泰芯半导体
TaiXin Semiconductor

TaiXin non-OS WiFi Driver Development Guide

珠海泰芯半导体有限公司
TaiXin Semiconductor Co. , Limited
珠海市高新区港湾一号科创园港 11 栋 3 楼

Liability and Copyright

Limitation of Liability

THIS DOCUMENT IS INTENDED FOR REFERENCE ONLY. Zhuhai Taixin Semiconductor Co,. Ltd (hereinafter referred to as "Taixin") does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Taixin reserves the right to make corrections, enhancements, and other changes to this document without notice.

Taixin assumes no liability for applications assistance or the design of customers' products. **Customers are solely responsible for the design, validation, and testing of its applications as well as for compliance with all legal, regulatory, and safety-related requirements concerning its applications.**

Taixin shall not be responsible for any damages, costs, losses, and/or liabilities arising out of customer's non-compliance with this section；concurrently, Customer will fully indemnify Taixin against any damages, costs, losses, and/or liabilities arising out of customer's non-compliance with this section.

Copyright Notice

Without the the written consent of Taixin, no party shall modify, adapt, alter, translate, or create derivative works from this document for commercial purposes.

Without the written consent of Taixin, no party shall disclose or distribute any or parts of the source code, SDK, binaries and object code mentioned in this document to any third party.

No party shall modify, reverse engineer, disassemble, decompile or otherwise attempt to discover the source code of any non-source code parts of the SDK including, but not limited to pre-compiled binaries and object code.

Furthermore, any other actions that may infringe upon Taixin's rights or other intellectual property owners are strictly prohibited.

For the subject who commits the above infringement, Taixin has the right to take necessary legal measure in accordance with the laws of the People's Republic of China or other applicable laws and international treaties, including but not limited to filing a lawsuit or arbitration against the infringer, or applying for legal compulsory measures.

Zhuhai Taixin Semiconductor Co., Ltd.

September 24,2024

Revision History

Date	Version	Revision Notes	Reviser
2023-10-3	V2.2	Modified the description of set_wakehost_reasons; Modified the description for set_pri_chan and set_cust_iso;	WY
2023-07-28	V2.1	Added the description for send_customer_mgmt;	WY
2023-07-07	V2.0	Removed the setting for multicast encryption;	WY
2023-05-23	V1.9	Modified the description of multicast parameters;	WY
2023-04-07	V1.8	Modified the get bssid;	DY
2023-04-06	V1.7.1	Modified the header and footer;	WY
2023-01-30	V1.7	Added newly added API description and organise the order of sections;	CWY
2022-10-21	V1.6	Added OTA related descriptions;	CWY
2022-09-22	V1.5	Added low power consumption related parameters, and other previously undescribed parameters;	CWY
2022-07-6	V1.4.2	Modified file name;	WY
2022-02-18	V1.4.1	Modified the logo;	XYJ
2021-07-27	V1.4	Added hgic_raw using description;	DY
2021-07-17	V1.3	Modified the description of spidrv_read: full duplex read SPI data while sending data must be 0xFF;	LHY
2021-02-05	V1.2	Removed old SPI interface, add new SPI interface;	LHY
2020-12-05	V1.1	Added SPI pin description;	LHY
2020-10-20	V1.0	Initial version;	LHY

Table of Contents

1. Overview

The Taixin non-OS WiFi driver is suitable for product solutions with low transmission rate requirements. It has low software and hardware environment requirements for the main control chip and supports connecting the WiFi module via SPI/UART interfaces.

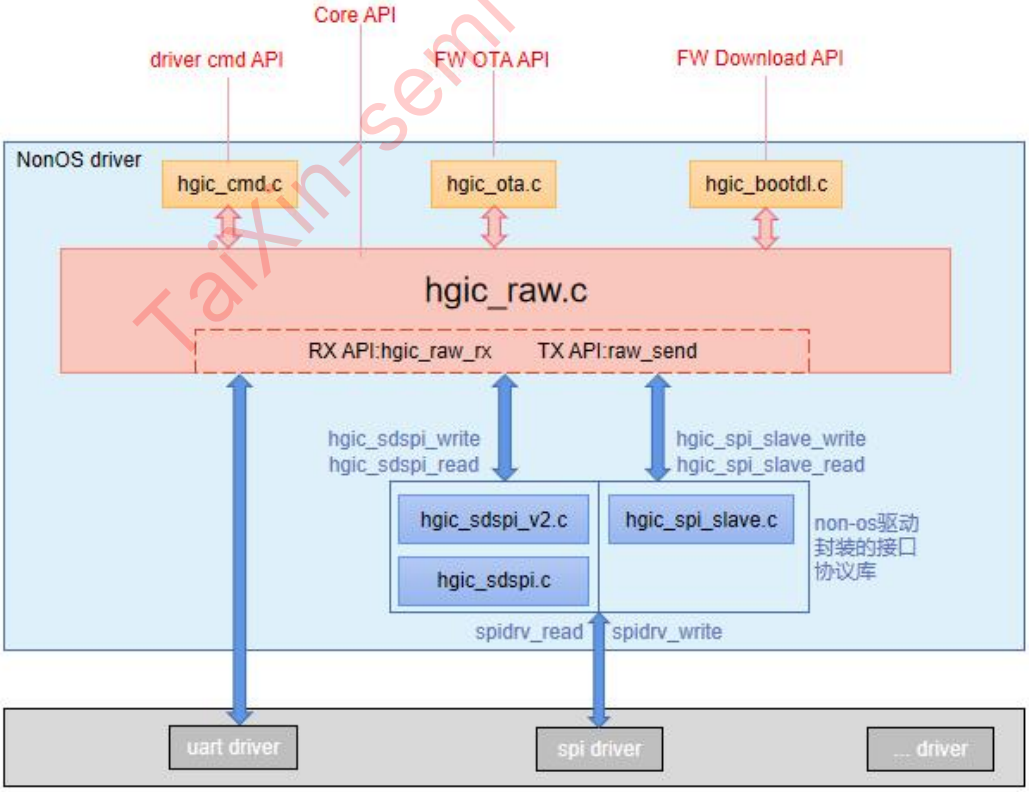
Currently, the non-OS driver supports the AH module, and the WiFi firmware type is vx.x.x.5.

2. Working Principle of Non-OS Driver

2.1. Non-OS Driver Framework

The framework of the non-OS WiFi driver is shown in the figure below, consisting of the following modules:

- **Parameter Setting Interface:** hgic_cmd.c. This file provides the driver parameter setting API. For specific API usage instructions, please refer to Chapter 3's Driver Parameter Setting API Instructions.
- **Firmware Upgrade Module:** hgic_ota.c. This file provides the WiFi module firmware upgrade function. For WiFi modules without flash, this function is invalid.
- **Firmware Download Module:** hgic_bootdl.c. This file provides the function of downloading WiFi firmware to the WiFi module via the interface. When the WiFi module does not have flash, or there is no firmware in the flash, this module can be used to download and run the firmware.
- **Data Transmission Module:** hgic_raw.c. This file is the core module of the non-OS driver, responsible for data transmission processing, including command data, event data, application data. When sending, hgic_raw.c encapsulates various data into HGIC FRM format; when receiving, hgic_raw.c parses the HGIC FRM format and performs unpacking processing.

