Hi1131S V100
软件使用指南

文档版本　01

发布日期　2017-04-17

商标声明

 、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

深圳市海思半导体有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：http://www.hisilicon.com/cn/

客户服务电话：4008302118

客户服务邮箱：support@hisilicon.com

前 言

概述

本文档主要介绍 Hi1131S Wi-Fi 需要使用到的配置、开发指南。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi1131S 芯片	V100

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2017-04-17	01	TR5 版本发布

目 录

插图目录

1 概述

1.1 特性

Hi1131S V100 Wi-Fi 模块(Huawei Liteos)有如下特性：

- 支持 SDIO 接口；
- 支持 SoftAP、STA 模式；
- 支持待机唤醒主机；
- 支持 HiLink；
- 海思 link 功能，详细见：3.1.6 Hisi_link
- 支持远程 keep-alive；

WiFi 相关代码、库和文档以 wifi_project.tar.gz 方式提供。

注：如果需要使用 HiLink 协议或加入 HiLink 生态，请邮件联系：庾征 robert.yu@huawei.com，以便得到授权并开展后续对接。

1.2 目录说明

wifi_project

|—docs ---------- 各种类型 Wi-Fi 的补充文档

|—drv ---------- Wi-Fi 驱动，源码或者库的形式

| ├—sdio_hi1131sv100 ----------海思 Hi1131 SDIO WiFi 驱动目录

|—tools ---------- Wi-Fi 使用到的工具代码

| ├—wpa_supplicant-2.2---------- wpa_supplicant 和 hostapd 代码

|—sample ---------- Wi-Fi 使用样例代码

|----Makefile ---------- 编译脚本

2 配置说明

2.1 编译配置

修改 wifi_project 根目录下的 Makefile，做如下配置：

1. 配置要使用的 Hi1131S V100 WiFi 设备：

   ```
   WIFI_DEVICE ?= sdio_hi1131sv100
   ```

   支持的 WiFi 设备请看兼容性器件列表，不同芯片支持的 WiFi 设备不一样。

2. 配置 Huawei LiteOS 根目录，如：

   ```
   LITEOSTOPDIR ?= /home/user/liteos
   ```

3. 编译步骤为：

   （1） 在 liteos 目录下输入 make;

   （2） 在 wifi_project 目录下输入 make;

   （3） 在 wifi_project 目录下输入

   make sample HISI_WIFI_PLATFORM_HI3518EV200=y

   Make sample 时加的 HISI_WIFI_PLATFORM_HI3518EV200 宏是 3518ev200 平台专有宏，定义在 wifi_project/sample/sdio_hi1131sv100/Makefile 中。例：

   LITEOS_CFLAGS += -DHISI_WIFI_PLATFORM_HI3518EV200

4. 烧录

   （1） 烧录 bin 文件，路径：wifi_project/sample/sample.bin

   使用 Hitool 进行烧录

   （2） 进入 uboot 模式设置启动命令

   setenv bootcmd 'sf probe 0;sf read 0x80008000 0x100000 0x600000; go 0x80008000;';setenv bootdelay 2;sa;

3 Wi-Fi 开发指南

3.1 Hi1131S V100

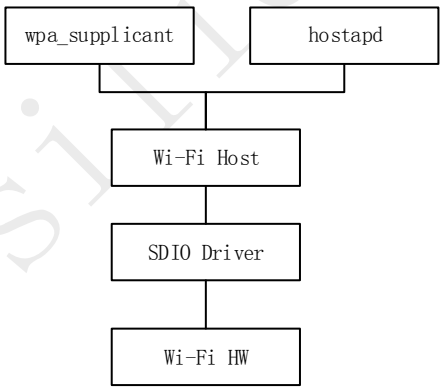
3.1.1 说明

Hi1131S V100 是一款海思 Wi-Fi 芯片，支持 STA、SoftAP、HiLink 以及待机唤醒等功能。

驱动提供开启/关闭 Wi-Fi、睡眠唤醒、HiLink 以及 keep-alive 的 API，头文件为 driver_hisi_lib_api.h。

wpa_supplicant 包含 wpa_supplicant 和 hostapd 两个 task，wpa_supplicant 是负责 STA 模式的配置和连接协议，hostapd 负责 SoftAP 模式的配置和连接协议。STA 或 SoftAP 配置、扫描、连接使用 wpa_supplciant 和 hostapd 的接口来进行。wpa_supplicant 的头文件为 wpa_supplicant.h，hostapd 的头文件为 hostapd_if.h。

图3-1 Hi1131S V100 Wi-Fi 框架

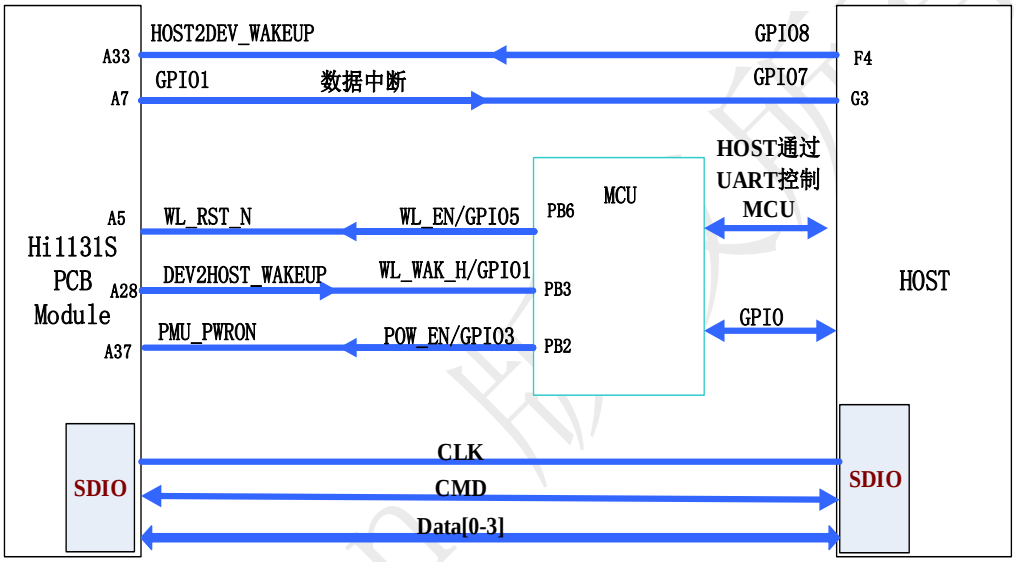


3.1.2 关键业务流程

3.1.2.1 Wi-Fi 板级适配

Hi1131S V100 WiFi 驱动中使用的板级硬件相关参数由应用层传入，针对不同平台，使用此驱动时需要在应用层做相关适配，适配说明详细请查看下面章节,文中的 host 代表主芯片，如 Hi3516C 等，device 代表 wifi 芯片，下图为 Hi1131S 与 Hi3516C 逻辑连接图。

图3-2 Hi1131S 与 Hi3516C 逻辑连接图



WiFi 使用的 GPIO 相关管脚的连接情况以及功能说明如下表；

GPIO 管脚名	说明
WIFI_DATA_INTR_GPIO	与 Hi1131S 芯片 GPIO1 管脚相连，功能：wifi 数据中断。
HOST_WAK_DEV_GPIO	与 Hi1131S 芯片 HOST2DEV_WAKEUP 管脚相连，功能：host 唤醒 device。
WIFI_WAK_FLAG_GPIO	与 MCU 相连，功能：待机唤醒标志。
DEV_WAK_HOST_GPIO	与 MCU 相连，功能：浅睡唤醒中断。

下表为 WiFi 使用的 GPIO 相关的宏命名以及说明，适配时根据连接的实际管脚号定义对应的宏，例如 WIFI_DATA_INTR_GPIO 管脚为 gpio6_0，则需要将 WIFI_DATA_INTR_GPIO_GROUP 定义为 6，WIFI_DATA_INTR_GPIO_OFFSET 定义为 0；

宏命名	说明
WIFI_IRQ	WIFI 驱动使用的 GPIO 中断号
WIFI_DATA_INTR_GPIO_GROUP	WIFI_DATA_INTR_GPIO 管脚组号
WIFI_DATA_INTR_GPIO_OFFSET	WIFI_DATA_INTR_GPIO 管脚偏移
HOST_WAK_DEV_GPIO_GROUP	HOST_WAK_DEV_GPIO 管脚组号
HOST_WAK_DEV_GPIO_OFFSET	HOST_WAK_DEV_GPIO 管脚偏移
WIFI_WAK_FLAG_GPIO_GROUP	WIFI_WAK_FLAG_GPIO 管脚组号
WIFI_WAK_FLAG_GPIO_OFFSET	WIFI_WAK_FLAG_GPIO 管脚偏移
DEV_WAK_HOST_GPIO_GROUP	DEV_WAK_HOST_GPIO 管脚组号
DEV_WAK_HOST_GPIO_OFFSET	DEV_WAK_HOST_GPIO 管脚偏移
REG_MUXCTRL_WIFI_DATA_INTR_GPIO_MAP	WIFI_DATA_INTR_GPIO 管脚复用配置地址
REG_MUXCTRL_HOST_WAK_DEV_GPIO_MAP	HOST_WAK_DEV_GPIO 管脚复用配置地址
REG_MUXCTRL_WIFI_WAK_FLAG_GPIO_MAP	WIFI_WAK_FLAG_GPIO 管脚复用配置地址
REG_MUXCTRL_DEV_WAK_HOST_GPIO_MAP	DEV_WAK_HOST_GPIO 管脚复用配置地址

MCU_UART 相关文件在 mcu_uart 目录。

首先需要确认与 MCU 相连的 UART 的两个管脚复用配置，在 mcu_uart.c 文件中的 hi_uart2_open 接口中：

图3-3 Hi1131S 配置 uart 管脚复用

```
#ifdef HISI_WIFI_PLATFORM_HI3518EV200
        writew(0x04, IO_MUX_REG_BASE + 0x088);
        writew(0x04, IO_MUX_REG_BASE + 0x094);
#else /*default HISI_WIFI_PLATFORM_HI3516CV200*/
        writew(0x3, IO_MUX_REG_BASE + 0x0CC);
        writew(0x3, IO_MUX_REG_BASE + 0x0D0);
#endif
```

根据硬件实际使用的 uart 管脚，查看 mcu 手册做相应的适配。

以下接口在 hi_ext_hal_mcu.c 文件中。

- void HI_HAL_MCUHOST_WiFi_Clr_Flag(void)

