Software Programming Guide

MM32W0x2xxB

32-Bit Bluetooth Low Energy Chip Based on ARM Cortex M0

Version: 1.2

MindMotion reserves the right to change the relevant information without prior notification.

Contents

1. Introduction 5
2. Software architecture 5
3. Communication method of control module and RF module 6
3.1 Communication block diagram 6
3.2 Communication functions 6
3.2.1 Initialize the SPI function 6
3.2.2 Reset internal NSS pins for the selected SPI software 6
3.2.3 Disable the SPI data transmission in the two-way mode 7
3.2.4 SPI configuration for sending and receiving data 7
3.2.5 Interrupt pin configuration 8
4. MM32W0x2xxB characteristic values and the operation method of two-way data communication 8
4.1 Service and characteristic value definition 8
4.2 Data read and write 10
4.2.1 Read operation 11
4.2.2 Write operation 11
4.2.3 Notify data transmition operation 11
4.2.4 Device name 12
4.2.5 Software version information 12
5. Code porting instructions 12
6. Dependencies 13
7. Notes 13
8. Interface functions 13
8.1 List of interface functions 14
8.2 Interface function instructions 16
8.2.1 radio_initBle 16
8.2.2 ble_run 16
8.2.3 ble_set_adv_data 16
8.2.4 ble_set_adv_rsp_data 16
8.2.5 ble_set_name 17

8.2.6 ble_set_interval 17
8.2.7 ble_set_adv_enableFlag 17
8.2.8 ble_disconnect 17
8.2.9 get_ble_version 17
8.2.10 GetFirmwareInfo 18
8.2.11 ser_write_rsp_pkt 18
8.2.12 att_notFd 18
8.2.13 att_server_rdByGrTypeRspDeviceInfo 19
8.2.14 att_server_rdByGrTypeRspPrimaryService 19
8.2.15 att_server_rd 19
8.2.16 sconn_notifydata 20
8.2.17 GetMgBleStateInfo 20
8.2.18 SetFixAdvChannel 20
8.2.19 set_mg_ble_dbg_flag 21
8.2.20 radio_standby 21
8.2.21 GetLTKInfo 21
8.2.22 GetMasterDeviceMac 21
9. Porting functional interfaces related to Bluetooth functions 22
10. Specific configuration Q&A 22
11. Revision history 24

Illustration

Figure 1. Communication block diagram 6
Figure 2. Data structure of characteristic value definition 8
Figure 3. Server and characteristic value database definition diagram................... 9
Figure 4. Diagram of user-defined service invocation method 10

Table

Table 1. List of interface functions 13

1. Introduction

The purpose of this Software Programming Guide is to help MM32W0x2xxB users to get started with the software code implementation quickly. BLE two-way data transmission, power consumption control, digital communication interface signal transmission and other functions can be enabled with the MM32W0x2xxB chip. In addition, this document will facilitate users to use the MM32W0x2xxB chip to port and develop BLE products quickly.

2. Software architecture

The directory results for the software codes are as follows:

```
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
```

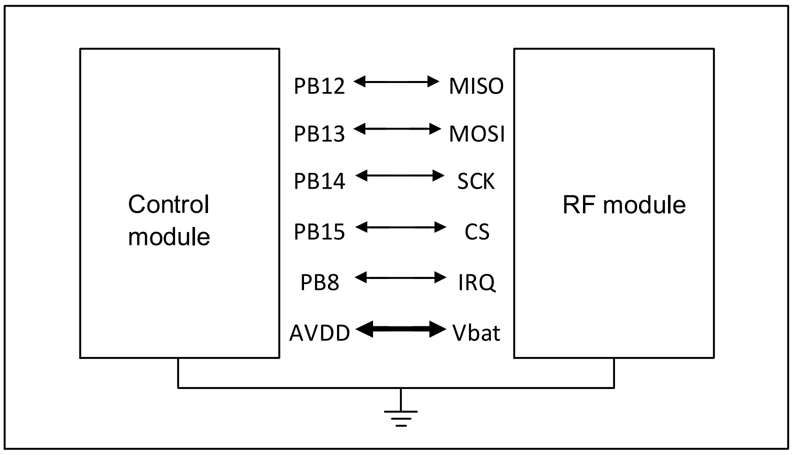
├─**README.TXT**

Among them, the BLE chip driver and the BLE protocol stack lib are placed in the SRC_LIB directory, while the interface function definitions are placed in the inc directory.

3. Communication method of control module and RF module

3.1 Communication block diagram

Figure 1. Communication block diagram



Note: 1. The control module SPI2 can only be used to communicate with the RF module.
2. The IRQ signal pin can be used for the RF module and control module wakeup, while the PB8 pin can only be used for the control module wakeup.
3. The AVDD supply voltage is 2.2V ~ 3.6V.