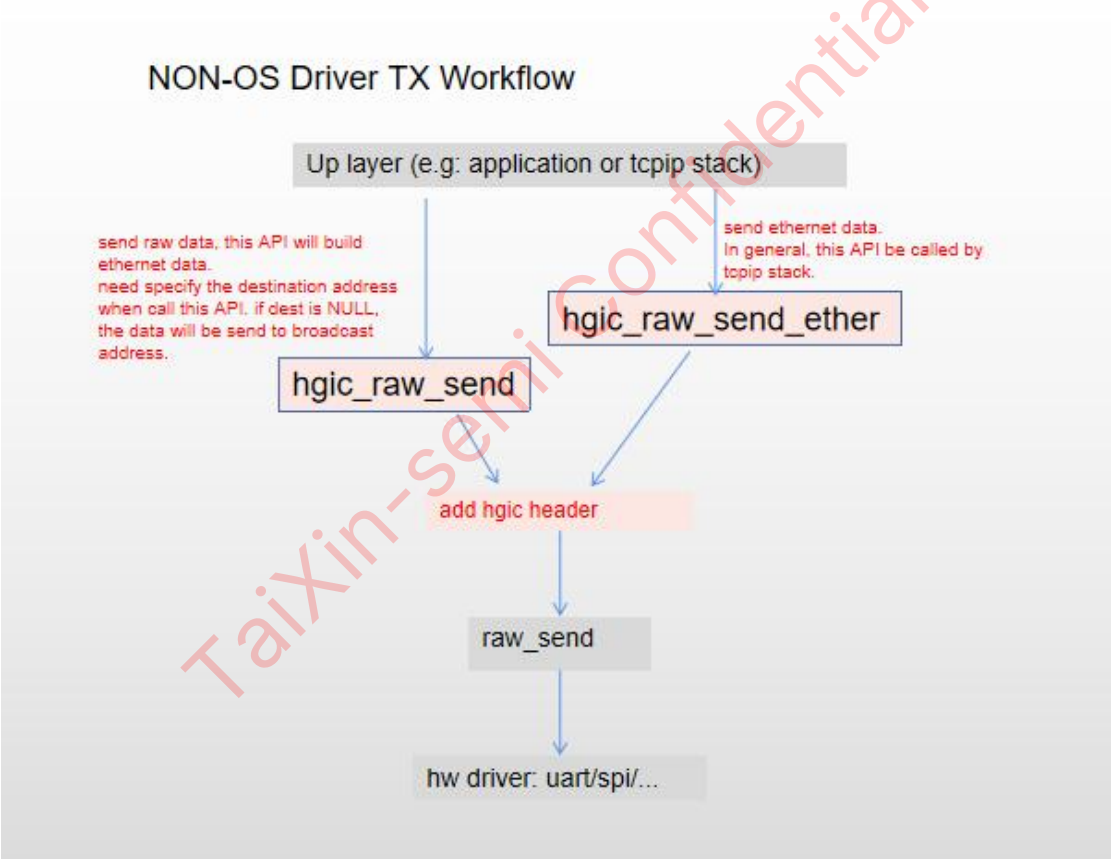


2.2. Non-OS Driver Workflow

2.2.1. Data Transmission Process

The data transmission process of the non-OS WiFi driver is shown in the figure below, providing two transmission APIs:

- **hgic_raw_send:** This API is used for transmitting raw data (non-Ethernet data). When the main control does not run the TCP/IP protocol stack, this API can be used for data transmission. This API will encapsulate the data into Ethernet frames and then transmit it.
- **hgic_raw_send_ether:** This API is used for transmitting Ethernet data. When the main control runs the TCP/IP protocol stack, this API needs to be used.



2.2.2. Data Reception Process

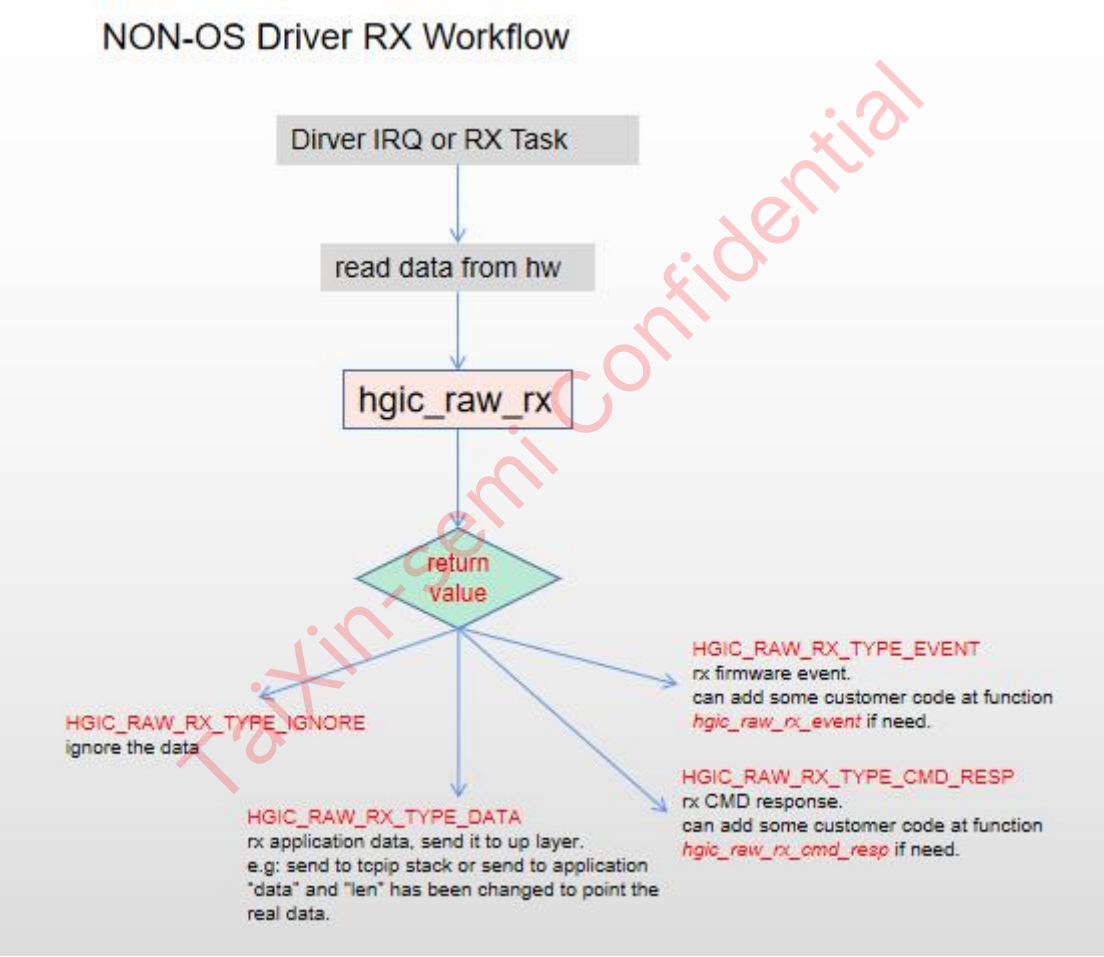
The data reception process of the non-OS WiFi driver is shown in the figure below. The driver provides one reception entry function: hgic_raw_rx.

After the underlying interface driver receives the data, it inputs the data to this interface for processing. The return value of this interface will indicate the type of received data, and the

definition of the data type is as follows:

```
typedef enum {
    HGIC_RAW_RX_TYPE_IGNORE       = 0,  //invalid data, just ignore it.
    HGIC_RAW_RX_TYPE_DATA         = 1,  //recieved data.
    HGIC_RAW_RX_TYPE_CMD_RESP     = 2,  //recieved cmd response.
    HGIC_RAW_RX_TYPE_BOOTDL_RESP  = 3,  //bootdl response.
    HGIC_RAW_RX_TYPE_OTA_RESP     = 4,  //ota response.
    HGIC_RAW_RX_TYPE_EVENT        = 5,  //firmware event.
} HGIC_RAW_RX_TYPE;
```

The data reception code needs to process accordingly based on the return value type.



3. Non-OS WiFi Driver Parameter Settings

3.1. Basic Networking API

3.1.1. set_ssid

API	int hgic_raw_set_ssid(char *ssid);
Description	Set the SSID, with a maximum length of 32 characters.
Example	hgic_raw_set_ssid(“hgic_ah_test”);

3.1.2. set_key_mgmt

API	int hgic_raw_set_key_mgmt(char *key_mgmt);
Description	Set the encryption method. **"WPA-PSK":** Enable encryption. **Any other value:** Disable encryption.

Example	hgic_raw_set_key_mgmt(“WPA-PSK”); //Enable encryption
	hgic_raw_set_key_mgmt(“NONE”); //Disable encryption

3.1.3. set_wpa_psk

API	int hgic_raw_set_wpa_psk(char *psk);
Description	Set the encryption key, psk must be 64 hex characters.
Example	hgic_raw_set_wpa_psk(“1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef”);

3.1.4. set_freq_range

API	int hgic_raw_set_freq_range(int freq_start, int freq_end, int bss_bw);
Description	Set the frequency range. freq_start is the starting frequency, freq_end is the ending frequency, and bss_bw is the frequency step. Frequency values are in 100 kHz units.
Example	hgic_raw_set_freq_range(7600,7840,8); // Set frequency range from 760 MHz to 784 MHz with an 8 MHz bandwidth.

3.1.5. set_bss_bw

API	int hgic_raw_set_bss_bw(char bss_bw);

Description	Set the BSS bandwidth. Valid bss_bw values are 1/2/4/8 MHz.
Example	hgic_raw_set_bss_bw(8);

3.1.6. set_acs

API	int hgic_raw_set_acs(char acs, char acs_tmo);
Description	Set the frequency background scan. Scan the noise floor of each channel and selects the cleanest channel for communication. **acs:** 1 to enable scanning, 0 to disable scanning. **acs_tmo:** Channel scan time in milliseconds.
Example	hgic_raw_set_acs(1,10);

3.1.7. set_mac

API	int hgic_raw_set_mac(char *mac);
Description	Set the device MAC address.
Example	char addr[] = {11,22,33,44,52,65}; hgic_raw_set_mac(addr);

3.1.8. set_chan_list

API	int hgic_raw_set_chan_list(int *chan_list, int count);
Description	Set a custom frequency point list, which can be used to set non-continuous frequency points. Support up to 16 custom frequency points. **chan_list:** Array of custom frequency points, with frequency values in 100 kHz units. **count:** Number of frequency points in the array.

Example	unsigned short list[]={7600,7680,7740};
	hgic_raw_set_chan_list(list,3);
	// Set 3 custom frequency points: 760 MHz, 768 MHz, 774 MHz.

3.1.9. set_mode

API	int hgic_raw_set_mode(char *mode);
Description	Set the AH operating mode. **"ap":** AP mode. **"sta":** STA mode.
Example	hgic_raw_set_mode("ap"); // Set to AP mode.

