



# **HiLink蓝牙厂家适配接口说明**

**V1.0**

**2021.07.08**

## 一、华为实现蓝牙协议栈

### 1 准备编译环境

- (1) 在iotcode/iot\_sdk目录下输入./gradlew sdkfact 编译对应蓝牙 SDK和WIFI SDK的静态/动态链接库 (.a or .so)
- (2) WIFI SDK打开SUPPORT\_BLE宏, 设置BUILD\_SUPPORT\_BLE=1
- (3) 蓝牙 SDK 如需使用speke加密则打开LINKLAYER\_ENCRYPT宏,使用harmonyOS需要打开SUPPORT\_HARMONY宏
- (4) 蓝牙SDK如果需要使用WIFI SDK进行辅助配网, 需要打开SUPPORT\_WIFI宏, 设置BUILD\_SUPPORT\_WIFI=1
- (5) 在厂家的工程中添加文件夹参考以下内容 (以下文件均在**发布件**中能找到)

vendor\hilink				
名称	类型	修改日期	大小	
debug	文件夹	2021/4/29 17:03		
include	文件夹	2021/4/29 17:10		
release	文件夹	2021/4/28 11:24		
hilink_demo.c	C 文件	2021/5/6 17:52	21 KB	
hilink_device.c	C 文件	2021/4/29 17:04	4 KB	
hilink_device_sdk.c	C 文件	2021/4/29 17:04	8 KB	
hilink_ota.c	C 文件	2021/4/20 14:34	6 KB	

  

vendor\hilink\debug				
名称	类型	修改日期	大小	
libhilinkbtsdk.a	A 文件	2021/5/6 17:13	582 KB	
libhilinkdevicesdk.a	A 文件	2021/4/25 13:35	2,775 KB	
libhilinkota.a	A 文件	2021/4/25 13:35	183 KB	

  

vendor\hilink\include				
名称	类型	修改日期	大小	
ble_cfg_net_api.h	H 文件	2021/5/31 10:40	6 KB	
hilink.h	H 文件	2021/5/29 14:13	12 KB	
hilink_bt_api.h	H 文件	2021/5/31 10:17	10 KB	
hilink_device.h	H 文件	2021/5/29 11:36	4 KB	
hilink_log.h	H 文件	2021/5/29 11:36	4 KB	
hilink_netconfig_mode_mgt.h	H 文件	2021/5/29 13:48	3 KB	
hilink_osadapter.h	H 文件	2021/5/29 11:36	9 KB	
hilink_ota.h	H 文件	2021/5/29 11:36	5 KB	
hilink_ota_mcu.h	H 文件	2021/5/29 11:36	10 KB	
hilink_typedef.h	H 文件	2021/5/29 11:36	1 KB	

- (6) 修改厂家编译脚本, 将上述文件添加进编译工程中, 生成可烧录的二进制固件

## 2 厂家适配代码

### 2.1 修改内容 hilink\_demo.c 进行初始化

(1) 此处根据设备不同, 厂家自行修改内容

```
/*
 * 设备基本信息
 * 与hilink_sdk相同定义, 双模组模式只需一份, 已提供给第三方厂家, 暂不按编程规范整改
 */
#define DEVICE_SN "12345678"
#define PRODUCT_ID "2XXX"
#define DEVICE_MODEL "model-X"
#define DEVICE_TYPE "000"
#define MANUFACTURER "000"
#define DEVICE_MAC "AABBCCDDEEFF"

/*
 * 请确保设备类型英文名和厂商英文名长度之和不超过17字节
 * 如果需要发送蓝牙广播, 设备类型英文名和厂商英文名长度分别不能超过4字节
 * 与hilink_sdk相同定义, 双模组模式只需一份, 已提供给第三方厂家, 暂不按编程规范整改
 */
#define DEVICE_TYPE_NAME "other"
#define MANUFACTURER_NAME "HW"
```

