esp_iot_sdk_freertos

v1.3.0

Generated by Doxygen 1.8.10

Wed Oct 28 2015 13:42:01

Contents

1 esp_iot_rtos_sdk **1**

2 Module Index **3**

2.1 Modules . . . 3

3 Data Structure Index **5**

3.1 Data Structures . . . 5

4 Module Documentation **7**

4.1 Misc APIs . . . 7

4.1.1 Detailed Description . . . 7

4.1.2 Macro Definition Documentation . . . 8

4.1.2.1 IP2STR . . . 8

4.1.3 Enumeration Type Documentation . . . 8

4.1.3.1 dhcp_status . . . 8

4.1.3.2 dhcps_offer_option . . . 8

4.1.4 Function Documentation . . . 8

4.1.4.1 os_delay_us(uint16 us) . . . 8

4.1.4.2 os_install_putc1(void(*p)(char c)) . . . 8

4.1.4.3 os_putc(char c) . . . 9

4.2 WiFi Related APIs . . . 10

4.2.1 Detailed Description . . . 10

4.3 SoftAP APIs . . . 11

4.3.1 Detailed Description . . . 11

4.3.2 Function Documentation . . . 12

4.3.2.1 wifi_softap_dhcps_start(void) . . . 12

4.3.2.2 wifi_softap_dhcps_status(void) . . . 12

4.3.2.3 wifi_softap_dhcps_stop(void) . . . 12

4.3.2.4 wifi_softap_free_station_info(void) . . . 12

4.3.2.5 wifi_softap_get_config(struct softap_config *config) . . . 13

4.3.2.6 wifi_softap_get_config_default(struct softap_config *config) . . . 13

4.3.2.7 wifi_softap_get_dhcps_lease(struct dhcps_lease *please) . . . 13

4.3.2.8 wifi_softap_get_dhcps_lease_time(void) . . . 13
4.3.2.9 wifi_softap_get_station_info(void) . . . 14
4.3.2.10 wifi_softap_get_station_num(void) . . . 14
4.3.2.11 wifi_softap_reset_dhcps_lease_time(void) . . . 14
4.3.2.12 wifi_softap_set_config(struct softap_config *config) . . . 15
4.3.2.13 wifi_softap_set_config_current(struct softap_config *config) . . . 15
4.3.2.14 wifi_softap_set_dhcps_lease(struct dhcps_lease *please) . . . 15
4.3.2.15 wifi_softap_set_dhcps_lease_time(uint32 minute) . . . 16
4.3.2.16 wifi_softap_set_dhcps_offer_option(uint8 level, void *optarg) . . . 16
4.4 Spiffs APIs . . . 17
4.4.1 Detailed Description . . . 17
4.4.2 Function Documentation . . . 17
4.4.2.1 esp_spiffs_deinit(uint8 format) . . . 17
4.4.2.2 esp_spiffs_init(struct esp_spiffs_config *config) . . . 17
4.5 SSC APIs . . . 18
4.5.1 Detailed Description . . . 18
4.5.2 Function Documentation . . . 18
4.5.2.1 ssc_attach(SscBaudRate bandrate) . . . 18
4.5.2.2 ssc_param_len(void) . . . 18
4.5.2.3 ssc_param_str(void) . . . 18
4.5.2.4 ssc_parse_param(char *pLine, char *argv[]) . . . 19
4.5.2.5 ssc_register(ssc_cmd_t *cmdset, uint8 cmdnum, void(*help)(void)) . . . 19
4.6 Station APIs . . . 20
4.6.1 Detailed Description . . . 21
4.6.2 Typedef Documentation . . . 21
4.6.2.1 scan_done_cb_t . . . 21
4.6.3 Enumeration Type Documentation . . . 21
4.6.3.1 STATION_STATUS . . . 21
4.6.4 Function Documentation . . . 22
4.6.4.1 wifi_station_ap_change(uint8 current_ap_id) . . . 22
4.6.4.2 wifi_station_ap_number_set(uint8 ap_number) . . . 23
4.6.4.3 wifi_station_connect(void) . . . 23
4.6.4.4 wifi_station_dhcpc_start(void) . . . 23
4.6.4.5 wifi_station_dhcpc_status(void) . . . 24
4.6.4.6 wifi_station_dhcpc_stop(void) . . . 24
4.6.4.7 wifi_station_disconnect(void) . . . 24
4.6.4.8 wifi_station_get_ap_info(struct station_config config[]) . . . 25
4.6.4.9 wifi_station_get_auto_connect(void) . . . 25
4.6.4.10 wifi_station_get_config(struct station_config *config) . . . 25
4.6.4.11 wifi_station_get_config_default(struct station_config *config) . . . 25

4.6.4.12 wifi_station_get_connect_status(void) . . . 26
4.6.4.13 wifi_station_get_current_ap_id(void) . . . 26
4.6.4.14 wifi_station_get_reconnect_policy(void) . . . 26
4.6.4.15 wifi_station_get_rssi(void) . . . 26
4.6.4.16 wifi_station_scan(struct scan_config *config, scan_done_cb_t cb) . . . 27
4.6.4.17 wifi_station_set_auto_connect(bool set) . . . 27
4.6.4.18 wifi_station_set_config(struct station_config *config) . . . 27
4.6.4.19 wifi_station_set_config_current(struct station_config *config) . . . 28
4.6.4.20 wifi_station_set_reconnect_policy(bool set) . . . 28
4.7 System APIs . . . 29
4.7.1 Detailed Description . . . 30
4.7.2 Enumeration Type Documentation . . . 30
4.7.2.1 rst_reason . . . 30
4.7.3 Function Documentation . . . 30
4.7.3.1 system_adc_read(void) . . . 30
4.7.3.2 system_deep_sleep(uint32 time_in_us) . . . 31
4.7.3.3 system_deep_sleep_set_option(uint8 option) . . . 31
4.7.3.4 system_get_chip_id(void) . . . 31
4.7.3.5 system_get_free_heap_size(void) . . . 32
4.7.3.6 system_get_rst_info(void) . . . 32
4.7.3.7 system_get_rtc_time(void) . . . 32
4.7.3.8 system_get_sdk_version(void) . . . 33
4.7.3.9 system_get_time(void) . . . 34
4.7.3.10 system_get_vdd33(void) . . . 34
4.7.3.11 system_param_load(uint16 start_sec, uint16 offset, void *param, uint16 len) . . . 34
4.7.3.12 system_param_save_with_protect(uint16 start_sec, void *param, uint16 len) . . . 35
4.7.3.13 system_phy_set_max_tpw(uint8 max_tpw) . . . 35
4.7.3.14 system_phy_set_rfoption(uint8 option) . . . 35
4.7.3.15 system_phy_set_tpw_via_vdd33(uint16 vdd33) . . . 36
4.7.3.16 system_print_meminfo(void) . . . 36
4.7.3.17 system_restart(void) . . . 36
4.7.3.18 system_restore(void) . . . 37
4.7.3.19 system_rtc_clock_cali_proc(void) . . . 37
4.7.3.20 system_rtc_mem_read(uint8 src, void *dst, uint16 n) . . . 37
4.7.3.21 system_rtc_mem_write(uint8 dst, const void *src, uint16 n) . . . 38
4.7.3.22 system_uart_de_swap(void) . . . 38
4.7.3.23 system_uart_swap(void) . . . 38
4.8 Boot APIs . . . 40
4.8.1 Detailed Description . . . 40
4.8.2 Macro Definition Documentation . . . 40

4.8.2.1 SYS_BOOT_ENHANCE_MODE . . . 40
4.8.2.2 SYS_BOOT_NORMAL_BIN . . . 40
4.8.2.3 SYS_BOOT_NORMAL_MODE . . . 41
4.8.2.4 SYS_BOOT_TEST_BIN . . . 41
4.8.3 Enumeration Type Documentation . . . 41
4.8.3.1 flash_size_map . . . 41
4.8.4 Function Documentation . . . 41
4.8.4.1 system_get_boot_mode(void) . . . 41
4.8.4.2 system_get_boot_version(void) . . . 41
4.8.4.3 system_get_cpu_freq(void) . . . 41
4.8.4.4 system_get_flash_size_map(void) . . . 42
4.8.4.5 system_get_userbin_addr(void) . . . 42
4.8.4.6 system_restart_enhance(uint8 bin_type, uint32 bin_addr) . . . 42
4.8.4.7 system_update_cpu_freq(uint8 freq) . . . 43
4.9 Software timer APIs . . . 44
4.9.1 Detailed Description . . . 44
4.9.2 Function Documentation . . . 44
4.9.2.1 os_timer_arm(os_timer_t *ptimer, uint32 msec, bool repeat_flag) . . . 44
4.9.2.2 os_timer_disarm(os_timer_t *ptimer) . . . 44
4.9.2.3 os_timer_setfn(os_timer_t *ptimer, os_timer_func_t *pfunction, void *parg) . . . 45
4.10 Common APIs . . . 46
4.10.1 Detailed Description . . . 48
4.10.2 Typedef Documentation . . . 48
4.10.2.1 freedom_outside_cb_t . . . 48
4.10.2.2 rfid_locp_cb_t . . . 48
4.10.2.3 wifi_event_handler_cb_t . . . 48
4.10.3 Enumeration Type Documentation . . . 48
4.10.3.1 AUTH_MODE . . . 48
4.10.3.2 SYSTEM_EVENT . . . 49
4.10.3.3 WIFI_INTERFACE . . . 49
4.10.3.4 WIFI_MODE . . . 49
4.10.3.5 WIFI_PHY_MODE . . . 49
4.10.4 Function Documentation . . . 49
4.10.4.1 wifi_get_ip_info(WIFI_INTERFACE if_index, struct ip_info *info) . . . 49
4.10.4.2 wifi_get_macaddr(WIFI_INTERFACE if_index, uint8 *macaddr) . . . 50
4.10.4.3 wifi_get_opmode(void) . . . 50
4.10.4.4 wifi_get_opmode_default(void) . . . 50
4.10.4.5 wifi_get_phy_mode(void) . . . 51
4.10.4.6 wifi_get_sleep_type(void) . . . 52
4.10.4.7 wifi_register_rfid_locp_recv_cb(rfid_locp_cb_t cb) . . . 52

4.10.4.8 wifi_register_send_pkt_freedom_cb(freedom_outside_cb_t cb) . . . 52
4.10.4.9 wifi_rfid_locp_recv_close(void) . . . 52
4.10.4.10 wifi_rfid_locp_recv_open(void) . . . 53
4.10.4.11 wifi_send_pkt_freedom(uint8 *buf, uint16 len, bool sys_seq) . . . 53
4.10.4.12 wifi_set_event_handler_cb(wifi_event_handler_cb_t cb) . . . 53
4.10.4.13 wifi_set_ip_info(WIFI_INTERFACE if_index, struct ip_info *info) . . . 54
4.10.4.14 wifi_set_macaddr(WIFI_INTERFACE if_index, uint8 *macaddr) . . . 54
4.10.4.15 wifi_set_opmode(WIFI_MODE opmode) . . . 55
4.10.4.16 wifi_set_opmode_current(WIFI_MODE opmode) . . . 55
4.10.4.17 wifi_set_phy_mode(WIFI_PHY_MODE mode) . . . 55
4.10.4.18 wifi_set_sleep_type(sleep_type type) . . . 56
4.10.4.19 wifi_status_led_install(uint8 gpio_id, uint32 gpio_name, uint8 gpio_func) . . . 56
4.10.4.20 wifi_status_led_uninstall(void) . . . 56
4.10.4.21 wifi_unregister_rfid_locp_recv_cb(void) . . . 56
4.10.4.22 wifi_unregister_send_pkt_freedom_cb(void) . . . 57
4.11 Force Sleep APIs . . . 58
4.11.1 Detailed Description . . . 58
4.11.2 Function Documentation . . . 58
4.11.2.1 wifi_fpm_close(void) . . . 58
4.11.2.2 wifi_fpm_do_sleep(uint32 sleep_time_in_us) . . . 58
4.11.2.3 wifi_fpm_do_wakeup(void) . . . 59
4.11.2.4 wifi_fpm_get_sleep_type(void) . . . 59
4.11.2.5 wifi_fpm_open(void) . . . 59
4.11.2.6 wifi_fpm_set_sleep_type(sleep_type type) . . . 60
4.11.2.7 wifi_fpm_set_wakeup_cb(fpm_wakeup_cb cb) . . . 60
4.12 Rate Control APIs . . . 61
4.12.1 Detailed Description . . . 62
4.12.2 Function Documentation . . . 62
4.12.2.1 wifi_get_user_fixed_rate(uint8 *enable_mask, uint8 *rate) . . . 62
4.12.2.2 wifi_get_user_limit_rate_mask(void) . . . 62
4.12.2.3 wifi_set_user_fixed_rate(uint8 enable_mask, uint8 rate) . . . 62
4.12.2.4 wifi_set_user_limit_rate_mask(uint8 enable_mask) . . . 63
4.12.2.5 wifi_set_user_rate_limit(uint8 mode, uint8 ifidx, uint8 max, uint8 min) . . . 63
4.12.2.6 wifi_set_user_sup_rate(uint8 min, uint8 max) . . . 64
4.13 User IE APIs . . . 65
4.13.1 Detailed Description . . . 65
4.13.2 Typedef Documentation . . . 65
4.13.2.1 user_ie_manufacturer_recv_cb_t . . . 65
4.13.3 Function Documentation . . . 66
4.13.3.1 wifi_register_user_ie_manufacturer_recv_cb(user_ie_manufacturer_recv_cb_t cb) 66

4.13.3.2 wifi_set_user_ie(bool enable, uint8 *m_oui, user_ie_type type, uint8 *user_ie, uint8 len) . . . 67
4.13.3.3 wifi_unregister_user_ie_manufacturer_recv_cb(void) . . . 67
4.14 Sniffer APIs . . . 68
4.14.1 Detailed Description . . . 68
4.14.2 Typedef Documentation . . . 68
4.14.2.1 wifi_promiscuous_cb_t . . . 68
4.14.3 Function Documentation . . . 68
4.14.3.1 wifi_get_channel(void) . . . 68
4.14.3.2 wifi_promiscuous_enable(uint8 promiscuous) . . . 69
4.14.3.3 wifi_promiscuous_set_mac(const uint8_t *address) . . . 69
4.14.3.4 wifi_set_channel(uint8 channel) . . . 69
4.14.3.5 wifi_set_promiscuous_rx_cb(wifi_promiscuous_cb_t cb) . . . 70
4.15 WPS APIs . . . 71
4.15.1 Detailed Description . . . 71
4.15.2 Typedef Documentation . . . 71
4.15.2.1 wps_st_cb_t . . . 71
4.15.3 Enumeration Type Documentation . . . 72
4.15.3.1 wps_cb_status . . . 72
4.15.4 Function Documentation . . . 72
4.15.4.1 wifi_set_wps_cb(wps_st_cb_t cb) . . . 72
4.15.4.2 wifi_wps_disable(void) . . . 72
4.15.4.3 wifi_wps_enable(WPS_TYPE_t wps_type) . . . 72
4.15.4.4 wifi_wps_start(void) . . . 73
4.16 Network Espconn APIs . . . 74
4.16.1 Detailed Description . . . 76
4.16.2 Macro Definition Documentation . . . 76
4.16.2.1 ESPCONN_ABRT . . . 76
4.16.2.2 ESPCONN_ARG . . . 76
4.16.2.3 ESPCONN_CLSD . . . 76
4.16.2.4 ESPCONN_CONN . . . 76
4.16.2.5 ESPCONN_IF . . . 76
4.16.2.6 ESPCONN_INPROGRESS . . . 76
4.16.2.7 ESPCONN_ISCONN . . . 76
4.16.2.8 ESPCONN_MAXNUM . . . 77
4.16.2.9 ESPCONN_MEM . . . 77
4.16.2.10 ESPCONN_OK . . . 77
4.16.2.11 ESPCONN_RST . . . 77
4.16.2.12 ESPCONN_RTE . . . 77
4.16.2.13 ESPCONN_TIMEOUT . . . 77

4.16.3 Typedef Documentation . . . 77
4.16.3.1 dns_found_callback . . . 77
4.16.3.2 espconn_connect_callback . . . 77
4.16.3.3 espconn_reconnect_callback . . . 78
4.16.3.4 espconn_recv_callback . . . 78
4.16.4 Enumeration Type Documentation . . . 78
4.16.4.1 espconn_level . . . 78
4.16.4.2 espconn_option . . . 79
4.16.4.3 espconn_state . . . 79
4.16.4.4 espconn_type . . . 79
4.16.5 Function Documentation . . . 79
4.16.5.1 espconn_accept(struct espconn *espconn) . . . 79
4.16.5.2 espconn_clear_opt(struct espconn *espconn, uint8 opt) . . . 80
4.16.5.3 espconn_connect(struct espconn *espconn) . . . 80
4.16.5.4 espconn_create(struct espconn *espconn) . . . 80
4.16.5.5 espconn_delete(struct espconn *espconn) . . . 81
4.16.5.6 espconn_disconnect(struct espconn *espconn) . . . 81
4.16.5.7 espconn_dns_setserver(char numdns, ip_addr_t *dnsserver) . . . 81
4.16.5.8 espconn_get_connection_info(struct espconn *pespconn, remot_info **pconn_↩ info, uint8 typeflags) . . . 81
4.16.5.9 espconn_get_keepalive(struct espconn *espconn, uint8 level, void *optarg) . . . 82
4.16.5.10 espconn_gethostbyname(struct espconn *pespconn, const char *hostname, ip↩ _addr_t *addr, dns_found_callback found) . . . 82
4.16.5.11 espconn_igmp_join(ip_addr_t *host_ip, ip_addr_t *multicast_ip) . . . 83
4.16.5.12 espconn_igmp_leave(ip_addr_t *host_ip, ip_addr_t *multicast_ip) . . . 83
4.16.5.13 espconn_port(void) . . . 83
4.16.5.14 espconn_recv_hold(struct espconn *pespconn) . . . 83
4.16.5.15 espconn_recv_unhold(struct espconn *pespconn) . . . 84
4.16.5.16 espconn_regist_connectcb(struct espconn *espconn, espconn_connect_↩ callback connect_cb) . . . 84
4.16.5.17 espconn_regist_disconcb(struct espconn *espconn, espconn_connect_callback discon_cb) . . . 84
4.16.5.18 espconn_regist_reconcb(struct espconn *espconn, espconn_reconnect_callback recon_cb) . . . 85
4.16.5.19 espconn_regist_recvcb(struct espconn *espconn, espconn_recv_callback recv↩ _cb) . . . 85
4.16.5.20 espconn_regist_sentcb(struct espconn *espconn, espconn_sent_callback sent↩ _cb) . . . 85
4.16.5.21 espconn_regist_time(struct espconn *espconn, uint32 interval, uint8 type_flag) . . . 86
4.16.5.22 espconn_regist_write_finish(struct espconn *espconn, espconn_connect_↩ callback write_finish_fn) . . . 86
4.16.5.23 espconn_send(struct espconn *espconn, uint8 *psent, uint16 length) . . . 87

4.16.5.24 espconn_sendto(struct espconn *espconn, uint8 *psent, uint16 length) 87
4.16.5.25 espconn_sent(struct espconn *espconn, uint8 *psent, uint16 length) 88
4.16.5.26 espconn_set_keepalive(struct espconn *espconn, uint8 level, void *optarg) . . . 88
4.16.5.27 espconn_set_opt(struct espconn *espconn, uint8 opt) 89
4.16.5.28 espconn_tcp_get_max_con(void) 89
4.16.5.29 espconn_tcp_get_max_con_allow(struct espconn *espconn) 89
4.16.5.30 espconn_tcp_set_max_con(uint8 num) 90
4.16.5.31 espconn_tcp_set_max_con_allow(struct espconn *espconn, uint8 num) 91
4.17 ESP-NOW APIs 92
4.17.1 Detailed Description 93
4.17.2 Typedef Documentation 93
4.17.2.1 esp_now_recv_cb_t 93
4.17.2.2 esp_now_send_cb_t 93
4.17.3 Function Documentation 95
4.17.3.1 esp_now_add_peer(uint8 *mac_addr, uint8 role, uint8 channel, uint8 *key, uint8 key_len) 95
4.17.3.2 esp_now_deinit(void) 95
4.17.3.3 esp_now_del_peer(uint8 *mac_addr) 95
4.17.3.4 esp_now_fetch_peer(bool restart) 96
4.17.3.5 esp_now_get_cnt_info(uint8 *all_cnt, uint8 *encrypt_cnt) 96
4.17.3.6 esp_now_get_peer_channel(uint8 *mac_addr) 96
4.17.3.7 esp_now_get_peer_key(uint8 *mac_addr, uint8 *key, uint8 *key_len) 96
4.17.3.8 esp_now_get_peer_role(uint8 *mac_addr) 97
4.17.3.9 esp_now_get_self_role(void) 97
4.17.3.10 esp_now_init(void) 97
4.17.3.11 esp_now_is_peer_exist(uint8 *mac_addr) 97
4.17.3.12 esp_now_register_recv_cb(esp_now_recv_cb_t cb) 98
4.17.3.13 esp_now_register_send_cb(esp_now_send_cb_t cb) 98
4.17.3.14 esp_now_send(uint8 *da, uint8 *data, uint8 len) 98
4.17.3.15 esp_now_set_kok(uint8 *key, uint8 len) 99
4.17.3.16 esp_now_set_peer_channel(uint8 *mac_addr, uint8 channel) 100
4.17.3.17 esp_now_set_peer_key(uint8 *mac_addr, uint8 *key, uint8 key_len) 100
4.17.3.18 esp_now_set_peer_role(uint8 *mac_addr, uint8 role) 100
4.17.3.19 esp_now_set_self_role(uint8 role) 101
4.17.3.20 esp_now_unregister_recv_cb(void) 101
4.17.3.21 esp_now_unregister_send_cb(void) 101
4.18 Mesh APIs 102
4.18.1 Detailed Description 102
4.18.2 Enumeration Type Documentation 103
4.18.2.1 mesh_node_type 103