3.1.10. set_pairing

API	int hgic_raw_set_pairing(char enable);
Description	**enable=1:** Start pairing. **enable=0:** Stop pairing.
Example	hgic_raw_set_pairing(1);

3.1.11. set_pair_autostop

API	int hgic_raw_set_pair_autostop(char enable);
Description	Set whether the AH module stops pairing automatically after a successful pairing. **enable=1:** Enable auto-stop pairing.
Example	hgic_raw_set_pair_autostop(1);

3.2. Advanced Networking API

3.2.1. set_txpower

API	hgic_raw_set_txpower(int power);
Description	**power:** Transmission power for TX; range: 1~20.
Example	hgic_raw_set_txpower(15);

3.2.2. set_tx_mcs

API	int hgic_raw_set_tx_mcs(char tx_mcs);
Description	Set TX MCS, valid values for tx_mcs are 0~7, others set to auto mode.
Example	hgic_raw_set_tx_mcs(255); //tx mcs auto mode

3.2.3. set_bss_max_idle

API	hgic_raw_set_bss_max_idle(int bss_max_idle);
Description	Set BSS max idle time in seconds. The STA must send a packet to the AP within the max idle time to maintain the connection. If the AP does not receive a packet from the STA within this time, it assumes the STA is disconnected. Note that this parameter affects the STA's sleep power

	consumption—the smaller the setting, the higher the power consumption. The default is 300 seconds, corresponding to 200uA@DTIM10.
Example	hgic_raw_set_bss_max_idle(300);

3.2.4. set_channel

API	int hgic_raw_set_channel(int channel);
Description	Set Channel, the channel value is the channel number, starting from 1.
Example	hgic_raw_set_channel(1); // Set the first channel to use

3.2.5. unpair

API	int hgic_raw_unpair(char *mac);
Description	This command can be executed on both AP and STA to unpair from a specified MAC address.
Example	char addr[] = {11,22,33,44,52,65}; hgic_raw_unpair(addr);

3.2.6. set_paired_stas

API	int hgic_raw_set_paired_stas(char *stas, int len);
Description	During pairing, the module automatically generates STA information, forming a STA list. If the module is equipped with flash, the STA list is saved to the flash and automatically loaded on reboot. If not, the STA list cannot be saved and will be lost on reboot. For APs without flash, the set paired_stas command can be used after powering on to reset the pairing list. Setting paired_stas along with conn_paironly=1 allows only these STAs to connect to the AP. Other STAs cannot connect even with the correct SSID and password, effectively implementing a whitelist for AP connections. The maximum number of STAs that can be set is limited by the firmware's support for the maximum number of STAs.
Example	char stas[][] = {{11,22,33,44,52,65}, {11,22,33,44,52,77}}; hgic_raw_set_paired_stas(stas, sizeof(stas));

3.2.7. set_auto_chswitch

API	int hgic_raw_set_auto_chswitch(char enable);
Description	Enable or disable automatic frequency hopping: (default is enabled). This feature allows the AP to automatically switch to a cleaner channel if interference is detected on the current channel. The AP notifies the STA to switch channels, but it cannot notify STAs in sleep mode. After switching, sleeping STAs will detect AP timeout, restart, scan, and reconnect.
Example	hgic_raw_set_auto_chswitch(1);

3.2.8. set_primary_chan

API	int hgic_raw_set_primary_chan(char primary_chan);
Description	Set pri_chan, the default is 3. This parameter is no longer valid.
Example	hgic_raw_set_primary_chan(1);

3.2.9. set_mac_filter_en

API	int hgic_raw_set_mac_filter_en(char enable);
Description	Enable MAC address filtering.
Example	hgic_raw_set_mac_filter_en(1);

3.2.10. set_supper_pwr

API	int hgic_raw_set_supper_pwr(char enable);
Description	Set super power function whether enable or not. When enabled, the AH module increases transmission power up to 25dBm for long-distance communication. This function is enabled by default in normal mode and disabled in test mode.
Example	hgic_raw_set_supper_pwr(1);

3.2.11. set_acktmo

API	int hgic_raw_set_acktmo(int acktmo);
Description	Set ACK timeout value for WiFi protocol parameters, the unit is microseconds, default is 0. This parameter is needed only for communication over distances greater than 1km.

	Formula: 10 * (distance in km - 1). For example, for 2km: acktmo=10;
Example	hgic_raw_set_acktmo(10);

3.2.12. set_beacon_int

API	int hgic_raw_set_beacon_int(int beacon_int);
Description	Set beacon packet period. Unit is microseconds, default is 500ms.
Example	hgic_raw_set_beacon_int(100);

3.2.13. set_heartbeat_int

API	int hgic_raw_set_heartbeat_int(int heartbeat_int);
Description	Set STA heartbeat timeout on AP side: Unit is milliseconds, default is 500ms. The connection is considered disconnected if 7 consecutive heartbeat packets are not received.
Example	hgic_raw_set_heartbeat_int(100);

3.2.14. set_ap_chan_switch

API	int hgic_raw_set_ap_chan_switch(char chan_index, char counter);
Description	On the AP side, set the switch to another channel after counter seconds. **chan_index:** Channel index **counter:** Time to wait before switching channels.
Example	hgic_raw_set_ap_chan_switch(1,10);

3.2.15. set_agg_cnt

API	int hgic_raw_set_agg_cnt(char agg_cnt);
Description	Set maximum number of aggregated packets, default is 16.
Example	hgic_raw_set_agg_cnt(10);

3.2.16. set_ap_hide

API	int hgic_raw_set_ap_hide(char en);
Description	Enable AP hiding function. Valid only in AP mode. When enabled, STAs cannot discover the AP through scanning.
Example	hgic_raw_set_ap_hide(1);

3.2.17. set_max_txcnt

API	int hgic_raw_set_max_txcnt(char txcnt);
Description	Set maximum retransmissions for WiFi frames: 0 is invalid, N means a maximum of N total transmissions, default is 7.
Example	hgic_raw_set_max_txcnt(7);

3.2.18. set_dup_filter_en

API	int hgic_raw_set_dup_filter_en(char en);
Description	Filter received retransmission packets.
Example	hgic_raw_set_dup_filter_en(1);

3.2.19. set_dis_1v1_m2u

API	int hgic_raw_set_dis_1v1_m2u(char dis);
Description	Disable multicast to unicast conversion in 1-to-1 mode.
Example	hgic_raw_set_dis_1v1_m2u(1);

3.2.20. set_kick_assoc

API	int hgic_raw_set_kick_assoc(char en);
Description	Initiate a kick connection operation.
Example	hgic_raw_set_kick_assoc(1);

3.2.21. set_connect_paironly

API	int hgic_raw_set_connect_paironly(char en);
Description	Set conn_paironly: When enabled, the AP only allows STAs in the pairing list to connect. STAs not in the list cannot connect even with the correct SSID and password, creating a whitelist effect. **conn_paironly=1 :** Only allow STAs in the pairing list to connect. **conn_paironly=0 :** Allow all STAs to connect with the correct SSID and password. Default is 0.
Example	hgic_raw_set_connect_paironly(1);

3.2.22. set_disassoc_sta

API	int hgic_raw_disassoc_sta(char *addr);
Description	Disconnect specified STA: The AP can use this command to actively disconnect a STA. In normal mode, the STA will automatically reconnect after disconnection.
Example	char addr[] = {0xf6,0xde,0x09,0x6e,0x5a,0x50}; hgic_raw_disassoc_sta(addr);

3.2.23. set_sta_freqinfo

API	int hgic_raw_set_sta_freqinfo(char *addr, struct hgic_freqinfo* freqinfo);
Description	Set pairing device frequency, bandwidth, etc.
Example	char addr[] = {0xf6,0xde,0x09,0x6e,0x5a,0x50}; hgic_raw_set_sta_freqinfo(addr, &hgic.freqinfo);

3.2.24. set_cust_isolation

API	int hgic_raw_set_cust_isolation(char en);
Description	Set whether chips with different customer IDs can interconnect. Default is enabled.
Example	hgic_raw_set_cust_isolation(1);

3.3. Status Query API

3.3.1. get_fwinfo

API	int hgic_raw_get_fwinfo(void);
Description	Get module firmware information.
Example	hgic_raw_get_fwinfo();

3.3.2. get_connect_state

API	int hgic_raw_get_connect_state(void);
Description	Check connection status.
Example	hgic_raw_get_connect_state();

3.3.3. get_mode

API	int hgic_raw_get_mode(void);
Description	Get module role.
Example	hgic_raw_get_mode();

3.3.4. get_sta_list

API	int hgic_raw_get_sta_list(void);
Description	Check connected STA information: Includes aid, MAC address, ps_mode (module sleep mode), rssi, evm, tx_snr (remote device statistics of rssi - bgrssi, air feedback), rx_snr (local statistics of rssi - bgrssi). This command executed on the AP can retrieve connected STA information. Executed on the STA, it retrieves information about the currently connected AP.
Example	int hgic_raw_get_sta_list(void);