  【功能】

  清除待机唤醒标志

  【参数】

  无

  【返回值】

  无

- void HI_HAL_MCUHOST_WiFi_Power_Set(unsigned char val)

  【功能】

  通知 MCU 对 WiFi 芯片上下电

  【参数】

  0：下电；1：上电

  【返回值】

  无

- void HI_HAL_MCUHOST_WiFi_Rst_Set(unsigned char val)

  【功能】

  通知 MCU 对 WiFi 芯片复位

  【参数】

  0：复位；1：解复位

  【返回值】

  无

3.1.2.2 Wi-Fi 低功耗模式

低功耗提供浅睡和强制睡两种机制。

浅睡：在通道无数据交互一段时间后由 host 自动发起，通道休眠，dev 进入低功耗模式，有数据上报或下发时自动唤醒。

强制睡：

host 下电，通道断开，dev 进入低功耗模式，提供 hisi_wlan_suspend 接口。

- void hisi_wlan_suspend(void)、

  【功能】

  启动强制睡眠流程。

  【参数】

  无

  【返回值】

  无

一旦调用此接口，host 业务备份后先和 dev 做一轮交互，双方相互认为对方进入睡眠状态，之后平台走低功耗流程断开通道，host 通过 uart 通知 mcu 给 host 下电。

强制睡眠唤醒：

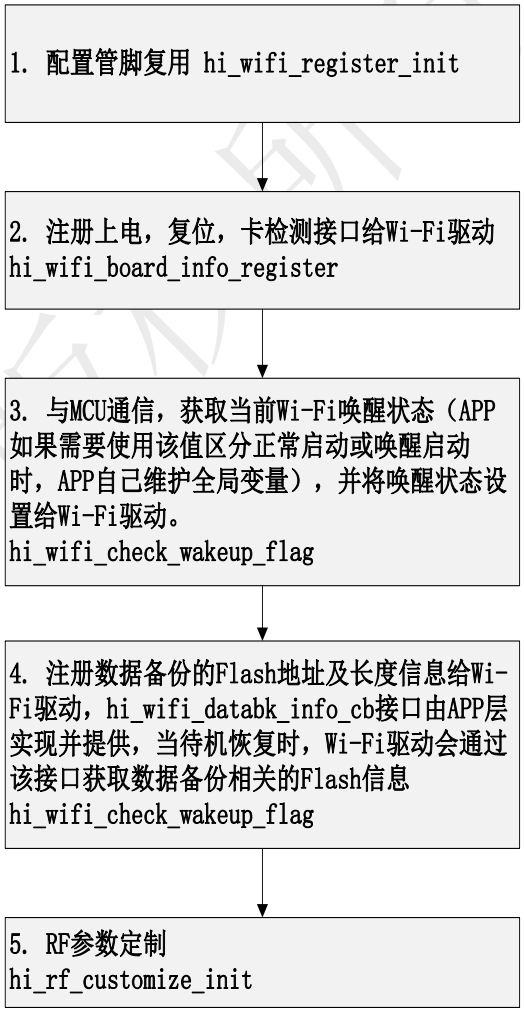
HOST 下电之后通过 DEV 拉高 DEV_WAK_HOST_GPIO 通知 MCU 需要唤醒 HOST，MCU 在接收到此上升沿中断后退出 HALT 模式，拉高 WIFI_WAK_FLAG_GPIO，同时给 HOST 上电，HOST 在上电后读取 WIFI_WAK_FLAG_GPIO 状态，确认唤醒后走唤醒流程，之后详细流程见图 3-7。

3.1.2.3 Wi-Fi 启动配置

鉴于后期可维护性， Hi1131S 将相关管脚约束、上下电硬件操作、与 MCU 通信实现、数据备份位置以及 RF 参数定制等可变部分，上移到应用层，由用户根据实际需求进行更改。相关信息及回调函数，由应用层提供给 Wi-Fi 驱动。

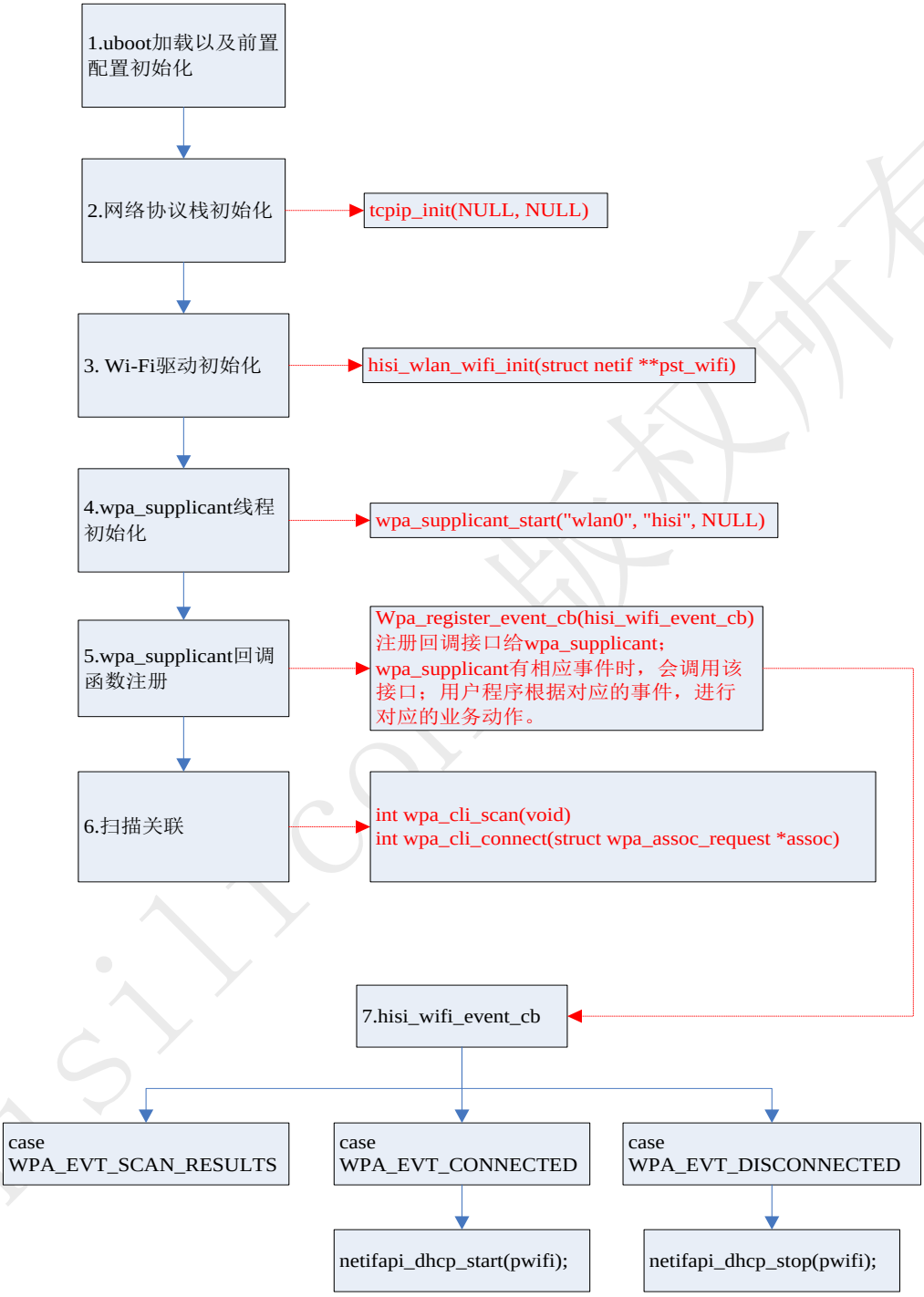
图3-4 Wi-Fi 启动配置图

hi_wifi_pre_proc:
相关逻辑要在
mcu_uart_proc();
msleep(500);之后；原因：第3步需要和mcu进行通信，获取Wi-Fi待机唤醒状态。



