


## 泰芯 non-os WiFi 驱动开发指南





保密等级	A	泰芯 non-os WiFi 驱动开发指南	文件编号	
发行日期	2025/12/4		文件版本	V2.4


修订记录

日期	版本	描述	修订人
2025-12-4	V2.4	修改 hgic_raw_send 的地址说明； 修改 set_sysdbg 里面 lmac 打印的说明；	WY
2024-5-14	V2.3	修改 set_wakeup_io 的说明；	WY
2023-10-3	V2.2	修改 set_wakehost_reasons 的说明； 修改 set_pri_chan 和 set_cust_iso 的说明；	WY
2023-7-28	V2.1	增加 send_customer_mgmt 的说明	WY
2023-7-7	V2.0	删除组播加密的设置	WY
2023-5-23	V1.9	修改组播参数的描述	WY
2023-4-7	V1.8	修改 get_bssid	DY
2023-4-6	V1.7.1	修改页眉页脚	WY
2023-1-30	V1.7	增加新添加的 API 说明，整理章节顺序	CWY
2022-10-21	V1.6	增加 OTA 相关描述	CWY
2022-9-22	V1.5	增加低功耗相关的参数，和其他之前未描述的参数	CWY
2022-7-6	V1.4.2	修改文件名	WY
2022-2-18	V1.4.1	修改 logo	XYJ
2021-07-27	V1.4	添加 hgic_raw 使用说明	DY
2021-07-17	V1.3	修改 spidrv_read 的说明：全双工读取 SPI 数据同时发送的数据必须为 0xFF。	LHY
2021-02-05	V1.2	移除旧版 SPI 接口，增加新版 SPI 接口	LHY
2020-12-05	V1.1	增加 SPI 的 pin 说明	LHY
2020-10-20	V1.0	第一版	LHY

保密等级	A	泰芯 non-os WiFi 驱动开发指南	文件编号	
发行日期	2025/12/4		文件版本	V2.4
目录				
泰芯 non-os WiFi 驱动开发指南 .....1				
1 概述 .....1				
2 non-os 驱动工作原理 .....2				
2.1 non-os 驱动框图 .....2				
2.2 non-os 驱动工作流程 .....3				
2.2.1 数据发送流程 .....3				
2.2.2 数据接收流程 .....3				
3 non-os WiFi 驱动参数设置 .....5				
3.1 组网基础功能 API .....5				
3.1.1 set_ssid .....5				
3.1.2 set_key_mgmt .....5				
3.1.3 set_wpa_psk .....5				
3.1.4 set_freq_range .....5				
3.1.5 set_bss_bw .....6				
3.1.6 set_acs .....6				
3.1.7 set_mac .....6				
3.1.8 set_chan_list .....6				
3.1.9 set_mode .....7				
3.1.10 set_pairing .....7				
3.1.11 set_pair_autostop .....7				
3.2 组网高级功能 API .....8				
3.2.1 set_txpower .....8				
3.2.2 set_tx_mcs .....8				
3.2.3 set_bss_max_idle .....8				
3.2.4 set_channel .....8				
3.2.5 unpair .....9				
3.2.6 set_paired_stas .....9				
3.2.7 set_auto_chswitch .....9				
3.2.8 set_primary_chan .....10				
3.2.9 set_mac_filter_en .....10				
3.2.10 set_supper_pwr .....10				
3.2.11 set_acktmo .....10				
3.2.12 set_beacon_int .....11				
3.2.13 set_heartbeat_int .....11				
3.2.14 set_ap_chan_switch .....11				
3.2.15 set_agg_cnt .....11				
3.2.16 set_ap_hide .....12				
3.2.17 set_max_txcnt .....12				
3.2.18 set_dup_filter_en .....12				
		珠海泰芯半导体有限公司 TaiXin Semiconductor Co., Limited		
版权所有侵权必究 Copyright © 2025 by TaiXin Semiconductor All rights reserved				

保密等级	A	泰芯 non-os WiFi 驱动开发指南	文件编号	
发行日期	2025/12/4		文件版本	V2.4
<div>3.2.19 set_dis_lvl_m2u ..... 12</div> <div>3.2.20 set_kick_assoc ..... 12</div> <div>3.2.21 set_connect_paironly ..... 13</div> <div>3.2.22 set_disassoc_sta ..... 13</div> <div>3.2.23 set_sta_freqinfo ..... 13</div> <div>3.2.24 set_cust_isolation ..... 14</div> <div>3.3 状态查询 API ..... 14</div> <div>3.3.1 get_fwinfo ..... 14</div> <div>3.3.2 get_connect_state ..... 14</div> <div>3.3.3 get_mode ..... 14</div> <div>3.3.4 get_sta_list ..... 14</div> <div>3.3.5 get_bgrssi ..... 15</div> <div>3.3.6 scan ..... 15</div> <div>3.3.7 get_scan_list ..... 15</div> <div>3.3.8 get_temperature ..... 15</div> <div>3.3.9 get_ssid ..... 16</div> <div>3.3.10 get_bssid ..... 16</div> <div>3.3.11 get_wpa_psk ..... 16</div> <div>3.3.12 get_sta_count ..... 16</div> <div>3.3.13 get_txpower ..... 16</div> <div>3.3.14 get_aggcnt ..... 17</div> <div>3.3.15 get_freq_range ..... 17</div> <div>3.3.16 get_chan_list ..... 17</div> <div>3.3.17 get_bss_bw ..... 17</div> <div>3.3.18 get_key_mgmt ..... 17</div> <div>3.3.19 get_module_type ..... 18</div> <div>3.3.20 get_dhcpc_result ..... 18</div> <div>3.3.21 get_disassoc_reason ..... 18</div> <div>3.3.22 get_rtc ..... 18</div> <div>3.3.23 get_center_freq ..... 18</div> <div>3.3.24 get_wakeup_reason ..... 19</div> <div>3.4 低功耗相关 API ..... 19</div> <div>3.4.1 sleep ..... 19</div> <div>3.4.2 set_wkio_mode ..... 19</div> <div>3.4.3 set_dtim_period ..... 19</div> <div>3.4.4 set_ps_mode ..... 20</div> <div>3.4.5 set_ps_connect ..... 20</div> <div>3.4.6 set_wait_psmode ..... 21</div> <div>3.4.7 set_standby ..... 21</div> <div>3.4.8 set_aplost_time ..... 21</div>				
		珠海泰芯半导体有限公司 TaiXin Semiconductor Co., Limited		
版权所有侵权必究 Copyright © 2025 by TaiXin Semiconductor All rights reserved				

保密等级	A	泰芯 non-os WiFi 驱动开发指南	文件编号	
发行日期	2025/12/4		文件版本	V2.4
<div>3.4.9 set_reassoc_wkhost.....21</div> <div>3.4.10 set_wakeup_io.....22</div> <div>3.4.11 set_autosleep_time.....22</div> <div>3.4.12 set_dcdc13.....22</div> <div>3.4.13 set_ap_psmode_en.....23</div> <div>3.4.14 set_pa_pwrctl_dis.....23</div> <div>3.4.15 set_dis_psconnect.....24</div> <div>3.4.16 set_wkdata_save_en.....24</div> <div>3.4.17 get_wkdata_buff.....24</div> <div>3.4.18 set_wkhost_reasons.....24</div> <div>3.4.19 wakeup_sta.....25</div> <div>3.4.20 set_ps_heartbeat.....25</div> <div>3.4.21 set_heartbeat_resp.....26</div> <div>3.4.22 set_ps_wkdata.....26</div> <div>3.4.23 set_wkdata_mask.....27</div> <div>3.5 组播模式 API.....27</div> <div>3.5.1 join_group.....27</div> <div>3.5.2 set_mcast_txparam.....28</div> <div>3.6 中继模式 API.....28</div> <div>3.6.1 set_r_ssid.....28</div> <div>3.6.2 set_r_psk.....28</div> <div>3.7 漫游功能 API.....29</div> <div>3.7.1 set_roaming.....29</div> <div>3.8 双天线功能 API.....29</div> <div>3.8.1 set_ant_auto.....29</div> <div>3.8.2 set_dual_ant.....29</div> <div>3.8.3 set_ant_sel.....29</div> <div>3.8.4 get_ant_sel.....30</div> <div>3.9 调试功能 API.....30</div> <div>3.9.1 set_dbginfo.....30</div> <div>3.9.2 set_sysdbg.....30</div> <div>3.9.3 set_assert_holdup.....31</div> <div>3.9.4 set_atcmd.....31</div> <div>3.10 其他 API.....31</div> <div>3.10.1 open.....31</div> <div>3.10.2 close.....31</div> <div>3.10.3 save.....31</div> <div>3.10.4 set_auto_save.....32</div> <div>3.10.5 set_radio_onoff.....32</div> <div>3.10.6 set_rtc.....32</div>				
		珠海泰芯半导体有限公司 TaiXin Semiconductor Co., Limited		
版权所有侵权必究 Copyright © 2025 by TaiXin Semiconductor All rights reserved				