3.3.5. get_bgrssi

API	int hgic_raw_get_bgrssi(char chan_index);
Description	Get background noise of a specified channel, values start from 1.
Example	hgic_raw_get_bgrssi(2);

3.3.6. scan

API	int hgic_raw_scan(void);
Description	Scan surrounding AP information in STA mode.
Example	hgic_raw_scan();

3.3.7. get_scan_list

API	int hgic_raw_get_scan_list(void);
Description	Retrieve scanned AP list after executing scan command.
Example	hgic_raw_get_scan_list();

3.3.8. get_temperature

API	int hgic_raw_get_temperature(void);
Description	Get module temperature information.
Example	hgic_raw_get_temperature();

3.3.9. get_ssid

API	int hgic_raw_get_ssid(void);
Description	Get module SSID.
Example	hgic_raw_get_ssid();

3.3.10. get_bssid

API	int hgic_raw_get_bssid(char *bssid);
Description	Get connected AP MAC in STA mode, also return the STA's AID.
Example	aid = hgic_raw_get_bssid(ap_mac);

3.3.11. get_wpa_psk

API	int hgic_raw_get_wpa_psk(void);
Description	Get module encryption key.
Example	hgic_raw_get_wpa_psk();

3.3.12. get_sta_count

API	int hgic_raw_get_sta_count(void);
Description	Get the number of STA connections to the module.
Example	hgic_raw_get_sta_count();

3.3.13. get_txpower

API	int hgic_raw_get_txpower(void);
Description	Get module TX power level.
Example	hgic_raw_get_txpower();

3.3.14. get_aggcnt

API	int hgic_raw_get_aggcnt(void);

Description	Get maximum packet aggregation value of the module.
Example	hgic_raw_get_aggcnt();

3.3.15. get_freq_range

API	int hgic_raw_get_freq_range(void);
Description	Get module frequency range.
Example	hgic_raw_get_freq_range();

3.3.16. get_chan_list

API	int hgic_raw_get_chan_list(void);
Description	Get module frequency list.
Example	hgic_raw_get_chan_list();

3.3.17. get_bss_bw

API	int hgic_raw_get_bss_bw(void);
Description	Get module bandwidth.
Example	hgic_raw_get_bss_bw();

3.3.18. get_key_mgmt

API	int hgic_raw_get_key_mgmt(void);
Description	Get module encryption method.
Example	hgic_raw_get_key_mgmt();

3.3.19. get_module_type

API	int hgic_raw_get_module_type(void);
Description	Get module type.
Example	hgic_raw_get_module_type();

3.3.20. get_dhcpc_result

API	int hgic_raw_get_dhcpc_result(void);
Description	Get module dhcpc results.
Example	hgic_raw_get_dhcpc_result();

3.3.21. get_disassoc_reason

API	int hgic_raw_get_disassoc_reason(void);
Description	Get device disconnection information.
Example	hgic_raw_get_disassoc_reason();

3.3.22. get_rtc

API	int hgic_raw_get_rtc(void);
Description	Get module RTC clock.
Example	hgic_raw_get_rtc();

3.3.23. get_center_freq

API	int hgic_raw_get_center_freq(void);
Description	Get the center frequency of the module's current channel.
Example	hgic_raw_get_center_freq();

3.3.24. get_wakeup_reason

API	int hgic_raw_get_wakeup_reason(void);
Description	Get the module wake-up reason.
Example	hgic_raw_get_wakeup_reason();

3.4. Low Power Consumption Related APIs

3.4.1.sleep

API	int hgic_raw_sleep(char sleep);
Description	Set the sleep state. The sleep parameter determines whether to enable sleep mode.
Example	hgic_raw_sleep(1); // Enter sleep mode.

3.4.2. set_wkio_mode

API	int hgic_raw_set_wkio_mode(char wkio_mode);
Description	Set the AH wake-up Host IO (IOB0) working mode. The default is pulse mode.

	1: Level mode, high level when WiFi is running normally, low level when in sleep. **0:** Pulse mode, a 2ms high pulse signal when the WiFi module wakes up the host.
Example	hgic_raw_set_wkio_mode(1);

3.4.3. set_dtim_period

API	hgic_raw_set_dtim_period(int dtim_period);
Description	Set the DTIM period of the WiFi module in sleep mode, in milliseconds, which is a multiple of the beacon interval. The WiFi will wake up periodically according to the DTIM period to check the connection with the AP and receive wake-up calls from the AP. A DTIM10 setting value is 1000 (default), DTIM20 is 2000, and so on. This parameter affects sleep power consumption and wake-up time. A larger DTIM value reduces sleep power consumption but increases wake-up time. Conversely, a smaller DTIM value increases sleep power consumption but reduces wake-up time. If you want to use a value smaller than 1000, i.e., a keep-alive period less than 1 second, you need to set both dtim_period and beacon_int accordingly.
Example	hgic_raw_set_dtim_period(2000);

3.4.4. set_ps_mode

API	int hgic_raw_set_ps_mode(char ps_mode);
Description	Set the sleep mode of the WiFi module when connected: **0:** No sleep mode set, similar to mode 3. **1:** The module keeps alive with the server while in sleep (module keeps alive with server). **2:** The AP keeps alive with the server while the module is in sleep (lowest power consumption). **3:** The module only maintains connection with the AP during sleep, any unicast packet can wake the module. **4:** The module only keeps alive with the AP during sleep, and can only be woken up through the AP's interface/proc/hgic/wakeup. For server keep-alive functionality, it is recommended to set ps_mode=2.

	In some cases (e.g., performance testing), if you do not want the device to enter sleep, you can set ps_mode to 0. **6:** Regardless of the connection state, directly enter sleep, can only be woken by IO.
Example	hgic_raw_set_ps_mode(0);

3.4.5. set_ps_connect

API	hgic_raw_set_ps_connect(char period, char roundup);
Description	Set the sleep interval and maximum increment times for PS Connect. When the STA's WiFi module is in sleep mode and gets disconnected, it will wake up to reconnect to the AP. If the connection fails, the WiFi module will enter PS Connect mode: cyclic sleep/wake/reconnect. The intermediate sleep is to prevent excessive power consumption from continuous reconnections. **period:** 60; **roundup:** 3; For the 1st failed connection, sleep for 1 minute; for the 2nd failed connection, sleep for 2 minutes; for the 3rd failed connection, sleep for 3 minutes. After incrementing sleep time 3 times, it wraps around to the 1st interval, and this cycle repeats.
Example	hgic_raw_set_ps_connect(60,3);

3.4.6. set_wait_psmode

API	int hgic_raw_set_wait_psmode(char mode);
Description	Set the sleep mode when not connected. When the AP is powered off, the STA enters periodic sleep and can quickly wake up to connect when the AP powers on. Default 0 is the previous PS Connect, 1 is standby mode.
Example	hgic_raw_set_wait_psmode(1);

3.4.7. set_standby

API	int hgic_raw_set_standby(char channel, int period_ms);
Description	Set the agreed frequency point and time in standby mode (time unit changed to ms).
Example	hgic_raw_set_standby(2,1000);

3.4.8. set_aplost_time

API	int hgic_raw_set_aplost_time(int aplost_time);
Description	Set the time (in seconds) to detect AP loss during module sleep. If no beacon from the AP is received within the specified time, the AP is considered lost. The default value is 10 seconds.
Example	hgic_raw_set_aplost_time(10);

3.4.9. set_reassoc_wkhost

API	int hgic_raw_set_reassoc_wkhost(char enable);
Description	After detecting an AP anomaly in sleep mode, the module will restart scanning to connect to the AP. By default, the module will not notify the main control that it has reconnected to the AP. If you set reassoc_wkhost=1, the module will notify the main control after reconnecting to the AP. This interface is currently deprecated and not in use. To set the host wake-up, you can use set_wkhost_reasons.
Example	hgic_raw_set_reassoc_wkhost(1);

3.4.10. set_wakeup_io

API	int hgic_raw_set_wakeup_io(char wakeup_io, char edge);
Description	Set the module's sleep host to wake up the AH module's IO and IO **trigger edge:** 0: rising edge, 1: falling edge. If this command is not set, the mclr is used to wake up the AH module by default. **The corresponding IO numbers are:** IOA0~31: 0~31, IOB0~7: 32~39, mclr: 51. Example: Set the wake-up IO to IOA1, wake up on the rising edge.
Example	hgic_raw_set_wakeup_io(1,0);

3.4.11. set_autosleep_time

API	int hgic_raw_set_autosleep_time(char autosleep_time);
Description	Set the module's auto sleep time, in seconds, the default is 10 seconds, and the maximum value is 65 seconds. auto sleep refers to the module automatically entering sleep if it does not receive a command from the main control within a specified time

	after booting up. Example: Set the auto sleep time to 20 seconds.
Example	hgic_raw_set_autosleep_time(20);

