Program Guide

MS1793 编程指南

32 位基于 ARM Cortex M0 核心的蓝牙低功耗芯片

Revision History:

Rev. No.	History	Issue Date	Remark
1.0	Initial issue	July 10, 2017	release
1.1	增加中断运行模式，增加接口函数和说明	Sept 1, 2017	
1.2	重新排版，消除 pdf 目录中的错误显示	Sept 25, 2017	
1.3	同步 api 接口到 2.1.4mmb 版本	Nov 25, 2017	
1.4	同步 api 接口到 2.1.4mmd 版本	Dec 6, 2017	
1.5	同步 api 接口到 3.1.1 版本	Dec 15, 2017	
1.6	同步 api 接口到 3.2.0 版本，使能 server_blob_rd_rsp 函数	Jan 8, 2018	

目录

1. 概述

本文是使用 MS1793 BLE 芯片实现 BLE 连接应用的编程指导，方便用户快速开发 BLE 产品和应用。

2. 软件架构

软件代码工程的目录和文件结构如下：

```
software\
├─app-xxx
    ├─main.c
    ├─app.c
    ├─mm32f031_it.c
    └─BSP.c
├─doc\
├─BSP\
├─MM32x031\
└─Src\
    ├─inc\
    ├─lib\
    ├─ADC\
    └─IWDG\
```

其中 BLE 协议栈 lib 放置在 lib 目录下，接口定义头文件在 inc 目录下。应用入口文件为 main.c，应用实现代码在 app.c，硬件相关的配置在 BSP.c。

3. BLE 特征值及双向数据通信操作方法

用户如需调整 BLE 数据交互的特征值、服务及数据的收发，可按照如下的几个步骤进行调整（代码均包含在文件 app.c 中）：

服务（service）及特征值定义

如图 1 所示，用户需要自行分配句柄（递增，不得重复），数据结构定义如下

```
typedef struct ble_character16{
    u16 type16;          //type2
    u16 handle_rec;      //handle
    u8  characterInfo[5];//property1 - handle2 - uuid2
    u8  uuid128_idx;     //0xff means uuid16,other is idx of uuid128
}BLE_CHAR;

typedef struct ble_UUID128{
    u8  uuid128[16];//uuid128 string: little endian
}BLE_UUID128;
```

图 1 特征值定义数据结构

图 1 中 type16 为 database 每个记录的类型，具体取值根据蓝牙规范定义。宏定义 TYPE_CHAR 为特征值（character），宏定义 TYPE_CFG 为 client configuration，宏定义 TYPE_INFO 为特征值描述信息；handle_rec 为对应记录的句柄；characterInfo 保存了对应特征值的属性（property1）、句柄（handle2）及 uuid（uuid2），其中 handle2 及 uuid2 为 16 bit 小端格式；uuid128_idx 表示 uuid2 的格式，如该值为 UUID16_FORMAT 则表示 uuid2 为 16bit 格式，反之则表示 uuid2 为 128bit 的 uuid 信息对应的索引值，该索引值对应于 AttUuid128List 的内容索引。uuid128 为小端格式保存。

图 2 给出了具体的定义示例，示例中包含三个 Service 配置，句柄分别对应的范围为 1-6，7-15 及 16-25，其中 7-15 为 Device Info Service，16-25 为用户自定义 Service。句柄 0x0004 为 Device Name 特征值，句柄 0x0009 为 Manufacture Info 特征值，句柄 0x000b 为 Firmware version 特征值，句柄 0x000f 为软件版本特征值。句柄 0x0012、0x0015 和 0x0018 为用户自定义特征值。


```
//
///STEP0:Character declare
//
const BLE_CHAR AttCharList[] = {
// ======  gatt =====  Do NOT Change!!
    {TYPE_CHAR,0x03,ATT_CHAR_PROP_RD, 0x04,0,0x00,0x2a,UUID16_FORMAT},//name
    //05-06 reserved
// ======  device info =====    Do NOT Change if using the default!!!
    {TYPE_CHAR,0x08,ATT_CHAR_PROP_RD, 0x09,0,0x29,0x2a,UUID16_FORMAT},//manufacture
    {TYPE_CHAR,0x0a,ATT_CHAR_PROP_RD, 0x0b,0,0x26,0x2a,UUID16_FORMAT},//firmware version
    {TYPE_CHAR,0x0e,ATT_CHAR_PROP_RD, 0x0f,0,0x28,0x2a,UUID16_FORMAT},//sw version
// ======  LED service or other services added here =====  User defined
    {TYPE_CHAR,0x11,ATT_CHAR_PROP_W|ATT_CHAR_PROP_RD|ATT_CHAR_PROP_NTF, 0x12,0, 0,0, 1/*uuid128-idx1*/ },//status
    {TYPE_CFG, 0x13,ATT_CHAR_PROP_RD|ATT_CHAR_PROP_W},//cfg
    {TYPE_CHAR,0x14,ATT_CHAR_PROP_W|ATT_CHAR_PROP_RD,                  0x15,0, 0,0, 2/*uuid128-idx2*/ },//cmd
    {TYPE_CHAR,0x17,ATT_CHAR_PROP_W|ATT_CHAR_PROP_RD,                  0x18,0, 0,0, 3/*uuid128-idx3*/ },//OTA
    {TYPE_INFO, 0x19,ATT_CHAR_PROP_RD},//description,"Mg OTA"
};

const BLE_UUID128 AttUuid128List[] = {
    {0x10,0x19,0x0d,0xc,0xb,0xa,9,8,7,6,5,4,3,2,1,0}, //idx0,little endian, service uuid
    {0x11,0x19,0x0d,0xc,0xb,0xa,9,8,7,6,5,4,3,2,1,0}, //idx1,little endian, character status uuid
    {0x12,0x19,0x0d,0xc,0xb,0xa,9,8,7,6,5,4,3,2,1,0}, //idx2,little endian, character cmd uuid
    {0x13,0x19,0x0d,0xc,0xb,0xa,9,8,7,6,5,4,3,2,1,0}, //idx3,little endian, character OTA uuid
};
```