保密等级	A	泰芯 non-os WiFi 驱动开发指南	文件编号	
发行日期	2025/12/4		文件版本	V2.4
<div>3. 10. 7 set_ether_type..... 32</div> <div>3. 10. 8 set_load_def..... 33</div> <div>3. 10. 9 mcu_reset..... 33</div> <div>3. 10. 10 start_assoc ..... 33</div> <div>3. 10. 11 send_customer_mgmt ..... 33</div> <div>4 non-os WiFi 驱动固件下载 ..... 35</div> <div>4. 1 固件信息解析 ..... 35</div> <div>4. 2 固件下载状态控制 ..... 35</div> <div>4. 3 发送固件数据 ..... 36</div> <div>4. 4 示例代码 ..... 36</div> <div>5 non-os WiFi 驱动固件升级 ..... 38</div> <div>5. 1 固件信息解析 ..... 38</div> <div>5. 2 固件数据发送 ..... 38</div> <div>5. 3 示例代码 ..... 39</div> <div>6 non-os WiFi 驱动数据收发 ..... 40</div> <div>6. 1 发送数据 ..... 40</div> <div>6. 1. 1 hgic_raw_send ..... 40</div> <div>6. 1. 2 hgic_raw_send_ether..... 40</div> <div>6. 2 接收数据 ..... 41</div> <div>7 non-os WiFi 移植说明 ..... 43</div> <div>7. 1 对接底层驱动 ..... 43</div> <div>7. 2 对接上层模块 ..... 44</div> <div>7. 3 对接 SPI 接口 ..... 45</div> <div>7. 3. 1 SPI 接口运行 SDIO 协议 ..... 45</div> <div>7. 4 对接 UART 接口 ..... 49</div>				
		珠海泰芯半导体有限公司 TaiXin Semiconductor Co., Limited		
版权所有侵权必究 Copyright © 2025 by TaiXin Semiconductor All rights reserved				

## 1 概述

泰芯 non-os WiFi 驱动适用于低速率传输需求的产品方案，对主控芯片的软硬件环境要求较低，支持使用 SPI/UART 接口连接 WiFi 模块。

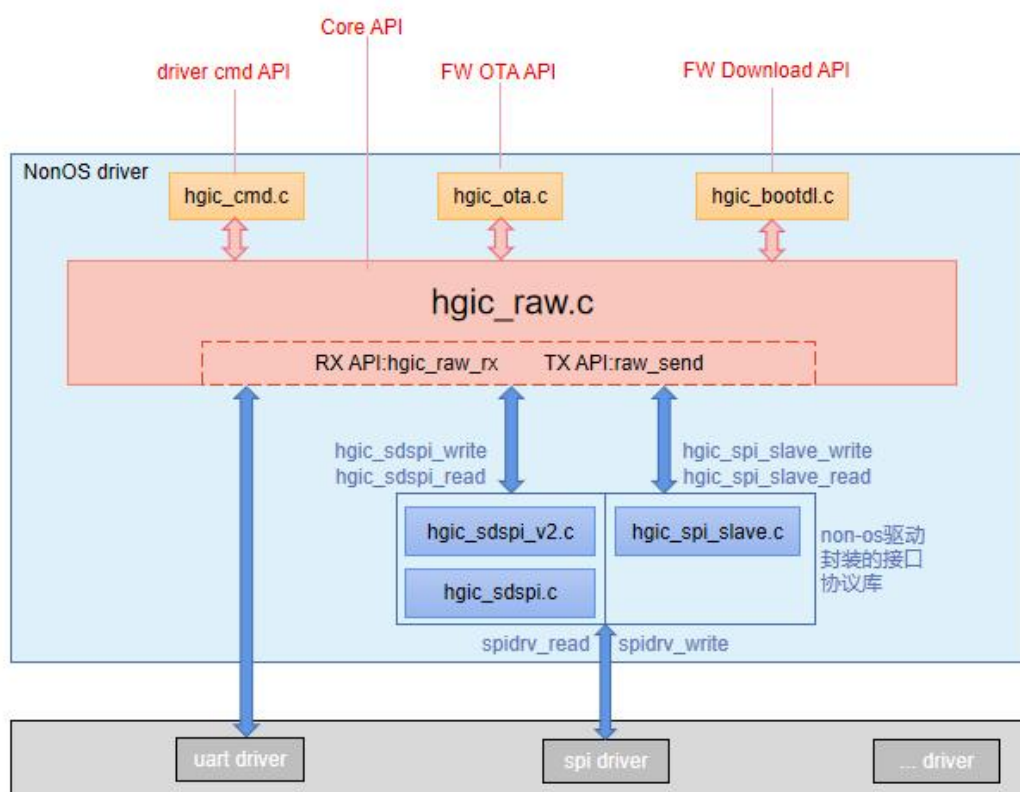
目前 non-os 驱动支持 AH 模块，WiFi 固件类型为 vx.x.x.5。

## 2 non-os 驱动工作原理

### 2.1 non-os 驱动框图

non-os WiFi 驱动框图如下图所示，由以下几个模块组成：

- **参数设置接口：**hgic\_cmd.c，该文件提供了驱动参数设置 API，具体 API 使用说明请参考第 3 章的驱动[参数设置 API](#)说明。
- **固件升级模块：**hgic\_ota.c，该文件提供了 WiFi 模块固件升级功能。对于无 flash 的 WiFi 模块，该功能无效。
- **固件下载模块：**hgic\_bootdl.c，该文件提供了通过接口下载 WiFi 固件到 WiFi 模块运行的功能。当 WiFi 模块没有 flash，或者 flash 中没有固件时，使用该模块可以下载固件运行。
- **数据收发模块：**hgic\_raw.c，该文件是 non-os 驱动的核心模块，负责数据的收发处理，包括命令数据，事件数据，应用数据。发送时 hgic\_raw.c 将各种数据封装成 HGIC FRM 格式；接收时 hgic\_raw.c 解析 HGIC FRM 格式，并进行解封装处理。



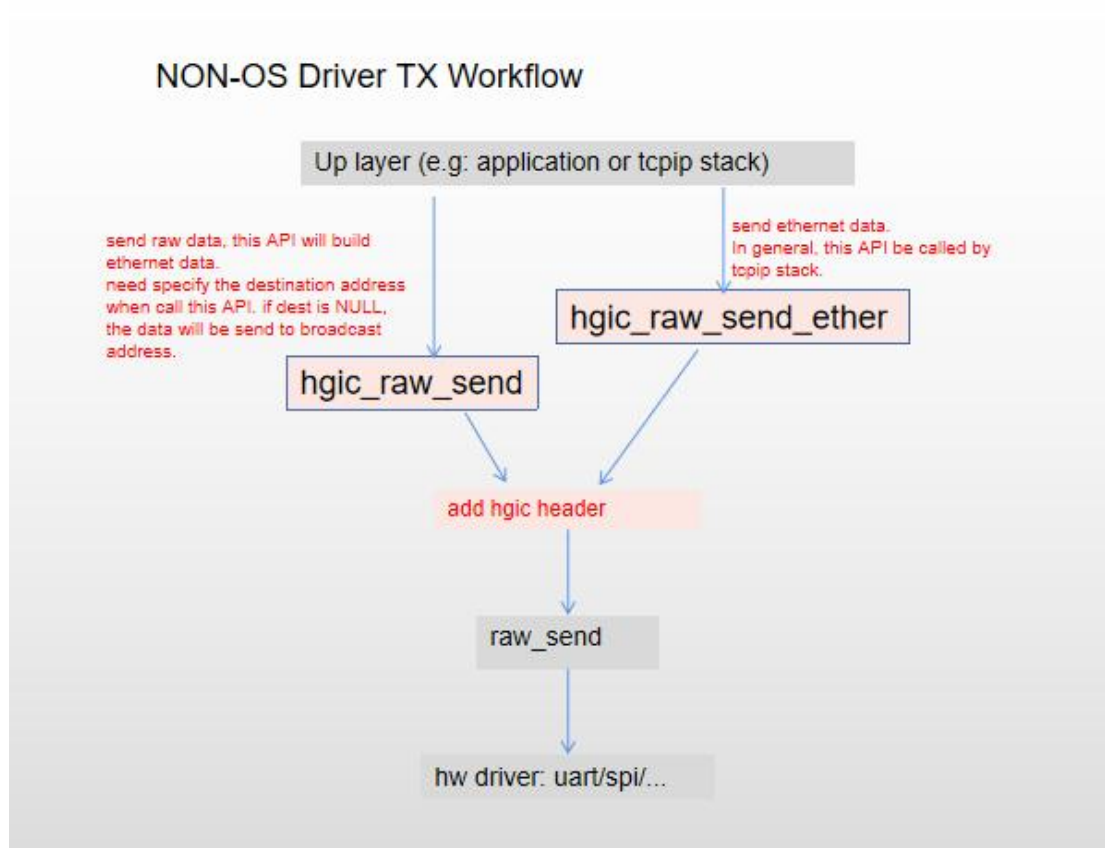


## 2.2 non-os 驱动工作流程

### 2.2.1 数据发送流程

non-os WiFi 驱动数据发送流程如下图所示，提供了 2 个发送 API：

- `hgic_raw_send`: 该 API 用于发送原始数据（非以太网数据）。当主控没有运行 `tcp/ip` 协议栈时，可以使用该 API 进行数据发送。该 API 将会把数据封装为以太网帧，然后再进行发送。
- `hgic_raw_send_ether`: 该 API 用于发送以太网数据。当主控运行了 `tcp/ip` 协议栈时需要使用该 API。