3.4.12. set_dcdc13

API	int hgic_raw_set_dcdc13(char enable);
Description	Set whether the AH module uses external 1.3V DCDC power supply or internal LDO power supply. Using external 1.3V DCDC power can reduce the overall power consumption of the module by about 30%. **0:** Use internal LDO; **1:** Use external DCDC (hardware circuit must have 1.3V DCDC); **2:** Use external DCDC during sleep (hardware circuit must have 1.3V DCDC). If external DCDC affects RF performance during testing, you can set it to mode 2 to use internal LDO during normal operation and external DCDC during sleep for power savings. This parameter must be set according to the actual hardware circuit. If set to a non-zero value without an external DCDC, the module may fail to power on. If set to 0 with an external DCDC, power savings may not be achieved.
Example	hgic_raw_set_dcdc13(1);

3.4.13. set_ap_psmode_en

API	int hgic_raw_set_ap_psmode_en(char enable);
Description	Set AP low power enable, when you need to use AP low power, you need to enable this function for both AP and STA.
Example	hgic_raw_set_ap_psmode_en(1);

3.4.14. set_pa_pwrctl_dis

API	int hgic_raw_set_pa_pwrctl_dis(char disable);
Description	Set the PA power supply control logic switch when the module is in sleep. The PA power supply control logic is enabled by default. **0:** Turn off the power supply control logic, PA is always powered; **1:** Turn on the power supply control logic, PA is not powered during

	sleep, it needs to be used with hardware, that is, IOA30 controls RF power off during sleep. If the hardware does not have this control circuit reserved, the software will be ineffective even if it is turned on. In fact, the default value for this parameter is fine.
Example	hgic_raw_set_pa_pwrctl_dis(1);

3.4.15. set_dis_psconnect

API	int hgic_raw_set_dis_psconnect(char disable);
Description	If you do not want the STA to automatically enter sleep mode when not connected to the AP, you can disable the PS Connect mode. For example, in certain testing scenarios, it might be desirable for the device to remain awake when it is not connected. Currently, when the STA is in sleep mode and disconnects, it does not report this to the host. Therefore, it is recommended not to disable PS Connect mode in normal low-power devices. If it is disabled and the STA disconnects, it will continuously try to reconnect to the AP. If it fails to reconnect, it will rapidly drain the battery.
Example	hgic_raw_set_dis_psconnect(1);

3.4.16. set_wkdata_save_en

API	int hgic_raw_set_wkdata_save_en(char enable);
Description	The module can be set to save wake-up data sent by the server. After the host starts up, it can retrieve the cached wake-up data from the module via an interface. The module can cache up to four wake-up data packets.
Example	hgic_raw_set_wkdata_save_en(1);

3.4.17. get_wkdata_buff

API	int hgic_raw_get_wkdata_buff(void);
Description	Read the cached wake-up data from the module.
Example	int hgic_raw_get_wkdata_buff(void);

3.4.18. set_wkhost_reasons

API	int hgic_raw_set_wkhost_reasons(char *reasons, int count);
Description	Set which wake-up reasons need to wake up the host. STA wake-up reasons:

	1. Timer wake-up in a non-connected state (this reason generally does not need to wake up the host). 2. Unicast TIM wake-up, initiated by the AP or another device (generally needs to wake up the host). 3. Broadcast TIM wake-up, initiated by broadcast data (this reason has been removed). 4. Button IO wake-up (default MCLR pin, MCLR pulled for around 500us) (if pulled by the host, it may not need to wake up the host again). 5. Beacon loss wake-up (this reason generally does not need to wake up the host). 6. AP reboot detection wake-up (this reason generally does not need to wake up the host). 7. Heartbeat timeout wake-up (used in sleep mode 2). 8. Wake-up packet wake-up (used in sleep mode 2). 9. MCLR IO Reset wake-up (MCLR pulled for over 2ms in sleep mode causing a reset, this reason generally does not need to wake up the host). 10. LVD low voltage reset (this reason is not set to wake up the host by default). 11. PIR IO wake-up (requires special hardware support, otherwise not set). 12. AP API wake-up, executed by the AP through wnb_wakeup_sta or hgic_iwpriv_wakeup_sta (generally needs to wake up the host). 13. STA detected to be offline by the AP during sleep (this reason generally does not need to wake up the host). 14. Wake-up when connecting to AP in STANDBY state (needs to be set when using the STANDBY function).
Example	char reasons[] = {2,7,8,12,14}; hgic_raw_set_wkhost_reasons(reasons, 5);

3.4.19. wakeup_sta

API	int hgic_raw_wakeup_sta(char *mac);
Description	Wake up the STA with the specified MAC address.
Example	char addr[] = {11,22,33,44,54,65}; hgic_raw_wakeup_sta(addr);

3.4.20. set_ps_heartbeat

API	int hgic_raw_set_ps_heartbeat(unsigned int ipaddr, unsigned int dport, unsigned int period, unsigned int hb_tmo);

Description	Low power mode parameter interface file, used to set the IP address, port number, heartbeat period, and heartbeat timeout of the heartbeat server. **ipaddr:** IP address **dport:** Port number **period:** Heartbeat period, in seconds. **hb_tmo:** Heartbeat timeout, in seconds.
Example	unsigned int ipaddr = (192 << 24) | (168 << 16) | (0 << 8) | 100; hgic_raw_set_ps_heartbeat(ipaddr, 60001, 60, 300);

3.4.21. set_heartbeat_resp

API	int hgic_raw_set_heartbeat_resp(char *heartbeat_resp, int len);
Description	Set the heartbeat response content to the AH module. In SLEEP mode, when the AH module performs heartbeat interaction with the heartbeat server, the AH module detects the heartbeat response according to the packet content. If no response is received after sending the heartbeat, the AH module will continue to send the heartbeat to the server upon the next wake-up. If no response is received for seven consecutive heartbeat packets, the AH module will consider the connection to the server lost and notify the host of the network anomaly via IO.
Example	char *heartbeat_resp = “this is heartbeat response data for xxxx device”; hgic_raw_set_heartbeat_resp(heartbeat_resp, sizeof(heartbeat_resp));

3.4.22. set_ps_wkdata

API	int hgic_raw_set_ps_wkdata(char *ps_wkdata, int len);
Description	Set the wake-up packet content of the heartbeat server to the AH module. Upon recognizing the wake-up packet, the AH module will wake up the host via IO, and the AH module itself will also be awakened.
Example	char *ps_wkdata = “this is wakeup data for xxxxx device”; hgic_raw_set_ps_wkdata(ps_wkdata, sizeof(ps_wkdata));

3.4.23. set_wkdata_mask

API	int hgic_raw_set_wkdata_mask(unsigned short offset, unsigned char *mask, int mask_len);

Description	Set the match mask for the wake-up data.
Example	hgic_raw_set_wkdata_mask(offset, mask, mask_len);

3.5. Multicast Mode API

3.5.1.join_group

API	int hgic_raw_join_group(char *group_addr, char aid);
Description	After setting the WiFi module's operating mode to group, this command can be used to add the WiFi module to a multicast network. Once in a multicast network, the WiFi module will only receive data within that multicast network. All data communication will occur using the multicast address. If the operating mode is set to group but no multicast network is joined, all data communication will be conducted in broadcast form. Note: The JOINGROUP command must be set after configuring the GROUP mode. **Parameter Description:** **group_addr:** The address of the multicast network to join. **aid:** The AID of the device in the multicast network. Valid AID values: 1~N (N is the maximum number of STAs supported by the firmware). The AID of each device in the network should be unique. **Valid AID setting:** The WiFi module will periodically send a heartbeat in the multicast network to announce its presence to other WiFi modules. **Invalid AID setting:** The WiFi module will not send a heartbeat or notify other WiFi modules. If all devices set the AID to 0, the maximum number of STAs supported by the firmware is not limited.
Example	char group_addr[] = {11,22,33,44,54,65}; hgic_raw_join_group(group_addr,3);

3.5.2. set_mcast_txparam

API	int hgic_raw_set_mcast_txparam(char dupcnt, char txbw, char txmcs, char clearch);
Description	Set the multicast communication TX parameters: **dupcnt:** Number of times to retransmit multicast data. Default is to send once. After setting this parameter, each multicast data will be

	repeatedly sent n times. **tx_bw:** Set the TX bandwidth for multicast data. **tx_mcs:** Set the MCS for multicast data. **clearch:** Set whether to clear the channel before sending multicast data. This parameter is currently invalid and should be set to 0.
Example	int hgic_raw_set_mcast_txparam(3,8,7,0);

3.6. Relay Mode API

3.6.1. set_r_ssid

API	int hgic_raw_set_r_ssid(char *r_ssid);
Description	Set the SSID of the upper-level AP in relay mode, up to 32 characters. The usage requirements are the same as the ssid parameter.
Example	int hgic_raw_set_r_ssid("ah_relay");

3.6.2. set_r_psk

API	int hgic_raw_set_r_psk(char *r_psk);
Description	Set the password for the upper-level AP in relay mode. This key value is 64 hex characters. The usage requirements are the same as the wpa_psk parameter.
Example	hgic_raw_set_r_psk("1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef");

3.7. Roaming Function API

3.7.1. set_roaming

API	int hgic_raw_set_roaming(char enable, char same_freq);
Description	Enable the roaming feature. **enable=1:** Enable the roaming feature. **same_freq=1:** Set roaming devices to operate on the same frequency.
Example	int hgic_raw_set_roaming(1,1);