图 2 服务器及特征值数据库定义示意图

如果客户需要自行定义 Service，可按需更改接口如下函数的实现

```
void att_server_rdByGrType( u8 pdu_type,
                            u8 attOpcode,
                            u16 st_hd,
                            u16 end_hd,
                            u16 att_type );
```

图 3 给出了调用 Service 的例子。缺省情况下如果客户对 Device Info 不做特别修改，可直接调用缺省函数 att_server_rdByGrTypeRspDeviceInfo(pdu_type)即可。

自定义 service 的调用方式可参考图 3 中红色框内的方式，对应的调用接口为

```
void att_server_rdByGrTypeRspPrimaryService( u8 pdu_type,
                                             u16 start_hd,
                                             u16 end_hd,
                                             u8*uuid,
                                             u8 uuidlen);
```

其中 start_hd 与 end_hd 为对应 Service handle 取值范围，uuid 为字符串，对应的长度由 uuidlen 给出。

```
////////////////////////////////////////////////////////////////////////////
///STEP1:Service declare
// read by type request handle, primary service declare implementation
void att_server_rdByGrType( u8 pdu_type, u8 attOpcode, u16 st_hd, u16 end_hd, u16 att_type )
{
 //!!!!!!!!!  hard code for gap and gatt, make sure here is 100% matched with database:[AttCharList] !!!!!

    if((att_type == GATT_PRIMARY_SERVICE_UUID) && (st_hd == 1))//hard code for device info service
    {
        att_server_rdByGrTypeRspDeviceInfo(pdu_type);//using the default device info service
        //apply user defined (device info)service example
        //{
        //    u8 t[] = {0xa,0x18};
        //    att_server_rdByGrTypeRspPrimaryService(pdu_type,0x7,0xf,(u8*)(t),2);
        //}
        return;
    }

    //hard code
    else if((att_type == GATT_PRIMARY_SERVICE_UUID) && (st_hd <= 0x10)) //usr's service
    {
        att_server_rdByGrTypeRspPrimaryService(pdu_type,0x10,0x19,(u8*)(AttUuid128List[0].uuid128),16);
        return;
    }
    //other service added here if any
    //to do....

    ///error handle
    att_notFd( pdu_type, attOpcode, st_hd );
}
```

用户自定义Service代码示例

图 3 自定义 service 调用方式示意图

数据读写

BLE 的数据读写就是对应特征值（句柄）的读写操作，协议对应的接口函数为：

【读操作】

```
void server_rd_rsp( u8 attOpcode,
                    u16 att_hd,
                    u8 pdu_type);
```

其中 att_hd 为手机通过 BLE 希望读取特征值的句柄，应答数据通过如下接口反馈

```
void att_server_rd( u8 pdu_type,
                    u8 attOpcode,
                    u16 att_hd,
                    u8* attValue,
                    u8 datalen );
```

其中 attValue 为应答的数据指针，数据长度为 datalen。注意数据最长不得超过 20 字节。

【写操作】

```
void ser_write_rsp( u8 pdu_type,
                    u8 attOpcode,
                    u16 att_hd,
```

u8* attValue,
u8 valueLen_w);

其中 att_hd 为从手机 BLE 传（写）过来数据对应的特征值的句柄，数据内容保存在变量 attValue 中，数据长度为 valueLen_w。

【Notify 数据发送操作】

Notify 通过如下接口进行

u8 sconn_notifydata(u8* data, u8 len);

其中 data 为需要发送到手机的数据，长度由 len 指定。原则上数据长度可以超过 20 字节，协议会自动拆包发送，该函数返回实际发送的数据长度。

需要注意的是，Notify 接口没有指定对应的句柄，如果用户定义并使用多个 Notify 的特征值，需要在发送数据前通过调用如下接口指定对应的句柄

void set_notifyhandle(u16 hd);

或者直接修改变量 u16 cur_notifyhandle 即可。

回调函数 void gatt_user_send_notify_data_callback(void)可用于蓝牙模块端主动发送数据，代码实现方式推荐使用循环缓冲区的形式。

【DeviceName】

用户可以根据需要修改设备名称。缺省情况下，设备名称由宏 DeviceInfo 定义，BLE 协议栈内会通过接口 u8* getDeviceInfoData(u8* len)获取该信息，用户可根据实际情况重新实现该函数（app.c 文件内）。

