softap_get_config_default (struct softap_config ∗ *config*)

Get the configuration of the ESP32 WiFi soft-AP saved in the flash.

Parameters

struct	softap_config ∗config : ESP32 soft-AP configuration

Returns

true : succeed
false : fail

4.6.2.7 bool wifi_softap_get_dhcps_lease (struct dhcps_lease ∗ *please*)

Query the IP range that can be got from the ESP32 soft-AP DHCP server.

Attention

This API can only be called during ESP32 soft-AP DHCP server enabled.

Parameters

struct	dhcps_lease ∗please : IP range of the ESP32 soft-AP DHCP server.

Returns

true : succeed
false : fail

4.6.2.8 uint32 wifi_softap_get_dhcps_lease_time (void)

Get ESP32 soft-AP DHCP server lease time.

Attention

This API can only be called during ESP32 soft-AP DHCP server enabled.

Parameters

null	

Returns

lease time, uint: minute.

4.6.2.9 struct station_info∗ wifi_softap_get_station_info (void)

Get the information of stations connected to the ESP32 soft-AP, including MAC and IP.

Attention

wifi_softap_get_station_info depends on DHCP, it can only be used when DHCP is enabled, so it can not get the static IP.

Parameters

null	

Returns

struct station_info∗ : station information structure

4.6.2.10 uint8 wifi_softap_get_station_num (void)

Get the number of stations connected to the ESP32 soft-AP.

Attention

The ESP32 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP32 station.

Parameters

null	

Returns

the number of stations connected to the ESP32 soft-AP

4.6.2.11 bool wifi_softap_reset_dhcps_lease_time (void)

Reset ESP32 soft-AP DHCP server lease time which is 120 minutes by default.

Attention

This API can only be called during ESP32 soft-AP DHCP server enabled.

Parameters

null	

Returns

true : succeed
false : fail

4.6.2.12 bool wifi_softap_set_config (struct softap_config ∗ *config*)

Set the configuration of the WiFi soft-AP and save it to the Flash.

Attention

1. This configuration will be saved in flash system parameter area if changed
2. The ESP32 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP32 station.

Parameters

struct	softap_config ∗config : ESP32 soft-AP configuration

Returns

true : succeed
false : fail

4.6.2.13 bool wifi_softap_set_config_current (struct **softap_config** ∗ *config*)

Set the configuration of the WiFi soft-AP; the configuration will not be saved to the Flash.

Attention

The ESP32 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP32 station.

Parameters

struct	softap_config ∗config : ESP32 soft-AP configuration

Returns

true : succeed
false : fail

4.6.2.14 bool wifi_softap_set_dhcps_lease (struct **dhcps_lease** ∗ *please*)

Set the IP range of the ESP32 soft-AP DHCP server.

Attention

1. The IP range should be in the same sub-net with the ESP32 soft-AP IP address.
2. This API should only be called when the DHCP server is disabled (wifi_softap_dhcps_stop).
3. This configuration will only take effect the next time when the DHCP server is enabled (wifi_softap_dhcps←_start).

  - If the DHCP server is disabled again, this API should be called to set the IP range.
  - Otherwise, when the DHCP server is enabled later, the default IP range will be used.

Parameters

struct	dhcps_lease ∗please : IP range of the ESP32 soft-AP DHCP server.

Returns

true : succeed
false : fail

4.6.2.15 bool wifi_softap_set_dhcps_lease_time (uint32 *minute*)

Set ESP32 soft-AP DHCP server lease time, default is 120 minutes.

Attention

This API can only be called during ESP32 soft-AP DHCP server enabled.

Parameters

uint32	minute : lease time, uint: minute, range:[1, 2880].

Returns

true : succeed
false : fail

4.6.2.16 bool wifi_softap_set_dhcps_offer_option (uint8 *level,* void ∗ *optarg*)

Set the ESP32 soft-AP DHCP server option.

Example:

```
uint8 mode = 0;
wifi_softap_set_dhcps_offer_option(OFFER_ROUTER, &mode);
```

Parameters

uint8	level : OFFER_ROUTER, set the router option.
void∗	optarg : • bit0, 0 disable the router information; • bit0, 1 enable the router information.

Returns

true : succeed
false : fail

4.7 SSC APIs

SSC APIs.

Functions

- void ssc_attach (SscBaudRate bandrate)

  Initial the ssc function.
- int ssc_param_len (void)

  Get the length of the simple serial command.
- char ∗ ssc_param_str (void)

  Get the simple serial command string.
- int ssc_parse_param (char ∗pLine, char ∗argv[ ])

  Parse the simple serial command (ssc).
- void ssc_register (ssc_cmd_t ∗cmdset, uint8 cmdnum, void(∗help)(void))

  Register the user-defined simple serial command (ssc) set.

4.7.1 Detailed Description

SSC APIs.

SSC means simple serial command. SSC APIs allows users to define their own command, users can refer to spiffs_test/test_main.c.

4.7.2 Function Documentation

4.7.2.1 void ssc_attach (SscBaudRate *bandrate*)

Initial the ssc function.

Attention

param is no use, just compatible with ESP8266, default bandrate is 115200

Parameters

SscBaudRate	bandrate : baud rate

Returns

null

4.7.2.2 int ssc_param_len (void)

Get the length of the simple serial command.

Parameters

null	

Returns

length of the command.

4.7.2.3 char∗ ssc_param_str (void)

Get the simple serial command string.

Parameters

null	

Returns

the command.

4.7.2.4 int ssc_parse_param (char ∗ *pLine,* char ∗ *argv[ ]*)

Parse the simple serial command (ssc).

Parameters

char	∗pLine : [input] the ssc string
char	∗argv[] : [output] parameters of the ssc

Returns

the number of parameters.

4.7.2.5 void ssc_register (ssc_cmd_t ∗ *cmdset,* uint8 *cmdnum,* void(∗)(void) *help*)

Register the user-defined simple serial command (ssc) set.

Parameters

ssc_↩ cmd_t	∗cmdset : the ssc set
uint8	cmdnum : number of commands
void	(∗ help)(void) : callback of user-guide

Returns

null

4.8 Station APIs

ESP32 station APIs.

Data Structures

- struct station_config
- struct scan_config
- struct bss_info

Typedefs

- typedef void(∗ scan_done_cb_t) (void ∗arg, STATUS status)

  Callback function for wifi_station_scan.

Enumerations

- enum STATION_STATUS {
  STATION_IDLE = 0, STATION_CONNECTING, STATION_WRONG_PASSWORD, STATION_NO_AP_F←
  OUND,
  STATION_CONNECT_FAIL, STATION_GOT_IP }

Functions

- bool wifi_station_get_config (struct station_config ∗config)

  Get the current configuration of the ESP32 WiFi station.
- bool wifi_station_get_config_default (struct station_config ∗config)

  Get the configuration parameters saved in the Flash of the ESP32 WiFi station.
- bool wifi_station_set_config (struct station_config ∗config)

  Set the configuration of the ESP32 station and save it to the Flash.
- bool wifi_station_set_config_current (struct station_config ∗config)

  Set the configuration of the ESP32 station. And the configuration will not be saved to the Flash.
- bool wifi_station_connect (void)

  Connect the ESP32 WiFi station to the AP.
- bool wifi_station_disconnect (void)

  Disconnect the ESP32 WiFi station from the AP.
- bool wifi_station_scan (struct scan_config ∗config, scan_done_cb_t cb)

  Scan all available APs.
- bool wifi_station_get_auto_connect (void)

  Check if the ESP32 station will connect to the recorded AP automatically when the power is on.
- bool wifi_station_set_auto_connect (bool set)

  Set whether the ESP32 station will connect to the recorded AP automatically when the power is on. It will do so by default.
- bool wifi_station_get_reconnect_policy (void)

  Check whether the ESP32 station will reconnect to the AP after disconnection.
- bool wifi_station_set_reconnect_policy (bool set)

  Set whether the ESP32 station will reconnect to the AP after disconnection. It will do so by default.
- STATION_STATUS wifi_station_get_connect_status (void)

Get the connection status of the ESP32 WiFi station.

- uint8 wifi_station_get_current_ap_id (void)

  Get the information of APs (5 at most) recorded by ESP32 station.
- bool wifi_station_ap_change (uint8 current_ap_id)

  Switch the ESP32 station connection to a recorded AP.
- bool wifi_station_ap_number_set (uint8 ap_number)

  Set the number of APs that can be recorded in the ESP32 station. When the ESP32 station is connected to an AP, the SSID and password of the AP will be recorded.
- uint8 wifi_station_get_ap_info (struct station_config config[ ])

  Get the information of APs (5 at most) recorded by ESP32 station.
- sint8 wifi_station_get_rssi (void)

  Get rssi of the AP which ESP32 station connected to.
- bool wifi_station_dhcpc_start (void)

  Enable the ESP32 station DHCP client.
- bool wifi_station_dhcpc_stop (void)

  Disable the ESP32 station DHCP client.
- enum dhcp_status wifi_station_dhcpc_status (void)

  Get the ESP32 station DHCP client status.

4.8.1 Detailed Description

ESP32 station APIs.

Attention

To call APIs related to ESP32 station has to enable station mode first (wifi_set_opmode)

4.8.2 Typedef Documentation

4.8.2.1 typedef void(∗ scan_done_cb_t) (void ∗arg, STATUS status)

Callback function for wifi_station_scan.

Parameters

void	∗arg : information of APs that are found; save them as linked list; refer to struct bss_info
STATUS	status : status of scanning

Returns

null

4.8.3 Enumeration Type Documentation

4.8.3.1 enum STATION_STATUS

Enumerator

STATION_IDLE	ESP32 station idle

STATION_CONNECTING ESP32 station is connecting to AP

STATION_WRONG_PASSWORD the password is wrong

STATION_NO_AP_FOUND ESP32 station can not find the target AP

STATION_CONNECT_FAIL ESP32 station fail to connect to AP

STATION_GOT_IP ESP32 station got IP address from AP

4.8.4 Function Documentation

4.8.4.1 bool wifi_station_ap_change (uint8 *current_ap_id*)

Switch the ESP32 station connection to a recorded AP.

Parameters

uint8	new_ap_id : AP's record id, start counting from 0.

Returns

true : succeed
false : fail

4.8.4.2 bool wifi_station_ap_number_set (uint8 *ap_number*)

Set the number of APs that can be recorded in the ESP32 station. When the ESP32 station is connected to an AP, the SSID and password of the AP will be recorded.

Attention

This configuration will be saved in the Flash system parameter area if changed.

Parameters

uint8	ap_number : the number of APs that can be recorded (MAX: 5)

Returns

true : succeed
false : fail

4.8.4.3 bool wifi_station_connect (void)

Connect the ESP32 WiFi station to the AP.

Attention

1. This API should be called when the ESP32 station is enabled, and the system initialization is completed. Do not call this API in user_init.
2. If the ESP32 is connected to an AP, call wifi_station_disconnect to disconnect.

Parameters

null	

Returns

true : succeed
false : fail

4.8.4.4 bool wifi_station_dhcpc_start (void)

Enable the ESP32 station DHCP client.

Attention

1. The DHCP is enabled by default.
2. The DHCP and the static IP API ((wifi_set_ip_info)) influence each other, and if the DHCP is enabled, the static IP will be disabled; if the static IP is enabled, the DHCP will be disabled. It depends on the latest configuration.

Parameters

null	

Returns

true : succeed
false : fail

4.8.4.5 enum dhcp_status wifi_station_dhcpc_status (void)

Get the ESP32 station DHCP client status.

Parameters

null	

Returns

enum dhcp_status

4.8.4.6 bool wifi_station_dhcpc_stop (void)

Disable the ESP32 station DHCP client.

Attention

1. The DHCP is enabled by default.
2. The DHCP and the static IP API ((wifi_set_ip_info)) influence each other, and if the DHCP is enabled, the static IP will be disabled; if the static IP is enabled, the DHCP will be disabled. It depends on the latest configuration.

Parameters

null	

Returns

true : succeed
false : fail

4.8.4.7 bool wifi_station_disconnect (void)

Disconnect the ESP32 WiFi station from the AP.

Attention

This API should be called when the ESP32 station is enabled, and the system initialization is completed. Do not call this API in user_init.

Parameters

null	

Returns

true : succeed
false : fail

4.8.4.8 uint8 wifi_station_get_ap_info (struct station_config *config[ ]*)

Get the information of APs (5 at most) recorded by ESP32 station.

Example:

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

Parameters

struct	station_config config[] : information of the APs, the array size should be 5.

Returns

The number of APs recorded.

4.8.4.9 bool wifi_station_get_auto_connect (void)

Check if the ESP32 station will connect to the recorded AP automatically when the power is on.

Parameters

null	

Returns

true : connect to the AP automatically
false : not connect to the AP automatically

4.8.4.10 bool wifi_station_get_config (struct **station_config** ∗ *config*)

Get the current configuration of the ESP32 WiFi station.

Parameters

struct	station_config ∗config : ESP32 station configuration

Returns

true : succeed
false : fail

4.8.4.11 bool wifi_station_get_config_default (struct **station_config** ∗ *config*)

Get the configuration parameters saved in the Flash of the ESP32 WiFi station.

Parameters

struct	station_config ∗config : ESP32 station configuration

Returns

true : succeed
false : fail

4.8.4.12 STATION_STATUS wifi_station_get_connect_status (void)

Get the connection status of the ESP32 WiFi station.

Parameters

null	

Returns

the status of connection

4.8.4.13 uint8 wifi_station_get_current_ap_id (void)

Get the information of APs (5 at most) recorded by ESP32 station.

Parameters

struct	station_config config[] : information of the APs, the array size should be 5.

Returns

The number of APs recorded.

4.8.4.14 bool wifi_station_get_reconnect_policy (void)

Check whether the ESP32 station will reconnect to the AP after disconnection.

Parameters

null	

Returns

true : succeed
false : fail

4.8.4.15 sint8 wifi_station_get_rssi (void)

Get rssi of the AP which ESP32 station connected to.

Parameters

null	

Returns

31 : fail, invalid value.
others : succeed, value of rssi. In general, rssi value $<$ 10

4.8.4.16 bool wifi_station_scan (struct **scan_config** ∗ *config,* **scan_done_cb_t** *cb*)

Scan all available APs.

