XR806 BLE Controller 驱动

使用指南

版本号：1.1

发布时间：2020-12-18

版本历史

版本	日期	责任人	版本描述
1.0	2020-12-8	AWA 1679	创建文档。
1.1	2020-12-18	AWA 1679	删除 ble_ctrl_star()和 ble_ctrl_stop()相关内容

目录

图片目录

表格目录

1 前言

1.1 文档简介

本文档介绍了 XR806 BLE Controller 驱动接口函数的功能及使用说明。本文档可以指导阅读者使用 XR806 BLE Controlle 提供的接口对接自身的 host。

1.2 目标读者

想了解 XR806 BLE Controller 驱动接口的用户。

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品。

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
 **说明**	为准确理解文中指令、正确实施操作而提供的补充或强调信息。

2 概述

2.1 背景说明

蓝牙协议规定，Host 和 Controller 的交互需要通过 HCI 协议实现。XR806 BLE Controller 驱动位于 HCI 层，是 BLE Firmware 提供的 API。其主要功能有 BLE 初始化与反初始化、HCI 传输接口初始化、HCI 命令传输。

2.2 规格特性

BLE Controlle 的 LL 规格为 5.0 版本。

2.3 文件位置

XR806 BLE Controller 接口属于 BLE FW 的 API，以 FW 为根目录，其代码位置如表 2-1 所示。

FW 的 API 接口定义在 blec.h，已同步至 SDK 的 appos\include\blec 目录。

表 2-1 XR806 BLE Controller 驱动代码位置

模块	文件类型	代码位置
BLE 驱动接口	header	./appos/include/blec/blec.h

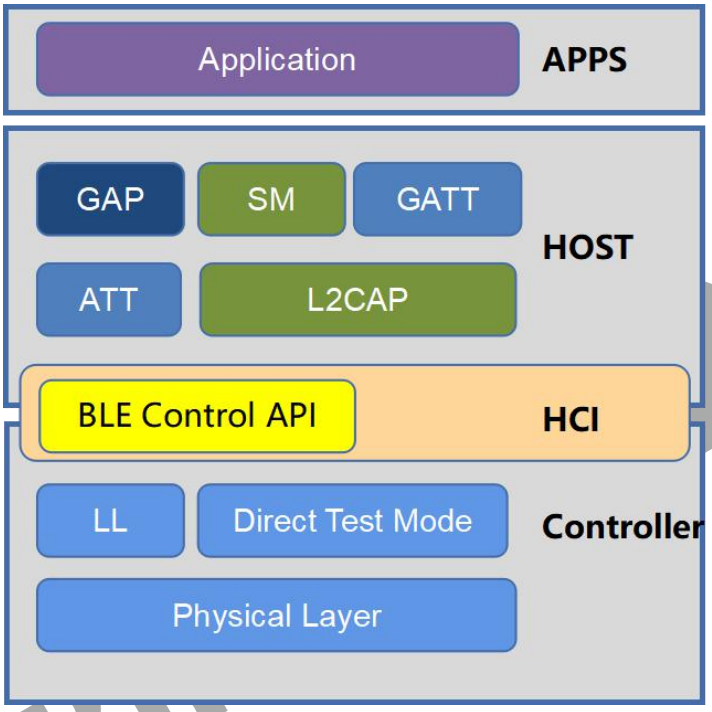
3 技术说明

3.1 模块框架

XR806 BLE Controller 驱动模块的系统层次如下图 3-1 所示。

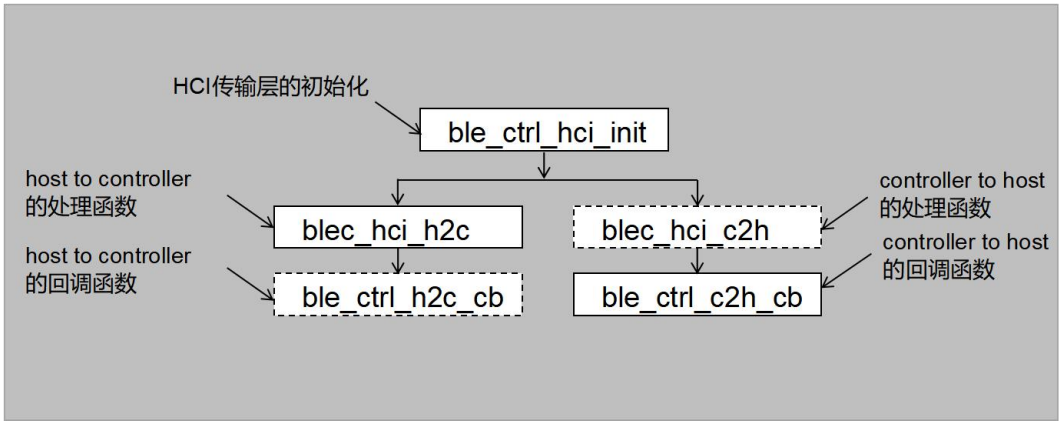
XR806 BLE Controller 驱动模块位于 BLE 系统的 HCI 层，是 BLE Firmware 提供给 Host 的接口。Host 调用通过 BLE Controller API，实现 BLE 功能。

