## 功能

**1.1函数原型：s32 M203C\_BlePowerOn(void);**

## 1.1.1功能:打开蓝牙开关

**1.1.2输入项：void 无输入参数**

**1.1.3输出项：**关闭成功打开返回0，关闭失败返回-1

**1.1.4 代码**

s32 M203C\_BlePowerOn(void)

{

s32 cur\_pwrstate = 0 ;

s32 ret = RIL\_AT\_SUCCESS ;

ret = RIL\_BT\_GetPwrState(&cur\_pwrstate);//查询蓝牙状态

if(RIL\_AT\_SUCCESS != ret) //查询失败

{

ble\_scan\_print("Get BT device power status failed.\r\n");

return;

}

if(1 == cur\_pwrstate//蓝牙已经打开

{

ble\_scan\_print("BT device already power on.\r\n");

}

else if(0 == cur\_pwrstate)//蓝牙处于关闭状态

{

ret = RIL\_BT\_Switch(1);//打开蓝牙开关

if(RIL\_AT\_SUCCESS != ret)//蓝牙打开失败

{

ble\_scan\_print("BT power on failed,ret=%d.\r\n",ret);

return ble\_error;

}

ble\_scan\_print("BT device power on.\r\n");

}

RIL\_BT\_GetPwrState(&cur\_pwrstate);//查询目前的蓝牙状态

ble\_scan\_print("BT power cur\_pwrstate=%d.\r\n",cur\_pwrstate);

return ble\_ok;

**1.1.5测试代码：**

void testbleopen(void)

{

s32 ret;

ret=M203C\_BlePowerOn();

ble\_at\_print("ble\_open=%d",ret);

}

测试方法：执行函数后，打开手机能扫描到蓝牙设备

}

## **1.2函数原型：s32 M203C\_ BlePowerOff (void);**

## 1.2.1功能:关闭蓝牙开关

**1.2.2输入项：void 无输入参数**

**1.2.3输出项：**

关闭成功打开返回0，关闭失败返回-1

**1.2.4 代码**

s32 M203C\_BlePowerOff (void)

{

s32 cur\_pwrstate = 0 ;

s32 ret = RIL\_AT\_SUCCESS ;

ret = RIL\_BT\_GetPwrState(&cur\_pwrstate);//查询蓝牙状态

if(RIL\_AT\_SUCCESS != ret) //查询失败直接返回

{

ble\_at\_print("Get BT device power status failed.\r\n");

return ;

}

if(1 == cur\_pwrstate)//蓝牙为打开状态

{

ble\_at\_print("BT device already power on.\r\n");

ret = RIL\_BT\_Switch(0);//关闭蓝牙开关

if(RIL\_AT\_SUCCESS != ret)//关闭失败，返回-1

{

ble\_scan\_print("BT power on failed,ret=%d.\r\n",ret);

return -1;

}

ble\_scan\_print("BT device power off.\r\n");

}

return 0;

}

**1.2.5测试代码**

void testbleclose(void)

{

s32 ret;

ret=M203C\_BlePowerOff();

ble\_at\_print("ble\_close=%d",ret);

}

测试方法：执行函数后，打开手机扫描不到蓝牙设备了

**1.3函数原型：s32 M203C\_SetBleName (u8 \*name,u8 len);**

## 1.3.1功能:蓝牙命名

**1.3.2输入项：buf [in] : 接收到的数据缓存**

**len [in] : 要接收的数据长度，最大31个字节**

**1.3.3输出项：**

设置成功打开返回0，设置失败返回-1

**1.3.4代码**

s32 M203C\_SetBleName (u8 \*name,u8 len)

{

s32 ret = RIL\_AT\_SUCCESS ;

ret = RIL\_BT\_SetName(name,len);//定义蓝牙名字

if(RIL\_AT\_SUCCESS == ret)//命名成功，返回0

{

ble\_scan\_print("BT device name set successful.\r\n");

}

else//命名失败，返回-1

{

ble\_scan\_print("BT device name set failed,ret=%d.\r\n",ret);

return ble\_error;

}

return ble\_ok;