Attention

This API should be called when the ESP32 station is enabled, and the system initialization is completed. Do not call this API in user_init.

Parameters

struct	scan_config ∗config : configuration of scanning
struct	scan_done_cb_t cb : callback of scanning

Returns

true : succeed
false : fail

4.8.4.17 bool wifi_station_set_auto_connect (bool *set*)

Set whether the ESP32 station will connect to the recorded AP automatically when the power is on. It will do so by default.

Attention

1. If this API is called in user_init, it is effective immediately after the power is on. If it is called in other places, it will be effective the next time when the power is on.
2. This configuration will be saved in Flash system parameter area if changed.

Parameters

bool	set : If it will automatically connect to the AP when the power is on • true : it will connect automatically • false: it will not connect automatically

Returns

true : succeed
false : fail

4.8.4.18 bool wifi_station_set_config (struct **station_config** ∗ *config*)

Set the configuration of the ESP32 station and save it to the Flash.

Attention

1. This API can be called only when the ESP32 station is enabled.
2. If wifi_station_set_config is called in user_init , there is no need to call wifi_station_connect. The ESP32 station will automatically connect to the AP (router) after the system initialization. Otherwise, wifi_station_↩ connect should be called.
3. Generally, station_config.bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.
4. This configuration will be saved in the Flash system parameter area if changed.

Parameters

struct	station_config ∗config : ESP32 station configuration

Returns

true : succeed
false : fail

4.8.4.19 bool wifi_station_set_config_current (struct station_config ∗ *config*)

Set the configuration of the ESP32 station. And the configuration will not be saved to the Flash.

Attention

1. This API can be called only when the ESP32 station is enabled.
2. If wifi_station_set_config_current is called in user_init , there is no need to call wifi_station_connect. The ESP32 station will automatically connect to the AP (router) after the system initialization. Otherwise, wifi_↩ station_connect should be called.
3. Generally, station_config.bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

Parameters

struct	station_config ∗config : ESP32 station configuration

Returns

true : succeed
false : fail

4.8.4.20 bool wifi_station_set_reconnect_policy (bool *set*)

Set whether the ESP32 station will reconnect to the AP after disconnection. It will do so by default.

Attention

If users want to call this API, it is suggested that users call this API in user_init.

Parameters

bool	set : if it's true, it will enable reconnection; if it's false, it will disable reconnection.

Returns

true : succeed
false : fail

4.9 System APIs

System APIs.

Modules

- Boot APIs

  boot APIs
- Hardware MAC APIs

  Hardware MAC address APIs.

Enumerations

- enum **adc1_read_pad** {
  ADC1_PAD_GPIO36 = 0, **ADC1_PAD_GPIO37**, **ADC1_PAD_GPIO38**, **ADC1_PAD_GPIO39**,
  ADC1_PAD_GPIO32, **ADC1_PAD_GPIO33**, **ADC1_PAD_GPIO34**, **ADC1_PAD_GPIO35** }
- enum **adc1_read_atten** { **ADC1_ATTEN_0DB** = 0, **ADC1_ATTEN_3DB**, **ADC1_ATTEN_6DB**, **ADC1_A↩
  TTEN_12DB** }

Functions

- const char ∗ system_get_sdk_version (void)

  Get information of the SDK version.
- void system_restore (void)

  Reset to default settings.
- void system_restart (void)

  Restart system.
- void system_deep_sleep (uint64 time_in_us)

  Set the chip to deep-sleep mode.
- uint32 system_get_time (void)

  Get system time, unit: microsecond.
- void system_print_meminfo (void)

  Print the system memory distribution, including data/rodata/bss/heap.
- uint32 system_get_free_heap_size (void)

  Get the size of available heap.
- bool system_get_chip_id (uint8 ∗chip_id)

  Get the chip ID.
- uint64 system_get_rtc_time (void)

  Get RTC time, unit: RTC clock cycle.
- bool system_rtc_mem_read (uint16 src, void ∗dst, uint16 n)

  Read user data from the RTC memory.
- bool system_rtc_mem_write (uint16 dst, const void ∗src, uint16 n)

  Write user data to the RTC memory.
- uint16 system_adc1_read (adc1_read_pad pad, adc1_read_atten atten)

  Read ADC1.
- uint16 system_get_vdd33 (void)

  Measure the power voltage of VDD3P3 pin 3 and 4, unit : 1/1024 V.
- bool system_param_save_with_protect (uint16 start_sec, void ∗param, uint16 len)

  Write data into flash with protection.
- bool system_param_load (uint16 start_sec, uint16 offset, void ∗param, uint16 len)

  Read the data saved into flash with the read/write protection.

4.9.1 Detailed Description

System APIs.

4.9.2 Function Documentation

4.9.2.1 uint16 system_adc1_read (adc1_read_pad *pad,* adc1_read_atten *atten*)

Read ADC1.

Parameters

adc1_read_pad	pad : the corresponding GPIO
adc1_read_atten	atten : value of attenuation

Returns

range of the return value is [0, 4096].

- If atten == 0, the range of voltage can be measured is [0, 1] V.
- If atten == 1, the range of voltage can be measured is [0, 1.4] V.
- If atten == 2, the range of voltage can be measured is [0, 2] V.
- If atten == 3, the range of voltage can be measured is [0, 4] V.

4.9.2.2 void system_deep_sleep (uint64 *time_in_us*)

Set the chip to deep-sleep mode.

The device will automatically wake up after the deep-sleep time set by the users. Upon waking up, the device boots up from user_init.

Attention

The parameter time_in_us to be "uint64" is for further development. Only the low 32 bits of parameter time↩ _in_us are avalable now.

Parameters

uint64	time_in_us : deep-sleep time, only the low 32bits are avalable now. unit: microsecond

Returns

null

4.9.2.3 bool system_get_chip_id (uint8 ∗ *chip_id*)

Get the chip ID.

Example:

```
        uint8 chip_id[6];
        system_get_chip_id(chip_id);
```

Parameters

uint8	∗chip_id : the chip ID

Returns

true : succeed
false : fail

4.9.2.4 uint32 system_get_free_heap_size (void)

Get the size of available heap.

Parameters

null	

Returns

Available heap size.

4.9.2.5 uint64 system_get_rtc_time (void)

Get RTC time, unit: RTC clock cycle.

Parameters

null	

Returns

RTC time.

4.9.2.6 const char∗ system_get_sdk_version (void)

Get information of the SDK version.

Parameters

null	

Returns

Information of the SDK version.

4.9.2.7 uint32 system_get_time (void)

Get system time, unit: microsecond.

Parameters

null	

Returns

System time, unit: microsecond.

4.9.2.8 uint16 system_get_vdd33 (void)

Measure the power voltage of VDD3P3 pin 3 and 4, unit : 1/1024 V.

Attention

system_get_vdd33 depends on RF, please do not use it if RF is disabled.

Parameters

null	

Returns

Power voltage of VDD33, unit : 1/1024 V

4.9.2.9 bool system_param_load (uint16 *start_sec,* uint16 *offset,* void ∗ *param,* uint16 *len*)

Read the data saved into flash with the read/write protection.

Flash read/write has to be 4-bytes aligned.

Read/write protection of flash: use 3 sectors (4KB per sector) to save 4KB data with protect, sector 0 and sector 1 are data sectors, back up each other, save data alternately, sector 2 is flag sector, point out which sector is keeping the latest data, sector 0 or sector 1.

Parameters

uint16	start_sec : start sector (sector 0) of the 3 sectors used for flash read/write protection. It cannot be sector 1 or sector 2. • For example, in IOT_Demo, the 3 sectors (3 ∗ 4KB) starting from flash 0x3D000 can be used for flash read/write protection. The parameter start_sec is 0x3D, and it cannot be 0x3E or 0x3F.
uint16	offset : offset of data saved in sector
void	∗param : data pointer
uint16	len : data length, offset + len =< 4 ∗ 1024

Returns

true : succeed
false : fail

4.9.2.10 bool system_param_save_with_protect (uint16 *start_sec,* void ∗ *param,* uint16 *len*)

Write data into flash with protection.

Flash read/write has to be 4-bytes aligned.

Protection of flash read/write : use 3 sectors (4KBytes per sector) to save 4KB data with protect, sector 0 and sector 1 are data sectors, back up each other, save data alternately, sector 2 is flag sector, point out which sector is keeping the latest data, sector 0 or sector 1.

Parameters

uint16	start_sec : start sector (sector 0) of the 3 sectors which are used for flash read/write protection. • For example, in IOT_Demo we can use the 3 sectors (3 ∗ 4KB) starting from flash 0x3D000 for flash read/write protection, so the parameter start_sec should be 0x3D
void	∗param : pointer of the data to be written
uint16	len : data length, should be less than a sector, which is 4 ∗ 1024

Returns

true : succeed
false : fail

4.9.2.11 void system_print_meminfo (void)

Print the system memory distribution, including data/rodata/bss/heap.

Parameters

null	

Returns

null

4.9.2.12 void system_restart (void)

Restart system.

Parameters

null	

Returns

null

4.9.2.13 void system_restore (void)

Reset to default settings.

Reset to default settings of the following APIs : wifi_station_set_auto_connect, wifi_set_phy_mode, wifi_softap_↩ set_config related, wifi_station_set_config related, and wifi_set_opmode.

Parameters

null	

Returns

null

4.9.2.14 bool system_rtc_mem_read (uint16 *src,* void ∗ *dst,* uint16 *n*)

Read user data from the RTC memory.

The user data segment (1024 bytes, as shown below) is used to store user data.

|<-— system data(512 bytes) -—>|<---------— user data(1024 bytes) --------—>|

Attention

Read and write unit for data stored in the RTC memory is 4 bytes.
src_addr is the block number (4 bytes per block). So when reading data at the beginning of the user data segment, src_addr will be 512/4 = 128, n will be data length.

Parameters

uint16	src : source address of rtc memory, src_addr $>$= 128
void	$*$dst : data pointer
uint16	n : data length, unit: byte

Returns

true : succeed
false : fail

4.9.2.15 bool system_rtc_mem_write (uint16 *dst,* const void $*$ *src,* uint16 *n*)

Write user data to the RTC memory.

During deep-sleep, only RTC is working. So users can store their data in RTC memory if it is needed. The user data segment below (1024 bytes) is used to store the user data.

$|<$-— system data(512 bytes) -—$>|<$---------— user data(1024 bytes) -------—$>|$

Attention

Read and write unit for data stored in the RTC memory is 4 bytes.
src_addr is the block number (4 bytes per block). So when storing data at the beginning of the user data segment, src_addr will be 512/4 = 128, n will be data length.

Parameters

uint16	src : source address of rtc memory, src_addr $>$= 128
void	$*$dst : data pointer
uint16	n : data length, unit: byte

Returns

true : succeed
false : fail

4.10 Boot APIs

boot APIs

Data Structures

- struct b_info

Enumerations

- enum flash_size {
  FLASH_SIZE_1MB = 0, FLASH_SIZE_2MB, FLASH_SIZE_4MB, FLASH_SIZE_8MB,
  FLASH_SIZE_16MB, **FLASH_SIZE_MAX** }

Functions

- enum flash_size system_get_flash_size (void)

  Get the current Flash size.
- uint8 system_get_cpu_freq (void)

  Get CPU frequency.
- bool system_get_bin_info (uint8 bin_id, struct b_info ∗b_if)

  Get bin info named by b_id.
- bool system_set_bin_info (uint8 bin_id, struct b_info ∗b_if)

  Set bin info named by b_id.
- uint8 system_get_current_bin_id (void)

  Get current bin's bin_id.
- bool system_reboot_to_userbin (uint8 bin_id)

  reboot and jump to bin named by bin_id

4.10.1 Detailed Description

boot APIs

4.10.2 Enumeration Type Documentation

4.10.2.1 enum **flash_size**

Enumerator

FLASH_SIZE_1MB	Flash size : 1M Bytes
FLASH_SIZE_2MB	Flash size : 2M Bytes
FLASH_SIZE_4MB	Flash size : 4M Bytes
FLASH_SIZE_8MB	Flash size : 8M Bytes
FLASH_SIZE_16MB	Flash size : 16M Bytes

4.10.3 Function Documentation

4.10.3.1 bool system_get_bin_info (uint8 *bin_id,* struct **b_info** ∗ *b_if*)

Get bin info named by b_id.

Parameters

uint8	bin_id : b_id number, must < 5
struct	b_info $*$b_if : bin info of bin named by b_id

Returns

true : succeed
false : fail

4.10.3.2 uint8 system_get_cpu_freq (void)

Get CPU frequency.

Parameters

null	

Returns

CPU frequency, unit : MHz.

4.10.3.3 uint8 system_get_current_bin_id (void)

Get current bin's bin_id.

Parameters

uint8	bin_id : b_id number, must < 5

Returns

uint8 type b_id

4.10.3.4 enum **flash_size** system_get_flash_size (void)

Get the current Flash size.

Parameters

null	

Returns

enum flash_size

4.10.3.5 bool system_reboot_to_userbin (uint8 *bin_id*)

reboot and jump to bin named by bin_id

Parameters

uint8	bin_id : b_id number, must $<$ 5

Returns

true : succeed
false : fail

4.10.3.6 bool system_set_bin_info (uint8 *bin_id,* struct b_info $*$ *b_if*)

Set bin info named by b_id.

Parameters

uint8	bin_id : b_id number, must $<$ 5
struct	b_info $*$b_if : bin info of bin named by b_id

Returns

true : succeed
false : fail

4.11 Hardware MAC APIs

Hardware MAC address APIs.

Enumerations

- enum mac_group { DEFAULT_MAC = 0, USER_MAC }
- enum mac_type { WIFI_MAC = 0, BT_MAC }

Functions

- int system_efuse_program_user_mac (mac_type type, uint8 ∗mac)
  Set user-define hardware MAC address.
- bool system_efuse_read_mac (mac_group group, mac_type type, uint8 ∗mac)
  Read hardware MAC address.
- bool system_efuse_set_mac_group (mac_group group)
  Set hardware MAC group, default MAC or user-defined MAC.
- mac_group system_efuse_get_mac_group (void)
  Get hardware MAC group, default MAC or user-defined MAC.