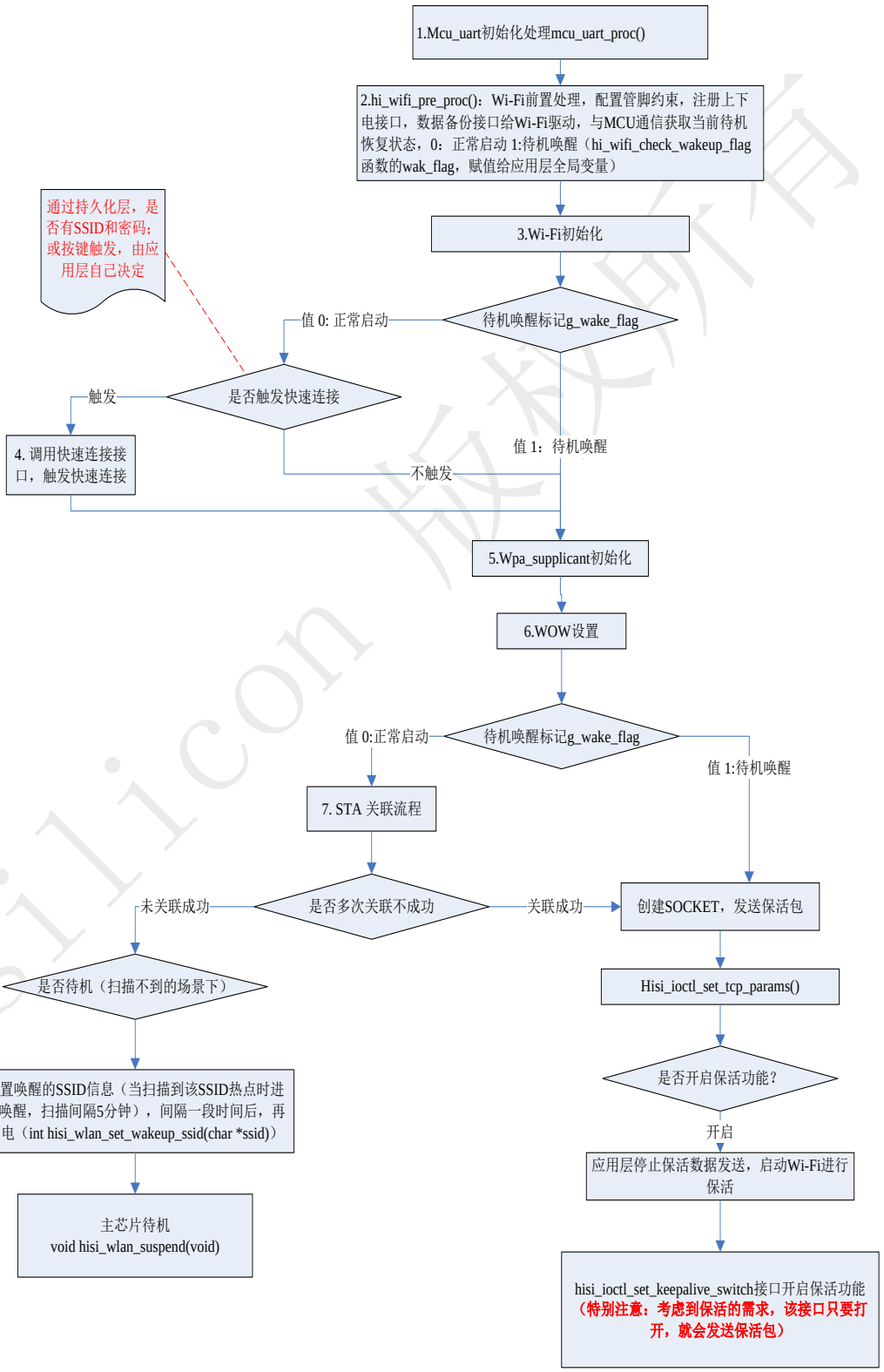
3.1.2.4 Wi-Fi STA 模式关联业务

图3-5 Wi-Fi STA 模式关联业务流程图



3.1.2.5 待机唤醒业务流程及建议约束

图3-6 待机唤醒业务流程图



3.1.2.6 TCP Keepalive 业务流程及建议约束

图3-7 TCP Keepalive 业务流程图

注：相关接口：请参考 3.1.3.1 驱动接口；相关 DEMO 及说明：请参考 3.1.5 TCP Keepalive 功能。

3.1.3 接口说明

3.1.3.1 驱动接口

- int hisi_wlan_wifi_init(struct netif **pst_wifi)

  【功能】

  初始化 Wi-Fi 业务

  【参数】

  pst_wifi：出参，带出 Wi-Fi 初始化后创建的网络接口指针

  【返回值】

  0 表示成功，非 0 表示失败

- int hisi_wlan_wifi_deinit(void)

  【功能】

  去初始化 Wi-Fi 业务

  【参数】

  无

  【返回值】

  0 表示成功，非 0 表示失败

- void hisi_wlan_suspend(void)

  【功能】

  强制睡眠接口。

  【参数】

  无。

  【返回值】

  无。

- void hisi_wlan_set_wow_event(unsigned int ul_event)

  【功能】

  设置强制睡眠功能开关接口。

  【参数】

  - ul_event：入参，事件开关值

  【返回值】

  0 表示成功，非 0 表示失败。

- unsigned int hisi_wlan_add_netpattern(

  ```
  unsigned int       ul_netpattern_index,
  unsigned char    *puc_netpattern_data,
  unsigned int       ul_netpattern_len
  )
  ```

【功能】

添加强制睡眠 netpattern 唤醒报文格式的接口。

【参数】

- ul_netpattern_index：netpattern 的索引, 0~3
- puc_netpattern_data：netpattern 的内容
- ul_netpattern_len： netpattern 的内容长度, 0~64

【返回值】

0 表示成功，非 0 表示失败。

- unsigned int hisi_wlan_del_netpattern(unsigned int ul_netpattern_index)

  【功能】

  删除强制睡眠 netpattern 唤醒报文格式的接口。

  【参数】

  - ul_netpattern_index：netpattern 的索引, 0~3

  【返回值】

  0 表示成功，非 0 表示失败。

- unsigned int hisi_wlan_get_wakeup_reason(unsigned int * pul_wakeup_reason)

  【功能】

  获取强制睡眠唤醒原因的接口。

  【参数】

  - pul_wakeup_reason：出参，输出强制睡眠后的唤醒原因

  【返回值】

  0 表示成功，非 0 表示失败。

- int hisi_wlan_set_wakeup_ssid(char *ssid)

  【功能】

  针对路由器下电或用户家里停电场景，鉴于功耗考虑，在这个情况下，Hi1131S 支持该场景下主芯片下电（下电前设置 SSID，用于唤醒；如果在 hisi_wlan_suspend 接口前设置，建议间隔一小段时间，确保参数下发成功 ），当扫描到该热点时，唤醒主芯片；该接口用户设置扫描唤醒 SSID。

  【参数】

  - ssid：出参，设置扫描唤醒的 SSID，长度最大为 32 个字符。

  【返回值】

  0 表示成功，非 0 表示失败。

- int hisi_wlan_clear_wakeup_ssid(void);

  【功能】

  该接口与 hisi_wlan_set_wakeup_ssid 是一对接口，用于清理扫描 SSID。

  【参数】

  无

  【返回值】

  0 表示成功，非 0 表示失败。

- int hisi_wlan_set_tcp_params(hisi_tcp_params_stru *tcp_params)

  【功能】

  设置 TCP 保活链路的参数信息

  【参数】

  tcp_params：TCP 保活参数的地址

  【返回值】

  0 表示成功，非 0 表示失败

- int hisi_wlan_set_keepalive_switch(unsigned char keepalive_switch, unsigned int keepalive_num)

  【功能】

  开关 TCP 保活功能

  【参数】

  keepalive_switch：0 关闭保活 1 打开保活

  keepalive_num：当下发开启保活功能命令时，同时设置 TCP 保活链路断链多少条需唤醒 host 侧。在下发关闭保活命令时该值应为 0.

  【返回值】

  0 表示成功，非 0 表示失败

- int hisi_wlan_get_lose_tcpid(void)

  【功能】

  TCP 保活断链导致 host 唤醒时，用来获取断链 TCP 链路 ID。当前最大支持 4 条链路。

  【参数】

  无

  【返回值】

  返回值的最低 4 个 bit 位标示 TCP 保活链路断链的状态。

  例如：

  返回值为 1 表示 TCP 链路 ID 为 1 的链路断链

  返回值为 2 表示 TCP 链路 ID 为 2 的链路断链

  返回值为 5 表示 TCP 链路 ID 为 3 和 1 的链路断链

- int hisi_wlan_set_monitor(unsigned char monitor_switch)

  【功能】

  设置 Wi-Fi 芯片进入 monitor 模式，该功能目前只支持 AP 模式下开启

  【参数】

  monitor_switch：0 表示关闭，1 表示打开

  【返回值】

  0 表示成功，非 0 表示失败

- int hisi_wlan_set_channel(hisi_channel_stru *channel_info)

  【功能】

  设置当前 AP 的工作信道，仅限 hilink 适配层使用。

【参数】

channel_info：将要设置的信道信息

【返回值】

0 表示成功，非 0 表示失败

- unsigned int hisi_wlan_register_upload_frame_cb(hisi_upload_frame_cb func)

  【功能】

  monitor 模式下上报组播数据包的回调函数注册接口。

  【参数】

  func：上报组播数据包的处理函数

  【返回值】

  0 表示成功，非 0 表示失败

- int hisi_wlan_ip_notify(unsigned int ip, unsigned int mode)

  【功能】

  通知驱动 IP 地址发生变化，在获取 IP 成功后需要调用该接口通知驱动 IP 获取成功并将 IP 值传给驱动，在去关联时也需要调用该接口通知驱动 IP 的变法。

  【参数】

  ip：当前 IP 的值

  mode：当前是获取 IP 还是释放 IP

  【返回值】

  0 表示成功，非 0 表示失败

- void hisi_cmd_get_rssi(void)

  【功能】

  获取当前的 RSSI 值

  【参数】

  无

  【返回值】

  无

- void hisi_cmd_set_pm_switch(int argc, unsigned char *argv[])

  【功能】

  动态开关低功耗

  【参数】

  0：关闭低功耗

  1：开启低功耗

  【返回值】

  无

- void hisi_cmd_get_country(void)

  【功能】

  获取当前国家码

  【参数】

无

【返回值】

无

- void hisi_cmd_set_country(int argc, unsigned char *argv[])

  【功能】

  设置国家码

  【参数】

  国家码标志，例如中国，参数即 CN

  【返回值】

  无

- void hisi_cmd_rx_info(void)

  【功能】

  常收模式下查看收包数

  【参数】

  无

  【返回值】

  无

- void hisi_cmd_set_always_rx(int argc, unsigned char *argv[])

  【功能】

  设置常收模式

  【参数】

  共需输入 3 个参数

  参数格式为: 常收开关(0|1) 模式 信道

  【返回值】

  无

- void hisi_cmd_set_always_tx(int argc, unsigned char *argv[])

  【功能】

  设置常发模式

  【参数】

  共需输入 4 个参数

  参数格式为: 常发开关(0|1) 模式 信道 速率

  【返回值】

  无

3.1.3.2 SoftAP 模式接口

- int hostapd_start(char *ifname, struct hostapd_conf *hconf)

  【功能】

  开启 AP 模式，启动 hostapd 线程。

【参数】

- ifname：网络接口名
- hconf：配置 AP 参数，包括 SSID、信道、加密方式、密码等

【返回值】

0 表示成功，-1 表示失败。

- int hostapd_stop(void)

【功能】

关闭 AP 模式，删除 hostapd 线程。

【参数】

无。

【返回值】

0 表示成功，-1 表示失败。

编程实例：

- AP 模式启动

```
int softap_demo_start()
{
    struct netif *pst_netif = null;
    if(0 != hisi_wlan_wifi_init(&pst_netif))
    {
        printf("fail to start hi1131 wifi\n");
        if(pst_netif != NULL)
            hisi_wifi_deinit();
    }
    /* 创建 AP 启动用配置结构体变量 */
    struct hostapd_conf st_hapd_conf;
    struct netif *pst_lwip_netif = null;
    memset(&st_hapd_conf, 0, sizeof(struct hostapd_conf));
    /* 初始化 AP 配置参数 */
    memcpy(st_hapd_conf.driver, "hisi", 5);
    st_hapd_conf.channel_num = 6;//设置信道数
    memcpy(st_hapd_conf.ssid, “hisi_demo”, 10);
    st_hapd_conf.ignore_broadcast_ssid = 0;//是否隐藏，1 为是，0 为否
    memcpy(st_hapd_conf.ht_capab, "[HT20]", strlen("[HT20]"));//带宽
    st_hapd_conf.authmode = HOSTAPD_SECURITY_WPA2PSK;//加密类型
```

```
    strcpy((char *)st_hapd_conf.key, “12345678”);//密码，open 不需要设置
    st_hapd_conf.wpa_pairwise = WPA_CIPHER_CCMP;//加密算法，仅 wpa wpa2 需要设置

    if (0 != hostapd_start(HISI_NULL, &gst_hapd_conf))
    {
        printf("\nhostapd start failed\n");
        return -1;
    }

    pst_lwip_netif = netif_find("wlan0");
    netif_set_up(pst_lwip_netif);
    dhcps_start(pst_lwip_netif);
    return 0;
}
```

