CMT453x API 参考手册

简介

本文档包含CMT453x蓝牙SDK的完整API参考手册，旨在让开发者能够快速熟悉CMT453x系列蓝牙SoC芯片的API函数的使用方式，以减少开发前期的准备时间，降低开发难度。

免责声明

深圳市华普微电子股份有限公司保留在不另行通知的情况下，更改产品以提升其可靠性、功能或设计的权利。本公司亦不承担因使用此处所述产品或电路而引致的任何责任。

关于涉及生命维持设备的应用

深圳市华普微电子股份有限公司的产品并不适用于生命维持设备、装置或系统，因为这些产品的故障可能会导致人身伤害。使用或销售本产品作上述用途的客户须自行承担风险，并同意就因使用或销售不当而引致的任何损害，向本公司作出全面赔偿。

联系方式

深圳市华普微电子股份有限公司

地址：深圳市南山区西丽街道万科云城三期8栋A座30层

电话：+86-0755-82973805

邮箱：sales@hoperf.com

网址：http://www.hoperf.cn

目录

1 蓝牙应用模块：hp_ble.h

API目录：middlewares\HopeRF\ble_library\hp_library\ble 源码文件：

hp_ble.c, hp_ble.h, hp_ble_task.c, hp_ble_task.h

简介：蓝牙应用相关API，沟通用户的应用代码和蓝牙协议栈。

1.1 hp_ble_stack_init

功能：

蓝牙协议栈初始化，注册蓝牙消息回调函数和用户自定义消息回调函数。

在所传入的参数结构体struct hp_stack_cfg_t里，ble_msg_handler为蓝牙消息回调函数，可处理的消息为enum app_ble_msg所声明。user_msg_handler为用户消息回调函数，用户可以在声明主机的消息，注意消息序号从APP_FREE_EVE_FOR_USER开始，具体参考enum user_msg_id， 用户可以通过函数 ke_msg_send_basic立即挂起一个消息，或者函数ke_timer_set定时挂起一个消息，如需循环定时消息可以在消息处理函数中再次定时挂起消息。

语法：

```
void hp_ble_stack_init(struct hp_stack_cfg_t const* p_init)
```

参数：

[in] p_init：蓝牙应用回调函数配置，详情参考struct hp_stack_cfg_t定义。

返回：

无

示例：

```
struct hp_stack_cfg_t app_handler = {0};
app_handler.ble_msg_handler  = app_ble_msg_handler;
app_handler.user_msg_handler = app_user_msg_handler;
app_handler.lsc_cfg          = BLE_LSC_LSI_32000HZ;
//initialization ble stack
hp_ble_stack_init(&app_handler);
```

用户自定义消息回调处理实现：

```
void app_user_msg_handler(ke_msg_id_t const msgid, void const *p_param)
```

```
{
    switch (msgid)
    {
        case APP_CUSTS_TEST_EVT:
            app_usart_tx_process();
            break;
        default:
            break;
    }
}
```

蓝牙消息回调实现：

```
void app_ble_msg_handler(struct ble_msg_t const *p_ble_msg)
{
    switch (p_ble_msg->msg_id)
    {
        case APP_BLE_OS_READY:
            HP_LOG_INFO("APP_BLE_OS_READY\r\n");
            break;
        case APP_BLE_GAP_CONNECTED:
            app_rdtsc_enable_prf(app_env.conidx);
            app_ble_connected();
            break;
        case APP_BLE_GAP_DISCONNECTED:
            app_ble_disconnected();
            break;
        default:
            break;
    }
}
```