4.18.2.2 mesh_status . . . 103
4.18.3 Function Documentation . . . 103
4.18.3.1 espconn_mesh_connect(struct espconn *usr_esp) . . . 103
4.18.3.2 espconn_mesh_disable(espconn_mesh_callback disable_cb) . . . 103
4.18.3.3 espconn_mesh_disconnect(struct espconn *usr_esp) . . . 104
4.18.3.4 espconn_mesh_enable(espconn_mesh_callback enable_cb, enum mesh_type type) . . . 104
4.18.3.5 espconn_mesh_encrypt_init(AUTH_MODE mode, uint8_t *passwd, uint8_t passwd_len) . . . 104
4.18.3.6 espconn_mesh_get_max_hops() . . . 105
4.18.3.7 espconn_mesh_get_node_info(enum mesh_node_type type, uint8_t **info, uint8_t *count) . . . 105
4.18.3.8 espconn_mesh_get_status() . . . 105
4.18.3.9 espconn_mesh_group_id_init(uint8_t *grp_id, uint16_t gid_len) . . . 105
4.18.3.10 espconn_mesh_init() . . . 106
4.18.3.11 espconn_mesh_local_addr(struct ip_addr *ip) . . . 106
4.18.3.12 espconn_mesh_sent(struct espconn *usr_esp, uint8 *pdata, uint16 len) . . . 106
4.18.3.13 espconn_mesh_set_dev_type(uint8_t dev_type) . . . 107
4.18.3.14 espconn_mesh_set_max_hops(uint8_t max_hops) . . . 107
4.18.3.15 espconn_mesh_set_ssid_prefix(uint8_t *prefix, uint8_t prefix_len) . . . 107
4.19 Driver APIs . . . 109
4.19.1 Detailed Description . . . 109
4.20 PWM Driver APIs . . . 110
4.20.1 Detailed Description . . . 110
4.20.2 Function Documentation . . . 110
4.20.2.1 pwm_get_duty(uint8 channel) . . . 110
4.20.2.2 pwm_get_period(void) . . . 110
4.20.2.3 pwm_init(uint32 period, uint32 *duty, uint32 pwm_channel_num, uint32(*pin_info_list)[3]) . . . 111
4.20.2.4 pwm_set_duty(uint32 duty, uint8 channel) . . . 111
4.20.2.5 pwm_set_period(uint32 period) . . . 111
4.21 Smartconfig APIs . . . 113
4.21.1 Detailed Description . . . 113
4.21.2 Typedef Documentation . . . 113
4.21.2.1 sc_callback_t . . . 113
4.21.3 Enumeration Type Documentation . . . 114
4.21.3.1 sc_status . . . 114
4.21.3.2 sc_type . . . 114
4.21.4 Function Documentation . . . 114
4.21.4.1 esptouch_set_timeout(uint8 time_s) . . . 114
4.21.4.2 smartconfig_get_version(void) . . . 115

4.21.4.3 smartconfig_set_type(sc_type type) . . . 115
4.21.4.4 smartconfig_start(sc_callback_t cb,...) . . . 115
4.21.4.5 smartconfig_stop(void) . . . 116
4.22 SPI Driver APIs . . . 117
4.22.1 Detailed Description . . . 117
4.22.2 Macro Definition Documentation . . . 117
4.22.2.1 SPI_FLASH_SEC_SIZE . . . 117
4.22.3 Enumeration Type Documentation . . . 117
4.22.3.1 SpiFlashOpResult . . . 117
4.22.4 Function Documentation . . . 118
4.22.4.1 spi_flash_erase_sector(uint16 sec) . . . 118
4.22.4.2 spi_flash_get_id(void) . . . 119
4.22.4.3 spi_flash_read(uint32 src_addr, uint32 *des_addr, uint32 size) . . . 119
4.22.4.4 spi_flash_read_status(uint32 *status) . . . 119
4.22.4.5 spi_flash_write(uint32 des_addr, uint32 *src_addr, uint32 size) . . . 119
4.22.4.6 spi_flash_write_status(uint32 status_value) . . . 120
4.23 Upgrade APIs . . . 121
4.23.1 Detailed Description . . . 121
4.23.2 Macro Definition Documentation . . . 122
4.23.2.1 SPI_FLASH_SEC_SIZE . . . 122
4.23.2.2 UPGRADE_FLAG_FINISH . . . 122
4.23.2.3 UPGRADE_FLAG_IDLE . . . 122
4.23.2.4 UPGRADE_FLAG_START . . . 122
4.23.2.5 UPGRADE_FW_BIN1 . . . 122
4.23.2.6 UPGRADE_FW_BIN2 . . . 122
4.23.2.7 USER_BIN1 . . . 122
4.23.2.8 USER_BIN2 . . . 122
4.23.3 Typedef Documentation . . . 122
4.23.3.1 upgrade_states_check_callback . . . 122
4.23.4 Function Documentation . . . 123
4.23.4.1 system_upgrade(uint8 *data, uint32 len) . . . 123
4.23.4.2 system_upgrade_deinit() . . . 124
4.23.4.3 system_upgrade_flag_check() . . . 124
4.23.4.4 system_upgrade_flag_set(uint8 flag) . . . 124
4.23.4.5 system_upgrade_init() . . . 125
4.23.4.6 system_upgrade_reboot(void) . . . 126
4.23.4.7 system_upgrade_start(struct upgrade_server_info *server) . . . 126
4.23.4.8 system_upgrade_userbin_check(void) . . . 126
4.24 GPIO Driver APIs . . . 127
4.24.1 Detailed Description . . . 127

4.24.2 Macro Definition Documentation . . . 127
4.24.2.1 GPIO_AS_INPUT . . . 127
4.24.2.2 GPIO_AS_OUTPUT . . . 128
4.24.2.3 GPIO_DIS_OUTPUT . . . 128
4.24.2.4 GPIO_INPUT_GET . . . 128
4.24.2.5 GPIO_OUTPUT . . . 128
4.24.2.6 GPIO_OUTPUT_SET . . . 129
4.24.3 Function Documentation . . . 129
4.24.3.1 gpio16_input_conf(void) . . . 129
4.24.3.2 gpio16_input_get(void) . . . 129
4.24.3.3 gpio16_output_conf(void) . . . 129
4.24.3.4 gpio16_output_set(uint8 value) . . . 130
4.24.3.5 gpio_input_get(void) . . . 130
4.24.3.6 gpio_intr_handler_register(void ∗fn, void ∗arg) . . . 130
4.24.3.7 gpio_output_conf(uint32 set_mask, uint32 clear_mask, uint32 enable_mask, uint32 disable_mask) . . . 130
4.24.3.8 gpio_pin_intr_state_set(uint32 i, GPIO_INT_TYPE intr_state) . . . 131
4.24.3.9 gpio_pin_wakeup_disable() . . . 131
4.24.3.10 gpio_pin_wakeup_enable(uint32 i, GPIO_INT_TYPE intr_state) . . . 131
4.25 Hardware timer APIs . . . 132
4.25.1 Detailed Description . . . 132
4.25.2 Function Documentation . . . 132
4.25.2.1 hw_timer_arm(uint32 val) . . . 132
4.25.2.2 hw_timer_init(uint8 req) . . . 132
4.25.2.3 hw_timer_set_func(void(∗user_hw_timer_cb_set)(void)) . . . 132
4.26 UART Driver APIs . . . 134
4.26.1 Detailed Description . . . 134
4.26.2 Function Documentation . . . 134
4.26.2.1 UART_ClearIntrStatus(UART_Port uart_no, uint32 clr_mask) . . . 134
4.26.2.2 uart_init_new(void) . . . 135
4.26.2.3 UART_intr_handler_register(void ∗fn, void ∗arg) . . . 135
4.26.2.4 UART_IntrConfig(UART_Port uart_no, UART_IntrConfTypeDef ∗pUARTIntrConf) 135
4.26.2.5 UART_ParamConfig(UART_Port uart_no, UART_ConfigTypeDef ∗pUARTConfig) 135
4.26.2.6 UART_ResetFifo(UART_Port uart_no) . . . 136
4.26.2.7 UART_SetBaudrate(UART_Port uart_no, uint32 baud_rate) . . . 136
4.26.2.8 UART_SetFlowCtrl(UART_Port uart_no, UART_HwFlowCtrl flow_ctrl, uint8 rx_↩ thresh) . . . 136
4.26.2.9 UART_SetIntrEna(UART_Port uart_no, uint32 ena_mask) . . . 136
4.26.2.10 UART_SetLineInverse(UART_Port uart_no, UART_LineLevelInverse inverse_↩ mask) . . . 137
4.26.2.11 UART_SetParity(UART_Port uart_no, UART_ParityMode Parity_mode) . . . 137

4.26.2.12 UART_SetPrintPort(UART_Port uart_no) . . . 137
4.26.2.13 UART_SetStopBits(UART_Port uart_no, UART_StopBits bit_num) . . . 137
4.26.2.14 UART_SetWordLength(UART_Port uart_no, UART_WordLength len) . . . 138
4.26.2.15 UART_WaitTxFifoEmpty(UART_Port uart_no) . . . 138

5 Data Structure Documentation **139**

5.1 _esp_event Struct Reference . . . 139
5.1.1 Field Documentation . . . 139
5.1.1.1 event_id . . . 139
5.1.1.2 event_info . . . 139
5.2 _esp_tcp Struct Reference . . . 139
5.2.1 Field Documentation . . . 140
5.2.1.1 connect_callback . . . 140
5.2.1.2 disconnect_callback . . . 140
5.2.1.3 local_ip . . . 140
5.2.1.4 local_port . . . 140
5.2.1.5 reconnect_callback . . . 140
5.2.1.6 remote_ip . . . 140
5.2.1.7 remote_port . . . 140
5.2.1.8 write_finish_fn . . . 140
5.3 _esp_udp Struct Reference . . . 140
5.3.1 Field Documentation . . . 141
5.3.1.1 local_ip . . . 141
5.3.1.2 local_port . . . 141
5.3.1.3 remote_ip . . . 141
5.3.1.4 remote_port . . . 141
5.4 _os_timer_t Struct Reference . . . 141
5.5 _remot_info Struct Reference . . . 141
5.5.1 Field Documentation . . . 141
5.5.1.1 remote_ip . . . 141
5.5.1.2 remote_port . . . 142
5.5.1.3 state . . . 142
5.6 bss_info Struct Reference . . . 142
5.6.1 Member Function Documentation . . . 142
5.6.1.1 STAILQ_ENTRY(bss_info) next . . . 142
5.6.2 Field Documentation . . . 142
5.6.2.1 authmode . . . 142
5.6.2.2 bssid . . . 142
5.6.2.3 channel . . . 142
5.6.2.4 freq_offset . . . 143

5.6.2.5 is_hidden . . . 143
5.6.2.6 rssi . . . 143
5.6.2.7 ssid . . . 143
5.6.2.8 ssid_len . . . 143
5.7 cmd_s Struct Reference . . . 143
5.8 dhcps_lease Struct Reference . . . 143
5.8.1 Field Documentation . . . 143
5.8.1.1 enable . . . 143
5.8.1.2 end_ip . . . 144
5.8.1.3 start_ip . . . 144
5.9 esp_spiffs_config Struct Reference . . . 144
5.9.1 Field Documentation . . . 144
5.9.1.1 cache_buf_size . . . 144
5.9.1.2 fd_buf_size . . . 144
5.9.1.3 log_block_size . . . 144
5.9.1.4 log_page_size . . . 144
5.9.1.5 phys_addr . . . 144
5.9.1.6 phys_erase_block . . . 144
5.9.1.7 phys_size . . . 145
5.10 espconn Struct Reference . . . 145
5.10.1 Detailed Description . . . 145
5.10.2 Field Documentation . . . 145
5.10.2.1 link_cnt . . . 145
5.10.2.2 recv_callback . . . 145
5.10.2.3 reserve . . . 145
5.10.2.4 sent_callback . . . 145
5.10.2.5 state . . . 145
5.10.2.6 type . . . 146
5.11 Event_Info_u Union Reference . . . 146
5.11.1 Field Documentation . . . 146
5.11.1.1 ap_probereqrecved . . . 146
5.11.1.2 auth_change . . . 146
5.11.1.3 connected . . . 146
5.11.1.4 disconnected . . . 146
5.11.1.5 got_ip . . . 146
5.11.1.6 scan_done . . . 146
5.11.1.7 sta_connected . . . 146
5.11.1.8 sta_disconnected . . . 147
5.12 Event_SoftAPMode_ProbeReqRecved_t Struct Reference . . . 147
5.12.1 Field Documentation . . . 147

5.12.1.1 mac . . . 147
5.12.1.2 rssi . . . 147
5.13 Event_SoftAPMode_StaConnected_t Struct Reference . . . 147
5.13.1 Field Documentation . . . 147
5.13.1.1 aid . . . 147
5.13.1.2 mac . . . 147
5.14 Event_SoftAPMode_StaDisconnected_t Struct Reference . . . 148
5.14.1 Field Documentation . . . 148
5.14.1.1 aid . . . 148
5.14.1.2 mac . . . 148
5.15 Event_StaMode_AuthMode_Change_t Struct Reference . . . 148
5.15.1 Field Documentation . . . 148
5.15.1.1 new_mode . . . 148
5.15.1.2 old_mode . . . 148
5.16 Event_StaMode_Connected_t Struct Reference . . . 148
5.16.1 Field Documentation . . . 149
5.16.1.1 bssid . . . 149
5.16.1.2 channel . . . 149
5.16.1.3 ssid . . . 149
5.16.1.4 ssid_len . . . 149
5.17 Event_StaMode_Disconnected_t Struct Reference . . . 149
5.17.1 Field Documentation . . . 149
5.17.1.1 bssid . . . 149
5.17.1.2 reason . . . 149
5.17.1.3 ssid . . . 149
5.17.1.4 ssid_len . . . 149
5.18 Event_StaMode_Got_IP_t Struct Reference . . . 150
5.18.1 Field Documentation . . . 150
5.18.1.1 gw . . . 150
5.18.1.2 ip . . . 150
5.18.1.3 mask . . . 150
5.19 Event_StaMode_ScanDone_t Struct Reference . . . 150
5.19.1 Field Documentation . . . 150
5.19.1.1 bss . . . 150
5.19.1.2 status . . . 150
5.20 GPIO_ConfigTypeDef Struct Reference . . . 150
5.20.1 Field Documentation . . . 151
5.20.1.1 GPIO_IntrType . . . 151
5.20.1.2 GPIO_Mode . . . 151
5.20.1.3 GPIO_Pin . . . 151

5.20.1.4 GPIO_Pullup . . . 151
5.21 ip_info Struct Reference . . . 151
5.21.1 Field Documentation . . . 151
5.21.1.1 gw . . . 151
5.21.1.2 ip . . . 151
5.21.1.3 netmask . . . 151
5.22 pwm_param Struct Reference . . . 152
5.22.1 Field Documentation . . . 152
5.22.1.1 duty . . . 152
5.22.1.2 freq . . . 152
5.22.1.3 period . . . 152
5.23 rst_info Struct Reference . . . 152
5.23.1 Field Documentation . . . 152
5.23.1.1 reason . . . 152
5.24 scan_config Struct Reference . . . 153
5.24.1 Field Documentation . . . 153
5.24.1.1 bssid . . . 153
5.24.1.2 channel . . . 153
5.24.1.3 show_hidden . . . 153
5.24.1.4 ssid . . . 153
5.25 softap_config Struct Reference . . . 153
5.25.1 Field Documentation . . . 153
5.25.1.1 authmode . . . 153
5.25.1.2 beacon_interval . . . 154
5.25.1.3 channel . . . 154
5.25.1.4 max_connection . . . 154
5.25.1.5 password . . . 154
5.25.1.6 ssid . . . 154
5.25.1.7 ssid_hidden . . . 154
5.25.1.8 ssid_len . . . 154
5.26 station_config Struct Reference . . . 154
5.26.1 Field Documentation . . . 154
5.26.1.1 bssid . . . 154
5.26.1.2 bssid_set . . . 155
5.26.1.3 password . . . 155
5.26.1.4 ssid . . . 155
5.27 station_info Struct Reference . . . 155
5.27.1 Member Function Documentation . . . 155
5.27.1.1 STAILQ_ENTRY(station_info) next . . . 155
5.27.2 Field Documentation . . . 155

5.27.2.1 bssid . . . 155
5.27.2.2 ip . . . 155
5.28 UART_ConfigTypeDef Struct Reference . . . 156
5.29 UART_IntrConfTypeDef Struct Reference . . . 156
5.30 upgrade_server_info Struct Reference . . . 156
5.30.1 Field Documentation . . . 156
5.30.1.1 check_cb . . . 156
5.30.1.2 check_times . . . 156
5.30.1.3 pre_version . . . 157
5.30.1.4 sockaddrin . . . 157
5.30.1.5 upgrade_flag . . . 157
5.30.1.6 upgrade_version . . . 157
5.30.1.7 url . . . 157

Chapter 1

esp_iot_rtos_sdk

- Misc APIs : misc APIs
- WiFi APIs : WiFi related APIs
  - SoftAP APIs : ESP8266 Soft-AP APIs
  - Station APIs : ESP8266 station APIs
  - Common APIs : WiFi common APIs
  - Force Sleep APIs : WiFi Force Sleep APIs
  - Rate Control APIs : WiFi Rate Control APIs
  - User IE APIs : WiFi User IE APIs
  - Sniffer APIs : WiFi sniffer APIs
  - WPS APIs : WiFi WPS APIs
  - Smartconfig APIs : SmartConfig APIs
- Spiffs APIs : Spiffs APIs
- SSC APIs : Simple Serial Command APIs
- System APIs : System APIs
  - Boot APIs : Boot mode APIs
  - Upgrade APIs : Firmware upgrade (FOTA) APIs
- Software timer APIs : Software timer APIs
- Network Espconn APIs : Network espconn APIs
- ESP-NOW APIs : ESP-NOW APIs
- Mesh APIs : Mesh APIs
- Driver APIs : Driver APIs
  - PWM Driver APIs : PWM driver APIs
  - UART Driver APIs : UART driver APIs
  - GPIO Driver APIs : GPIO driver APIs
  - SPI Driver APIs : SPI Flash APIs
  - Hardware timer APIs : Hardware timer 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. Since esp_iot_rtos_sdk_v1.0.4, functions are stored in CACHE by default, need not be added 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. Network programming use socket, please do not bind to the same port.
(1). If users want to create 3 or more than 3 TCP connection, please add "TCP_WND = 2 x TCP_MSS;" in "user_init".
4. 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:

- Misc APIs . . . 7
- WiFi Related APIs . . . 10
  - SoftAP APIs . . . 11
  - Station APIs . . . 20
  - Common APIs . . . 46
  - Force Sleep APIs . . . 58
  - Rate Control APIs . . . 61
  - User IE APIs . . . 65
  - Sniffer APIs . . . 68
  - WPS APIs . . . 71
  - Smartconfig APIs . . . 113
- Spiffs APIs . . . 17
- SSC APIs . . . 18
- System APIs . . . 29
  - Boot APIs . . . 40
  - Upgrade APIs . . . 121
- Software timer APIs . . . 44
- Network Espconn APIs . . . 74
- ESP-NOW APIs . . . 92
- Mesh APIs . . . 102
- Driver APIs . . . 109
  - PWM Driver APIs . . . 110
  - SPI Driver APIs . . . 117
  - GPIO Driver APIs . . . 127
  - Hardware timer APIs . . . 132
  - UART Driver APIs . . . 134

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

_esp_event . . . 139
_esp_tcp . . . 139
_esp_udp . . . 140
_os_timer_t . . . 141
_remot_info . . . 141
bss_info . . . 142
cmd_s . . . 143
dhcps_lease . . . 143
esp_spiffs_config . . . 144
espconn . . . 145
Event_Info_u . . . 146
Event_SoftAPMode_ProbeReqRecved_t . . . 147
Event_SoftAPMode_StaConnected_t . . . 147
Event_SoftAPMode_StaDisconnected_t . . . 148
Event_StaMode_AuthMode_Change_t . . . 148
Event_StaMode_Connected_t . . . 148
Event_StaMode_Disconnected_t . . . 149
Event_StaMode_Got_IP_t . . . 150
Event_StaMode_ScanDone_t . . . 150
GPIO_ConfigTypeDef . . . 150
ip_info . . . 151
pwm_param . . . 152
rst_info . . . 152
scan_config . . . 153
softap_config . . . 153
station_config . . . 154
station_info . . . 155
UART_ConfigTypeDef . . . 156
UART_IntrConfTypeDef . . . 156
upgrade_server_info . . . 156

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"

Enumerations

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

Functions

- void os_delay_us (uint16 us)

  Delay function, maximum value: 65535 us.
- void os_install_putc1 (void(∗p)(char c))

  Register the print output function.
- void os_putc (char c)

  Print a character. Start from from UART0 by default.

