FR801xH 系列芯片 FAQ

Bluetooth Low Energy SOC

www.freqchip.com

目录

1. 概述

该文章总结了 Fr801xH 芯片开发和生产阶段客户提到的问题及其解答。

主要分为如下几个章节：

1. 工具以及开发环境使用
2. 外设驱动
3. 协议栈说明
4. 电路以及电器参数

本文为“ Fr801xH 系列芯片 FAQ”针对的是 Fr801xH 系列芯片都存在的特性和使用方法，并不针对某个特定的芯片。虽然每个芯片都有这样的特性，但是实现方式上会略有区别，这里可能只是以某个芯片来概括说明，具体到某个芯片还要灵活运用。一般而言对于已经存在的芯片如果其他芯片的实现方式不一样，本文会提及。

2. 工具以及开发环境使用

2.1. Fr801xH 的芯片是否有入门指南，简单概况芯片的开发和使用？

A：请看文章《Fr801xH 快速入门.pdf》和《Fr801xH 如何构建系统.pdf》，这两篇文章简单介绍了 Fr801xH 芯片的开发过程以及 SDK 的使用方法，该文章可以从 Fr801xH 的 SDK\docs\Application Notes 上拿到。

2.2. 如何使用 SWD 进行 JLINK 仿真和使用 UART 串口打印？

A：请看文章《Fr801xH 快速入门.pdf》里面第 7 节和第 8.2 节，简单介绍了 Fr801xH 芯片如何查看串口 log 和使用 jlink 进行下载调试，该文章可以从 Fr801xH 的 SDK\docs\Application Notes 上拿到。

2.3. 如何使用 PC 工具进行程序下载

A： Fr801xH 的 PC 工具采用串口下载，具体的操作步骤参见《FREQ BLE SDK User Guide.pdf》内部 1.6.2 节。该文章可以从 Fr801xH 的 SDK\docs/上拿到。

2.4. 如何使用手机 app 进行 OTA 在线升级

A： 第一步，在手机上安装 Fr801xH 的 OTA 升级 app 软件； 第二步，在工程中引用 ota service profile 的文件，文件在 sdk\ components\ble\profiles\ble_ota 内；第三步，在工程初始化时，调用 ota_gatt_add_service()添加 OTA 的服务。第四步，打开手机 OTA 的 app 软件，链接设备后，选择要下载的 bin 文件，开始进行 OTA 下载。

2.5. 如何使用烧录板进行工厂模式量产烧录

A: Fr801xH 系列芯片量产烧录工具，可以支持烧录螺片，也可烧录 PCBA，具体实施方式可以联系代理商或应用工程师。

一般烧录步骤如下：

1 插上 USB 线，接到 PC 机上，PC 上会识别出一个 U 盘，把要烧录的 bin 文件，放入 U 盘。

2 拔插 USB 线，让烧录板识别要烧录的 bin 文件。

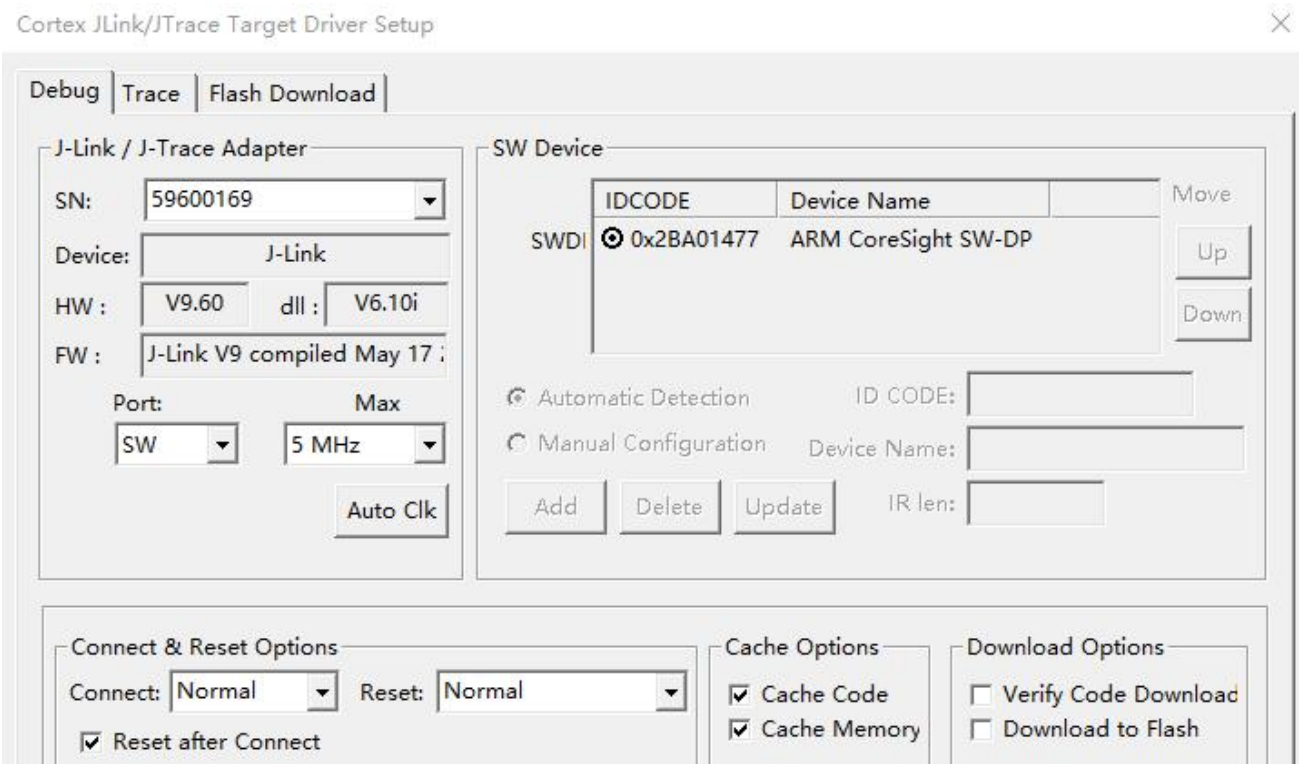
3 将烧录板的右侧插针中的 VBAT/GND/TXD/RXD 分别接到被烧录芯片的 VBAT/GND/PA3/PA2 管脚。