此处注意如果需要发送蓝牙广播, 请确保设备类型英文名和厂商英文名长度分别不能超过4字节, 超过则会被截断

(2) 此处获取设备SN

```
/*
 * 功能: 获取设备sn号
 * 参数[in]: len.sn的最大长度, 39字节
 * 参数[out]: sn.设备sn
 * 注意: sn指针的字符串长度为0时将使用设备mac地址作为sn
 * 与hilink_sdk相同定义, 双模组模式只需一份, 已提供给第三方厂家, 暂不按编程规范整改
 */
DEFEAT(0) | STORY(0) | DOC(0)
void HilinkGetDeviceSn(unsigned int len, char *sn)
{
    if (sn == NULL) {
        return;
    }
    const char *ptr = DEVICE_SN;
    int tmp = MIN_LEN(len, sizeof(DEVICE_SN));
    for (int i = 0; i < tmp; i++) {
        sn[i] = ptr[i];
    }
    return;
}
```

如果没有设置SN, COMBO设备会使用WIFI MAC地址, 纯BLE会使用BLE MAC地址。

(3) 此处获取subProdId

```
/*
 * 获取设备的子型号，长度固定两个字节
 * subProdId为保存子型号的缓冲区，len为缓冲区的长度
 * 如果产品定义有子型号，则填入两字节子型号，并以'\0'结束，返回0
 * 没有定义子型号，则返回-1
 * 该接口需设备开发者实现
 * 与hilink_sdk相同定义，双模组模式只需一份，已提供给第三方厂家，暂不按编程规范整改
 */
DEFFECT(0) | STORY(0) | DOC(0)
int HILINK_GetSubProdId(char *subProdId, int len)
{
    if (subProdId == NULL) {
        return -1;
    }
    const char *ptr = SUB_PRODUCT_ID;
    int tmp = MIN_LEN((unsigned int)len, sizeof(SUB_PRODUCT_ID));
    for (int i = 0; i < tmp; i++) {
        subProdId[i] = ptr[i];
    }
    return 0;
}
```

(4) 此处获取广播功率强度

```
/*
 * 获取设备表面的最强点信号发射功率强度，最强点位置的确定以及功率测试方法，参照hilink认证蓝牙靠近发现功率设置及测试方法指导文档，power为出参，单位dbm，返回设备表面的最强信号强度值，如果厂商不想使用蓝牙靠近发现功能，接口直接返-1，如果需要使用蓝牙靠近发现，则接口返回0，如需及时生效，需调用HILINK_BT_StartAdvertise()方法启动广播
 */
DEFFECT(0) | STORY(0) | DOC(0)
int HILINK_BT_GetDevSurfacePower(char *power)
{
    if (power == NULL) {
        return -1;
    }
    *power = ADV_TX_POWER;
    return 0;
}
```

(6) 如果为 COMBO 模组，厂家通常需要根据以下接口获取 WIFI 状态，根据业务需求自行开启关闭广播

```

/*
 * 通知设备的状态
 * status表示设备当前的状态
 * 注意，此函数由设备厂商根据产品业务选择性实现
 */
void hilink_notify_devstatus(int status)
{
    switch (status) {
        case HILINK_M2M_CLOUD_OFFLINE:
            /* 设备与云端连接断开，请在此处添加实现 */
            break;
        case HILINK_M2M_CLOUD_ONLINE:
            /* 设备连接云端成功，请在此处添加实现 */
            break;
        case HILINK_M2M_LONG_OFFLINE:
            /* 设备与云端连接长时间断开，请在此处添加实现 */
            break;
        case HILINK_M2M_LONG_OFFLINE_REBOOT:
            /* 设备与云端连接长时间断开后进行重启，请在此处添加实现 */
            break;
        case HILINK_UNINITIALIZED:
            /* HiLink线程未启动，请在此处添加实现 */
            break;
        case HILINK_LINK_UNDER_AUTO_CONFIG:
            /* 设备处于配网模式，请在此处添加实现 */
            break;
    }
}

```

## 2.2 对外 API 接口介绍

### 2.2.1 BLE\_CfgNetInit（此函数目前不可重入）

```

/* BLE 配网资源申请：BLE 协议栈启动、配网回调函数挂接 */
int BLE_CfgNetInit(BLE_InitPara *para, BLE_CfgNetCb *cb);