3.8. Dual Antenna Feature API

3.8.1. set_ant_auto

API	int hgic_raw_set_ant_auto(char en);
Description	Use automatic selection mode for dual antennas.
Example	int hgic_raw_set_ant_auto(1);

3.8.2. set_dual_ant

API	int hgic_raw_set_dual_ant(char en);
Description	Enable or disable antenna switching.
Example	int hgic_raw_set_dual_ant(1);

3.8.3. set_ant_sel

API	int hgic_raw_set_ant_sel(char en);
Description	Manually select antenna 0 or antenna 1. When manually selecting, enable antenna switching and turn off the automatic selection mode.
Example	int hgic_raw_set_ant_sel(1);

3.8.4. get_ant_sel

API	int hgic_raw_get_ant_sel(void);
Description	Get the antenna selection of the module.
Example	hgic_raw_get_ant_sel();

3.9. Debugging Feature API

3.9.1. set_dbginfo

API	int hgic_raw_set_dbginfo(char enable);
Description	Enable or disable firmware debugging information output. When enabled, firmware debugging information will be output to the WiFi driver and printed out. This feature is primarily used for capturing

	debugging information for problem analysis. **dbginfo=1 :** Enable firmware debugging information output. **dbginfo=0 :** Disable firmware debugging information output.
Example	hgic_raw_set_dbginfo(1);

3.9.2. set_sysdbg

API	int hgic_raw_set_sysdbg(char *sysdbg);
Description	Enable or disable certain categories of firmware debugging information output. 0 means disable the corresponding print, 1 means enable the corresponding print. The following are the categories of debugging information: **"heap,0" :** Heap information, default = 0. **"top,0" :** CPU usage information by each thread, default = 0. **"wnb,0" :** WNB layer (network layer statistics), default = 0. **"lmac,1" :** LMAC layer (air interface statistics), default = 1. LMAC statistics are enabled by default, and there are many; disable them if necessary. WNB statistics are disabled by default.
Example	hgic_raw_set_sysdbg("wnb,1");

3.9.3. set_assert_holdup

API	int hgic_raw_set_assert_holdup(char assert_holdup);
Description	Set whether to hold the system and pause operation when an assert is triggered.
Example	hgic_raw_set_assert_holdup(1);

3.9.4. set_atcmd

API	int hgic_raw_set_atcmd(char *atcmd);
Description	Send AT commands to the module via the host.
Example	hgic_raw_set_atcmd("at+mode=ap");

3.10. Other APIs

3.10.1. open

API	int hgic_raw_open(void);
Description	Activate AH module. The AH module is in the OPEN state by default. If the close API has been called, the device must be opened before use.
Example	hgic_raw_open();

3.10.2. close

API	int hgic_raw_close(void);
Description	Deactivate AH module. After deactivating the AH module, signal transmission stops, and the connection is terminated.
Example	hgic_raw_close();

3.10.3. save

API	int hgic_raw_save(void);
Description	Set AH module to save parameters (when AH module has configured NOR flash). The AH firmware automatically saves parameters by default. When parameters are modified, they are automatically saved to flash if a change is detected. If auto_save is set to 0, you need to call this save function to save parameters.
Example	int hgic_raw_save(void);

3.10.4. set_auto_save

API	int hgic_raw_set_auto_save(char enable);
Description	Configure AH module for automatic parameter saving (when AH module has configured NOR flash). After disabling the automatic save feature, parameters will only be saved by calling hgic_raw_save() (the parameter value is saved immediately).
Example	hgic_raw_set_auto_save(1);

3.10.5. set_radio_onoff

API	int hgic_raw_set_radio_onoff(char on);
Description	Used to control wifi radio on/off.
Example	hgic_raw_set_radio_onoff(1);

3.10.6. set_rtc

API	int hgic_raw_set_rtc(char on);
Description	Set Firmware Timer Function, in milliseconds, starting from 0.
Example	hgic_raw_set_rtc(1);

3.10.7. set_ether_type

API	int hgic_raw_set_ether_type(unsigned short ether_type);
Description	Set a type of Ethernet packet to filter out when received.
Example	hgic_raw_set_ether_type(1);

3.10.8. set_load_def

API	int hgic_raw_set_load_def(char reset_en);
Description	Restore Factory Settings **reset_en=1 :** Immediately restart after restoring factory settings to apply the parameters. **reset_en=0 :** Do not restart immediately; settings will take effect on the next reboot.
Example	hgic_raw_set_load_def(1);

3.10.9. mcu_reset

API	int hgic_raw_mcu_reset(void);
Description	Reboot Module
Example	hgic_raw_mcu_reset();

3.10.10. start_assoc

API	int hgic_raw_start_assoc(void);
Description	Switch to the initial connection state.

Example	hgic_raw_start_assoc();

3.10.11. send_customer_mgmt

API	int hgic_raw_send_customer_mgmt(char *dest, struct hgic_tx_info *info, char *data, int len);

Description	Send user-defined data.
	
	dest: Destination MAC address.
	hgic_tx_info *info: Defined as follows.
	`struct hgic_tx_info {`
	`  unsigned char  band;`
	`  unsigned char  tx_bw;`
	`  unsigned char  tx_mcs;`
	`  unsigned char  freq_idx: 5, antenna: 3;`
	`  unsigned int  tx_flags;`
	`  unsigned short  tx_flags2;`
	`  unsigned char  priority;`
	`  unsigned char  tx_power;`
	`};`
	Use NULL for default values if bw and mcs parameters are not specified
	
	Data: The data to be sent.
	len: The length of the data.

Example	char dest[] = {0x0,0x11,0x22,0x33,0x44,0x55}; hgic_raw_send_customer_mgmt(dest,NULL,"1234567890",10);

4. Non-OS WiFi Driver Firmware Download

The non-OS WiFi driver provides a firmware download feature, which is necessary when the WiFi module does not have flash memory. The firmware needs to be downloaded from the main controller.

The firmware download functionality requires the following three APIs:

1) **hgic_bootdl_parse_fw:** Parse firmware information.
2) **hgic_bootdl_request:** Control firmware download status.
3) **hgic_bootdl_send:** Send firmware data.

4.1. Firmware Information Parsing

Parse the firmware header information, including version number, chip ID, firmware length, AES encryption enablement, and CRC checksum. After successfully parsing the firmware, this API fills the struct hgic_bootdl with information for subsequent firmware sending and status control.

API	int hgic_bootdl_parse_fw(unsigned char *fw_data, struct hgic_bootdl *bootdl);
Parameters	fw_data: Firmware content. bootdl: Firmware locating parameters.
Return Values	0: Firmware parsing successful. -1: Firmware parsing failed.

4.2. Firmware Download Status Control

API	int hgic_bootdl_request(unsigned char cmd, struct hgic_bootdl *bootdl);
Parameters	**cmd:** Firmware loading command. HGIC_BOOTDL_CMD_ENTER: Start firmware loading. HGIC_BOOTDL_CMD_WRITE_MEM: Write firmware data. HGIC_BOOTDL_CMD_RUN: Firmware download complete, run the firmware. **bootdl:** Firmware loading status control information, which must be filled by the hgic_bootdl_parse_fw API.
Return Values	0: Execution successful. -1: Execution failed.

4.3. Sending Firmware Data

API	int hgic_bootdl_send(unsigned char *data, unsigned int len);
Parameters	**data:** Firmware data to be sent. **len:** Length of the data to be sent.
Return Values	0: Sending successful. -1: Sending failed.

4.4. Example Code

```
static uint8_t bootdl_buf[TEST_FW_FRAG_SIZE];
struct hgic_bootdl bootdl;
HGIC_RAW_RX_TYPE resp;

// Read firmware header information from flash
spi_flash_read(bootdl_buf,FLASH_FW_OFFEST,sizeof(struct hgic_fw_info_hdr));

// Check if the firmware can be parsed correctly
if (hgic_bootdl_parse_fw(bootdl_buf, &bootdl) != 0) {
    printf("Bootdl fail\r\n");
    while(1);
}

// Send start firmware loading request
hgic_bootdl_request(HGIC_BOOTDL_CMD_ENTER, &bootdl);

while (1) {
    // Read data from hardware to p_buf
    ......
    // Check for bootdl response
    resp = hgic_raw_rx(&p_buf, &p_len);

    if (resp == HGIC_RAW_RX_TYPE_BOOTDL_RESP) {

        // Handle special case if remaining firmware length is less than one frame length
        if (bootdl.offset + bootdl.frag_size > bootdl.fw_info.fw_size) {
            bootdl.frag_size = bootdl.fw_info.fw_size - bootdl.offset;
        }
        // Send write request
        hgic_bootdl_request(HGIC_BOOTDL_CMD_WRITE_MEM, &bootdl);

        // Read firmware from flash
        spi_flash_read(bootdl_buf,FLASH_FW_OFFEST+bootdl.fw_info.hdr_len+
    bootdl.offset, bootdl.frag_size);

        // Increase offset
        bootdl.offset += bootdl.frag_size;

        // Send firmware data
        hgic_bootdl_send(bootdl_buf, bootdl.frag_size);
```

```
        }
        if (bootdl.offset >= bootdl.fw_info.fw_size) {
            break; // Exit loop if sending is complete
        }
    }
    // After firmware loading is complete, run the firmware
    hgic_bootdl_request(HGIC_BOOTDL_CMD_RUN, &bootdl);
```

5. Non-OS WiFi Driver Firmware Upgrade

The non-OS WiFi driver provides a firmware upgrade feature for updating the firmware in the WiFi module's flash. The firmware upgrade functionality requires the following two functions:

1) hgic_ota_parse_fw: Parse firmware information.
2) hgic_ota_send_packet: Send firmware data.