4 测试烧录，按下做下方的按钮，开始。工厂机台需要发送 100ms 高电压脉冲信号到下面排针中的 start 脚。

5 烧录结束后，如果烧录成功，亮绿灯，同时下面排针中的 OK 脚出现 100ms 高电平脉冲。 烧录失败，亮红灯，同时下面排针中的 NG 脚出现 100ms 高电平脉冲。

2.6. Jlink 在线调试功能为什么不能使用

A: 首先在 keil 工程的 Options for Target 页面，Debug 页面检查 Jlink 是否找到 arm 设备。如下图。



然后在 patch.c 屏蔽掉如下两个 patch 入口点。

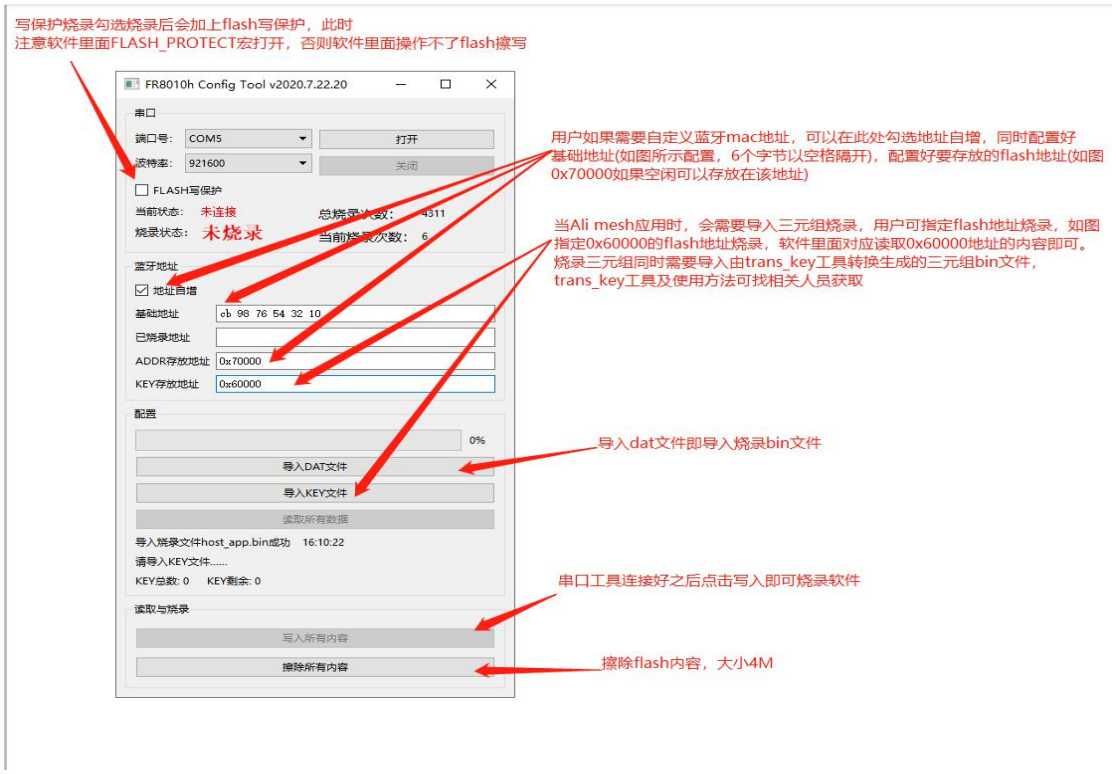
```
#if 0
    [4] = {
        .patch_pc = 0x00004aae,
        .replace_function = frspim_rd,
    },
    [3] = {
        .patch_pc = 0x00004b02,
        .replace_function = frspim_wr,
    },
#endif
```

调试完毕后，正常代码不能屏蔽以上两个 patch 入口点。

2.7. 如何关闭串口 log

user_custom_parameters 函数里加上两句
__jump_table.system_option &= ~SYSTEM_OPTION_PRINT_MASK;
__jump_table.system_option |= SYSTEM_OPTION_PRINT_SWO;

2.8. 串口烧录工具配置介绍



3. 外设驱动

3.1. Fr801xH 系列芯片 GPIO 默认状态

A： Fr801xH 系列芯片所有的 GPIO 管脚除了 PA2 和 PA3 外上电默认都处于低电压。芯片的 Rom Code 将 PA2，PA3 默认初始化为 UART1 串口的 RXD 脚和 TXD 脚（波特率 115200）。并且 PA3 脚会打印一次 freq 的字符

3.2. Fr801xH 芯片如何开始/停止保持链接的休眠和进入完全的 powerdown 模式

A：芯片的 sdk 自带的 lib 库默认会开启可以保持链接的休眠，在无广播和无链接的情况下，芯片每隔 10 钟会自动唤醒一次，然后继续休眠。开启广播，保持链接，或者启动了软件定时器的情况下，芯片的唤醒时间由这三个事件的执行时间决定。

在项目中调用 system_sleep_disable();停止保持链接的休眠。调用 system_sleep_enable();开始保持链接的休眠。

在项目中调用 void system_power_off(bool aldo_bypass)后系统进入 powerdown 模式，功耗在 2uA。在进入 pwoerdown 模式之前，设置了 gpio 按键唤醒的 gpio 能唤醒。