【软件版本信息】

用户可根据需要修改软件的版本信息。缺省情况下，宏 SOFTWARE_INFO 定义了软件版本信息。

4. 代码移植开发说明

1） 系统时钟设置 48MHz, SPI 时钟设置（至少）6MHz

2） 移植与蓝牙协议相关的接口函数，具体见文件 BSP.c 中 porting functions

此外，unsigned char* get_local_addr(void)；已经在 main.c 中实现，该接口会在配对情况下用到。

3） app.c 为对应应用对外接口，其余 app_xxx.c 为各种用户自定义特征情况下的使用示例。

4） 需要移植系统相关的函数如下：

获取系统 tick 值（单位 1ms）：GetSysTickCount（）

由于使用了低功耗，需要移植两个 MCU 频率切换的接口函数(在 bsp.c 文件中)

void SysClk8to48(void); 切回 PLL

void SysClk48to8(void); 切到 8MHz RC

5） 蓝牙协议运行的入口函数为 ble_run(...)，**该函数不会返回**。

6） 获取版本信息函数为 u8* GetFirmwareInfo(void);

7） 通过调试接口函数可以查看蓝牙状态机信息。

函数 u8* GetMgBleStateInfo(int* StateInfoSize/*Output*/);

获取对应的内存地址即长度（StateInfoSize）。

8） 请注意配置好 StackSize，推荐配置 2KB。

9） 为保证 BLE 协议栈状态正确，代码中在 BLE 协议栈处理数据的时侯会关闭中断，处理结束后打开中断，关闭中断的时间通常是～100us 的量级。开关中断使用了回调函数 DisableEnvINT（）、EnableEnvINT（）。

Details:

【mm32fxxx_it.c】增加

void SysTick_Handler(void);

unsigned int GetSysTickCount(void);

【BSP.c】

增加 GPIO 初始化、SystemTick 使能初始化实现。

设置 SPI 初始化，clock 6Mhz（至少）

【main.c】

调用（示例）蓝牙协议入口函数

radio_initBle(..);

ble_run(...); //never return

5. 注意事项

1、所有接口函数**不得阻塞**调用

2、函数 att_server_rd(...)每次调用发送的数据长度不得超过 20 字节

3、函数 sconn_notifydata(...)只能在协议主循环体内调用，函数不可重入，可以发送多于 20 字节的数据，协议会自动分包发送，且每个分包长度最大为 20 字节。推荐一次发送的数据尽量不超过 3 个分包。

4、UUID 支持 16bit 和 128bit 两种

6. 接口函数列表

接口函数定义在文件 **mg_api.h** 中，主要包括如下的函数：

1) void radio_initBle(unsigned char txpwr, unsigned char**addr/*Output*/);
2) void radio_standby(void);
3) void ble_run(unsigned short adv_interval);
4) void ble_set_adv_type(unsigned char type);
5) void ble_set_adv_data(unsigned char* adv, unsigned char len);
6) void ble_set_adv_rsp_data(unsigned char* rsp, unsigned char len);
7) void ble_set_name(unsigned char* name,unsigned char len);
8) void ble_set_interval(unsigned short interval);
9) void ble_set_adv_enableFlag(char sEnableFlag);
10) void ble_disconnect(void);
11) unsigned char *get_ble_version(void);
12) unsigned char *GetFirmwareInfo(void);
13) void ser_write_rsp_pkt(unsigned char pdu_type);
14) void att_notFd(unsigned char pdu_type, unsigned char attOpcode, unsigned short attHd);
15) void att_ErrorFd_ecode(unsigned char pdu_type, unsigned char attOpcode, unsigned short attHd, unsigned char errCode);
16) void att_server_rdByGrTypeRspDeviceInfo(unsigned char pdu_type);
17) void att_server_rdByGrTypeRspPrimaryService(unsigned char pdu_type,
    unsigned short start_hd,
    unsigned short end_hd,
    unsigned char*uuid,

unsigned char uuidlen);

18) void att_server_rd(unsigned char pdu_type,
unsigned char attOpcode,
unsigned short att_hd,
unsigned char* attValue,
unsigned char datalen);
19) unsigned char sconn_notifydata(unsigned char* data, unsigned char len);
20) void SetFixAdvChannel(unsigned char isFixCh37Flag);
21) void test_carrier(unsigned char freq, unsigned char txpwr);
22) void radio_setBleAddr(u8 addr[6]);
23) unsigned char sconn_UpdateConnParaReq(unsigned short IntervalMin, unisnged short InteralMax, unsigned char PreferredPeriodicity);
24) unsigned char sconn_UpdateConnParaReqS(unsigned short PreferConInterval);
25) void SIG_ConnParaUpdateReq(unsigned short IntervalMin, unsigned short IntervalMax, unsigned short SlaveLatency, unsigned short TimeoutMultiplier);
26) unsigned short sconn_GetConnInterval(void);
27) Unsigned char GetRssiData(void);

7. 接口函数说明

7.1 radio_initBle

函数原型：void radio_initBle(unsigned char txpwr, unsigned char**addr/*Output*/);