### 2.2.2 数据接收流程

non-os WiFi 驱动的数据接收流程如下图所示，驱动提供了 1 个接收入口函数：`hgic_raw_rx`。

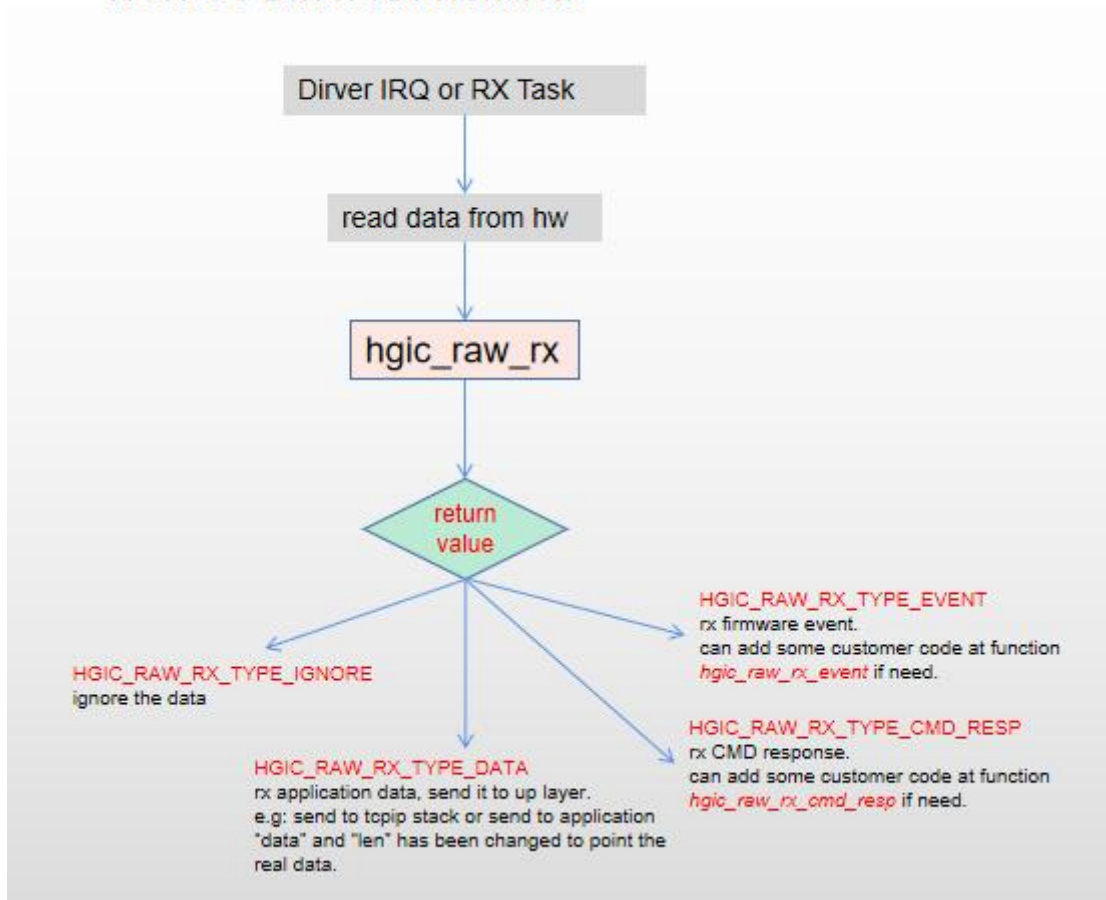
底层接口驱动接收到数据后，把数据输入到该接口进行处理，该接口的

返回值会标识出接收的数据类型，数据类型的定义如下：

```
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;
```

数据接收代码需要根据返回值类型，进行相应的处理。

### NON-OS Driver RX Workflow



### 3 non-os WiFi 驱动参数设置

#### 3.1 组网基础功能 API

##### 3.1.1 set\_ssid

API	int hgic_raw_set_ssid(char *ssid);
说明	设置 ssid, ssid 最长 32 个字符。
示例	hgic_raw_set_ssid("hgic_ah_test");

##### 3.1.2 set\_key\_mgmt

API	int hgic_raw_set_key_mgmt(char *key_mgmt);
说明	设置加密方式。 “WPA-PSK”表示开启加密功能，其他则关闭加密功能。
示例	hgic_raw_set_key_mgmt("WPA-PSK"); //开启加密 hgic_raw_set_key_mgmt("NONE"); //关闭加密

##### 3.1.3 set\_wpa\_psk

API	int hgic_raw_set_wpa_psk(char *psk);
说明	设置加密密钥，psk 必须是 64 个 hex 字符。
示例	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);
说明	设置频点范围，freq_start 为起始频率，freq_end 为结束频率，bss_bw 为频率步长。频率值单位为 100kHz。

示例	hgic_raw_set_freq_range(7600,7840,8); // 设置频率范围为 760M-784M, bss 带宽为 8M。
----	---

### 3.1.5 set\_bss\_bw

API	int hgic_raw_set_bss_bw(char bss_bw);
说明	设置 BSS 带宽, bss_bw 有效值为 1/2/4/8M。
示例	hgic_raw_set_bss_bw(8);

### 3.1.6 set\_acs

API	int hgic_raw_set_acs(char acs, char acs_tmo);
说明	设置频点背景扫描, 会对每个信道底噪进行扫描, 选择最干净的信道进行通信。 acs: 1:开启扫描, 0:关闭扫描。 acs_tmo: 信道扫描时间, 单位为 ms。
示例	hgic_raw_set_acs(1,10);

### 3.1.7 set\_mac

API	int hgic_raw_set_mac(char *mac)
说明	设置设备 mac 地址。
示例	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);
说明	设置自定义频点列表, 可用于设置非连续的频点列表, 最多支持 16 个自定义频点。 chan_list: 自定义频点数组, 频率单位为 100kHz

	count: 频点数组的个数。
示例	unsigned short list[]={7600,7680,7740}; hgic_raw_set_chan_list(list,3);//设置 3 个自定义频点，分别是 760M，768M，774M。

### 3.1.9 set\_mode

API	int hgic_raw_set_mode(char *mode);
说明	设置 AH 工作模式。 “ap”: AP 模式 “sta”: STA 模式
示例	hgic_raw_set_mode(“ap");//设置为 AP 模式。

### 3.1.10 set\_pairing

API	int hgic_raw_set_pairing(char enable)
说明	Enable=1:开始配对; Enable=0:停止配对;
示例	hgic_raw_set_pairing(1);

### 3.1.11 set\_pair\_autostop

API	int hgic_raw_set_pair_autostop(char enable)
说明	设置 AH 模块在配对成功后是否自动停止配对。 Enable=1:启用配对自动停止
示例	hgic_raw_set_pair_autostop(1)

## 3.2 组网高级功能 API

### 3.2.1 set\_txpower

API	hgic_raw_set_txpower(int power)
说明	power: tx 发送功率; 范围: 1~20
示例	hgic_raw_set_txpower(15);

### 3.2.2 set\_tx\_mcs

API	int hgic_raw_set_tx_mcs(char tx_mcs);
说明	设置 TX MCS, tx_mcs 有效值为 0~7, 其他为 auto 模式。
示例	hgic_raw_set_tx_mcs(255); //tx mcs auto 模式

### 3.2.3 set\_bss\_max\_idle

API	hgic_raw_set_bss_max_idle(int bss_max_idle)
说明	设置 BSS max idle 时间, 单位 S。 sta 在 max idle 时间必须向 AP 发送 1 个包以保持与 AP 端连接。 AP 在超过 max idle 时间未收到 sta 信息, 则认为 sta 掉线。 注意该参数会影响 sta 的 sleep 功耗, 设置越小功耗越大, 默认 300 秒对应 200uA@DTIM10。
示例	hgic_raw_set_bss_max_idle(300);

### 3.2.4 set\_channel

API	int hgic_raw_set_channel(int channel);
说明	设置信道, channel 值为频道号, 数值从 1 开始。
示例	hgic_raw_set_channel(1); //设置使用第 1 个 channel

### 3.2.5 unpair

API	int hgic_raw_unpair(char *mac)
说明	解除与指定 mac 的模块之间的配对。在 AP/STA 端均可执行该命令。
示例	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)
说明	<p>在配对的过程中模块会自动生成 STA 信息，形成 STA 列表。如果模块有挂 flash，STA 列表会保存到 flash 中，模块重启自动加载 STA 列表；如果模块没有挂 flash，则 STA 列表无法保存，模块重启后丢失。</p> <p>对于不带 flash 的 AP，上电后可以用 set paired_stas 来重新设置配对列表。设置了 paired_stas, 同时再设置 conn_paironly=1，产生的效果是：只允许设置的这些 STA 连接 AP，其它 STA 即使有正确的 SSID 和密码也无法连接，可以实现 AP 限制 STA 的连接，效果类似白名单。</p> <p>该命令最多可以设置的 STA 数量，受限于固件最多支持的 STA 数量；</p>
示例	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)
说明	<p>设置模块自动跳频功能的开启或关闭（默认开启）。</p> <p>自动跳频功能指是 AP 在运行过程中检测到当前信道出现干扰，会自动跳频到另 1 个相对干净的信道。AP 跳频时会通知 STA 一起切换频点，但是对处于 sleep 状态的 STA 无法通知切</p>

	频。AP 跳频到另 1 个频点后，处于 sleep 的 sta 会检测 AP 超时，会重启再次扫描连接 AP，连接成功后会再次进入 sleep。
示例	hgic_raw_set_auto_chswitch(1)

### 3.2.8 set\_primary\_chan

API	int hgic_raw_set_primary_chan(char primary_chan)
说明	设置 pri_chan,默认是 3. 此参数无效了
示例	hgic_raw_set_primary_chan(1)

### 3.2.9 set\_mac\_filter\_en

API	int hgic_raw_set_mac_filter_en(char enable)
说明	开启 mac 地址过滤
示例	hgic_raw_set_mac_filter_en(1)

### 3.2.10 set\_supper\_pwr

API	int hgic_raw_set_supper_pwr(char enable)
说明	设置 AH 模块是否开启 super power 功能。 开启该功能后，在远距离通信时 AH 模块会加大发射功率，最大到 25dbm。该功能在正常模式下默认开启，在 AH 的测试模式下默认关闭。
示例	hgic_raw_set_supper_pwr(1)

### 3.2.11 set\_acktmo

API	int hgic_raw_set_acktmo(int acktmo)
说明	设置增加 AH 模块 WiFi 协议参数 ack timeout 值,单位为微秒,默认为 0。只有在进行超过 1km 通信时才需要设置该参数。计



	算公式为 $10 * (\text{距离公里数} - 1)$ ，例如 2km 设置： acktmo=10
示例	hgic_raw_set_acktmo(10)

### 3.2.12 set\_beacon\_int

API	int hgic_raw_set_beacon_int(int beacon_int)
说明	设置 beacon 包的周期，单位为微秒，默认值为 500ms。
示例	hgic_raw_set_beacon_int(100)