3.3. Fr801xH 系列芯片 IO 口驱动能力说明

A： Fr801xH 芯片的 IO 口在输出高电平是默认的最大驱动电流为 12mA。使用芯片 ALDO 管脚做为电源输出

时，ALDO 管脚的最大驱动电流为 100mA。

3.4. 如何获取外设驱动代码和驱动使用的示例代码

A：芯片的外围设备大部分的驱动文件在 SDK\components\driver 内，驱动的使用示例代码在 SDK\ \examples\none_evm\ble_drivers_demo 示例工程内。

3.5. Fr801xH 芯片的 LED1、LED2 管脚能否用作普通 GPIO，能否用作 pwm 输出?

A：芯片的 LED1、LED2 管脚可以在管脚紧张的情况下，用作普通的 GPIO 输出，或 pwm 的输出，但不能做为输入使用。具体的使用示例代码可以参考 SDK\ \examples\none_evm\ble_drivers_demo 示例工程里 led 部分。

3.6. 芯片处于掉电模式下，电源低电压报警能否唤醒芯片?

A: 低电压报警属于 pmu 的逻辑模块，在芯片处于掉电模式时，是持续工作的，这种情况下，低电压报警可以通过 pmu 中断唤醒芯片。低电压报警的最高电压为 2.5V，不适用于锂电池供电场景。

3.7. 关于 adc 的使用

Adc 相关的参考代码可以看 driver_adc.h 里面，有相应的参数配置；

Adc 分为两个部分，一个是内部直接检测 VBAT 脚电压的 adc，另外是 IO 口复用的几路 adc 做外部电压采集，精度是 10bit；

```
/*******************************************************************************
                          USAGE SAMPLE
    1. get vbat value
        struct adc_cfg_t cfg;
        uint16_t result, ref_vol;

        memset((void*)&cfg, 0, sizeof(cfg));
        cfg.src = ADC_TRANS_SOURCE_VBAT;
        cfg.ref_sel = ADC_REFERENCE_INTERNAL;
        cfg.int_ref_cfg = ADC_INTERNAL_REF_1_2;
        cfg.clk_sel = ADC_SAMPLE_CLK_24M_DIV13;
        cfg.clk_div = 0x3f;
        adc_init(&cfg);
        adc_enable(NULL, NULL, 0);

        adc_get_result(ADC_TRANS_SOURCE_VBAT, 0, &result);
        ref_vol = adc_get_ref_voltage(ADC_REFERENCE_INTERNAL);
        // vbat_vol = (result * 4 * ref_vol) / 1024 mV.

    2. sample voltage from PAD
        struct adc_cfg_t cfg;
        uint16_t result;

        system_set_port_mux(GPIO_PORT_D, GPIO_BIT_4, PORTD4_FUNC_ADC0);

        memset((void*)&cfg, 0, sizeof(cfg));
        cfg.src = ADC_TRANS_SOURCE_PAD;
        cfg.ref_sel = ADC_REFERENCE_AVDD;
        cfg.channels = 0x01;
        cfg.route.pad_to_sample = 1;
        cfg.clk_sel = ADC_SAMPLE_CLK_24M_DIV13;
        cfg.clk_div = 0x3f;
        adc_init(&cfg);
        adc_enable(NULL, NULL, 0);

        adc_get_result(ADC_TRANS_SOURCE_PAD, 0x01, &result);
*/
```

A：用内部 adc 直接采集 VBAT 脚电压，参考例程 1：get vbat value，通过获取校准值 ref_vol 与 adc 采样值计算出采样电压：vbat_vol = (result * 4 * ref_vol) / 1024;单位 mV；

B：IO 复用 adc 采集外部电压，参考例程 2：sample voltage form PAD，也是可以直接 copy 配置使用，注意通道配置，外部 ADC 有四路，对应 PD4~PD7,初始化时将对应 IO 配置成对应 ADC 功能，然后 cfg.channels 做通道配

置，一个 bit 表示一个通道：PD4=BIT0=通道 0、PD5=BIT1=通道 1、PD6=BIT2=通道 2、PD7=BIT3=通道 3；可以四个通道同时使能，即 chl=0x0f;Get result 时也注意下通道参数配置。

通过获取的 adc 值得到电压值方式：

ref_vol = adc_get_ref_voltage(ADC_REFERENCE_AVDD);

vbat_vol = (result * ref_vol) / 1024;

注意：改参考电压是对 aldo 2.9v 做的校准，如果软件配置 aldo 电压改变了，该校准值 ref_vol 就没有太大参考价值了。

3.8. 中断的使能以及中断接口和中断向量表的对应

app_boot_vectors.s

```
22  ; <h> Stack Configuration
23  ;   <o> Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>
24  ; </h>
25
26                  import Reset_Handler
27                  import NMI_Handler
28                  ;import HardFault_Handler
29                  import MemManage_Handler
30                  import BusFault_Handler
31                  import UsageFault_Handler
32                  ;import SVC_Handler
33                  import DebugMon_Handler
34                  import PendSV_Handler
35                  import SysTick_Handler
36                  import rwble_isr_patch
37                  import timer0_isr_ram
38                  import timer1_isr_ram
39                  import uart0_isr_ram
40                  import uart1_isr
41                  ;import pmu_isr_ram
42                  import i2s_isr_ram
43                  import exti_isr_ram
44                  import pdm_isr
45
46
```