```

此接口包含GATT服务创建，广播参数，是否配对绑定参数以及提供给厂家实现的回调函数，启动广播需调用BLE\_CfgNetAdvCtrl

- (1) 如果需要通过WIFI SDK进行配网，则不需要实现回调函数内容

```

/* BLE 配网回调函数 */
BLE_CfgNetCb g_BleCfgNetCb = {
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
}

```

```
};
```

(2) 如果需要自行实现WIFI配网, 则需实现

```
/* BLE 配网回调函数 */
BLE_CfgNetCb g_BleCfgNetCb = {
    GetDevPinCodeCb,
    GetDeviceInfoCb,
    SetCfgNetInfoCb,
    RcvCustomDataCb,
    CfgNetProcessCb,
};
```

GetDevPinCodeCb --> 输入speke协商中的pincode,需与手机保持一致

GetDeviceInfoCb --> 获取设备信息, 例:

```
"{"pId":"2AQO","sn":"112233445566","vendor":"media"}"
```

SetCfgNetInfoCb --> 获取配网时需要的wifi ssid和密码

RcvCustomDataCb --> 获取手机发送给设备的自定义信息, 秒控通过此接口

CfgNetProcessCb --> 获取speke协商等实时状态

(3) 如果需要发送hilink构建的广播内容, 将此变量配置为空initPara.advInfo = NULL;  
如果想发送自定义广播, 请自行设置。

(4) 如果需要在初始化后销毁蓝牙协议栈需告之手机侧, 设置  
isBlePair.isDeinitBleStack = BLE\_DEINIT\_FALG;并由厂家调用API销毁协议栈。

### 2.2.2 BLE\_CfgNetDeInit (此函数目前不可重入)

```
/*
 * BLE 配网资源注销: 配网回调函数清理、BLE 协议栈销毁
 * flag 为 0: 只销毁控制和调度线程, flag 为 1 销毁蓝牙协议栈, 该函数不可重入
 */
int BLE_CfgNetDeInit(const BLE_GattHandleList *handleList, unsigned int flag);
```

此函数销毁资源不是同步实现, 资源销毁需要ms级延迟

### 2.2.3 BLE\_CfgNetAdvCtrl

```
/* BLE 配网广播控制: 参数代表广播时间, 0:停止; 0xFFFFFFFF:一直广播, 其他: 广播指定时间后停止, 单位秒 */
int BLE_CfgNetAdvCtrl(unsigned int advSecond);
```

如不使用hilink能力设置广播参数和发送广播，请不要使用此函数

#### 2.2.4 BLE\_CfgNetAdvUpdate

```
/* 更新广播参数，更新完成后需调用 BLE_CfgNetAdvCtrl 启动广播 */  
int BLE_CfgNetAdvUpdate(BLE_AdvInfo *advInfo);
```

启动广播需调用BLE\_CfgNetAdvCtrl

#### 2.2.5 BLE\_CfgNetDisconnect

```
/* BLE 配网断开连接：防止其他任务长时间占用 BLE 连接 */  
int BLE_CfgNetDisconnect(void);
```

#### 2.2.6 BLE\_SendCustomData

```
/* BLE 发送用户数据：用户数据发送，与接收回调函数配套使用 */  
int BLE_SendCustomData(BLE_DataType dataType, const unsigned char *buff  
, unsigned int len);
```

设备发送自定义信息给手机，秒控通过此接口

## 2.3 在 demo.c 中需实现函数

### 2.3.1 示例

```
void main(void)
{
    HILINK_SetNetConfigMode(HILINK_NETCONFIG_OTHER);
    hilink_main();

    /* BLE 配网资源申请: BLE 协议栈启动、配网回调函数 */
    int ret = BLE_CfgNetInit(&initPara, &g_BleCfgNetCb);
    if (ret != 0) {
        printf("ble cfg net init fail");
        return ret;
    }

    /* BLE 配网广播控制: 参数代表广播时间, 0:停止, 0xFFFFFFFF:一直广播, 其他: 广播指定时间后停止, 单位秒 */
    ret = BLE_CfgNetAdvCtrl(180);
    if (ret != 0) {
        printf("ble cfg net adv ctrl fail\r\n");
        return ret;
    }
}
```