4.11.1 Detailed Description

Hardware MAC address APIs.

In WiFi MAC, only ESP32 station MAC is the hardware MAC, ESP32 softAP MAC is a software MAC calculated from ESP32 station MAC. So users need to call wifi_get_macaddr to query the ESP32 softAP MAC if ESP32 station MAC changed.

4.11.2 Enumeration Type Documentation

4.11.2.1 enum **mac_group**

Enumerator

DEFAULT_MAC	Default hardware MAC provided by Espressif Systems
USER_MAC	User-define hardware MAC

4.11.2.2 enum **mac_type**

Enumerator

WIFI_MAC	Hardware MAC address of ESP32 WiFi
BT_MAC	Hardware MAC address of ESP32 bluetooth

4.11.3 Function Documentation

4.11.3.1 **mac_group** system_efuse_get_mac_group (void)

Get hardware MAC group, default MAC or user-defined MAC.

Parameters

null	

Returns

mac_group, the hardware MAC group.

4.11.3.2 int system_efuse_program_user_mac (mac_type *type,* uint8 ∗ *mac*)

Set user-define hardware MAC address.

Attention

Hardware MAC address can only be set ONCE for each ESP32 chip.

Parameters

mac_type	type : type of hardware MAC address.
uint8	∗mac : user-define hardware MAC address, length: 6 bytes.

Returns

0 : succeed to set.
1 : the hardware MAC has been set once, users can not set it any more.
2 : fail to set.
3 : invalid parameter.

4.11.3.3 bool system_efuse_read_mac (mac_group *group,* mac_type *type,* uint8 ∗ *mac*)

Read hardware MAC address.

Parameters

mac_group	group : default MAC or user-defined MAC.
mac_type	type : type of hardware MAC address.
uint8	∗mac : the hardware MAC address, length: 6 bytes.

Returns

true : succeed
false : fail

4.11.3.4 bool system_efuse_set_mac_group (mac_group *group*)

Set hardware MAC group, default MAC or user-defined MAC.

Attention

This API needs system_restart to take effect.

Parameters

mac_group	group : default MAC or user-defined MAC.

Returns

true : succeed
false : fail

4.12 Software timer APIs

Software timer APIs.

Functions

- void os_timer_setfn (os_timer_t ∗ptimer, os_timer_func_t ∗pfunction, void ∗parg)

  Set the timer callback function.
- void os_timer_arm (os_timer_t ∗ptimer, uint32 msec, bool repeat_flag)

  Enable the millisecond timer.
- void os_timer_disarm (os_timer_t ∗ptimer)

  Disarm the timer.

4.12.1 Detailed Description

Software timer APIs.

Timers of the following interfaces are software timers. Functions of the timers are executed during the tasks. Since a task can be stopped, or be delayed because there are other tasks with higher priorities, the following os_timer interfaces cannot guarantee the precise execution of the timers.

- For the same timer, os_timer_arm (or os_timer_arm_us) cannot be invoked repeatedly. os_timer_disarm should be invoked first.
- os_timer_setfn can only be invoked when the timer is not enabled, i.e., after os_timer_disarm or before os↩_timer_arm (or os_timer_arm_us).

4.12.2 Function Documentation

4.12.2.1 void os_timer_arm (**os_timer_t** ∗ *ptimer,* uint32 *msec,* bool *repeat_flag*)

Enable the millisecond timer.

Parameters

os_↩ timer_t	∗ptimer : timer structure
uint32_t	milliseconds : Timing, unit: millisecond, the maximum value allowed is 0x41893
bool	repeat_flag : Whether the timer will be invoked repeatedly or not

Returns

null

4.12.2.2 void os_timer_disarm (**os_timer_t** ∗ *ptimer*)

Disarm the timer.

Parameters

os_↩ timer_t	∗ptimer : Timer structure

Returns

null

4.12.2.3 void os_timer_setfn (os_timer_t ∗ *ptimer,* os_timer_func_t ∗ *pfunction,* void ∗ *parg*)

Set the timer callback function.

Attention

1. The callback function must be set in order to enable the timer.
2. Operating system scheduling is disabled in timer callback.

Parameters

os_timer_t	∗ptimer : Timer structure
os_timer_↩ func_t	∗pfunction : timer callback function
void	∗parg : callback function parameter

Returns

null

4.13 Common APIs

WiFi common APIs.

Data Structures

- struct ip_info
- struct Event_StaMode_ScanDone_t
- struct Event_StaMode_Connected_t
- struct Event_StaMode_Disconnected_t
- struct Event_StaMode_AuthMode_Change_t
- struct Event_StaMode_Got_IP_t
- struct Event_SoftAPMode_StaConnected_t
- struct Event_SoftAPMode_StaDisconnected_t
- struct Event_SoftAPMode_ProbeReqRecved_t
- union Event_Info_u
- struct _esp_event

Typedefs

- typedef struct _esp_event **System_Event_t**
- typedef void(∗ wifi_event_handler_cb_t) (System_Event_t ∗event)

    The Wi-Fi event handler.

Enumerations

- enum WIFI_MODE {
  NULL_MODE = 0, STATION_MODE, SOFTAP_MODE, STATIONAP_MODE,
  MAX_MODE }
- enum AUTH_MODE {
  AUTH_OPEN = 0, AUTH_WEP, AUTH_WPA_PSK, AUTH_WPA2_PSK,
  AUTH_WPA_WPA2_PSK, **AUTH_MAX** }
- enum WIFI_INTERFACE { STATION_IF = 0, SOFTAP_IF, **MAX_IF** }
- enum SYSTEM_EVENT {
  EVENT_STAMODE_SCAN_DONE = 0, EVENT_STAMODE_CONNECTED, EVENT_STAMODE_DISCO↩
  NNECTED, EVENT_STAMODE_AUTHMODE_CHANGE,
  EVENT_STAMODE_GOT_IP, EVENT_STAMODE_DHCP_TIMEOUT, EVENT_SOFTAPMODE_STACO↩
  NNECTED, EVENT_SOFTAPMODE_STADISCONNECTED,
  EVENT_SOFTAPMODE_PROBEREQRECVED, **EVENT_MAX** }
- enum {
  REASON_UNSPECIFIED = 1, **REASON_AUTH_EXPIRE** = 2, **REASON_AUTH_LEAVE** = 3, **REASON_↩
  ASSOC_EXPIRE** = 4,
  REASON_ASSOC_TOOMANY = 5, **REASON_NOT_AUTHED** = 6, **REASON_NOT_ASSOCED** = 7, **RE↩
  ASON_ASSOC_LEAVE** = 8,
  REASON_ASSOC_NOT_AUTHED = 9, **REASON_DISASSOC_PWRCAP_BAD** = 10, **REASON_DISAS↩
  SOC_SUPCHAN_BAD** = 11, **REASON_IE_INVALID** = 13,
  REASON_MIC_FAILURE = 14, **REASON_4WAY_HANDSHAKE_TIMEOUT** = 15, **REASON_GROUP_K↩
  EY_UPDATE_TIMEOUT** = 16, **REASON_IE_IN_4WAY_DIFFERS** = 17,
  REASON_GROUP_CIPHER_INVALID = 18, **REASON_PAIRWISE_CIPHER_INVALID** = 19, **REASON_↩
  AKMP_INVALID** = 20, **REASON_UNSUPP_RSN_IE_VERSION** = 21,
  REASON_INVALID_RSN_IE_CAP = 22, **REASON_802_1X_AUTH_FAILED** = 23, **REASON_CIPHER_S↩
  UITE_REJECTED** = 24, **REASON_BEACON_TIMEOUT** = 200,
  REASON_NO_AP_FOUND = 201, **REASON_AUTH_FAIL** = 202, **REASON_ASSOC_FAIL** = 203, **REAS↩
  ON_HANDSHAKE_TIMEOUT** = 204 }

Functions

- WIFI_MODE wifi_get_opmode (void)

  Get the current operating mode of the WiFi.
- WIFI_MODE wifi_get_opmode_default (void)

  Get the operating mode of the WiFi saved in the Flash.
- bool wifi_set_opmode (WIFI_MODE opmode)

  Set the WiFi operating mode, and save it to Flash.
- bool wifi_set_opmode_current (WIFI_MODE opmode)

  Set the WiFi operating mode, and will not save it to Flash.
- WIFI_INTERFACE wifi_get_interface (void ∗dev)

  Get the ESP32 WiFi interface (station or the soft-AP).
- bool wifi_get_ip_info (WIFI_INTERFACE if_index, struct ip_info ∗info)

  Get the IP address of the ESP32 WiFi station or the soft-AP interface.
- bool wifi_set_ip_info (WIFI_INTERFACE if_index, struct ip_info ∗info)

  Set the IP address of the ESP32 WiFi station or the soft-AP interface.
- bool wifi_get_macaddr (WIFI_INTERFACE if_index, uint8 ∗macaddr)

  Get MAC address of the ESP32 WiFi station or the soft-AP interface.
- bool wifi_set_macaddr (WIFI_INTERFACE if_index, uint8 ∗macaddr)

  Set MAC address of the ESP32 WiFi station or the soft-AP interface.
- bool wifi_set_event_handler_cb (wifi_event_handler_cb_t cb)

  Register the Wi-Fi event handler.

4.13.1 Detailed Description

WiFi common APIs.

The Flash system parameter area is the last 16KB of the Flash.

4.13.2 Typedef Documentation

4.13.2.1 typedef void(∗ wifi_event_handler_cb_t) (System_Event_t ∗event)

The Wi-Fi event handler.

Attention

No complex operations are allowed in callback. If users want to execute any complex operations, please post message to another task instead.

Parameters

System_↩ Event_t	∗event : WiFi event

Returns

null

4.13.3 Enumeration Type Documentation

4.13.3.1 enum AUTH_MODE

Enumerator

AUTH_OPEN	authenticate mode : open
AUTH_WEP	authenticate mode : WEP
AUTH_WPA_PSK	authenticate mode : WPA_PSK
AUTH_WPA2_PSK	authenticate mode : WPA2_PSK
AUTH_WPA_WPA2_PSK	authenticate mode : WPA_WPA2_PSK

4.13.3.2 enum SYSTEM_EVENT

Enumerator

EVENT_STAMODE_SCAN_DONE	ESP32 station finish scanning AP
EVENT_STAMODE_CONNECTED	ESP32 station connected to AP
EVENT_STAMODE_DISCONNECTED	ESP32 station disconnected to AP
EVENT_STAMODE_AUTHMODE_CHANGE	the auth mode of AP connected by ESP32 station changed
EVENT_STAMODE_GOT_IP	ESP32 station got IP from connected AP
EVENT_STAMODE_DHCP_TIMEOUT	ESP32 station dhcp client got IP timeout
EVENT_SOFTAPMODE_STACONNECTED	a station connected to ESP32 soft-AP
EVENT_SOFTAPMODE_STADISCONNECTED	a station disconnected to ESP32 soft-AP
EVENT_SOFTAPMODE_PROBEREQRECVED	Receive probe request packet in soft-AP interface

4.13.3.3 enum WIFI_INTERFACE

Enumerator

STATION_IF	ESP32 station interface
SOFTAP_IF	ESP32 soft-AP interface

4.13.3.4 enum WIFI_MODE

Enumerator

NULL_MODE	null mode
STATION_MODE	WiFi station mode
SOFTAP_MODE	WiFi soft-AP mode
STATIONAP_MODE	WiFi station + soft-AP mode

4.13.4 Function Documentation

4.13.4.1 **WIFI_INTERFACE** wifi_get_interface (void ∗ *dev*)

Get the ESP32 WiFi interface (station or the soft-AP).

Returns

WIFI_INTERFACE

4.13.4.2 bool wifi_get_ip_info (**WIFI_INTERFACE** *if_index,* struct **ip_info** ∗ *info*)

Get the IP address of the ESP32 WiFi station or the soft-AP interface.

Attention

Users need to enable the target interface (station or soft-AP) by wifi_set_opmode first.

Parameters

WIFI_INTERFACE	if_index : get the IP address of the station or the soft-AP interface, 0x00 for STATION_IF, 0x01 for SOFTAP_IF.
struct	ip_info ∗info : the IP information obtained.

Returns

true : succeed
false : fail

4.13.4.3 bool wifi_get_macaddr (**WIFI_INTERFACE** *if_index,* uint8 ∗ *macaddr*)

Get MAC address of the ESP32 WiFi station or the soft-AP interface.

Parameters

WIFI_INTERFACE	if_index : get the IP address of the station or the soft-AP interface, 0x00 for STATION_IF, 0x01 for SOFTAP_IF.
uint8	∗macaddr : the MAC address.

Returns

true : succeed
false : fail

4.13.4.4 **WIFI_MODE** wifi_get_opmode (void)

Get the current operating mode of the WiFi.

Parameters

null	

Returns

WiFi operating modes:

- 0x01: station mode;
- 0x02: soft-AP mode
- 0x03: station+soft-AP mode

4.13.4.5 WIFI_MODE wifi_get_opmode_default (void)

Get the operating mode of the WiFi saved in the Flash.

Parameters

null	

Returns

WiFi operating modes:

- 0x01: station mode;
- 0x02: soft-AP mode
- 0x03: station+soft-AP mode

4.13.4.6 bool wifi_set_event_handler_cb (wifi_event_handler_cb_t *cb*)

Register the Wi-Fi event handler.

Parameters

wifi_event_handler_↩ cb_t	cb : callback function

Returns

true : succeed
false : fail

4.13.4.7 bool wifi_set_ip_info (WIFI_INTERFACE *if_index,* struct ip_info ∗ *info*)

Set the IP address of the ESP32 WiFi station or the soft-AP interface.

Attention

1. Users need to enable the target interface (station or soft-AP) by wifi_set_opmode first.
2. To set static IP, users need to disable DHCP first (wifi_station_dhcpc_stop or wifi_softap_dhcps_stop):
   - If the DHCP is enabled, the static IP will be disabled; if the static IP is enabled, the DHCP will be disabled. It depends on the latest configuration.