app_boot_vectors.s

```
58
59  __initial_sp    EQU     0x40004000
60
61                  DCD     __initial_sp ;  ?ú3?ê??¯oóò??°?úMAINo¯êy?ú??ê1ó?EXMI
62                  DCD     Reset_Handler             ; Reset Handler
63                  DCD     NMI_Handler               ; NMI Handler
64                  DCD     HardFault_Handler_Ram     ; Hard Fault Handler
65                  DCD     MemManage_Handler         ; MPU Fault Handler
66                  DCD     BusFault_Handler          ; Bus Fault Handler
67                  DCD     UsageFault_Handler        ; Usage Fault Handler
68                  DCD     0                         ; Reserved
69                  DCD     0                         ; Reserved
70                  DCD     0                         ; Reserved
71                  DCD     0                         ; Reserved
72                  DCD     SVC_Handler               ; SVCall Handler
73                  DCD     DebugMon_Handler          ; Debug Monitor Handler
74                  DCD     0                         ; Reserved
75                  DCD     PendSV_Handler            ; PendSV Handler
76                  DCD     SysTick_Handler           ; SysTick Handler
77
78                  DCD     rwble_isr_patch             ; 0
79                  DCD     timer0_isr_ram              ; 1
80                  DCD     timer1_isr_ram              ; 2
81                  DCD     uart0_isr_ram               ; 3
82                  DCD     uart1_isr                 ; 4
83                  DCD     ssp_isr_ram                 ; 5
84                  DCD     0                         ; 6
85                  DCD     0                         ; 7
86                  DCD     0                         ; 8
87                  DCD     i2s_isr_ram               ; 9
88                  DCD     exti_isr_ram              ; 10
89                  DCD     pmu_isr_ram               ; 11
90                  DCD     adc_isr                   ; 12
91                  DCD     pdm_isr                   ; 13
92                  DCD     0                         ; 14
93                  DCD     0                         ; 15
94
```

例如写好串口中断 uart0_isr_ram，需要在 app_boot_vectors.s 里面做好对应，位置就按当前 app_boot_vectors.s 的来写。非 PMU 的中断需用“NVIC_EnableIRQ”接口使能中断，有必要设置中断优先级时可用“NVIC_SetPriority”接口设置中断优先级，优先级设置最好从 2 往上设置(即 2/3/4...)。Pmu 的中断直接用“NVIC_EnableIRQ”使能 pmu 中断即可。

3.9. 参考模块 button.c 里面怎么改成高电平触发

```
static void button_anti_shake_timeout_handler(void *param)
{
    uint32_t curr_button;
    os_event_t toggle_event;

    curr_button = ool_read32(PMU_REG_GPIOA_V);
    co_printf("=shake=%x %x\r\n",curr_button,button_io_mask);
    curr_button &= button_io_mask;

    if(curr_button == curr_button_before_anti_shake)
    {
        //curr_button ^= button_io_mask;  // 高电平做按键触发则屏蔽该行
        toggle_event.event_id = BUTTON_TOGGLE;
        toggle_event.param = (void *)&curr_button;
        toggle_event.param_len = sizeof(uint32_t);
        os_msg_post(button_task_id, &toggle_event);
    }
}
```

3.10. rtc 怎么使用，如何获取从 rtc 启动开始经过了多长时间

rtc_init();
rtc_alarm(RTC_A,10000); // 10S
触发接口：rtc_isr_ram
从 rtc 启动开始之后经过的时间：
Uint32_t rtc_count = ool_read32(PMU_REG_RTC_ALMA_VALUE_0);
rtc_count = rtc_count/pmu_get_rc_clk(false); // ms
最终的 rtc_count 即为经过的时间。

3.11. 软件如何看睡眠唤醒流程

打开睡眠调用 system_sleep_enable 就可以了，关闭睡眠调用 system_sleep_disable。可以在 user_entry_before_sleep_imp 里面搞个 uart_putc_noint(UART1,'s');然后再 user_entry_after_sleep_imp 里面搞个 uart_putc_noint(UART1,'w');用串口打印看睡眠唤醒流程以及间隔时间。

4. 协议栈说明

4.1. Fr801xH 芯片的蓝牙事件处理回调函数中各个分支代表着怎么样的事件？在什么时候发生？

A: 要获取协议栈底层的蓝牙事件，首先要调用函数 gap_set_cb_func(proj_ble_gap_evt_func);设置一个接收事件的回调函数。协议栈如果监听到底层蓝牙的状态发生变化时，将通过回调函数通知应用层，传回的变量为 gap_event_t 类型的结构体。通过查看结构体的 type 元素获得事件类型。

```
typedef enum
{
    GAP_EVT_ALL_SVC_ADDED,          //!< All GATT servcie have been added

    GAP_EVT_SLAVE_CONNECT,          //!< Connected as slave role
    GAP_EVT_MASTER_CONNECT,         //!< Connected as master role
    GAP_EVT_DISCONNECT,             //!< Disconnected
    GAP_EVT_LINK_PARAM_REJECT,      //!< Parameter update rejected
    GAP_EVT_LINK_PARAM_UPDATE,      //!< Parameter update successful
    GAP_EVT_PHY_REJECT,             //!< Parameter update rejected
    GAP_EVT_PHY_UPDATE,             //!< PHY update indication
    GAP_EVT_ADV_END,                //!< Advertising ended
    GAP_EVT_SCAN_END,               //!< Scanning ended
    GAP_EVT_PER_SYNC_ESTABLISHED,   //!< Per_sync is established
    GAP_EVT_PER_SYNC_END,           //!< Periodic adv sync event ended
    GAP_EVT_ADV_REPORT,             //!< Scan result report
    GAP_EVT_CONN_END,               //!< Connecion action is ended
    GAP_EVT_PEER_FEATURE,           //!< Got peer device supported features
    GAP_EVT_MTU,                    //!< MTU exchange event
    GAP_EVT_LINK_RSSI,              //!< Got RSSI value of link peer device

    GAP_SEC_EVT_MASTER_AUTH_REQ,    //!< As master role, got authentication request from slave
    GAP_SEC_EVT_MASTER_ENCRYPT,     //!< Link is encryted as master role
    GAP_SEC_EVT_SLAVE_ENCRYPT,      //!< Link is enrypted as slave role
} gap_event_type_t;
```