图 3-1 BLE 系统层次



3.2 HCI 传输接口使用流程

图 3-2 HCI 传输接口使用流程



ble_ctrl_hci_init()是 HCI 传输接口的初始化函数，使用图 3-3 中的处理函数和回调函数前必须要先成功对 HCI 传输接口进行初始化。

虚线框的函数是回调函数，它们的注册是在 ble_ctrl_hci_init（）内完成的。

HCI 传输接口的使用分两种情况：

（1）h2c：全称 Host to Controller ,既是 HCI 包从 Host 层传输到 Controller 层。

（2）c2h：全称 Controller to Host，既是 HCI 包从 Controller 层传输到 Host 层。

下面会分别对两种情况的 HCI 传输接口的使用进行使用说明。

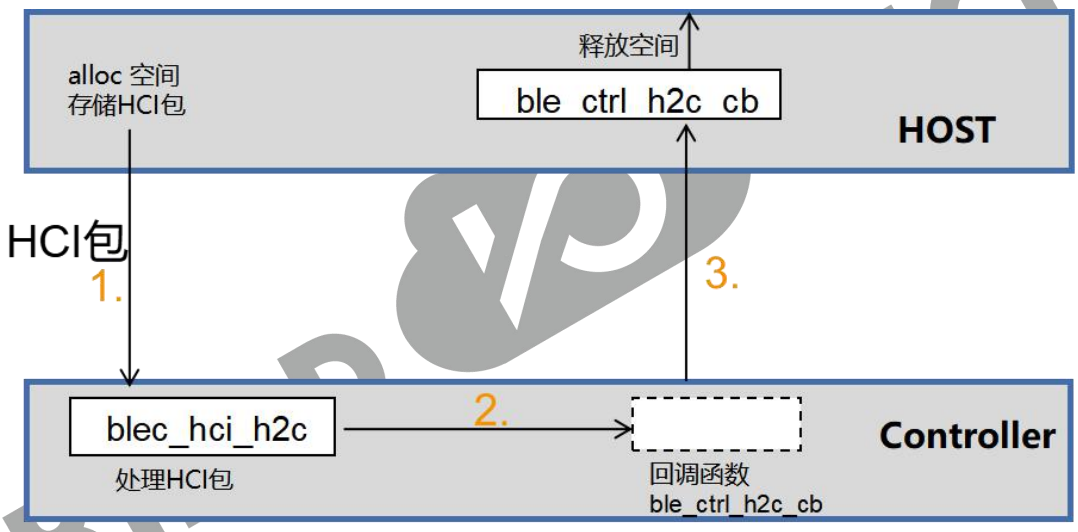
3.2.1 Host to Controller

简单举例 Host 层向 Controller 层传输 HCI 包

(1) Host 通过 blec_hci_h2c 接口把 HCI 包传输给 Controller 进行处理。

(2) Controller 处理完，调用回调函数 blec_hci_h2c_cb()告知 Host 该 HCI 包已使用完，可以进行释放。

图 3-3 Host to Controller 传输 data



3.2.2 Controller to Host

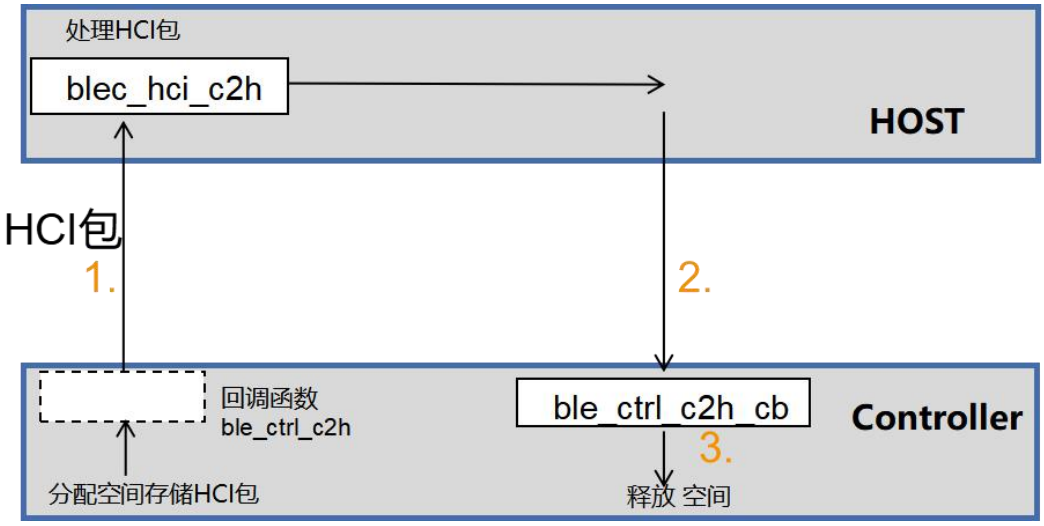
简单举例 Controller 层向 Host 层传输 HCI 包 。

(1)Controller 层 分配空间存放 HCI 包

(2)Controller 调用回调函数 blec_hci_c2h，把 HCI 包传输给 Host 进行处理。

(3)Host 处理完，调用 blec_hci_c2h_cb 告知 Controller 该 HCI 包已使用完，可进行释放