1.2 hp_ble_gap_init

功能：

蓝牙通用参数配置，比如蓝牙mac地址，名称，角色，连接参数等，详情参考结构体struct hp_gap_params_t的定义。

语法：

```
void hp_ble_gap_init(struct hp_gap_params_t const* p_init)
```

参数：

[in] `p_init`：蓝牙通用参数结构体指针。

返回：

无

示例：

```
struct hp_gap_params_t dev_info = {0};
uint8_t *p_mac = SystemGetMacAddr();
//get UUID from trim stored
if(p_mac != NULL)
{
    //set the uuid as mac address
    memcpy(dev_info.mac_addr.addr, p_mac , BD_ADDR_LEN);
}
else{
    memcpy(dev_info.mac_addr.addr, "\x01\x02\x03\x04\x05\x06" , BD_ADDR_LEN);
}

/* init params*/
dev_info.mac_addr_type = GAPM_STATIC_ADDR;
dev_info.appearance = 0;
dev_info.dev_role = GAP_ROLE_PERIPHERAL;

dev_info.dev_name_len = sizeof(CUSTOM_DEVICE_NAME)-1;
memcpy(dev_info.dev_name, CUSTOM_DEVICE_NAME, dev_info.dev_name_len);

dev_info.dev_conn_param.intv_min = MSECS_TO_UNIT(MIN_CONN_INTERVAL,MSECS_UNIT_1_25_MS);
dev_info.dev_conn_param.intv_max = MSECS_TO_UNIT(MAX_CONN_INTERVAL,MSECS_UNIT_1_25_MS);
dev_info.dev_conn_param.latency  = SLAVE_LATENCY;
dev_info.dev_conn_param.time_out = MSECS_TO_UNIT(CONN_SUP_TIMEOUT,MSECS_UNIT_10_MS);
dev_info.conn_param_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;

hp_ble_gap_init(&dev_info);
```

1.3 hp_ble_add_prf_func_register

功能：

注册服务（profile）添加函数，后续系统将调用注册好的服务添加函数，添加相应的服务。

语法：

```
bool hp_ble_add_prf_func_register(hp_ble_add_prf_func_t func)
```

参数：

[in] func：添加服务（profile）函数，函数实现方式参考例程函数app_dis_add_dis。

返回：

true：注册成功。false：注册失败。

示例：

```
hp_ble_add_prf_func_register(app_dis_add_dis);
```

1.4 hp_ble_prf_task_register

功能：

注册服务（profile）的子任务事件到蓝牙应用事件回调列表。

语法：

```
bool hp_ble_prf_task_register(struct prf_task_t *prf)
```

参数：

[in] prf：服务（profile）子任务结构体指针，详情参考结构体prf_task_t。

返回：

true：注册成功。false：注册失败。

示例：

```
//register application subtask to app task
struct prf_task_t prf;
prf.prf_task_id = TASK_ID_RDTSC;
prf.prf_task_handler = &app_rdtsc_handlers;
hp_ble_prf_task_register(&prf);
```

1.5 prf_get_itf_func_register

功能：

注册服务（profile）任务接口函数的获取函数。注意此函数在prf.c文件实现，在prf.h文件中声明。

语法：

```
#include "prf.h"
bool prf_get_itf_func_register(struct prf_get_func_t *prf);
```

参数：

[in] prf：服务任务接口获取函数的指针。

返回：

true：注册成功。 false：注册失败。

示例：

```
//register get itf function to prf.c
struct prf_get_func_t get_func;
get_func.task_id = TASK_ID_DISS;
get_func.prf_itf_get_func = diss_prf_itf_get;
prf_get_itf_func_register(&get_func);
```

1.6 hp_ble_adv_init

功能：

初始化ble蓝牙广播参数并初始化。

语法：

```
void hp_ble_adv_init(struct hp_adv_params_t const* p_init)
```

参数：

[in] p_init：广播初始化参数结构体指针。

返回：

无

示例：

```
struct hp_adv_params_t user_adv = {0};

//init advertising data
user_adv.adv_data_len = CUSTOM_USER_ADVERTISE_DATA_LEN;
memcpy(user_adv.adv_data,CUSTOM_USER_ADVERTISE_DATA,CUSTOM_USER_ADVERTISE_DATA_LEN);
user_adv.scan_rsp_data_len = CUSTOM_USER_ADV_SCNRSP_DATA_LEN;
memcpy(user_adv.scan_rsp_data,CUSTOM_USER_ADV_SCNRSP_DATA,CUSTOM_USER_ADV_SCNRSP_DATA_LEN);

user_adv.attach_appearance  = false;
user_adv.attach_name        = true;
user_adv.ex_adv_enable      = false;
user_adv.adv_phy            = PHY_1MBPS_VALUE;
```

```
//init advertising params
user_adv.directed_adv.enable = false;

user_adv.fast_adv.enable    = true;
user_adv.fast_adv.duration  = CUSTOM_ADV_FAST_DURATION;
user_adv.fast_adv.adv_intv  = CUSTOM_ADV_FAST_INTERVAL;

user_adv.slow_adv.enable    = true;
user_adv.slow_adv.duration  = CUSTOM_ADV_SLOW_DURATION;
user_adv.slow_adv.adv_intv  = CUSTOM_ADV_SLOW_INTERVAL;

user_adv.ble_adv_msg_handler = app_ble_adv_msg_handler;

hp_ble_adv_init(&user_adv);
```