3.2 Communication functions

3.2.1 Initialize the SPI function

Function prototype: void SPIM_Init(SPI_TypeDef* SPIx,unsigned short spi_baud_div);

Function: Initialize the SPI2 and set the SPI communication baud rate

Input: SPIx (only the SPI2 can be selected to communicate with the RF module); spi_baud_div.

Output: None

Return value: None

Notes: None

3.2.2 Reset internal NSS pins for the selected SPI software

Function prototype: void SPI_CS_Enable_(void);

Function: Reset internal NSS pins for the selected SPI software

Input: None

Output: None

Return value: None

Notes: None

Function prototype: void SPI_CS_Disable_(void);

Function: Reset internal NSS pins for the selected SPI software

Input: None

Output: None

Return value: None

Notes: None

3.2.3 Disable the SPI data transmission in the two-way mode

Function prototype: void SPIM_TXEn(SPI_TypeDef* SPIx);

Function:

Input: SPIx (only the SPI2 can be selected to communicate with the RF module)

Output: None

Return value: None

Notes: None

Function prototype: void SPIM_TXDisable(SPI_TypeDef* SPIx);

Function: Initialize the SPI2 and set the SPI communication baud rate

Input: SPIx (only the SPI2 can be selected to communicate with the RF module)

Output: None

Return value: None

Notes: None

Function prototype: void SPIM_RXEn(SPI_TypeDef* SPIx);

Function: Initialize the SPI2 and set the SPI communication baud rate

Input: SPIx (only the SPI2 can be selected to communicate with the RF module)

Output: None

Return value: None

Notes: None

Function prototype: void SPIM_RXDisable(SPI_TypeDef* SPIx);

Function:

Input: SPIx (only the SPI2 can be selected to communicate with the RF module).

Output: None

Return value: None

Notes: None

3.2.4 SPI configuration for sending and receiving data

Function prototype: unsigned int SPIMReadWriteByte(unsigned char tx_data);

Function: Send and receive data via the peripheral SPI, which is used in the full duplex mode (simultaneous sending and receiving data)

Input: tx_data
Output: None

Return value: None

Notes: None

3.2.5 Interrupt pin configuration

Function prototype: void IRQ_Contrl_Init (void);

Function: Interrupt control pins for the RF module

Input: tx_data

Output: None

Return value: None

Notes: None

For example:

```
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 characteristic values and the operation method of two-way data communication

If users need to properly adjust the BLE exchange characteristic values, services and data sending and receiving according to their own requirements, they should follow these steps (codes are included in the app.c file).

4.1 Service and characteristic value definition

As shown in the Figure 1, users need to allocate handles (incremental, and not duplicated) by themselves, and the data structure is defined as follows:

Figure 2. Data structure of characteristic value definition

```
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;
```

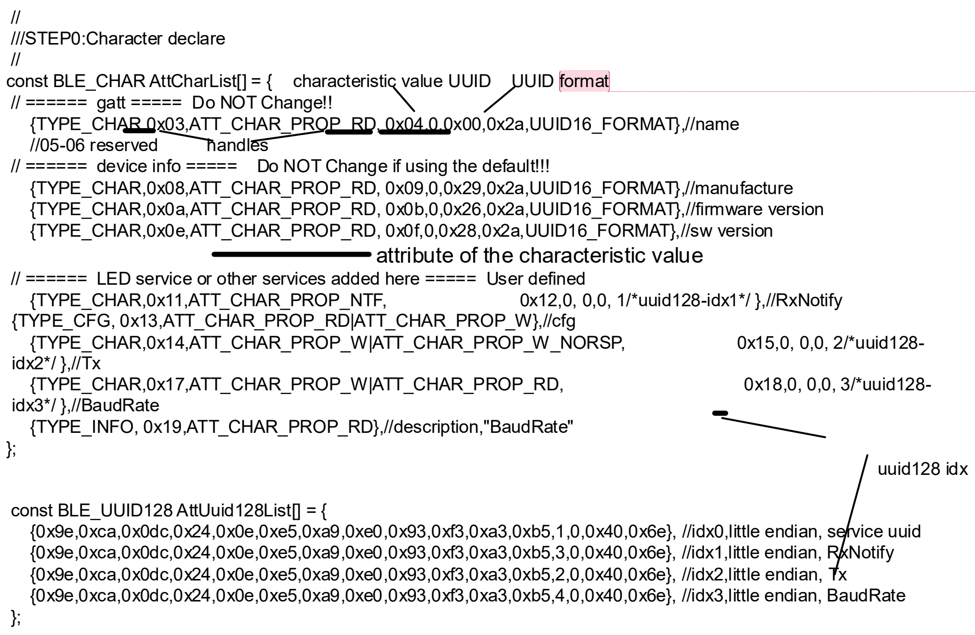
Commented [A1]: 此段代码和 pdf 原文的格式不同，请注意。

The type16 in Figure 1 is the type of each record in the database, the value of which should be defined according to the Bluetooth specification. The TYPE_CHAR is defined as the characteristic value record in macro. The TYPE_CFG is defined as the client configuration in macro. The TYPE_INFO is defined as the description information of the characteristic value and the handle_rec as the corresponding record in macros. The characterInfo stores the property (property1), handle (handle2) and uuid (uuid2) of the corresponding characteristic value, of which the handle2 and uuid2 are in the 16-bit little-endian format. The uuid128_idx indicates the format of uuid2. If the value is UUID16_FORMAT, it means uuid2 is in the 16-bit format. If it is another value, it means uuid2 is the corresponding index value of 128-bit uuid. The index value corresponds to the content index of AttUuid128List. The uuid128 is saved in the little-endian format.

Commented [p2]: ?

Figure 2 shows a specific example of the definition. The example provides three sets of service configurations. The handle ranges are 1-6, 7-15 and 16-25 respectively, where 7-5 is Device Info Service and 16-25 is User-Defined Service. The handle 0x0004 is the characteristic value of Device Name, the handle 0x0009 is the characteristic value of Manufacture Info, the handle 0x000b is the characteristic value of Firmware version, and the handle 0x000f is the characteristic value of software version. The handles 0x0012, 0x0015 and 0x0018 are user-defined characteristic values.

Figure 3. Server and characteristic value database definition diagram



