编程指导手册

SoftWare Programming Guide

MM32W0x2xxB

32 位基于 ARM Cortex M0 核心的蓝牙低功耗芯片

版本:1.2

目录

图片目录

表格目录

1. 简介

SoftWareProgrammingGuide 是为了让 MM32W0x2xxB 用户快速上手软件代码实现的编程指导。使用 MM32W0x2xxB 芯片实现 BLE 数据双向传输、功耗控制、数字通信接口信号传输等功能，方便用户用 MM32W0x2xxB 芯片快速移植开发 BLE 产品。

2. 软件架构

软件代码的目录结果如下：

Projectgroup

├─User

├─main

├─STARTUP

├─system_MM32W0x2xxB.c

├─startup_MM32W0x2xxB_md.s

├─SYSTEM

├─sys

├─delay

├─uart

├─HALlib

├─HARDWARE

├─spi.c

├─callback.c

├─iic.c

├─IRQ_RF.c

├─rcc.h

├─AT_CMD.c

├─SRC_LIB

├─app.c

├─mg_BLE_lib_MM32W0x2xxB.lib

└─README

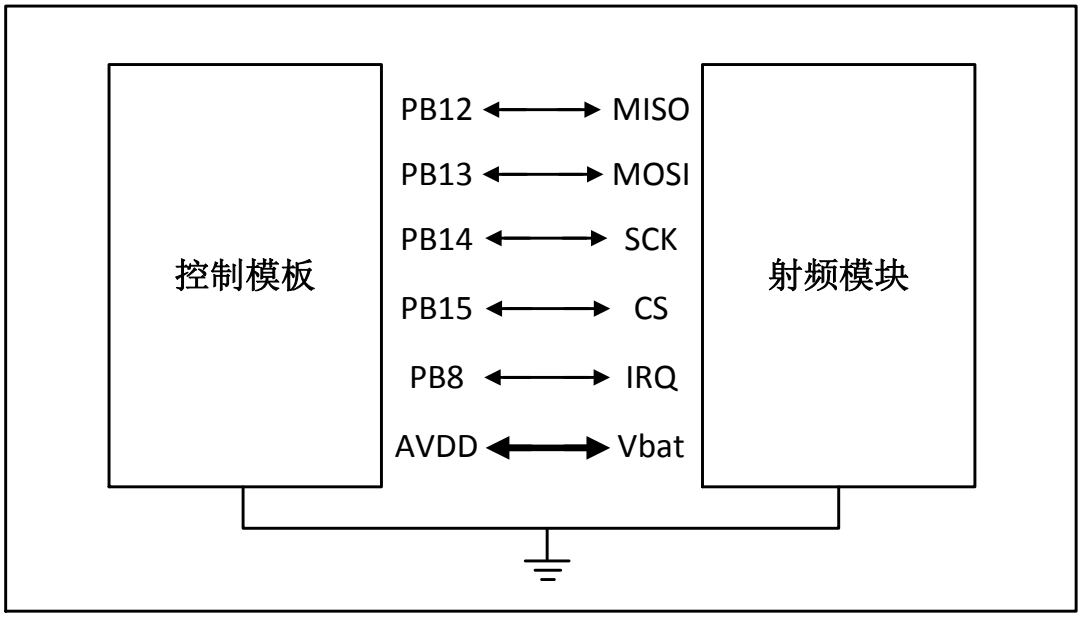
├─README.TXT

其中 BLE 芯片驱动及 BLE 协议栈 lib 放置在 SRC_LIB 目录下，接口函数定义放在 inc 目录下。

3. 控制模块和射频模块通信方式

3.1 通信框图

图 1. 通信框图



注：1、控制模块 SPI2 仅且只能用于与射频模块的通信。
2、IRQ 信号引脚用于射频模块与控制模块的唤醒，且 PB8 引脚只能用于控制模块唤醒。
3、AVDD 供电电压为 2.2V ~ 3.6V。

3.2 通信函数

3.2.1 SPI 函数初始化

函数原型：void SPIM_Init(SPI_TypeDef* SPIx,unsigned short spi_baud_div);

函数功能：初始化 SPI2 及设置 SPI 通信波特率

输入参数：SPIx，与射频模块通信只能选择 SPI2；spi_baud_div。

输出参数：无

返回值：无

注意事项：无

3.2.2 为选定的 SPI 软件重置内部 NSS 管脚

函数原型：void SPI_CS_Enable_(void);