图 3-4 Controller to Host 传输 data

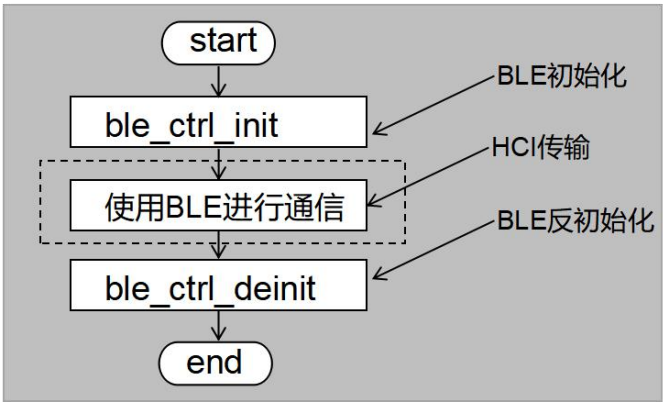


4 应用说明

4.1 应用简述

使用 BLE HCI 传输接口传输 HCI 包前，需要先进行 BLE 初始化。具体的接口流程如图 4-1：

图 4-1 BLE LL 入口模块使用流程



使用 BLE 的 HCI 传输接口的步骤如下：

（1）使用接口 ble_ctrl_init()进行 BLE 初始化，对 BLE 所用到的软硬件资源进行配置。

（2）此时就可以通过 HCI 传输接口进行 HCI 包收发。

（3）需要卸载彻底卸载 ble 功能时，可以使用 ble_ctrl_deinit()。

补充说明：

a. ble 只允许被初始化一次，除非断电重启。

4.2 配置说明

启用 BLE Controller 需要使能 BLE 的相关配置，通过工程配置即可启用，如表 4-1 所示。

表 4-1 BLE Controller 配置列表

配置项	配置说明
BLE Controller 功能使能	设置说明： 此项配置用于在 SDK 中启用 BLE Controller 功能。 设置方式： 在控制台执行命令“make menuconfig”，将 BLE 底下“Ble Controller”功能项打开。

4.3 接口说明

XR806 BLE Controller 驱动总共提供 5 个接口，主要功能点涵括了 BLE 初始化与反初始化、HCI 传输接口初始化、HCI 命令传输。

4.3.1 ble_ctrl_init

表 4-2 ble_ctrl_init 接口函数说明

信息项	说明
原型	int ble_ctrl_init(void);
功能	BLE 的初始化，硬件初始化和资源配置等。
参数	无
返回值	int 类型 0 ： 成功 非 0 ： 失败