5.1. Firmware Information Parsing

API	int hgic_ota_parse_fw(unsigned char *fw_data, struct hgic_ota *ota);
Description	**fw_data:** Firmware content. **ota:** Firmware upgrade parameters. Parse the firmware header information, including version number, chip ID, firmware length, AES encryption enablement, and CRC checksum. After successfully parsing the firmware, this API fills the struct hgic_ota with information for subsequent firmware sending and status control.
Return Values	0: Firmware parsing successful. -1: Firmware parsing failed.

5.2. Firmware Data Sending

API	int hgic_ota_send_packet(struct hgic_ota *ota, unsigned char *data, unsigned int len);
Parameters	**ota:** Firmware upgrade status control information. This parameter must be filled by the hgic_ota_parse_fw API. **data:** Firmware data to be sent. **len:** Length of the data to be sent.
Return Values	0: Sending successful. -1: Sending failed.
Example	hgic_ota_send_packet(&ota, ota_buf, ota.frag_size);

5.3. Example Code

```
static uint8_t ota_buf[TEST_FW_FRAG_SIZE + 28]; // 28 bytes for firmware header
struct hgic_ota ota;

// Read firmware header information from flash
spi_flash_read(ota_buf, 0, sizeof(struct hgic_fw_info_hdr));

// Check if the firmware header information can be parsed correctly
if (hgic_ota_parse_fw(ota_buf, &ota) != 0) {
```

```
        printf("OTA fail\r\n");
        while(1);
    }

    /* Loop to send firmware */

    while (1) {
        // Read data from hardware to p_buf
        ......
        // Check for OTA response
        resp = hgic_raw_rx(&p_buf, &p_len);

        if (resp == HGIC_RAW_RX_TYPE_OTA_RESP) {
            // Handle special case if remaining firmware length is less than one frame length
            if (ota.offset + ota.frag_size > ota.fw_info.hdr_len + ota.fw_info.fw_size) {
                ota.frag_size = ota.fw_info.hdr_len + ota.fw_info.fw_size - ota.offset;
            }
            // Read firmware from flash (28 bytes for firmware header)
            spi_flash_read(ota_buf+28, ota.offset, ota.frag_size);

            // Send firmware
            hgic_ota_send_packet(&ota, ota_buf, ota.frag_size);

            // Increase offset after sending a packet
            ota.offset += ota.frag_size;

            if (ota.offset >= ota.fw_info.hdr_len + ota.fw_info.fw_size) {
                break;// If offset exceeds firmware length, sending is complete
            }
        }
    }
```

6. Non-OS WiFi Driver Data Transmission and Reception

6.1. Data Transmission

The non-OS WiFi driver provides two data transmission APIs for sending different types of data.

6.1.1. hgic_raw_send

This API is used to send raw data (non-Ethernet data). When the main controller does not run a TCP/IP stack, this API can be used for data transmission. It encapsulates the data into Ethernet frames before sending.

API	int hgic_raw_send(char *dest, char *data, int len);
Parameters	**dest:** Destination address of the data (the application needs to manage device MAC addresses). If the destination address is empty, it will be a broadcast packet. **data:** Raw data to be sent. **len:** Length of the data to be sent.
Return Values	0: Success -1: Failure

Example	// Sending broadcast data
	char dest[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
	hgic_raw_send(dest, xxx_data, xxx_data_len);
	// Sending unicast data
	char dev_addr[6]; // dev_addr must be the address of a specific device
	hgic_raw_send(dev_addr, xxx_data, xxx_data_len);

6.1.2. hgic_raw_send_ether

This API is used to send Ethernet data. When the main controller runs a TCP/IP stack, this API is required. The driver porting needs to interface with the TCP/IP stack.

API	int hgic_raw_send_ether(char *data, int len);
Parameters	**data:** Ethernet format data to be sent. **len:** Length of the data to be sent.
Return Values	0: Success -1: Failure
Example	// Called by the output function of LwIP static err_t netif_output(struct netif *netif, struct pbuf *p){

```
    hgic_raw_send_ether(p->payload, p->len);
    return ERR_OK;
}
```

6.2. Data Reception

The non-OS WiFi driver provides a single entry function for data reception:

HGIC_RAW_RX_TYPE **hgic_raw_rx**(unsigned char **data, unsigned int *len)

After the application receives data from the underlying interface driver, this function is called for data processing. The return value of this interface identifies the type of received data and modifies the parameters **data and *len. The return value data types are defined as follows:

```
typedef enum {
    HGIC_RAW_RX_TYPE_IGNORE       = 0,  //invalid data, just ignore it.
    HGIC_RAW_RX_TYPE_DATA         = 1,  //recieved data.
    HGIC_RAW_RX_TYPE_CMD_RESP     = 2,  //recieved cmd response.
    HGIC_RAW_RX_TYPE_BOOTDL_RESP  = 3,  //bootdl response.
    HGIC_RAW_RX_TYPE_OTA_RESP     = 4,  //ota response.
    HGIC_RAW_RX_TYPE_EVENT        = 5,  //firmware event.
} HGIC_RAW_RX_TYPE;
```

- **HGIC_RAW_RX_TYPE_IGNORE**: Received invalid data that can be ignored.
- **HGIC_RAW_RX_TYPE_DATA**: Received application data. The parameters data and len will be modified: **data points to the start of valid data, and *len is set to the length of the valid data.
- **HGIC_RAW_RX_TYPE_CMD_RESP**: Received the result of CMD execution. The content of **data is modified to either +RESP:OK on success or +RESP:Fail on failure. Additional processing code can be added in the hgic_raw_rx_cmd_resp function.
- **HGIC_RAW_RX_TYPE_BOOTDL_RESP**: Received the result of a firmware download command execution.
- **HGIC_RAW_RX_TYPE_OTA_RESP**: Received the result of an OTA operation command execution.
- **HGIC_RAW_RX_TYPE_EVENT:** Received an Event message generated by the firmware. Additional processing code can be added in the hgic_raw_rx_event function.

Example Code for Data Reception:

```
char buff[1024];
char *ptr = buff;
int    len = 0;
int    ret = 0;
char   *src_addr;

//read data from hardware interface
......

//hgic raw parse data
```

```
ret = hgic_raw_rx(&ptr, &len);
if(ret == HGIC_RAW_RX_TYPE_DATA){ // rx data
src_addr = ptr+6;    //get source address
printf("rx %d bytes data from "MACSTR"\r\n", len-14, MAC2STR(src_addr));

lwip_app_input(ptr, len); //deliver the data to tcp/ip stack

ptr += 14; len -= 14; //or skip ether frame header
//application proc data
.....

}else if(ret == HGIC_RAW_RX_TYPE_CMD_RESP){
if(strstr(ptr, "+RESP:OK"){
printf("cmd run success\r\n");
}
}
```

7. Non-OS WiFi Driver Porting Description

Porting the non-OS WiFi driver requires completing the following two tasks:

1) Interfacing with the underlying driver.
2) Interfacing with the upper-layer application or TCP/IP stack.

7.1. Interfacing with the Underlying Driver

The non-OS WiFi driver interacts with the underlying driver through two interfaces: raw_send and hgic_raw_rx.

1) **extern int raw_send(char *data, int len);**
   Data sending interface. The non-OS WiFi driver calls this function to send data. This function must be implemented on the specific platform, invoking the SPI/UART driver to send data.

2) **HGIC_RAW_RX_TYPE hgic_raw_rx(char **data, int *len)**
   Data reception processing function. When the SPI/UART driver receives data, call this function for parsing and processing. The return value of this function defines the type of received data.

```
typedef enum {
    HGIC_RAW_RX_TYPE_IGNORE      = 0,  //invalid data, just ignore it.
    HGIC_RAW_RX_TYPE_DATA        = 1,  //recieved data.
    HGIC_RAW_RX_TYPE_CMD_RESP    = 2,  //recieved cmd response.
    HGIC_RAW_RX_TYPE_BOOTDL_RESP = 3,  //bootdl response.
    HGIC_RAW_RX_TYPE_OTA_RESP    = 4,  //ota response.
    HGIC_RAW_RX_TYPE_EVENT       = 5,  //firmware event.
} HGIC_RAW_RX_TYPE;
```

The application needs to focus on: HGIC_RAW_RX_TYPE_DATA. This type indicates that valid data has been received. After the function executes, data and len will be modified as follows:

data: The modified value points to the Ethernet frame header.

len: The modified value is the length of the Ethernet frame header plus the length of the valid data.