- AP 模式删除

```
void softap_demo_stop()
{
    struct netif *pst_lwip_netif = null;
    if (0 != hostapd_stop())
    {
        printf("\nhostapd stop failed\n");
        return;
    }
    pst_lwip_netif = netif_find("wlan0");
    dhcps_stop(pst_lwip_netif);
    netif_set_down(pst_lwip_netif);
}
```

创建不同认证方式的 AP 时参数不同，各认证方式的参数差异如下：

- OPEN：

  ```
  gst_hapd_conf.authmode = HOSTAPD_SECURITY_OPEN;
  ```

- WPA：

  ```
  gst_hapd_conf.authmode = HOSTAPD_SECURITY_WPAPSK;
  ```

- WPA2：

  gst_hapd_conf.authmode = HOSTAPD_SECURITY_WPA2PSK;

- WPA+WPA2：

  gst_hapd_conf.authmode = HOSTAPD_SECURITY_WPAPSK_WPA2PSK_MIX;

其中在 WPA、WPA2、WPA+WPA2 的认证方式中各加密算法的参数如下：

- TKIP：

  gst_hapd_conf.wpa_pairwise = WPA_CIPHER_TKIP;

- AES：

  gst_hapd_conf.wpa_pairwise = WPA_CIPHER_CCMP;

- TKIP+AES：

  gst_hapd_conf.wpa_pairwise = WPA_CIPHER_TKIP | WPA_CIPHER_CCMP;

3.1.3.3 STA 模式接口

- int wpa_supplicant_start(char *ifname, char *driver, struct wpa_supplicant_conf *conf)

  【功能】

  开启 STA 模式，启动 wpa_supplicant 线程。

  【参数】

  - ifname：网络接口名，如 “wlan0”
  - driver：Wi-Fi 驱动名，如 “hisi”
  - conf：wpa_supplicant 配置参数，无文件系统时传入 NULL

  【返回值】

  0 表示成功，-1 表示失败。

- int wpa_supplicant_stop (void)

  【功能】

  关闭 STA 模式，删除 wpa_supplicant 线程。

  【参数】

  无。

  【返回值】

  0 表示成功，-1 表示失败。

- int wpa_cli_scan(void)

  【功能】

  启动扫描。

  【参数】

  无。

  【返回值】

  0 表示成功，-1 表示失败。

- int wpa_cli_scan_results(struct wpa_ap_info *results, unsigned int *num)

【功能】

返回扫描结果到 result 中。

【参数】

- result：保存扫描结果
- num：最大扫描结果个数

【返回值】

0 表示成功，-1 表示失败。

- int wpa_cli_connect(struct wpa_assoc_request *assoc)

【功能】

关联 AP。

【参数】

- assoc：根据该结构体变量中的 ssid、加密方式、密码等参数，关联相应的 AP

【返回值】

0 表示成功，-1 表示失败。

- int wpa_cli_disconnect(void)

【功能】

去关联。

【参数】

无。

【返回值】

0 表示成功，-1 表示失败。

编程实例：

wifi_init_sample wifi_deinit_sample 参看 STA 模式

- STA 模式下的回调函数

```
void wifi_event_cb(enum wpa_event event)
{
    if(pwifi == 0)
        return ;
    switch(event) {
        case WPA_EVT_SCAN_RESULTS:
            //TODO:
        break;
        case WPA_EVT_CONNECTED:
            netifapi_dhcp_stop(pwifi);
            netifapi_dhcp_start(pwifi);
            //TODO:
        break;
        case WPA_EVT_DISCONNECTED:
            netifapi_dhcp_stop(pwifi);
```

```
            //TODO:
        break;
        default:
        break;
    }
}
```

- STA 模式下关联

```
#include "wpa_supplicant/wpa_supplicant.h"

void wifi_connect_sample()
{
    struct wpa_assoc_request wpa_assoc_req;
    int             l_ret = 0;
    unsigned int ul_ret = 0;
    ul_ret = hisi_wlan_wifi_init(&pwifi);
    if(0 != ul_ret)
    {
        if(pwifi != NULL)
            hisi_wlan_wifi_deinit();
    }
    wpa_register_event_cb(wifi_event_cb);
    l_ret = wpa_supplicant_start("wlan0", "hisi", NULL);
    if (l_ret != 0)
    {
        printf("sta_demo_start fail.\n");
    }
    wpa_cli_scan();//扫描
    //TODO
    memset(&wpa_assoc_req , 0 ,sizeof(struct wpa_assoc_request));
    wpa_assoc_req.hidden_ssid=0;
    strcpy(wpa_assoc_req.ssid, "ap_ssid");
    wpa_assoc_req.auth = WPA_SECURITY_WPA2PSK;
    strcpy(wpa_assoc_req.key, "12345678");
    //wpa_assoc_req.hex_pwd = 1234567890//AP 是 wep 加密方式且 16 进制密码
时，设置
    wpa_connect_interface(&wpa_assoc_req);
    //TODO
}
```

- STA 模式关闭

```
void sta_stop_demo()
{
    int l_ret = 0;
```

```
    struct netif *pst_lwip_netif = null;
    l_ret = wpa_supplicant_stop("wlan0", "hisi", NULL);
    if (l_ret != 0)
    {
        printf("sta_demo_stop fail.\n");
        return;
    }
    pst_lwip_netif = netif_find("wlan0");
    dhcp_stop(pst_lwip_netif);
    netif_set_down(pst_lwip_netif);
}
```

关联事例中展示的是 WPA2 认证方式的连接参数，其他认证方式连接参数如下：

- OPEN：该模式下不需要对 key 成员进行赋值

  ```
  wpa_assoc_req.auth = WPA_SECURITY_OPEN;
  ```

- WEP：

  ```
  wpa_assoc_req.auth = WPA_SECURITY_WEP;
  ```

- WPA：

  ```
  wpa_assoc_req.auth = WPA_SECURITY_WPAPSK;
  ```

- WPA+WPA2：

  ```
  wpa_assoc_req.auth = WPA_SECURITY_WPAPSK_WPA2PSK_MIX;
  ```

3.1.3.4 HiLink 适配层接口

- int hisi_hilink_parse_cb(void* p_frame, unsigned int ul_len)

  【功能】

  处理驱动上报上来的数据包，将收到的数据包如队等待 hilink 库函数处理。

  【参数】

  无。

  【返回值】

  0 表示成功，-1 表示失败。

- int hisi_hilink_adapt_init(void)

  【功能】

  hilink 初始化驱动相关的配置

  【参数】

  无

  【返回值】

  0 表示成功，-1 表示失败。

- int hisi_hilink_change_channel (void)

【功能】

hilink 适配的切信道接口，函数内部实现切信道逻辑

【参数】

无

【返回值】

0 表示成功，-1 表示失败。

- int hisi_hilink_adapt_deinit (void)

【功能】

hilink 去初始化驱动相关的配置

【参数】

无

【返回值】

0 表示成功，-1 表示失败。

- int hisi_hilink_online_notice(hilink_s_result* pst_result);

【功能】

hilink 发送上线通知包

【参数】

无

【返回值】

0 表示成功，-1 表示失败。

3.1.3.5 海思快连适配层接口

- int hisi_hsl_parse_cb(void* p_frame, unsigned int ul_len)

【功能】

处理驱动上报上来的数据包，将收到的数据包如队等待 hisi_link 库函数处理。

【参数】

无。

【返回值】

0 表示成功，-1 表示失败。

- int hisi_hsl_adapt_init(void)

【功能】

hisi_link 初始化驱动相关的配置

【参数】

无

【返回值】

0 表示成功，-1 表示失败。

- int hisi_hsl_change_channel (void)

  【功能】

  hisi_link 适配的切信道接口，函数内部实现切信道逻辑

  【参数】

  无

  【返回值】

  0 表示成功，-1 表示失败。

- int hisi_hsl_adapt_deinit (void)

  【功能】

  hisi_link 去初始化驱动相关的配置

  【参数】

  无

  【返回值】

  0 表示成功，-1 表示失败。

- int hisi_hsl_online_notice(hsl_result_stru *pst_result)

  【功能】

  hisi_link 发送上线通知包

  【参数】

  无

  【返回值】

  0 表示成功，-1 表示失败。

3.1.3.6 异常处理配置接口

- void hisi_wlan_no_fs_config(unsigned long ul_base_addr, unsigned int u_length)

  【功能】

  wlan 无文件系统时异常处理文件保存配置

  【参数】

  ul_base_addr：文件保存 flash 基地址；u_length：文件保存 flash 长度

  【返回值】

  无

3.1.4 待机唤醒功能

对于通过电池供电的产品，需要考虑电池的续航时间，需要支持待机模式。产品包含三块芯片，主芯片、Wi-Fi 芯片、MCU。待机模式下，主芯片会下电，以达到省电的目的，MCU 和 Wi-Fi 芯片会继续工作，使得 Wi-Fi 继续处于连接状态。

待机前会备份主芯片相关的数据，唤醒后再恢复主芯片相关的数据，待机唤醒不会影响 Wi-Fi 连接状态。

3.1.4.1 唤醒条件

待机状态下，可以通过 Wi-Fi 来唤醒。Wi-Fi 唤醒条件：

- 收到 Magic packet 唤醒报文。

  Magic packet 报文为 UDP 广播包，由局域网络内的其他客户端发送；

- Wi-Fi 连接断开。

  STA 模式下，与 AP 的连接断开，包括两种情况：STA keep alive 失败主动去关联、AP 断开 STA 连接；

- 收到指定格式的 netpattern 唤醒报文。

  Netpattern 报文为自定义的 TCP/UDP 格式的单播报文，由局域网络内的其他客户端发送；

- 与远程服务器 tcp keep-alive 失败。
- 针对路由器下电或用户家里停电场景，鉴于功耗考虑，在这个情况下，Hi1131S 支持该场景下下电（下电前设置 SSID，用于唤醒），当扫描到该热点时，唤醒主芯片。