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)
```

4.1.3 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

4.1.4 Function Documentation

4.1.4.1 void os_delay_us (uint16 *us*)

Delay function, maximum value: 65535 us.

Parameters

uint16	us : delay time, uint: us, maximum value: 65535 us

Returns

null

4.1.4.2 void os_install_putc1 (void(*)(char c) *p*)

Register the print output function.

Attention

os_install_putc1((void *)uart1_write_char) in uart_init will set printf to print from UART 1, otherwise, printf will start from UART 0 by default.

Parameters

void(∗p)(char	c) - pointer of print function

Returns

null

4.1.4.3 void os_putc (char *c*)

Print a character. Start from from UART0 by default.

Parameters

char	c - character to be printed

Returns

null

4.2 WiFi Related APIs

WiFi APIs.

Modules

- SoftAP APIs

  ESP8266 Soft-AP APIs.
- Station APIs

  ESP8266 station APIs.
- Common APIs

  WiFi common APIs.
- Force Sleep APIs

  WiFi Force Sleep APIs.
- Rate Control APIs

  WiFi Rate Control APIs.
- User IE APIs

  WiFi User IE APIs.
- Sniffer APIs

  WiFi sniffer APIs.
- WPS APIs

  ESP8266 WPS APIs.
- Smartconfig APIs

  SmartConfig APIs.

4.2.1 Detailed Description

WiFi APIs.

4.3 SoftAP APIs

ESP8266 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 ESP8266 WiFi soft-AP.
- bool wifi_softap_get_config_default (struct softap_config *config)

  Get the configuration of the ESP8266 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 ESP8266 soft-AP.
- struct station_info * wifi_softap_get_station_info (void)

  Get the information of stations connected to the ESP8266 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 ESP8266 soft-AP DHCP server.
- bool wifi_softap_dhcps_stop (void)

  Disable the ESP8266 soft-AP DHCP server. The DHCP is enabled by default.
- enum dhcp_status wifi_softap_dhcps_status (void)

  Get the ESP8266 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 ESP8266 soft-AP DHCP server.
- bool wifi_softap_set_dhcps_lease (struct dhcps_lease *please)

  Set the IP range of the ESP8266 soft-AP DHCP server.
- uint32 wifi_softap_get_dhcps_lease_time (void)

  Get ESP8266 soft-AP DHCP server lease time.
- bool wifi_softap_set_dhcps_lease_time (uint32 minute)

  Set ESP8266 soft-AP DHCP server lease time, default is 120 minutes.
- bool wifi_softap_reset_dhcps_lease_time (void)

  Reset ESP8266 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 ESP8266 soft-AP DHCP server option.

4.3.1 Detailed Description

ESP8266 Soft-AP APIs.

Attention

To call APIs related to ESP8266 soft-AP has to enable soft-AP mode first (wifi_set_opmode)

4.3.2 Function Documentation

4.3.2.1 bool wifi_softap_dhcps_start (void)

Enable the ESP8266 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

4.3.2.2 enum dhcp_status wifi_softap_dhcps_status (void)

Get the ESP8266 soft-AP DHCP server status.

Parameters

null	

Returns

enum dhcp_status

4.3.2.3 bool wifi_softap_dhcps_stop (void)

Disable the ESP8266 soft-AP DHCP server. The DHCP is enabled by default.

Parameters

null	

Returns

true : succeed
false : fail

4.3.2.4 void wifi_softap_free_station_info (void)

Free the space occupied by station_info when wifi_softap_get_station_info is called.

Attention

The ESP8266 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 ESP8266 station.

Parameters

null	

Returns

null

4.3.2.5 bool wifi_softap_get_config (struct softap_config * *config*)

Get the current configuration of the ESP8266 WiFi soft-AP.

Parameters

struct	softap_config *config : ESP8266 soft-AP configuration

Returns

true : succeed
false : fail

4.3.2.6 bool wifi_softap_get_config_default (struct softap_config * *config*)

Get the configuration of the ESP8266 WiFi soft-AP saved in the flash.

Parameters

struct	softap_config *config : ESP8266 soft-AP configuration

Returns

true : succeed
false : fail

4.3.2.7 bool wifi_softap_get_dhcps_lease (struct dhcps_lease * *please*)

Query the IP range that can be got from the ESP8266 soft-AP DHCP server.

Attention

This API can only be called during ESP8266 soft-AP DHCP server enabled.

Parameters

struct	dhcps_lease *please : IP range of the ESP8266 soft-AP DHCP server.

Returns

true : succeed
false : fail

4.3.2.8 uint32 wifi_softap_get_dhcps_lease_time (void)

Get ESP8266 soft-AP DHCP server lease time.

Attention

This API can only be called during ESP8266 soft-AP DHCP server enabled.

Parameters

null	

Returns

lease time, uint: minute.

4.3.2.9 struct station_info* wifi_softap_get_station_info (void)

Get the information of stations connected to the ESP8266 soft-AP, including MAC and IP.

Attention

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

Parameters

null	

Returns

struct station_info* : station information structure

4.3.2.10 uint8 wifi_softap_get_station_num (void)

Get the number of stations connected to the ESP8266 soft-AP.

Attention

The ESP8266 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 ESP8266 station.

Parameters

null	

Returns

the number of stations connected to the ESP8266 soft-AP

4.3.2.11 bool wifi_softap_reset_dhcps_lease_time (void)

Reset ESP8266 soft-AP DHCP server lease time which is 120 minutes by default.

Attention

This API can only be called during ESP8266 soft-AP DHCP server enabled.

Parameters

null	

Returns

true : succeed
false : fail

4.3.2.12 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 ESP8266 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 ESP8266 station.

Parameters

struct	softap_config *config : ESP8266 soft-AP configuration

Returns

true : succeed
false : fail

4.3.2.13 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 ESP8266 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 ESP8266 station.

Parameters

struct	softap_config *config : ESP8266 soft-AP configuration

Returns

true : succeed
false : fail

4.3.2.14 bool wifi_softap_set_dhcps_lease (struct dhcps_lease * *please*)

Set the IP range of the ESP8266 soft-AP DHCP server.

Attention

1. The IP range should be in the same sub-net with the ESP8266 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 ESP8266 soft-AP DHCP server.

Returns

true : succeed
false : fail

4.3.2.15 bool wifi_softap_set_dhcps_lease_time (uint32 *minute*)

Set ESP8266 soft-AP DHCP server lease time, default is 120 minutes.

Attention

This API can only be called during ESP8266 soft-AP DHCP server enabled.

Parameters

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

Returns

true : succeed
false : fail

4.3.2.16 bool wifi_softap_set_dhcps_offer_option (uint8 *level,* void * *optarg*)

Set the ESP8266 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

4.4 Spiffs APIs

Spiffs APIs.

Data Structures

- struct esp_spiffs_config

Functions

- sint32 esp_spiffs_init (struct esp_spiffs_config *config)

  Initialize spiffs.
- void esp_spiffs_deinit (uint8 format)

  Deinitialize spiffs.

4.4.1 Detailed Description

Spiffs APIs.

More details about spiffs on https://github.com/pellepl/spiffs

4.4.2 Function Documentation

4.4.2.1 void esp_spiffs_deinit (uint8 *format*)

Deinitialize spiffs.

Parameters

uint8	format : 0, only deinit; otherwise, deinit spiffs and format.

Returns

null

4.4.2.2 sint32 esp_spiffs_init (struct esp_spiffs_config * *config*)

Initialize spiffs.

Parameters

struct	esp_spiffs_config *config : ESP8266 spiffs configuration

Returns

0 : succeed
otherwise : fail

4.5 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.

4.5.1 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.

4.5.2 Function Documentation

4.5.2.1 void ssc_attach (SscBaudRate *bandrate*)

Initial the ssc function.

Parameters

SscBaudRate	bandrate : baud rate

Returns

null

4.5.2.2 int ssc_param_len (void)

Get the length of the simple serial command.

Parameters

null	

Returns

length of the command.

4.5.2.3 char* ssc_param_str (void)

Get the simple serial command string.

Parameters

null	

Returns

the command.

4.5.2.4 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.

4.5.2.5 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

4.6 Station APIs

ESP8266 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 ESP8266 WiFi station.
- bool wifi_station_get_config_default (struct station_config *config)

  Get the configuration parameters saved in the Flash of the ESP8266 WiFi station.
- bool wifi_station_set_config (struct station_config *config)

  Set the configuration of the ESP8266 station and save it to the Flash.
- bool wifi_station_set_config_current (struct station_config *config)

  Set the configuration of the ESP8266 station. And the configuration will not be saved to the Flash.
- bool wifi_station_connect (void)

  Connect the ESP8266 WiFi station to the AP.
- bool wifi_station_disconnect (void)

  Disconnect the ESP8266 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 ESP8266 station will connect to the recorded AP automatically when the power is on.
- bool wifi_station_set_auto_connect (bool set)

  Set whether the ESP8266 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 ESP8266 station will reconnect to the AP after disconnection.
- bool wifi_station_set_reconnect_policy (bool set)

  Set whether the ESP8266 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 ESP8266 WiFi station.
- uint8 wifi_station_get_current_ap_id (void)

Get the information of APs (5 at most) recorded by ESP8266 station.

- bool wifi_station_ap_change (uint8 current_ap_id)

  Switch the ESP8266 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 ESP8266 station. When the ESP8266 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 ESP8266 station.

- sint8 wifi_station_get_rssi (void)

  Get rssi of the AP which ESP8266 station connected to.

- bool wifi_station_dhcpc_start (void)

  Enable the ESP8266 station DHCP client.

- bool wifi_station_dhcpc_stop (void)

  Disable the ESP8266 station DHCP client.

- enum dhcp_status wifi_station_dhcpc_status (void)

  Get the ESP8266 station DHCP client status.

4.6.1 Detailed Description

ESP8266 station APIs.

Attention

To call APIs related to ESP8266 station has to enable station mode first (wifi_set_opmode)

4.6.2 Typedef Documentation

4.6.2.1 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

4.6.3 Enumeration Type Documentation

4.6.3.1 enum STATION_STATUS

Enumerator

STATION_IDLE ESP8266 station idle

STATION_CONNECTING ESP8266 station is connecting to AP

STATION_WRONG_PASSWORD the password is wrong

STATION_NO_AP_FOUND ESP8266 station can not find the target AP

STATION_CONNECT_FAIL ESP8266 station fail to connect to AP

STATION_GOT_IP ESP8266 station got IP address from AP

4.6.4 Function Documentation

4.6.4.1 bool wifi_station_ap_change (uint8 *current_ap_id*)

Switch the ESP8266 station connection to a recorded AP.

Parameters

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

Returns

true : succeed
false : fail

4.6.4.2 bool wifi_station_ap_number_set (uint8 *ap_number*)

Set the number of APs that can be recorded in the ESP8266 station. When the ESP8266 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

4.6.4.3 bool wifi_station_connect (void)

Connect the ESP8266 WiFi station to the AP.

Attention

1. This API should be called when the ESP8266 station is enabled, and the system initialization is completed. Do not call this API in user_init.
2. If the ESP8266 is connected to an AP, call wifi_station_disconnect to disconnect.

Parameters

null	

Returns

true : succeed
false : fail

4.6.4.4 bool wifi_station_dhcpc_start (void)

Enable the ESP8266 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

4.6.4.5 enum dhcp_status wifi_station_dhcpc_status (void)

Get the ESP8266 station DHCP client status.

Parameters

null	

Returns

enum dhcp_status

4.6.4.6 bool wifi_station_dhcpc_stop (void)

Disable the ESP8266 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

4.6.4.7 bool wifi_station_disconnect (void)

Disconnect the ESP8266 WiFi station from the AP.

Attention

This API should be called when the ESP8266 station is enabled, and the system initialization is completed. Do not call this API in user_init.

Parameters

null	

Returns

true : succeed
false : fail

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

Get the information of APs (5 at most) recorded by ESP8266 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.

4.6.4.9 bool wifi_station_get_auto_connect (void)

Check if the ESP8266 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

4.6.4.10 bool wifi_station_get_config (struct station_config * *config*)

Get the current configuration of the ESP8266 WiFi station.

Parameters

struct	station_config *config : ESP8266 station configuration

Returns

true : succeed
false : fail

4.6.4.11 bool wifi_station_get_config_default (struct station_config * *config*)

Get the configuration parameters saved in the Flash of the ESP8266 WiFi station.

Parameters

struct	station_config *config : ESP8266 station configuration

Returns

true : succeed
false : fail

4.6.4.12 STATION_STATUS wifi_station_get_connect_status (void)

Get the connection status of the ESP8266 WiFi station.

Parameters

null	

Returns

the status of connection

4.6.4.13 uint8 wifi_station_get_current_ap_id (void)

Get the information of APs (5 at most) recorded by ESP8266 station.

Parameters

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

Returns

The number of APs recorded.

4.6.4.14 bool wifi_station_get_reconnect_policy (void)

Check whether the ESP8266 station will reconnect to the AP after disconnection.

Parameters

null	

Returns

true : succeed
false : fail

4.6.4.15 sint8 wifi_station_get_rssi (void)

Get rssi of the AP which ESP8266 station connected to.

Parameters

null	

Returns

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

4.6.4.16 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 ESP8266 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

4.6.4.17 bool wifi_station_set_auto_connect (bool *set*)

Set whether the ESP8266 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

4.6.4.18 bool wifi_station_set_config (struct station_config * *config*)

Set the configuration of the ESP8266 station and save it to the Flash.

Attention

1. This API can be called only when the ESP8266 station is enabled.
2. If wifi_station_set_config is called in user_init , there is no need to call wifi_station_connect. The ESP8266 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 : ESP8266 station configuration

Returns

true : succeed
false : fail

4.6.4.19 bool wifi_station_set_config_current (struct station_config * *config*)

Set the configuration of the ESP8266 station. And the configuration will not be saved to the Flash.

Attention

1. This API can be called only when the ESP8266 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 ESP8266 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 : ESP8266 station configuration

Returns

true : succeed
false : fail

4.6.4.20 bool wifi_station_set_reconnect_policy (bool *set*)

Set whether the ESP8266 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

4.7 System APIs

System APIs.

Modules

- Boot APIs

  boot APIs
- Upgrade APIs

  Firmware upgrade (FOTA) APIs.

Data Structures

- struct rst_info

Enumerations

- enum rst_reason {
  REASON_DEFAULT_RST = 0, REASON_WDT_RST, REASON_EXCEPTION_RST, REASON_SOFT_W↩
  DT_RST,
  REASON_SOFT_RESTART, REASON_DEEP_SLEEP_AWAKE, REASON_EXT_SYS_RST }

Functions

- struct rst_info * system_get_rst_info (void)

  Get the reason of restart.
- 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 (uint32 time_in_us)

  Set the chip to deep-sleep mode.
- bool system_deep_sleep_set_option (uint8 option)

  Call this API before system_deep_sleep to set the activity after the next deep-sleep wakeup.
- 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.
- uint32 system_get_chip_id (void)

  Get the chip ID.
- uint32 system_rtc_clock_cali_proc (void)

  Get the RTC clock cycle.
- uint32 system_get_rtc_time (void)

  Get RTC time, unit: RTC clock cycle.
- bool system_rtc_mem_read (uint8 src, void *dst, uint16 n)

  Read user data from the RTC memory.

- bool system_rtc_mem_write (uint8 dst, const void *src, uint16 n)
  Write user data to the RTC memory.
- void system_uart_swap (void)
  UART0 swap.
- void system_uart_de_swap (void)
  Disable UART0 swap.
- uint16 system_adc_read (void)
  Measure the input voltage of TOUT pin 6, unit : 1/1024 V.
- 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.
- void system_phy_set_max_tpw (uint8 max_tpw)
  Set the maximum value of RF TX Power, unit : 0.25dBm.
- void system_phy_set_tpw_via_vdd33 (uint16 vdd33)
  Adjust the RF TX Power according to VDD33, unit : 1/1024 V.
- void system_phy_set_rfoption (uint8 option)
  Enable RF or not when wakeup from deep-sleep.

4.7.1 Detailed Description

System APIs.

4.7.2 Enumeration Type Documentation

4.7.2.1 enum rst_reason

Enumerator

REASON_DEFAULT_RST normal startup by power on
REASON_WDT_RST hardware watch dog reset
REASON_EXCEPTION_RST exception reset, GPIO status won't change
REASON_SOFT_WDT_RST software watch dog reset, GPIO status won't change
REASON_SOFT_RESTART software restart ,system_restart , GPIO status won't change
REASON_DEEP_SLEEP_AWAKE wake up from deep-sleep
REASON_EXT_SYS_RST external system reset

4.7.3 Function Documentation

4.7.3.1 uint16 system_adc_read (void)

Measure the input voltage of TOUT pin 6, unit : 1/1024 V.

Attention

1. system_adc_read can only be called when the TOUT pin is connected to the external circuitry, and the TOUT pin input voltage should be limited to 0∼1.0V.
2. When the TOUT pin is connected to the external circuitry, the 107th byte (vdd33_const) of esp_init_data↩_default.bin(0∼127byte) should be set as the real power voltage of VDD3P3 pin 3 and 4.
3. The unit of vdd33_const is 0.1V, the effective value range is [18, 36]; if vdd33_const is in [0, 18) or (36, 255), 3.3V is used to optimize RF by default.

Parameters

null	

Returns

Input voltage of TOUT pin 6, unit : 1/1024 V

4.7.3.2 void system_deep_sleep (uint32 *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

1. XPD_DCDC should be connected to EXT_RSTB through 0 ohm resistor in order to support deep-sleep wakeup.
2. system_deep_sleep(0): there is no wake up timer; in order to wake up, connect a GPIO to pin RST, the chip will wake up by a falling-edge on pin RST

Parameters

uint32	time_in_us : deep-sleep time, unit: microsecond

Returns

null

4.7.3.3 bool system_deep_sleep_set_option (uint8 *option*)

Call this API before system_deep_sleep to set the activity after the next deep-sleep wakeup.

If this API is not called, default to be system_deep_sleep_set_option(1).

Parameters

uint8	option :
0	: Radio calibration after the deep-sleep wakeup is decided by byte 108 of esp_init_data_↩ default.bin (0∼127byte).
1	: Radio calibration will be done after the deep-sleep wakeup. This will lead to stronger current.
2	: Radio calibration will not be done after the deep-sleep wakeup. This will lead to weaker current.
4	: Disable radio calibration after the deep-sleep wakeup (the same as modem-sleep). This will lead to the weakest current, but the device can't receive or transmit data after waking up.

Returns

true : succeed
false : fail

4.7.3.4 uint32 system_get_chip_id (void)

Get the chip ID.

Parameters

null	

Returns

The chip ID.

4.7.3.5 uint32 system_get_free_heap_size (void)

Get the size of available heap.

Parameters

null	

Returns

Available heap size.

4.7.3.6 struct rst_info* system_get_rst_info (void)

Get the reason of restart.

Parameters

null	

Returns

struct rst_info* : information of the system restart

4.7.3.7 uint32 system_get_rtc_time (void)

Get RTC time, unit: RTC clock cycle.

Example: If system_get_rtc_time returns 10 (it means 10 RTC cycles), and system_rtc_clock_cali_proc returns 5.75 (it means 5.75 microseconds per RTC clock cycle), (then the actual time is 10 x 5.75 = 57.5 microseconds.

Attention