4.2. 如何在链接状态下获取 RSSI 信号质量？

A: 第一步在链接建立的时间回调内，使能 rssi_report 功能。例如在协议栈事件回调函数内分支：连接成功 运行如下代码：

```
case GAP_EVT_SLAVE_CONNECT:
{
    gap_set_link_rssi_report(true);
    gap_get_link_rssi(event->param.slave_connect.conidx);
```

调用一次查询函数 void gap_get_link_rssi(uint8_t conidx)，底层就会上报一次当前链接对端设备的 rssi 值。该值通过协议栈事件回调函数分支：得到 rssi 值 来获取。如下代码所示：

```
case GAP_EVT_LINK_RSSI:
    co_printf("link rssi %d\r\n",event->param.link_rssi);
    break;
```

4.3. 可否调用接口获取真随机数？

A: 可以调用接口获取真随机数，接口如下：uint32_t rand(void)。每调用一次返回一次真随机的 32bit 的数。协议栈在初始化时会获取真随机的种子数值，将该数值做为 rand 函数的种子，决定 rand 函数返回的是真随机数。

4.4. Fr801xH 芯片如何进行多连接的操作?

A： Fr801xH 芯片最多支持 20 个链接，可以在链接的同时进行广播和扫描的操作。同时支持最多 2 个广播。用户在进行多链接的工程时，可以参考文档《Fr801xH 如何构建多从机工程.pdf》。文档在 SDK\docs\Application Notes 获取。

另可以参考多链接的示例工程：SDK\examples\none_evm\ble_multi_role。该工程同时使能广播和扫描，如果扫描到要连设备名字，进行主动连接。如果被主机设备连接上后，能再次发送广播，再次被连接上。

也可参考 AT 的示例工程，该工程稍微复杂。

4.5. 是否有 hid 键盘和鼠标的示例工程代码?

A: Fr801xH SDK 内部有 hid 键盘和鼠标的示例工程代码。工程目录为:

键盘工程，SDK\examples\developing\ble_hid_kbd_mice。

鼠标工程，SDK\examples\developing\ble_hid_mouse_mice。

键盘和鼠标的工程，均能在主流的 IOS，安卓，和 win7/10 平台对接运行。

4.6. Fr801xH 系列芯片支持 5.0 的各种广播类型吗？

A: Fr801xH 系列芯片支持 5.0 协议规定的各种扩展广播和周期性广播，同时支持广播的白名单过滤。可以参考示例工程，SDK\ examples\none_evm\ble_aux_adv_demo 内定义的各种广播类型示例代码。

一共有 14 种广播类型的示例代码

```
#define TEST_ADV_MODE_UNDIRECT    (0)    //undirect connectable & scannable adv
#define TEST_ADV_MODE_DIRECT      (0)    //Low duty direct connectable & non-scannable adv
#define TEST_ADV_MODE_HDC_DIRECT (0)    //High duty direct connectable & non-scannable adv. 1.5ms interval

#define TEST_ADV_MODE_EXTEND_CONN_UNDIRECT    (0)    //extended undirect connectable & non-scannable adv
#define TEST_ADV_MODE_EXTEND_CONN_DIRECT      (0)    //extended direct connectable & non-scannable adv
#define TEST_ADV_MODE_EXTEND_NON_CONN_SCAN    (0)    //extended non-connectable & scannable adv
#define TEST_ADV_MODE_EXTEND_CONN_UNDIRECT_LONGRANGE    (0)  //extended longrange undirect connectable & non-scannable adv
#define TEST_ADV_MODE_EXTEND_CONN_DIRECT_LONGRANGE      (0)  //extended longrange direct connectable & non-scannable adv
#define TEST_ADV_MODE_EXTEND_NON_CONN_SCAN_LONGRANGE    (0)  //extended longrange non-connectable & scannable adv
#define TEST_ADV_MODE_PER_ADV_UNDIRECT        (0)    //periodic undirect non-connectable & scannable adv
#define TEST_ADV_MODE_PER_ADV_DIRECT          (0)    //periodic direct non-connectable & scannable adv

#define TEST_ADV_MODE_UNDIRECT_WHITE_LIST         (0)   //undirect connectable & scannable adv with withlist
#define TEST_ADV_MODE_UNDIRECT_DOUBLE_ADV         (1)


#define TEST_BOARD_ADV                (1)
#define TEST_BOARD_SCAN_OR_CONN       (0)
```

使用时，打开其中一个广播的宏，然后最下面的广播板宏打开和扫描_连接板的宏打开后，分别编译链接。将生成的广播板的 bin 文件烧录到第一块开发板，将扫描_连接板的 bin 文件烧录到第 2 块开发板，然后对两块板上电开始运行。

4.7. 有没有禁止和使能 latency 的接口函数

A: 可以调用接口禁止和使能链接的 latency，接口如下：void patch_set_lantency(uint16_t lantency)。在链接参数启用 latency 的情况下，可以通过调用 patch_set_lantency(0);和 patch_set_lantency(org_latency)来禁止和使能 latency。禁止 latency 的功能可以在大吞吐量传输的情况下使用，例如，在传输语音之前禁止 latency，语音传输结束后，在使能 latency。

4.8. 如何使用 ANCS 的 profile

A： Fr801xH 芯片的 SDK 内部提供苹果 IOS 平台 ANCS 服务的 profile 代码。代码文件在 SDK\ components\ble\profiles\ble_ANCS 获取。使用时引用改 profile 文件到工程中，然后参考该文件夹下的《说明.txt》文件进行调用。

4.9. Fr801xH 是否支持扫描以及使用方法

A: Fr801xH 的 BLE 芯片支持扫描功能，扫描的示例代码参考
示例工程，SDK\ examples\none_evm\ble_aux_adv_demo 内的扫描代码。