Parameters

WIFI_INTERFACE	if_index : get the IP address of the station or the soft-AP interface, 0x00 for STATION_IF, 0x01 for SOFTAP_IF.
struct	ip_info ∗info : the IP information obtained.

Returns

true : succeed
false : fail

4.13.4.8 bool wifi_set_macaddr (WIFI_INTERFACE *if_index,* uint8 ∗ *macaddr*)

Set MAC address of the ESP32 WiFi station or the soft-AP interface.

Attention

1. This API can only be called in user_init.
2. Users need to enable the target interface (station or soft-AP) by wifi_set_opmode first.
3. ESP32 soft-AP and station have different MAC addresses, do not set them to be the same.
   - The bit0 of the first byte of ESP32 MAC address can not be 1. For example, the MAC address can set to be "1a:XX:XX:XX:XX:XX", but can not be "15:XX:XX:XX:XX:XX".

Parameters

WIFI_INTERFACE	if_index : get the IP address of the station or the soft-AP interface, 0x00 for STATION_IF, 0x01 for SOFTAP_IF.
uint8	∗macaddr : the MAC address.

Returns

true : succeed
false : fail

4.13.4.9 bool wifi_set_opmode (WIFI_MODE *opmode*)

Set the WiFi operating mode, and save it to Flash.

Set the WiFi operating mode as station, soft-AP or station+soft-AP, and save it to Flash. The default mode is soft-AP mode.

Attention

This configuration will be saved in the Flash system parameter area if changed.

Parameters

uint8	opmode : WiFi operating modes: • 0x01: station mode; • 0x02: soft-AP mode • 0x03: station+soft-AP mode

Returns

true : succeed
false : fail

4.13.4.10 bool wifi_set_opmode_current (WIFI_MODE *opmode*)

Set the WiFi operating mode, and will not save it to Flash.

Set the WiFi operating mode as station, soft-AP or station+soft-AP, and the mode won't be saved to the Flash.

Parameters

uint8	opmode : WiFi operating modes: • 0x01: station mode; • 0x02: soft-AP mode • 0x03: station+soft-AP mode

Returns

true : succeed
false : fail

4.14 Sniffer APIs

WiFi sniffer APIs.

Typedefs

- typedef void(∗ wifi_promiscuous_cb_t) (uint8 ∗buf, uint16 len)

    The RX callback function in the promiscuous mode.

Functions

- void wifi_set_promiscuous_rx_cb (wifi_promiscuous_cb_t cb)

    Register the RX callback function in the promiscuous mode.
- uint8 wifi_get_channel (void)

    Get the channel number for sniffer functions.
- bool wifi_set_channel (uint8 channel)

    Set the channel number for sniffer functions.
- void wifi_promiscuous_enable (uint8 promiscuous)

    Enable the promiscuous mode.

4.14.1 Detailed Description

WiFi sniffer APIs.

4.14.2 Typedef Documentation

4.14.2.1 typedef void(∗ wifi_promiscuous_cb_t) (uint8 ∗buf, uint16 len)

The RX callback function in the promiscuous mode.

Each time a packet is received, the callback function will be called.

Parameters

uint8	∗buf : the data received
uint16	len : data length

Returns

null

4.14.3 Function Documentation

4.14.3.1 uint8 wifi_get_channel (void)

Get the channel number for sniffer functions.

Parameters

null	

Returns

channel number

4.14.3.2 void wifi_promiscuous_enable (uint8 *promiscuous*)

Enable the promiscuous mode.

Attention

1. The promiscuous mode can only be enabled in the ESP32 station mode.
2. When in the promiscuous mode, the ESP32 station and soft-AP are disabled.
3. Call wifi_station_disconnect to disconnect before enabling the promiscuous mode.
4. Don't call any other APIs when in the promiscuous mode. Call wifi_promiscuous_enable(0) to quit sniffer before calling other APIs.

Parameters

uint8	promiscuous : • 0: to disable the promiscuous mode • 1: to enable the promiscuous mode

Returns

null

4.14.3.3 bool wifi_set_channel (uint8 *channel*)

Set the channel number for sniffer functions.

Parameters

uint8	channel : channel number

Returns

true : succeed
false : fail

4.14.3.4 void wifi_set_promiscuous_rx_cb (wifi_promiscuous_cb_t *cb*)

Register the RX callback function in the promiscuous mode.

Each time a packet is received, the registered callback function will be called.

Parameters

wifi_promiscuous_↩ cb_t	cb : callback

Returns

null

4.15 WPS APIs

ESP32 WPS APIs.

Typedefs

- typedef enum wps_type **WPS_TYPE_t**
- typedef void(∗ wps_st_cb_t) (int status)

  WPS callback.

Enumerations

- enum **wps_type** {
  WPS_TYPE_DISABLE = 0, **WPS_TYPE_PBC**, **WPS_TYPE_PIN**, **WPS_TYPE_DISPLAY**,
  WPS_TYPE_MAX }
- enum wps_cb_status {
  WPS_CB_ST_SUCCESS = 0, WPS_CB_ST_FAILED, WPS_CB_ST_TIMEOUT, WPS_CB_ST_WEP,
  WPS_CB_ST_SCAN_ERR }

Functions

- bool wifi_wps_enable (WPS_TYPE_t wps_type)

  Enable Wi-Fi WPS function.
- bool wifi_wps_disable (void)

  Disable Wi-Fi WPS function and release resource it taken.
- bool wifi_wps_start (void)

  WPS starts to work.
- bool wifi_set_wps_cb (wps_st_cb_t cb)

  Set WPS callback.

4.15.1 Detailed Description

ESP32 WPS APIs.

WPS can only be used when ESP32 station is enabled.

4.15.2 Typedef Documentation

4.15.2.1 typedef void(∗ wps_st_cb_t) (int status)

WPS callback.

Parameters

int	status : status of WPS, enum wps_cb_status. • If parameter status == WPS_CB_ST_SUCCESS in WPS callback, it means WPS got AP's information, user can call wifi_wps_disable to disable WPS and release resource, then call wifi_station_connect to connect to target AP. • Otherwise, it means that WPS fail, user can create a timer to retry WPS by wifi_wps_start after a while, or call wifi_wps_disable to disable WPS and release resource.

Returns

null

4.15.3 Enumeration Type Documentation

4.15.3.1 enum wps_cb_status

Enumerator

WPS_CB_ST_SUCCESS	WPS succeed
WPS_CB_ST_FAILED	WPS fail
WPS_CB_ST_TIMEOUT	WPS timeout, fail
WPS_CB_ST_WEP	WPS failed because that WEP is not supported
WPS_CB_ST_SCAN_ERR	can not find the target WPS AP

4.15.4 Function Documentation

4.15.4.1 bool wifi_set_wps_cb (wps_st_cb_t *cb*)

Set WPS callback.

Attention

WPS can only be used when ESP32 station is enabled.

Parameters

wps_st_↩ cb_t	cb : callback.

Returns

true : WPS starts to work successfully, but does not mean WPS succeed.
false : fail

4.15.4.2 bool wifi_wps_disable (void)

Disable Wi-Fi WPS function and release resource it taken.

Parameters

null	

Returns

true : succeed
false : fail

4.15.4.3 bool wifi_wps_enable (WPS_TYPE_t *wps_type*)

Enable Wi-Fi WPS function.

Attention

WPS can only be used when ESP32 station is enabled.

Parameters

WPS_TYP↩ E_t	wps_type : WPS type, so far only WPS_TYPE_PBC is supported

Returns

true : succeed
false : fail

4.15.4.4 bool wifi_wps_start (void)

WPS starts to work.

Attention

WPS can only be used when ESP32 station is enabled.

Parameters

null	

Returns

true : WPS starts to work successfully, but does not mean WPS succeed.
false : fail

4.16 Smartconfig APIs

SmartConfig APIs.

Typedefs

- typedef void(∗ sc_callback_t) (sc_status status, void ∗pdata)

  The callback of SmartConfig, executed when smart-config status changed.

Enumerations

- enum sc_status {
  SC_STATUS_WAIT = 0, SC_STATUS_FIND_CHANNEL, SC_STATUS_GETTING_SSID_PSWD, SC_S←
  TATUS_LINK,
  SC_STATUS_LINK_OVER }
- enum sc_type { SC_TYPE_ESPTOUCH = 0, SC_TYPE_AIRKISS, SC_TYPE_ESPTOUCH_AIRKISS }

Functions

- const char ∗ smartconfig_get_version (void)

  Get the version of SmartConfig.
- bool smartconfig_start (sc_callback_t cb,...)

  Start SmartConfig mode.
- bool smartconfig_stop (void)

  Stop SmartConfig, free the buffer taken by smartconfig_start.
- bool esptouch_set_timeout (uint8 time_s)

  Set timeout of SmartConfig.
- bool smartconfig_set_type (sc_type type)

  Set protocol type of SmartConfig.

4.16.1 Detailed Description

SmartConfig APIs.

SmartConfig can only be enabled in station only mode. Please make sure the target AP is enabled before enable SmartConfig.

4.16.2 Typedef Documentation

4.16.2.1 typedef void(∗ sc_callback_t) (sc_status status, void ∗pdata)

The callback of SmartConfig, executed when smart-config status changed.

Parameters

sc_status	status : status of SmartConfig: • if status == SC_STATUS_GETTING_SSID_PSWD, parameter void ∗pdata is a pointer of sc_type, means SmartConfig type: AirKiss or ESP-TOUCH. • if status == SC_STATUS_LINK, parameter void ∗pdata is a pointer of struct station_config; • if status == SC_STATUS_LINK_OVER, parameter void ∗pdata is a pointer of mobile phone's IP address, 4 bytes. This is only available in ESPTOUCH, otherwise, it is NULL. • otherwise, parameter void ∗pdata is NULL.
void	∗pdata : data of SmartConfig

Returns

null

4.16.3 Enumeration Type Documentation

4.16.3.1 enum sc_status

Enumerator

SC_STATUS_WAIT	waiting, do not start connection in this phase
SC_STATUS_FIND_CHANNEL	find target channel, start connection by APP in this phase
SC_STATUS_GETTING_SSID_PSWD	getting SSID and password of target AP
SC_STATUS_LINK	connecting to target AP
SC_STATUS_LINK_OVER	got IP, connect to AP successfully

4.16.3.2 enum sc_type

Enumerator

SC_TYPE_ESPTOUCH	protocol: ESPTouch
SC_TYPE_AIRKISS	protocol: AirKiss
SC_TYPE_ESPTOUCH_AIRKISS	protocol: ESPTouch and AirKiss

4.16.4 Function Documentation

4.16.4.1 bool esptouch_set_timeout (uint8 *time_s*)

Set timeout of SmartConfig.

Attention

SmartConfig timeout start at SC_STATUS_FIND_CHANNEL, SmartConfig will restart if timeout.

Parameters

uint8	time_s : range 15s∼255s, offset:45s.

Returns

true : succeed
false : fail

4.16.4.2 const char∗ smartconfig_get_version (void)

Get the version of SmartConfig.

Parameters

null	

Returns

SmartConfig version

4.16.4.3 bool smartconfig_set_type (sc_type *type*)

Set protocol type of SmartConfig.

Attention

If users need to set the SmartConfig type, please set it before calling smartconfig_start.

Parameters

sc_type	type : AirKiss, ESP-TOUCH or both.

Returns

true : succeed
false : fail

4.16.4.4 bool smartconfig_start (sc_callback_t *cb,* ...)

Start SmartConfig mode.

Start SmartConfig mode, to connect ESP32 station to AP, by sniffing for special packets from the air, containing SSID and password of desired AP. You need to broadcast the SSID and password (e.g. from mobile device or computer) with the SSID and password encoded.

Attention

1. This api can only be called in station mode.
2. During SmartConfig, ESP32 station and soft-AP are disabled.
3. Can not call smartconfig_start twice before it finish, please call smartconfig_stop first.
4. Don't call any other APIs during SmartConfig, please call smartconfig_stop first.

Parameters

sc_↩ callback_t	cb : SmartConfig callback; executed when SmartConfig status changed;
uint8	log : 1, UART output logs; otherwise, UART only outputs the result.

Returns

true : succeed
false : fail

4.16.4.5 bool smartconfig_stop (void)

Stop SmartConfig, free the buffer taken by smartconfig_start.

Attention

Whether connect to AP succeed or not, this API should be called to free memory taken by smartconfig_start.

Parameters

null	

Returns

true : succeed
false : fail

4.17 Driver APIs

Driver APIs.

Modules

- SPI Driver APIs

  SPI Flash APIs.
- GPIO Driver APIs

  GPIO APIs.
- I2S Driver APIs

  I2S driver APIs.
- PWM Driver APIs

  PWM driver APIs.
- UART Driver APIs

  UART driver APIs.

4.17.1 Detailed Description

Driver APIs.

4.18 SPI Driver APIs

SPI Flash APIs.

Macros

- #define SPI_FLASH_SEC_SIZE 4096

Enumerations

- enum SpiFlashOpResult { SPI_FLASH_RESULT_OK, SPI_FLASH_RESULT_ERR, SPI_FLASH_RESUL↩T_TIMEOUT }

Functions

- SpiFlashOpResult spi_flash_erase_sector (uint16 sec)

  Erase the Flash sector.
- SpiFlashOpResult spi_flash_write (uint32 des_addr, uint32 ∗src_addr, uint32 size)

  Write data to Flash.
- SpiFlashOpResult spi_flash_read (uint32 src_addr, uint32 ∗des_addr, uint32 size)

  Read data from Flash.

4.18.1 Detailed Description

SPI Flash APIs.

4.18.2 Macro Definition Documentation

4.18.2.1 #define SPI_FLASH_SEC_SIZE 4096

SPI Flash sector size

4.18.3 Enumeration Type Documentation

4.18.3.1 enum SpiFlashOpResult

Enumerator

SPI_FLASH_RESULT_OK	SPI Flash operating OK
SPI_FLASH_RESULT_ERR	SPI Flash operating fail
SPI_FLASH_RESULT_TIMEOUT	SPI Flash operating time out

4.18.4 Function Documentation

4.18.4.1 SpiFlashOpResult spi_flash_erase_sector (uint16 *sec*)

Erase the Flash sector.