3.1.4.2 配置说明

待机唤醒相关配置：

- 可以灵活配置 Wi-Fi 的唤醒条件，通过 hisi_wlan_set_wow_event 接口来开启或关闭各个唤醒条件。

  相关接口：

  void hisi_wlan_set_event(unsigned int ul_event);

  枚举类型：

  typedef enum

  {

  MAC_WOW_FIELD_ALL_CLEAR = 0, /* Clear all events */

  MAC_WOW_FIELD_MAGIC_PACKET = BIT0, /* Wakeup on Magic Packet */

  MAC_WOW_FIELD_NETPATTERN_TCP = BIT1, /* Wakeup on TCP NetPattern */

  MAC_WOW_FIELD_NETPATTERN_UDP = BIT2, /* Wakeup on UDP NetPattern */

  MAC_WOW_FIELD_DISASSOC = BIT3, /* 去关联/去认证，Wakeup on Disassociation/Deauth */

```
    MAC_WOW_FIELD_AUTH_RX                = BIT4,        /*  对端关联请求，Wakeup on auth */

    MAC_WOW_FIELD_HOST_WAKEUP            = BIT5,        /* Host wakeup */

    MAC_WOW_FIELD_TCP_UDP_KEEP_ALIVE = BIT6,        /* Wakeup on TCP/UDP keep alive timeout */

    MAC_WOW_FIELD_OAM_LOG_WAKEUP         = BIT7,        /* OAM LOG wakeup */

    MAC_WOW_FIELD_SSID_WAKEUP            = BIT8,        /* SSID Scan wakeup */

}mac_wow_field_enum;
```

- 可以配置自定义 netpattern 唤醒报文的内容，目前最多同时支持 4 种唤醒报文格式，每种报文内容的最大长度为 64 字节。

  相关接口：

```
unsigned int hisi_wlan_add_netpattern(
                    unsigned int      ul_netpattern_index,
                    unsigned char    *puc_netpattern_data,
                    unsigned int      ul_netpattern_len
                    );
unsigned int hisi_wlan_del_netpattern(unsigned int ul_netpattern_index);
```

3.1.4.3 唤醒原因码

唤醒后可通过接口 hisi_wlan_get_wakeup_reason 查询唤醒原因码：

- 相关接口：

```
unsigned int hisi_wlan_get_wakeup_reason(unsigned int * pul_wakeup_reason);
```

- 枚举类型：

```
typedef enum
{
    MAC_WOW_WKUP_REASON_TYPE_NULL                = 0,        /* None */

    MAC_WOW_WKUP_REASON_TYPE_MAGIC_PACKET    = 1,        /* Wakeup on Magic Packet */

    MAC_WOW_WKUP_REASON_TYPE_NETPATTERN_TCP  = 2,        /* Wakeup on TCP NetPattern */

    MAC_WOW_WKUP_REASON_TYPE_NETPATTERN_UDP  = 3,        /* Wakeup on UDP NetPattern */
```

```
    MAC_WOW_WKUP_REASON_TYPE_DISASSOC_RX        = 4,        /*
对端去关联/去认证，Wakeup on Disassociation/Deauth */

    MAC_WOW_WKUP_REASON_TYPE_DISASSOC_TX        = 5,        /*
本端主动去关联/去认证，Wakeup on Disassociation/Deauth */

    MAC_WOW_WKUP_REASON_TYPE_AUTH_RX            = 6,        /*
对端关联请求，Wakeup on auth */

    MAC_WOW_WKUP_REASON_TYPE_TCP_UDP_KEEP_ALIVE = 7,
/* Wakeup on TCP/UDP keep alive timeout */

    MAC_WOW_WKUP_REASON_TYPE_HOST_WAKEUP        = 8,

     /* Host wakeup */

    MAC_WOW_WKUP_REASON_TYPE_OAM_LOG            = 9,

    /* OAM LOG wakeup */

    MAC_WOW_WKUP_REASON_TYPE_SSID_SCAN          = 10,

    /* SSID Scan wakeup */

    MAC_WOW_WKUP_REASON_TYPE_BUT

}mac_wow_wakeup_reason_type_enum;
```

3.1.4.4 示例代码

- 睡眠示例代码：

```
void 1131S_Suspend_Sample(void)
{
    unsigned int      ul_netpattern_index = 0;
    unsigned char     auc_pattern[8] = {0x98,0x3B,0x98,0x3B,0x98,0x3B,0x98,0x3B};

    /*  前提条件是 Wi-Fi 初始化  且已处于关联状态  */
    /* wifi_init_sample(); */
    /* wifi_connect_sample(); */

    /*  可选配置:设置  wow event  接口测试，默认配置为  0xDF。
        如需关闭  Magic  报文唤醒功能，则清零  MAC_WOW_FIELD_
        MAGIC_PACKET = BIT0  即设置为  0xDE。如需恢复  Magic
        报文唤醒功能，则重新配置，hisi_wlan_set_wow_event(0xDF); */
    hisi_wlan_set_wow_event(0xDE);
```

```
    /* 可选配置:删除接口测试，删除默认的 netpattern，系统
       默认存储了一条 netpettern，0x98,0x3B,0x16,0xF8,0xF3,
       0x9C， index 为 0 */
    ul_netpattern_index = 0;
    hisi_wlan_del_netpattern(ul_netpattern_index);

    /* 可选配置:添加接口测试，添加新的 netpattern，index 范围
       0~3，这里指定 index 为 3 */
    ul_netpattern_index = 3;
    hisi_wlan_add_netpattern(
        ul_netpattern_index,
        auc_pattern,
        sizeof(auc_pattern)
        );

    /* Host 下电接口 */
    hisi_wifi_suspend();
}
```

- 唤醒原因示例代码：

（首先需要通过 Netpattern 报文等唤醒方式唤醒）

```
void 1131S_Wakeup_Sample(void)
{
    unsigned int ul_wakeup_reason;

    /* 获取唤醒原因码，预期唤醒原因码不是
       MAC_WOW_WKUP_REASON_TYPE_MAGIC_PACKET = 1 */
    hisi_wlan_get_wakeup_reason(&ul_wakeup_reason);
}
```

3.1.5 TCP Keepalive 功能

TCP Keepalive 的功能是保证设备主芯片下电后 Wi-Fi 芯片依然能够保证设备在服务器端处于在线状态。该功能目的是实现设备主芯片下电后用户能够通过远程服务器对设备进行唤醒等操作。

保活过程：

- 保活功能开启前需要创建用于保活的 TCP 链路。
- 用户下电前调用 hisi_ioctl_set_tcp_params 接口将保活的 TCP 参数下发到 Wi-Fi 芯片（具体参数及含义请参考代码）。
- 保活参数下发成功后用户需调用 hisi_ioctl_set_keepalive_switch 接口开启保活功能。保活功能开启成功后 Wi-Fi 芯片开始保活过程。
- 保活参数下发成功后，用户可以调用下电接口进行下电操作。
- 当用户发起唤醒命令或 TCP 链路保活失败个数达到设定上限时设备主芯片会被上电唤醒，唤醒后由用户调用 hisi_ioctl_set_keepalive_switch 接口关闭保活功能。

```
int keepalive_demo_send_tcp_params(int sockfd)
{
    struct tcpip_conn       st_sockt_info;
    hisi_tcp_params_stru st_tcp_params;
    unsigned int               ul_ret;
    int                          l_ret;
    char                        *pc_buf;

    l_ret = lwip_get_conn_info(sockfd, &st_sockt_info);
    if (0 > l_ret)
    {
        HISI_PRINT("%s[%d]:get sockt info fail\n",__func__,__LINE__);
        return -HISI_EFAIL;
    }
    /* 准备参数 */
    memcpy(st_tcp_params.uc_dst_mac,&st_sockt_info.dst_mac,6);
    st_tcp_params.ul_sess_id    = 1;
    st_tcp_params.us_src_port = st_sockt_info.srcport;
    st_tcp_params.us_dst_port = st_sockt_info.dstport;
    st_tcp_params.ul_seq_num    = st_sockt_info.seqnum;
    st_tcp_params.ul_ack_num    = st_sockt_info.acknum;
```

```
        st_tcp_params.us_window      = st_sockt_info.tcpwin;
        st_tcp_params.us_interval_timer            = INTERVAL_TIMER;    //45 秒发一次
        st_tcp_params.us_retry_interval_timer = RETRY_INTERVAL_TIMER;
        st_tcp_params.us_retry_max_count           = RETRY_TIMES;
        pc_buf = (char*)malloc(sizeof(char) * sizeof(KEEPALIVE_BUF));
        memset(pc_buf, 0, sizeof(char) * sizeof(KEEPALIVE_BUF));
        memcpy(pc_buf, KEEPALIVE_BUF, sizeof(char) * sizeof(KEEPALIVE_BUF));
        st_tcp_params.puc_tcp_payload                = pc_buf;
st_tcp_params.ul_payload_len                 = sizeof(KEEPALIVE_BUF)
…
        ul_ret = hisi_wlan_set_tcp_params(&st_tcp_params);
        if (0 != ul_ret)
        {
            HISI_PRINT("%s[%d]:set tcp params fail\n",__func__,__LINE__);
            return -HISI_EFAIL;
        }
        return HISI_SUCC;
}

int keepalive_demo_set_switch(unsigned char keepalive_switch)
{
    unsigned char    uc_index;
    unsigned int     ul_ret;
    if(0 == keepalive_switch)
    {
        ul_ret = hisi_wlan_set_keepalive_switch(HISI_KEEPALIVE_OFF);
        if (HISI_SUCC != ul_ret)
        {
            HISI_PRINT("%s[%d]:set keepalive switch
fail[%d]\n",__func__,__LINE__,ul_ret);
            return -HISI_EFAIL;
        }
    }
    else
```

```
    {
        keepalive_demo_send_tcp_params(sockfd);
        hisi_wlan_set_keepalive_switch(HISI_KEEPALIVE_ON);
    }
    HISI_PRINT("%s[%d]:keepalive_switch=%d\n",__func__,__LINE__,keepalive_switch);
    return HISI_SUCC;
}
```