```
//
///STEP0:Character declare
//
const BLE_CHAR AttCharList[] = {     characteristic value UUID    UUID format
// ======  gatt =====  Do NOT Change!!
    {TYPE_CHAR,0x03,ATT_CHAR_PROP_RD, 0x04,0,0x00,0x2a,UUID16_FORMAT},//name
    //05-06 reserved         handles
// ======  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
                                      attribute of the characteristic value
// ======  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
};
```

Commented [A3]: 此段代码跟 pdf 原文格式对比略有变动，请注意

If the user needs to define his own service, the interface function can be changed on demand and implemented as follows:

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

An example of service invocation is presented in Figure 3. By default, the default function att_server_rdByGrTypeRspDeviceInfo(pdu_type) can be invoked directly if the user does not make any special changes to the Device Info.

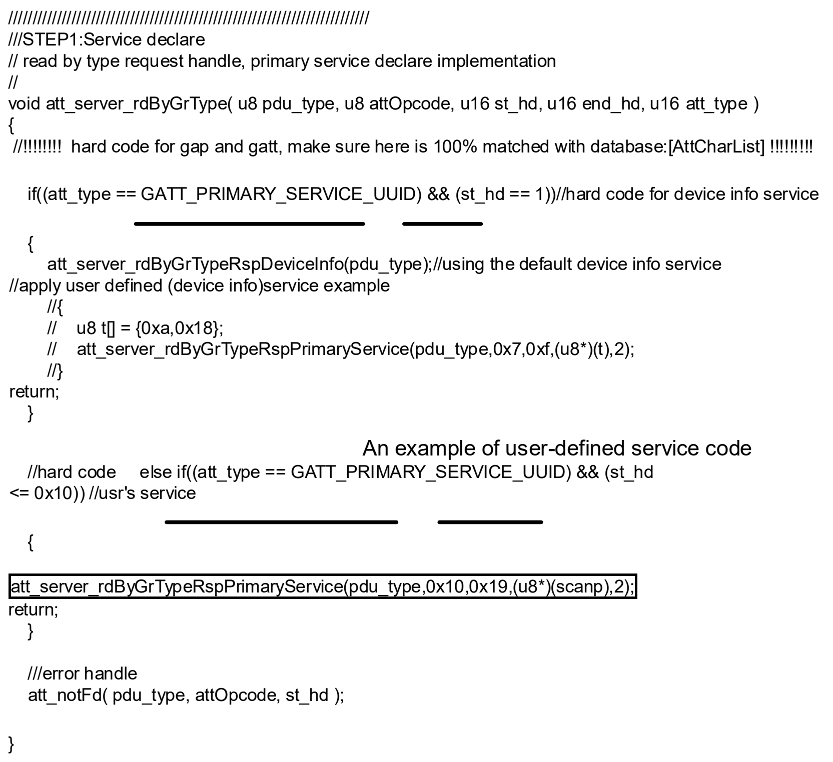
For the invocation method of the user-defined service, refer to the red box in Figure 3. The corresponding invocation interface is

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

                                             u8 uuidlen);
```

where the start_hd and end_hd set the value range of the corresponding service handle, while uuid is a string whose length is given by uuidlen.