Parameters

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

Returns

SpiFlashOpResult

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

Read data from Flash.

Parameters

uint32	src_addr : source address of the data.
uint32	∗des_addr : destination address in Flash.
uint32	size : length of data

Returns

SpiFlashOpResult

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

Write data to Flash.

Parameters

uint32	des_addr : destination address in Flash.
uint32	∗src_addr : source address of the data.
uint32	size : length of data

Returns

SpiFlashOpResult

4.19 OTA APIs

OTA APIs.

Data Structures

- struct remote_bin_info
- struct server_info
- struct upgrade_info

Macros

- #define **SPI_FLASH_SEC_SIZE** 4096

Typedefs

- typedef void(∗ upgrade_states_check_callback) (void ∗arg)

  upgrade check call function type

Enumerations

- enum ota_error_id {
  UPGRADE_OK, UPGRADE_FLAG_ERROR = 10000, NO_STATION_IP_ERROR = 10001, UPGRADE_←
  MEM_ERROR = 10002,
  CREATE_SOCKET_ERROR = 10003, SEND_QUEUE_ERROR = 10004, SERVER_CONNECT_ERROR =
  10005, SEND_URL_ERROR = 10006,
  HTTP_HEAD_ERROR = 10007, DOWNLOAD_TIMEOUT_ERROR = 10008, GET_BIN_LENGTH_ERROR
  = 10009, ERASE_FLASH_ERROR = 10010,
  RECV_DATA_ERROR = 10011, BIN_MAGIC_ERROR = 10012, USER_ID_CONFLICT_ERROR = 10013,
  FLASH_ID_CONFLICT_ERROR = 10014,
  CRC_CHECK_FAILED_ERROR = 10015 }
- enum { NO_READY = 1, TENTATIVE, READY }

Functions

- bool system_upgrade_start (struct upgrade_info ∗server)

  start upgrade progress
- bool system_upgrade_init (uint8 b_id, uint8 start_flash_id)

  init upgrade progress
- void system_upgrade_deinit (void)

  deinit upgrade progress
- enum ota_error_id system_upgrade_get_error_id (void)

  get error id of upgrade progress
- uint8 upgrade_get_process_rate (void)

  get download rate process
- bool upgrade_get_remote_bin_info (struct server_info ∗s_if, struct remote_bin_info ∗rb_if)

  get bin's sum length and lenth of irom1 part in remote server

4.19.1 Detailed Description

OTA APIs.

4.19.2 Typedef Documentation

4.19.2.1 typedef void(∗ upgrade_states_check_callback) (void ∗arg)

upgrade check call function type

Parameters

void	∗ arg : call back parameter

Returns

void

4.19.3 Enumeration Type Documentation

4.19.3.1 anonymous enum

Enumerator

NO_READY bin file is broken or not a correct bin file

TENTATIVE bin file has been downloaded in flash but never run

READY bin is runing or has run before

4.19.3.2 enum ota_error_id

Enumerator

UPGRADE_OK OTA succeed

UPGRADE_FLAG_ERROR OTA is in progress, can not start it again

NO_STATION_IP_ERROR ESP32 station does not get IP address

UPGRADE_MEM_ERROR fail to alloc memory, maybe NULL pointer, or out of memory

CREATE_SOCKET_ERROR fail to create socket

SEND_QUEUE_ERROR fail to send message into queue

SERVER_CONNECT_ERROR fail to connect to the OTA server

SEND_URL_ERROR fail to send HTTP request

HTTP_HEAD_ERROR can not parse the HTTP response from OTA server

DOWNLOAD_TIMEOUT_ERROR OTA time out

GET_BIN_LENGTH_ERROR fail to get the length of the OTA bin file

ERASE_FLASH_ERROR fail to erase flash

RECV_DATA_ERROR fail to receive the OTA bin file

BIN_MAGIC_ERROR the OTA bin file's magic check fail, invalid bin file

USER_ID_CONFLICT_ERROR the bin ID is using, can not set the same bin ID when calling system_↩ upgrade_init

FLASH_ID_CONFLICT_ERROR the new OTA bin will overlap with the current bin, so the downloading is rejected

CRC_CHECK_FAILED_ERROR the new OTA bin's CRC check fail

4.19.4 Function Documentation

4.19.4.1 void system_upgrade_deinit (void)

deinit upgrade progress

Attention

this API should be called in upgrade check call back function or called after it

Parameters

void	

Returns

void

4.19.4.2 enum ota_error_id system_upgrade_get_error_id (void)

get error id of upgrade progress

Parameters

void	

Returns

enum ota_error_id : explanation in enum ota_error_id

4.19.4.3 bool system_upgrade_init (uint8 *b_id,* uint8 *start_flash_id*)

init upgrade progress

Parameters

uint8	b_id : b_id number, must $<$ 5
uint8	start_flash_id : define a block in flash is 256KB, a block correspond a flash id start flash id means bin file's start flash id

Returns

true : succeed
false : fail

4.19.4.4 bool system_upgrade_start (struct upgrade_info ∗ *server*)

start upgrade progress

Attention

call this API should init server param firstly

Parameters

struct	upgrade_info ∗server : upgrade info contains remote server and callback func

Returns

true : succeed
false : fail

4.19.4.5 uint8 upgrade_get_process_rate (void)

get download rate process

Parameters

void	

Returns

uint8 : return x means x%

4.19.4.6 bool upgrade_get_remote_bin_info (struct server_info ∗ *s_if,* struct remote_bin_info ∗ *rb_if*)

get bin's sum length and lenth of irom1 part in remote server

Parameters

struct	server_info ∗s_if :remote server info
struct	remote_bin_info ∗rb_if : point to remote bin's info

Returns

true : succeed
false : fail

4.20 GPIO Driver APIs

GPIO APIs.

Macros

- #define GPIO_OUTPUT_SET(gpio_no, bit_value)

  Set GPIO pin output level.
- #define GPIO_OUTPUT(gpio_bits, bit_value)

  Set GPIO pin output level,This function only config GPIO0-GPIO31 .
- #define GPIO_OUTPUT_HIGH(gpio_bits, bit_value)

  Set GPIO pin output level,This function only config GPIO32-GPIO39.
- #define GPIO_DIS_OUTPUT(gpio_no)

  Disable GPIO pin output.
- #define GPIO_AS_INPUT(gpio_bits) gpio_output_conf(0, 0, 0, gpio_bits)

  Enable GPIO pin intput,This function only config GPIO0-GPIO31.
- #define GPIO_AS_INPUT_HIGH(gpio_bits) gpio_output_conf_high(0, 0, 0, gpio_bits)

  Enable GPIO pin intput,This function only config GPIO32-GPIO39.
- #define GPIO_AS_OUTPUT(gpio_bits) gpio_output_conf(0, 0, gpio_bits, 0)

  Enable GPIO pin output,This function only config GPIO0-GPIO31.
- #define GPIO_AS_OUTPUT_HIGH(gpio_bits) gpio_output_conf_high(0, 0, gpio_bits, 0)

  Enable GPIO pin output,This function only config GPIO32-GPIO39.
- #define GPIO_INPUT_GET(gpio_no)

  Sample the level of GPIO input.

Functions

- void gpio_config (GPIO_ConfigTypeDef ∗pGPIOConfig)

  GPIO init .
- void gpio_output_conf (uint32 set_mask, uint32 clear_mask, uint32 enable_mask, uint32 disable_mask)

  Configure GPIO pins out or input.
- void gpio_output_conf_high (uint32 set_mask, uint32 clear_mask, uint32 enable_mask, uint32 disable_mask)

  Configure GPIO pins out or input.
- void gpio_intr_handler_register (void ∗fn, void ∗arg)

  Register an application-specific interrupt handler for GPIO pin interrupts.
- void gpio_pin_wakeup_enable (uint32 i, GPIO_INT_TYPE intr_state)

  Configure GPIO wake up to light sleep,Only level way is effective.
- void gpio_pin_wakeup_disable (void)

  Disable GPIO wake up to light sleep.
- void gpio_pin_intr_state_set (uint32 i, GPIO_INT_TYPE intr_state)

  Config interrupt types of GPIO pin.
- uint32 gpio_input_get (void)

  Sample the value of GPIO input pins and returns a bitmask. This function only get the level GPIO0-GPIO31.
- uint32 gpio_input_get_high (void)

  Sample the value of GPIO input pins and returns a bitmask. This function only get the level GPIO32-GPIO39.
- void gpio_output_sigmadelta_enable (uint32 gpio_num, uint32 sigma_num, uint32 prescale)

  Enable GPIO sigmadelta function.
- void gpio_output_sigmadelta_disable (uint32 gpio_num)

  Disable GPIO sigmadelta function.

- void gpio_intr_config (uint32 gpio_num, uint32 intr_num, GPIO_INT_TYPE intr_type)

  Configure GPIO interrupr.
- void gpio_intr_process (void)

  The GPIO interrupt function.
- void gpio_matrix_in (uint32 gpio, uint32 signal_idx)

  To bind GPIO input and a certain road input signal.
- void gpio_matrix_out (uint32 gpio, uint32 signal_idx)

  To bind GPIO ouput and a certain road output signal.
- void intr_matrix_set (uint32 model_num, uint32 intr_num)

  To bind mode interrupt and interrupt sequence number.

4.20.1 Detailed Description

GPIO APIs.

4.20.2 Macro Definition Documentation

4.20.2.1 #define GPIO_AS_INPUT(*gpio_bits*) gpio_output_conf(0, 0, 0, gpio_bits)

Enable GPIO pin intput,This function only config GPIO0-GPIO31.

Parameters

gpio_bits	: The GPIO bit number.

Returns

null

4.20.2.2 #define GPIO_AS_INPUT_HIGH(*gpio_bits*) gpio_output_conf_high(0, 0, 0, gpio_bits)

Enable GPIO pin intput,This function only config GPIO32-GPIO39.

Parameters

gpio_bits	: The GPIO bit number.

Returns

null

4.20.2.3 #define GPIO_AS_OUTPUT(*gpio_bits*) gpio_output_conf(0, 0, gpio_bits, 0)

Enable GPIO pin output,This function only config GPIO0-GPIO31.

Parameters

gpio_bits	: The GPIO bit number.

Returns

null

4.20.2.4 #define GPIO_AS_OUTPUT_HIGH(*gpio_bits*) gpio_output_conf_high(0, 0, gpio_bits, 0)

Enable GPIO pin output,This function only config GPIO32-GPIO39.

Parameters

gpio_bits	: The GPIO bit number.

Returns

null

4.20.2.5 #define GPIO_DIS_OUTPUT(*gpio_no*)

Value:

```
((gpio_no < 32) ? \
    gpio_output_conf(0, 0, 0, 1 << gpio_no) :
      gpio_output_conf_high(0, 0, 0, 1 << gpio_no))
```

Disable GPIO pin output.

Parameters

gpio_no	: The GPIO sequence number.

Returns

null

4.20.2.6 #define GPIO_INPUT_GET(*gpio_no*)

Value:

```
((gpio_no < 32) ? \
    ((gpio_input_get() >> gpio_no) & BIT0) : ((gpio_input_get_high() >> (
      gpio_no - 32)) & BIT0))
```

Sample the level of GPIO input.

Parameters

gpio_no	: The GPIO sequence number.

Returns

the level of GPIO input

4.20.2.7 #define GPIO_OUTPUT(*gpio_bits,* *bit_value*)

Value:

```
if(bit_value) gpio_output_conf(gpio_bits, 0, gpio_bits, 0);\
    else gpio_output_conf(0, gpio_bits, gpio_bits, 0)
```

Set GPIO pin output level,This function only config GPIO0-GPIO31 .

Parameters

gpio_bits	: The GPIO bit number.
bit_value	: GPIO pin output level.

Returns

null

4.20.2.8 #define GPIO_OUTPUT_HIGH(*gpio_bits,* *bit_value*)

Value:

```
if(bit_value) gpio_output_conf_high(gpio_bits, 0, gpio_bits, 0);\
    else gpio_output_conf_high(0, gpio_bits, gpio_bits, 0)
```

Set GPIO pin output level,This function only config GPIO32-GPIO39.

Parameters

gpio_bits	: The GPIO bit number.
bit_value	: GPIO pin output level.

Returns

null

4.20.2.9 #define GPIO_OUTPUT_SET(*gpio_no,* *bit_value*)

Value:

```
((gpio_no < 32) ? gpio_output_conf(bit_value << gpio_no, (bit_value ? 0 : 1) << gpio_no, 1
      << gpio_no, 0) : \
         gpio_output_conf_high(bit_value << (gpio_no - 32), (bit_value ? 0 : 1) << (
      gpio_no - 32), 1 << (gpio_no - 32),0))
```

Set GPIO pin output level.

Parameters

gpio_no	: The GPIO sequence number.
bit_value	: GPIO pin output level.

Returns

null

4.20.3 Function Documentation

4.20.3.1 void gpio_config (GPIO_ConfigTypeDef ∗ *pGPIOConfig*)

GPIO init .

Parameters

pGPIOConfig	: through this structure initialization GPIO.

Returns

null

4.20.3.2 uint32 gpio_input_get (void)

Sample the value of GPIO input pins and returns a bitmask. This function only get the level GPIO0-GPIO31.

Parameters

null	

Returns

bitmask of GPIO pins input

4.20.3.3 uint32 gpio_input_get_high (void)

Sample the value of GPIO input pins and returns a bitmask. This function only get the level GPIO32-GPIO39.

Parameters

null	

Returns

bitmask of GPIO pins input

4.20.3.4 void gpio_intr_config (uint32 *gpio_num,* uint32 *intr_num,* GPIO_INT_TYPE *intr_type*)

Configure GPIO interrupr.

Parameters

uint32	gpio_num : The GPIO sequence number.
uint32	intr_num : the interrupt source sequence number 0-7.
GPIO_INT_TYPE	intr_type : The type of interrupt.

Returns

null

4.20.3.5 void gpio_intr_handler_register (void ∗ *fn,* void ∗ *arg*)

Register an application-specific interrupt handler for GPIO pin interrupts.