API	int raw_send(unsigned char *data, unsigned int len);
Purpose	The non-OS WiFi driver packages the data in HGIC FRM format and calls this function to send data. This function interfaces with the hardware's underlying send function, invoking the SPI/UART driver for data transmission.
Return Values	0: Send successful. -1: Send failed.
Example	int raw_send(unsigned char *data, unsigned int len)

	{
	return hgic_sdspi_write(0, data, len);
	}

API	HGIC_RAW_RX_TYPE hgic_raw_rx(unsigned char **data, unsigned int *len);
Purpose	Data reception processing function. When the SPI/UART driver receives data, it calls this function to parse and process it. The return value of this function defines the type of received data.

Return Values	HGIC_RAW_RX_TYPE_IGNORE = 0, //invalid data, just ignore it.
	HGIC_RAW_RX_TYPE_DATA = 1,//recieved data.
	HGIC_RAW_RX_TYPE_CMD_RESP = 2,//recieved cmd response.
	HGIC_RAW_RX_TYPE_BOOTDL_RESP = 3,//bootdl response.
	HGIC_RAW_RX_TYPE_OTA_RESP = 4,//ota response.

7.2. Interfacing with the Upper-layer Module

The non-OS WiFi driver interfaces with the upper-layer module through data send interfaces: hgic_raw_send and hgic_raw_send_ether.

API	int hgic_raw_send_ether(char *data, int len);
Parameters	**data:** Ethernet format data to be sent. **len:** Length of the data to be sent.
Return Values	0: Send successful. -1: Send failed.

Example	// Called by the output function of LwIP
	static err_t netif_output(struct netif *netif, struct pbuf *p){
	hgic_raw_send_ether(p->payload, p->len);
	return ERR_OK;
	}

API	int hgic_raw_send(char *dest, char *data, int len);
Parameters	**dest:** Destination address of the data (the application needs to manage the MAC addresses of devices). If the destination address is empty, it will be a broadcast packet. **data:** Raw data to be sent. **len:** Length of the data to be sent.
Return Values	0: Send successful. -1: Send failed.

Example	// Sending broadcast data
	char dest[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
	hgic_raw_send(dest, xxx_data, xxx_data_len);

```
// Sending unicast data
char dev_addr[6];    // dev_addr must be the address of a specific device
hgic_raw_send(dev_addr, xxx_data, xxx_data_len);
```

7.3. Integrating the SPI Interface

The non-OS WiFi driver provides two methods to interface with the SPI:

1) Using the SPI interface with the SDIO protocol.
2) Using the conventional SPI Master/Slave communication mode.

Both methods offer encapsulation and example code.

7.3.1. Using the SPI Interface with the SDIO Protocol

When using the SPI interface with the SDIO protocol, the TaiXin AH module acts as the SDIO slave (operating in SPI mode), and the MCU acts as the SPI master.

The entire SPI communication follows the SDIO 2.0 standard protocol. Detailed protocol specifications can be found in the documents "SD Specifications Part 1 Physical Layer Specification Version 2.00" and "SD Specifications Part E1 SDIO Specification Version 2.00".

The TaiXin AH module's SDIO slave supports high-speed mode, with clock speeds up to 50MHz.

The host SPI interface uses standard SPI pins (CLK, CS, MOSI, MISO) and an interrupt pin (INTIO) for notifying the host to receive data. The SDIO SPI mode interface on the slave side uses standard SDIO pins (CLK, DAT3, CMD, DAT0, DAT1). The pin connections are as follows:

Pin	SDIO	SPI
1	DAT3	CS
2	CMD	MOSI
3	CLK	CLK
4	DAT0	MISO
5	DAT1	INTIO

Except for the CLK pin, other IOs require pull-up resistors to ensure stable communication. Generally, the TaiXin AH module's SDIO interface already includes these pull-up resistors, so the host SPI driver does not need to enable internal pull-ups.

7.3.1.1. hgic_sdspi Module Description

The non-OS WiFi driver provides the `hgic_sdspi` module, which implements the SDIO

protocol and supports two modes: full-duplex (using hgic_sdspi_v2.c) and half-duplex (using hgic_sdspi.c).

Users must implement the following APIs to interface with the SPI hardware driver:

- **spidrv_write**: Interface with the SPI hardware driver to implement data sending.
- **spidrv_read**: Interface with the SPI hardware driver to implement data reading.
- **spidrv_cs** : Interface with the SPI hardware CS functionality.
- **spidrv_hw_crc**: Interface with the hardware CRC functionality. If hardware CRC is not supported, this function should return 0.

7.3.1.2. hgic_sdspi Porting Work

API	void spidrv_write(void *priv, char *data, int len, char dma_flag);
Description	Interface with the SPI hardware driver for the write operation, defaulting to CPU write functionality, with an option to implement DMA.

Example	void spidrv_write(void *priv, unsigned char *data, unsigned int len, char dma_flag){
	  if (dma_flag) {
	  HAL_SPI_TransmitReceive_DMA(&hspi2, (uint8_t *)data, (uint8_t *)temp_buf, len);
	  while (!dma2_done);
	  dma2_done = 0;
	  } else {
	  HAL_SPI_TransmitReceive(&hspi2, (uint8_t *)data, (uint8_t *)temp_buf, len, 0xffff);//Interface with the hspi2
	  }
	}

API	void spidrv_read(void *priv, char *data, int len, char dma_flag);
Description	Interface with the SPI hardware driver for the read operation, defaulting to CPU read functionality, with an option to implement DMA. In full-duplex mode, data sent while reading SPI must be 0xFF.

Example	void spidrv_read(void *priv, unsigned char *data, unsigned int len, char dma_flag)
	{
	  if (dma_flag) {
	  memset(temp_buf, 0xff, len);
	  HAL_SPI_TransmitReceive_DMA(&hspi2, (uint8_t *)temp_buf, (uint8_t *)data, len);
	  while (!dma2_done);
	  dma2_done = 0;
	  } else {
	  HAL_SPI_TransmitReceive(&hspi2, (uint8_t *)temp_buf, (uint8_t *)data, len, 0xffff);

	} }

API	void spidrv_cs(void *priv, char enable);
Description	Control the CS IO to implement the SPI CS functionality.

	void spidrv_cs(void *priv, char enable)
	{
	if (enable) {
	HAL_GPIO_WritePin(SPI_CS_GPIO_Port, SPI_CS_Pin, GPIO_PIN_RESET);
	} else {
	HAL_GPIO_WritePin(SPI_CS_GPIO_Port, SPI_CS_Pin, GPIO_PIN_SET);
	}
	}

API	void spidrv_hw_crc(void *priv, char *data, int len, char flag);
Description	Implement the SPI data CRC check function. If hardware CRC is available, it can improve data reliability at the cost of some communication efficiency. Without hardware CRC, this function can be an empty function returning 0. The parameter flag indicates whether the data needs a 7-bit command CRC or a 16-bit data CRC.

7.3.1.3. hgic_sdspi Usage Description

The non-OS WiFi driver has integrated the hgic_raw and hgic_sdspi modules, but users need to implement initialization and data reception flow control themselves. The driver provides example code to reference for porting.

hgic_sdspi module API description:

API	int hgic_sdspi_init(void *priv);
Description	Initialize the interface, set the SDIO slave into SPI mode, configure the block size, enable interrupt functionality, and enable high-speed mode.

API	int hgic_sdspi_init(void *priv);
Description	Initialize the interface, set the SDIO slave into SPI mode, configure the block size, enable interrupt functionality, and enable high-speed mode.

API	int hgic_sdspi_detect_alive(void *priv);

Description	Interface status detection function. Check if the interface is normal and reinitializes it if necessary.

API	int hgic_sdspi_write(void *priv, char *data, int len);
Description	Data write function: The host sends data to the TaiXin AH module and returns the actual length of data written.

API	int hgic_sdspi_read(void *priv, char *buf, int len);
Description	Data read function. The host reads data from the TaiXin AH module. When the module wants to send data to the host, it will pull down the SDIO DAT1 line. The host can configure a falling edge interrupt to receive data in a timely manner or periodically call this function to attempt to receive data. The function returns a non-zero, non-negative value on successful data reception. Before calling the function, the length of data to be received is unknown; the user must reserve enough space in the buf parameter according to the actual communication content and indicate the maximum receive length in the len parameter. The function returns the actual read length on success. The TaiXin AH module will consider a data send signal timed out after 2 seconds and reset the SDIO state machine. Please receive data in a timely manner to avoid communication issues.

The SDIO slave will initialize its internal state machine on communication abnormalities. However, after initialization, it defaults to SD mode. The host SPI needs to reinitialize to set the SDIO slave back into SPI mode. Therefore, an interface check function, hgic_sdspi_detect_alive, is provided to periodically check interface status and restore normal communication.

7.4. Interfacing the UART Interface

The UART interface is used for data transmission, with the TaiXin AH module typically using UART0 for communication with the host. Another UART interface is used for debugging prints and AT commands, detailed in the "TaiXin AH Module AT Command Development Guide."

The default UART settings are:

- **Baud Rate:** 115200
- **Data Bits:** 8 bits
- **Stop Bits:** 1 bit
- **Parity:** None