1.7 hp_ble_adv_start

功能：

开启蓝牙广播。

语法：

```
void hp_ble_adv_start(void)
```

参数：

无

返回：

无

示例：

```
//start adv
hp_ble_adv_start();
```

1.8 hp_ble_adv_stop

功能：

停止蓝牙广播。

语法：

```
void hp_ble_adv_stop(void)
```

参数：

无

返回：

无

示例：

```
// Stop Advertising
hp_ble_adv_stop();
```

1.9 hp_ble_adv_data_set

功能：

设置广播数据包的内容。

语法：

```
void hp_ble_adv_data_set(uint8_t* p_dat, uint16_t len)
```

参数：

[in] p_dat：设置的广播包数据。[in] len：设置的广播包数据的长度。

返回：

无

示例：

```
hp_ble_adv_data_set(CUSTOM_USER_ADVERTISE_DATA,CUSTOM_USER_ADVERTISE_DATA_LEN);
```

1.10 hp_ble_scan_rsp_data_set

功能：

设置广播扫描回应数据包的内容。

语法：

```
void hp_ble_scan_rsp_data_set(uint8_t* p_dat, uint16_t len)
```

参数：

[in] p_dat：设置的广播扫描回应包数据。[in] len：设置的广播扫描回应包数据的长度。

返回：

无

示例：

```
hp_ble_scan_rsp_data_set(CUSTOM_USER_ADV_SCNRSP_DATA,CUSTOM_USER_ADV_SCNRSP_DATA_LEN);
```

1.11 hp_ble_ex_adv_data_set

功能：

设置扩展广播数据包的内容。

语法：

```
void hp_ble_ex_adv_data_set(uint8_t* p_dat, uint16_t len)
```

参数：

[in] p_dat：设置扩展广播包数据，注意这里不可以使用局部变量的指针。[in] len：设置扩展广播包数据的长度。

返回：

无

示例：

```
const static uint8_t ex_adv[] = {"\x29\xff""12345678901234567890123456789012345678901234567890"};
hp_ble_ex_adv_data_set((uint8_t*) ex_adv, sizeof(ex_adv)-1);
```

1.12 hp_ble_scan_init

功能：

初始化蓝牙BLE扫描功能的参数。

语法：

```
void hp_ble_scan_init(struct hp_scan_params_t *p_init)
```

参数：

[in]p_init：扫描功能参数结构体指针，详细参考结构体hp_scan_params_t内部定义。

返回：

无

示例：

```
struct hp_scan_params_t init = {0};
static const uint8_t target_name[] = {"HP_RDTSS"};

init.type               = SCAN_PARAM_TYPE;
init.dup_filt_pol       = SCAN_PARAM_DUP_FILT_POL;
init.connect_enable     = SCAN_PARAM_CONNECT_EN;
init.prop_active_enable = SCAN_PARAM_PROP_ACTIVE;
init.scan_intv          = SCAN_PARAM_INTV;
init.scan_wd            = SCAN_PARAM_WD;
init.duration           = SCAN_PARAM_DURATION;

init.filter_type    = SCAN_FILTER_BY_NAME;//SCAN_FILTER_BY_ADDRESS;
init.filter_data    = (uint8_t*)&target_name;

init.ble_scan_data_handler  = app_ble_scan_data_handler;
init.ble_scan_state_handler = app_ble_scan_state_handler;

hp_ble_scan_init(&init);
```