Parameters

void	∗fn : interrupt handler for GPIO pin interrupts.
void	∗arg : interrupt handler's arg

Returns

null

4.20.3.6 void gpio_intr_process (void)

The GPIO interrupt function.

Parameters

null	

Returns

null

4.20.3.7 void gpio_matrix_in (uint32 *gpio,* uint32 *signal_idx*)

To bind GPIO input and a certain road input signal.

Parameters

uint32	gpio_num : The GPIO sequence number.
uint32	signal_idx : input signal sequence number.

Returns

null

4.20.3.8 void gpio_matrix_out (uint32 *gpio,* uint32 *signal_idx*)

To bind GPIO ouput and a certain road output signal.

Parameters

uint32	gpio_num : The GPIO sequence number.
uint32	signal_idx : out signal sequence number.

Returns

null

4.20.3.9 void gpio_output_conf (uint32 *set_mask,* uint32 *clear_mask,* uint32 *enable_mask,* uint32 *disable_mask*)

Configure GPIO pins out or input.

Parameters

uint32	set_mask : Set the output for the high bit, the corresponding bit is 1, the output of high, the corresponding bit is 0, do not change the state.
uint32	set_mask : Set the output for the high bit, the corresponding bit is 1, the output of low, the corresponding bit is 0, do not change the state.
uint32	enable_mask : Enable Output
uint32	disable_mask : Enable Input

Returns

null

4.20.3.10 void gpio_output_conf_high (uint32 *set_mask,* uint32 *clear_mask,* uint32 *enable_mask,* uint32 *disable_mask*)

Configure GPIO pins out or input.

Parameters

uint32	set_mask : Set the output for the high bit, the corresponding bit is 1, the output of high, the corresponding bit is 0, do not change the state.
uint32	set_mask : Set the output for the high bit, the corresponding bit is 1, the output of low, the corresponding bit is 0, do not change the state.
uint32	enable_mask : Enable Output
uint32	disable_mask : Enable Input

Returns

null

4.20.3.11 void gpio_output_sigmadelta_disable (uint32 *gpio_num*)

Disable GPIO sigmadelta function.

Parameters

uint32	gpio_num : The GPIO sequence number

Returns

null

4.20.3.12 void gpio_output_sigmadelta_enable (uint32 *gpio_num,* uint32 *sigma_num,* uint32 *prescale*)

Enable GPIO sigmadelta function.

Parameters

uint32	gpio_num : The GPIO sequence number.
uint32	sigma_num : the sigmadelta source sequence number 0-7.
uint32	prescale : Clock divide factor.

Returns

null

4.20.3.13 void gpio_pin_intr_state_set (uint32 *i*, GPIO_INT_TYPE *intr_state*)

Config interrupt types of GPIO pin.

Parameters

uint32	i : The GPIO sequence number.
GPIO_INT_TYPE	intr_state : GPIO interrupt types.

Returns

null

4.20.3.14 void gpio_pin_wakeup_disable (void)

Disable GPIO wake up to light sleep.

Parameters

null	

Returns

null

4.20.3.15 void gpio_pin_wakeup_enable (uint32 *i*, GPIO_INT_TYPE *intr_state*)

Configure GPIO wake up to light sleep,Only level way is effective.

Parameters

uint32	i : GPIO sequence number
GPIO_INT_TYPE	intr_state : the level of wake up to light sleep

Returns

null

4.20.3.16 void intr_matrix_set (uint32 *model_num*, uint32 *intr_num*)

To bind mode interrupt and interrupt sequence number.

Parameters

uint32	model_num : The mode sequence number.
uint32	intr_num : interrupt sequence number.

Returns

null

4.21 I2S Driver APIs

I2S driver APIs.

Data Structures

- struct sdio_queue

Macros

- #define **TX_MASTER** 0
- #define **TX_SLAVE** 1
- #define **RX_MASTER** 2
- #define **RX_SLAVE** 3
- #define **ETS_SLC_INTR_ENABLE**() xt_ints_on(1 << ETS_SLC_INUM)
- #define **CONF_RXLINK_ADDR**(addr)
- #define **CONF_TXLINK_ADDR**(addr)
- #define **START_RXLINK**() SET_PERI_REG_MASK(I2SRX_LINK, I2S_I2S_RXLINK_START)
- #define **START_TXLINK**() SET_PERI_REG_MASK(I2STX_LINK, I2S_I2S_TXLINK_START)

Functions

- void i2s_GPIO_init (uint8 mode)

    GPIO initialization, including config DATA, WS and BCK GPIO pin.
- void i2s_init (void)

    I2S module initialization, including FIFO, M/S mode, data format, clock frequency.
- void slc_init (void)

    DMA module initialization, including DMA mode and interrupt.
- void slc_isr (void ∗para)

    Process data received/treansmitted when interrupt occurs.
- void create_one_link (uint8 own, uint8 eof, uint8 sub_sof, uint16 size, uint16 length, uint32 ∗buf_ptr, struct sdio_queue ∗nxt_ptr, struct sdio_queue ∗i2s_queue)

    Create DMA buffer descriptors.
- void i2s_test (void)

    Functional DEMO for i2s module.

4.21.1 Detailed Description

I2S driver APIs.

4.21.2 Macro Definition Documentation

4.21.2.1 #define CONF_RXLINK_ADDR(*addr*)

Value:

```
CLEAR_PERI_REG_MASK(I2SRX_LINK, I2S_I2S_RXLINK_ADDR);\
    SET_PERI_REG_MASK(I2SRX_LINK, ((uint32)(addr)) & I2S_I2S_RXLINK_ADDR)
```

4.21.2.2 #define CONF_TXLINK_ADDR(*addr*)

Value:

```
CLEAR_PERI_REG_MASK(I2STX_LINK, I2S_I2S_TXLINK_ADDR);\
    SET_PERI_REG_MASK(I2STX_LINK, ((uint32)(addr)) & I2S_I2S_TXLINK_ADDR)
```

4.21.3 Function Documentation

4.21.3.1 void create_one_link (uint8 *own,* uint8 *eof,* uint8 *sub_sof,* uint16 *size,* uint16 *length,* uint32 ∗ *buf_ptr,* struct sdio_queue ∗ *nxt_ptr,* struct sdio_queue ∗ *i2s_queue*)

Create DMA buffer descriptors.

Parameters

uint8	own : select the owner of the current link to be either software or hardware
uint8	eof : mark for end of file
uint8	sub_sof : mark for sub start of file
uint16	size : the actual size of the buffer
uint16	length : the total size of the buffer
uint32∗	buf_ptr : the start address of the buffer
struct	sdio_queue∗ nxt_ptr : the address of the next descriptor
struct	sdio_queue∗ i2s_queue : the address of the current descriptor

Returns

null

4.21.3.2 void i2s_GPIO_init (uint8 *mode*)

GPIO initialization, including config DATA, WS and BCK GPIO pin.

Attention

This API can be called only once per mode.

Parameters

uint8	mode : i2s mode select between TX_MASTER, TX_SLAVE, RX_MASTER, RX_SLAVE;

Returns

null

4.21.3.3 void i2s_init (void)

I2S module initialization, including FIFO, M/S mode, data format, clock frequency.

Attention

This API can be called only once per mode.

Parameters

null	

Returns

null

4.21.3.4 void i2s_test (void)

Functional DEMO for i2s module.

Parameters

null	

Returns

null

4.21.3.5 void slc_init (void)

DMA module initialization, including DMA mode and interrupt.

Attention

This API can be called only once per mode.

Parameters

null	

Returns

null

4.21.3.6 void slc_isr (void ∗ *para*)

Process data received/treansmitted when interrupt occurs.

Attention

This API can be called only once per mode.

Parameters

void	∗para: pointer to parameter

Returns

null

4.22 PWM Driver APIs

PWM driver APIs.

Data Structures

- struct pwm_param

Macros

- #define **PWM_CHANNEL_NUM_MAX** 8
- #define **TIMER0** 0
- #define **TIMER1** 1
- #define **TIMER2** 2
- #define **TIMER3** 3
- #define **CHANNEL0** 0
- #define **CHANNEL1** 1
- #define **CHANNEL2** 2
- #define **CHANNEL3** 3
- #define **CHANNEL4** 4
- #define **CHANNEL5** 5
- #define **CHANNEL6** 6
- #define **CHANNEL7** 7
- #define **OUTPUT_LOW** 0
- #define **OUTPUT_HIGH** 1
- #define **REF_TICK_CLK** 0
- #define **APB_CLK** 1

Functions

- void pwm_init (uint32 period, uint32 ∗duty, uint32 pwm_channel_num, uint32(∗pin_info_list)[3])

  PWM function initialization, including GPIO, frequency and duty cycle.
- void pwm_set_duty (uint32 duty, uint8 channel)

  Set the duty cycle of a PWM channel.
- uint32 pwm_get_duty (uint8 channel)

  Get the duty cycle of a PWM channel.
- void pwm_set_period (uint32 period)

  Set PWM period, unit : us.
- uint32 pwm_get_period (void)

  Get PWM period, unit : us.
- void pwm_start (void)

  Starts PWM.
- void ledc_set_base_hclk (uint8 timer_sel, uint8 apb_clk_sel)

  Set high_speed channel base clock.
- void ledc_set_base_lclk (uint8 timer_sel, uint8 apb_clk_sel)

  Set low_speed channel base clock.
- void ledc_set_hperiod (uint8 timer_sel, uint32 div_num, uint8 timer_lim)

  Set high_speed channel frequency.
- void ledc_set_lperiod (uint8 timer_sel, uint32 div_num, uint8 timer_lim)

Set low_speed channel frequency.
- void ledc_set_ltimer (uint8 chan_num, uint8 timer_sel)

  Select one timer for one low_speed channel.
- void ledc_set_htimer (uint8 chan_num, uint8 timer_sel)

  Select one timer for one high_speed channel.
- void ledc_set_idle_hlevel (uint8 chan_num, uint8 idle_level)

  Set high_speed channel output (as high or low) when idle.
- void ledc_set_idle_llevel (uint8 chan_num, uint8 idle_level)

  Set low_speed channel output (as high or low) when idle.
- void ledc_set_hduty (uint8 chan_num, uint32 hpoint_val, uint32 duty_val, uint8 increase, uint16 duty_num, uint16 duty_cycle, uint16 duty_scale)

  Set high_speed channel duty.
- void ledc_set_lduty (uint8 chan_num, uint32 hpoint_val, uint32 duty_val, uint8 increase, uint16 duty_num, uint16 duty_cycle, uint16 duty_scale)

  Set low_speed channel duty.
- void ledc_hstart (uint8 chan_num)

  Enable one high_speed channel.
- void ledc_lstart (uint8 chan_num)

  Enable one low_speed channel.
- void ledc_timer_hpause (uint8 timer_sel)

  Pause one of the timers for high_speed channel.
- void ledc_timer_lpause (uint8 timer_sel)

  Pause one of the timers for low_speed channel.
- void ledc_timer_hunpause (uint8 timer_sel)

  Unpause one of the timers for high_speed channel.
- void ledc_timer_lunpause (uint8 timer_sel)

  Unpause one of the timers for low_speed channel.
- void ledc_timer_hstop (uint8 timer_sel)

  Stop one of the timers for high_speed channel.
- void ledc_timer_lstop (uint8 timer_sel)

  Stop one of the timers for low_speed channel.

4.22.1 Detailed Description

PWM driver APIs.

4.22.2 Function Documentation

4.22.2.1 void ledc_hstart (uint8 *chan_num*)

Enable one high_speed channel.

Parameters

uint8	chan_num : channel to pick

Returns

null

4.22.2.2 void ledc_lstart (uint8 *chan_num*)

Enable one low_speed channel.

Parameters

uint8	chan_num : channel to pick

Returns

null

4.22.2.3 void ledc_set_base_hclk (uint8 *timer_sel,* uint8 *apb_clk_sel*)

Set high_speed channel base clock.

Parameters

uint8	timer_sel : timer to set
uint8	apb_clk_sel : pick clock source for timer

Returns

null

4.22.2.4 void ledc_set_base_lclk (uint8 *timer_sel,* uint8 *apb_clk_sel*)

Set low_speed channel base clock.

Parameters

uint8	timer_sel : timer to set
uint8	apb_clk_sel :pick clock source for timer

Returns

null

4.22.2.5 void ledc_set_hduty (uint8 *chan_num,* uint32 *hpoint_val,* uint32 *duty_val,* uint8 *increase,* uint16 *duty_num,* uint16 *duty_cycle,* uint16 *duty_scale*)

Set high_speed channel duty.

Parameters

uint8	chan_num : 8 channels in total,value from 0∼7
uint32	hpoint_val : output high when counter equals this value
uint32	duty_val : output low after counter equals this value
uint8	increase : 1 - increase duty ratio, 0 - decrease duty ratio
uint16	duty_num : generate interrupt after duty_num ∗ duty_cycle outputs
uint16	duty_cycle : increase or decrease duty ratio every duty_cycle outputs
uint16	duty_scale : the range of changing on duty ratio

Returns

null

4.22.2.6 void ledc_set_hperiod (uint8 *timer_sel,* uint32 *div_num,* uint8 *timer_lim*)

Set high_speed channel frequency.

frequency=base_clk_frequency∗div_num∗(2^timer_lim)/256

Parameters

uint8	timer_sel : timer to set
uint32	div_num : set first divider
uint8	timer_lim : set second divider

Returns

null

4.22.2.7 void ledc_set_htimer (uint8 *chan_num,* uint8 *timer_sel*)

Select one timer for one high_speed channel.

Parameters

uint8	chan_num : channel to pick
uint8	timer_sel : timer to set

Returns

null

4.22.2.8 void ledc_set_idle_hlevel (uint8 *chan_num,* uint8 *idle_level*)

Set high_speed channel output (as high or low) when idle.

Parameters

uint8	chan_num : channel to pick
uint8	idle_level : choose output as high or low

Returns

null

4.22.2.9 void ledc_set_idle_llevel (uint8 *chan_num,* uint8 *idle_level*)

Set low_speed channel output (as high or low) when idle.