System time will return to zero because of system_restart, but the RTC time still goes on. If the chip is reset by pin EXT_RST or pin CHIP_EN (including the deep-sleep wakeup), situations are shown as below:
1. reset by pin EXT_RST : RTC memory won't change, RTC timer returns to zero
2. watchdog reset : RTC memory won't change, RTC timer won't change
3. system_restart : RTC memory won't change, RTC timer won't change
4. power on : RTC memory is random value, RTC timer starts from zero
5. reset by pin CHIP_EN : RTC memory is random value, RTC timer starts from zero

Parameters

null	

Returns

RTC time.

4.7.3.8 const char* system_get_sdk_version (void)

Get information of the SDK version.

Parameters

null	

Returns

Information of the SDK version.

4.7.3.9 uint32 system_get_time (void)

Get system time, unit: microsecond.

Parameters

null	

Returns

System time, unit: microsecond.

4.7.3.10 uint16 system_get_vdd33 (void)

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

Attention

1. system_get_vdd33 can only be called when TOUT pin is suspended.
2. The 107th byte in esp_init_data_default.bin (0∼127byte) is named as "vdd33_const", when TOUT pin is suspended vdd33_const must be set as 0xFF, that is 255.

Parameters

null	

Returns

Power voltage of VDD33, unit : 1/1024 V

4.7.3.11 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

4.7.3.12 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

4.7.3.13 void system_phy_set_max_tpw (uint8 *max_tpw*)

Set the maximum value of RF TX Power, unit : 0.25dBm.

Parameters

uint8	max_tpw : the maximum value of RF Tx Power, unit : 0.25dBm, range [0, 82]. It can be set refer to the 34th byte (target_power_qdb_0) of esp_init_data_default.bin(0∼127byte)

Returns

null

4.7.3.14 void system_phy_set_rfoption (uint8 *option*)

Enable RF or not when wakeup from deep-sleep.

Attention

1. This API can only be called in user_rf_pre_init.
2. Function of this API is similar to system_deep_sleep_set_option, if they are both called, it will disregard system_deep_sleep_set_option which is called before deep-sleep, and refer to system_phy_set_rfoption which is called when deep-sleep wake up.
3. Before calling this API, system_deep_sleep_set_option should be called once at least.

Parameters

uint8	option : • 0 : Radio calibration after deep-sleep wake up depends on esp_init_data_default.bin (0∼127byte) byte 108. • 1 : Radio calibration is done after deep-sleep wake up; this increases the current consumption. • 2 : No radio calibration after deep-sleep wake up; this reduces the current consumption. • 4 : Disable RF after deep-sleep wake up, just like modem sleep; this has the least current consumption; the device is not able to transmit or receive data after wake up.

Returns

null

4.7.3.15 void system_phy_set_tpw_via_vdd33 (uint16 *vdd33*)

Adjust the RF TX Power according to VDD33, unit : 1/1024 V.

Attention

1. When TOUT pin is suspended, VDD33 can be measured by system_get_vdd33.
2. When TOUT pin is connected to the external circuitry, system_get_vdd33 can not be used to measure VDD33.

Parameters

uint16	vdd33 : VDD33, unit : 1/1024V, range [1900, 3300]

Returns

null

4.7.3.16 void system_print_meminfo (void)

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

Parameters

null	

Returns

null

4.7.3.17 void system_restart (void)

Restart system.

Parameters

null	

Returns

null

4.7.3.18 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

4.7.3.19 uint32 system_rtc_clock_cali_proc (void)

Get the RTC clock cycle.

Attention

1. The RTC clock cycle has decimal part.
2. The RTC clock cycle will change according to the temperature, so RTC timer is not very precise.

Parameters

null	

Returns

RTC clock period (unit: microsecond), bit11∼ bit0 are decimal.

4.7.3.20 bool system_rtc_mem_read (uint8 *src,* void * *dst,* uint16 *n*)

Read user data from the RTC memory.

The user data segment (512 bytes, as shown below) is used to store user data.

|<— system data(256 bytes) —>|<———- user data(512 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 256/4 = 64, n will be data length.

Parameters

uint8	src : source address of rtc memory, src_addr >= 64
void	*dst : data pointer
uint16	n : data length, unit: byte

Returns

true : succeed
false : fail

4.7.3.21 bool system_rtc_mem_write (uint8 *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 (512 bytes) is used to store the user data.

|<-— system data(256 bytes) -—>|<---------— user data(512 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 256/4 = 64, n will be data length.

Parameters

uint8	src : source address of rtc memory, src_addr >= 64
void	*dst : data pointer
uint16	n : data length, unit: byte

Returns

true : succeed
false : fail

4.7.3.22 void system_uart_de_swap (void)

Disable UART0 swap.

Use the original UART0, not MTCK and MTDO.

Parameters

null	

Returns

null

4.7.3.23 void system_uart_swap (void)

UART0 swap.

Use MTCK as UART0 RX, MTDO as UART0 TX, so ROM log will not output from this new UART0. We also need to use MTDO (U0CTS) and MTCK (U0RTS) as UART0 in hardware.

Parameters

null	

Returns

null

4.8 Boot APIs

boot APIs

Macros

- #define SYS_BOOT_ENHANCE_MODE 0
- #define SYS_BOOT_NORMAL_MODE 1
- #define SYS_BOOT_NORMAL_BIN 0
- #define SYS_BOOT_TEST_BIN 1
- #define **SYS_CPU_80MHZ** 80
- #define **SYS_CPU_160MHZ** 160

Enumerations

- enum flash_size_map {
  FLASH_SIZE_4M_MAP_256_256 = 0, FLASH_SIZE_2M, FLASH_SIZE_8M_MAP_512_512, FLASH_SIZE_16M_MAP_512_512,
  FLASH_SIZE_32M_MAP_512_512, FLASH_SIZE_16M_MAP_1024_1024, FLASH_SIZE_32M_MAP_1024_1024 }

Functions

- uint8 system_get_boot_version (void)

  Get information of the boot version.
- uint32 system_get_userbin_addr (void)

  Get the address of the current running user bin (user1.bin or user2.bin).
- uint8 system_get_boot_mode (void)

  Get the boot mode.
- bool system_restart_enhance (uint8 bin_type, uint32 bin_addr)

  Restarts the system, and enters the enhanced boot mode.
- flash_size_map system_get_flash_size_map (void)

  Get the current Flash size and Flash map.
- bool system_update_cpu_freq (uint8 freq)

  Set CPU frequency. Default is 80MHz.
- uint8 system_get_cpu_freq (void)

  Get CPU frequency.

4.8.1 Detailed Description

boot APIs

4.8.2 Macro Definition Documentation

4.8.2.1 #define SYS_BOOT_ENHANCE_MODE 0

It can load and run firmware at any address, for Espressif factory test bin

4.8.2.2 #define SYS_BOOT_NORMAL_BIN 0

user1.bin or user2.bin

4.8.2.3 #define SYS_BOOT_NORMAL_MODE 1

It can only load and run at some addresses of user1.bin (or user2.bin)

4.8.2.4 #define SYS_BOOT_TEST_BIN 1

can only be Espressif test bin

4.8.3 Enumeration Type Documentation

4.8.3.1 enum flash_size_map

Enumerator

FLASH_SIZE_4M_MAP_256_256 Flash size : 4Mbits. Map : 256KBytes + 256KBytes
FLASH_SIZE_2M Flash size : 2Mbits. Map : 256KBytes
FLASH_SIZE_8M_MAP_512_512 Flash size : 8Mbits. Map : 512KBytes + 512KBytes
FLASH_SIZE_16M_MAP_512_512 Flash size : 16Mbits. Map : 512KBytes + 512KBytes
FLASH_SIZE_32M_MAP_512_512 Flash size : 32Mbits. Map : 512KBytes + 512KBytes
FLASH_SIZE_16M_MAP_1024_1024 Flash size : 16Mbits. Map : 1024KBytes + 1024KBytes
FLASH_SIZE_32M_MAP_1024_1024 Flash size : 32Mbits. Map : 1024KBytes + 1024KBytes

4.8.4 Function Documentation

4.8.4.1 uint8 system_get_boot_mode (void)

Get the boot mode.

Parameters

null	

Returns

#define SYS_BOOT_ENHANCE_MODE 0
#define SYS_BOOT_NORMAL_MODE 1

4.8.4.2 uint8 system_get_boot_version (void)

Get information of the boot version.

Attention

If boot version >= 1.3 , users can enable the enhanced boot mode (refer to system_restart_enhance).

Parameters

null	

Returns

Information of the boot version.

4.8.4.3 uint8 system_get_cpu_freq (void)

Get CPU frequency.

Parameters

null	

Returns

CPU frequency, unit : MHz.

4.8.4.4 flash_size_map system_get_flash_size_map (void)

Get the current Flash size and Flash map.

Flash map depends on the selection when compiling, more details in document "2A-ESP8266__IOT_SDK_User↩ _Manual"

Parameters

null	

Returns

enum flash_size_map

4.8.4.5 uint32 system_get_userbin_addr (void)

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

Parameters

null	

Returns

The address of the current running user bin.

4.8.4.6 bool system_restart_enhance (uint8 *bin_type,* uint32 *bin_addr*)

Restarts the system, and enters the enhanced boot mode.

Attention

SYS_BOOT_TEST_BIN is used for factory test during production; users can apply for the test bin from Espressif Systems.

Parameters

uint8	bin_type : type of bin • #define SYS_BOOT_NORMAL_BIN 0 // user1.bin or user2.bin • #define SYS_BOOT_TEST_BIN 1 // can only be Espressif test bin

uint32	bin_addr : starting address of the bin file

Returns

true : succeed
false : fail

4.8.4.7 bool system_update_cpu_freq (uint8 *freq*)

Set CPU frequency. Default is 80MHz.

System bus frequency is 80MHz, will not be affected by CPU frequency. The frequency of UART, SPI, or other peripheral devices, are divided from system bus frequency, so they will not be affected by CPU frequency either.

Parameters

uint8	freq : CPU frequency, 80 or 160.

Returns

true : succeed
false : fail

4.9 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.

4.9.1 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).

4.9.2 Function Documentation

4.9.2.1 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

4.9.2.2 void os_timer_disarm (os_timer_t * *ptimer*)

Disarm the timer.

Parameters

os_timer_t	*ptimer : Timer structure

Returns

null

4.9.2.3 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

4.10 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.
- typedef void(∗ freedom_outside_cb_t) (uint8 status)

  Callback of sending user-define 802.11 packets.
- typedef void(∗ rfid_locp_cb_t) (uint8 ∗frm, int len, sint8 rssi)

  RFID LOCP (Location Control Protocol) receive callback .

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 WIFI_PHY_MODE { PHY_MODE_11B = 1, PHY_MODE_11G = 2, PHY_MODE_11N = 3 }
- 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 }

- enum **sleep_type** { **NONE_SLEEP_T** = 0, **LIGHT_SLEEP_T**, **MODEM_SLEEP_T** }

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.
- bool wifi_get_ip_info (WIFI_INTERFACE if_index, struct ip_info *info)

  Get the IP address of the ESP8266 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 ESP8266 WiFi station or the soft-AP interface.
- bool wifi_get_macaddr (WIFI_INTERFACE if_index, uint8 *macaddr)

  Get MAC address of the ESP8266 WiFi station or the soft-AP interface.
- bool wifi_set_macaddr (WIFI_INTERFACE if_index, uint8 *macaddr)

  Set MAC address of the ESP8266 WiFi station or the soft-AP interface.
- void wifi_status_led_install (uint8 gpio_id, uint32 gpio_name, uint8 gpio_func)

  Install the WiFi status LED.
- void wifi_status_led_uninstall (void)

  Uninstall the WiFi status LED.
- WIFI_PHY_MODE wifi_get_phy_mode (void)

  Get the ESP8266 physical mode (802.11b/g/n).
- bool wifi_set_phy_mode (WIFI_PHY_MODE mode)

  Set the ESP8266 physical mode (802.11b/g/n).
- bool wifi_set_event_handler_cb (wifi_event_handler_cb_t cb)

  Register the Wi-Fi event handler.
- sint32 wifi_register_send_pkt_freedom_cb (freedom_outside_cb_t cb)

  Register a callback for sending user-define 802.11 packets.
- void wifi_unregister_send_pkt_freedom_cb (void)

  Unregister the callback for sending user-define 802.11 packets.
- sint32 wifi_send_pkt_freedom (uint8 *buf, uint16 len, bool sys_seq)

  Send user-define 802.11 packets.
- sint32 wifi_rfid_locp_recv_open (void)

  Enable RFID LOCP (Location Control Protocol) to receive WDS packets.
- void wifi_rfid_locp_recv_close (void)

  Disable RFID LOCP (Location Control Protocol) .
- sint32 wifi_register_rfid_locp_recv_cb (rfid_locp_cb_t cb)

  Register a callback of receiving WDS packets.
- void wifi_unregister_rfid_locp_recv_cb (void)

  Unregister the callback of receiving WDS packets.
- bool wifi_set_sleep_type (sleep_type type)

  Sets sleep type.
- sleep_type wifi_get_sleep_type (void)

  Gets sleep type.

4.10.1 Detailed Description

WiFi common APIs.

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

4.10.2 Typedef Documentation

4.10.2.1 typedef void(* freedom_outside_cb_t) (uint8 status)

Callback of sending user-define 802.11 packets.

Parameters

uint8	status : 0, packet sending succeed; otherwise, fail.

Returns

null

4.10.2.2 typedef void(* rfid_locp_cb_t) (uint8 *frm, int len, sint8 rssi)

RFID LOCP (Location Control Protocol) receive callback .

Parameters

uint8	*frm : point to the head of 802.11 packet
int	len : packet length
int	rssi : signal strength

Returns

null

4.10.2.3 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

4.10.3 Enumeration Type Documentation

4.10.3.1 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

4.10.3.2 enum SYSTEM_EVENT

Enumerator

EVENT_STAMODE_SCAN_DONE ESP8266 station finish scanning AP
EVENT_STAMODE_CONNECTED ESP8266 station connected to AP
EVENT_STAMODE_DISCONNECTED ESP8266 station disconnected to AP
EVENT_STAMODE_AUTHMODE_CHANGE the auth mode of AP connected by ESP8266 station changed
EVENT_STAMODE_GOT_IP ESP8266 station got IP from connected AP
EVENT_STAMODE_DHCP_TIMEOUT ESP8266 station dhcp client got IP timeout
EVENT_SOFTAPMODE_STACONNECTED a station connected to ESP8266 soft-AP
EVENT_SOFTAPMODE_STADISCONNECTED a station disconnected to ESP8266 soft-AP
EVENT_SOFTAPMODE_PROBEREQRECVED Receive probe request packet in soft-AP interface

4.10.3.3 enum WIFI_INTERFACE

Enumerator

STATION_IF ESP8266 station interface
SOFTAP_IF ESP8266 soft-AP interface

4.10.3.4 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

4.10.3.5 enum WIFI_PHY_MODE

Enumerator

PHY_MODE_11B 802.11b
PHY_MODE_11G 802.11g
PHY_MODE_11N 802.11n

4.10.4 Function Documentation

4.10.4.1 bool wifi_get_ip_info (WIFI_INTERFACE *if_index*, struct ip_info * *info*)

Get the IP address of the ESP8266 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_INTERF↩ ACE	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

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

Get MAC address of the ESP8266 WiFi station or the soft-AP interface.

Parameters

WIFI_INTERF↩ ACE	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

4.10.4.3 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

4.10.4.4 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

4.10.4.5 WIFI_PHY_MODE wifi_get_phy_mode (void)

Get the ESP8266 physical mode (802.11b/g/n).

Parameters

null	

Returns

enum WIFI_PHY_MODE

4.10.4.6 sleep_type wifi_get_sleep_type (void)

Gets sleep type.

Parameters

null	

Returns

sleep type

4.10.4.7 sint32 wifi_register_rfid_locp_recv_cb (rfid_locp_cb_t *cb*)

Register a callback of receiving WDS packets.

Register a callback of receiving WDS packets. Only if the first MAC address of the WDS packet is a multicast address.

Parameters

rfid_locp_cb_t	cb : callback

Returns

0, succeed;
otherwise, fail.

4.10.4.8 sint32 wifi_register_send_pkt_freedom_cb (freedom_outside_cb_t *cb*)

Register a callback for sending user-define 802.11 packets.

Attention

Only after the previous packet was sent, entered the freedom_outside_cb_t, the next packet is allowed to send.

Parameters

freedom_↩ outside_cb_t	cb : sent callback

Returns

0, succeed;
-1, fail.

4.10.4.9 void wifi_rfid_locp_recv_close (void)

Disable RFID LOCP (Location Control Protocol) .

Parameters

null	

Returns

null

4.10.4.10 sint32 wifi_rfid_locp_recv_open (void)

Enable RFID LOCP (Location Control Protocol) to receive WDS packets.

Parameters

null	

Returns

0, succeed;
otherwise, fail.

4.10.4.11 sint32 wifi_send_pkt_freedom (uint8 * *buf,* uint16 *len,* bool *sys_seq*)

Send user-define 802.11 packets.

Attention

1. Packet has to be the whole 802.11 packet, does not include the FCS. The length of the packet has to be longer than the minimum length of the header of 802.11 packet which is 24 bytes, and less than 1400 bytes.
2. Duration area is invalid for user, it will be filled in SDK.
3. The rate of sending packet is same as the management packet which is the same as the system rate of sending packets.
4. Only after the previous packet was sent, entered the sent callback, the next packet is allowed to send. Otherwise, wifi_send_pkt_freedom will return fail.

Parameters

uint8	*buf : pointer of packet
uint16	len : packet length
bool	sys_seq : follow the system's 802.11 packets sequence number or not, if it is true, the sequence number will be increased 1 every time a packet sent.

Returns

0, succeed;
-1, fail.

4.10.4.12 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

4.10.4.13 bool wifi_set_ip_info (WIFI_INTERFACE *if_index,* struct ip_info * *info*)

Set the IP address of the ESP8266 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_INTERF↩ ACE	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

4.10.4.14 bool wifi_set_macaddr (WIFI_INTERFACE *if_index,* uint8 * *macaddr*)

Set MAC address of the ESP8266 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. ESP8266 soft-AP and station have different MAC addresses, do not set them to be the same.
   - The bit0 of the first byte of ESP8266 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_INTERF↩ ACE	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

4.10.4.15 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

4.10.4.16 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

4.10.4.17 bool wifi_set_phy_mode (WIFI_PHY_MODE *mode*)

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

Attention

The ESP8266 soft-AP only supports bg.

Parameters

WIFI_PHY_MODE	mode : physical mode

Returns

true : succeed
false : fail

4.10.4.18 bool wifi_set_sleep_type (sleep_type *type*)

Sets sleep type.

Set NONE_SLEEP_T to disable sleep. Default to be Modem sleep.

Attention

Sleep function only takes effect in station-only mode.

Parameters

sleep_type	type : sleep type

Returns

true : succeed
false : fail

4.10.4.19 void wifi_status_led_install (uint8 *gpio_id,* uint32 *gpio_name,* uint8 *gpio_func*)

Install the WiFi status LED.

Parameters

uint8	gpio_id : GPIO ID
uint8	gpio_name : GPIO mux name
uint8	gpio_func : GPIO function

Returns

null

4.10.4.20 void wifi_status_led_uninstall (void)

Uninstall the WiFi status LED.

Parameters

null	

Returns

null

4.10.4.21 void wifi_unregister_rfid_locp_recv_cb (void)

Unregister the callback of receiving WDS packets.

Parameters

null	

Returns

null

4.10.4.22 void wifi_unregister_send_pkt_freedom_cb (void)

Unregister the callback for sending user-define 802.11 packets.

Parameters

null	

Returns

null

4.11 Force Sleep APIs

WiFi Force Sleep APIs.

Typedefs

- typedef void(∗ **fpm_wakeup_cb**) (void)

Functions

- void wifi_fpm_open (void)
  Enable force sleep function.
- void wifi_fpm_close (void)
  Disable force sleep function.
- void wifi_fpm_do_wakeup (void)
  Wake ESP8266 up from MODEM_SLEEP_T force sleep.
- void wifi_fpm_set_wakeup_cb (fpm_wakeup_cb cb)
  Set a callback of waken up from force sleep because of time out.
- sint8 wifi_fpm_do_sleep (uint32 sleep_time_in_us)
  Force ESP8266 enter sleep mode, and it will wake up automatically when time out.
- void wifi_fpm_set_sleep_type (sleep_type type)
  Set sleep type for force sleep function.
- sleep_type wifi_fpm_get_sleep_type (void)
  Get sleep type of force sleep function.

4.11.1 Detailed Description

WiFi Force Sleep APIs.

4.11.2 Function Documentation

4.11.2.1 void wifi_fpm_close (void)

Disable force sleep function.

Parameters

null	

Returns

null

4.11.2.2 sint8 wifi_fpm_do_sleep (uint32 *sleep_time_in_us*)

Force ESP8266 enter sleep mode, and it will wake up automatically when time out.

Attention

1. This API can only be called when force sleep function is enabled, after calling wifi_fpm_open. This API can not be called after calling wifi_fpm_close.
2. If this API returned 0 means that the configuration is set successfully, but the ESP8266 will not enter sleep mode immediately, it is going to sleep in the system idle task. Please do not call other WiFi related function right after calling this API.

Parameters