3.1.6 HiLink

当前门铃、随身拍、运动 DV 等场景，没有 GUI 界面（硬件屏幕），无法通过传统界面输入 Wi-Fi 密码的方式接入 Wi-Fi 网络。故提供一种零配置的方式，使得设备上电后能够快速的连接到网络。

零配置的过程：

- 用户首先调用 HiLink 库函数接口 hilink_link_get_devicessid 获取将要创建 AP 的特殊 SSID。
- 获取到特殊 SSID 后用户调用 hostapd_start 接口创建特殊 SSID 的 AP。
- AP 创建成功后，hostapd 回调函数会调用 hisi_hilink_adapt_init 接口实现开启 Wi-Fi 芯片的 monitor 模式以及注册上报数据包处理函数的的操作。
- 同时 hilink 线程会循环处理数据队列中的数据包，并周期性切换信道。
- 当解码成功或解码超时时用户需调用 hisi_hilink_adapt_deinit 关闭 monitor 功能。
- 当解码成功时会调用 hilink_demo_connect_prepare 将设备模式由 AP 切回 STA。
- 当 STA 创建成功后 wpa_supplicant 的回调函数会调用关联函数实现关联。

编程实例：

```
#include "hostapd/hostapd_if.h"
#include "wpa_supplicant/wpa_supplicant.h"
#include "hilink_adapt.h"
```

该函数是 hilink 的总入口函数，负责调用创建 AP 的函数和起 hilink 线程。

```
int hilink_demo_main(void)
{
    int                 l_ret;
    unsigned int        ul_ret;
    unsigned char       uc_status;
    TSK_INIT_PARAM_S stappTask;
    hsl_demo_set_status(0);
    uc_status = hilink_demo_get_status();
```

```
    if (HILINK_STATUS_RECEIVE == uc_status)
    {
        HISI_PRINT_ERROR("hsl already start,cannot start again");
        return -HISI_EFAIL;
    }
    hilink_demo_set_status(HILINK_STATUS_RECEIVE);
    /*  起特殊 AP */
    l_ret = hilink_demo_prepare();
    if (HISI_SUCC != l_ret)
    {
        HISI_PRINT_ERROR("%s[%d]:demo init fail\n",__func__,__LINE__);
        return -HISI_EFAIL;
    }

    /*  创建切信道线程,线程结束后自动释放  */
    memset(&stappTask, 0, sizeof(TSK_INIT_PARAM_S));
    stappTask.pfnTaskEntry = (TSK_ENTRY_FUNC)hilink_demo_task_channel_change;
    stappTask.uwStackSize    = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
    stappTask.pcName = "hilink_thread";
    stappTask.usTaskPrio = 8;
    stappTask.uwResved      = LOS_TASK_STATUS_DETACHED;
    ul_ret = LOS_TaskCreate(&gul_hilink_taskid, &stappTask);
    if(0 != ul_ret)
    {
        HISI_PRINT_ERROR("%s[%d]:create task fail[%d]\n",__func__,__LINE__,ul_ret);
        return -HISI_EFAIL;
    }
    return HISI_SUCC;
}
```

该函数主要负责起特殊 SSID 的 AP 并初始化 hilink 库

```
int hilink_demo_prepare(void)
{
     hilink_s_pkt0len          st_pkt_len;
```

```
    int                         l_ret;
    char                        auc_ssid[33];
    unsigned int                ul_ssid_len = 0;
    struct hostapd_conf     hapd_conf;

    /* 设备信息 */
    unsigned char           auc_ssid_type[]                 = "01";
    unsigned char           auc_device_id[]                 = "9001";
    unsigned char           auc_device_sn[]                 = "X";
    unsigned char           auc_sec_type[]                  = "B";
    unsigned char           auc_sec_key[]                   =
"00000000000000000000";

    /* 尝试删除 wpa_supplicant 和 hostapd */
    l_ret = wpa_supplicant_stop();
    if (HISI_SUCC != l_ret)
    {
        HISI_PRINT_WARNING("%s[%d]:wpa_supplicant stop
fail",__func__,__LINE__);
    }
    l_ret = hostapd_stop();
    if (HISI_SUCC != l_ret)
    {
        HISI_PRINT_WARNING("%s[%d]:hostapd stop fail",__func__,__LINE__);
    }
    memset(&g_st_hilink_context, 0, sizeof(hilink_s_context));

    /* 初始化 hilink 库函数 */
    l_ret = hilink_link_init(&g_st_hilink_context);
    if (HISI_SUCC == l_ret)
    {
        hilink_link_reset();
        st_pkt_len.len_open = OPEN_BASE_LEN;
        st_pkt_len.len_wep   = WEP_BASE_LEN;
```

```
            st_pkt_len.len_tkip = TKIP_BASE_LEN;
            st_pkt_len.len_aes    = AES_BASE_LEN;
            /*  设置基准长度  */
            hilink_link_set_pkt0len(&st_pkt_len);
        }
        memset(auc_ssid, 0, sizeof(unsigned char)*33);
        /*  启动 hostapd */

hilink_link_get_devicessid(auc_ssid_type,auc_device_id,auc_device_sn,auc_sec_type,auc_se
c_key,auc_ssid,&ul_ssid_len);
        /*  设置 AP 参数  */
        memset(&hapd_conf, 0, sizeof(struct hostapd_conf));
        if (ul_ssid_len > MAX_ESSID_LEN + 1)
        {
            HISI_PRINT_ERROR("%s[%d]:ul_ssid_len = %d more than
33",__func__,__LINE__,ul_ssid_len);
            return -HISI_EFAIL;
        }
        memcpy(hapd_conf.ssid, auc_ssid, ul_ssid_len);
        memcpy(hapd_conf.ht_capab, "[HT40+]", strlen("[HT40+]"));
        hapd_conf.channel_num                = 6;
        hapd_conf.wpa_key_mgmt               = WPA_KEY_MGMT_PSK;
        hapd_conf.authmode                   = HOSTAPD_SECURITY_WPAPSK;
        hapd_conf.wpa                        = 2;

        hapd_conf.wpa_pairwise               = WPA_CIPHER_TKIP | WPA_CIPHER_CCMP;
        memcpy(hapd_conf.driver, "hisi", 5);
        memcpy((char *)hapd_conf.key,"hilink123",strlen("hilink123"));
        l_ret = hostapd_start(NULL, &hapd_conf);
        if (0 != l_ret)
        {
            HISI_PRINT_ERROR("%s[%d]:hostapd start
failed[%d]",__func__,__LINE__,l_ret);
            return -HISI_EFAIL;
        }
```

```
    return HISI_SUCC;
}
```

该函数主要负责配置 wifi 芯片进入 monitor 模式并注册处理上报数据包的回调函数

```
int hilink_demo_init(void)
{
    int l_ret;

    memset(&gst_hilink_result, 0, sizeof(hilink_s_result));
    l_ret = hisi_hilink_adapt_init();
    if (HISI_SUCC != l_ret)
    {
        HISI_PRINT_ERROR("%s[%d]:hilink_adapt_init fail",__func__,__LINE__);
        return -HISI_EFAIL;
    }
    return HISI_SUCC;
}
```

该函数是 hilink 主线程，负责处理上报的数据包并定时切换信道。

```
void hilink_demo_task_channel_change(void)
{
    int                 l_ret;
    unsigned int        ul_ret;
    unsigned int        ul_time          = 0;
    unsigned int        ul_time_temp = 0;
    unsigned int        ul_wait_time = 0;
    unsigned char       uc_status;

    ul_time_temp = hilink_get_time();
    uc_status = hilink_demo_get_status();
    while((HILINK_STATUS_RECEIVE == uc_status) &&
(HILINK_PERIOD_TIMEOUT >= ul_wait_time))
    {
        if (RECEIVE_FLAG_ON == hilink_receive_flag)
        {
            ul_time_temp = hilink_get_time();
```

```
                if (HILINK_PERIOD_CHANNEL < ul_time_temp - ul_time)
                {
                    ul_wait_time += (ul_time_temp - ul_time);
                    ul_time = ul_time_temp;

                    l_ret = hilink_link_get_lock_ready();
                    if (0 == l_ret)
                    {
                        l_ret = hisi_hilink_change_channel();
                        if (HISI_SUCC != l_ret)
                        {
                            HISI_PRINT_WARNING("%s[%d]:change channel
fail\n",__func__,__LINE__);
                        }
                        /* 复位一下 hilink 参数 */
                        hilink_link_reset();
                    }
                }
            }
            l_ret = hisi_hilink_parse_data();
            if (HI_WIFI_STATUS_FINISH == l_ret)
            {
                LOS_TaskLock();
                hilink_receive_flag = RECEIVE_FLAG_OFF;
                LOS_TaskUnlock();
                hilink_demo_set_status(HILINK_STATUS_CONNECT);
                break;
            }
            msleep(1);
            uc_status = hilink_demo_get_status();
        }
        hisi_hilink_adapt_deinit();
        if (HILINK_PERIOD_TIMEOUT >= ul_wait_time)
        {
```

```
            hsl_memset(&gst_hilink_result, 0, sizeof(hilink_s_result));
            l_ret = hilink_link_get_result(&gst_hilink_result);
            if (HISI_SUCC == l_ret)
            {
                l_ret = hilink_demo_connect_prepare();
                if (HISI_SUCC != l_ret)
                {
                    hilink_demo_set_status(HILINK_STATUS_UNCREATE);
                }
            }
        }
        else
        {
            hilink_demo_set_status(HILINK_STATUS_UNCREATE);

        }
        ul_ret = LOS_TaskDelete(gul_hilink_taskid);
        if (HISI_SUCC != ul_ret)
        {
            HISI_PRINT_WARNING("%s[%d]:delete task
fail[%d]",__func__,__LINE__,ul_ret);
        }
}
```

该函数负责将设备有 AP 模式切换到 STA 模式

```
int hilink_demo_connect_prepare(void)
{
    int                         l_ret;
    hilink_s_result             st_hilink_result;

    /* 删除 AP 模式 */
    l_ret = hostapd_stop();
    if (HISI_SUCC!= l_ret)
    {
        HISI_PRINT_ERROR("%s[%d]:stop hostapd fail\n",__func__,__LINE__);
```

```
            return -HISI_EFAIL;
        }

#ifdef CONFIG_NO_CONFIG_WRITE
        l_ret = wpa_supplicant_start("wlan0", "hisi", NULL);
#else
        l_ret = wpa_supplicant_start("wlan0", "hisi", WPA_SUPPLICANT_CONFIG_PATH);
#endif
        if (HISI_SUCC != l_ret)
        {
            HISI_PRINT_ERROR("%s[%d]:start wpa_supplicant fail\n",__func__,__LINE__);
            return -HISI_EFAIL;
        }
return HISI_SUCC;
}
```