Parameters

uint8	chan_num : channel to pick
uint8	idle_level : choose output as high or low

Returns

null

4.22.2.10 void ledc_set_lduty (uint8 *chan_num,* uint32 *hpoint_val,* uint32 *duty_val,* uint8 *increase,* uint16 *duty_num,* uint16 *duty_cycle,* uint16 *duty_scale*)

Set low_speed channel duty.

Parameters

uint8	chan_num : 8 channels in total, value from 0∼7
uint32	hpoint_val : output high when counter equals this value
uint32	duty_val : output low after counter equals this value
uint8	increase : 1 - increase duty ratio, 0 - decrease duty ratio
uint16	duty_num : generate interrupt after duty_num ∗ duty_cycle outputs
uint16	duty_cycle : increase or decrease duty ratio every duty_cycle outputs
uint16	duty_scale : the range of changing on duty ratio

Returns

null

4.22.2.11 void ledc_set_lperiod (uint8 *timer_sel,* uint32 *div_num,* uint8 *timer_lim*)

Set low_speed channel frequency.

frequency=base_clk_frequency∗div_num∗(2^timer_lim)/256

Parameters

uint8	timer_sel : timer to set
uint32	div_num : set first divider
uint8	timer_lim : set second divider

Returns

null

4.22.2.12 void ledc_set_ltimer (uint8 *chan_num,* uint8 *timer_sel*)

Select one timer for one low_speed channel.

Parameters

uint8	chan_num : channel to pick
uint8	timer_sel : timer to set

Returns

null

4.22.2.13 void ledc_timer_hpause (uint8 *timer_sel*)

Pause one of the timers for high_speed channel.

Parameters

uint8	timer_sel : timer to set

Returns

null

4.22.2.14 void ledc_timer_hstop (uint8 *timer_sel*)

Stop one of the timers for high_speed channel.

Parameters

uint8	timer_sel : timer to set

Returns

null

4.22.2.15 void ledc_timer_hunpause (uint8 *timer_sel*)

Unpause one of the timers for high_speed channel.

Parameters

uint8	timer_sel : timer to set

Returns

null

4.22.2.16 void ledc_timer_lpause (uint8 *timer_sel*)

Pause one of the timers for low_speed channel.

Parameters

uint8	timer_sel : timer to set

Returns

null

4.22.2.17 void ledc_timer_lstop (uint8 *timer_sel*)

Stop one of the timers for low_speed channel.

Parameters

uint8	timer_sel : timer to set

Returns

null

4.22.2.18 void ledc_timer_lunpause (uint8 *timer_sel*)

Unpause one of the timers for low_speed channel.

Parameters

uint8	timer_sel : timer to set

Returns

null

4.22.2.19 uint32 pwm_get_duty (uint8 *channel*)

Get the duty cycle of a PWM channel.

Duty cycle will be (duty ∗ 45)/(period ∗1000).

Parameters

uint8	channel : PWM channel number

Returns

Duty cycle of PWM output.

4.22.2.20 uint32 pwm_get_period (void)

Get PWM period, unit : us.

Parameters

null	

Returns

PWM period, unit : us.

4.22.2.21 void pwm_init (uint32 *period,* uint32 ∗ *duty,* uint32 *pwm_channel_num,* uint32(∗) *pin_info_list[3]*)

PWM function initialization, including GPIO, frequency and duty cycle.

Attention

This API can be called only once.

Parameters

uint32	period : pwm frequency
uint32	∗duty : duty cycle
uint32	pwm_channel_num : PWM channel number
uint32	(∗pin_info_list)[3] : GPIO parameter of PWM channel, it is a pointer of n x 3 array which defines GPIO register, IO reuse of corresponding pin and GPIO number.

Returns

null

4.22.2.22 void pwm_set_duty (uint32 *duty,* uint8 *channel*)

Set the duty cycle of a PWM channel.

Set the time that high level signal will last, duty depends on period, the maximum value can be period $*1000$ / 45. For example, 1KHz PWM, duty range is 0∼22222

Attention

After set configuration, pwm_start needs to be called to take effect.

Parameters

uint32	duty : duty cycle
uint8	channel : PWM channel number

Returns

null

4.22.2.23 void pwm_set_period (uint32 *period*)

Set PWM period, unit : us.

For example, for 1KHz PWM, period is 1000 us.

Attention

After set configuration, pwm_start needs to be called to take effect.

Parameters

uint32	period : PWM period, unit : us.

Returns

null

4.22.2.24 void pwm_start (void)

Starts PWM.

Attention

This function needs to be called after PWM configuration is changed.

Parameters

null	

Returns

null

4.23 UART Driver APIs

UART driver APIs.

Functions

- void uart_div_modify (UART_Port uart_no, uint16 div)

  Set UART baud rate.
- void UART_WaitTxFifoEmpty (UART_Port uart_no)

  Wait uart tx fifo empty, do not use it if tx flow control enabled.
- void UART_ResetFifo (UART_Port uart_no)

  Clear uart tx fifo and rx fifo.
- void UART_ClearIntrStatus (UART_Port uart_no, uint32 clr_mask)

  Clear uart interrupt flags.
- void UART_SetIntrEna (UART_Port uart_no, uint32 ena_mask)

  Enable uart interrupts .
- void UART_intr_handler_register (void ∗fn, void ∗arg)

  Register an application-specific interrupt handler for Uarts interrupts.
- void UART_SetPrintPort (UART_Port uart_no)

  Config from which serial output printf function.
- void UART_ParamConfig (UART_Port uart_no, UART_ConfigTypeDef ∗pUARTConfig)

  Config Common parameters of serial ports.
- void UART_IntrConfig (UART_Port uart_no, UART_IntrConfTypeDef ∗pUARTIntrConf)

  Config types of uarts.
- void UART_SetWordLength (UART_Port uart_no, UART_WordLength len)

  Config the length of the uart communication data bits.
- void UART_SetStopBits (UART_Port uart_no, UART_StopBits bit_num)

  Config the length of the uart communication stop bits.
- void UART_SetParity (UART_Port uart_no, UART_ParityMode Parity_mode)

  Configure whether to open the parity.
- void UART_SetBaudrate (UART_Port uart_no, uint32 baud_rate)

  Configure the Baud rate.
- void UART_SetFlowCtrl (UART_Port uart_no, UART_HwFlowCtrl flow_ctrl, uint8 rx_thresh)

  Configure Hardware flow control.
- void UART_SetLineInverse (UART_Port uart_no, UART_LineLevelInverse inverse_mask)

  Configure trigging signal of uarts.
- void uart_init_new (void)

  An example illustrates how to configure the serial port.

4.23.1 Detailed Description

UART driver APIs.

4.23.2 Function Documentation

4.23.2.1 void UART_ClearIntrStatus (UART_Port *uart_no,* uint32 *clr_mask*)

Clear uart interrupt flags.

Parameters

UART_Port	uart_no : UART0 or UART1
uint32	clr_mask : To clear the interrupt bits

Returns

null

4.23.2.2 void uart_div_modify (UART_Port *uart_no,* uint16 *div*)

Set UART baud rate.

Example : uart_div_modify(uart_no, UART_CLK_FREQ / (UartDev.baut_rate));

Parameters

UART_Port	uart_no : UART0 or UART1
uint16	div : frequency divider

Returns

null

4.23.2.3 void uart_init_new (void)

An example illustrates how to configure the serial port.

Parameters

null	

Returns

null

4.23.2.4 void UART_intr_handler_register (void ∗ *fn,* void ∗ *arg*)

Register an application-specific interrupt handler for Uarts interrupts.

Parameters

void	∗fn : interrupt handler for Uart interrupts.
void	∗arg : interrupt handler's arg.

Returns

null

4.23.2.5 void UART_IntrConfig (UART_Port *uart_no,* UART_IntrConfTypeDef ∗ *pUARTIntrConf*)

Config types of uarts.

Parameters

UART_Port	uart_no : UART0 or UART1
UART_IntrConfTypeDef	∗pUARTIntrConf : parameters structure

Returns

null

4.23.2.6 void UART_ParamConfig (UART_Port *uart_no,* UART_ConfigTypeDef ∗ *pUARTConfig*)

Config Common parameters of serial ports.

Parameters

UART_Port	uart_no : UART0 or UART1
UART_ConfigTypeDef	∗pUARTConfig : parameters structure

Returns

null

4.23.2.7 void UART_ResetFifo (UART_Port *uart_no*)

Clear uart tx fifo and rx fifo.

Parameters

UART_Port	uart_no : UART0 or UART1

Returns

null

4.23.2.8 void UART_SetBaudrate (UART_Port *uart_no,* uint32 *baud_rate*)

Configure the Baud rate.

Parameters

UART_Port	uart_no : UART0 or UART1
uint32	baud_rate : the Baud rate

Returns

null

4.23.2.9 void UART_SetFlowCtrl (UART_Port *uart_no,* UART_HwFlowCtrl *flow_ctrl,* uint8 *rx_thresh*)

Configure Hardware flow control.

Parameters

UART_Port	uart_no : UART0 or UART1
UART_HwFlowCtrl	flow_ctrl : Hardware flow control mode
uint8	rx_thresh : threshold of Hardware flow control

Returns

null

4.23.2.10 void UART_SetIntrEna (UART_Port *uart_no,* uint32 *ena_mask*)

Enable uart interrupts .

Parameters

UART_Port	uart_no : UART0 or UART1
uint32	ena_mask : To enable the interrupt bits

Returns

null

4.23.2.11 void UART_SetLineInverse (UART_Port *uart_no,* UART_LineLevelInverse *inverse_mask*)

Configure trigging signal of uarts.

Parameters

UART_Port	uart_no : UART0 or UART1
UART_LineLevelInverse	inverse_mask : Choose need to flip the IO

Returns

null

4.23.2.12 void UART_SetParity (UART_Port *uart_no*, UART_ParityMode *Parity_mode*)

Configure whether to open the parity.

Parameters

UART_Port	uart_no : UART0 or UART1
UART_ParityMode	Parity_mode : the enum of uart parity configuration

Returns

null

4.23.2.13 void UART_SetPrintPort (UART_Port *uart_no*)

Config from which serial output printf function.

Parameters

UART_Port	uart_no : UART0 or UART1

Returns

null

4.23.2.14 void UART_SetStopBits (UART_Port *uart_no*, UART_StopBits *bit_num*)

Config the length of the uart communication stop bits.

Parameters

UART_Port	uart_no : UART0 or UART1
UART_StopBits	bit_num : the length uart communication stop bits

Returns

null

4.23.2.15 void UART_SetWordLength (UART_Port *uart_no*, UART_WordLength *len*)

Config the length of the uart communication data bits.

Parameters

UART_Port	uart_no : UART0 or UART1
UART_WordLength	len : the length of the uart communication data bits

Returns

null

4.23.2.16 void UART_WaitTxFifoEmpty (UART_Port *uart_no*)

Wait uart tx fifo empty, do not use it if tx flow control enabled.

Parameters

UART_Port	uart_no : UART0 or UART1

Returns

null

Chapter 5

Data Structure Documentation

5.1 _esp_event Struct Reference

Data Fields

- SYSTEM_EVENT event_id
- Event_Info_u event_info

5.1.1 Field Documentation

5.1.1.1 SYSTEM_EVENT event_id

even ID

5.1.1.2 Event_Info_u event_info

event information

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.2 _os_timer_t Struct Reference

Data Fields

- struct _os_timer_t ∗ **timer_next**
- void ∗ **timer_handle**
- uint32 **timer_expire**
- uint32 **timer_period**
- os_timer_func_t ∗ **timer_func**
- bool **timer_repeat_flag**
- void ∗ **timer_arg**

The documentation for this struct was generated from the following file:

- include/esp_timer.h

5.3 b_info Struct Reference

Data Fields

- char bin_start_flash_id
- char bin_end_flash_id
- char bin_status
- char pad
- unsigned int jump_addr

5.3.1 Field Documentation

5.3.1.1 char bin_end_flash_id

end flash id of bin file

5.3.1.2 char bin_start_flash_id

start flash id of bin file

5.3.1.3 char bin_status

bin's run status

5.3.1.4 unsigned int jump_addr

jump_addr is irom0_flash.bin's start addr in flash

5.3.1.5 char pad

padding

The documentation for this struct was generated from the following file:

- include/esp_system.h

5.4 bss_info Struct Reference

Public Member Functions

- STAILQ_ENTRY (bss_info) next

Data Fields

- uint8 bssid [6]
- uint8 ssid [32]
- uint8 ssid_len
- uint8 channel
- uint8 chan_width
- sint8 rssi
- AUTH_MODE authmode
- uint8 is_hidden
- sint16 freq_offset
- sint16 **freqcal_val**
- uint8 ∗ **esp_mesh_ie**

5.4.1 Member Function Documentation

5.4.1.1 STAILQ_ENTRY (bss_info)

information of next AP

5.4.2 Field Documentation

5.4.2.1 AUTH_MODE authmode

authmode of AP

5.4.2.2 uint8 bssid[6]

MAC address of AP

5.4.2.3 uint8 chan_width

channel width of AP, 0-ht20, 1-ht40U 2-ht40D

5.4.2.4 uint8 channel

channel of AP

5.4.2.5 sint16 freq_offset

frequency offset

5.4.2.6 uint8 is_hidden

SSID of current AP is hidden or not.