4.3.2 ble_ctrl_deinit

表 4-3 ble_ctrl_deinit 接口函数说明

信息项	说明
原型	void ble_ctrl_deinit(void);
功能	BLE 的卸载，硬件复位和资源释放等。
参数	无
返回值	无

4.3.3 ble_ctrl_hci_init

表 4-4 ble_ctrl_hci_init 接口函数说明

信息项	说明
原型	int ble_ctrl_hci_init(blec_hci_t * blec_hci);
功能	HCI 传输接口的初始化
参数	blec_hci 含义解释：用于注册回调函数，初始化 HCI 传输接口 使用说明：blec_hci_t 结构体的说明，请查看表 4-7
返回值	int 类型 0 ： 初始化成功 1 ： 错误，重复调用初始化函数 2 ： 错误，参数 belc_hci 无效，回调函数注册失败

表 4-5 blec_hci_t 结构体的成员说明

成员项	说明
int (* blec_hci_c2h)(unsigned char hci_type, const unsigned char * buff_start, unsigned int buff_offset, unsigned int buff_len);	含义解释：回调函数，blec_hci_c2h 函数功能是把 HCI 包传给 HOST 进行处理。 参数说明：参数定义与 int blec_hci_h2c()一致，参考表 4-6 中的参数说明
int (* blec_hci_h2c_cb)(unsigned char status, const unsigned char * buff_start, unsigned int buff_offset, unsigned int buff_len);	含义解释：回调函数，blec_hci_h2c_cb 函数功能是通知 HOST 该 HCI 包已经处理完成，可进行释放。 参数说明：参数定义与 blec_hci_c2h_cb()一致，参考表 4-7 中的参数说明

4.3.4 blec_hci_h2c

表 4-6 blec_hci_h2c 接口函数说明

信息项	说明
原型	int blec_hci_h2c(unsigned char hci_type, const unsigned char * buff_start, unsigned int buff_offset, unsigned int buff_len);
功能	把 HCI 包传给 Controller 进行处理。。
参数	unsigned char hci_type 含义解释：HCI 层传输类型， 使用说明：不同数值代表不同类型。 0x01：指令, 0x02：数据, 0x04:事件。 const unsigned char * buff_start 含义：BUFF 的起始地址 使用说明：N/A unsigned int buff_offset 含义：HCI 包真正起始地址相对于在 BUFF 起始地址的偏移量 使用说明：HCI 包中不包含 HCI type unsigned int buff_len 含义：HCI 包的长度 使用说明：HCI 包中不包含 HCI type
返回值	int 类型 0 ： 数据读取成功 1 ： 数据读取失败，HCI 层传输接口为被初始化 2 ： 数据读取失败，上一次的数据未读取完。

4.3.5 blec_hci_c2h_cb

表 4-7 blec_hci_c2h_cb 接口函数说明

信息项	说明
原型	int blec_hci_c2h_cb(unsigned char status, const unsigned char * buff_start, unsigned int buff_offset, unsigned int buff_len);
功能	通知 Controller 该 HCI 包已经处理完，可进行释放。
参数	unsigned char status 含义解释：表示 Host 层接收 HCI 包的状态， 使用说明：0:成功 ；非 0：失败 const unsigned char * buff_start 含义：BUFF 的起始地址 使用说明：与对应的 ble_hci_c2h()的 buff_start 一致 unsigned int buff_offset 含义：HCI 包真正起始地址相对于在 BUFF 起始地址的偏移量 使用说明：与对应的 ble_hci_c2h()的 buff_offset 一致 unsigned int buff_len 含义：HCI 包的长度 使用说明：与对应的 ble_hci_c2h()的 buff_len 一致
返回值	int 类型 0 ：成功 1 ：失败，HCI 层传输接口未被初始化 其他：失败