}

**1.3.5测试代码**

**void testblerename(void)**

**{**

**s32 ret;**

**M203C\_BlePowerOn();**

**ret=M203C\_SetBleName(“00123456”,10);**

**ble\_at\_print("ble\_rename=%d",ret);**

**}**

测试方法：执行此函数后，蓝牙名字就变成了自己命名的名称了

**1.4函数原型： u8 M203C\_BleGatSetadv (u8\* manufacture\_data,u8\* service\_data,u16 service\_uuid);**

## 1.4.1功能:设置BLE广播参数

1.4.2输入项：manufacture\_data [in] : 厂家信息,最大长度31字节

service\_data [in] : 服务信息，最大长度31字节

service\_uuid : 广播包UUID信息，仅支持16位

## 1.4.3输出型：

设置成功返回0，设置失败返回-1

**1.4.4 代码**

**u8 M203C\_BleGatSetadv (u8\* manufacture\_data,u8\* service\_data,u16** service\_uuid)

{

s32 ret;

ble\_setserver();//初始化蓝牙参数,不初始化蓝牙参数，无法进行广播参数初始化

ble\_adv\_data1.services\_data=service\_data;/ /服务信息，最大长度31字节

ble\_adv\_data1.appearance=25;

ble\_adv\_data1.Complete\_UUIDs[0]=service\_uuid;//广播包UUID信息，仅支持16位

return RIL\_BT\_QGatSetadv(Qserver1.gserv\_id,ble\_adv\_data1.appearance,0,manufacture\_data,ble\_adv\_data1.services\_data,ble\_adv\_data1.Complete\_UUIDs[0]);//返回0表示设置成功，返回-1表示设置失败

}

void ble\_setserver()//初始化蓝牙参数，具体参数什么意思，不懂？

{

u8 s\_i=0,c\_i=0,d\_id=0;

Qserver1.gserv\_id[0]='A';

Qserver1.gserv\_id[1]='B';

Qserver1.gserv\_id[2]='\0';

Qserver1.sid=0;

Qserver1.service\_id[Qserver1.sid].service\_uuid=0x6618;

Qserver1.service\_id[Qserver1.sid].num\_handles=5;

Qserver1.service\_id[Qserver1.sid].is\_primary=1;

Qserver1.service\_id[Qserver1.sid].inst=254;

Qserver1.service\_id[Qserver1.sid].cid=0;

c\_i= Qserver1.service\_id[Qserver1.sid].cid;

Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].char\_uuid=0x232a;

Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].inst=2;

Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].prop=58;

Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].permission=17;

Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].did=0;

d\_id =Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].did;

Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].desc\_id[d\_id].desc\_uuid=0x1329;

Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].desc\_id[d\_id].inst=1;

Qserver1.service\_id[Qserver1.sid].char\_id[c\_i].desc\_id[d\_id].permission=17;

Qserver1.service\_id[Qserver1.sid].transport=2;

}

**1.4.5 测试代码**

void testbleSetadv(void)

{

u8 num[1] = {0xDC};

s32 ret;

M203C\_BlePowerOn();

ble\_setserver();

Ble\_Server\_App();

ret=M203C\_BleGatSetadv("5800A100","9612",num);

ble\_at\_print("ble\_setadv=%d",ret);

}

测试方法：先进行蓝牙参数初始化后，设置蓝牙工作状态，开启GATT服务后才可进行ble广播参数初始化，看返回值是否为0，为0则表示设置成功。

## **1.5函数原型： u8 M203C\_BleSetadvSta (u8 sta);**

## 1.5.1功能:设置BLE广播状态

## 1.5.2输入项:sta: 广播状态，0为关闭广播，1为开启广播

## 1.5.3输出项:

设置成功返回0，设置失败返回-1

**1.5.4 代码**

u8 M203C\_BleSetadvSta (u8 sta)

{

u8 i;

char strAT[100];

s32 ret;

RIL\_BT\_Gatsreg\_203C(1,"ABCD");//注册广播名字

Ql\_memset(strAT, 0x0, sizeof(strAT));//清零操作

Ql\_sprintf(strAT, "AT+QBTGATCSCAN=%d,\"%s\"", sta,"ABCD");//传入广播状态和名字

ble\_scan\_print("%s",strAT);

return

Ql\_RIL\_SendATCmd(strAT, Ql\_strlen(strAT), ATRsp\_QBTGatsreg\_Hdlr\_203C,NULL,0);