Figure 4. Diagram of user-defined service invocation method



```
//////////////////////////////////////////////////////////////////////
///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;
    }

                                      An example of user-defined service code
    //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 );

}
```

4.2 Data read and write

The MM32W0x2xxB data read/write is the read/write operation corresponding to the characteristic value (handle). According to the protocol, interface functions are:

4.2.1 Read operation

```
void server_rd_rsp( u8 attOpcode,
                    u16 att_hd,

                    u8 pdu_type);
```

where the att_hd is the handle of the characteristic value that the mobile phone wants to read through BLE, and the response data is fed through the following interface

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

where the attValue is the data pointer for the response and the data length is datalen. Note that the maximum data length should not exceed 20 bytes.

4.2.2 Write operation

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

                    u8* attValue,

                    u8 valueLen_w);
```

where the att_hd is the handle of the characteristic value corresponding to the data transferred (written) from the mobile phone's BLE, and the data content is stored in the variable attValue.
The length of the data is valueLen_w.

4.2.3 Notify data transmition operation

Notify is performed through the following interface:
u8 sconn_notifydata(u8* data, u8 len);

where the data is the data to be sent to the mobile phone and the length is specified by len. In principle, the data length can exceed 20 bytes, because the protocol will automatically unpack and send such data. The function returns the actual length of the data sent.

Note: The Notify interface does not specify a corresponding handle. To define and use multiple characteristic values of Notify, the user need to specify the corresponding handle by invoking the following interface before sending the data

void set_notifyhandle(u16 hd);

or just modify the variable u16 cur_notifyhandle.

The callback function void gatt_user_send_notify_data_callback(void) can be used to send data initiatively from the Bluetooth module side. The cyclic buffer is recommended to implement the code.

4.2.4 Device name

Users can modify the BLE device name as needed. By default, the device name is defined by the macro DeviceInfo. This information is obtained within the BLE protocol stack through the interface u8* getDeviceInfoData(u8* len). Users can re-implement this function (in the app.c file) according to the actual situation.

4.2.5 Software version information

Users can modify the software version information as needed. By default, the software version information is defined by the macro SOFTWARE_INFO.

5. Code porting instructions

(1) The system clock should be set (at least) at 48Mhz and the SPI clock at 6MHz.

(2) For porting interface functions related to the Bluetooth protocol, see the porting functions in the BSP.c file. In addition, unsigned char* get_local_addr(void) ; has already been implemented in the main.c. This interface will be used in the pairing.

(3) The app.c is the external interface of the corresponding application and other app_xxx.c are samples used for various user-defined characteristics. Furthermore, the interface function get_local_addr() used to obtain the Bluetooth address has been implemented in the main.c file, which can be used directly.

(4) System-related functions that should be ported are as follows:

Get the system tick value (unit of 1ms): GetSysTickCount ()

Due to the use of low power consumption, it is necessary to port the interface function of two MCU frequency switching (in the bsp.c file) void SysClk8to48(void); switch back to PLL void SysClk48to8(void); switch to 8MHz RC.

(5) The entry function for the Bluetooth protocol is ble_run(), and this function will not return.

(6) The function to get the version information is u8* GetFirmwareInfo(void);

(7) The debug interface function allows users to check the information of the Bluetooth state machine.

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

Get the corresponding memory address, or length (StateInfoSize).

(8) Please remember to configure the StackSize and 2KB is recommended.

(9) To ensure the correct state of the BLE stack, the interrupt will be disabled in the code when the BLE protocol stack is processing data and enabled after the processing. The duration of interrupt disabling is usually in the order of 100us. Callback functions DisableEnvINT() and EnableEnvINT() are used to disable and enable the interrupt.

Details:

【SYSTEM/sys.c】has added

void SysTick_Handler(void);

unsigned intGetSysTickCount(void);

【HARDWARE】

spi.c: mainly used for the SPI communication interface configuration between MCU and BLE. With regard to corresponding pins, PB12 is configured as MISO, PB13 as MOSI, PB14 as SCK, and PB15 as chip select. Initialize the SPI2 and the clock should be (at least) 6MHz. Users need to configure these GPIO functions respectively during use.

irq.c: This interface is mainly used to control the interrupt of BLE.

app.c: app.c is the external interface of the corresponding application.