### 3.2.13 set\_heartbeat\_int

API	int hgic_raw_set_heartbeat_int(int heartbeat_int)
说明	ap 端设置 sta 心跳超时时间，单位为毫秒，默认值为 500ms。 如果连续 7 次心跳包未收到，则认为连接断开；
示例	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)
说明	在 ap 端设置，在“counter”秒后，切到另一个信道去； chan_index: 信道序号 counter: 等待频点切换的时间
示例	hgic_raw_set_ap_chan_switch(1,10);

### 3.2.15 set\_agg\_cnt

API	int hgic_raw_set_agg_cnt(char agg_cnt)
说明	设置最大聚合的包个数，默认值为 16。
示例	hgic_raw_set_agg_cnt(10)

### 3.2.16 set\_ap\_hide

API	int hgic_raw_set_ap_hide(char en)
说明	设置 AP 隐藏功能，只在 AP 模式下有效。设置 AP 隐藏后，sta 将无法通过扫描发现 AP。
示例	hgic_raw_set_ap_hide(1)

### 3.2.17 set\_max\_txcnt

API	int hgic_raw_set_max_txcnt(char txcnt)
说明	设置 WiFi 帧的最大重传次数；0 无效，N 表示最多总共发 N 次，默认 7 次
示例	hgic_raw_set_max_txcnt(7)

### 3.2.18 set\_dup\_filter\_en

API	int hgic_raw_set_dup_filter_en(char en)
说明	过滤收到的重传的包
示例	hgic_raw_set_dup_filter_en(1)

### 3.2.19 set\_dis\_1v1\_m2u

API	int hgic_raw_set_dis_1v1_m2u(char dis)
说明	设置 1 对 1 时，是否 disable 多播转单播功能
示例	hgic_raw_set_dis_1v1_m2u(1)

### 3.2.20 set\_kick\_assoc

API	int hgic_raw_set_kick_assoc(char en)
-----	--------------------------------------

说明	主动 kick 一次连接操作
示例	hgic_raw_set_kick_assoc(1)

### 3.2.21 set\_connect\_paironly

API	int hgic_raw_set_connect_paironly(char en)
说明	<p>设置 conn_paironly 之后，AP 将只允许在配对列表中的 sta 连接。不在配对列表中的 STA，即使设置了正确的 SSID 和密码，也无法连接，实现 AP 限制 STA 的连接，效果类似白名单。</p> <p>conn_paironly=1：只允许在配对列表中的 sta 建立连接。</p> <p>conn_paironly=0：允许所有 STA 发起连接，只需要正确的 SSID 和密码就可以成功连接。参数默认值是 0。</p>
示例	hgic_raw_set_connect_paironly(1)

### 3.2.22 set\_disassoc\_sta

API	int hgic_raw_disassoc_sta(char *addr)
说明	断开指定 sta 的连接。AP 可以使用该命令主动断开 STA。但是 STA 在正常模式下，断开连接后会自动重连。
示例	<pre>char addr[] = {0xf6,0xde,0x09,0x6e,0x5a,0x50}; hgic_raw_disassoc_sta(addr)</pre>

### 3.2.23 set\_sta\_freqinfo

API	int hgic_raw_set_sta_freqinfo(char *addr, struct hgic_freqinfo* freqinfo)
说明	设置配对设备的频点、带宽等信息
示例	<pre>char addr[] = {0xf6,0xde,0x09,0x6e,0x5a,0x50}; hgic_raw_set_sta_freqinfo(addr, &amp;hgic_freqinfo)</pre>

### 3.2.24 set\_cust\_isolation

API	int hgic_raw_set_cust_isolation(char en)
说明	设置不同客户 ID 的芯片，是否可以相互连接 默认可以连接
示例	hgic_raw_set_cust_isolation(1)

## 3.3 状态查询 API

### 3.3.1 get\_fwinfo

API	int hgic_raw_get_fwinfo(void)
说明	获取模块固件信息
示例	hgic_raw_get_fwinfo()

### 3.3.2 get\_connect\_state

API	int hgic_raw_get_connect_state(void)
说明	查看连接状态。
示例	hgic_raw_get_connect_state()

### 3.3.3 get\_mode

API	int hgic_raw_get_mode(void)
说明	获取模块角色
示例	hgic_raw_get_mode()

### 3.3.4 get\_sta\_list

API	int hgic_raw_get_sta_list(void)
说明	查看当前已连接的 sta 信息: aid, mac 地址, ps_mode(模块 sleep

	模式), rssi, evm, tx_snr(对方设备统计的 rssi - bgrssi, 空中反馈过来), rx_snr(本设备统计的 rssi - bgrssi) 在 AP 端执行该命令, 可获取已连接的 sta 信息。 在 STA 端执行该命令, 可以查看 sta 当前连接的 AP 的信息。
示例	int hgic_raw_get_sta_list(void)

### 3.3.5 get\_bgrssi

API	int hgic_raw_get_bgrssi(char chan_index)
说明	获取指定信道的背景噪声, 数值从 1 开始
示例	hgic_raw_get_bgrssi(2)

### 3.3.6 scan

API	int hgic_raw_scan(void)
说明	在 STA 模式执行该命令, 用于扫描周围 AP 信息。
示例	hgic_raw_scan()

### 3.3.7 get\_scan\_list

API	int hgic_raw_get_scan_list(void)
说明	获取模块执行 scan 命令后, 可以通过这个命令获取扫描的 AP 列表
示例	hgic_raw_get_scan_list()

### 3.3.8 get\_temperature

API	int hgic_raw_get_temperature(void)
说明	获取模块温度信息
示例	hgic_raw_get_temperature()

### 3.3.9 get\_ssid

API	int hgic_raw_get_ssid(void)
说明	获取模块 ssid
示例	hgic_raw_get_ssid()

### 3.3.10 get\_bssid

API	int hgic_raw_get_bssid(char *bssid)
说明	STA 模式下获取已连接的 AP MAC，同时返回 STA 的 AID
示例	aid = hgic_raw_get_bssid(ap_mac)

### 3.3.11 get\_wpa\_psk

API	int hgic_raw_get_wpa_psk(void)
说明	获取模块加密 key
示例	hgic_raw_get_wpa_psk()

### 3.3.12 get\_sta\_count

API	int hgic_raw_get_sta_count(void)
说明	获取模块的 sta 连接数目
示例	hgic_raw_get_sta_count()

### 3.3.13 get\_txpower

API	int hgic_raw_get_txpower(void)
说明	获取模块 tx power 大小
示例	hgic_raw_get_txpower()

### 3.3.14 get\_aggcnt

API	int hgic_raw_get_aggcnt(void)
说明	获取模块包聚合数的最大值
示例	hgic_raw_get_aggcnt()

### 3.3.15 get\_freq\_range

API	int hgic_raw_get_freq_range(void)
说明	获取模块频率范围
示例	hgic_raw_get_freq_range()

### 3.3.16 get\_chan\_list

API	int hgic_raw_get_chan_list(void)
说明	获取模块频点列表
示例	hgic_raw_get_chan_list()

### 3.3.17 get\_bss\_bw

API	int hgic_raw_get_bss_bw(void)
说明	获取模块带宽
示例	hgic_raw_get_bss_bw()

### 3.3.18 get\_key\_mgmt

API	int hgic_raw_get_key_mgmt(void)
说明	获取模块加密方式
示例	hgic_raw_get_key_mgmt()

### 3.3.19 get\_module\_type

API	int hgic_raw_get_module_type(void)
说明	获取模块的模组类型
示例	hgic_raw_get_module_type()

### 3.3.20 get\_dhcpc\_result

API	int hgic_raw_get_dhcpc_result(void)
说明	获取模块 dhcpc 结果
示例	hgic_raw_get_dhcpc_result()

### 3.3.21 get\_disassoc\_reason

API	int hgic_raw_get_disassoc_reason(void)
说明	获取设备的断连信息
示例	hgic_raw_get_disassoc_reason()

### 3.3.22 get\_rtc

API	int hgic_raw_get_rtc(void)
说明	获取模块的 rtc 时钟
示例	hgic_raw_get_rtc()

### 3.3.23 get\_center\_freq

API	int hgic_raw_get_center_freq(void)
说明	获取模块当前信道的中心频点
示例	hgic_raw_get_center_freq()



### 3.3.24 get\_wakeup\_reason

API	int hgic_raw_get_wakeup_reason(void)
说明	获取模块唤醒原因
示例	hgic_raw_get_wakeup_reason()

## 3.4 低功耗相关 API

### 3.4.1 sleep

API	int hgic_raw_sleep(char sleep);
说明	设置休眠状态。sleep 选择是否使能。
示例	hgic_raw_sleep(1); //进入 sleep 模式。

### 3.4.2 set\_wkio\_mode

API	int hgic_raw_set_wkio_mode(char wkio_mode)
说明	设置 AH wake up Host IO(IOB0)的工作模式。默认脉冲模式。 1: 电平模式, WiFi 正常运行时保持高电平, sleep 时为低电平。 0: 脉冲模式, WiFi 模块唤醒主控时, 拉高 2ms 的脉冲信号。
示例	hgic_raw_set_wkio_mode(1);

### 3.4.3 set\_dtim\_period

API	hgic_raw_set_dtim_period(int dtim_period)
说明	设置 WiFi 模块 sleep 模式下的 dtim 周期, 单位为毫秒, 为 beacon 包周期的倍数。WiFi 会在指定的 dtim 周期定时唤醒检查与 AP 的连接, 并接收 AP 唤醒。 DTIM10 的设置值为 1000(默认值), DTIM20 的设置值为 2000, 依次类推。 该参数会影响 SLEEP 功耗和唤醒时间。DTIM 值大, sleep 功

	耗变低，唤醒时间变长。DTIM 值小，SLEEP 功耗变大，唤醒时间变短。 如果希望用小于 1000 的值，即保活周期小于 1S，除了设置 dtim_period，还需要合理设置 beacon_int 才行。
示例	hgic_raw_set_dtim_period(2000);