1.13 hp_ble_start_scan

功能：

开启蓝牙BLE扫描功能。

语法：

```
void hp_ble_start_scan(void)
```

参数：

无

返回：

无

示例：

```
hp_ble_start_scan();
```

1.14 hp_ble_stop_scan

功能：

停止蓝牙BLE扫描功能。

语法：

```
void hp_ble_ stop_scan(void);
```

参数：

无

返回：

无

示例：

```
hp_ble_stop_scan();
```

1.15 hp_ble_start_init

功能：

主机发起蓝牙BLE连接。注意仅供主机设备调用。

语法：

```
void hp_ble_start_init(uint8_t *addr, uint8_t addr_type)
```

参数：

[in] addr：将连接的从设备地址。[in] addr_type：将连接的设备地址类型，扫描返回的信息包含地址类型。

返回：

无

示例：

```
hp_ble_start_init("\x11\x11\x11\x11\x11\x11",GAPM_STATIC_ADDR);
```

1.16 hp_ble_update_param

功能：

主机或从机主动发起蓝牙BLE连接参数更新请求。

语法：

```
bool hp_ble_update_param(struct gapc_conn_param *p_conn_param)
```

参数：

[in] conn_param：连接参数结构体指针。

返回：

无

示例：

```
struct gapc_conn_param conn_param;
conn_param.intv_min = 12;                    // 12*1.25 = 15ms
conn_param.intv_max = 12;                    // 12*1.25 = 15ms
conn_param.latency = 5;
conn_param.time_out = 100;                   // 1000ms
hp_ble_update_param(&conn_param);
```

1.17 hp_ble_mtu_set

功能：

主机或从机主动发起蓝牙BLE mtu参数更新请求。应注意用户数据包有效数据长度比MTU小3字节。

语法：

```
void hp_ble_mtu_set(uint16_t mtu)
```

参数：

[in] mtu：蓝牙BLE mtu值，最大值为517。

返回：

无

示例：

```
hp_ble_mtu_set(247);          //set mtu as 247
```

1.18 hp_ble_phy_set

功能：

主机或从机主动发起蓝牙BLE phy参数更新请求。

语法：

```
void hp_ble_phy_set(enum gap_phy_val phy)
```

参数：

[in] phy：蓝牙BLE PHY参数值，可选参数参考enum gap_phy_val声明。

返回：

无

示例：

```
hp_ble_phy_set(GAP_PHY_125KBPS);
```

1.19 hp_ble_active_rssi

功能：

主机或从机主动发起蓝牙BLE rssi读取请求。读取的值通过蓝牙事件回调函数的APP_BLE_GAP_RSSI_IND消息返回。

语法：

```
void hp_ble_active_rssi(uint32_t interval)
```

参数：

[in] interval：蓝牙BLE rssi读取间隙，时间单位毫秒，interval为0时，表示只读取一次。

返回：

无

示例：

```
hp_ble_active_rssi(500);
```

1.20 hp_ble_disconnect

功能：

主机或从机主动发起蓝牙BLE断开连接请求。成功断开连接会通过蓝牙事件回调函数的APP_BLE_GAP_DISCONNECTED消息返回。

语法：

```
void hp_ble_disconnect(void)
```

参数：

无

返回：

无

示例：

```
hp_ble_disconnect();
```

1.21 hp_ble_dle_set

功能：

主机或从机请求设置DLE (Data Length Extension)参数。如需变更，建议使用示例

代码的参数。

语法：

```
void hp_ble_dle_set(uint16_t tx_octets, uint16_t tx_time)
```

参数：

[in] `tx_octets`：单个链路数据通道PDU的最大数据量。[in] `tx_time`：单个链路数据通道PDU的发送时间最大微秒数。

返回：

无

示例：

```
hp_ble_dle_set(251, 2120); //suggest parameter
```

1.22 rf_tx_power_set

功能：

主机或从机设置射频信号发射功率。

语法：

```
void rf_tx_power_set(rf_tx_power_t pwr)
```

参数：

[in] pwr：设置的射频信号发射功率，可选参数如下。

```
typedef enum
{
    TX_POWER_0_DBM = 0, /*  0 dBm */
    TX_POWER_Neg2_DBM,  /* -2 dBm */
    TX_POWER_Neg4_DBM,  /* -4 dBm */
    TX_POWER_Neg8_DBM,  /* -8 dBm */
    TX_POWER_Neg15_DBM, /* -12 dBm */
    TX_POWER_Neg20_DBM, /* -20 dBm */
    TX_POWER_Pos2_DBM,  /* +2 dBm */
    TX_POWER_Pos3_DBM,  /* +3 dBm */
    TX_POWER_Pos4_DBM,  /* +4 dBm */
    TX_POWER_Pos6_DBM,  /* +6 dBm */
}rf_tx_power_t;
```