### 2.3.2 HILINK\_BT\_SetConfigDirPath

```
/*
 * 厂家设置 linux 系统读写文件的路径, 路径长度不能超过 33 个字节, 包含结束符
 * 要求存储路径是非易失性的, 版本升级不影响该路径文件
 */
void HILINK_BT_SetConfigDirPath(const char *configDirPath);
```

设置路径为绝对路径, 如果厂家使用的是 linux 系统, 读写文件设置的默认路径为 /usr/data/hilink\_bt, 如该路径不可使用, 可调用此函数自行设置接口



### 2.3.3 HILINK\_GetSubProdId

```
/*
 * 获取设备的子型号，长度固定两个字节
 * subProdId 为保存子型号的缓冲区，len 为缓冲区的长度
 * 如果产品定义有子型号，则填入两字节子型号，并以'\0'结束，返回 0
 * 没有定义子型号，则返回-1
 * 该接口需设备开发者实现
 * 与 hilink sdk 相同定义，双模组模式只需一份，已提供给第三方厂家，暂不按编程
规范整改
 */
int HILINK_GetSubProdId(char *subProdId, int len);
```

### 2.3.4 HILINK\_BT\_GetDevSurfacePower

```
/*
 * 获取设备表面的最强点信号发射功率强度，最强点位置的确定以及功率测试方
 * 法，参照 hilink 认证蓝牙靠近发现功率设置及测试方法指导文档，power 为出参
 * ，单位 dbm，返回设备表面的最强信号强度值，如果厂商不想使用蓝牙靠近发现功
 * 能，接口直接返-1，如果需要使用蓝牙靠近发现，则接口返回 0，如需及时生效，需
 * 调用 HILINK_BT_StartAdvertise()方法启动广播
 */
int HILINK_BT_GetDevSurfacePower(char *power);
```

## 2.4 系统 OS 接口实现

需要实现OS接口包括, 如果使用harmony OS, 可复用

```
/* 填写 harmonyos 相关适配的实现声明 */
static HILINK_BT_OsInterface g_osInterface = {
    GetTime,
    Msleep,
    GetTimeStamp,
    malloc,
    free,
    TaskCreate,
    TaskDelete,
    GetCurrentTaskId,
    MutexCreate,
    MutexDestroy,
    MutexLock,
    MutexUnlock,
    CreateSem,
    WaitSemTime,
    PostSem,
    DestroySem,
    printf,
};
```

## 2.5 读写 flash/文件系统接口适配

需要实现以下接口

```
static HILINK_BT_ConfigInterface g_confMgrInterface = {
    CreateItem,
    ReadItem,
    WriteItem,
    DeleteItem
};
```

## 2.6 蓝牙协议栈接口适配

使用harmony OpenAPI其中的函数

### 2.6.1 BleSetSecurityIoCap

```
/*  
 * @brief set security IO capability  
 * @param[in] <mode> BleIoCapMode  
 * @return 0-success, other-fail  
 */  
int BleSetSecurityIoCap(BleIoCapMode mode);
```

设置模组IO口输入输出能力

### 2.6.2 BleSetSecurityAuthReq

```
/*  
 * @brief set security authority  
 * @param[in] <mode> BleAuthReqMode  
 * @return 0-success, other-fail  
 */  
int BleSetSecurityAuthReq(BleAuthReqMode mode);
```

设置安全权限, 是否需要配对绑定

### 2.6.3 InitBtStack

```
/*  
 * @brief Initialize the Bluetooth protocol stack  
 * @param[in] void  
 * @return 0-success, other-fail  
 */  
int InitBtStack(void);
```

初始化蓝牙协议栈

### 2.6.4 EnableBtStack

```
/*  
 * @brief Bluetooth protocol stack enable  
 * @param[in] void
```

```
* @return 0-success, other-fail
*/
int EnableBtStack(void);
```

使能蓝牙协议栈

### 2.6.5 GAP 回调->AdvStartCompleteCb

```
/* Callback invoked when start adv operation has completed */
typedef void (*AdvEnableCallback)(int advId, int status);
```

仅检查返回状态

### 2.6.6 GAP 回调->AdvStopCompleteCb

```
/* Callback invoked when stop adv operation has completed */
typedef void (*AdvDisableCallback)(int advId, int status);
```