### 3.4.4 set\_ps\_mode

API	int hgic_raw_set_ps_mode(char ps_mode)
说明	<p>设置 WiFi 模块<b>连接状态下</b> sleep 模式：</p> <p>0：未设置 sleep 模式，效果与模式 3 一样。</p> <p>1：模块进入 sleep 时与服务器之间保活（模块自己与服务器保活）。</p> <p>2：模块进入 sleep 时与服务器之间保活（AP 代替模块与服务器保活，功耗最低）。</p> <p>3：模块进入 sleep 时只与 AP 之间保持连接，任意单播包可以唤醒模块。</p> <p>4：模块进入 sleep 只与 AP 保活，只能通过 AP 的接口 /proc/hgic/wakeup 唤醒。</p> <p>需要服务器保活功能时建议 ps_mode=2.</p> <p>如果在某些情况下(例如性能测试时)不希望设备进入休眠，可以将 ps_mode 设置为 0。</p> <p>6：无视连接状态直接进入休眠，只能 IO 唤醒</p>
示例	hgic_raw_set_ps_mode(0)

### 3.4.5 set\_ps\_connect

API	hgic_raw_set_ps_connect(char period, char roundup)
说明	<p>设置 ps connect 的 sleep 间隔,和最大递增次数。 STA 的 WiFi 模块在休眠状态下断线后，将会唤醒重新连接 AP。如果连接失败 WiFi 模块将会进入 PS Connect 模式：循环的 sleep/唤醒/重连。中间 Sleep 是为了防止一直重连功耗太大。</p> <p>period:60;</p>

	roundup:3; 第 1 次连接失败 sleep 1 分钟,第 2 次连接失败 sleep 2 分钟,第 3 次连接失败 sleep 3 分钟。sleep 时间递增 3 次后, 回绕到第 1 次的间隔, 依次规律循环。
示例	hgic_raw_set_ps_connect(60,3)

### 3.4.6 set\_wait\_psmode

API	int hgic_raw_set_wait_psmode(char mode)
说明	设置非连接状态下的休眠模式, ap 断电的情况下, sta 进入定时休眠, 在 ap 上电后 sta 能快速唤醒连接。默认 0 是之前的 ps connect, 1 是 standby 模式
示例	hgic_raw_set_wait_psmode(1)

### 3.4.7 set\_standby

API	int hgic_raw_set_standby(char channel, int period_ms)
说明	设置 standby 模式下约定的频点和时间 (时间单位修改为 ms)
示例	hgic_raw_set_standby(2, 1000)

### 3.4.8 set\_aplost\_time

API	int hgic_raw_set_aplost_time(int aplost_time)
说明	设置模块休眠保活时检测 AP 丢失的时间 (单位 S)。在指定的时间内未接收到 AP 的 beacon, 则认为 AP 丢失。默认值为 10 秒。
示例	hgic_raw_set_aplost_time(10)

### 3.4.9 set\_reassoc\_wkhost

API	int hgic_raw_set_reassoc_wkhost(char enable)
-----	--

说明	模块在休眠状态下检测 AP 异常后，会重启扫描连接 AP。模块默认不会通知主控发生了重连 AP 的情况。设置 reassoc_wkhost=1，发生重连 AP 后，模块通知主控。 此接口目前废弃不用了，需要设置唤醒 host，可以用 set_wkhost_reasons。
示例	hgic_raw_set_reassoc_wkhost(1)

### 3.4.10 set\_wakeup\_io

API	int hgic_raw_set_wakeup_io(char wakeup_io, char edge)
说明	设置模块的休眠 host 唤醒 AH 模块的 IO 和 IO 触发沿：0:上升沿（正脉冲），1:下降沿（负脉冲）。不设置这个命令的时候，默认用 mclr 唤醒 AH 模块。 IO 对应的序号为：IOA0~31：0~31，IOB0~7：32~39，mclr：51。 示例：设置唤醒 IO 为 IOA1，上升沿唤醒。
示例	hgic_raw_set_wakeup_io(1,0)

### 3.4.11 set\_autosleep\_time

API	int hgic_raw_set_autosleep_time(char autosleep_time)
说明	设置模块 auto sleep 时间，单位为秒，默认为 10 秒，最大值为 65 秒。 auto sleep 是指模块开机后在指定时间内未接收到主控的命令，则认为主控未开机，自动进入 sleep。 示例： 设置 auto sleep 时间为 20 秒。
示例	hgic_raw_set_autosleep_time(20)

### 3.4.12 set\_dcdc13

API	int hgic_raw_set_dcdc13(char enable)
说明	设置 AH 模块是使用外部 1.3V DCDC 供电，还是内部 LDO 供

	<p>电。</p> <p>使用外部 1.3v DCDC 供电，可使模组的整体功耗降低约 30%。</p> <p>0:使用内部 LDO</p> <p>1:使用外部 DCDC（硬件电路必须有 1.3V DCDC）</p> <p>2:sleep 时使用外部 DCDC（硬件电路必须有 1.3V DCDC），如果测试发现外部 DCDC 影响到 RF 性能的时候，可设置为模式 2，正常工作时使用内部 LDO，休眠时仍然能利用 DCDC 节约功耗。</p> <p>该参数必须根据实际硬件电路进行设置，否则如果设置成非 0 值，外部没有 dcdc，可能会出现无法开机的情况；如果设置成 0，但是外部又有 dcdc，可能会省不了电。</p>
示例	hgic_raw_set_dcdc13(1)

### 3.4.13 set\_ap\_psmode\_en

API	int hgic_raw_set_ap_psmode_en(char enable)
说明	设置 AP 低功耗使能，需要使用 AP 低功耗时，需要对 AP 和 STA 双方开启该功能。
示例	hgic_raw_set_ap_psmode_en(1)

### 3.4.14 set\_pa\_pwrctl\_dis

API	int hgic_raw_set_pa_pwrctl_dis(char disable)
说明	<p>设置模块休眠时的 PA 供电控制逻辑开关。默认开启 PA 供电控制逻辑。</p> <p>0:关闭供电控制逻辑，PA 常供电；</p> <p>1:开启供电控制逻辑，休眠时 PA 不供电，需要配合硬件使用，即用了 IOA30 控制 RF 在休眠时掉电。如果硬件没有预留这个控制电路，软件开启也无效。</p> <p>实际这个参数用默认值就好。</p>
示例	hgic_raw_set_pa_pwrctl_dis(1)

### 3.4.15 set\_dis\_psconnect

API	int hgic_raw_set_dis_psconnect(char disable)
说明	如果不希望 STA 在未连接 AP 时自动进入休眠，可以关掉 PS Connect 模式。例如在某些测试场景下，希望设备在未连接状态下不进休眠。 由于目前休眠 STA 在断开连接后，不会上报信息给主控，所以建议低功耗设备正常情况下不要关掉 PS Connect 模式，否则掉线了，就会一直回连 AP，如果一直连不上，会导致电池消耗很快。
示例	hgic_raw_set_dis_psconnect(1)

### 3.4.16 set\_wkdata\_save\_en

API	int hgic_raw_set_wkdata_save_en(char enable)
说明	设置模块是否保存服务器发送的唤醒数据。 唤醒数据保存后，在主控开机后可以通过接口读取模块缓存的唤醒数据。模块最多缓存 4 个唤醒数据。
示例	hgic_raw_set_wkdata_save_en(1)

### 3.4.17 get\_wkdata\_buff

API	int hgic_raw_get_wkdata_buff(void)
说明	读取模块缓存的唤醒数据。
示例	int hgic_raw_get_wkdata_buff(void)

### 3.4.18 set\_wkhost\_reasons

API	int hgic_raw_set_wkhost_reasons(char *reasons, int count)
说明	设置哪些唤醒原因需要唤醒主控。 STA 唤醒原因： 1: 非连接状态下的 Timer 唤醒(此原因一般不用设置唤醒主控)

	<p>2: 单播 TIM 唤醒, 由 AP 端主动唤醒或其它设备单播唤醒(一般需要设置唤醒主控)</p> <p>3: 广播 TIM 唤醒, 由广播数据唤醒(此原因目前已删除)</p> <p>4: 按键 IO 唤醒 (默认 MCLR 引脚, MCLR 拉 500us 左右) (如果是主控拉的, 可以不用再唤醒主控)</p> <p>5: beacon 丢失唤醒(此原因一般不用设置唤醒主控)</p> <p>6: AP 重启检测唤醒(此原因一般不用设置唤醒主控)</p> <p>7: 心跳超时唤醒(休眠模式 2 下用的)</p> <p>8: 唤醒包唤醒(休眠模式 2 下用的)</p> <p>9: MCLR IO Reset 唤醒(MCLR 在休眠状态拉了超过 2ms 导致的复位, 此原因一般不用设置唤醒主控)</p> <p>10: LVD 低电复位(此原因默认不设置唤醒主控)</p> <p>11: PIR IO 唤醒 (需要特殊硬件电路支持, 否则不用设置)</p> <p>12: AP API 唤醒, AP 端执行 wnb_wakeup_sta 或 hgic_iwpriv_wakeup_sta (一般需要设置唤醒)</p> <p>13: STA 休眠期间, AP 检测到 STA 掉线(此原因一般不用设置唤醒主控)</p> <p>14: STANDBY 状态下连上 AP 时唤醒 (在用了 STANDBY 功能后才要设置)</p>
示例	<pre>char reasons[] = {2,7,8,12,14}; hgic_raw_set_wkhost_reasons(reasons, 5);</pre>

### 3.4.19 wakeup\_sta

API	int hgic_raw_wakeup_sta(char *mac)
说明	唤醒指定 MAC 地址的 sta。
示例	<pre>char addr[] = {11,22,33,44,54,65}; hgic_raw_wakeup_sta(addr)</pre>