启动扫描前，如果要获取正确的 rssi 值，需要在启动扫描前调用 gap_set_link_rssi_report(true)开启 rssi 换算功能。在扫描结束后调用 gap_set_link_rssi_report(false)以提高射频性能。

扫描的搜到的广播包通过协议栈事件回调函数分支事件：GAP_EVT_ADV_REPORT 进行获取。

扫描结束后会触发协议栈事件回调函数分支事件：GAP_EVT_SCAN_END。

4.10. 启动睡眠、关闭广播、不开任何定时器的时候，为什么系统会 20s 唤醒一次

因为协议栈会 20s 起来一次，如果有小于 20s 的 timer 时，该 20s 的 timer 就不会触发。

4.11. 芯片唯一标识怎么获取

最新 SDK 有添加接口获取：efuse_get_chip_unique_id。

4.12. master 怎么与 slave 设备进行通信

Master 与 slave 通信时主要几个操作接口：
gatt_client_enable_ntf---使能 notify 功能
gatt_client_write_cmd---master 向 slave 写数据
gatt_client_read---master 读取 slave 的数据
gatt_notification---slave 设备上传数据到 master
Master 与 slave 通信时有一个 att_id 的参数，该 ID 号即是在 client 服务创建时“client_att_tb”数组中对应的序列号，该数组中涉及到的 UUID 就是 master 与 slave 通信时需要用到的 uuid，如下图所示，分别标记了 16 位 UUID 和 128 位 UUID 的用法：

```
#define SP_CHAR1_UUID              0xFFF1
#define SP_CHAR2_UUID              0xFFF2

const gatt_uuid_t client_att_tb[] =
{
    [0]  =
    { UUID_SIZE_2, UUID16_ARR(SP_CHAR1_UUID)}, // notify
    [1]  =
    { UUID_SIZE_2, UUID16_ARR(SP_CHAR2_UUID)}, // write
};
```

att_id　uuid length　uuid

```
#define SPSC_UUID128_ARR_RX   {0xb8, 0x5c, 0x49, 0xd2, 0x04, 0xa3, 0x40, 0x71, 0xa0, 0xb5, 0x35, 0x85, 0x3e, 0xb0, 0x83, 0x07}
#define SPSC_UUID128_ARR_TX   {0xba, 0x5c, 0x49, 0xd2, 0x04, 0xa3, 0x40, 0x71, 0xa0, 0xb5, 0x35, 0x85, 0x3e, 0xb0, 0x83, 0x07}

at_recv_data_func_t spsc_recv_data_ind_func = NULL;
uint16_t spsc_client_id;
gatt_uuid_t spsc_att_tb[] =
{
    [0]  ={ UUID_SIZE_16, SPSC_UUID128_ARR_RX}, //RX,   and for spss_TX
    [1]  ={ UUID_SIZE_16, SPSC_UUID128_ARR_TX}, //TX,,  and for spss_RX
};
```

128bit UUID

```
51: static void mr_central_init(void)
52: {
53:     gatt_client_t client;
54:
55:     client.p_att_tb = client_att_tb;
56:     client.att_nb = 2;
57:     client.gatt_msg_handler = mr_central_msg_handler;
58:     client_id = gatt_add_client(&client);
59:
60:     mr_central_start_scan();
61: }
```

master初始化，此处的att_nb对应client_att_tb里面的max index

5. 电路以及电气参数

5.1. Fr801xH 系列芯片的参考外围设计电路在哪里获取？

A:　Fr801xH 芯片的参考电路设计文档在 SDK\docs\Hardware Reference Design 内部，一共有 Fr8012H，Fr8016H 和 Fr8018H 三种型号的参考外围电路。

5.2. Fr801xH 芯片如何进行频偏和天线匹配校准？

A： 通常情况下用户在画完 PCB 板后，需要对射频性能进行一次性的优化。优化分为频偏校准和天线匹配校准。具体请联系我们的代理商或 AE 工程师。

5.3. Fr801xH 芯片调试功耗时如何测试底电流？

A： 在工程内调用 system_sleep_enable()后，芯片运行时就会使能保持链接的休眠。用电流表串接到电源线，然后给芯片上电，运行时将电流表打到毫安档，然后突然打到微安档后，看到电流跳变的最小值就是芯片休眠时的底电流。长时间运行在微安档，可能导致芯片重启。

另外推荐使用 EFM32 Kit(Leopard GECKO) 开发板进行电流测试。

5.4. Fr801xH 芯片的运行供电电压范围是多少？

A： Fr801xH 芯片的运行电压范围是 1.8V~4.2V。不在此范围的电压供电是，芯片不能工作。

5.5. Fr801xH 芯片的 IO 高电平是的电压范围是多少？

A： Fr801xH 芯片的 IO 口做为高电平输出时的电压与 ALDO 管脚的电压一致，可以在工程内调用函数 void pmu_set_aldo_voltage(enum pmu_aldo_work_mode_t mode, enum pmu_aldo_voltage_t value)进行设置，能够设置的范围是：2.1V~3.5V。

注意：该值设置越大，芯片运行时的功耗越大。如果设置的电压值比 VBAT 的电压高时，IO 口的实际高电平电压值就是 VBAT 的值。比如：VBAT 当前值是 3V，调用上面函数设置 IO 口高电平电压值为 3.5V，实际的 IO 口高电平电压值为 3V。

5.6. 该如何选择合适的发射功率

A: 用户可以在系统初始化时的入口函数内调用 void system_set_tx_power(enum rf_tx_power_t tx_power);调节射频的发射功率，范围从-16db~+10db。默认的发射功率是 1db。要提高远距离的射频收发性能时，需要提高发射功率。

5.7. 如何切换 CPU 的工作频率

A: 用户可以在 void user_custom_parameters(void)的入口函数内决定系统默认的 CPU 运行频率。示例代码如下：__jump_table.system_clk = SYSTEM_SYS_CLK_48M;表示 CPU 运行频率是 48MHz。