函数功能：初始化蓝牙芯片及蓝牙协议栈

输入参数：txpwr 该参数用于初始化蓝牙芯片发射功率，可取的值有 TXPWR_0DBM，TXPWR_3DBM 等

输出参数：addr 该参数返回蓝牙 MAC 地址信息，6 个字节长度。

返回值：无

注意事项：协议一开始调用。

7.2 radio_standby

函数原型：void radio_standby(void);

函数功能：蓝牙芯片进入 standby, 处于最省电状态

输入参数：无

输出参数：无

返回值：无

注意事项：此函数仅在系统进入 standby 前调用。系统进入 standby 后功耗约为 3.5uA。

7.3 ble_run

函数原型：void ble_run(unsigned short adv_interval);

函数功能：运行蓝牙协议

输入参数：adv_interval，参数的单位为 0.625ms，如 160 表示 100ms 的广播间隔

输出参数：无

返回值：无

注意事项：**本函数为堵塞调用，不返回**。

7.4 ble_set_adv_type

函数原型：void ble_set_adv_type(unsigned char type);

函数功能：设置 BLE 广播 Adv PDU Header

输入参数：type, 0-adv_ind, 2-adv_nonconn_ind, 缺省为 80。bit0-3: adv_type, bit4:RAdd, bit5:TxAdd, bit6-7:00

输出参数：无

返回值：无

注意事项：本函数可以在任意时侯调用，广播数据内容立即起效。

7.5 ble_set_adv_data

函数原型：void ble_set_adv_data(unsigned char* adv, unsigned char len);

函数功能：设置 BLE 广播数据

输入参数：adv，广播数据指针

len，广播数据长度

输出参数：无

返回值：无

注意事项：本函数可以在任意时侯调用，广播数据内容立即起效。

7.6 ble_set_adv_rsp_data

函数原型：void ble_set_adv_rsp_data(unsigned char* rsp, unsigned char len);

函数功能：设置 BLE 广播扫描应答数据

输入参数：rsp，广播扫描应答数据指针

len，广播扫描应答数据长度

输出参数：无

返回值：无

注意事项：1）本函数可以在任意时侯调用，广播数据内容立即起效。

2）调用本函数会导致函数 ble_set_name()失效，但 app.c 中函数 getDeviceInfoData()依然有效。

7.7 ble_set_name

函数原型：void ble_set_name(unsigned char* name,unsigned char len);

函数功能：设置 BLE 广播应答包内名字内容

输入参数：name，名字数据指针

len，名字数据长度

输出参数：无

返回值：无

注意事项：1）本函数可以在任意时侯调用，数据内容立即起效。

2）调用本函数仅会改变广播扫描应答的数据内容，为了与 GATT 对应内容一致，需要同步修改 app.c 中 DeviceInfo 的内容，具体可参考 app.c 中函数 updateDeviceInfoData()的实现。

7.8 ble_set_interval

函数原型：void ble_set_interval(unsigned short interval);

函数功能：设置 BLE 广播间隔时间

输入参数：interval，广播间隔，单位 0.625ms

输出参数：无

返回值：无

注意事项：本函数可以在任意时侯调用，数据内容立即起效。

7.9 ble_set_adv_enableFlag

函数原型：void ble_set_adv_enableFlag(char sEnableFlag);

函数功能：设置 BLE 是否广播

输入参数：sEnableFlag，1--运行广播；0--停止广播

输出参数：无

返回值：无

注意事项：本函数可以在任意时侯调用，立即起效。

7.10 ble_disconnect

函数原型：void ble_disconnect(void);

函数功能：断开已有的连接

输入参数：无

输出参数：无

返回值：无

注意事项：本函数可以在任意时侯调用，立即起效。

7.11 get_ble_version

函数原型：unsigned char *get_ble_version(void);

函数功能：获取蓝牙协议栈版本信息字符串

输入参数：无

输出参数：无

返回值：蓝牙协议栈版本信息字符串

注意事项：按需调用

7.12 GetFirmwareInfo

函数原型：unsigned char *GetFirmwareInfo(void);

函数功能：获取蓝牙基带版本信息字符串

输入参数：无

输出参数：无

返回值：蓝牙基带版本信息字符串

注意事项：按需调用

7.13 ser_write_rsp_pkt

函数原型：void ser_write_rsp_pkt(unsigned char pdu_type);

函数功能：对具有 Write With Response 属性特征值写操作后的应答函数

输入参数：pdu 类型参数，直接引用回调函数 ser_write_rsp 中对应参数

输出参数：无

返回值：无

注意事项：对需要写应答的特征值，如果不应答会导致连接的断开。

7.14 att_notFd

函数原型：void att_notFd(unsigned char pdu_type, unsigned char attOpcode, unsigned short attHd);

函数功能：对无效特征值（或没有定义的特征值）进行操作的应答函数

输入参数：pdu_type PDU 类型参数

直接引用回调函数 ser_write_rsp、att_server_rdByGrType 或 server_rd_rsp 中对应参数

attOpcode 操作类型参数

直接引用回调函数 ser_write_rsp、att_server_rdByGrType 或 server_rd_rsp 中对应参数

attHd 对应特征值句柄值

直接引用回调函数 ser_write_rsp、att_server_rdByGrType 或 server_rd_rsp 中对应参数

输出参数：无

返回值：无

注意事项：凡是无效特征值的操作需要应答本函数，可将本函数作为缺省调用。

7.15 att_ErrorFd_eCode

函数原型：void att_ErrorFd_eCode(unsigned char pdu_type, unsigned char attOpcode, unsigned short attHd, unsigned char errorCode);