### 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)
说明	低功耗模式参数接口文件, 用于设置心跳服务器的 IP 地址/端

	口号/心跳周期/心跳超时时间。 ipaddr: IP 地址 dport:端口号 period:心跳周期, 单位为秒 hb_tmo:心跳超时时间,单位为秒
示例	<pre>unsigned int ipaddr = (192 &lt;&lt; 24)   (168 &lt;&lt; 16)   (0 &lt;&lt; 8)   100; hgic_raw_set_ps_heartbeat(ipaddr, 60001, 60, 300);</pre>

### 3.4.21 set\_heartbeat\_resp

API	int hgic_raw_set_heartbeat_resp(char *heartbeat_resp, int len)
说明	把心跳 response 内容设置给 AH 模块。在 SLEEP 模式下, AH 模块与心跳服务器进行心跳交互时, AH 模块会根据数据包内容识别到心跳 response。 如果发送心跳后未收到 response, AH 模块则会下次唤醒时继续发送心跳给服务器。如果连续 7 次心跳包未收到 response, 则认为与服务器断开连接, 将通过 IO 通知主控网络异常。
示例	<pre>char *heartbeat_resp = "this is heartbeat response data for xxxx device"; hgic_raw_set_heartbeat_resp(heartbeat_resp, sizeof(heartbeat_resp));</pre>

### 3.4.22 set\_ps\_wkdata

API	int hgic_raw_set_ps_wkdata(char *ps_wkdata, int len)
说明	把心跳服务器的唤醒包内容设置给 AH 模块, AH 模块识别到唤醒包后, 则会通过 IO 唤醒主控, 同时 AH 模块自身也被唤醒。
示例	<pre>char *ps_wkdata = "this is wakeup data for xxxxx device"; hgic_raw_set_ps_wkdata(ps_wkdata, sizeof(ps_wkdata));</pre>



3. 4. 23 set\_wkdata\_mask

API	int hgic_raw_set_wkdata_mask(unsigned short offset, unsigned char *mask, int mask_len)
说明	设置唤醒数据的匹配掩码
示例	hgic_raw_set_wkdata_mask(offset, mask, mask_len)

3. 5 组播模式 API

3. 5. 1 join\_group

API	int hgic_raw_join_group(char *group_addr, char aid)
说明	<p>在设置 WiFi 模块的工作模式为 group 之后，可以使用该命令设置 WiFi 模块加入某个组播网络。加入组播网络后，WiFi 模块将只接收该组播网络中的数据。所有的数据通信都以组播地址进行通信。</p> <p>如果设置了工作模式为 group，但是没有加入组播网络，则所有的数据通信都以广播形式进行收发。</p> <p>注意 JOINGROUP 命令，需要在设置了 GROUP 模式后才能设置。</p> <p>参数说明：</p> <p>group_addr: 需要加入的组播网络的地址。</p> <p>aid: 该设备在组播网络中的 AID，AID 有效值：AID 有效值：1~N（N 为固件支持的最大 STA 个数）。网络中各个设备的 AID 应保持唯一。</p> <p>设置有效 AID: WiFi 模块将会定时在组播网络中发送心跳，向其它 WiFi 模块宣示自己的存在。</p> <p>设置无效 AID: WiFi 模块不会发送心跳，不会通知其它 WiFi 模块。如果所有设备都设置 AID 为 0，则可以不受固件支持最大 STA 个数的限制。</p>
示例	<pre>char group_addr[] = {11,22,33,44,54,65}; hgic_raw_join_group(group_addr,3);</pre>

### 3.5.2 set\_mcast\_txparam

API	int hgic_raw_set_mcast_txparam(char dupcnt, char txbw, char txmcs, char clearch)
说明	设置组播通信 TX 参数 dupcnt: 组播数据重发次数 n，默认只发送一次。设置该参数后，每个组播数据都会被重复发送 n 次。 tx_bw: 设置组播数据 tx 带宽： tx_mcs: 设置组播数据 mcs。 clearch: 设置组播数据发送前是否需要清空信道，这个参数目前无效，设置为 0 即可。
示例	int hgic_raw_set_mcast_txparam(3,8,7,0);

## 3.6 中继模式 API

### 3.6.1 set\_r\_ssid

API	int hgic_raw_set_r_ssid(char *r_ssid)
说明	设置中继模式下，连接上一级 AP 的 SSID，最大为 32 个字符。使用要求和 ssid 参数相同。
示例	int hgic_raw_set_r_ssid("ah_relay")

### 3.6.2 set\_r\_psk

API	int hgic_raw_set_r_psk(char *r_psk)
说明	设置中继模式下，连接上一级 AP 的密码。该 key 值为 64 个 hex 字符。使用要求和 wpa_psk 参数相同。
示例	hgic_raw_set_r_psk("1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef");

## 3.7 漫游功能 API

### 3.7.1 set\_roaming

API	int hgic_raw_set_roaming(char enable, char same_freq)
说明	设置漫游功能。 enable=1 :使能漫游功能 same_freq=1 : 设置漫游设备都在同一个频点上。
示例	int hgic_raw_set_roaming(1,1);

## 3.8 双天线功能 API

### 3.8.1 set\_ant\_auto

API	int hgic_raw_set_ant_auto(char en)
说明	使用双天线自动选择模式。
示例	int hgic_raw_set_ant_auto(1)

### 3.8.2 set\_dual\_ant

API	int hgic_raw_set_dual_ant(char en)
说明	是否开启天线切换
示例	int hgic_raw_set_dual_ant(1)

### 3.8.3 set\_ant\_sel

API	int hgic_raw_set_ant_sel(char en)
说明	手动选择天线 0 或 天线 1。 手动选的时候，需要开启天线切换，将自动选择模式关掉
示例	int hgic_raw_set_ant_sel(1)

### 3.8.4 get\_ant\_sel

API	int hgic_raw_get_ant_sel(void)
说明	获取模块的天线选择
示例	hgic_raw_get_ant_sel()

## 3.9 调试功能 API

### 3.9.1 set\_dbginfo

API	int hgic_raw_set_dbginfo(char enable)
说明	<p>开启或关闭固件调试信息输出。</p> <p>该功能开启后，固件的调试信息将会输出到 WiFi 驱动，并打印出来；该功能主要用于抓取调试信息进行问题分析。</p> <p>dbginfo=1，开启固件调试信息输出</p> <p>dbginfo=0，关闭固件调试信息输出</p>
示例	hgic_raw_set_dbginfo(1);

### 3.9.2 set\_sysdbg

API	int hgic_raw_set_sysdbg(char *sysdbg)
说明	<p>开启或关闭固件某些类别的调试信息输出。0 表示关闭相应打印，1 表示打开相应打印。</p> <p>有下面调试信息的类别：</p> <p>“heap,0”：Heap 信息，默认 = 0；</p> <p>“top,0”：各线程占用 CPU 信息，默认=0；</p> <p>“wnb,0”：wnb 层（网络层统计信息），默认=0；</p> <p>“lmac,1”：lmac 层（空口统计信息），默认 = 2；</p> <p>LMAC 统计信息是默认=2 的，如果需要更详细打印，设置=1；</p> <p>WNB 统计信息是默认不开的。</p>
示例	hgic_raw_set_sysdbg(“wnb,1”)

### 3.9.3 set\_assert\_holdup

API	int hgic_raw_set_assert_holdup(char assert_holdup)
说明	设置触发 assert 时，是否 hold 住系统暂停运行
示例	hgic_raw_set_assert_holdup(1)

### 3.9.4 set\_atcmd

API	int hgic_raw_set_atcmd(char *atcmd)
说明	通过主控发送 AT 命令给模组。
示例	hgic_raw_set_atcmd("at+mode=ap")

## 3.10 其他 API

### 3.10.1 open

API	int hgic_raw_open(void);
说明	开启 AH 模块。 AH 模组默认处于 OPEN 状态。如果调用了 close API，则在使用前需要 open 设备。
示例	hgic_raw_open();

### 3.10.2 close

API	int hgic_raw_close(void);
说明	关闭 AH 模块。关闭 AH 模块后，则停止信号发送，断开连接。
示例	hgic_raw_close();

### 3.10.3 save

API	int hgic_raw_save(void)
-----	-------------------------

说明	设置 AH 模块保存参数（AH 模块有配置 nor flash 的情况下）。AH 固件默认会自动保存参数；在修改参数时检测参数发生变化就会自动保存到 flash。如果 auto_save 设置为 0 了，那么需要保持参数的时候就调用这个 save
示例	int hgic_raw_save(void)

### 3.10.4 set\_auto\_save

API	int hgic_raw_set_auto_save(char enable)
说明	设置 AH 模块是否自动保存参数（AH 模块有配置 nor flash 的情况下）。 关闭自动保存功能后，只有 hgic_raw_save() 才会进行保存参数（该命令的参数值立即保存）。
示例	hgic_raw_set_auto_save(1)

### 3.10.5 set\_radio\_onoff

API	int hgic_raw_set_radio_onoff(char on)
说明	用于控制 wifi radio 的打开/关闭。
示例	hgic_raw_set_radio_onoff(1)

### 3.10.6 set\_rtc

API	int hgic_raw_set_rtc(char on)
说明	设置固件的计时功能，毫秒值，从 0 开始算
示例	hgic_raw_set_rtc(1)

### 3.10.7 set\_ether\_type

API	int hgic_raw_set_ether_type(unsigned short ether_type)
说明	设置一种以太网包的类型，收到这种包就过滤掉

示例	hgic_raw_set_ether_type(1)
----	----------------------------

### 3.10.8 set\_load\_def

API	int hgic_raw_set_load_def(char reset_en)
说明	恢复出厂设置。 reset_en=1 : 恢复出厂设置后立即重启，使参数设置生效 reset_en=0 : 不立即重启，下一次重启时设置生效
示例	hgic_raw_set_load_def(1)