返回：

无

示例：

```
rf_tx_power_set(TX_POWER_Pos4_DBM);
```

1.23 过程函数

函数：

```
void hp_ble_adv_fsm_next(void);
```

功能：

从机设备的广播状态管理函数。用户代码不需要单独调用。

函数：

```
void hp_ble_create_scan(void);
```

功能：

主机创建扫描activity，正确调用函数`hp_ble_scan_init`初始化扫描模块，将自动创建，用户代码不需要单独调用。

函数：

```
void hp_ble_delete_scan(void);
```

功能：

主机删除扫描activity，用户代码不需要单独调用。

函数：

```
void hp_ble_create_init(void);
```

功能：

主机创建主机连接activity，将自动创建，用户代码不需要单独调用。

函数：

```
void hp_ble_delete_init(void);
```

功能：

主机删除连接activity，用户代码不需要单独调用。

函数：

```
bool hp_ble_scan_data_find(uint8_t types, const uint8_t *p_filter_data, uint8_t *p_data, uint8_t
len);
```

功能：

主机在扫描返回的广播数据包中查找指定数据，正确调用函数hp_ble_scan_init初始化扫描模块，将自动基于过滤配置调用，用户代码不需要单独调用。

函数：

```
bool hp_ble_add_svc(void);
```

功能：

初始化过程中的，根据已注册的profile添加服务。用户代码不需要单独调用。

2 蓝牙安全加密模块：hp_sec.h

API目录：middlewares\HopeRF\ble_library\hp_library\sec

源码文件：hp_sec.c, hp_sec.h

简介：蓝牙安全加密应用相关API，应用代码调用相关API来进行安全相关设置。

2.1 hp_sec_init

功能：

初始化蓝牙安全加密模块。

语法：

```
void hp_sec_init(struct hp_sec_init_t const* init)
```

参数：

[in] init：蓝牙安全模块初始化参数结构体指针。详情参考结构体hp_sec_init_t内部定义。

返回：

无

示例：

```
struct hp_sec_init_t sec_init = {0};

sec_init.rand_pin_enable = false;
sec_init.pin_code = 123456;

sec_init.pairing_feat.auth      = ( SEC_PARAM_BOND | (SEC_PARAM_MITM<<2) | (SEC_PARAM_LESC<<3) |
(SEC_PARAM_KEYPRESS<<4) );
sec_init.pairing_feat.iocap     = SEC_PARAM_IO_CAPABILITIES;
sec_init.pairing_feat.key_size  = SEC_PARAM_KEY_SIZE;
sec_init.pairing_feat.oob       = SEC_PARAM_OOB;
sec_init.pairing_feat.ikey_dist = SEC_PARAM_IKEY;
sec_init.pairing_feat.rkey_dist = SEC_PARAM_RKEY;
sec_init.pairing_feat.sec_req   = SEC_PARAM_SEC_MODE_LEVEL;

sec_init.bond_enable            = BOND_STORE_ENABLE;
```

```
sec_init.bond_db_addr           = BOND_DATA_BASE_ADDR;
sec_init.bond_max_peer          = MAX_BOND_PEER;
sec_init.bond_sync_delay        = 2000;

sec_init.hp_sec_msg_handler     = NULL;

hp_sec_init(&sec_init);
```

2.2 hp_sec_get_bond_status

功能：

返回是否已绑定的状态。

语法：

```
bool hp_sec_get_bond_status(void)
```

参数：

无

返回：

ture：已绑定。false：未绑定。

示例：

```
bool bond_status = hp_sec_get_bond_status();
```

2.3 hp_sec_get_iocap

功能：

返回设备的接口能力参数，供蓝牙相关库调用，一般用户代码不需要调用。

语法：

```
uint8_t hp_sec_get_iocap(void);
```

参数：

无

返回：

设备的接口能力参数，是函数hp_sec_init初始化设置进去的值。

示例：

```
uint8_t io_cap = hp_sec_get_iocap();
```

2.4 hp_sec_bond_db_erase_all

功能：

擦除所有已绑定的设备。因为擦除flash会阻塞运行，建议不要在已连接状态下执行，必须影响连接的可能性。

语法：

```
void hp_sec_bond_db_erase_all(void);
```

参数：

无

返回：

无

示例：

```
hp_sec_bond_db_erase_all();
```

2.5 hp_sec_send_security_req

功能：

发送安全请求到主机设备。（一般用户代码不需要调用。）

2.6 hp_sec_bond_store_evt_handler

功能：

已绑定设备信息数据存储任务回调函数，内部使用。

3 软件定时器：hp_timer.h

API目录：middlewares\HopeRF\ble_library\hp_library\timer

源码文件：hp_timer.c, hp_timer.h

简介：软件定时器模块，由蓝牙协议栈内部库ke_timer模块封装，需要在协议栈初始化好后，即APP_BLE_OS_READY消息发出之后才能使用。由于睡眠唤醒之后，必须超过一个低速时钟拍（约32us）才能调用ke_timer_ser。