函数功能：对无效操作的应答函数

输入参数：pdu_type PDU 类型参数

直接引用回调函数 ser_write_rsp、att_server_rdByGrType 或 server_rd_rsp 中对应参数

attOpcode 操作类型参数

直接引用回调函数 ser_write_rsp、att_server_rdByGrType 或 server_rd_rsp 中对应参数

attHd 对应特征值句柄值

直接引用回调函数 ser_write_rsp、att_server_rdByGrType 或 server_rd_rsp 中对应参数

errorCode 错误代码，请参考 ATT Error Code

输出参数：无

返回值：无

ATT Error Code define:

```
#define ATT_ERR_INVALID_HANDLE              0x01
#define ATT_ERR_READ_NOT_PERMITTED          0x02
#define ATT_ERR_WRITE_NOT_PERMITTED         0x03
#define ATT_ERR_INVALID_PDU                 0x04
#define ATT_ERR_INSUFFICIENT_AUTHEN         0x05
#define ATT_ERR_UNSUPPORTED_REQ             0x06
#define ATT_ERR_INVALID_OFFSET              0x07
#define ATT_ERR_INSUFFICIENT_AUTHOR         0x08
#define ATT_ERR_PREPARE_QUEUE_FULL          0x09
#define ATT_ERR_ATTR_NOT_FOUND              0x0a
```



```
#define ATT_ERR_ATTR_NOT_LONG                0x0b
#define ATT_ERR_INSUFFICIENT_KEY_SIZE        0x0c
#define ATT_ERR_INVALID_VALUE_SIZE           0x0d
#define ATT_ERR_UNLIKELY                     0x0e
#define ATT_ERR_INSUFFICIENT_ENCRYPT         0x0f
#define ATT_ERR_UNSUPPORTED_GRP_TYPE         0x10
#define ATT_ERR_INSUFFICIENT_RESOURCES       0x11
```

7.16 att_server_rdByGrTypeRspDeviceInfo

函数原型：void att_server_rdByGrTypeRspDeviceInfo(unsigned char pdu_type);

函数功能：对缺省 Device Info 内容的应答可调用本接口函数

输入参数：pdu 类型参数，直接引用回调函数 att_server_rdByGrType 中对应参数

输出参数：无

返回值：无

注意事项：如果用户直接使用发布包代码，可直接调用本接口函数。

7.17 att_server_rdByGrTypeRspPrimaryService

函数原型：void att_server_rdByGrTypeRspPrimaryService(unsigned char pdu_type,
unsigned short start_hd,
unsigned short end_hd,
unsigned char*uuid,
unsigned char uuidlen);

函数功能：应答 Primary Service 的查询，用户需按特征值实际定义的句柄及 UUID 填充对应数据

输入参数：pdu_type PDU 类型参数，直接引用回调函数 att_server_rdByGrType 中对应参数

start_hd, 某个 Service 对应的起始句柄值

end_hd，某个 Service 对应的结束句柄值

uuid，某个 Service 对应的 UUID 字串（Hex 值），如 0x180A 表示为 0x0a，0x18。

uuidlen，某个 Service 对应 UUID 字串的长度

输出参数：无

返回值：无

注意事项：需要严格按照特征值定义规划填充对应参数。

7.18 att_server_rd

函数原型：void att_server_rd(unsigned char pdu_type,

unsigned char attOpcode,

unsigned short att_hd,

unsigned char* attValue,

unsigned char datalen);

函数功能：读取某特征值的值。

输入参数：pdu_type PDU 类型参数，直接引用回调函数 server_rd_rsp 中对应参数

attOpcode 操作对应的值，直接引用回调函数 server_rd_rsp 中对应参数

att_hd 特征值对应的句柄值，直接引用回调函数 server_rd_rsp 中对应参数

attValue 特征值对应的值字串指针

datalen 特征值字串长度

输出参数：无

返回值：无

注意事项：需要按需将对应特征值内容作为应答内容，如果对应特征值内容无效可应答 att_notFd（）。

7.19 sconn_notifydata

函数原型：unsigned char sconn_notifydata(unsigned char* data, unsigned char len);

函数功能：通过蓝牙发送数据

输入参数：data 需要发送的数据指针

len 数据长度

输出参数：无

返回值：成功发送的数据个数

注意事项：本接口函数会根据系统缓存情况自动拆包发送数据。但不得在原地阻塞等待循环调用本接口。

7.20 SetFixAdvChannel

函数原型：void SetFixAdvChannel(unsigned char isFixCh37Flag);

函数功能：在调试阶段，为了便于空中抓包，协议栈会固定在 37 通道广播，协议栈缺省在所有通道广播。

输入参数：isFixCh37Flag 设置是否仅在 37 通道广播。

输出参数：无

返回值：无

注意事项：正式代码中可在协议栈初始化前调用 SetFixAdvChannel(0)

7.21 test_carrier

函数原型：void test_carrier(unsigned char freq, unsigned char txpwr);