uint32	sleep_time_in_us : sleep time, ESP8266 will wake up automatically when time out. Unit: us. Range: 10000 ∼ 268435455(0xFFFFFFF). • If sleep_time_in_us is 0xFFFFFFF, the ESP8266 will sleep till • if wifi_fpm_set_sleep_type is set to be LIGHT_SLEEP_T, ESP8266 can wake up by GPIO. • if wifi_fpm_set_sleep_type is set to be MODEM_SLEEP_T, ESP8266 can wake up by wifi_fpm_do_wakeup.

Returns

0, setting succeed;
-1, fail to sleep, sleep status error;
-2, fail to sleep, force sleep function is not enabled.

4.11.2.3 void wifi_fpm_do_wakeup (void)

Wake ESP8266 up from MODEM_SLEEP_T force sleep.

Attention

This API can only be called when MODEM_SLEEP_T force sleep function is enabled, after calling wifi_fpm↩_open. This API can not be called after calling wifi_fpm_close.

Parameters

null	

Returns

null

4.11.2.4 sleep_type wifi_fpm_get_sleep_type (void)

Get sleep type of force sleep function.

Parameters

null	

Returns

sleep type

4.11.2.5 void wifi_fpm_open (void)

Enable force sleep function.

Attention

Force sleep function is disabled by default.

Parameters

null	

Returns

null

4.11.2.6 void wifi_fpm_set_sleep_type (sleep_type *type*)

Set sleep type for force sleep function.

Attention

This API can only be called before wifi_fpm_open.

Parameters

sleep_type	type : sleep type

Returns

null

4.11.2.7 void wifi_fpm_set_wakeup_cb (fpm_wakeup_cb *cb*)

Set a callback of waken up from force sleep because of time out.

Attention

1. This API can only be called when force sleep function is enabled, after calling wifi_fpm_open. This API can not be called after calling wifi_fpm_close.
2. fpm_wakeup_cb_func will be called after system woke up only if the force sleep time out (wifi_fpm_do_sleep and the parameter is not 0xFFFFFFF).
3. fpm_wakeup_cb_func will not be called if woke up by wifi_fpm_do_wakeup from MODEM_SLEEP_T type force sleep.

Parameters

void	(∗fpm_wakeup_cb_func)(void) : callback of waken up

Returns

null

4.12 Rate Control APIs

WiFi Rate Control APIs.

Macros

- #define **FIXED_RATE_MASK_NONE** 0x00
- #define **FIXED_RATE_MASK_STA** 0x01
- #define **FIXED_RATE_MASK_AP** 0x02
- #define **FIXED_RATE_MASK_ALL** 0x03
- #define **RC_LIMIT_11B** 0
- #define **RC_LIMIT_11G** 1
- #define **RC_LIMIT_11N** 2
- #define **RC_LIMIT_P2P_11G** 3
- #define **RC_LIMIT_P2P_11N** 4
- #define **RC_LIMIT_NUM** 5
- #define **LIMIT_RATE_MASK_NONE** 0x00
- #define **LIMIT_RATE_MASK_STA** 0x01
- #define **LIMIT_RATE_MASK_AP** 0x02
- #define **LIMIT_RATE_MASK_ALL** 0x03

Enumerations

- enum **FIXED_RATE** {
  PHY_RATE_48 = 0x8, **PHY_RATE_24** = 0x9, **PHY_RATE_12** = 0xA, **PHY_RATE_6** = 0xB,
  PHY_RATE_54 = 0xC, **PHY_RATE_36** = 0xD, **PHY_RATE_18** = 0xE, **PHY_RATE_9** = 0xF }
- enum **support_rate** {
  RATE_11B5M = 0, **RATE_11B11M** = 1, **RATE_11B1M** = 2, **RATE_11B2M** = 3,
  RATE_11G6M = 4, **RATE_11G12M** = 5, **RATE_11G24M** = 6, **RATE_11G48M** = 7,
  RATE_11G54M = 8, **RATE_11G9M** = 9, **RATE_11G18M** = 10, **RATE_11G36M** = 11 }
- enum **RATE_11B_ID** { **RATE_11B_B11M** = 0, **RATE_11B_B5M** = 1, **RATE_11B_B2M** = 2, **RATE_11B_↩ B1M** = 3 }
- enum **RATE_11G_ID** {
  RATE_11G_G54M = 0, **RATE_11G_G48M** = 1, **RATE_11G_G36M** = 2, **RATE_11G_G24M** = 3,
  RATE_11G_G18M = 4, **RATE_11G_G12M** = 5, **RATE_11G_G9M** = 6, **RATE_11G_G6M** = 7,
  RATE_11G_B5M = 8, **RATE_11G_B2M** = 9, **RATE_11G_B1M** = 10 }
- enum **RATE_11N_ID** {
  RATE_11N_MCS7S = 0, **RATE_11N_MCS7** = 1, **RATE_11N_MCS6** = 2, **RATE_11N_MCS5** = 3,
  RATE_11N_MCS4 = 4, **RATE_11N_MCS3** = 5, **RATE_11N_MCS2** = 6, **RATE_11N_MCS1** = 7,
  RATE_11N_MCS0 = 8, **RATE_11N_B5M** = 9, **RATE_11N_B2M** = 10, **RATE_11N_B1M** = 11 }

Functions

- sint32 wifi_set_user_fixed_rate (uint8 enable_mask, uint8 rate)

  Set the fixed rate and mask of sending data from ESP8266.
- int wifi_get_user_fixed_rate (uint8 ∗enable_mask, uint8 ∗rate)

  Get the fixed rate and mask of ESP8266.
- sint32 wifi_set_user_sup_rate (uint8 min, uint8 max)

  Set the support rate of ESP8266.
- bool wifi_set_user_rate_limit (uint8 mode, uint8 ifidx, uint8 max, uint8 min)

  Limit the initial rate of sending data from ESP8266.
- uint8 wifi_get_user_limit_rate_mask (void)

  Get the interfaces of ESP8266 whose rate of sending data is limited by wifi_set_user_rate_limit.
- bool wifi_set_user_limit_rate_mask (uint8 enable_mask)

  Set the interfaces of ESP8266 whose rate of sending packets is limited by wifi_set_user_rate_limit.

4.12.1 Detailed Description

WiFi Rate Control APIs.

4.12.2 Function Documentation

4.12.2.1 int wifi_get_user_fixed_rate (uint8 * *enable_mask,* uint8 * *rate*)

Get the fixed rate and mask of ESP8266.

Parameters

uint8	*enable_mask : pointer of the enable_mask
uint8	*rate : pointer of the fixed rate

Returns

0 : succeed
otherwise : fail

4.12.2.2 uint8 wifi_get_user_limit_rate_mask (void)

Get the interfaces of ESP8266 whose rate of sending data is limited by wifi_set_user_rate_limit.

Parameters

null	

Returns

LIMIT_RATE_MASK_NONE - disable the limitation on both ESP8266 station and soft-AP
LIMIT_RATE_MASK_STA - enable the limitation on ESP8266 station
LIMIT_RATE_MASK_AP - enable the limitation on ESP8266 soft-AP
LIMIT_RATE_MASK_ALL - enable the limitation on both ESP8266 station and soft-AP

4.12.2.3 sint32 wifi_set_user_fixed_rate (uint8 *enable_mask,* uint8 *rate*)

Set the fixed rate and mask of sending data from ESP8266.

Attention

1. Only if the corresponding bit in enable_mask is 1, ESP8266 station or soft-AP will send data in the fixed rate.
2. If the enable_mask is 0, both ESP8266 station and soft-AP will not send data in the fixed rate.
3. ESP8266 station and soft-AP share the same rate, they can not be set into the different rate.

Parameters

uint8	enable_mask : 0x00 - disable the fixed rate • 0x01 - use the fixed rate on ESP8266 station • 0x02 - use the fixed rate on ESP8266 soft-AP • 0x03 - use the fixed rate on ESP8266 station and soft-AP
uint8	rate : value of the fixed rate

Returns

0 : succeed
otherwise : fail

4.12.2.4 bool wifi_set_user_limit_rate_mask (uint8 *enable_mask*)

Set the interfaces of ESP8266 whose rate of sending packets is limited by wifi_set_user_rate_limit.

Parameters

uint8	enable_mask : • LIMIT_RATE_MASK_NONE - disable the limitation on both ESP8266 station and soft-AP • LIMIT_RATE_MASK_STA - enable the limitation on ESP8266 station • LIMIT_RATE_MASK_AP - enable the limitation on ESP8266 soft-AP • LIMIT_RATE_MASK_ALL - enable the limitation on both ESP8266 station and soft-AP

Returns

true : succeed
false : fail

4.12.2.5 bool wifi_set_user_rate_limit (uint8 *mode,* uint8 *ifidx,* uint8 *max,* uint8 *min*)

Limit the initial rate of sending data from ESP8266.

Example: wifi_set_user_rate_limit(RC_LIMIT_11G, 0, RATE_11G_G18M, RATE_11G_G6M);

Attention

The rate of retransmission is not limited by this API.

Parameters

uint8	mode : WiFi mode • #define RC_LIMIT_11B 0 • #define RC_LIMIT_11G 1 • #define RC_LIMIT_11N 2
uint8	ifidx : interface of ESP8266 • 0x00 - ESP8266 station • 0x01 - ESP8266 soft-AP

uint8	max : the maximum value of the rate, according to the enum rate corresponding to the first parameter mode.
uint8	min : the minimum value of the rate, according to the enum rate corresponding to the first parameter mode.

Returns

0 : succeed
otherwise : fail

4.12.2.6 sint32 wifi_set_user_sup_rate (uint8 *min,* uint8 *max*)

Set the support rate of ESP8266.

Set the rate range in the IE of support rate in ESP8266's beacon, probe req/resp and other packets. Tell other devices about the rate range supported by ESP8266 to limit the rate of sending packets from other devices. Example : wifi_set_user_sup_rate(RATE_11G6M, RATE_11G24M);

Attention

This API can only support 802.11g now, but it will support 802.11b in next version.

Parameters

uint8	min : the minimum value of the support rate, according to enum support_rate.
uint8	max : the maximum value of the support rate, according to enum support_rate.

Returns

0 : succeed
otherwise : fail

4.13 User IE APIs

WiFi User IE APIs.

Typedefs

- typedef void(∗ user_ie_manufacturer_recv_cb_t) (user_ie_type type, const uint8 sa[6], const uint8 m_oui[3], uint8 ∗ie, uint8 ie_len, sint32 rssi)

  User IE received callback.

Enumerations

- enum **user_ie_type** {
  USER_IE_BEACON = 0, **USER_IE_PROBE_REQ**, **USER_IE_PROBE_RESP**, **USER_IE_ASSOC_REQ**, **USER_IE_ASSOC_RESP**, **USER_IE_MAX** }

Functions

- bool wifi_set_user_ie (bool enable, uint8 ∗m_oui, user_ie_type type, uint8 ∗user_ie, uint8 len)

  Set user IE of ESP8266.
- sint32 wifi_register_user_ie_manufacturer_recv_cb (user_ie_manufacturer_recv_cb_t cb)

  Register user IE received callback.
- void wifi_unregister_user_ie_manufacturer_recv_cb (void)

  Unregister user IE received callback.

4.13.1 Detailed Description

WiFi User IE APIs.

4.13.2 Typedef Documentation

4.13.2.1 typedef void(∗ user_ie_manufacturer_recv_cb_t) (user_ie_type type, const uint8 sa[6], const uint8 m_oui[3], uint8 ∗ie, uint8 ie_len, sint32 rssi)

User IE received callback.

Parameters

user_ie_type	type : type of user IE.
const	uint8 sa[6] : source address of the packet.
const	uint8 m_oui[3] : factory tag.
uint8	∗user_ie : pointer of user IE.
uint8	ie_len : length of user IE.
sint32	rssi : signal strength.

Returns

null

4.13.3 Function Documentation

4.13.3.1 sint32 wifi_register_user_ie_manufacturer_recv_cb (user_ie_manufacturer_recv_cb_t *cb*)

Register user IE received callback.

Parameters

user_ie_↩ manufacturer_↩ recv_cb_t	cb : callback

Returns

0 : succeed
-1 : fail

4.13.3.2 bool wifi_set_user_ie (bool *enable,* uint8 ∗ *m_oui,* user_ie_type *type,* uint8 ∗ *user_ie,* uint8 *len*)

Set user IE of ESP8266.

The user IE will be added to the target packets of user_ie_type.

Parameters

bool	enable : • true, enable the corresponding user IE function, all parameters below have to be set. • false, disable the corresponding user IE function and release the resource, only the parameter "type" below has to be set.
uint8	∗m_oui : factory tag, apply for it from Espressif System.
user_ie_type	type : IE type. If it is USER_IE_BEACON, please disable the IE function and enable again to take the configuration effect immediately .
uint8	∗user_ie : user-defined information elements, need not input the whole 802.11 IE, need only the user-define part.
uint8	len : length of user IE, 247 bytes at most.

Returns

true : succeed
false : fail

4.13.3.3 void wifi_unregister_user_ie_manufacturer_recv_cb (void)

Unregister user IE received callback.

Parameters

null	

Returns

null

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.
- bool wifi_promiscuous_set_mac (const uint8_t ∗address)

  Set the MAC address filter for the sniffer mode.
- 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 ESP8266 station mode.
2. When in the promiscuous mode, the ESP8266 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_promiscuous_set_mac (const uint8_t * *address*)

Set the MAC address filter for the sniffer mode.

Attention

This filter works only for the current sniffer mode. If users disable and then enable the sniffer mode, and then enable sniffer, they need to set the MAC address filter again.

Parameters

const	uint8_t *address : MAC address

Returns

true : succeed
false : fail

4.14.3.4 bool wifi_set_channel (uint8 *channel*)

Set the channel number for sniffer functions.

Parameters

uint8	channel : channel number

Returns

true : succeed
false : fail

4.14.3.5 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

ESP8266 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

ESP8266 WPS APIs.

WPS can only be used when ESP8266 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 ESP8266 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 ESP8266 station is enabled.

Parameters

WPS_TYPE_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 ESP8266 station is enabled.

Parameters

null	

Returns

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

4.16 Network Espconn APIs

Network espconn APIs.

Data Structures

- struct _esp_tcp
- struct _esp_udp
- struct _remot_info
- struct espconn

Macros

- #define ESPCONN_OK 0
- #define ESPCONN_MEM -1
- #define ESPCONN_TIMEOUT -3
- #define ESPCONN_RTE -4
- #define ESPCONN_INPROGRESS -5
- #define ESPCONN_MAXNUM -7
- #define ESPCONN_ABRT -8
- #define ESPCONN_RST -9
- #define ESPCONN_CLSD -10
- #define ESPCONN_CONN -11
- #define ESPCONN_ARG -12
- #define ESPCONN_IF -14
- #define ESPCONN_ISCONN -15

Typedefs

- typedef void(* espconn_connect_callback) (void *arg)

  Connect callback.
- typedef void(* espconn_reconnect_callback) (void *arg, sint8 err)

  Reconnect callback.
- typedef struct _esp_tcp **esp_tcp**
- typedef struct _esp_udp **esp_udp**
- typedef struct _remot_info **remot_info**
- typedef void(* espconn_recv_callback) (void *arg, char *pdata, unsigned short len)
- typedef void(* **espconn_sent_callback**) (void *arg)
- typedef void(* dns_found_callback) (const char *name, ip_addr_t *ipaddr, void *callback_arg)

  Callback which is invoked when a hostname is found.

Enumerations

- enum espconn_type { ESPCONN_INVALID = 0, ESPCONN_TCP = 0x10, ESPCONN_UDP = 0x20 }
- enum espconn_state {
  ESPCONN_NONE, ESPCONN_WAIT, ESPCONN_LISTEN, ESPCONN_CONNECT,
  ESPCONN_WRITE, ESPCONN_READ, ESPCONN_CLOSE }
- enum espconn_option {
  ESPCONN_START = 0x00, ESPCONN_REUSEADDR = 0x01, ESPCONN_NODELAY = 0x02, ESPCON↩
  N_COPY = 0x04,
  ESPCONN_KEEPALIVE = 0x08, ESPCONN_END }
- enum espconn_level { ESPCONN_KEEPIDLE, ESPCONN_KEEPINTVL, ESPCONN_KEEPCNT }
- enum {
  ESPCONN_IDLE = 0, **ESPCONN_CLIENT**, **ESPCONN_SERVER**, **ESPCONN_BOTH**,
  ESPCONN_MAX }

Functions

- sint8 espconn_connect (struct espconn *espconn)

  Connect to a TCP server (ESP8266 acting as TCP client).
- sint8 espconn_disconnect (struct espconn *espconn)

  Disconnect a TCP connection.
- sint8 espconn_delete (struct espconn *espconn)

  Delete a transmission.
- sint8 espconn_accept (struct espconn *espconn)

  Creates a TCP server (i.e. accepts connections).
- sint8 espconn_create (struct espconn *espconn)

  Create UDP transmission.
- uint8 espconn_tcp_get_max_con (void)

  Get maximum number of how many TCP connections are allowed.
- sint8 espconn_tcp_set_max_con (uint8 num)

  Set the maximum number of how many TCP connection is allowed.
- sint8 espconn_tcp_get_max_con_allow (struct espconn *espconn)

  Get the maximum number of TCP clients which are allowed to connect to ESP8266 TCP server.
- sint8 espconn_tcp_set_max_con_allow (struct espconn *espconn, uint8 num)

  Set the maximum number of TCP clients allowed to connect to ESP8266 TCP server.
- sint8 espconn_regist_time (struct espconn *espconn, uint32 interval, uint8 type_flag)

  Register timeout interval of ESP8266 TCP server.
- sint8 espconn_get_connection_info (struct espconn *pespconn, remot_info **pcon_info, uint8 typeflags)

  Get the information about a TCP connection or UDP transmission.
- sint8 espconn_regist_sentcb (struct espconn *espconn, espconn_sent_callback sent_cb)

  Register data sent callback which will be called back when data are successfully sent.
- sint8 espconn_regist_write_finish (struct espconn *espconn, espconn_connect_callback write_finish_fn)

  Register a callback which will be called when all sending TCP data is completely write into write-buffer or sent.
- sint8 espconn_send (struct espconn *espconn, uint8 *psent, uint16 length)

  Send data through network.
- sint8 espconn_sent (struct espconn *espconn, uint8 *psent, uint16 length)

  Send data through network.
- sint16 espconn_sendto (struct espconn *espconn, uint8 *psent, uint16 length)

  Send UDP data.
- sint8 espconn_regist_connectcb (struct espconn *espconn, espconn_connect_callback connect_cb)

  Register connection function which will be called back under successful TCP connection.
- sint8 espconn_regist_recvcb (struct espconn *espconn, espconn_recv_callback recv_cb)

  register data receive function which will be called back when data are received.
- sint8 espconn_regist_reconcb (struct espconn *espconn, espconn_reconnect_callback recon_cb)

  Register reconnect callback.
- sint8 espconn_regist_disconcb (struct espconn *espconn, espconn_connect_callback discon_cb)

  Register disconnection function which will be called back under successful TCP disconnection.
- uint32 espconn_port (void)

  Get an available port for network.
- sint8 espconn_set_opt (struct espconn *espconn, uint8 opt)

  Set option of TCP connection.
- sint8 espconn_clear_opt (struct espconn *espconn, uint8 opt)

  Clear option of TCP connection.
- sint8 espconn_set_keepalive (struct espconn *espconn, uint8 level, void *optarg)

  Set configuration of TCP keep alive.
- sint8 espconn_get_keepalive (struct espconn *espconn, uint8 level, void *optarg)

Get configuration of TCP keep alive.

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

  DNS function.

- sint8 espconn_igmp_join (ip_addr_t *host_ip, ip_addr_t *multicast_ip)

  Join a multicast group.

- sint8 espconn_igmp_leave (ip_addr_t *host_ip, ip_addr_t *multicast_ip)

  Leave a multicast group.

- sint8 espconn_recv_hold (struct espconn *pespconn)

  Puts in a request to block the TCP receive function.

- sint8 espconn_recv_unhold (struct espconn *pespconn)

  Unblock TCP receiving data (i.e. undo espconn_recv_hold).

- void espconn_dns_setserver (char numdns, ip_addr_t *dnsserver)

  Set default DNS server. Two DNS server is allowed to be set.

4.16.1 Detailed Description

Network espconn APIs.

4.16.2 Macro Definition Documentation

4.16.2.1 #define ESPCONN_ABRT -8

Connection aborted.

4.16.2.2 #define ESPCONN_ARG -12

Illegal argument.

4.16.2.3 #define ESPCONN_CLSD -10

Connection closed.

4.16.2.4 #define ESPCONN_CONN -11

Not connected.

4.16.2.5 #define ESPCONN_IF -14

UDP send error.

4.16.2.6 #define ESPCONN_INPROGRESS -5

Operation in progress.

4.16.2.7 #define ESPCONN_ISCONN -15

Already connected.

4.16.2.8 #define ESPCONN_MAXNUM -7

Total number exceeds the maximum limitation.

4.16.2.9 #define ESPCONN_MEM -1

Out of memory.

4.16.2.10 #define ESPCONN_OK 0

No error, everything OK.

4.16.2.11 #define ESPCONN_RST -9

Connection reset.

4.16.2.12 #define ESPCONN_RTE -4

Routing problem.

4.16.2.13 #define ESPCONN_TIMEOUT -3

Timeout.

4.16.3 Typedef Documentation

4.16.3.1 typedef void(∗ dns_found_callback) (const char ∗name, ip_addr_t ∗ipaddr, void ∗callback_arg)

Callback which is invoked when a hostname is found.

Parameters

const	char ∗name : hostname
ip_addr_t	∗ipaddr : IP address of the hostname, or to be NULL if the name could not be found (or on any other error).
void	∗callback_arg : callback argument.

Returns

null

4.16.3.2 typedef void(∗ espconn_connect_callback) (void ∗arg)

Connect callback.

Callback which will be called if successful listening (ESP8266 as TCP server) or connection (ESP8266 as TCP client) callback, register by espconn_regist_connectcb.

Attention

The pointer "void ∗arg" may be different in different callbacks, please don't use this pointer directly to distinguish one from another in multiple connections, use remote_ip and remote_port in espconn instead.

Parameters

void	∗arg : pointer corresponding structure espconn.

Returns

null

4.16.3.3 typedef void(∗ espconn_reconnect_callback) (void ∗arg, sint8 err)

Reconnect callback.

Enter this callback when error occurred, TCP connection broke. This callback is registered by espconn_regist_↩ reconcb.

Attention

The pointer "void ∗arg" may be different in different callbacks, please don't use this pointer directly to distinguish one from another in multiple connections, use remote_ip and remote_port in espconn instead.

Parameters

void	∗arg : pointer corresponding structure espconn.
sint8	err : error code • ESCONN_TIMEOUT - Timeout • ESPCONN_ABRT - TCP connection aborted • ESPCONN_RST - TCP connection abort • ESPCONN_CLSD - TCP connection closed • ESPCONN_CONN - TCP connection • ESPCONN_HANDSHAKE - TCP SSL handshake fail • ESPCONN_PROTO_MSG - SSL application invalid

Returns

null

4.16.3.4 typedef void(∗ espconn_recv_callback) (void ∗arg, char ∗pdata, unsigned short len)