仅检查返回状态

### 2.6.7 GAP 回调-> SecurityRespondCb

```
/* Callback invoked when security response operation has completed */
typedef void (*SecurityRespondCallback)(const BdAddr *bdAddr);
```

在此回调中会调用BleGattSecurityRsp来授予安全请求访问权限

### 2.6.8 BleGattSecurityRsp

```
/*
 * @brief The device accept or reject the connection initiator.
 * @param[in] <bdAddr> initiator's address
 * @param[in] <accept> 0-reject, 1-accept
 * @return 0-success, other-fail
 */
int BleGattSecurityRsp(BdAddr bdAddr, bool accept);
```

在SecurityRespondCb回调中会调用此函数来授予安全请求访问权限

### 2.6.9 BleGattRegisterCallbacks

```
/*
 * @brief Callback invoked for gatt common function
 * @param[in] <BtGattCallbacks> Callback funcs
 * @return 0-success, other-fail
 */
int BleGattRegisterCallbacks(BtGattCallbacks *func);
```

注册GAP回调函数

### 2.6.10 BleGattsSetEncryption

```
/*
 * @brief Set the encryption level of the data transmission channel during connection
 * @param[in] <bdAddr> remote address
 * @param[in] <secAct> BleSecAct
 *
 * @return 0-success, other-fail
 */
int BleGattsSetEncryption(BdAddr bdAddr, BleSecAct secAct);
```

在ConnectServerCb回调中会调用此函数来设置安全连接加密等级

### 2.6.11 GATT 回调->ConnectServerCb

```
/* Callback indicating that a remote device has connected */
typedef void (*ConnectServerCallback)(int connId, int serverId, const BdAddr *bdAddr);
```

在此回调中会调用BleGattsSetEncryption来设置安全连接加密等级

### 2.6.12 GATT 回调-> DisconnectServerCb

```
/* Callback indicating that a remote device has disconnected */
typedef void (*DisconnectServerCallback)(int connId, int serverId, const BdAddr *bdAddr);
```

仅检查断开连接状态

### 2.6.13 GATT 回调-> ServiceStartCb

```
/* Callback invoked in response to start_service */  
typedef void (*ServiceStartCallback)(int status, int serverId, int srvcHandle);
```

仅检查创建批量 GATT 服务状态

### 2.6.14 GATT 回调-> ServiceStopCb

```
/* Callback invoked in response to stop_service */  
typedef void (*ServiceStopCallback)(int status, int serverId, int srvcHandle);
```

仅检查删除批量 GATT 服务状态

### 2.6.15 GATT 回调->IndicationSentCb

```
/* Callback confirming that a notification or indication has been sent  
to a remote device */  
typedef void (*IndicationSentCallback)(int connId, int status);
```

仅检查发送indication状态

### 2.6.16 GATT 回调->MtuChangeCb

```
/* Callback invoked when the MTU for a given connection changes */  
typedef void (*MtuChangeCallback)(int connId, int mtu);
```

仅查看手机和设备协商的MTU值大小, 不做处理

### 2.6.17 BleGattsRegisterCallbacks

```
/*  
 * @brief Callback invoked for gatt server function  
 * @param[in] <BtGattServerCallbacks> Callback funcs  
 * @return 0-success, other-fail  
 */
```

```
int BleGattsRegisterCallbacks(BtGattServerCallbacks *func);
```

注册GATT回调

#### 2.6.18 SetDeviceName

```
/*  
 * @brief set this device's name for friendly  
 * @param[in] <name> device name  
 * @param[in] <len> length  
 * @return 0-success, other-fail  
 */  
int SetDeviceName(const char *name, unsigned int len);
```

目前并未有具体使用, 厂家打桩即可

#### 2.6.19 BleStartAdvEx

```
/*  
 * @brief Start advertising include set adv data.  
 * This API will not described in the development manual, only for Hili  
nk.  
 * @return 0-success, other-fail  
 */  
int BleStartAdvEx(int *advId, const StartAdvRawData rawData, BleAdvPara  
ms advParam);
```

发送广播, 包括广播参数和广播数据, 厂家返回advId, 设置结果厂家触发ServiceStartCb回调返回

#### 2.6.20 BleStopAdv

```
/*  
 * @brief stop ble advertising  
 * @param[in] <advId> specified by upper layer  
 * @return 0-success, other-fail  
 */  
int BleStopAdv(int advId);
```