该函数主要负责关联

```
int hilink_demo_connect(hilink_s_result* pst_result)
{
        struct wpa_assoc_request wpa_assoc_req;

        memset(&wpa_assoc_req, 0, sizeof(struct wpa_assoc_request));
        wpa_assoc_req.hidden_ssid = 1;
        if (33 < pst_result->ssid_len)
        {
            HISI_PRINT_ERROR("%s[%d]:ssd_len = %d more than
33",__func__,__LINE__,pst_result->ssid_len);
            wpa_supplicant_stop();
            return -HISI_EFAIL;
        }
        strncpy(wpa_assoc_req.ssid, pst_result->ssid, pst_result->ssid_len);
        HISI_PRINT_INFO("SSID:%s\n",pst_result->ssid);
        HISI_PRINT_INFO("en_type:%d\n",pst_result->enc_type);
        HISI_PRINT_INFO("sendtype:%d\n",pst_result->sendtype);
        wpa_assoc_req.auth = pst_result->enc_type;
```

```
    if (HI_WIFI_ENC_OPEN != wpa_assoc_req.auth)
    {
        if (128 < pst_result->pwd_len)
        {
            HISI_PRINT_ERROR("%s[%d]:pwd_len = %d is more than
128",__func__,__LINE__,pst_result->pwd_len);
            wpa_supplicant_stop();
            return -HISI_EFAIL;
        }
        strncpy(wpa_assoc_req.key, pst_result->pwd, pst_result->pwd_len);
    }
    wpa_cli_connect (&wpa_assoc_req);
    return HISI_SUCC;
}
```

3.1.7 海思快连

- 用户首先调用 Hisi_link 库函数接口 hsl_get_ap_params 获取将要创建 AP 的特殊 SSID。
- 获取到特殊 SSID 后用户调用 hostapd_start 接口创建特殊 SSID 的 AP。
- AP 创建成功后，hostapd 回调函数会调用 hsl_os_init 函数初始化 hisi_link 库，调用 hisi_hsl_adapt_init 接口实现开启 Wi-Fi 芯片的 monitor 模式以及注册上报数据包处理函数的的操作。
- 同时 hisi_link 线程会循环处理数据队列中的数据包，并周期性切换信道。
- 当解码成功或解码超时时用户需调用 hisi_hsl_adapt_deinit 关闭 monitor 功能。
- 当解码成功时会调用 hsl_demo_connect_prepare 将设备模式由 AP 切回 STA。
- 当 STA 创建成功后 wpa_supplicant 的回调函数会调用关联函数实现关联。

编程示例：

```
#include "hostapd/hostapd_if.h"
#include "wpa_supplicant/wpa_supplicant.h"
#include "hisilink_adapt.h"
```

该函数是 hisi_link 的总入口，负责起特殊 SSID 的 AP 以及创建 hisi_link 的主线程

```
hsl_int32 hsl_demo_main(void)
{
    unsigned int        ul_ret;
    int                 l_ret;
```

```
unsigned char       uc_status;
TSK_INIT_PARAM_S    st_hsl_task;
TSK_INIT_PARAM_S    st_apmodetask;

hilink_demo_set_status(0);
uc_status = hsl_demo_get_status();
if (HSL_STATUS_RECEIVE == uc_status)
{
    HISI_PRINT_ERROR("hsl already start,cannot start again");
    return -HSL_FAIL;
}
hsl_demo_set_status(HSL_STATUS_RECEIVE);
l_ret = hsl_demo_prepare();
if (0 != l_ret)
{
    hsl_demo_set_status(HSL_STATUS_UNCREATE);
    HISI_PRINT_ERROR("%s[%d]:demo init fail",__func__,__LINE__);
    return -HSL_FAIL;
}

/* 创建切信道线程,线程结束后自动释放 */
memset(&st_hsl_task, 0, sizeof(TSK_INIT_PARAM_S));
st_hsl_task.pfnTaskEntry = (TSK_ENTRY_FUNC)hsl_demo_task_channel_change;

st_hsl_task.uwStackSize    = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
st_hsl_task.pcName         = "hsl_thread";
st_hsl_task.usTaskPrio     = 8;
st_hsl_task.uwResved       = LOS_TASK_STATUS_DETACHED;
ul_ret = LOS_TaskCreate(&gul_hsl_taskid, &st_hsl_task);
if(0 != ul_ret)
{
    hsl_demo_set_status(HSL_STATUS_UNCREATE);
    HISI_PRINT_ERROR("%s[%d]:create task fail[%d]",__func__,__LINE__,ul_ret);
```

```
        return -HSL_FAIL;
    }
    return HSL_SUCC;
}
```

该函数主要是通过 hisi_link 库函数获取特殊 SSID 并启动 hostapd

```
int hsl_demo_prepare(void)
{
    int                 l_ret;
    char                ac_ssid[33];
    unsigned char   auc_password[65];
    unsigned int    ul_ssid_len = 0;
    unsigned int    ul_pass_len = 0;
    unsigned char   auc_manufacturer_id[3] = {'0','0','3'};
    unsigned char   auc_device_id[3]       = {'0','0','1'};
    struct netif   *pst_netif;
    ip_addr_t       st_gw;
    ip_addr_t       st_ipaddr;
    ip_addr_t       st_netmask;
    struct hostapd_conf hapd_conf;

    /* 尝试删除 wpa_supplicant 和 hostapd */
    l_ret = wpa_supplicant_stop();
    if (0 != l_ret)
    {
        HISI_PRINT_WARNING("%s[%d]:wpa_supplicant stop
fail",__func__,__LINE__);
    }
    pst_netif = netif_find("wlan0");
    if (NULL != pst_netif)
    {
        dhcps_stop(pst_netif);
    }
    l_ret = hostapd_stop();
    if (0 != l_ret)
```

```
    {
        HISI_PRINT_WARNING("%s[%d]:hostapd stop fail",__func__,__LINE__);
    }
    memset(ac_ssid, 0, sizeof(char)*33);
    /* 获取启动 AP 需要的 SSID 及密码 */
    l_ret = hsl_get_ap_params(auc_manufacturer_id, auc_device_id, ac_ssid, &ul_ssid_len,
auc_password, &ul_pass_len);
    if (HSL_SUCC != l_ret)
    {
        HISI_PRINT_ERROR("%s[%d]:get ap params fail",__func__,__LINE__);
        return -HSL_FAIL;
    }

    /* 设置 AP 参数 */
    hsl_memset(&hapd_conf, 0, sizeof(struct hostapd_conf));
    hsl_memcpy(hapd_conf.ssid, ac_ssid, ul_ssid_len);
    hsl_memcpy(hapd_conf.key, auc_password, ul_pass_len);
    hsl_memcpy(hapd_conf.ht_capab, "[HT40+]", strlen("[HT40+]"));
    hapd_conf.channel_num                = 6;
    hapd_conf.wpa_key_mgmt               = WPA_KEY_MGMT_PSK;
    hapd_conf.wpa                        = 2;
    hapd_conf.authmode                   = HOSTAPD_SECURITY_WPAPSK;
    hapd_conf.wpa_pairwise               = WPA_CIPHER_TKIP | WPA_CIPHER_CCMP;
    hsl_memcpy(hapd_conf.driver, "hisi", 5);
#ifdef HISI_CONNETIVITY_PATCH
    hapd_conf.ignore_broadcast_ssid = 0;
#endif
    l_ret = hostapd_start(NULL, &hapd_conf);
    if (HSL_SUCC != l_ret)
    {
        HISI_PRINT_ERROR("HSL_start_hostapd:start failed[%d]",l_ret);
        return -HSL_FAIL;
    }
    return HSL_SUCC;
```

```
}
```

该函数在 hostapd 创建成功后被调用，用来配置 wifi 芯片并初始化 hisi_link 库函数

```
int hsl_demo_init(void)
{
    int l_ret;

    memset(&gst_hsl_params, 0, sizeof(hsl_result_stru));
    l_ret = hsl_os_init(&st_context);
    if (HSL_SUCC != l_ret)
    {
        hsl_demo_set_status(HSL_STATUS_UNCREATE);
        return -HSL_FAIL;
    }
    l_ret = hisi_hsl_adapt_init();
    if (HISI_SUCC != l_ret)
    {
        hsl_demo_set_status(HSL_STATUS_UNCREATE);
        HISI_PRINT_ERROR("%s[%d]:hisi_HSL_adapt_init fail",__func__,__LINE__);
        return -HSL_FAIL;
    }
    return HSL_SUCC;
}
```