函数功能：为选定的 SPI 软件重置内部 NSS 管脚

输入参数：无

输出参数：无

返回值：无

注意事项：无

函数原型：void SPI_CS_Disable_(void);

函数功能：为选定的 SPI 软件重置内部 NSS 管脚

输入参数：无

输出参数：无

返回值：无

注意事项：无

3.2.3 关闭 SPI 在双向模式下的数据传输

函数原型：void SPIM_TXEn(SPI_TypeDef* SPIx);

函数功能：

输入参数：SPIx，与射频模块通信只能选择 SPI2。

输出参数：无

返回值：无

注意事项：无

函数原型：void SPIM_TXDisable(SPI_TypeDef* SPIx);

函数功能：初始化 SPI2 及设置 SPI 通信波特率

输入参数：SPIx，与射频模块通信只能选择 SPI2。

输出参数：无

返回值：无

注意事项：无

函数原型：void SPIM_RXEn(SPI_TypeDef* SPIx);

函数功能：初始化 SPI2 及设置 SPI 通信波特率

输入参数：SPIx，与射频模块通信只能选择 SPI2。

输出参数：无

返回值：无

注意事项：无

函数原型：void SPIM_RXDisable(SPI_TypeDef* SPIx);

函数功能：

输入参数：SPIx，与射频模块通信只能选择 SPI2

输出参数：无

返回值：无

注意事项：无

3.2.4 SPI 收发数据配置

函数原型：unsigned int SPIMReadWriteByte(unsigned char tx_data);

函数功能：通过外设 SPI 收发数据，用于全双工模式（同时收发）

输入参数：tx_data

输出参数：无

返回值：无

注意事项：无

3.2.5 中断引脚配置

函数原型：void IRQ_Contrl_Init (void);

函数功能：射频模块中断控制引脚

输入参数：tx_data

输出参数：无

返回值：无

注意事项：无

如：

```
GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_8;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

4. MM32W0x2xxB 特征值及双向数据通信操作方法

用户如果需要根据自己的需求适当调整 BLE 交换的特征值、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）。

4.1 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：

图 2. 特征值定义数据结构

```
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 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 为特征值记录，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，示例给出了三个的 Service 配置，句柄分别对应的范围为 1-6，7-15 及 16- 25，其中 7-5 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。

图 3. 服务器及特征值数据库定义示意图

```
//
///STEP0:Character declare
//
const BLE_CHAR AttCharList[] = {                    特征值UUID        UUID格式
// ======  gatt =====  Do NOT Change!!
   {TYPE_CHAR,0x03,ATT_CHAR_PROP_RD, 0x04,0,0x00,0x2a,UUID16_FORMAT},//name
   //05-06 reserved          句柄
// ======  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
                          特征值属性
// ======  LED service or other services added here =====  User defined
   {TYPE_CHAR,0x11,ATT_CHAR_PROP_NTF,                          0x12,0, 0,0, 1/*uuid128-idx1*/ },//RxNotify
   {TYPE_CFG, 0x13,ATT_CHAR_PROP_RD|ATT_CHAR_PROP_W},//cfg
   {TYPE_CHAR,0x14,ATT_CHAR_PROP_W|ATT_CHAR_PROP_W_NORSP,      0x15,0, 0,0, 2/*uuid128-idx2*/ },//Tx
   {TYPE_CHAR,0x17,ATT_CHAR_PROP_W|ATT_CHAR_PROP_RD,           0x18,0, 0,0, 3/*uuid128-idx3*/ },//BaudRate
   {TYPE_INFO, 0x19,ATT_CHAR_PROP_RD},//description,"BaudRate"
};                                                                          uuid128 idx

const BLE_UUID128 AttUuid128List[] = {
   {0x9e,0xca,0x0dc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,0xf3,0xa3,0xb5,1,0,0x40,0x6e}, //idx0,little endian, service uuid
   {0x9e,0xca,0x0dc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,0xf3,0xa3,0xb5,3,0,0x40,0x6e}, //idx1,little endian, RxNotify
   {0x9e,0xca,0x0dc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,0xf3,0xa3,0xb5,2,0,0x40,0x6e}, //idx2,little endian, Tx
   {0x9e,0xca,0x0dc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,0xf3,0xa3,0xb5,4,0,0x40,0x6e}, //idx3,little endian, BaudRate
};
```

如果用户需要自行定义 **service**，可按需更改接口函数，实现方式如下：

```
void att_server_rdByGrType( u8 pdu_type,
                            u8 attOpcode,
                            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。

图 4. 自定义 service 调用方式示意图