A callback prototype to inform about events for a espconn

4.16.4 Enumeration Type Documentation

4.16.4.1 enum espconn_level

Enumerator

ESPCONN_KEEPIDLE TCP keep-alive interval, unit : second.

ESPCONN_KEEPINTVL packet interval during TCP keep-alive, unit : second.

ESPCONN_KEEPCNT maximum packet retry count of TCP keep-alive.

4.16.4.2 enum espconn_option

Enumerator

ESPCONN_START no option, start enum.

ESPCONN_REUSEADDR free memory after TCP disconnection happen, need not wait 2 minutes.

ESPCONN_NODELAY disable nagle algorithm during TCP data transmission, quicken the data transmission.

ESPCONN_COPY enable espconn_regist_write_finish, enter write_finish_callback means that the data espconn_send sending was written into 2920 bytes write-buffer waiting for sending or already sent.

ESPCONN_KEEPALIVE enable TCP keep alive.

ESPCONN_END no option, end enum.

4.16.4.3 enum espconn_state

Current state of the espconn.

Enumerator

ESPCONN_NONE idle state, no connection

ESPCONN_WAIT ESP8266 is as TCP client, and waiting for connection

ESPCONN_LISTEN ESP8266 is as TCP server, and waiting for connection

ESPCONN_CONNECT connected

ESPCONN_WRITE sending data

ESPCONN_READ receiving data

ESPCONN_CLOSE connection closed

4.16.4.4 enum espconn_type

Protocol family and type of the espconn

Enumerator

ESPCONN_INVALID invalid type

ESPCONN_TCP TCP

ESPCONN_UDP UDP

4.16.5 Function Documentation

4.16.5.1 sint8 espconn_accept (struct espconn * *espconn*)

Creates a TCP server (i.e. accepts connections).

Parameters

struct	espconn *espconn : the network connection structure

Returns

0 : succeed
Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_ISCONN - Already connected
- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.2 sint8 espconn_clear_opt (struct espconn * *espconn,* uint8 *opt*)

Clear option of TCP connection.

Parameters

struct	espconn *espconn : the TCP connection structure
uint8	opt : enum espconn_option

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.3 sint8 espconn_connect (struct espconn * *espconn*)

Connect to a TCP server (ESP8266 acting as TCP client).

Attention

If espconn_connect fail, returns non-0 value, there is no connection, so it won't enter any espconn callback.

Parameters

struct	espconn *espconn : the network connection structure, the espconn to listen to the connection

Returns

0 : succeed
Non-0 : error code

- ESPCONN_RTE - Routing Problem
- ESPCONN_MEM - Out of memory
- ESPCONN_ISCONN - Already connected
- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.4 sint8 espconn_create (struct espconn * *espconn*)

Create UDP transmission.

Attention

Parameter remote_ip and remote_port need to be set, do not set to be 0.

Parameters

struct	espconn *espconn : the UDP control block structure

Returns

0 : succeed
Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_ISCONN - Already connected
- ESPCONN_ARG - illegal argument, can't find the corresponding UDP transmission according to structure espconn

4.16.5.5 sint8 espconn_delete (struct espconn * *espconn*)

Delete a transmission.

Attention

Corresponding creation API :

- TCP: espconn_accept,
- UDP: espconn_create

Parameters

struct	espconn *espconn : the network connection structure

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding network according to structure espconn

4.16.5.6 sint8 espconn_disconnect (struct espconn * *espconn*)

Disconnect a TCP connection.

Attention

Don't call this API in any espconn callback. If needed, please use system task to trigger espconn_disconnect.

Parameters

struct	espconn *espconn : the network connection structure

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.7 void espconn_dns_setserver (char *numdns,* ip_addr_t * *dnsserver*)

Set default DNS server. Two DNS server is allowed to be set.

Attention

Only if ESP8266 DHCP client is disabled (wifi_station_dhcpc_stop), this API can be used.

Parameters

char	numdns : DNS server ID, 0 or 1
ip_addr_t	*dnsserver : DNS server IP

Returns

null

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

Get the information about a TCP connection or UDP transmission.

Parameters

struct	espconn *espconn : the network connection structure
remot_info	**pcon_info : connect to client info
uint8	typeflags : 0, regular server; 1, ssl server

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding transmission according to structure espconn

4.16.5.9 sint8 espconn_get_keepalive (struct espconn * *espconn,* uint8 *level,* void * *optarg*)

Get configuration of TCP keep alive.

Parameters

struct	espconn *espconn : the TCP connection structure
uint8	level : enum espconn_level
*void**	optarg : value of parameter

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

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

DNS function.

Parse a hostname (string) to an IP address.

Parameters

struct	espconn *pespconn : espconn to parse a hostname.
const	char *hostname : the hostname.
ip_addr_t	*addr : IP address.
dns_found_↩ callback	found : callback of DNS

Returns

err_t :

- ESPCONN_OK - succeed
- ESPCONN_INPROGRESS - error code : already connected
- ESPCONN_ARG - error code : illegal argument, can't find network transmission according to structure espconn

4.16.5.11 sint8 espconn_igmp_join (ip_addr_t * *host_ip,* ip_addr_t * *multicast_ip*)

Join a multicast group.

Attention

This API can only be called after the ESP8266 station connects to a router.

Parameters

ip_addr_t	*host_ip : IP of UDP host
ip_addr_t	*multicast_ip : IP of multicast group

Returns

0 : succeed
Non-0 : error code

- ESPCONN_MEM - Out of memory

4.16.5.12 sint8 espconn_igmp_leave (ip_addr_t * *host_ip,* ip_addr_t * *multicast_ip*)

Leave a multicast group.

Attention

This API can only be called after the ESP8266 station connects to a router.

Parameters

ip_addr_t	*host_ip : IP of UDP host
ip_addr_t	*multicast_ip : IP of multicast group

Returns

0 : succeed
Non-0 : error code

- ESPCONN_MEM - Out of memory

4.16.5.13 uint32 espconn_port (void)

Get an available port for network.

Parameters

null	

Returns

Port number.

4.16.5.14 sint8 espconn_recv_hold (struct espconn * *pespconn*)

Puts in a request to block the TCP receive function.

Attention

The function does not act immediately; we recommend calling it while reserving 5*1460 bytes of memory. This API can be called more than once.

Parameters

struct	espconn ∗espconn : corresponding TCP connection structure

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn.

4.16.5.15 sint8 espconn_recv_unhold (struct **espconn** ∗ *pespconn*)

Unblock TCP receiving data (i.e. undo espconn_recv_hold).

Attention

This API takes effect immediately.

Parameters

struct	espconn ∗espconn : corresponding TCP connection structure

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn.

4.16.5.16 sint8 espconn_regist_connectcb (struct **espconn** ∗ *espconn,* **espconn_connect_callback** *connect_cb*)

Register connection function which will be called back under successful TCP connection.

Parameters

struct	espconn ∗espconn : the TCP connection structure
espconn_↩ connect_↩ callback	connect_cb : registered callback function

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.17 sint8 espconn_regist_disconcb (struct **espconn** ∗ *espconn,* **espconn_connect_callback** *discon_cb*)

Register disconnection function which will be called back under successful TCP disconnection.

Parameters

struct	espconn *espconn : the TCP connection structure
espconn_connect_callback	discon_cb : registered callback function

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.18 sint8 espconn_regist_reconcb (struct espconn * *espconn,* espconn_reconnect_callback *recon_cb*)

Register reconnect callback.

Attention

espconn_reconnect_callback is more like a network-broken error handler; it handles errors that occurs in any phase of the connection. For instance, if espconn_send fails, espconn_reconnect_callback will be called because the network is broken.

Parameters

struct	espconn *espconn : the TCP connection structure
espconn_reconnect_callback	recon_cb : registered callback function

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.19 sint8 espconn_regist_recvcb (struct espconn * *espconn,* espconn_recv_callback *recv_cb*)

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

Parameters

struct	espconn *espconn : the network transmission structure
espconn_recv_callback	recv_cb : registered callback function

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.20 sint8 espconn_regist_sentcb (struct espconn * *espconn,* espconn_sent_callback *sent_cb*)

Register data sent callback which will be called back when data are successfully sent.

Parameters

struct	espconn ∗espconn : the network connection structure
espconn_sent↩ _callback	sent_cb : registered callback function which will be called if the data is successfully sent

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding transmission according to structure espconn

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

Register timeout interval of ESP8266 TCP server.

Attention

1. If timeout is set to 0, timeout will be disable and ESP8266 TCP server will not disconnect TCP clients has stopped communication. This usage of timeout=0, is deprecated.
2. This timeout interval is not very precise, only as reference.

Parameters

struct	espconn ∗espconn : the TCP connection structure
uint32	interval : timeout interval, unit: second, maximum: 7200 seconds
uint8	type_flag : 0, set for all connections; 1, set for a specific connection • If the type_flag set to be 0, please call this API after espconn_accept, before listened a TCP connection. • If the type_flag set to be 1, the first parameter ∗espconn is the specific connection.

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.22 sint8 espconn_regist_write_finish (struct espconn ∗ *espconn,* espconn_connect_callback *write_finish_fn*)

Register a callback which will be called when all sending TCP data is completely write into write-buffer or sent.

Need to call espconn_set_opt to enable write-buffer first.

Attention

1. write-buffer is used to keep TCP data that waiting to be sent, queue number of the write-buffer is 8 which means that it can keep 8 packets at most. The size of write-buffer is 2920 bytes.
2. Users can enable it by using espconn_set_opt.
3. Users can call espconn_send to send the next packet in write_finish_callback instead of using espconn_↩ sent_callback.

Parameters

struct	espconn *espconn : the network connection structure
espconn_connect_callback	write_finish_fn : registered callback function which will be called if the data is completely write into write buffer or sent.

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.23 sint8 espconn_send (struct espconn * *espconn,* uint8 * *psent,* uint16 *length*)

Send data through network.

Attention

1. Please call espconn_send after espconn_sent_callback of the pre-packet.
2. If it is a UDP transmission, it is suggested to set espconn->proto.udp->remote_ip and remote_port before every calling of espconn_send.

Parameters

struct	espconn *espconn : the network connection structure
uint8	*psent : pointer of data
uint16	length : data length

Returns

0 : succeed
Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_ARG - illegal argument, can't find the corresponding network transmission according to structure espconn
- ESPCONN_MAXNUM - buffer of sending data is full
- ESPCONN_IF - send UDP data fail

4.16.5.24 sint16 espconn_sendto (struct espconn * *espconn,* uint8 * *psent,* uint16 *length*)

Send UDP data.

Parameters

struct	espconn *espconn : the UDP structure
uint8	*psent : pointer of data
uint16	length : data length

Returns

0 : succeed
Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_MAXNUM - buffer of sending data is full
- ESPCONN_IF - send UDP data fail

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

Send data through network.

This API is deprecated, please use espconn_send instead.

Attention

1. Please call espconn_sent after espconn_sent_callback of the pre-packet.
2. If it is a UDP transmission, it is suggested to set espconn->proto.udp->remote_ip and remote_port before every calling of espconn_sent.

Parameters

struct	espconn *espconn : the network connection structure
uint8	*psent : pointer of data
uint16	length : data length

Returns

0 : succeed
Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_ARG - illegal argument, can't find the corresponding network transmission according to structure espconn
- ESPCONN_MAXNUM - buffer of sending data is full
- ESPCONN_IF - send UDP data fail

4.16.5.26 sint8 espconn_set_keepalive (struct espconn * *espconn,* uint8 *level,* void * *optarg*)

Set configuration of TCP keep alive.

Attention

In general, we need not call this API. If needed, please call it in espconn_connect_callback and call espconn↩ _set_opt to enable keep alive first.

Parameters

struct	espconn *espconn : the TCP connection structure
uint8	level : To do TCP keep-alive detection every ESPCONN_KEEPIDLE. If there is no response, retry ESPCONN_KEEPCNT times every ESPCONN_KEEPINTVL. If still no response, considers it as TCP connection broke, goes into espconn_reconnect_callback. Notice, keep alive interval is not precise, only for reference, it depends on priority.
*void**	optarg : value of parameter

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.27 sint8 espconn_set_opt (struct espconn * *espconn,* uint8 *opt*)

Set option of TCP connection.

Attention

In general, we need not call this API. If call espconn_set_opt, please call it in espconn_connect_callback.

Parameters

struct	espconn *espconn : the TCP connection structure
uint8	opt : option of TCP connection, refer to enum espconn_option • bit 0: 1: free memory after TCP disconnection happen need not wait 2 minutes; • bit 1: 1: disable nagle algorithm during TCP data transmission, quiken the data transmission. • bit 2: 1: enable espconn_regist_write_finish, enter write finish callback means the data espconn_send sending was written into 2920 bytes write-buffer waiting for sending or already sent. • bit 3: 1: enable TCP keep alive

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.28 uint8 espconn_tcp_get_max_con (void)

Get maximum number of how many TCP connections are allowed.

Parameters

null	

Returns

Maximum number of how many TCP connections are allowed.

4.16.5.29 sint8 espconn_tcp_get_max_con_allow (struct espconn * *espconn*)

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

Parameters

struct	espconn *espconn : the TCP server structure

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.30 sint8 espconn_tcp_set_max_con (uint8 *num*)

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

Parameters

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

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.16.5.31 sint8 espconn_tcp_set_max_con_allow (struct espconn * *espconn,* uint8 *num*)

Set the maximum number of TCP clients allowed to connect to ESP8266 TCP server.

Parameters

struct	espconn *espconn : the TCP server structure
uint8	num : Maximum number of TCP clients which are allowed

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.17 ESP-NOW APIs

ESP-NOW APIs.

Typedefs

- typedef void(∗ esp_now_recv_cb_t) (uint8 ∗mac_addr, uint8 ∗data, uint8 len)

  ESP-NOW send callback.
- typedef void(∗ esp_now_send_cb_t) (uint8 ∗mac_addr, uint8 status)

  ESP-NOW send callback.

Enumerations

- enum **esp_now_role** { **ESP_NOW_ROLE_IDLE** = 0, **ESP_NOW_ROLE_CONTROLLER**, **ESP_NOW_R↩ OLE_SLAVE**, **ESP_NOW_ROLE_MAX** }

Functions

- sint32 esp_now_init (void)

  ESP-NOW initialization.
- sint32 esp_now_deinit (void)

  Deinitialize ESP-NOW.
- sint32 esp_now_register_send_cb (esp_now_send_cb_t cb)

  Register ESP-NOW send callback.
- sint32 esp_now_unregister_send_cb (void)

  Unregister ESP-NOW send callback.
- sint32 esp_now_register_recv_cb (esp_now_recv_cb_t cb)

  Register ESP-NOW receive callback.
- sint32 esp_now_unregister_recv_cb (void)

  Unregister ESP-NOW receive callback.
- sint32 esp_now_send (uint8 ∗da, uint8 ∗data, uint8 len)

  Send ESP-NOW packet.
- sint32 esp_now_add_peer (uint8 ∗mac_addr, uint8 role, uint8 channel, uint8 ∗key, uint8 key_len)

  Add an ESP-NOW peer, store MAC address of target device into ESP-NOW MAC list.
- sint32 esp_now_del_peer (uint8 ∗mac_addr)

  Delete an ESP-NOW peer, delete MAC address of the device from ESP-NOW MAC list.
- sint32 esp_now_set_self_role (uint8 role)

  Set ESP-NOW role of device itself.
- sint32 esp_now_get_self_role (void)

  Get ESP-NOW role of device itself.
- sint32 esp_now_set_peer_role (uint8 ∗mac_addr, uint8 role)

  Set ESP-NOW role for a target device. If it is set multiple times, new role will cover the old one.
- sint32 esp_now_get_peer_role (uint8 ∗mac_addr)

  Get ESP-NOW role of a target device.
- sint32 esp_now_set_peer_channel (uint8 ∗mac_addr, uint8 channel)

  Record channel information of a ESP-NOW device.
- sint32 esp_now_get_peer_channel (uint8 ∗mac_addr)

  Get channel information of a ESP-NOW device.
- sint32 esp_now_set_peer_key (uint8 ∗mac_addr, uint8 ∗key, uint8 key_len)

  Set ESP-NOW key for a target device.

- sint32 esp_now_get_peer_key (uint8 *mac_addr, uint8 *key, uint8 *key_len)

  Get ESP-NOW key of a target device.
- uint8 * esp_now_fetch_peer (bool restart)

  Get MAC address of ESP-NOW device.
- sint32 esp_now_is_peer_exist (uint8 *mac_addr)

  Check if target device exists or not.
- sint32 esp_now_get_cnt_info (uint8 *all_cnt, uint8 *encrypt_cnt)

  Get the total number of ESP-NOW devices which are associated, and the number count of encrypted devices.
- sint32 esp_now_set_kok (uint8 *key, uint8 len)

  Set the encrypt key of communication key.

4.17.1 Detailed Description

ESP-NOW APIs.

Attention

1. ESP-NOW do not support broadcast and multicast.
2. ESP-NOW is targeted to Smart-Light project, so it is suggested that slave role corresponding to soft-AP or soft-AP+station mode, controller role corresponding to station mode.
3. When ESP8266 is in soft-AP+station mode, it will communicate through station interface if it is in slave role, and communicate through soft-AP interface if it is in controller role.
4. ESP-NOW can not wake ESP8266 up from sleep, so if the target ESP8266 station is in sleep, ESP-NOW communication will fail.
5. In station mode, ESP8266 supports 10 encrypt ESP-NOW peers at most, with the unencrypted peers, it can be 20 peers in total at most.
6. In the soft-AP mode or soft-AP + station mode, the ESP8266 supports 6 encrypt ESP-NOW peers at most, with the unencrypted peers, it can be 20 peers in total at most.

4.17.2 Typedef Documentation

4.17.2.1 typedef void(* esp_now_recv_cb_t) (uint8 *mac_addr, uint8 *data, uint8 len)

ESP-NOW send callback.

Attention

The status will be OK, if ESP-NOW send packet successfully. But users need to make sure by themselves that key of communication is correct.

Parameters

uint8	*mac_addr : MAC address of target device
uint8	*data : data received
uint8	len : data length

Returns

null

4.17.2.2 typedef void(* esp_now_send_cb_t) (uint8 *mac_addr, uint8 status)

ESP-NOW send callback.

Attention

The status will be OK, if ESP-NOW send packet successfully. But users need to make sure by themselves that key of communication is correct.

Parameters

uint8	∗mac_addr : MAC address of target device
uint8	status : status of ESP-NOW sending packet, 0, OK; 1, fail.

Returns

null

4.17.3 Function Documentation

4.17.3.1 sint32 esp_now_add_peer (uint8 ∗ *mac_addr,* uint8 *role,* uint8 *channel,* uint8 ∗ *key,* uint8 *key_len*)

Add an ESP-NOW peer, store MAC address of target device into ESP-NOW MAC list.

Parameters

uint8	∗mac_addr : MAC address of device
uint8	role : role type of device, enum esp_now_role
uint8	channel : channel of device
uint8	∗key : 16 bytes key which is needed for ESP-NOW communication
uint8	key_len : length of key, has to be 16 bytes now