mg_BLE_lib_MM32W0x2xxB_mmb-t7.lib: It is mainly used to store the BLE protocol stack.

【USER\main.c】

Entry function to invoke the (sample) Bluetooth protocol

radio_initBle()；

ble_run(); //never return

6. Dependencies

(1) The GPIO alternate functions of SPI and the GPIO corresponding to the Ble baseband chip Irq.

(2) The speed of SPI needs to be guaranteed at 6MHz and the speed of MCU needs to be guaranteed at 48MHz.

(3) MCU system should implement System tick (1ms) and the corresponding invocation interface.

(4) Recommended memory for Bluetooth protocol stack: above 2KB RAM.

7. Notes

(1) The invocation of all interface functions must not be blocked.

(2) The length of data sent via the function att_server_rd(...) for each invocation must not exceed 20 bytes.

(3) The function sconn_notifydata(...) which is non-reentrant can only be invoked within the main loop body of the protocol. The length of data to be sent can exceed 20 bytes. The protocol will automatically unpack and send data in packets. The maximum length of each sub-packet is 20 bytes. It is recommended that the data sent at one time should not contain more than 3 sub-packets if possible.

(4) UUID supports both 16 bit and 128 bit.

8. Interface functions

MM32W0x2xxB Bluetooth function protocol stack is currently provided in the form of Lib. Users can invoke relevant interfaces to realize corresponding functions.

This section introduces the definition of each interface and provides using tips briefly.

8.1 List of interface functions

Table **1.** List of interface functions

	Name	Function
1	void radio_initBle(unsigned char txpwr, unsigned char**addr/*Output*/);	Initialize the Bluetooth chip and Bluetooth protocol stack
2	void ble_run(unsigned short adv_interval);	Run the Bluetooth protocol
3	void ble_set_adv_data(unsigned char* adv, unsigned char len);	Set the BLE advertising data
4	void ble_set_adv_rsp_data(unsigned char* rsp, unsigned char len);	Set the BLE advertising scan response data
5	void ble_set_name(unsigned char* name,unsigned char len);	Set the name in the BLE advertising response packet
6	void ble_set_interval(unsigned short interval);	Set the BLE advertising interval
7	void ble_set_adv_enableFlag(char sEnableFlag);	Set whether the BLE advertising is enabled
8	void ble_disconnect(void);	Disconnect an existing connection
9	unsigned char *get_ble_version(void);	Get the string of Bluetooth protocol stack version information
10	unsigned char *GetFirmwareInfo(void);	Get the string of Bluetooth baseband version information
11	void ser_write_rsp_pkt(unsigned char pdu_type);	Response function for writing the characteristic value with Write With Response attribute
12	void att_notFd(unsigned char pdu_type, unsigned char attOpcode, unsigned short attHd);	Response function for operations on invalid characteristic values (or undefined characteristic values)
13	void att_server_rdByGrTypeRspDeviceInfo(unsigned char pdu_type);	Invoke this interface function to respond to the default Device Info content
14	void att_server_rdByGrTypeRspPrimaryService(unsigned char pdu_type, unsigned short start_hd, unsigned short end_hd, unsigned char*uuid, unsigned char uuidlen);	Response to Primary Service query; users needs to fill the corresponding data according to the actually defined handle and UUID by characteristic values

	Name	Function
15	void att_server_rd(unsigned char pdu_type, unsigned char attOpcode, unsigned short att_hd, unsigned char* attValue, unsigned char datalen);	Read the attribute of some characteristic value
16	unsigned char sconn_notifydata(unsigned char* data, unsigned char len);	Send input data via the Bluetooth
17	unsigned char* GetMgBleStateInfo(int* StateInfoSize/*Output*/);	In the debug phase, get the location of the cached memory information of Bluetooth protocol stack state machine, which can be viewed through the memory debug window
18	void SetFixAdvChannel(unsigned char isFixCh37Flag);	In the debug phase, the protocol stack will set the fixed advertising channel as the channel 37, in order to compile the over-the-air packet capture. This function can be invoked to turn this feature off.
19	void set_mg_ble_dbg_flag(unsigned char EnableFlag);	In the debug phase, the default serial port (if used) will print some debug information. This function can be invoked to turn this feature off.
20	void radio_standby(void);	This function enables the RF module to enter the Standby mode
21	unsigned char* GetLTKInfo(u8* newFlag);	Get the encrypted diversifier (EDIV) of the currently connected device
22	unsigned char* GetMasterDeviceMac(unsigned char* MacType);	Get the Mac address of the currently or last connected master device

8.2 Interface function instructions

8.2.1 radio_initBle

Function prototype: void radio_initBle(unsigned char txpwr, unsigned char**addr/*Output*/);

Function: Initialize the Bluetooth chip and Bluetooth protocol stack.

Input: This txpwr parameter is used to initialize the transmitting power of Bluetooth chip. Desirable values are TXPWR_0DBM, TXPWR_3DBM, etc.

Output: This addr parameter returns the Bluetooth MAC address information with the length of 6 bytes.

Return value: None

Note: It is invoked at the beginning of the protocol.

8.2.2 ble_run