函数功能：测试实际载波发射功率，协议栈会固定在 2400+freq 频点发送载波，理想发射功率参考 txpwr。

输入参数：freq - 载波中心频率为(2400+freq)MHz

txpwr - 0x43 对应理想射频发射功率 0dBm

输出参数：无

返回值：无

注意事项：代码中在协议栈初始化后调用 test_carrier(80, 0x43)

7.22 radio_setBleAddr

函数原型：void radio_setBleAddr(unsigned char addr[6]);

函数功能：设置客户自己的蓝牙设备地址。

输入参数：addr - 蓝牙设备地址，6 字节

输出参数：无

返回值：无

注意事项：代码中在协议栈初始化前或者初始化后调用

7.23 sconn_UpdateConnParaReq

函数原型：unsigned char sconn_UpdateConnParaReq(unsigned short IntervalMin, unisnged short InteralMax, unsigned char PreferredPeriodicity);

函数功能：在连接状态下，尝试使用用户期望范围内的发射间隔。最终的发射间隔由 central 设备决定。

输入参数：IntervalMin - 最小发射间隔，单位 1.25ms。

IntervalMax - 最大发射间隔，单位 1.25ms。

PreferredPeriodicity - 发射间隔的基数，选择的发射间隔是这个数值的倍数。单位 1.25ms。

输出参数：无

返回值：连接状态返回 1；非连接状态返回 0

注：此接口不建议使用，请使用接口 SIG_ConnParaUpdateReq

7.24 sconn_UpdateConnParaReqS

函数原型：unsigned char sconn_UpdateConnParaReqS(unsigned short PreferredConnInterval);

函数功能：在连接状态下，尝试使用用户期望的发射间隔。最终的发射间隔由 central 设备决定。

输入参数：PreferredConnInterval - 期望的发射间隔

输出参数：无

返回值：连接状态返回 1；非连接状态返回 0

注：此接口不建议使用，请使用接口 SIG_ConnParaUpdateReq

7.25 SIG_ConnParaUpdateReq

函数原型：void SIG_ConnParaUpdateReq(unsigned short IntervalMin, unsigned short IntervalMax, unsigned short SlaveLatency, unsigned short TimeoutMultiplier);

函数功能：在连接状态下，尝试使用用户期望范围内的发射间隔。最终的发射间隔由 central 设备决定。

输入参数：IntervalMin - 最小发射间隔，单位 1.25ms。范围 6 到 3200。

IntervalMax - 最大发射间隔，单位 1.25ms。范围 6 到 3200。

SlaveLatency - 回应延迟，范围 0~500，还要小于(SupervisionTimeout / (IntervalMax*2)) -1。

TimeoutMultiplier - 连接断开时间。单位 10ms。范围 10 到 3200。

输出参数：无

返回值：无

注：SlaveLatency 建议范围 0-5。

7.26 sconn_GetConnInterval

函数原型：unsigned short sconn_GetConnInterval(void);

函数功能：获得目前蓝牙连接当前使用的发射间隔数值，单位为 1.25ms。

输入参数：无

输出参数：无

返回值：蓝牙连接当前的发射间隔

7.27 GetRssiData

函数原型：Unsigned char GetRssiData(void);

函数功能：接收广播包或者数据包时调用此函数，可以获取接收信号强度原始数据。可以根据参考的 rssi 数值，用此数据来计算接收信号强度。

输入参数：无

输出参数：无

返回值：接收信号强度

8. 硬件/系统通信/低功耗接口

配合蓝牙协议栈，需要系统实现 SPI 通信、定时器等接口。主要函数包括：

1、 unsigned int GetSysTickCount(void); 获取毫秒定时器累积值，用于计时等功能。

2、 void DisableEnvINT(void); 关闭中断

3、 void EnableEnvINT(void); 使能中断

4、 Char IsIrqEnabled(void) ; 判断 IRQ 信号是否产生中断（低电平为中断有效）。

5、 void SPI_CS_Enable_(void); SPI CSN 使能（低电平）

6、 void SPI_CS_Disable_(void); SPI CSN 去使能（高电平）

7、 unsigned char SPI_WriteRead(unsigned char SendData,unsigned char WriteFlag); SPI 读写 1 个字节的数据

另外，对于低功耗应用，蓝牙协议栈已经预留了接口供系统编程实现。主要函数包括：

1、 void SysClk48to8(void); 系统时钟由 48MHz 切换到 8MHz，用于 mcu 进入低功耗的准备，以及用于 mcu 关闭 pll 达到省电的效果。

2、 void SysClk8to48(void); 系统时钟由 8MHz 切换到 48MHz，用于 mcu 从低功耗唤醒后恢复正常的工作状态。

3、 void McuGotoSleepAndWakeup(void); mcu 进入低功耗，以及唤醒后的恢复操作

9. 与蓝牙应用开发相关的函数接口

为便于蓝牙差异化功能的灵活实现，蓝牙协议内设置了若干接口并以回调函数的方式由应用层 porting 实现，所有回调函数不得阻塞调用，具体函数的实现可参考 SDK 发布包中对应代码的实现示例。回调函数主要包括如下：

1、void gatt_user_send_notify_data_callback(void); 蓝牙连接成功后协议在空闲的时侯会调用本回调函数

2、void UsrProcCallback(void); 蓝牙协议会周期性回调本函数，无论是在广播状态还是连接状态

3、void ser_prepare_write(unsigned short handle,

unsigned char* attValue, unsigned short attValueLen, unsigned short att_offset);