3.1 hp_timer_create

功能：

创建一个软件定时器，在延迟delay毫秒数后调用fn这个函数。如需循环定时器可以在回调处理处再次创建一个定时器。注意需要在协议栈初始化好后，即APP_BLE_OS_READY消息发出之后才能使用。

语法：

```
timer_hnd_t hp_timer_create(const uint32_t delay, timer_callback_t fn)
```

参数：

[in] delay：延迟的时间，单位为1毫秒。[in] fn：延迟delay毫秒数的回调函数。

返回：

定时器的id

示例：

```
timer_hnd_t timer_id = hp_timer_create(1000, my_timer_callback);
```

3.2 hp_timer_modify

功能：

修改指定一个定时器的延迟时间。

语法：

```
timer_hnd_t hp_timer_modify(const timer_hnd_t timer_id, uint32_t delay)
```

参数：

[in] `timer_id`：需要修改的定时器`id`。[in] `Delay`：修改后的延迟时间，单位为`1`毫秒。

返回：

修改后的定时器id。

示例：

```
timer_id = hp_timer_modify(timer_id, 2000);
```

3.3 hp_timer_cancel

功能：

取消指定一个定时器。

语法：

```
void hp_timer_cancel(const timer_hnd_t timer_id)
```

参数：

[in] `timer_id`：需要取消的定时器id。

返回：

无

示例：

```
hp_timer_cancel(timer_id);
```

3.4 hp_timer_cancel_all

功能：

取消所有已创建的定时器。

语法：

```
void hp_timer_cancel_all(void)
```

参数：

无

返回：

无

示例：

```
hp_timer_cancel_all();
```

4 硬延迟模块：hp_delay.h

API目录：middlewares\HopeRF\ble_library\hp_library\delay

源码文件： hp_delay.h

简介：硬延迟函数模块，里面声明的相关函数具体实现在ROM代码，这里只需声明，用户代码添加头文件之后可以直接调用。应注意其延迟的时间并不精准，如需精准延迟建议使用硬件定时器功能。

4.1 delay_cycles

功能：

延迟等待指定数量的cycles返回。

语法：

```
void delay_cycles(uint32_t ui32Cycles);
```

参数：

[in] ui32Cycles：需要等待的cycles数量，一个cycles的时间约为(10/110) 微秒。

返回：

无

示例：

```
delay_cycles(1000);
```

4.2 delay_n_us

功能：

延迟等待指定数量的微秒。

语法：

```
void delay_n_us(uint32_t val);
```

参数：

[in] val：需要等待的微秒数。

返回：

无

示例：

```
delay_n_us(1000);
```

4.3 delay_n_10us

功能：

延迟等待指定数量的10微秒。

语法：

```
void delay_n_10us(uint32_t val);
```

参数：

[in] val：需要等待的10微秒数。

返回：

无

示例：

```
delay_n_10us(1000);
```

4.4 delay_n_100us

功能：

延迟等待指定数量的100微秒。

语法：

```
void delay_n_100us(uint32_t val);
```

参数：

[in] val：需要等待的100微秒数。

返回：

无

示例：

```
delay_n_100us(1000);
```

4.5 delay_n_ms

功能：

延迟等待指定数量的毫秒。

语法：

```
void delay_n_ms(uint32_t val);
```

参数：

[in] val：需要等待的毫秒数。

返回：

无

示例：

```
delay_n_ms(1000);
```

5 蓝牙睡眠模块：hp_sleep.h

API目录：middlewares\HopeRF\ble_library\hp_library\sleep

源码文件：hp_sleep.c, hp_sleep.h

简介：蓝牙协议栈睡眠函数模块。

5.1 hp_sleep

功能：

蓝牙睡眠功能进入函数，函数会基于蓝牙协议栈工作状态，如果没有需要执行的任务则自动进入睡眠状态，在定时任务或者硬件中断唤醒后恢复系统状态。一般调用于main函数主循环，调度函数rwip_schedule()后面。