Function prototype: void ble_run(unsigned short adv_interval);

Function: Run the Bluetooth protocol.

Input: The unit of this adv_interval parameter is 0.625us. A value of 160 means 100ms advertising interval.

Output: None

Return value: None

Note: This function is invoked in a blocking way.

8.2.3 ble_set_adv_data

Function prototype: void ble_set_adv_data(unsigned char* adv, unsigned char len);

Function: Set the BLE advertising data

Input: adv, advertising data pointer

len, advertising data length

Output: None

Return value: None

Note: This function can be invoked at any time. Once invoked, the advertising data takes effect immediately.

8.2.4 ble_set_adv_rsp_data

Function prototype: void ble_set_adv_rsp_data(unsigned char* rsp, unsigned char len);

Function: Set the BLE advertising scan response data

Input: rsp, advertising scan response data pointer

len, advertising scan response data length

Output: None

Return value: None

Note: 1) This function can be invoked at any time. Once invoked, the advertising data takes effect immediately.

2) Invoking this function will lead to the failure of function ble_set_name(), but the function getDeviceInfoData() in app.c is still valid.

8.2.5 ble_set_name

Function prototype: void ble_set_name(unsigned char* name,unsigned char len);

Function: Set the name in the BLE advertising response packet

Input: name, data pointer of name

len, data length of name

Output: None

Return value: None

Note: 1) This function can be invoked at any time. Once invoked, the data takes effect immediately.

2) Invoking this function will only change the advertising scan response data. In order to conform with the related content in the GATT, DeviceInfo in app.c should be modified in the same time. For details, please refer to the implementation of updateDeviceInfoData() in app.c.

8.2.6 ble_set_interval

Function prototype: void ble_set_interval(unsigned short interval);

Function: Set the BLE advertising interval

Input: interval, which is the advertising interval with the unit of 0.625ms

Output: None

Return value: None

Note: This function can be invoked at any time. Once invoked, the data takes effect immediately.

8.2.7 ble_set_adv_enableFlag

Function prototype: void ble_set_adv_enableFlag(char sEnableFlag);

Function: Set whether the BLE advertising is enabled

Input: sEnableFlag, 1 - enable advertising; 0 - disable advertising

Output: None

Return value: None

Note: This function can be invoked at any time and takes effect immediately.

8.2.8 ble_disconnect

Function prototype: void ble_disconnect(void);

Function: Disconnect an existing connection

Input: None

Output: None

Return value: None

Note: This function can be invoked at any time and takes effect immediately.

8.2.9 get_ble_version

Function prototype: unsigned char *get_ble_version(void);

Function: Get the string of Bluetooth protocol stack version information.

Input: None

Output: None

Return value: string of Bluetooth protocol stack version information

Note: It is invoked on demand.

8.2.10 GetFirmwareInfo

Function prototype: unsigned char *GetFirmwareInfo(void);

Function: Get the string of Bluetooth baseband version information.

Input: None
Output: None

Return value: string of Bluetooth baseband version information

Note: It is invoked on demand.

8.2.11 ser_write_rsp_pkt

Function prototype: void ser_write_rsp_pkt(unsigned char pdu_type);

Function: Response function for writing the characteristic value with Write With Response attribute.

Input: pdu type parameter, which references directly the corresponding parameter from the callback function ser_write_rsp.

Output: None

Return value: None

Note: Failure to respond to some characteristic value with Write With Response attribute will cause the disconnection.

8.2.12 att_notFd

Function prototype: void att_notFd(unsigned char pdu_type, unsigned char attOpcode, unsigned short attHd);

Function: Response function for operations on invalid characteristic values (or undefined characteristic values)

Input: For pdu_type PDU type parameters, directly reference corresponding parameters from the callback function ser_write_rsp, att_server_rdByGrType or server_rd_rsp.
For attOpcode operation type parameters, directly reference corresponding parameters from the callback function ser_write_rsp, att_server_rdByGrType or server_rd_rsp.
For attHd handles of corresponding characteristic values, directly reference corresponding parameters from the callback function ser_write_rsp, att_server_rdByGrType or server_rd_rsp.

Output: None

Return value: None

Note: For any operation on characteristic values to be answered, this function can be invoked as the default function.

8.2.13 att_server_rdByGrTypeRspDeviceInfo

Function prototype: void att_server_rdByGrTypeRspDeviceInfo(unsigned char pdu_type);

Function: Invoke this interface function to respond to the default Device Info content.

Input: pdu type parameter, referenced directly from the corresponding parameter in the callback function att_server_rdByGrType.

Output: None

Return value: None

Note: This interface function can be invoked directly if the code of release package is used directly.

8.2.14 att_server_rdByGrTypeRspPrimaryService

Function prototype: void att_server_rdByGrTypeRspPrimaryService(unsigned char pdu_type,