另外在程序运行过程中要动态的切换 CPU 的运行频率，可以定义一个如下的函数进行。

```
void user_set_cpu_clk(uint8_t clk)
{
    if(clk == 12)
        __jump_table.system_clk = SYSTEM_SYS_CLK_12M;
    else if (clk == 24)
        __jump_table.system_clk = SYSTEM_SYS_CLK_24M;
```

```
    else if (clk == 48)
        __jump_table.system_clk = SYSTEM_SYS_CLK_48M;
    system_set_pclk(__jump_table.system_clk);
}
```

5.8. 如何在项目中配置和使用充电功能

A：用户可以在入口函数 void user_entry_before_ble_init(void)内部调用如下示例代码，开启充电功能和中断。

```
void user_entry_before_ble_init(void)
{
    /* set system power supply in BUCK mode */
    pmu_set_sys_power_mode(PMU_SYS_POW_BUCK);
    //enalbe charge isr
    pmu_enable_irq(PMU_ISR_BIT_ACOK
                   | PMU_ISR_BIT_ACOFF
                   | PMU_ISR_BIT_BAT);
    NVIC_EnableIRQ(PMU_IRQn);
    //enable charge and set current and terminal voltage
    pmu_enable_charge(CHG_CUR_113MA, CHG_VOL_4_20V);

    system_set_port_pull(GPIO_PC4, true);
    system_set_port_mux(GPIO_PORT_C, GPIO_BIT_4, PORTC4_FUNC_UART1_RXD);
    system_set_port_mux(GPIO_PORT_C, GPIO_BIT_5, PORTC5_FUNC_UART1_TXD);
    uart_init(UART1, BAUD_RATE_921600);

    ool_write(PMU_REG_ADKEY_ALDO_CTRL, ool_read(PMU_REG_ADKEY_ALDO_CTRL) & (~(1<<3)));
}
```

同时重定义充电中断函数如下来获取充电事件：

```
__attribute__((section("ram_code"))) void charge_isr_ram(uint8_t type)
{
    if(type == 2)   //charge full isr
    {
        co_printf("charge full\r\n");
        pmu_disable_irq(PMU_ISR_BIT_BAT);
        pmu_enable_irq(PMU_ISR_BIT_ACOFF);
    }
    else if(type == 1)  //charge plug out
    {
        pmu_disable_irq(PMU_ISR_BIT_BAT);
        pmu_enable_irq(PMU_ISR_BIT_ACOK);
        co_printf("charge out\r\n");
    }
    else if(type == 0)  //charge plug in
    {
        pmu_disable_irq(PMU_ISR_BIT_ACOK);
        pmu_enable_irq(PMU_ISR_BIT_ACOFF|PMU_ISR_BIT_BAT);
        co_printf("charge in\r\n");
    }
}
```

5.9. 做 IEC 标准的 ESD 测试时，产品设计有哪些注意事项

A: 用户使用 Fr801xH 系列芯片，如果需要通过类似 IEC 标准的 ESD 测试，做产品设计时需要注意如下事项。

1. 蓝牙芯片的 pin16 VBAT 脚放置双向 3.3V TVS 管，参考型号：PESDNC2FD3V3B
2. 蓝牙芯片的 pin5 RF 脚放置双向 3.3V 低 Cj 值的 TVS 管，参考型号：PESDUC2FD3V3B
3. 蓝牙芯片的 pin14 reset 脚按照参考电路放置 1K 下拉电阻。

4. 模组需加上屏蔽罩。
5. 模组底部不要放置测试点，通过边上的邮票孔焊盘进行烧录测试。
6. 模组在主板上的安装位置应注意避空，应尽量避开产品外壳的金属。
7. 模组与主板 MCU 的通信之间串联 100R 电阻。
8. 主板上也需要做 ESD 防护。
9. 软件打开 watchdog。

6. 认证射频测试

6.1. 非信令模式测试，可以测试发射，不能测试接收(测试软件可以找支持人员获取)

命令串口：PA2(RX)/PA3(TX),波特率 115200，返回值：OK，命令如下：

AT#TAaa_bb_cc 测试带调制波
AT#TBaa_bb_cc 测试空载波
AT#TC 退出测试
(AT 指令以回车键结尾，否则指令不识别)

aa 是信道,0x00--2402,0x01--2404，、、、以此类推
bb 是 payload，一般配置 00 就好
cc 是发射功率，0x00~0x3f（过认证时注意把功率控制在 10dBm 内）

更改 channel，payload，需要发送 AT#TC 退出，再发送相应命令

例：
空载波 2402M 定频发射：AT#TB00_00_15
空载波 2440M 定频发射：AT#TB13_00_15
空载波 2480M 定频发射：AT#TB27_00_15

6.2. 信令模式，可以测试发射、接收，通常和综测仪连接测试

命令串口：PC6(RX)/PC7(TX),波特率 115200，常用命令如下：
read bdaddr:
01, 09, 10, 00

reset:
01, 03, 0c, 00

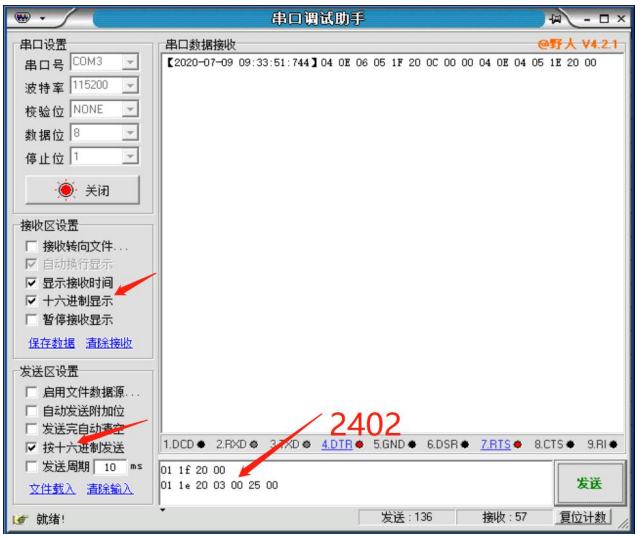
tx power:
01, 1f, 20, 00,
01, 1f, 20, 00,
01, 1e, 20, 03, 00, 25, 00,
01, 1f, 20, 00,