语法：

```
void hp_sleep(void)
```

参数：

无

返回：

无

示例：

```
hp_sleep();
```

5.2 hp_sleep_lock_acquire

功能：

蓝牙睡眠模式锁请求，即申请系统不进入睡眠模式。比如开启某些高速外设（如串口）不希望系统进入睡眠而关掉它，可以通过请求睡眠模式锁禁止系统进入睡眠。

语法：

```
uint8_t hp_sleep_lock_acquire(void)
```

参数：

无

返回：

true：请求锁成功。false：请求锁失败。

示例：

```
if(Cmd == ENABLE)
{
    hp_sleep_lock_acquire();
}
```

5.3 hp_sleep_lock_release

功能：

蓝牙睡眠模式锁释放，当所有的睡眠模式锁释放后，系统在没有任务挂起时将进入睡眠模式。

语法：

```
uint8_t hp_sleep_lock_release(void)
```

参数：

无

返回：

true：释放锁成功。false：释放锁失败，目前没有可以释放的锁。

示例：

```
if(Cmd == DISABLE)
{
    hp_sleep_lock_release();
}
```

5.4 进入和退出睡眠模式的虚函数

函数：

```
__weak void app_sleep_prepare_proc(void)
```

功能：

预设的虚函数，用于用户代码的实现，在进行睡眠模式之前会调用。

函数：

```
__weak void app_sleep_resume_proc(void)
```

功能：

预设的虚函数，用于给用户重新实现睡眠唤醒之后需要做任务，比如可以重新初始化因为睡眠关闭的硬件外设，例如使能GPIO的时钟。

6 调试信息打印模块：log.h

API目录：middlewares\HopeRF\ble_library\hp_library\log

源码文件：hp_log_lpuart.c, hp_log_lpuart.h, hp_log_usart.c, hp_log_usart.h, hp_log.h

简介：调试信息输出函数模块，目前可供选择的输出硬件有LPUART和USART，其中LPUART默认使用 PB1做为其TX脚，USART默认使用USART1的PB6作为其TX脚。

6.1 HP_LOG_INIT

功能：

初始化调试信息打印模块。调用该函数接口会初始化LPUART或USART1。

语法：

```
HP_LOG_INIT();
```

6.2 HP_LOG_DEINIT

功能：

取消初始化调试信息打印模块。

语法：

```
HP_LOG_DEINIT();
```

6.3 调试信息打印输出

功能：

调用不同级别的调试信息打印输出函数。语法使用和print函数相同。不同级别的输出函数通过不同的宏使能控制，HP_LOG_ERROR_ENABLE，HP_LOG_WARNING_ENABLE，HP_LOG_INFO_ENABLE和HP_LOG_DEBUG_ENABLE。

语法：

```
HP_LOG_DEBUG(...)
HP_LOG_INFO(...)
HP_LOG_WARNING(...)
HP_LOG_ERROR(...)
```

6.4 相关使能宏

```
#define HP_LOG_USART_ENABLE                0
#define HP_LOG_LPUART_ENABLE               1

#define HP_LOG_ERROR_ENABLE                1
#define HP_LOG_WARNING_ENABLE              1
#define HP_LOG_INFO_ENABLE                 1
#define HP_LOG_DEBUG_ENABLE                0

#define PRINTF_COLOR_ENABLE                0
```

7 蓝牙程序编写建议

中断处理：

不要在中断服务函数执行太长时间，避免阻碍蓝牙消息处理，建议通过挂起消息，定时器或者标志位的方式在任务或轮询里实现逻辑代码。

不要在中断服务函数调用硬件外设。睡眠后如果中断事件挂起，中断服务函数将会在app_sleep_resume_proc之前调用，外设在唤醒后还没初始化，如在中断服务函数调用硬件外设，则会因为外设未初始化而导致出错。

包含蓝牙的工程中断服务函数注意需要通过ModuleIrqRegister函数注册，并且用户代码的中断优先级只能使用2和3（蓝牙协议栈使用0和1）。

代码执行：

不要连续执行占用时间太长的代码，将会阻碍蓝牙消息处理。建议尽可实现用户代码粹片化，每个任务消息或轮询只占用小片时间。

用户的逻辑代码推荐在任务回调函数里面实现，可以是消息事件回调或者定时任务回调。

进入与退出低功耗模式

注意进入低功耗sleep模式下，高速外设（如USART）将被关闭，需要在唤醒后程序初始化才能使用。

如果使用中断从sleep模式唤醒，注意中断服务函数里面先开启需要操作硬件的时钟，特别是GPIO的时钟。在SLEEP模式下可以保持GPIO的状态，但没有保持时钟的使能。