5 示例说明

以 SDK 的 BLE Host virtual_hci 作为例子，讲述 HCI 传输接口的用法。

文件的路径：./appos/src/ble/hci/virtual_hci.c

5.1 ble_ctrl_hci_init 使用示例

5.1.1 示例简介

ble_ctrl_hci_init 分两步：

（1）对 blec_hci_t 类型全局变量 ble_hci_hst 进行初始化

（2）把 ble_hci_hst 作为参数，调用 ble_ctrl_hci_init（）初始化 HCI 传输接口

5.1.2 关键实现

```
/*BLE 初始化*/
ble_ctrl_init();
......
/*初始化 ble_hci_hst*/
static blec_hci_t ble_hci_hst = {
    .blec_hci_c2h = blec_hci_c2h,
    .blec_hci_h2c_cb = blec_hci_h2c_cb,
};
......

void virtual_hci_init1(void)
{
    /*初始化 HCI 传输接口*/
    ble_ctrl_hci_init(&ble_hci_hst);
}
```

5.2 blec_hci_c2h 和 blec_hci_c2h_cb 使用示例

5.2.1 示例简介

blec_hci_c2h（）函数内部会做三件事件：

（1）从 buff 中读取 HCI 包

（2）对 HCI 包进行处理

（3）调用 blec_hci_c2h_cb（）通知 Controller 对 buff 进行释放。

5.2.2 关键实现

```
int blec_hci_c2h(unsigned char hci_type, const unsigned char * buff, unsigned int offset, unsigned int len)
{
    struct net_buf *tmp_buf = NULL;
    uint8_t *origin_buff = (uint8_t *)buff;
    buff += offset;
        /*判断 type ，type 非法返回 1*/
    if ((hci_type != H4_ACL) && (hci_type != H4_EVT))
    {
        printf("hci c2h hci type errr: %d\n", hci_type);
        return 1;
    }
    tmp_buf = get_rx(hci_type, buff);
    ......
    /*拷贝 HCI 包*/
    net_buf_add_mem(tmp_buf, buff, len);
    ......
    /*上传到 HOST*/
    bt_recv(tmp_buf);
    ......
    /*通知 controller 该 HCI 包已处理完，可以进行释放*/
    blec_hci_c2h_cb(0, origin_buff, offset, len);
    return 0;
}
```

5.3 blec_hci_h2c 和 blec_hci_h2c_cb 使用示例

5.3.1 示例简介

5.3.2 关键实现

具体实现，以 int virtual_hci_h2c(struct net_buf *buf)为例

（1）调用接口 blec_hci_h2c 把 HCI 包传给 Controller

（2）等待信号量 tx_sem 释放。(当 Controller 处理完成 HCI 包会调用 blec_hci_h2c_cb 释放信号量 tx_sem 通知 Host。）

（3）释放 HCI 包

```
/*全局变量  tx_sem*/
static OS_Semaphore_t tx_sem;
int blec_hci_h2c_cb(unsigned char status,
                                const unsigned char * buff,
                                unsigned int offset,
                                unsigned int len)
{
    /*释放信号量 tx_sem*/
    OS_SemaphoreRelease(&tx_sem);
    return 0;
}

/*释放 buf*/
void virtual_hci_h2c_cb( int len )
{
    net_buf_unref(tx.buf);
    tx.buf = net_buf_get(&tx.fifo, K_NO_WAIT);
}

int virtual_hci_h2c(struct net_buf *buf)
{
    unsigned char status = 0;
    unsigned char h2c_type = 0xFF;
    uint8_t event_type = bt_buf_get_type(buf);

    ......

    /*调用接口 blec_hci_h2c  让 Controller 处理数据*/
    if ((h2c_type == 1) || (h2c_type == 2))/*type  判断，1：command 2：data*/
    {
      /*调用接口 blec_hci_h2c  把 HCI 包传给 Controller*/
        status = blec_hci_h2c(h2c_type, tx.buf->data, 0, tx.buf->len);

      /*Controller  调用 blec_hci_h2c_cb 告知 Host，HCI 包已经处理完成，可以进行释放*/
        if (OS_SemaphoreWait(&tx_sem, 5000) != OS_OK)/*超时判断*/
            printf("h2c cb timeout\n");

        virtual_hci_h2c_cb(tx.buf->len);
        if (status != 0) /*HCI 包处理状态判断  ，0：成功  非 0：失败*/
        {
            printf("h2c err %d %d!\n", event_type, status);
        }
    } else {
        printf("h2c err h2c_type %d!\n", h2c_type);
    }
```

著作权声明

商标声明

免责声明