//返回0表示设置成功，返回-1表示设置失败

}

**1.5.5测试代码**

void testbleSetadvSta(void)

{

u8 num[1] = {0xDC};

s32 ret;

M203C\_BlePowerOn();

ble\_setserver();

Ble\_Server\_App();

M203C\_BleGatSetadv("5800A100","9612",num);

ret=M203C\_BleSetadvSta (1);

ble\_at\_print("ble\_SetadvSta=%d", ret);

}测试方法：打开蓝牙后，ble广播参数初始化成功后，执行此函数，看返回值是否为0，为0表示设置成功，为1表示设置失败

问题：还没找到什么软件工具可以测试广播的数据，晚点找一下

## **1.6函数原型： u8 M203C\_BleSendData (u8 \*buf,u8 len);**

## 1.6.1功能:BLE发送数据

1.6.2 输入项：buf [in] : 发送的数据缓存

\* len [in] : 要发送的数据长度，最大31个字节

**1.6.3输出项：返回实际发送的数据字节长度，发送失败返回-1**

**1.6.4代码**

u8 M203C\_BleSendData (u8 \*buf,u8 len)

{

Hex2Str(buf, Qserver1.wrreq\_param.value, len);//字节流转换为十六进制字符串的另一种实现方式 ，将buf赋值给 Qserver1.wrreq\_param.value

Qserver1.sid = 0;

Qserver1.service\_id[Qserver1.sid].cid = 0;

Qserver1.wrreq\_param.attr\_handle = Qserver1.service\_id[Qserver1.sid].char\_id[Qserver1.service\_id[Qserver1.sid].cid].char\_handle;

Qserver1.wrreq\_param.need\_cnf = 0;

len=sizeof(Qserver1.wrreq\_param.value);

if (Qserver1.conn\_status.connect\_id == 1)//判断蓝牙已经建立连接

{

ret = RIL\_BT\_Gatsind(&Qserver1);//开始发送蓝牙数据

if (RIL\_AT\_SUCCESS != ret)//发送失败，返回-1

{

ble\_at\_print("\r\n<--RIL\_BT\_Gatsind: ret=%d,%d -->\r\n", ret, \_\_LINE\_\_);

return ble\_error;

}

else//发送成功，返回0

{

ble\_at\_print("\r\n<ble\_nus\_string\_send ok>\r\n");

Return len;

}

}

}

**1.6.5测试代码**

void testblesenddata(void)

{

s32 ret;

testbleSetadvSta();

RIL\_BT\_Gatsl(1,&Qserver1);//开始广播

ble\_at\_print("RIL\_BT\_Gatsl =%d\r\n",ret);

ret=203C\_BleSendData(123456,10); //发送蓝牙数据，测试用

ble\_at\_print("ble\_sendata=%d",ret);

}

测试：执行此函数后，手机用FastBLE软件连接上蓝牙，查看返回值，返回0代表发送成功，返回-1代表发送失败，至于怎么看收到的数据，还不清楚

## **1.7函数原型：u8 M203C\_BleReciveData (u8 \*buf,u8 len);**

## 1.7.1功能: BLE接收数据

1.7.2 输入项：buf [in] : 接收到的数据缓存

\* len [in] : 要接收的数据长度，最大31个字节

**1.7.3输出项：**返回实际接收的数据字节长度，未接收到返回0，接收错误返回-1

**1.7.4代码**

u8 M203C\_BleReciveData (u8 \*buf,u8 len)

{

s32 ret;

len=sizeof(Bike\_BLERD\_Buffer);//当203cs接收到数据后，将会通过回调函数存入到Bike\_BLERD\_Buffer

Ql\_memcpy(buf,Bike\_BLERD\_Buffer,len);

mprintf("\r\n-response data:%s\r\n",buf);

ret=RIL\_BT\_Gatsrsp(&Qserver1);//返回应答信号

if(RIL\_AT\_SUCCESS != ret)ble\_at\_print("RIL\_BT\_Gatsrsp %d,%d\r\n",ret,\_\_LINE\_\_);

return len;

}

**1.7.4测试：通过fastble软件，连接上蓝牙之后，发送数据给开发板，从回调函数的log里能看到我发送的数据，只需要把Bike\_BLERD\_Buffer赋值给buf就行**