### 3.10.9 mcu\_reset

API	int hgic_raw_mcu_reset(void)
说明	重启模块
示例	hgic_raw_mcu_reset();

### 3.10.10 start\_assoc

API	int hgic_raw_start_assoc(void)
说明	切换到开始连接状态
示例	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)
说明	发送用户定义的数据 Dest 是目的 mac 地址; hgic_tx_info *info 是如下定义:

	<pre>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; };</pre> <p>如果不想指定 bw 和 mcs 等参数，info 就用 NULL 表示用默认值；</p> <p>Data 是想发的数据，len 是长度；</p>
示例	<pre>char dest[] = {0x0,0x11,0x22,0x33,0x44,0x55}; hgic_raw_send_customer_mgmt(dest,NULL,"1234567890",10);</pre>



## 4 non-os WiFi 驱动固件下载

non-os WiFi 驱动提供了固件下载功能，当 WiFi 模块没有 flash 时，需要从主控下载固件运行。

固件下载功能需要使用以下 3 个 API：

1. hgic\_bootdl\_parse\_fw：固件信息解析
2. hgic\_bootdl\_request：固件下载状态控制
3. hgic\_bootdl\_send：下载发送固件数据

### 4.1 固件信息解析

解析固件头部信息，包括版本号、chip id、固件长度、是否使能 aes 加密、crc 校验，该 API 解析固件成功后，会填充 struct hgic\_bootdl 信息，用于后续发送固件的状态控制。

API	int hgic_bootdl_parse_fw(unsigned char *fw_data, struct hgic_bootdl *bootdl)
参数	fw_data: 固件内容 bootdl: 固件加载参数
返回值	0: 固件解析成功 -1: 固件解析失败

### 4.2 固件下载状态控制

API	int hgic_bootdl_request(unsigned char cmd, struct hgic_bootdl *bootdl)
参数	cmd: 固件加载命令 HGIC_BOOTDL_CMD_ENTER: 开始固件加载 HGIC_BOOTDL_CMD_WRITE_MEM: 写入固件数据 HGIC_BOOTDL_CMD_RUN: 下载完成，运行固件 bootdl : 固件加载状态控制信息，该参数必须由 hgic_bootdl_parse_fw API 进行填充。
返回值	0: 执行成功

	-1: 执行失败
--	----------

### 4.3 发送固件数据

<b>API</b>	int hgic_bootdl_send(unsigned char *data, unsigned int len)
<b>参数</b>	data: 需要发送的固件数据 len: 需要发送的数据长度
<b>返回值</b>	0: 发送成功 -1: 发送失败

### 4.4 示例代码

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

//从 flash 读取固件头部信息
spi_flash_read(bootdl_buf, FLASH_FW_OFFSET, sizeof(struct hgic_fw_info_hdr));

//判断固件能否被正常解析
if (hgic_bootdl_parse_fw(bootdl_buf, &bootdl) != 0) {
    printf("Bootdl fail\r\n");
    while(1);
}

//发送开始固件加载请求
hgic_bootdl_request(HGIC_BOOTDL_CMD_ENTER, &bootdl);

while (1) {
    //从硬件读数据到 p_buf
    .....
    //解析是否有 bootdl 回复
    resp = hgic_raw_rx(&p_buf, &p_len);

    if (resp == HGIC_RAW_RX_TYPE_BOOTDL_RESP) {

        //如果剩余的固件长度小于一帧的长度，发送长度特殊处理
        if (bootdl.offset + bootdl.frag_size > bootdl.fw_info.fw_size) {
            bootdl.frag_size = bootdl.fw_info.fw_size - bootdl.offset;
        }

        //发送 write 请求
        hgic_bootdl_request(HGIC_BOOTDL_CMD_WRITE_MEM, &bootdl);

        //从 flash 读取固件
```

```

        spi_flash_read(bootdl_buf, FLASH_FW_OFFEST+bootdl.fw_info.hdr_len+
bootdl.offset, bootdl.frag_size);

        //发送偏移量增加
        bootdl.offset += bootdl.frag_size;

        //发送固件数据
        hgic_bootdl_send(bootdl_buf, bootdl.frag_size);

    }
    if (bootdl.offset >= bootdl.fw_info.fw_size) {
        break; //如果已经发送完成，退出循环
    }
}
//固件加载完成后，运行固件
hgic_bootdl_request(HGIC_BOOTDL_CMD_RUN, &bootdl);

```

## 5 non-os WiFi 驱动固件升级

non-os WiFi 驱动提供了固件升级功能，用于更新 WiFi 模块 flash 中的固件。  
固件升级功能需要使用以下 2 个函数：

1. hgic\_ota\_parse\_fw：固件信息解析
2. hgic\_ota\_send\_packet：固件数据发送

### 5.1 固件信息解析

API	int hgic_ota_parse_fw(unsigned char *fw_data, struct hgic_ota *ota)
说明	fw_data: 固件内容 ota: 固件升级参数 解析固件头部信息，包括版本号、chip id、固件长度、是否使能 aes 加密、crc 校验，该 API 解析固件成功后，会填充 struct hgic_ota 信息，用于后续发送固件的状态控制。
返回值	0: 固件解析成功 -1: 固件解析失败

### 5.2 固件数据发送

API	int hgic_ota_send_packet(struct hgic_ota *ota, unsigned char *data, unsigned int len)
参数	ota: 固件升级状态控制信息，该参数必须由 hgic_ota_parse_fw API 进行填充。 data: 需要发送的固件数据 len: 需要发送的数据长度
返回值	0: 发送成功 -1: 发送失败
示例	hgic_ota_send_packet(&ota, ota_buf, ota.frag_size);

### 5.3 示例代码

```
static uint8_t ota_buf[TEST_FW_FRAG_SIZE + 28]; //28 bytes 为固件头
struct hgic_ota ota;

//从 flash 读取固件头部信息
spi_flash_read(ota_buf, 0, sizeof(struct hgic_fw_info_hdr));

//判断固件头部信息能否被正常解析
if (hgic_ota_parse_fw(ota_buf, &ota) != 0) {
    printf("OTA fail\r\n");
    while(1);
}

/*循环发送固件*/

while (1) {
    //从硬件读数据到 p_buf
    .....
    //解析是否有 OTA 回复
    resp = hgic_raw_rx(&p_buf, &p_len);

    if (resp == HGIC_RAW_RX_TYPE_OTA_RESP) {
        //如果剩余的固件长度小于一帧的长度，发送长度特殊处理
        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;
        }
        //从 flash 读取固件（28 bytes 为固件头）
        spi_flash_read(ota_buf+28, ota.offset, ota.frag_size);

        //发送固件
        hgic_ota_send_packet(&ota, ota_buf, ota.frag_size);

        //发送一包后，偏移量增加
        ota.offset += ota.frag_size;

        if (ota.offset >= ota.fw_info.hdr_len + ota.fw_info.fw_size) {
            break; // 如果偏移量超过固件长度，发送完成
        }
    }
}
```

## 6 non-os WiFi 驱动数据收发

### 6.1 发送数据

non-os WiFi 提供 2 个数据发送 API，用于发送不同类型数据。

#### 6.1.1 hgic\_raw\_send

该 API 用于发送原始数据（非以太网数据）。当主控没有运行 tcp/ip 协议栈时，可以使用该 API 进行数据发送。该 API 将会把数据封装为以太网帧，然后再进行发送。

API	int hgic_raw_send(char *dest, char *data, int len)
参数	dest: 数据的目地地址（应用程序需要维护管理设备的 MAC 地址），如果目的地址为全 FF，则为广播包 data: 需要发送的原始数据 len: 需要发送的数据长度
返回值	0: 发送成功 -1: 发送失败
示例	//发送广播数据 char dest[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff}; hgic_raw_send(dest, xxx_data, xxx_data_len); //发送单播数据 char dev_addr[6]; //dev_addr 必须是某个设备的地址 hgic_raw_send(dev_addr, xxx_data, xxx_data_len);

#### 6.1.2 hgic\_raw\_send\_ether

该 API 用于发送以太网数据，当主控运行了 tcp/ip 协议栈时需要使用该 API。驱动移植时需要对接 tcp/ip 协议栈。

API	int hgic_raw_send_ether(char *data, int len)
参数	data: 需要发送的以太网格式数据 len: 需要发送的数据长度

返回值	0: 发送成功 -1: 发送失败
示例	<pre>//被 LwIP 的 output 函数调用 static err_t netif_output(struct netif *netif, struct pbuf *p){     hgif_raw_send_ether(p-&gt;payload, p-&gt;len);     return ERR_OK; }</pre>

## 6.2 接收数据

non-os WiFi 驱动提供了 1 个接收入口函数：

**HGIC\_RAW\_RX\_TYPE** **hgic\_raw\_rx**(unsigned char \*\*data, unsigned int \*len)

应用程序从底层接口驱动接收到数据后，调用该函数进行数据处理，该接口的返回值会标识出接收的数据类型，同时会修改参数 \*\*data 和 \*len。返回值数据类型的定义如下：

```
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**: 收到无效数据，可以忽略掉
- **HGIC\_RAW\_RX\_TYPE\_DATA**: 收到应用数据，参数 data 和 len 会被修改：**\*\*data 被修改指向有效数据起始位置，\*len 被修改为有效数据的长度。**
- **HGIC\_RAW\_RX\_TYPE\_CMD\_RESP**: 收到 CMD 执行结果，**\*\*data 的内容会被重新修改填充**，命令执行成功则填充 **+RESP:OK**，执行失败则填充 **+RESP:Fail**。可以在 **hgic\_raw\_rx\_cmd\_resp** 函数中添加其它处理代码。
- **HGIC\_RAW\_RX\_TYPE\_BOOTDL\_RESP**: 收到固件下载命令执行结果
- **HGIC\_RAW\_RX\_TYPE\_OTA\_RESP**: 收到 OTA 操作命令的执行结果
- **HGIC\_RAW\_RX\_TYPE\_EVENT**: 收到固件产生的 Event 消息。可以在 **hgic\_raw\_rx\_event** 函数中添加其它处理代码。