```
////////////////////////////////////////////////////////////////////////
///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*)(scanp),2);
     return;
   }

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

 }
```

用户自定义Service代码示例

4.2 数据读写

MM32W0x2xxB 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：

4.2.1 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。需要注意数据最长不得超过 20 字节。

4.2.2 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。

4.2.3 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。

4.2.4 DeviceName

用户可以根据需要修改 BLE 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）。

4.2.5 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。

5. 代码移植说明

(1) 系统时钟设置（至少）48MHz, SPI 时钟设置 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。同时需要实现获取蓝牙地址的接口函数 get_local_addr（），文件 main.c 中已经实现，可直接使用。

(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:

【SYSTEM/sys.c】增加

void SysTick_Handler(void);

unsigned intGetSysTickCount(void);

【HARDWARE】

spi.c：主要用于 MCU 与 BLE 之间的 SPI 通信接口配置，其中对应的引脚，PB12 为 MISO，PB13 为 MOSI，PB14 为 SCK，PB15 为片选信号，设置 SPI2 初始化，clock 6MHz（至少），用户在使用过程中需要分别配置这几个 GPIO 对应的功能。

irq.c：该接口主要用于控制 BLE 的中断控制。

app.c: app.c 为对应应用对外接口。

mg_BLE_lib_MM32W0x2xxB_mmb-t7.lib：主要用于存放 BLE 协议栈。

【USER\main.c】

调用（示例）蓝牙协议入口函数

radio_initBle()；

ble_run(); //never return

6. 依赖关系

(1) SPI 的 GPIO 复用关系、Ble 基带芯片 Irq 对应的 GPIO。

(2) SPI 的速度需要保证 6MHz，MCU 速度需要保证 48MHz 。

(3) 需要 MCU 系统实现 System tick（1ms）并实现对应的调用接口。

(4) 蓝牙协议栈推荐内存需求：推荐 RAM 2KB 以上。

7. 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

8. 接口函数

MM32W0x2xxB 蓝牙功能协议栈目前以 Lib 形式提供，用户通过调用相关接口的方式实现对应功能。本章节介绍了各对应各接口的定义及简要使用注意事项。

8.1 接口函数列表

表 1. 接口函数列表

	函数名称	函数功能
1	void radio_initBle(unsigned char txpwr, unsigned char**addr/*Output*/);	初始化蓝牙芯片及蓝牙协议栈
2	void ble_run(unsigned short adv_interval);	运行蓝牙协议
3	void ble_set_adv_data(unsigned char* adv, unsigned char len);	设置 BLE 广播数据
4	void ble_set_adv_rsp_data(unsigned char* rsp, unsigned char len);	设置 BLE 广播扫描应答数据
5	void ble_set_name(unsigned char* name,unsigned char len);	设置 BLE 广播应答包内名字内容
6	void ble_set_interval(unsigned short interval);	设置 BLE 广播间隔时间
7	void ble_set_adv_enableFlag(char sEnableFlag);	设置 BLE 是否广播
8	void ble_disconnect(void);	断开已有的连接
9	unsigned char *get_ble_version(void);	获取蓝牙协议栈版本信息字符串
10	unsigned char *GetFirmwareInfo(void);	获取蓝牙基带版本信息字符串
11	void ser_write_rsp_pkt(unsigned char pdu_type);	对具有 Write With Response 属性特征值写操作后的应答函数
12	void att_notFd(unsigned char pdu_type, unsigned char attOpcode, unsigned short attHd);	对无效特征值（或没有定义的特征值）进行操作的应答函数
13	void att_server_rdByGrTypeRspDeviceInfo(unsigned char pdu_type);	对缺省 Device Info 内容的应答可调用本接口函数

	函数名称	函数功能
14	void att_server_rdByGrTypeRspPrimaryService(unsigned char pdu_type, unsigned short start_hd, unsigned short end_hd, unsigned char*uuid, unsigned char uuidlen);	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
15	void att_server_rd(unsigned char pdu_type, unsigned char attOpcode, unsigned short att_hd, unsigned char* attValue, unsigned char datalen);	读取某特征值的值
16	unsigned char sconn_notifydata(unsigned char* data, unsigned char len);	通过蓝牙发送数据输入
17	unsigned char* GetMgBleStateInfo(int* StateInfoSize/*Output*/);	在调试阶段，获取缓存蓝牙协议栈状态机内存信息位置，可通过内存调试窗口查看
18	void SetFixAdvChannel(unsigned char isFixCh37Flag);	在调试阶段，为了编译空中抓包，协议栈会固定在 37 通道广播，调用本函数关闭此功能
19	void set_mg_ble_dbg_flag(unsigned char EnableFlag);	在调试阶段缺省串口（如果使用）会打印一些调试信息，可通过本函数关闭此功能