4、void ser_execute_write(void);

以上两个函数为队列写数据回调函数。

5、unsigned char* getDeviceInfoData(unsigned char* len);

本函数是 GATT 中设备名称获取的回调函数。

6、void att_server_rdByGrType(unsigned char pdu_type, unsigned char attOpcode,

unsigned short st_hd, unsigned short end_hd, unsigned short att_type);

蓝牙 GATT 查询服务的回调函数

7、void ser_write_rsp(unsigned char pdu_type/*reserved*/, unsigned char attOpcode/*reserved*/,

unsigned short att_hd, unsigned char* attValue/*app data pointer*/,

unsigned char valueLen_w/*app data size*/);

蓝牙 GATT 写操作回调函数

8、void server_rd_rsp(unsigned char attOpcode, unsigned short attHandle, unsigned char pdu_type);

蓝牙 GATT 读操作回调函数，对 rd_req 的回应

9、void server_blob_rd_rsp(u8 attOpcode, u16 attHandle, u8 dataHdrP,u16 offset);

蓝牙 GATT 读操作回调函数，对 blob_rd_req 的回应

10、void ConnectStausUpdate(unsigned char IsConnectedFlag);

蓝牙连接状态更新回调函数

10. 配对/加密方式

一般 BLE 应用的 APP 建立连接过程不需要配对或者加密，但是对于 HID 等应用是需要的，所以我们也同时提供了支持配对/加密的蓝牙库。其中相关的接口函数包括：

函数原型：void SetLePinCode(unsigned char *PinCode/*6 0~9 digitals*/);

函数功能：设置 pin。如果不调用此函数，默认 pin 为 000000

输入参数：PinCode - 6 位数字(0~9）的字符串

输出参数：无

返回值：无

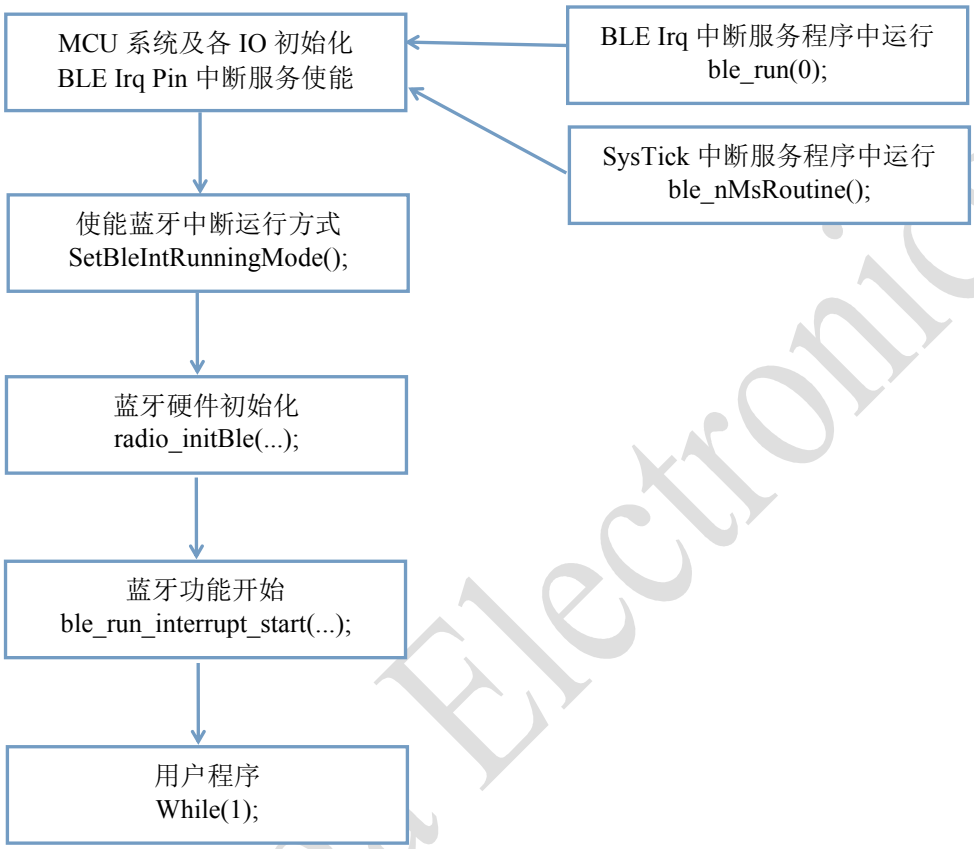
需要 app 实现的函数：

unsigned char aes_encrypt_HW(unsigned char *painText128bitBE,unsigned char *key128bitBE); //是否支持硬件 AES. 如果不支持请返回 0，蓝牙库支持软件 AES

11. 中断服务程序方式运行模式

蓝牙库以系统中断服务程序方式运行，适合于实现用户某功能需要占用较长 CPU 时间但可以被任意打断的应用场景。需要用到两个中断服务程序，一个是蓝牙 IRQ 中断 pin 对应的中断，一个是实现 SysTick 对应的中断。IRQ 对应的中断服务程序用以运行蓝牙协议，需要有较高的中断优先级（针对所有系统中断来说）。