### 数据接收示例代码：

```
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 移植说明

non-os WiFi 驱动移植需要完成以下 2 个工作：

- 1. 对接底层接口驱动
- 2. 对接上层应用程序或 tcp/ip 协议栈

## 7.1 对接底层驱动

non-os WiFi 驱动与底层接口驱动只有 2 个交互接口：raw\_send 和 hgic\_raw\_rx。

**1. extern int raw\_send(char \*data, int len);**

数据发送接口，non-os WiFi 驱动将调用该函数进行数据发送。该函数需要在具体平台上实现，调用 spi/uart 驱动发送数据。

**2. HGIC\_RAW\_RX\_TYPE hgic\_raw\_rx(char \*\*data, int \*len)**

数据接收处理函数，当 spi/uart 驱动接收到数据时，请调用该函数进行解析处理。该函数的返回值定义了接收数据的类型：

```
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;
```

应用程序需要关注的是：HGIC\_RAW\_RX\_TYPE\_DATA。该类型表示接收到有效数据。该函数执行后 data 和 len 将会被修改：

- data: 修改后的值指向以太网帧头的位置
- len: 修改后的值为以太网帧头长度 + 有效数据的长度

API	int raw_send(unsigned char *data, unsigned int len)
作用	non-os WiFi 驱动将数据封装 HGIC FRM 格式后，调用该函数发送数据；该函数对接硬件底层发送函数，调用 spi/uart 驱动进行数据发送。
返回值	0: 发送成功 -1: 发送失败
示例	int raw_send(unsigned char *data, unsigned int len)

	<pre>{     return hgic_sdspi_write(0, data, len); }</pre>
--	---

API	HGIC_RAW_RX_TYPE hgic_raw_rx(unsigned char **data, unsigned int *len);
作用	数据接收处理函数，当 spi/uart 驱动接收到数据时，调用该函数进行解析处理。该函数的返回值定义了接收数据的类型
返回值	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 对接上层模块

non-os WiFi 驱动与上层模块交互的接口为数据发送接口：hgic\_raw\_send 和 hgic\_raw\_send\_ether。

API	int hgic_raw_send_ether(char *data, int len)
参数	data: 需要发送的以太网格式数据 len: 需要发送的数据长度
返回值	0: 发送成功 -1: 发送失败
示例	<pre>//被 LwIP 的 output 函数调用 static err_t netif_output(struct netif *netif, struct pbuf *p){     hgic_raw_send_ether(p-&gt;payload, p-&gt;len);     return ERR_OK; }</pre>

API	int hgic_raw_send(char *dest, char *data, int len)
参数	dest: 数据的目地地址（应用程序需要维护管理设备的 MAC 地址），如果目地地址为空，则为广播包 data: 需要发送的原始数据 len: 需要发送的数据长度

返回值	0: 发送成功 -1: 发送失败
示例	<pre>//发送广播数据 char dest[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff}; hgic_raw_send(dest, xxx_data, xxx_data_len);  //发送单播数据 char dev_addr[6]; //dev_addr 必须是某个设备的地址 hgic_raw_send(dev_addr, xxx_data, xxx_data_len);</pre>

### 7.3 对接 SPI 接口

non-os WiFi 驱动提供了 2 种方式对接 SPI 接口：

1. 使用 SPI 接口运行 SDIO 协议
2. 使用常规 SPI Master/Slave 通信模式

两种使用方式都提供了封装代码和示例代码。

#### 7.3.1 SPI 接口运行 SDIO 协议

使用 SPI 接口运行 SDIO 协议时，泰芯 AH 模组做 SDIO slave（工作在 SPI 模式），MCU 做 SPI master。

整个 SPI 通信是采用 SDIO 2.0 标准的协议，协议细节部分可自行查阅《SD Specifications Part 1 Physical Layer Specification Version 2.00》和《SD Specifications Part E1 SDIO Specification Version 2.00》文档。

泰芯 AH 模组的 SDIO slave 支持高速模式，高速模式下时钟可以高达 50MHz。

主机 SPI 接口使用的 IO 是标准 SPI（CLK、CS、MOSI、MISO）和从机通知主机接收数据中断引脚（INTIO），从机 SDIO SPI 模式接口使用的 IO 是标准 SDIO（CLK、DAT3、CMD、DAT0、DAT1），引脚连接如下表所示：

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

SDIO 接口端除 CLK 引脚外，其他 IO 需要接上拉电阻以确保通信稳定，一般情况下，泰芯 AH 模组提供 SDIO 接口已经连接了上拉电阻，因此主机 SPI 驱动中不必开启内部上拉功能。

7.3.1.1 hgic\_sdspi 模块说明

non-os WiFi 驱动提供的 hgic\_sdspi 模块已经实现了 SDIO 协议，支持两种模式：全双工模式、半双工模式。全双工模式使用 **hgic\_sdspi\_v2.c**，半双工模式使用 **hgic\_sdspi.c**。

需要用户自行实现以下几个 API，对接 SPI 硬件驱动。

- **spidrv\_write**: 对接 SPI 硬件驱动，实现数据发送功能。
- **spidrv\_read**: 对接 SPI 硬件驱动，实现数据读取功能。
- **spidrv\_cs** : 对接 SPI 硬件 CS 功能。
- **spidrv\_hw\_crc**: 对接硬件 CRC 功能（如果不支持硬件 CRC，该函数返回 0 即可）。

7.3.1.2 hgic\_sdspi 移植工作

API	void spidrv_write(void *priv, char *data, int len, char dma_flag);
说明	对接 SPI 硬件驱动，执行 write 操作，默认实现 CPU 写功能，可选择实现 dma 功能。
示例	<pre>void spidrv_write(void *priv, unsigned char *data, unsigned int len, char dma_flag){     if (dma_flag) {         HAL_SPI_TransmitReceive_DMA(&amp;hspi2, (uint8_t *)data, (uint8_t *)temp_buf, len);         while (!dma2_done);         dma2_done = 0;     } else {         HAL_SPI_TransmitReceive(&amp;hspi2, (uint8_t *)data, (uint8_t *)temp_buf, len, 0xffff);//对接 hspi2     } }</pre>

API	void spidrv_read(void *priv, char *data, int len, char dma_flag);
说明	对接 SPI 接口驱动，执行 read 操作，默认实现 CPU 读功能，可选择实现 dma 功能。全双工读取 SPI 数据同时发送的数据必须为 0xFF。

示例	<pre> 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(&amp;hspi2, (uint8_t *)temp_buf, (uint8_t *)data, len);         while (!dma2_done);         dma2_done = 0;     } else {         HAL_SPI_TransmitReceive(&amp;hspi2, (uint8_t *)temp_buf, (uint8_t *)data, len, 0xffff);     } } </pre>
----	--

API	void spidrv_cs(void *priv, char enable);
说明	控制 CS IO，实现 SPI CS 功能。
	<pre> 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);     } } </pre>

API	void spidrv_hw_crc(void *priv, char *data, int len, char flag);
说明	<p>SPI 数据 CRC 校验函数。</p> <p>有硬件 CRC 时，可以用来提高数据可靠性，但是会牺牲部分通信效率。</p> <p>无硬件 CRC 时该函数可实现为空函数，返回 0 即可。</p> <p>参数 flag 指示该数据需要 7bit 的命令 CRC 还是 16bit 的数据 CRC。</p>

### 7.3.1.3 hgic\_sdspi 使用说明

non-os WiFi 驱动已经实现了 hgic\_raw 和 hgic\_sdspi 模块的对接，但是需要自行实现初始化和数据接收流程控制。non-os WiFi 驱动提供了示例代码，请参考示例代码进行移植修改。

hgic\_sdspi 模块 API 说明如下：

API	int hgic_sdspi_init(void *priv);
说明	接口初始化函数，使 SDIO 从机进入 SPI 模式，配置 blocksize，配置中断功能，使能高速模式。

API	int hgic_sdspi_detect_alive(void *priv);
说明	接口状态检测函数，检测接口是否正常，不正常时需要及时重新初始化接口。

API	int hgic_sdspi_write(void *priv, char *data, int len);
说明	接口写数据函数，主机向泰芯 AH 模组发送数据，函数返回实际写入数据长度。

API	int hgic_sdspi_read(void *priv, char *buf, int len);
说明	<p>接口读数据函数，主机向泰芯 AH 模组读取数据。当泰芯 AH 模组想要向主机发送数据时，会拉低 SDIO DAT1 线，主机可配置下降沿中断及时接收数据或定时调用此函数尝试接收数，返回非 0 且非-1 值是成功接收数据。</p> <p>在调用函数之前并不清楚此次接收数据的长度，参数传入 buf 需要用户按实际通信内容预留足够空间，并在参数 len 中指示 buf 最大接收长度，函数调用成功返回实际读取长度。</p> <p>泰芯 AH 模组会在产生发送数据信号后 2s 认为发送超时，复位 SDIO 状态机，请及时接收数据以免接口通信不正常。</p>

SDIO 从机在通信异常时会初始化内部状态机，但 SDIO 从机初始化状态机后默认在 SD 模式，主机 SPI 需要重新走初始化流程使 SDIO 从机进入 SPI 模式，因此提供了接口检查函数 **hgic\_sdspi\_detect\_alive**，用户在中需要定时检查接口状态以及时恢复通信正常。

## 7.4 对接 UART 接口

将 UART 作为数据接口进行传输数据，泰芯 AH 模组一般选用 UART0 作为与主机通信接口；调试打印和 AT 命令使用另 1 个 `uart` 接口，细节描述请查阅《泰芯 AH 模组 AT 指令开发指南》。

UART 的默认设置为：

波特率：115200

数据位：8bit

停止位：1bit

奇偶校验：无