Returns

0 : succeed
Non-0 : fail

4.17.3.2 sint32 esp_now_deinit (void)

Deinitialize ESP-NOW.

Parameters

null	

Returns

0 : succeed
Non-0 : fail

4.17.3.3 sint32 esp_now_del_peer (uint8 ∗ *mac_addr*)

Delete an ESP-NOW peer, delete MAC address of the device from ESP-NOW MAC list.

Parameters

u8	∗mac_addr : MAC address of device

Returns

0 : succeed
Non-0 : fail

4.17.3.4 uint8* esp_now_fetch_peer (bool *restart*)

Get MAC address of ESP-NOW device.

Get MAC address of ESP-NOW device which is pointed now, and move the pointer to next one in ESP-NOW MAC list or move the pointer to the first one in ESP-NOW MAC list.

Attention

1. This API can not re-entry
2. Parameter has to be true when you call it the first time.

Parameters

bool	restart : true, move pointer to the first one in ESP-NOW MAC list; false, move pointer to the next one in ESP-NOW MAC list

Returns

NULL, no ESP-NOW devices exist
Otherwise, MAC address of ESP-NOW device which is pointed now

4.17.3.5 sint32 esp_now_get_cnt_info (uint8 * *all_cnt,* uint8 * *encrypt_cnt*)

Get the total number of ESP-NOW devices which are associated, and the number count of encrypted devices.

Parameters

uint8	*all_cnt : total number of ESP-NOW devices which are associated.
uint8	*encryp_cnt : number count of encrypted devices

Returns

0 : succeed
Non-0 : fail

4.17.3.6 sint32 esp_now_get_peer_channel (uint8 * *mac_addr*)

Get channel information of a ESP-NOW device.

Attention

ESP-NOW communication needs to be at the same channel.

Parameters

uint8	*mac_addr : MAC address of target device.

Returns

1 $\sim$ 13 (some area may get 14) : channel number
Non-0 : fail

4.17.3.7 sint32 esp_now_get_peer_key (uint8 * *mac_addr,* uint8 * *key,* uint8 * *key_len*)

Get ESP-NOW key of a target device.

If it is set multiple times, new key will cover the old one.

Parameters

uint8	∗mac_addr : MAC address of target device.
uint8	∗key : pointer of key, buffer size has to be 16 bytes at least
uint8	key_len : key length

Returns

0 : succeed
> 0 : find target device but can't get key
< 0 : fail

4.17.3.8 sint32 esp_now_get_peer_role (uint8 ∗ *mac_addr*)

Get ESP-NOW role of a target device.

Parameters

uint8	∗mac_addr : MAC address of device.

Returns

ESP_NOW_ROLE_CONTROLLER, role type : controller
ESP_NOW_ROLE_SLAVE, role type : slave
otherwise : fail

4.17.3.9 sint32 esp_now_get_self_role (void)

Get ESP-NOW role of device itself.

Parameters

uint8	role : role type of device, enum esp_now_role.

Returns

0 : succeed
Non-0 : fail

4.17.3.10 sint32 esp_now_init (void)

ESP-NOW initialization.

Parameters

null	

Returns

0 : succeed
Non-0 : fail

4.17.3.11 sint32 esp_now_is_peer_exist (uint8 ∗ *mac_addr*)

Check if target device exists or not.

Parameters

uint8	∗mac_addr : MAC address of target device.

Returns

0 : device does not exist
< 0 : error occur, check fail
> 0 : device exists

4.17.3.12 sint32 esp_now_register_recv_cb (esp_now_recv_cb_t *cb*)

Register ESP-NOW receive callback.

Parameters

esp_now_recv_cb_t	cb : receive callback

Returns

0 : succeed
Non-0 : fail

4.17.3.13 sint32 esp_now_register_send_cb (esp_now_send_cb_t *cb*)

Register ESP-NOW send callback.

Parameters

esp_now_send_cb_t	cb : send callback

Returns

0 : succeed
Non-0 : fail

4.17.3.14 sint32 esp_now_send (uint8 ∗ *da,* uint8 ∗ *data,* uint8 *len*)

Send ESP-NOW packet.

Parameters

uint8	∗da : destination MAC address. If it's NULL, send packet to all MAC addresses recorded by ESP-NOW; otherwise, send packet to target MAC address.
uint8	∗data : data need to send
uint8	len : data length

Returns

0 : succeed
Non-0 : fail

4.17.3.15 sint32 esp_now_set_kok (uint8 * *key*, uint8 *len*)

Set the encrypt key of communication key.

All ESP-NOW devices share the same encrypt key. If users do not set the encrypt key, ESP-NOW communication key will be encrypted by a default key.

Parameters

uint8	∗key : pointer of encrypt key.
uint8	len : key length, has to be 16 bytes now.

Returns

0 : succeed
Non-0 : fail

4.17.3.16 sint32 esp_now_set_peer_channel (uint8 ∗ *mac_addr,* uint8 *channel*)

Record channel information of a ESP-NOW device.

When communicate with this device,

- call esp_now_get_peer_channel to get its channel first,
- then call wifi_set_channel to be in the same channel and do communication.

Parameters

uint8	∗mac_addr : MAC address of target device.
uint8	channel : channel, usually to be 1 $\sim$ 13, some area may use channel 14.

Returns

0 : succeed
Non-0 : fail

4.17.3.17 sint32 esp_now_set_peer_key (uint8 ∗ *mac_addr,* uint8 ∗ *key,* uint8 *key_len*)

Set ESP-NOW key for a target device.

If it is set multiple times, new key will cover the old one.

Parameters

uint8	∗mac_addr : MAC address of target device.
uint8	∗key : 16 bytes key which is needed for ESP-NOW communication, if it is NULL, current key will be reset to be none.
uint8	key_len : key length, has to be 16 bytes now

Returns

0 : succeed
Non-0 : fail

4.17.3.18 sint32 esp_now_set_peer_role (uint8 ∗ *mac_addr,* uint8 *role*)

Set ESP-NOW role for a target device. If it is set multiple times, new role will cover the old one.

Parameters

uint8	∗mac_addr : MAC address of device.
uint8	role : role type, enum esp_now_role.

Returns

0 : succeed
Non-0 : fail

4.17.3.19 sint32 esp_now_set_self_role (uint8 *role*)

Set ESP-NOW role of device itself.

Parameters

uint8	role : role type of device, enum esp_now_role.

Returns

0 : succeed
Non-0 : fail

4.17.3.20 sint32 esp_now_unregister_recv_cb (void)

Unregister ESP-NOW receive callback.

Parameters

null	

Returns

0 : succeed
Non-0 : fail

4.17.3.21 sint32 esp_now_unregister_send_cb (void)

Unregister ESP-NOW send callback.

Parameters

null	

Returns

0 : succeed
Non-0 : fail

4.18 Mesh APIs

Mesh APIs.

Enumerations

- enum mesh_status {
  MESH_DISABLE = 0, MESH_WIFI_CONN, MESH_NET_CONN, MESH_LOCAL_AVAIL,
  MESH_ONLINE_AVAIL }
- enum mesh_node_type { MESH_NODE_PARENT = 0, MESH_NODE_CHILD, MESH_NODE_ALL }

Functions

- bool espconn_mesh_local_addr (struct ip_addr *ip)

  Check whether the IP address is mesh local IP address or not.
- bool espconn_mesh_get_node_info (enum mesh_node_type type, uint8_t **info, uint8_t *count)

  Get the information of mesh node.
- bool espconn_mesh_encrypt_init (AUTH_MODE mode, uint8_t *passwd, uint8_t passwd_len)

  Set WiFi cryption algrithm and password for mesh node.
- bool espconn_mesh_set_ssid_prefix (uint8_t *prefix, uint8_t prefix_len)

  Set prefix of SSID for mesh node.
- bool espconn_mesh_set_max_hops (uint8_t max_hops)

  Set max hop for mesh network.
- bool espconn_mesh_group_id_init (uint8_t *grp_id, uint16_t gid_len)

  Set group ID of mesh node.
- bool espconn_mesh_set_dev_type (uint8_t dev_type)

  Set the curent device type.
- sint8 espconn_mesh_connect (struct espconn *usr_esp)

  Try to establish mesh connection to server.
- sint8 espconn_mesh_disconnect (struct espconn *usr_esp)

  Disconnect a mesh connection.
- sint8 espconn_mesh_get_status ()

  Get current mesh status.
- sint8 espconn_mesh_sent (struct espconn *usr_esp, uint8 *pdata, uint16 len)

  Send data through mesh network.
- uint8 espconn_mesh_get_max_hops ()

  Get max hop of mesh network.
- void espconn_mesh_enable (espconn_mesh_callback enable_cb, enum mesh_type type)

  To enable mesh network.
- void espconn_mesh_disable (espconn_mesh_callback disable_cb)

  To disable mesh network.
- void espconn_mesh_init ()

  To print version of mesh.

4.18.1 Detailed Description

Mesh APIs.

4.18.2 Enumeration Type Documentation

4.18.2.1 enum mesh_node_type

Enumerator

MESH_NODE_PARENT get information of parent node

MESH_NODE_CHILD get information of child node(s)

MESH_NODE_ALL get information of all nodes

4.18.2.2 enum mesh_status

Enumerator

MESH_DISABLE mesh disabled

MESH_WIFI_CONN WiFi connected

MESH_NET_CONN TCP connection OK

MESH_LOCAL_AVAIL local mesh is avaliable

MESH_ONLINE_AVAIL online mesh is avaliable

4.18.3 Function Documentation

4.18.3.1 sint8 espconn_mesh_connect (struct espconn * *usr_esp*)

Try to establish mesh connection to server.

Attention

If espconn_mesh_connect fail, returns non-0 value, there is no connection, so it won't enter any espconn callback.

Parameters

struct	espconn *usr_esp : the network connection structure, the usr_esp to listen to the connection

Returns

0 : succeed
Non-0 : error code

- ESPCONN_RTE - Routing Problem
- ESPCONN_MEM - Out of memory
- ESPCONN_ISCONN - Already connected
- ESPCONN_ARG - Illegal argument, can't find the corresponding connection according to structure espconn

4.18.3.2 void espconn_mesh_disable (espconn_mesh_callback *disable_cb*)

To disable mesh network.

Attention

When mesh network is disabed, the system will trigger disable_cb.

Parameters

espconn_↩ mesh_callback	disable_cb : callback function of mesh-disable
enum	mesh_type type : type of mesh, local or online.

Returns

null

4.18.3.3 sint8 espconn_mesh_disconnect (struct espconn * *usr_esp*)

Disconnect a mesh connection.

Attention

Do not call this API in any espconn callback. If needed, please use system task to trigger espconn_mesh_↩ disconnect.

Parameters

struct	espconn *usr_esp : the network connection structure

Returns

0 : succeed
Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.3.4 void espconn_mesh_enable (espconn_mesh_callback *enable_cb,* enum mesh_type *type*)

To enable mesh network.

Attention

1. the function should be called in user_init.
2. if mesh node can not scan the mesh AP, it will be isolate node without trigger enable_cb. user can use espconn_mesh_get_status to get current status of node.
3. if user try to enable online mesh, but node fails to establish mesh connection the node will work with local mesh.

Parameters

espconn_↩ mesh_callback	enable_cb : callback function of mesh-enable
enum	mesh_type type : type of mesh, local or online.

Returns

null

4.18.3.5 bool espconn_mesh_encrypt_init (AUTH_MODE *mode,* uint8_t * *passwd,* uint8_t *passwd_len*)

Set WiFi cryption algrithm and password for mesh node.

Attention

The function must be called before espconn_mesh_enable.

Parameters

AUTH_MODE	mode : cryption algrithm (WPA/WAP2/WPA_WPA2).
uint8_t	∗passwd : password of WiFi.
uint8_t	passwd_len : length of password (8 <= passwd_len <= 64).

Returns

true : succeed
false : fail

4.18.3.6 uint8 espconn_mesh_get_max_hops ( )

Get max hop of mesh network.

Parameters

null.	

Returns

the current max hop of mesh

4.18.3.7 bool espconn_mesh_get_node_info (enum mesh_node_type *type,* uint8_t ∗∗ *info,* uint8_t ∗ *count*)

Get the information of mesh node.

Parameters

enum	mesh_node_type typ : mesh node type.
uint8_t	∗∗info : the information will be saved in ∗info.
uint8_t	∗count : the node count in ∗info.

Returns

true : succeed
false : fail

4.18.3.8 sint8 espconn_mesh_get_status ( )

Get current mesh status.

Parameters

null	

Returns

the current mesh status, please refer to enum mesh_status.

4.18.3.9 bool espconn_mesh_group_id_init (uint8_t ∗ *grp_id,* uint16_t *gid_len*)

Set group ID of mesh node.

Attention

The function must be called before espconn_mesh_enable.

Parameters

uint8_t	∗grp_id : group ID.
uint16_t	gid_len: length of group ID, now gid_len = 6.

Returns

true : succeed
false : fail

4.18.3.10 void espconn_mesh_init ( )

To print version of mesh.

Parameters

null	

Returns

null

4.18.3.11 bool espconn_mesh_local_addr (struct ip_addr ∗ *ip*)

Check whether the IP address is mesh local IP address or not.

Attention

1. The range of mesh local IP address is 2.255.255.∗ ∼ max_hop.255.255.∗.
2. IP pointer should not be NULL. If the IP pointer is NULL, it will return false.

Parameters

struct	ip_addr ∗ip : IP address

Returns

true : the IP address is mesh local IP address
false : the IP address is not mesh local IP address

4.18.3.12 sint8 espconn_mesh_sent (struct espconn ∗ *usr_esp,* uint8 ∗ *pdata,* uint16 *len*)

Send data through mesh network.

Attention

Please call espconn_mesh_sent after espconn_sent_callback of the pre-packet.

Parameters

struct	espconn ∗usr_esp : the network connection structure

uint8	∗pdata : pointer of data
uint16	len : data length

Returns

0 : succeed
Non-0 : error code

- ESPCONN_MEM - out of memory
- ESPCONN_ARG - illegal argument, can't find the corresponding network transmission according to structure espconn
- ESPCONN_MAXNUM - buffer of sending data is full
- ESPCONN_IF - send UDP data fail

4.18.3.13 bool espconn_mesh_set_dev_type (uint8_t *dev_type*)

Set the curent device type.

Parameters

uint8_t	dev_type : device type of mesh node

Returns

true : succeed
false : fail

4.18.3.14 bool espconn_mesh_set_max_hops (uint8_t *max_hops*)

Set max hop for mesh network.

Attention

The function must be called before espconn_mesh_enable.

Parameters

uint8_t	max_hops : max hop of mesh network (1 <= max_hops < 10, 4 is recommended).

Returns

true : succeed
false : fail

4.18.3.15 bool espconn_mesh_set_ssid_prefix (uint8_t ∗ *prefix,* uint8_t *prefix_len*)

Set prefix of SSID for mesh node.

Attention

The function must be called before espconn_mesh_enable.

Parameters

uint8_t	∗prefix : prefix of SSID.
uint8_t	prefix_len : length of prefix (0 < passwd_len <= 22).

Returns

true : succeed
false : fail

4.19 Driver APIs

Driver APIs.

Modules

- PWM Driver APIs

  PWM driver APIs.
- SPI Driver APIs

  SPI Flash APIs.
- GPIO Driver APIs

  GPIO APIs.
- Hardware timer APIs

  Hardware timer APIs.
- UART Driver APIs

  UART driver APIs.

4.19.1 Detailed Description

Driver APIs.

4.20 PWM Driver APIs

PWM driver APIs.

Data Structures

- struct pwm_param

Macros

- #define **PWM_DEPTH** 1023

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.

4.20.1 Detailed Description

PWM driver APIs.

4.20.2 Function Documentation

4.20.2.1 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.

4.20.2.2 uint32 pwm_get_period (void)

Get PWM period, unit : us.

Parameters

null	

Returns

PWM period, unit : us.

4.20.2.3 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

4.20.2.4 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

4.20.2.5 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

4.21 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.

4.21.1 Detailed Description

SmartConfig APIs.

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

4.21.2 Typedef Documentation

4.21.2.1 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

4.21.3 Enumeration Type Documentation

4.21.3.1 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

4.21.3.2 enum sc_type

Enumerator

SC_TYPE_ESPTOUCH protocol: ESPTouch

SC_TYPE_AIRKISS protocol: AirKiss

SC_TYPE_ESPTOUCH_AIRKISS protocol: ESPTouch and AirKiss

4.21.4 Function Documentation

4.21.4.1 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

4.21.4.2 const char* smartconfig_get_version (void)

Get the version of SmartConfig.

Parameters

null	

Returns

SmartConfig version

4.21.4.3 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

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

Start SmartConfig mode.

Start SmartConfig mode, to connect ESP8266 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, ESP8266 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

4.21.4.5 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

4.22 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

- uint32 spi_flash_get_id (void)

  Get ID info of SPI Flash.
- SpiFlashOpResult spi_flash_read_status (uint32 *status)

  Read state register of SPI Flash.
- SpiFlashOpResult spi_flash_write_status (uint32 status_value)

  Write state register of SPI Flash.
- 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.

4.22.1 Detailed Description

SPI Flash APIs.

4.22.2 Macro Definition Documentation

4.22.2.1 #define SPI_FLASH_SEC_SIZE 4096

SPI Flash sector size

4.22.3 Enumeration Type Documentation

4.22.3.1 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

4.22.4 Function Documentation

4.22.4.1 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

4.22.4.2 uint32 spi_flash_get_id (void)

Get ID info of SPI Flash.

Parameters

null	

Returns

SPI Flash ID

4.22.4.3 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

4.22.4.4 SpiFlashOpResult spi_flash_read_status (uint32 * *status*)

Read state register of SPI Flash.

Parameters

uint32	*status : the read value (pointer) of state register.

Returns

SpiFlashOpResult

4.22.4.5 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

4.22.4.6 SpiFlashOpResult spi_flash_write_status (uint32 *status_value*)

Write state register of SPI Flash.

Parameters

uint32	status_value : Write state register value.

Returns

SpiFlashOpResult

4.23 Upgrade APIs

Firmware upgrade (FOTA) APIs.

Data Structures

- struct upgrade_server_info

Macros

- #define SPI_FLASH_SEC_SIZE 4096
- #define USER_BIN1 0x00
- #define USER_BIN2 0x01
- #define UPGRADE_FLAG_IDLE 0x00
- #define UPGRADE_FLAG_START 0x01
- #define UPGRADE_FLAG_FINISH 0x02
- #define UPGRADE_FW_BIN1 0x00
- #define UPGRADE_FW_BIN2 0x01

Typedefs

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

  Callback of upgrading firmware through WiFi.

Functions

- uint8 system_upgrade_userbin_check (void)

  Check the user bin.
- void system_upgrade_reboot (void)

  Reboot system to use the new software.
- uint8 system_upgrade_flag_check ()

  Check the upgrade status flag.
- void system_upgrade_flag_set (uint8 flag)

  Set the upgrade status flag.
- void system_upgrade_init ()

  Upgrade function initialization.
- void system_upgrade_deinit ()

  Upgrade function de-initialization.
- bool system_upgrade (uint8 ∗data, uint32 len)

  Upgrade function de-initialization.
- bool system_upgrade_start (struct upgrade_server_info ∗server)

  Start upgrade firmware through WiFi with normal connection.

4.23.1 Detailed Description

Firmware upgrade (FOTA) APIs.

4.23.2 Macro Definition Documentation

4.23.2.1 #define SPI_FLASH_SEC_SIZE 4096

SPI Flash sector size

4.23.2.2 #define UPGRADE_FLAG_FINISH 0x02

flag of upgrading firmware, finish upgrading

4.23.2.3 #define UPGRADE_FLAG_IDLE 0x00

flag of upgrading firmware, idle

4.23.2.4 #define UPGRADE_FLAG_START 0x01

flag of upgrading firmware, start upgrade

4.23.2.5 #define UPGRADE_FW_BIN1 0x00

firmware, user1.bin

4.23.2.6 #define UPGRADE_FW_BIN2 0x01

firmware, user2.bin

4.23.2.7 #define USER_BIN1 0x00

firmware, user1.bin

4.23.2.8 #define USER_BIN2 0x01

firmware, user2.bin

4.23.3 Typedef Documentation

4.23.3.1 typedef void(* upgrade_states_check_callback) (void *arg)

Callback of upgrading firmware through WiFi.

Parameters

void	* arg : information about upgrading server

Returns

null

4.23.4 Function Documentation

4.23.4.1 bool system_upgrade (uint8 * *data,* uint32 *len*)

Upgrade function de-initialization.

Parameters

uint8	∗data : segment of the firmware bin data
uint32	len : length of the segment bin data

Returns

null

4.23.4.2 void system_upgrade_deinit ( )

Upgrade function de-initialization.

Parameters

null	

Returns

null

4.23.4.3 uint8 system_upgrade_flag_check ( )

Check the upgrade status flag.

Parameters

null	

Returns

#define UPGRADE_FLAG_IDLE 0x00
#define UPGRADE_FLAG_START 0x01
#define UPGRADE_FLAG_FINISH 0x02

4.23.4.4 void system_upgrade_flag_set (uint8 *flag*)

Set the upgrade status flag.

Attention

After downloading new softwares, set the flag to UPGRADE_FLAG_FINISH and call system_upgrade_reboot to reboot the system in order to run the new software.

Parameters

uint8	flag: • UPGRADE_FLAG_IDLE 0x00 • UPGRADE_FLAG_START 0x01 • UPGRADE_FLAG_FINISH 0x02

Returns

null

4.23.4.5 void system_upgrade_init ( )

Upgrade function initialization.

Parameters

null	

Returns

null

4.23.4.6 void system_upgrade_reboot (void)

Reboot system to use the new software.

Parameters

null	

Returns

null

4.23.4.7 bool system_upgrade_start (struct upgrade_server_info * *server*)

Start upgrade firmware through WiFi with normal connection.

Parameters

struct	upgrade_server_info *server : the firmware upgrade server info

Returns

true : succeed
false : fail