```
              unsigned short start_hd,
              unsigned short end_hd,
              unsigned char*uuid,
              unsigned char uuidlen);
```

Function: Response to Primary Service query; users needs to fill the corresponding data according to the actually defined handle and UUID by characteristic values.

Input: For pdu_type PDU type parameters, directly reference corresponding parameters from the callback function att_server_rdByGrType; start_hd is the corresponding start handle of one service; end_hd is the corresponding end handle of one service; uuid is the corresponding UUID string (Hex value) of one service, e.g. 0x180A means 0x0a, 0x18. uuidlen is the length of the UUID string corresponding to one service.

Output: None

Return value: None

Note: The corresponding parameters should be filled in strict accordance with definitions of characteristic values.

8.2.15 att_server_rd

Function prototype: void att_server_rd(unsigned char pdu_type,

```
              unsigned char attOpcode
              unsigned short att_hd,
              unsigned char* attValue,
              unsigned char datalen );
```

Function: Read the attribute of some characteristic value.

Input: For pdu_type PDU type parameter, directly reference the corresponding parameter from the callback function server_rd_rsp.

For the parameter corresponding to attOpcode operation, directly reference the corresponding parameter from the callback function server_rd_rsp.

For att_hd handle of corresponding characteristic value, directly reference the corresponding parameter from the callback function server_rd_rsp.

attValue is the string pointer of corresponding characteristic value.

datalen is the string length of characteristic value.

Output: None

Return value: None

Note: The content of the corresponding characteristic value needs to be used as the response on demand. If the content of the corresponding characteristic value is invalid, att_notFd() can be used as the response function.

8.2.16 sconn_notifydata

Function prototype: unsigned char sconn_notifydata(unsigned char* data, unsigned char len);

Function: Send input data via the Bluetooth.

Input: data, data pointer to be sent
len, data length.

Output: None

Return value: the number of data successfully sent.

Note: This interface function will automatically unpack and send data based on the system cache, but it shall not be blocked in place and repeatedly invoked.

8.2.17 GetMgBleStateInfo

Function prototype: unsigned char* GetMgBleStateInfo(int* StateInfoSize);

Function: In the debug phase, get the location of the cached memory information of Bluetooth protocol stack state machine, which can be viewed through the memory debug window.

Input: None.

Output: state machine memory size.

Return value: return the state machine memory address.

Note: Only used in the debug stage, and not invoked in the formal code.

8.2.18 SetFixAdvChannel

Function prototype: void SetFixAdvChannel(unsigned char isFixCh37Flag);

Function: In the debug phase, the protocol stack will set the fixed advertising channel as the channel 37, in order to compile the over-the-air packet capture. This function can be invoked to turn this feature off.

Input: isFixCh37Flag sets whether to fix the advertising just on channel 37.

Output: None

Return value: None

Note: Please invoke SetFixAdvChannel(0); before initializing the protocol stack in the formal code.

8.2.19 set_mg_ble_dbg_flag

Function prototype: void set_mg_ble_dbg_flag(unsigned char EnableFlag);

Function: In the debug phase, the default serial port (if used) will print some debug information. This function can be invoked to turn this feature off..

Input: EnableFlag sets whether to enable the printing of debug information.

Output: None

Return value: None

Note: Please invoke set_mg_ble_dbg_flag(0); before initializing the protocol stack in the formal code.

8.2.20 radio_standby

Function prototype: void radio_standby(void);

Function: This function enables the RF module to enter the Standby mode.

Input: None

Output: None

Return value: None

Note: The RF module cannot wake up regularly after entering the Standby mode (it can wake up itself and the control module regularly after entering the Stop mode). In this case, IRQ should be provided with a rising edge level signal to wake up the RF module, and PA0 should be provided with a falling edge level to wake up the control module.

8.2.21 GetLTKInfo

Function prototype: unsigned chunsigned char* GetLTKInfo(u8* newFlag);

Function: This function will get the encrypted diversifier (EDIV) of the currently connected device.

Input: 1 means the information of new paired device, 0 means the information of old paired device

Output: None

Return value: u8* EDivData /*2 Bytes*/ (encrypted)

Note: 1. This function can only be invoked when StartEncryption == 1.

2. This function can only be used when the pairing function is enabled.

8.2.22 GetMasterDeviceMac

Function prototype: unsigned char* GetMasterDeviceMac(unsigned char* MacType);

Function: Get the Mac address of the currently or last connected master device.

Input: None.

Output: MacType (0 means common type, other values mean random type)

Return value: master device Mac address (6 bytes)

Note: This function can only be invoked when the mobile phone is connected. It should be invoked in the callback function (voidUsrProcCallback(void)).

9. Porting functional interfaces related to Bluetooth functions