20	void radio_standby(void);	通过该函数可以使射频模块进入 standby 模式
21	unsigned char* GetLTKInfo(u8* newFlag);	获取当前连接设备的加密多样化值（EDIV）
22	unsigned char* GetMasterDeviceMac(unsigned char* MacType);	获取当前或者最后一次连接主设备的 Mac 地址

8.2 接口函数说明

8.2.1 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。

8.2.2 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 0.625us，如果 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。

8.2.3 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。

8.2.4 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。

8.2.5 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。

8.2.6 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。

8.2.7 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。

8.2.8 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。

8.2.9 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。

8.2.10 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。

8.2.11 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。

8.2.12 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。

8.2.13 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。

8.2.14 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 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。

8.2.15 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（）。

8.2.16 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反复调用本接口。

8.2.17 GetMgBleStateInfo

函数原型：unsigned char* GetMgBleStateInfo(int* StateInfoSize);

函数功能：在调试阶段，获取缓存蓝牙协议栈状态机内存信息位置，可通过内存调试窗口查看。

输入参数：无。

输出参数：状态机内存 Size。

返回值：返回状态机内存地址。

注意事项：仅在调试阶段使用，正式代码中不需要调用。

8.2.18 SetFixAdvChannel

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

函数功能：在调试阶段，为了编译空中抓包，协议栈会固定在 37 通道广播，调用本函数关闭此功能。

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：正式代码中请在协议栈初始化前调用 SetFixAdvChannel(0);

8.2.19 set_mg_ble_dbg_flag

函数原型：void set_mg_ble_dbg_flag(unsigned char EnableFlag);

函数功能：在调试阶段缺省串口（如果使用）会打印一些调试信息，可通过本函数关闭此功能。

输入参数：EnableFlag 设置是否使能调试打印信息

输出参数：无

返回值：无

注意事项：正式代码中请在协议栈初始化前调用 set_mg_ble_dbg_flag(0);

8.2.20 radio_standby

函数原型：void radio_standby(void);

函数功能：在通过该函数可以使射频模块进入 standby 模式。

输入参数：无

输出参数：无

返回值：无

注意事项：射频模块进入 standby 后不能定时唤醒（射频模块进入 STOP 模式可以定时唤醒自身以及控制模块），此时需要外界给 IRQ 提供上升沿电平信号才能唤醒射频模块，给 PA0 提供下降沿电平才能唤醒控制模块。

8.2.21 GetLTKInfo

函数原型：unsigned chunsigned char* GetLTKInfo(u8* newFlag);

函数功能：通过此函数可以获取当前连接设备的加密多样化值（EDIV）。

输入参数：1 表示新配对设备的信息，0 表示旧配对设备的信息

输出参数：无

返回值：u8* EDivData /*2 Bytes*/ (encrypted)

注意事项：1、此函数仅在 StartEncryption == 1 时才能被调用

2、此函数仅在配对功能下使用

8.2.22 GetMasterDeviceMac

函数原型：unsigned char* GetMasterDeviceMac(unsigned char* MacType);

函数功能：获取当前或者最后一次连接主设备的 Mac 地址。

输入参数： 无

输出参数：MacType（0 表示常用型，其他表示随机型）

返回值：主设备 Mac 地址（6 个字节）

注意事项：该函数需要在手机连接后才能调用该函数，调用该函数需要在回调函数（void UsrProcCallback(void)）中调用。

9. 需要 Porting 与蓝牙功能相关的函数接口

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的一些：

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

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 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

9、 void ConnectStausUpdate(unsigned char IsConnectedFlag);

蓝牙连接状态更新回调函数

10. 详细配置问答

(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) 模块低功耗处理情况如何？

A：模块缺省情况下会 MCU 会使用 Stop 模式 UART@9600bps 数据透传模式。为便于调试， 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，此时可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。

11. 修改记录

版本	内容	日期
V1.0	Release version	2017/5/26
V1.1	修改软件代码的目录	2017/7/26
V1.2	增加射频模块 standby 模式、获取 EDIV、主设备 Mac 接口函数	2017/10/12