4.23.4.8 uint8 system_upgrade_userbin_check (void)

Check the user bin.

Parameters

null	

Returns

0x00 : UPGRADE_FW_BIN1, i.e. user1.bin
0x01 : UPGRADE_FW_BIN2, i.e. user2.bin

4.24 GPIO Driver APIs

GPIO APIs.

Macros

- #define GPIO_OUTPUT_SET(gpio_no, bit_value) gpio_output_conf(bit_value<<gpio_no, ((~bit_↩ value)&0x01)<<gpio_no, 1<<gpio_no, 0)
  Set GPIO pin output level.
- #define GPIO_OUTPUT(gpio_bits, bit_value)
  Set GPIO pin output level.
- #define GPIO_DIS_OUTPUT(gpio_no) gpio_output_conf(0, 0, 0, 1<<gpio_no)
  Disable GPIO pin output.
- #define GPIO_AS_INPUT(gpio_bits) gpio_output_conf(0, 0, 0, gpio_bits)
  Enable GPIO pin intput.
- #define GPIO_AS_OUTPUT(gpio_bits) gpio_output_conf(0, 0, gpio_bits, 0)
  Enable GPIO pin output.
- #define GPIO_INPUT_GET(gpio_no) ((gpio_input_get()>>gpio_no)&BIT0)
  Sample the level of GPIO input.

Functions

- void gpio16_output_conf (void)
  Enable GPIO16 output.
- void gpio16_output_set (uint8 value)
  Set GPIO16 output level.
- void gpio16_input_conf (void)
  Enable GPIO pin intput.
- uint8 gpio16_input_get (void)
  Sample the value of GPIO16 input.
- void gpio_output_conf (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 ()
  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.

4.24.1 Detailed Description

GPIO APIs.

4.24.2 Macro Definition Documentation

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

Enable GPIO pin intput.

Parameters

gpio_bits	: The GPIO bit number.

Returns

null

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

Enable GPIO pin output.

Parameters

gpio_bits	: The GPIO bit number.

Returns

null

4.24.2.3 #define GPIO_DIS_OUTPUT(*gpio_no*) gpio_output_conf(0, 0, 0, 1<<gpio_no)

Disable GPIO pin output.

Parameters

gpio_no	: The GPIO sequence number.

Returns

null

4.24.2.4 #define GPIO_INPUT_GET(*gpio_no*) ((gpio_input_get()>>gpio_no)&BIT0)

Sample the level of GPIO input.

Parameters

gpio_no	: The GPIO sequence number.

Returns

the level of GPIO input

4.24.2.5 #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.

Parameters

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

Returns

null

4.24.2.6 #define GPIO_OUTPUT_SET(*gpio_no, bit_value*) gpio_output_conf(bit_value<<gpio_no, ((~bit_value)&0x01)<<gpio_no, 1<<gpio_no, 0)

Set GPIO pin output level.

Parameters

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

Returns

null

4.24.3 Function Documentation

4.24.3.1 void gpio16_input_conf (void)

Enable GPIO pin intput.

Parameters

null	

Returns

null

4.24.3.2 uint8 gpio16_input_get (void)

Sample the value of GPIO16 input.

Parameters

null	

Returns

the level of GPIO16 input.

4.24.3.3 void gpio16_output_conf (void)

Enable GPIO16 output.

Parameters

null	

Returns

null

4.24.3.4 void gpio16_output_set (uint8 *value*)

Set GPIO16 output level.

Parameters

uint8	value : GPIO16 output level.

Returns

null

4.24.3.5 uint32 gpio_input_get (void)

Sample the value of GPIO input pins and returns a bitmask.

Parameters

null	

Returns

bitmask of GPIO pins input

4.24.3.6 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

4.24.3.7 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

4.24.3.8 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_TY↩ PE	intr_state : GPIO interrupt types.

Returns

null

4.24.3.9 void gpio_pin_wakeup_disable ( )

Disable GPIO wake up to light sleep.

Parameters

null	

Returns

null

4.24.3.10 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_TY↩ PE	intr_state : the level of wake up to light sleep

Returns

null

4.25 Hardware timer APIs

Hardware timer APIs.

Functions

- void hw_timer_init (uint8 req)
  Initialize the hardware ISR timer.
- void hw_timer_arm (uint32 val)
  Set a trigger timer delay to enable this timer.
- void hw_timer_set_func (void(∗user_hw_timer_cb_set)(void))
  Set timer callback function.

4.25.1 Detailed Description

Hardware timer APIs.

Attention

Hardware timer can not interrupt other ISRs.

4.25.2 Function Documentation

4.25.2.1 void hw_timer_arm (uint32 *val*)

Set a trigger timer delay to enable this timer.

Parameters

uint32	val : Timing • In autoload mode, range : 50 ∼ 0x7fffff • In non-autoload mode, range : 10 ∼ 0x7fffff

Returns

null

4.25.2.2 void hw_timer_init (uint8 *req*)

Initialize the hardware ISR timer.

Parameters

uint8	req : 0, not autoload; 1, autoload mode.

Returns

null

4.25.2.3 void hw_timer_set_func (void(∗)(void) *user_hw_timer_cb_set*)

Set timer callback function.

For enabled timer, timer callback has to be set.

Parameters

uint32	val : Timing • In autoload mode, range : 50 ∼ 0x7fffff • In non-autoload mode, range : 10 ∼ 0x7fffff

Returns

null

4.26 UART Driver APIs

UART driver APIs.

Functions

- 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.

4.26.1 Detailed Description

UART driver APIs.

4.26.2 Function Documentation

4.26.2.1 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

4.26.2.2 void uart_init_new (void)

An example illustrates how to configure the serial port.

Parameters

null	

Returns

null

4.26.2.3 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

4.26.2.4 void UART_IntrConfig (UART_Port *uart_no*, UART_IntrConfTypeDef * *pUARTIntrConf*)

Config types of uarts.

Parameters

UART_Port	uart_no : UART0 or UART1
UART_Intr↩ ConfTypeDef	*pUARTIntrConf : parameters structure

Returns

null

4.26.2.5 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_Config↩ TypeDef	∗pUARTConfig : parameters structure

Returns

null

4.26.2.6 void UART_ResetFifo (UART_Port *uart_no*)

Clear uart tx fifo and rx fifo.

Parameters

UART_Port	uart_no : UART0 or UART1

Returns

null

4.26.2.7 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

4.26.2.8 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_Hw↩ FlowCtrl	flow_ctrl : Hardware flow control mode
uint8	rx_thresh : threshold of Hardware flow control

Returns

null

4.26.2.9 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

4.26.2.10 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_Line↩ LevelInverse	inverse_mask : Choose need to flip the IO

Returns

null

4.26.2.11 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_Parity↩ Mode	Parity_mode : the enum of uart parity configuration

Returns

null

4.26.2.12 void UART_SetPrintPort (UART_Port *uart_no*)

Config from which serial output printf function.

Parameters

UART_Port	uart_no : UART0 or UART1

Returns

null

4.26.2.13 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

4.26.2.14 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_Word↩ Length	len : the length of the uart communication data bits

Returns

null

4.26.2.15 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/espressif/esp_wifi.h

5.2 _esp_tcp Struct Reference

Data Fields

- int remote_port
- int local_port
- uint8 local_ip [4]
- uint8 remote_ip [4]
- espconn_connect_callback connect_callback
- espconn_reconnect_callback reconnect_callback
- espconn_connect_callback disconnect_callback
- espconn_connect_callback write_finish_fn

5.2.1 Field Documentation

5.2.1.1 espconn_connect_callback connect_callback

connected callback

5.2.1.2 espconn_connect_callback disconnect_callback

disconnected callback

5.2.1.3 uint8 local_ip[4]

local IP of ESP8266

5.2.1.4 int local_port

ESP8266's local port of TCP connection

5.2.1.5 espconn_reconnect_callback reconnect_callback

as error handler, the TCP connection broke unexpectedly

5.2.1.6 uint8 remote_ip[4]

remote IP of TCP connection

5.2.1.7 int remote_port

remote port of TCP connection

5.2.1.8 espconn_connect_callback write_finish_fn

data send by espconn_send has wrote into buffer waiting for sending, or has sent successfully

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

- include/espressif/espconn.h

5.3 _esp_udp Struct Reference

Data Fields

- int remote_port
- int local_port
- uint8 local_ip [4]
- uint8 remote_ip [4]

5.3.1 Field Documentation

5.3.1.1 uint8 local_ip[4]

local IP of ESP8266

5.3.1.2 int local_port

ESP8266's local port for UDP transmission

5.3.1.3 uint8 remote_ip[4]

remote IP of UDP transmission

5.3.1.4 int remote_port

remote port of UDP transmission

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

- include/espressif/espconn.h

5.4 _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/espressif/esp_timer.h

5.5 _remot_info Struct Reference

Data Fields

- enum espconn_state state
- int remote_port
- uint8 remote_ip [4]

5.5.1 Field Documentation

5.5.1.1 uint8 remote_ip[4]

remote IP address

5.5.1.2 int remote_port

remote port

5.5.1.3 enum espconn_state state

state of espconn

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

- include/espressif/espconn.h

5.6 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
- sint8 rssi
- AUTH_MODE authmode
- uint8 is_hidden
- sint16 freq_offset
- sint16 **freqcal_val**
- uint8 * **esp_mesh_ie**

5.6.1 Member Function Documentation

5.6.1.1 STAILQ_ENTRY (bss_info)

information of next AP

5.6.2 Field Documentation

5.6.2.1 AUTH_MODE authmode

authmode of AP

5.6.2.2 uint8 bssid[6]

MAC address of AP

5.6.2.3 uint8 channel

channel of AP

5.6.2.4 sint16 freq_offset

frequency offset

5.6.2.5 uint8 is_hidden

SSID of current AP is hidden or not.

5.6.2.6 sint8 rssi

single strength of AP

5.6.2.7 uint8 ssid[32]

SSID of AP

5.6.2.8 uint8 ssid_len

SSID length

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

- include/espressif/esp_sta.h

5.7 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/espressif/esp_ssc.h

5.8 dhcps_lease Struct Reference

Data Fields

- bool enable
- struct ip_addr start_ip
- struct ip_addr end_ip

5.8.1 Field Documentation

5.8.1.1 bool enable

enable DHCP lease or not

5.8.1.2 struct ip_addr end_ip

end IP of IP range

5.8.1.3 struct ip_addr start_ip

start IP of IP range

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

- include/espressif/esp_misc.h

5.9 esp_spiffs_config Struct Reference

Data Fields

- uint32 phys_size
- uint32 phys_addr
- uint32 phys_erase_block
- uint32 log_block_size
- uint32 log_page_size
- uint32 fd_buf_size
- uint32 cache_buf_size

5.9.1 Field Documentation

5.9.1.1 uint32 cache_buf_size

cache buffer size

5.9.1.2 uint32 fd_buf_size

file descriptor memory area size

5.9.1.3 uint32 log_block_size

logical size of a block, must be on physical block size boundary and must never be less than a physical block

5.9.1.4 uint32 log_page_size

logical size of a page, at least log_block_size/8

5.9.1.5 uint32 phys_addr

physical offset in spi flash used for spiffs, must be on block boundary

5.9.1.6 uint32 phys_erase_block

physical size when erasing a block

5.9.1.7 uint32 phys_size

physical size of the SPI Flash

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

- include/espressif/esp_spiffs.h

5.10 espconn Struct Reference

`#include <espconn.h>`

Data Fields

- enum espconn_type type
- enum espconn_state state
- union {
  esp_tcp * **tcp**
  esp_udp * **udp**
  } **proto**

- espconn_recv_callback recv_callback
- espconn_sent_callback sent_callback
- uint8 link_cnt
- void * reserve

5.10.1 Detailed Description

A espconn descriptor

5.10.2 Field Documentation

5.10.2.1 uint8 link_cnt

link count

5.10.2.2 espconn_recv_callback recv_callback

data received callback

5.10.2.3 void* reserve

reserved for user data

5.10.2.4 espconn_sent_callback sent_callback

data sent callback

5.10.2.5 enum espconn_state state

current state of the espconn

5.10.2.6 enum espconn_type type

type of the espconn (TCP or UDP)

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

- include/espressif/espconn.h

5.11 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

5.11.1 Field Documentation

5.11.1.1 Event_SoftAPMode_ProbeReqRecved_t ap_probereqrecved

ESP8266 softAP receive probe request packet

5.11.1.2 Event_StaMode_AuthMode_Change_t auth_change

the auth mode of AP ESP8266 station connected to changed

5.11.1.3 Event_StaMode_Connected_t connected

ESP8266 station connected to AP

5.11.1.4 Event_StaMode_Disconnected_t disconnected

ESP8266 station disconnected to AP

5.11.1.5 Event_StaMode_Got_IP_t got_ip

ESP8266 station got IP

5.11.1.6 Event_StaMode_ScanDone_t scan_done

ESP8266 station scan (APs) done

5.11.1.7 Event_SoftAPMode_StaConnected_t sta_connected

a station connected to ESP8266 soft-AP

5.11.1.8 Event_SoftAPMode_StaDisconnected_t sta_disconnected

a station disconnected to ESP8266 soft-AP

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

- include/espressif/esp_wifi.h

5.12 Event_SoftAPMode_ProbeReqRecved_t Struct Reference

Data Fields

- int rssi
- uint8 mac [6]

5.12.1 Field Documentation

5.12.1.1 uint8 mac[6]

MAC address of the station which send probe request

5.12.1.2 int rssi

Received probe request signal strength

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

- include/espressif/esp_wifi.h

5.13 Event_SoftAPMode_StaConnected_t Struct Reference

Data Fields

- uint8 mac [6]
- uint8 aid

5.13.1 Field Documentation

5.13.1.1 uint8 aid

the aid that ESP8266 soft-AP gives to the station connected to

5.13.1.2 uint8 mac[6]

MAC address of the station connected to ESP8266 soft-AP

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

- include/espressif/esp_wifi.h

5.14 Event_SoftAPMode_StaDisconnected_t Struct Reference

Data Fields

- uint8 mac [6]
- uint8 aid

5.14.1 Field Documentation

5.14.1.1 uint8 aid

the aid that ESP8266 soft-AP gave to the station disconnects to

5.14.1.2 uint8 mac[6]

MAC address of the station disconnects to ESP8266 soft-AP

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

- include/espressif/esp_wifi.h

5.15 Event_StaMode_AuthMode_Change_t Struct Reference

Data Fields

- uint8 old_mode
- uint8 new_mode

5.15.1 Field Documentation

5.15.1.1 uint8 new_mode

the new auth mode of AP

5.15.1.2 uint8 old_mode

the old auth mode of AP

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

- include/espressif/esp_wifi.h

5.16 Event_StaMode_Connected_t Struct Reference

Data Fields

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

5.16.1 Field Documentation

5.16.1.1 uint8 bssid[6]

BSSID of connected AP

5.16.1.2 uint8 channel

channel of connected AP

5.16.1.3 uint8 ssid[32]

SSID of connected AP

5.16.1.4 uint8 ssid_len

SSID length of connected AP

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

- include/espressif/esp_wifi.h

5.17 Event_StaMode_Disconnected_t Struct Reference

Data Fields

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

5.17.1 Field Documentation

5.17.1.1 uint8 bssid[6]

BSSID of disconnected AP

5.17.1.2 uint8 reason

reason of disconnection

5.17.1.3 uint8 ssid[32]

SSID of disconnected AP

5.17.1.4 uint8 ssid_len

SSID length of disconnected AP

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

- include/espressif/esp_wifi.h

5.18 Event_StaMode_Got_IP_t Struct Reference

Data Fields

- struct ip_addr ip
- struct ip_addr mask
- struct ip_addr gw

5.18.1 Field Documentation

5.18.1.1 struct ip_addr gw

gateway that ESP8266 station got from connected AP

5.18.1.2 struct ip_addr ip

IP address that ESP8266 station got from connected AP

5.18.1.3 struct ip_addr mask

netmask that ESP8266 station got from connected AP

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

- include/espressif/esp_wifi.h

5.19 Event_StaMode_ScanDone_t Struct Reference

Data Fields

- uint32 status
- struct bss_info * bss

5.19.1 Field Documentation

5.19.1.1 struct bss_info* bss

list of APs found

5.19.1.2 uint32 status

status of scanning APs

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

- include/espressif/esp_wifi.h

5.20 GPIO_ConfigTypeDef Struct Reference

Data Fields

- uint16 GPIO_Pin

- GPIOMode_TypeDef GPIO_Mode
- GPIO_Pullup_IF GPIO_Pullup
- GPIO_INT_TYPE GPIO_IntrType

5.20.1 Field Documentation

5.20.1.1 GPIO_INT_TYPE GPIO_IntrType

GPIO interrupt type

5.20.1.2 GPIOMode_TypeDef GPIO_Mode

GPIO mode

5.20.1.3 uint16 GPIO_Pin

GPIO pin

5.20.1.4 GPIO_Pullup_IF GPIO_Pullup

GPIO pullup

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

- examples/driver_lib/include/gpio.h

5.21 ip_info Struct Reference

Data Fields

- struct ip_addr ip
- struct ip_addr netmask
- struct ip_addr gw

5.21.1 Field Documentation

5.21.1.1 struct ip_addr gw

gateway

5.21.1.2 struct ip_addr ip

IP address

5.21.1.3 struct ip_addr netmask

netmask

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

- include/espressif/esp_wifi.h

5.22 pwm_param Struct Reference

Data Fields

- uint32 period
- uint32 freq
- uint32 duty [8]

5.22.1 Field Documentation

5.22.1.1 uint32 duty[8]

PWM duty

5.22.1.2 uint32 freq

PWM frequency

5.22.1.3 uint32 period

PWM period

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

- include/espressif/pwm.h

5.23 rst_info Struct Reference

Data Fields

- rst_reason reason
- uint32 **exccause**
- uint32 **epc1**
- uint32 **epc2**
- uint32 **epc3**
- uint32 **excvaddr**
- uint32 **depc**
- uint32 **rtn_addr**

5.23.1 Field Documentation

5.23.1.1 rst_reason reason

enum rst_reason

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

- include/espressif/esp_system.h

5.24 scan_config Struct Reference

Data Fields

- uint8 * ssid
- uint8 * bssid
- uint8 channel
- uint8 show_hidden

5.24.1 Field Documentation

5.24.1.1 uint8* bssid

MAC address of AP

5.24.1.2 uint8 channel

channel, scan the specific channel

5.24.1.3 uint8 show_hidden

enable to scan AP whose SSID is hidden

5.24.1.4 uint8* ssid

SSID of AP

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

- include/espressif/esp_sta.h

5.25 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

5.25.1 Field Documentation

5.25.1.1 AUTH_MODE authmode

Auth mode of ESP8266 soft-AP. Do not support AUTH_WEP in soft-AP mode

5.25.1.2 uint16 beacon_interval

Beacon interval, 100 ~ 60000 ms, default 100

5.25.1.3 uint8 channel

Channel of ESP8266 soft-AP

5.25.1.4 uint8 max_connection

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

5.25.1.5 uint8 password[64]

Password of ESP8266 soft-AP

5.25.1.6 uint8 ssid[32]

SSID of ESP8266 soft-AP

5.25.1.7 uint8 ssid_hidden

Broadcast SSID or not, default 0, broadcast the SSID

5.25.1.8 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/espressif/esp_softap.h

5.26 station_config Struct Reference

Data Fields

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

5.26.1 Field Documentation

5.26.1.1 uint8 bssid[6]

MAC address of target AP

5.26.1.2 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.

5.26.1.3 uint8 password[64]

password of target AP

5.26.1.4 uint8 ssid[32]

SSID of target AP

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

- include/espressif/esp_sta.h

5.27 station_info Struct Reference

Public Member Functions

- STAILQ_ENTRY (station_info) next

Data Fields

- uint8 bssid [6]
- struct ip_addr ip

5.27.1 Member Function Documentation

5.27.1.1 STAILQ_ENTRY (station_info)

Information of next AP

5.27.2 Field Documentation

5.27.2.1 uint8 bssid[6]

BSSID of AP

5.27.2.2 struct ip_addr ip

IP address of AP

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

- include/espressif/esp_softap.h

5.28 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:

- examples/driver_lib/include/uart.h

5.29 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:

- examples/driver_lib/include/uart.h

5.30 upgrade_server_info Struct Reference

Data Fields

- struct sockaddr_in sockaddrin
- upgrade_states_check_callback check_cb
- uint32 check_times
- uint8 pre_version [16]
- uint8 upgrade_version [16]
- uint8 * url
- void * **pclient_param**
- uint8 upgrade_flag

5.30.1 Field Documentation

5.30.1.1 upgrade_states_check_callback check_cb

callback of upgrading

5.30.1.2 uint32 check_times

time out of upgrading, unit : ms

5.30.1.3 uint8 pre_version[16]

previous version of firmware

5.30.1.4 struct sockaddr_in sockaddrin

socket of upgrading

5.30.1.5 uint8 upgrade_flag

true, upgrade succeed; false, upgrade fail

5.30.1.6 uint8 upgrade_version[16]

the new version of firmware

5.30.1.7 uint8* url

the url of upgrading server

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

- include/espressif/upgrade.h