In order to flexibly realize differentiated Bluetooth functions, several interfaces are set in the Bluetooth protocol and callback functions are implemented through the application layer porting. The invocation of all callback functions must not be blocked. For the implementation of specific functions, refer to the corresponding code implementation examples in the SDK release package.

Callback functions mainly include the followings:

1. void gatt_user_send_notify_data_callback(void); the protocol will invoke this callback function at idle time after successful Bluetooth connection.
2. void UsrProcCallback(void); the Bluetooth protocol will periodically invoke this callback function.
3. void ser_prepare_write(unsigned short handle, unsigned char* attValue, unsigned short attValueLen, unsigned short att_offset);
4. void ser_execute_write(void);
   These are two callback functions for writing data into the queue.
5. unsigned char* getDeviceInfoData(unsigned char* len); this callback function is used to get the device name in the GATT.
6. void att_server_rdByGrType(unsigned char pdu_type, unsigned char attOpcode, unsigned short st_hd, unsigned short end_hd, unsigned short att_type);
   callback function for Bluetooth GATT query service
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*/);
   callback function for Bluetooth GATT write operation
8. void server_rd_rsp(unsigned char attOpcode, unsigned short attHandle, unsigned char pdu_type);
   callback function for Bluetooth GATT read operation
9. void ConnectStausUpdate(unsigned char IsConnectedFlag);
   callback function for Bluetooth connection status update.

10. Specific configuration Q&A

(1) How to set the advertising interval?

A: There are two methods: 1) to invoke the interface ble_run() and specify the interval by a parameter; 2) to invoke the interface function ble_set_interval() and set it dynamically.

(2) How to set the transmitting power?

A: Specified by a parameter in the chip initialization interface function radio_initBle().

(3) How to modify the advertising content and device name?

A: The advertising content can be set dynamically through the interface function ble_set_adv_data(). The device name can be modified on demand through the function getDeviceInfoData() in the app.c file.

(4) How to get the Bluetooth address (MAC)?

A: Provided by the output of the chip interface initialization function radio_initBle().

(5) How to customize the characteristic value?

A: Users can assign and define the AttCharList variable of the characteristic value directly with the app.c file.

(6) How to customize a Service?

A: Step1, allocate the Service and corresponding characteristic values directly with the app.c file.

Step2, invoke the callback function att_server_rdByGrType() in the app.c file to response with the Service handle range and UUID.

(7) How to display the user's own software version information with the characteristic value?

A: invoke the callback function server_rd_rsp() in the app.c file to response to the characteristic value of software version and get the user's own software version information.

(8) How to get the connection status?

A: Provided by the input of the callback function ConnectStausUpdate of app.c; zero means disconnected and non-zero means connected.

(9) How to send data via Bluetooth?

A: Firstly, you need to enable Notify of the corresponding characteristic value through the mobile App, and then invoke the interface function sconn_notifydata() from the callback function gatt_user_send_notify_data_callback() in the app.c file. However, the invocation should not be blocked.

(10) How to accept the data received by Bluetooth?

A: The data received by Bluetooth will be notified by the corresponding characteristic value in the callback function ser_write_rsp(), and the user can save it on demand.

(11) How does the module handle the low power condition?

A: By default, the MCU will apply the Stop mode for UART@9600bps data transparent transmission. To facilitate debugging, there is a time delay at the beginning for the program in the SDK code to burn through the SW port. The SW may be disconnected in case of the timeout. At this point, you should re-power the module and connect the SW as soon as possible.

(12) How to implement the key detection function and other user's own logic codes?

A: To ensure the normal operation of Bluetooth, the user should not block the operation codes for a long time.

Take key detection as an example. The key detection function includes: 1) detection; 2) key function execution.

Detection: The system interrupt function can be invoked to scan the corresponding IO status of the key as often as needed and update the key status information.

Execution: In principle, it is not recommended to invoke the interrupt function to execute too many tasks. Since each functional API of Bluetooth is non-reentrant, it is better to execute the corresponding key in the callback function of Bluetooth. If the corresponding function is notify, it can be executed in the callback function gatt_user_send_notify_data_callback(). Other functions (such as serial data processing included in the SDK) can be executed in the callback function UsrProcCallback().

11. Revision history

Revision	Changes	Date
V1.0	Release version	2017/5/26
V1.1	Modified the directory of software codes	2017/7/26
V1.2	Added interface functions of entering RF module Standby mode, getting EDIV, and getting master device Mac	2017/10/12