中断服务程序方式运行的软件架构如下图所示。



这种运行方式下不再使用接口函数 McuGotoSleepAndWakeup()，而是使用：

```
unsigned char ble_run_interrupt_McuCanSleep(void);
```

低功耗功能实现示例：

```
void IrqMcuGotoSleepAndWakeup(void)
{
    if(ble_run_interrupt_McuCanSleep())
    {
        //to do MCU sleep and wakeup as in McuGotoSleepAndWakeup() under normal mode
    }
}
```

此函数可以在 main()函数的用户程序循环调用。

12. FAQ

1. 如何设置广播间隔？

   A：有两种方法 1）在调用接口 ble_run（）中由参数指定；2）调用接口函数 ble_set_interval()动态设置。

2. 如何设置发射功率？

   A：在芯片初始化接口函数 radio_initBle（）中由参数指定。

3. 如何修改广播内容及设备名字（Device Name）？

   A：广播内容可通过接口函数 ble_set_adv_data()动态设置，设备名字也可通过 app.c 文件中的函数 getDeviceInfoData（）进行按需修改设备名字。

4. 如何获取蓝牙地址（MAC）？

   A：芯片接口初始化函数 radio_initBle（）输出参数给出。

5. 如何自定义特征值？

   A：通过 app.c 文件中直接给特征值变量 AttCharList 赋值定义即可。

6. 如何自定义 Service？

   A：step1 通过 app.c 文件中直接分配 Service 及对应的特征值；

   step2 通过 app.c 文件中在回调函数 att_server_rdByGrType（）内应答 Service 句柄范围及 UUID。

7. 如何用特征值显示用户自己的软件版本信息？

   A：通过 app.c 文件回调函数 server_rd_rsp（）内对软件版本特征值应答用户自己的版本信息。

8. 如何获取连接状态？

   A：通过 app.c 的回调函数 ConnectStausUpdate 的输入参数获得，零表示断开，非零表示连接。

9. 如何通过蓝牙发送数据？

   A：首先需要通过手机 App 端将对应特征值的 Notify 使能，然后在 app.c 文件中回调函数 gatt_user_send_notify_data_callback（）中调用接口函数 sconn_notifydata（）实现，但不得阻塞调用。

10. 如何接收蓝牙收到的数据？

    A：蓝牙收到的数据会通过回调函数 ser_write_rsp（）中对应特征值通知，用户可按需保存。

11. 低功耗处理情况如何？

    A：可以根据不同的应用来确定低功耗模式。

对于灯控应用，可以使用 SLEEP 低功耗模式，不能使用 STOP 和 STANDBY 模式，否则 PWM 输出会停止；

对于数据透传应用，可以使用 SLEEP 或者 STOP 低功耗模式；

对于超时关机的应用，可以使用 STANDBY 低功耗模式。这种低功耗模式下的电流消耗约 3uA，但是只能通过 PA0 引脚的上升沿唤醒，或者通过 NRST 引脚进行复位唤醒。

对于低功耗的处理，**SDK 代码中程序一开始有一个延时等待便于通过 SW 口烧录**，超时后有可能会出现无法连接 SW 的现象，此时需要重新给模块上电并尽快连接 SW 即可。

12. 如何实现按键检测功能等用户自己的逻辑代码？

A：为了保证蓝牙功能的正常工作，用户不得长时间阻塞运行代码。

以按键检测为例，按键检测功能包括：1）检测；2）按键功能执行。

检测：可以通过系统中断函数按需要的频次扫描按键对应 IO 状态并更新按键状态信息。

执行：原则上中断函数不建议执行过多的任务内容，同时由于蓝牙各功能 API 具有不可重入的特点，因此按键对应的执行内容建议在蓝牙的回调函数中执行，如果对应的功能是 notify，那么可以在回调函数 gatt_user_send_notify_data_callback()中执行。其它的功能（如 SDK 中包含的串口数据处理）可以在回调函数 UsrProcCallback()中执行。

13. 队列写数据回调函数如何使用？

A：用户（或 App）在手机端对某个具有写属性的特征值进行写操作，手机蓝牙驱动会根据写操作数据量的多少会相应触发两类不同的写操作回调函数。

当写操作数据量小于等于 20 字节，触发 ser_write_rsp(...)回调函数；

当写操作数据量大于 20 字节，手机蓝牙底层会自动拆分数据并触发队列写数据回调函数，每个被拆分的数据段会依次触发 ser_prepare_write(...)，该函数携带了数据段所在的偏移量及数据长度。当所有数据片段都传输完成后会触发 ser_execute_write()回调函数，用户需要准备足够长的 Buffer 来保存接收到的数据。队列写的机制在结束之前是无法知道本次队列传输总的数据长度的，缓冲 Buffer 的长度应由具体应用交互协议确定（不是蓝牙协议确定）。通常队列写的操作遵循如下的步骤：

ser_prepare_write(...) ： slice1，offset=0，slice1_length = L1

ser_prepare_write(...) ： slice2，offset=L1，slice1_length = L2

ser_prepare_write(...) ： slice3，offset=L1+L2，slice1_length = L3

....

ser_execute_write() 结束。