通过在BleStartAdvEx中返回的advId来停止广播, 设置结果厂家触发ServiceStopCb回调返回

### 2.6.21 BleGattsStartServiceEx

```
/*
 * @brief Start service include add service/characteristic/Descriptor option.
 * This API will not described in the development manual, only for Hili nk.
 * @return 0-success, other-fail
 */
int BleGattsStartServiceEx(int *srvCHandle, BleGattService *srvcInfo);
```

厂家通过此接口批量创建GATT服务，并返回serverHandle，BleGattService结构体中包含GATT服务以及相关的属性和描述，依次下发，

例如服务1+属性1+描述1+属性2+描述2（不存在

OHOS\_BLE\_ATTRIB\_TYPE\_CHAR\_VALUE和

OHOS\_BLE\_ATTRIB\_TYPE\_CHAR\_CLIENT\_CONFIG）

此接口仅允许创建单个服务，如需创建多个服务需多次调用此接口

### 2.6.22 BleGattsStopServiceEx

```
/*
 * @brief Stop service.
 * This API will not described in the development manual, only for Hili nk.
 * @return 0-success, other-fail
 */
int BleGattsStopServiceEx(int srvCHandle);
```

通过BleGattsStartServiceEx返回的serverHandle来批量删除服务



### 2.6.23 BleGattsSendIndication

```
/*
 * @brief Send a notification or indication that a local characteristic
 * has been updated
 * @param[in] <serverId> server interface id
 * @param[in] <GattsSendIndParam> indication param
 * @return 0-success, other-fail
 */
int BleGattsSendIndication(int serverId, GattsSendIndParam *param);
```

通过此接口设备发送数据给手机, serverId暂不使用, GattsSendIndParam结构体中的 attrHandle来确定是通过哪一个

例如创建的服务为: 服务1+属性1+描述1+属性2+描述2

如果要通过属性1发送indicate, attrHandle等于serverHandle+1

如果要通过属性2发送indicate, attrHandle等于serverHandle+3

### 2.6.24 ReadBtMacAddr

```
/*
 * @brief read bt mac address
 * @param[in] <mac> mac addr
 * @param[in] <len> addr length
 * @return 0-success, other-fail
 */
int ReadBtMacAddr(unsigned char *mac, unsigned int len);
```

读取设备的蓝牙MAC地址

## 二、厂家实现蓝牙协议栈

（华为提供蓝牙数据传输通道以及配网等功能）

### 3 准备编译环境

- （1） 厂家提供编译链，联系华为接口人使用厂家提供的编译链编译蓝牙 SDK和WIFI SDK的静态/动态链接库（.a or .so）
- （2） WIFI SDK打开SUPPORT\_BLE
- （3） 音箱设备还需打开SUPPORT\_SPEAKER\_CFG\_NET宏
- （4） 在厂家的工程中添加文件夹参考以下内容（以下文件均在**发布件**中找到）

vendor\hilink				
名称	类型	修改日期	大小	
debug	文件夹	2021/4/29 17:03		
include	文件夹	2021/4/29 17:10		
release	文件夹	2021/4/28 11:24		
hilink_demo.c	C 文件	2021/5/6 17:52	21 KB	
hilink_device.c	C 文件	2021/4/29 17:04	4 KB	
hilink_device_sdk.c	C 文件	2021/4/29 17:04	8 KB	
hilink_ota.c	C 文件	2021/4/20 14:34	6 KB	

  

vendor\hilink\debug				
名称	类型	修改日期	大小	
libhilinkbtsdk.a	A 文件	2021/5/6 17:13	582 KB	
libhilinkdevicesdk.a	A 文件	2021/4/25 13:35	2,775 KB	
libhilinkota.a	A 文件	2021/4/25 13:35	183 KB	