该函数是 hisi_link 主线程，用来循环处理队列的数据包，并周期性切换信道

```
void hsl_demo_task_channel_change(void)
{
    int                 l_ret;
    unsigned int        ul_ret;
    unsigned int        ul_time         = 0;
    unsigned int        ul_time_temp = 0;
    unsigned int        ul_wait_time = 0;
    unsigned char    *puc_macaddr;
    unsigned char     uc_status;
```

```
    ul_time_temp = hsl_get_time();
    uc_status       = hsl_demo_get_status();
    while((HSL_STATUS_RECEIVE == uc_status) && (HSL_PERIOD_TIMEOUT >= ul_wait_time))
    {
        if (RECEIVE_FLAG_ON == hsl_receive_flag)
        {
            ul_time_temp = hsl_get_time();
            if (HSL_PERIOD_CHANNEL < ul_time_temp - ul_time)
            {
                ul_time = ul_time_temp;
                ul_wait_time += HSL_PERIOD_CHANNEL;
                l_ret = hsl_get_lock_status();
                if (HSL_CHANNEL_UNLOCK == l_ret)
                {
                    l_ret = hisi_hsl_change_channel();
                    if (HISI_SUCC != l_ret)
                    {
                        HISI_PRINT_WARNING("%s[%d]:change channel fail",__func__,__LINE__);
                    }
                    hsl_os_reset();
                }
            }
        }
        l_ret = hisi_hsl_process_data();
        if (HSL_DATA_STATUS_FINISH == l_ret)
        {
            LOS_TaskLock();
            hsl_receive_flag = RECEIVE_FLAG_OFF;
            LOS_TaskUnlock();
            hsl_demo_set_status(HSL_STATUS_CONNECT);
            break;
        }
```

```
        msleep(1);
        uc_status = hsl_demo_get_status();
    }

    hisi_hsl_adapt_deinit();
    if (HSL_PERIOD_TIMEOUT >= ul_wait_time)
    {
        hsl_memset(&gst_hsl_params, 0, sizeof(hsl_result_stru));
        l_ret = hsl_get_result(&gst_hsl_params);
        if (HSL_SUCC == l_ret)
        {
            puc_macaddr = hisi_wlan_get_macaddr();
            if (HSL_NULL != puc_macaddr)
            {
                if ((puc_macaddr[4] == gst_hsl_params.auc_flag[0]) && (puc_macaddr[5]
== gst_hsl_params.auc_flag[1]))
                {
                    l_ret = hsl_demo_connect_prepare();
                    if (0 != l_ret)
                    {
                        hsl_demo_set_status(HSL_STATUS_UNCREATE);
                    }
                }
                else
                {
                    HISI_PRINT_ERROR("This device is not intended to be
associated:%02x %02x\n",gst_hsl_params.auc_flag[0],gst_hsl_params.auc_flag[1]);
                }
            }
        }
    }
    else
    {
        HISI_PRINT_INFO("hsl timeout\n");
```

```
            hsl_demo_set_status(HSL_STATUS_UNCREATE);
        }
        ul_ret = LOS_TaskDelete(gul_hsl_taskid);
        if (0 != ul_ret)
        {
            HISI_PRINT_WARNING("%s[%d]:delete task
fail[%d]",__func__,__LINE__,ul_ret);
        }
}
```

该函数是当获取结果成功后，将设备由 AP 模式切换到 STA 模式

```
hsl_int32 hsl_demo_connect_prepare(void)
{
    int                         l_ret;
    unsigned char               *puc_path       = HSL_NULL;
    struct netif                *pst_netif;

    /* 删除 AP 模式 */
    pst_netif = netif_find("wlan0");
    if (HSL_NULL != pst_netif)
    {
        dhcps_stop(pst_netif);
    }
    l_ret = hostapd_stop();
    if (0 != l_ret)
    {
        HISI_PRINT_ERROR("%s[%d]:stop hostapd fail",__func__,__LINE__);
        return -HSL_FAIL;
    }
#ifdef CONFIG_NO_CONFIG_WRITE
    l_ret = wpa_supplicant_start("wlan0", "hisi", HSL_NULL);
#else
    l_ret = wpa_supplicant_start("wlan0", "hisi", WPA_SUPPLICANT_CONFIG_PATH);
#endif
    if (0 != l_ret)
```

```
    {
        HISI_PRINT_ERROR("%s[%d]:start wpa_supplicant fail",__func__,__LINE__);
        return -HSL_FAIL;
    }
    return HSL_SUCC;
}
```

当 STA 启动成功后会调用关联函数实现关联

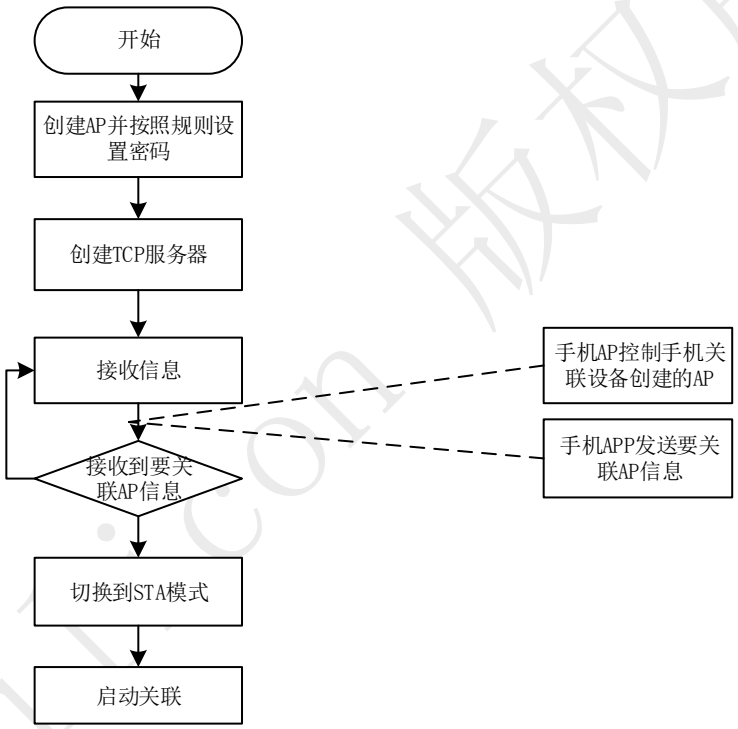
```
int hsl_demo_connect(hsl_result_stru* pst_params)
{
    struct wpa_assoc_request wpa_assoc_req;
    if (HSL_NULL == pst_params)
    {
        HISI_PRINT_ERROR("%s[%d]:pst_params is null",__func__,__LINE__);
        return -HSL_ERR_NULL;
    }
    hsl_memset(&wpa_assoc_req , 0 ,sizeof(struct wpa_assoc_request));
    wpa_assoc_req.hidden_ssid       = 1;
    hsl_memcpy(wpa_assoc_req.ssid, pst_params->auc_ssid, pst_params->uc_ssid_len);
    wpa_assoc_req.auth              = pst_params->en_auth_mode;

    if (HSL_AUTH_TYPE_OPEN != wpa_assoc_req.auth)
    {
        hsl_memcpy(wpa_assoc_req.key, pst_params->auc_pwd, pst_params->uc_pwd_len);
        HISI_PRINT_INFO("hsl_wpa_connect:pwd[%s]",wpa_assoc_req.key);
    }
    /* 开始创建连接 */
    wpa_cli_connect (&wpa_assoc_req);
    return HSL_SUCC;
}
```

3.1.8 传统模式

传统接入模式的流程如下图所示：

- 设备侧在进入传统接入模式后按照设定的规则生成 SSID 和 PWD（该规则由客户自己定义）。根据生成的 SSID 和 PWD 启动 hostapd 创建 AP。
- 在 AP 创建成功后，设备侧创建一个 TCP 服务器，等待手机 APP 发送过来的数据信息。
- 手机 APP 扫描到设备后，由用户点击连接按钮，手机 APP 会根据设定规则（该规则同设备侧保持一致）计算出设备侧 AP 的 PWD，并发起关联。
- 当手机关联到设备侧 AP 成功后，手机 APP 会向 TCP 服务器发送将要关联的路由器 SSID 和 PWD 等信息。
- 设备侧接收到该信息后进行解析，解析成功后设备侧由 AP 模式切换到 STA 模式，利用解析到的信息关联上将要关联的路由器。

图3-8 传统模式业务流程图



3.1.9 备份恢复

Flash 有擦写次数的约束，考虑到 Flash 的使用寿命，应用程序要考虑这个问题，设计相应的方案，比如：轮替擦写某几个 Flash 块。

3.1.9.1 下电备份写 flash 地址传递

1. 用户在 WiFi 初始化前调用 hisi_wlan_get_databk_addr_info()接口，获取 WiFi 驱动的备份信息结构体指针，并对以下参数赋值

(1) databk_addr ：写 flash 的起始地址；

(2) databk_length：写 flash 的长度。（不要小于 0x10000）；

(3) get_databk_info：上电恢复时重新获取备份存储信息的回调函数。

2. 编程实例：

```
void hi_wifi_register_backup_addr(void)
{
    struct databk_addr_info *wlan_databk_addr_info = hisi_wlan_get_databk_addr_info();
    if (NULL == wlan_databk_addr_info)
    {
        dprintf("hi_wifi_register_backup_addr:wlan_databk_addr_info is NULL!\n");
        return NULL;
    }
    /* 暂时写死，由应用侧补齐 */
    wlan_databk_addr_info->databk_addr = 0x7d0000;
    wlan_databk_addr_info->databk_length = 0x10000;
    wlan_databk_addr_info->get_databk_info = hi_wifi_databk_info_cb;
}

void hi_wifi_pre_proc()
{
    hi_wifi_register_init();
    hi_wifi_board_info_register();
    hi_wifi_check_wakeup_flag();
    hi_wifi_register_backup_addr();
    hi_rf_customize_init();
}
```

3.1.9.2 上电恢复获取 flash 地址信息

接上一节，已经为驱动的备份信息结构体指针传递了回调函数，驱动侧会调用该回调函数获取地址信息，回调函数变成实例：

```
struct databk_addr_info *hi_wifi_databk_info_cb(void)
{
    /* 暂时写死，由应用侧补齐 */
    struct databk_addr_info *wlan_databk_addr_info = NULL;
    wlan_databk_addr_info = malloc(sizeof(struct databk_addr_info));
    if (NULL == wlan_databk_addr_info)
    {
        dprintf("hi_wifi_databk_info_cb:wlan_databk_addr_info is NULL!\n");
        return NULL;
    }
    memset(wlan_databk_addr_info, 0, sizeof(struct databk_addr_info));
    wlan_databk_addr_info->databk_addr = 0x7d0000;
    wlan_databk_addr_info->databk_length = 0x10000;
    wlan_databk_addr_info->get_databk_info = NULL;
    return wlan_databk_addr_info;
}
```

注：初始化给 WiFi 驱动的用于下电备份的地址信息，一定要与上电恢复时从回调函数中获取到的地址信息一致。

3.1.10 异常处理

Wlan 发生异常时会保存关键异常信息到文件，对于没有文件系统的应用场景，wlan 驱动支持文件直接保存 flash，保存文件所使用的 flash 地址和长度由用户配置，配置接口请查看 3.1.3.6 小节。

文件保存所需 flash 空间大小有三种等级配置，详情如下：

WLAN_FILE_STORE_MIN_SIZE：不保存异常信息；

WLAN_FILE_STORE_MID_SIZE：保存部分异常信息；

WLAN_FILE_STORE_MAX_SIZE：保存全部异常信息；

注：Wlan 驱动默认配置为 WLAN_FILE_STORE_MIN_SIZE，用户请根据需要配置，配置时请确保 flash 地址和长度的合理性。

4 定制化说明

产测版本和正式版本的定制化操作，请参考文档《Hi1131S V100 Wi-Fi 硬件测试方案.docx》中的“定制化说明”这一章节，本文档不在赘述。