01, 1e, 20, 03, 13, 25, 00,
01, 1f, 20, 00,
01, 1e, 20, 03, 27, 25, 00,
01, 1f, 20, 00

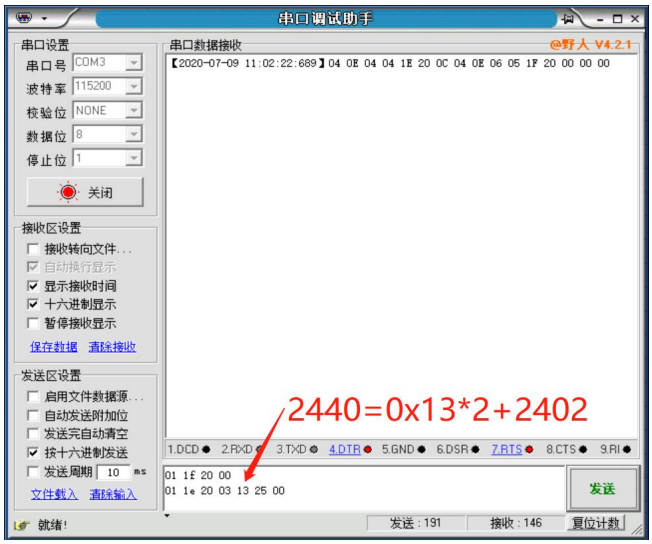
carrier drift:
01, 1f, 20, 00,
01, 1f, 20, 00,
01, 1e, 20, 03, 00, 25, 02,
01, 1f, 20, 00,
01, 1e, 20, 03, 13, 25, 02,
01, 1f, 20, 00,
01, 1e, 20, 03, 27, 25, 02,
01, 1f, 20, 00,

modulation index:
01, 1f, 20, 00,
01, 1f, 20, 00,
01, 1e, 20, 03, 00, 25, 01,
01, 1f, 20, 00,
01, 1e, 20, 03, 00, 25, 02,
01, 1f, 20, 00,
01, 1e, 20, 03, 13, 25, 01,
01, 1f, 20, 00,
01, 1e, 20, 03, 13, 25, 02,
01, 1f, 20, 00,
01, 1e, 20, 03, 27, 25, 01,
01, 1f, 20, 00,
01, 1e, 20, 03, 27, 25, 02,
01, 1f, 20, 00,

sensitivity:
01, 1f, 20, 00,
01, 1f, 20, 00,
01, 1d, 20, 01, 00,
01, 1f, 20, 00,
01, 1d, 20, 01, 13,
01, 1f, 20, 00,
01, 1d, 20, 01, 27,
01, 1f, 20, 00,
实例说明：
01 1f 20 00 统一是结束指令，每次命令前都要先发结束，再发对应命令。
如测试发射功率，频点 2402

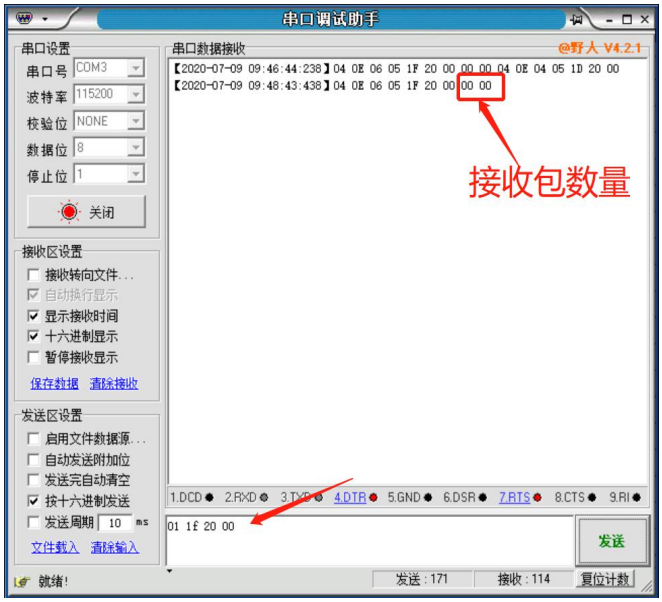


频点 2440



如测试接收，频点 2402

接收模式测试仪器发包完成之后，设备端发送结束指令，从返回中可以看到接收的数据包数量


串口调试助手
串口设置
串口号 COM3
波特率 115200
校验位 NONE
数据位 8
停止位 1
关闭
接收区设置
接收转向文件...
自动换行显示
显示接收时间
十六进制显示
暂停接收显示
保存数据 清除接收
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
发送周期 10 ms
文件载入 清除输入
串口数据接收
@野人 V4.2.1
【2020-07-09 09:46:44:238】04 0E 06 05 1F 20 00 00 00 04 0E 04 05 1D 20 00
【2020-07-09 09:48:43:438】04 0E 06 05 1F 20 00 00 00
接收包数量
1.DCD 2.RXD 3.TXD 4.DTR 5.GND 6.DSR 7.RTS 8.CTS 9.RI
01 1f 20 00
发送
就绪!
发送 : 171
接收 : 114
复位计数

7. 版本历史

Version	Date	Author	Description
0.1	2020-08-05	Dong Youcai	Draft
0.2	2020-11-23	HuangWentao	Update