  

vendor\hilink\include				
名称	类型	修改日期	大小	
ble_cfg_net_api.h	H 文件	2021/5/31 10:40	6 KB	
hilink.h	H 文件	2021/5/29 14:13	12 KB	
hilink_bt_api.h	H 文件	2021/5/31 10:17	10 KB	
hilink_device.h	H 文件	2021/5/29 11:36	4 KB	
hilink_log.h	H 文件	2021/5/29 11:36	4 KB	
hilink_netconfig_mode_mgt.h	H 文件	2021/5/29 13:48	3 KB	
hilink_osadapter.h	H 文件	2021/5/29 11:36	9 KB	
hilink_ota.h	H 文件	2021/5/29 11:36	5 KB	
hilink_ota_mcu.h	H 文件	2021/5/29 11:36	10 KB	
hilink_typedef.h	H 文件	2021/5/29 11:36	1 KB	

- （5） 修改厂家编译脚本，将上述文件添加进编译工程中，生成可烧录的二进制固件

## 4 厂家自实现内容

### 4.1 厂家按照以下内容自行发送广播

Total Length	Data Length	Length	Type	Value	说明
25	3	0x02	0x01	0x06	BLE可被发现
	22	0x15	0x16	0xEEFD010107040011F812XXXXXXXXF0004023939	未注册, 用0xF8功率值触发弹窗, XXXXXXX表示“XXXX”的ASCII码值, SN后两位“99”
Total Length	Data Length	Length	Type	Value	说明
23	3	0x02	0x01	0x06	BLE可被发现
	20	0x13	0x16	0xEEFD010107040011F812XXXXXXXXF023939	已注册, 用0xF8功率值触发详情页, SN后两位“99”
Total Length	Data Length	Length	Type	Value	说明
25	3	0x02	0x01	0x06	BLE可被发现
	22	0x15	0x16	0xEEFD010107040011F812XXXXXXXXF000C023939	已注册断网, 用0xF8功率值触发离网弹窗, SN后两位“99”
Total Length	Data Length	Length	Type	Value	说明
16	16	0x0F	0x09	“Hi-XXXX-XXXX00”	XXXX: 厂商自定义名, XXXX: product ID; 00: sub module ID

### 4.2 设置 GATT service

创建GATT service UUID为"15F1E600A27743FCA484DD39EF8A9100"

创建GATT characteristic UUID为"15F1E601A27743FCA484DD39EF8A9100"

(权限允许read和indicate)

创建GATT characteristic UUID为"15F1E602A27743FCA484DD39EF8A9100"

(权限允许write)

### 4.3 设置配对绑定并且配对方式为 justworks

## 5 厂家适配代码

### 5.1 参考 2.1 修改 hilink\_demo.c 的内容

### 5.2 参考 2.4 和 2.5 实现 OS 层接口和读写 flash 接口

linux等通用系统已适配完成，不需要厂家单独适配

### 5.3 在 main 函数中启动 hilink 相关 API

(1) 启动hilink\_main(), 此函数会初始化辅助配网依赖wifi组件的部分

(2) 调用BLE\_CfgNetInit(BLE\_InitPara \*para, BLE\_CfgNetCb \*cb)函数进行蓝牙初始化, 此处para传NULL, 则初始化函数只会调用注册的profile信息和厂家配网回调函数, 并不会启动hilink蓝牙协议栈, 厂家可根据需要实现BLE\_CfgNetCb回调函数

(3) 注册蓝牙数据发送接口, 注册GATT characteristic UUID为  
"15F1E601A27743FCA484DD39EF8A9100" 的indicate函数

```
int HILINK_BT_RegisterBtDataSendCallback(HILINK_BT_SendBtDataCallback callback);
```

(4) 处理蓝牙数据, 将下面的函数挂接到GATT characteristic UUID为  
"15F1E602A27743FCA484DD39EF8A9100" 的write回调中

```
int HILINK_BT_ProcessBtData(const unsigned char *buf, unsigned int len);
```

### 5.4 厂家判断当前设备是否支持本地控,根据当前条件发送广播

```
/* 厂家实现协议栈使用, 获取是否支持本地控广播状态 */  
int HILINK_BT_SupportLocalControl(void);
```

### 5.5 断开蓝牙连接时调用以下 API 释放资源

```
/* 断开蓝牙连接时, 释放通道占用资源 */  
void HILINK_BT_DisconnectFreeResource(void);
```

第一次建立连接后需调用此接口主动释放资源