5.4.2.7 sint8 rssi

single strength of AP

5.4.2.8 uint8 ssid[32]

SSID of AP

5.4.2.9 uint8 ssid_len

SSID length

The documentation for this struct was generated from the following file:

- include/esp_sta.h

5.5 cmd_s Struct Reference

Data Fields

- char ∗ **cmd_str**
- uint8 **flag**
- uint8 **id**
- void(∗ **cmd_func**)(void)
- void(∗ **cmd_callback**)(void ∗arg)

The documentation for this struct was generated from the following file:

- include/esp_ssc.h

5.6 dhcps_lease Struct Reference

Data Fields

- bool enable
- ip4_addr_t start_ip
- ip4_addr_t end_ip
- ip4_addr_t **net_mask**

5.6.1 Field Documentation

5.6.1.1 bool enable

enable DHCP lease or not

5.6.1.2 ip4_addr_t end_ip

end IP of IP range

5.6.1.3 ip4_addr_t start_ip

start IP of IP range

The documentation for this struct was generated from the following file:

- include/esp_misc.h

5.7 Event_Info_u Union Reference

Data Fields

- Event_StaMode_ScanDone_t scan_done
- Event_StaMode_Connected_t connected
- Event_StaMode_Disconnected_t disconnected
- Event_StaMode_AuthMode_Change_t auth_change
- Event_StaMode_Got_IP_t got_ip
- Event_SoftAPMode_StaConnected_t sta_connected
- Event_SoftAPMode_StaDisconnected_t sta_disconnected
- Event_SoftAPMode_ProbeReqRecved_t ap_probereqrecved

5.7.1 Field Documentation

5.7.1.1 Event_SoftAPMode_ProbeReqRecved_t ap_probereqrecved

ESP32 softAP receive probe request packet

5.7.1.2 Event_StaMode_AuthMode_Change_t auth_change

the auth mode of AP ESP32 station connected to changed

5.7.1.3 Event_StaMode_Connected_t connected

ESP32 station connected to AP

5.7.1.4 Event_StaMode_Disconnected_t disconnected

ESP32 station disconnected to AP

5.7.1.5 Event_StaMode_Got_IP_t got_ip

ESP32 station got IP

5.7.1.6 Event_StaMode_ScanDone_t scan_done

ESP32 station scan (APs) done

5.7.1.7 Event_SoftAPMode_StaConnected_t sta_connected

a station connected to ESP32 soft-AP

5.7.1.8 Event_SoftAPMode_StaDisconnected_t sta_disconnected

a station disconnected to ESP32 soft-AP

The documentation for this union was generated from the following file:

- include/esp_wifi.h

5.8 Event_SoftAPMode_ProbeReqRecved_t Struct Reference

Data Fields

- int rssi
- uint8 mac [6]

5.8.1 Field Documentation

5.8.1.1 uint8 mac[6]

MAC address of the station which send probe request

5.8.1.2 int rssi

Received probe request signal strength

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.9 Event_SoftAPMode_StaConnected_t Struct Reference

Data Fields

- uint8 mac [6]
- uint8 aid

5.9.1 Field Documentation

5.9.1.1 uint8 aid

the aid that ESP32 soft-AP gives to the station connected to

5.9.1.2 uint8 mac[6]

MAC address of the station connected to ESP32 soft-AP

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.10 Event_SoftAPMode_StaDisconnected_t Struct Reference

Data Fields

- uint8 mac [6]
- uint8 aid

5.10.1 Field Documentation

5.10.1.1 uint8 aid

the aid that ESP32 soft-AP gave to the station disconnects to

5.10.1.2 uint8 mac[6]

MAC address of the station disconnects to ESP32 soft-AP

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.11 Event_StaMode_AuthMode_Change_t Struct Reference

Data Fields

- uint8 old_mode
- uint8 new_mode

5.11.1 Field Documentation

5.11.1.1 uint8 new_mode

the new auth mode of AP

5.11.1.2 uint8 old_mode

the old auth mode of AP

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.12 Event_StaMode_Connected_t Struct Reference

Data Fields

- uint8 ssid [32]
- uint8 ssid_len
- uint8 bssid [6]
- uint8 channel

5.12.1 Field Documentation

5.12.1.1 uint8 bssid[6]

BSSID of connected AP

5.12.1.2 uint8 channel

channel of connected AP

5.12.1.3 uint8 ssid[32]

SSID of connected AP

5.12.1.4 uint8 ssid_len

SSID length of connected AP

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.13 Event_StaMode_Disconnected_t Struct Reference

Data Fields

- uint8 ssid [32]
- uint8 ssid_len
- uint8 bssid [6]
- uint8 reason

5.13.1 Field Documentation

5.13.1.1 uint8 bssid[6]

BSSID of disconnected AP

5.13.1.2 uint8 reason

reason of disconnection

5.13.1.3 uint8 ssid[32]

SSID of disconnected AP

5.13.1.4 uint8 ssid_len

SSID length of disconnected AP

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.14 Event_StaMode_Got_IP_t Struct Reference

Data Fields

- ip4_addr_t ip
- ip4_addr_t mask
- ip4_addr_t gw

5.14.1 Field Documentation

5.14.1.1 ip4_addr_t gw

gateway that ESP32 station got from connected AP

5.14.1.2 ip4_addr_t ip

IP address that ESP32 station got from connected AP

5.14.1.3 ip4_addr_t mask

netmask that ESP32 station got from connected AP

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.15 Event_StaMode_ScanDone_t Struct Reference

Data Fields

- uint32 status
- struct bss_info ∗ bss

5.15.1 Field Documentation

5.15.1.1 struct bss_info∗ bss

list of APs found

5.15.1.2 uint32 status

status of scanning APs

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.16 GPIO_ConfigTypeDef Struct Reference

Data Fields

- uint32 GPIO_Pin
- uint32 GPIO_Pin_high
- GPIOMode_TypeDef GPIO_Mode
- GPIO_Pullup_IF GPIO_Pullup
- GPIO_Pulldown_IF GPIO_Pulldown
- GPIO_INT_TYPE GPIO_IntrType

5.16.1 Field Documentation

5.16.1.1 GPIO_INT_TYPE GPIO_IntrType

GPIO interrupt type

5.16.1.2 GPIOMode_TypeDef GPIO_Mode

GPIO mode

5.16.1.3 uint32 GPIO_Pin

GPIO pin

5.16.1.4 uint32 GPIO_Pin_high

GPIO pin

5.16.1.5 GPIO_Pulldown_IF GPIO_Pulldown

GPIO pulldown

5.16.1.6 GPIO_Pullup_IF GPIO_Pullup

GPIO pullup

The documentation for this struct was generated from the following file:

- driver_lib/include/gpio.h

5.17 ip_info Struct Reference

Data Fields

- ip4_addr_t ip
- ip4_addr_t netmask
- ip4_addr_t gw

5.17.1 Field Documentation

5.17.1.1 ip4_addr_t gw

gateway

5.17.1.2 ip4_addr_t ip

IP address

5.17.1.3 ip4_addr_t netmask

netmask

The documentation for this struct was generated from the following file:

- include/esp_wifi.h

5.18 phy_config Struct Reference

Data Fields

- enum phy_mode phy_mode
- enum phy_bw phy_bw
- enum phy_2nd_chan phy_2nd_chan
- uint32 cfg_mask

5.18.1 Field Documentation

5.18.1.1 uint32 cfg_mask

configuration mask

5.18.1.2 enum phy_2nd_chan phy_2nd_chan

HT40 second channel, this parameter is for AP only, for sta the second channel is determined by AP

5.18.1.3 enum **phy_bw phy_bw**

phy bandwidth

5.18.1.4 enum **phy_mode phy_mode**

phy mode

The documentation for this struct was generated from the following file:

- include/esp_phy.h

5.19 pwm_param Struct Reference

Data Fields

- uint32 **period**
- uint32 **freq**
- uint32 **duty** [PWM_CHANNEL_NUM_MAX]

The documentation for this struct was generated from the following file:

- driver_lib/include/pwm.h

5.20 regdomain_info::regdomain_chan Struct Reference

Data Fields

- uint8_t schan
- uint8_t nchan
- uint8_t maxtxpwr

5.20.1 Field Documentation

5.20.1.1 uint8_t maxtxpwr

The max tranmit power, this field is ignored in ESP32

5.20.1.2 uint8_t nchan

The the count of channels in operation class

5.20.1.3 uint8_t schan

< regdomain_chan indicates on operation class Start channel number, the range is from 1 to 14

The documentation for this struct was generated from the following file:

- include/esp_regdomain.h

5.21 regdomain_info Struct Reference

Data Structures

- struct regdomain_chan

Data Fields

- uint8_t rd_sta_enable
- uint8_t rd_ap_enable
- uint16_t regdomain
- uint16_t country
- uint8_t location
- char isocc [2]
- uint8_t ngroup
- struct regdomain_info::regdomain_chan **chan** [REGDOMAIN_CHANGROUP_MAX]

5.21.1 Field Documentation

5.21.1.1 uint16_t country

ISO country code

5.21.1.2 char isocc[2]

country code string

5.21.1.3 uint8_t location

I (indoor), O(outdoor), other

5.21.1.4 uint8_t ngroup

For ESP32, ngroup should always be 1

5.21.1.5 uint8_t rd_ap_enable

softap regdomain enable/disable flag, 1 means enable, 0 means disable, default 0

5.21.1.6 uint8_t rd_sta_enable

station regdomain enable/disable flag, 1 means enable, 0 means disable, default 0

5.21.1.7 uint16_t regdomain

regdomain

The documentation for this struct was generated from the following file:

- include/esp_regdomain.h

5.22 remote_bin_info Struct Reference

Data Fields

- uint32 b_sumlen
- uint32 b_irom1len
- uint8 flash_id_num

5.22.1 Field Documentation

5.22.1.1 uint32 b_irom1len

bin's irom1 part length

5.22.1.2 uint32 b_sumlen

bin's sum length

5.22.1.3 uint8 flash_id_num

the number of flash id occupied by bin file ,can be calculated by b_sumlen & b_irom1len

The documentation for this struct was generated from the following file:

- include/upgrade.h

5.23 scan_config Struct Reference

Data Fields

- uint8 ∗ ssid
- uint8 ∗ bssid
- uint8 channel
- uint8 show_hidden

5.23.1 Field Documentation

5.23.1.1 uint8∗ bssid

MAC address of AP

5.23.1.2 uint8 channel

channel, scan the specific channel

5.23.1.3 uint8 show_hidden

enable to scan AP whose SSID is hidden

5.23.1.4 uint8∗ ssid

SSID of AP

The documentation for this struct was generated from the following file:

- include/esp_sta.h

5.24 sdio_queue Struct Reference

Data Fields

- uint32 **blocksize**: 12
- uint32 **datalen**: 12
- uint32 **unused**: 5
- uint32 **sub_sof**: 1
- uint32 **eof**: 1
- uint32 **owner**: 1
- uint32 **buf_ptr**
- uint32 **next_link_ptr**

The documentation for this struct was generated from the following file:

- driver_lib/include/i2s.h

5.25 server_info Struct Reference

Data Fields

- struct sockaddr_in sockaddrin
- char ∗ http_req

5.25.1 Field Documentation

5.25.1.1 char∗ http_req

http request url

5.25.1.2 struct sockaddr_in sockaddrin

remote server info,ip and port

The documentation for this struct was generated from the following file:

- include/upgrade.h

5.26 softap_config Struct Reference

Data Fields

- uint8 ssid [32]
- uint8 password [64]
- uint8 ssid_len
- uint8 channel
- AUTH_MODE authmode
- uint8 ssid_hidden
- uint8 max_connection
- uint16 beacon_interval

5.26.1 Field Documentation

5.26.1.1 AUTH_MODE authmode

Auth mode of ESP32 soft-AP. Do not support AUTH_WEP in soft-AP mode

5.26.1.2 uint16 beacon_interval

Beacon interval, 100 ∼ 60000 ms, default 100

5.26.1.3 uint8 channel

Channel of ESP32 soft-AP

5.26.1.4 uint8 max_connection

Max number of stations allowed to connect in, default 4, max 4

5.26.1.5 uint8 password[64]

Password of ESP32 soft-AP

5.26.1.6 uint8 ssid[32]

SSID of ESP32 soft-AP

5.26.1.7 uint8 ssid_hidden

Broadcast SSID or not, default 0, broadcast the SSID

5.26.1.8 uint8 ssid_len

Length of SSID. If softap_config.ssid_len==0, check the SSID until there is a termination character; otherwise, set the SSID length according to softap_config.ssid_len.

The documentation for this struct was generated from the following file:

- include/esp_softap.h

5.27 station_config Struct Reference

Data Fields

- uint8 ssid [32]
- uint8 password [64]
- uint8 bssid_set
- uint8 bssid [6]

5.27.1 Field Documentation

5.27.1.1 uint8 bssid[6]

MAC address of target AP

5.27.1.2 uint8 bssid_set

whether set MAC address of target AP or not. Generally, station_config.bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

5.27.1.3 uint8 password[64]

password of target AP

5.27.1.4 uint8 ssid[32]

SSID of target AP

The documentation for this struct was generated from the following file:

- include/esp_sta.h

5.28 station_info Struct Reference

Public Member Functions

- STAILQ_ENTRY (station_info) next

Data Fields

- uint8 bssid [6]
- ip4_addr_t ip

5.28.1 Member Function Documentation

5.28.1.1 STAILQ_ENTRY (station_info)

Information of next AP

5.28.2 Field Documentation

5.28.2.1 uint8 bssid[6]

BSSID of AP

5.28.2.2 ip4_addr_t ip

IP address of AP

The documentation for this struct was generated from the following file:

- include/esp_softap.h

5.29 UART_ConfigTypeDef Struct Reference

Data Fields

- UART_BautRate **baud_rate**
- UART_WordLength **data_bits**
- UART_ParityMode **parity**
- UART_StopBits **stop_bits**
- UART_HwFlowCtrl **flow_ctrl**
- uint8 **UART_RxFlowThresh**
- uint32 **UART_InverseMask**

The documentation for this struct was generated from the following file:

- driver_lib/include/uart.h

5.30 UART_IntrConfTypeDef Struct Reference

Data Fields

- uint32 **UART_IntrEnMask**
- uint8 **UART_RX_TimeOutIntrThresh**
- uint8 **UART_TX_FifoEmptyIntrThresh**
- uint8 **UART_RX_FifoFullIntrThresh**

The documentation for this struct was generated from the following file:

- driver_lib/include/uart.h

5.31 upgrade_info Struct Reference

Data Fields

- struct server_info s_if
- bool upgrade_flag
- uint32 check_times
- upgrade_states_check_callback check_cb

5.31.1 Field Documentation

5.31.1.1 upgrade_states_check_callback check_cb

check back function ,if user defined it ,it will be called whether OTA success or failed immediate

5.31.1.2 uint32 check_times

OTA time(ms) set by user

5.31.1.3 struct server_info s_if

remote server info

5.31.1.4 bool upgrade_flag

upgrade flag need be checked in upgrade check callback func ,false OTA failed ,true OTA success

The documentation for this struct was generated from the following file:

- include/upgrade.h