乐升半导体
LEVETOP

LT5926B

低功耗蓝牙 4.0 + 32 位 MCU 整合芯片

2.4G BLE Transceiver with 32bit MCU

应 用 手 册

V1.1

www.levetop.cn
Levetop Semiconductor Co., Ltd.

Revision History:

Rev	History	Issue Date	Remark
1.0	Initial issue	March 12,2018	Release
1.1	Add FAQ	June 20,2018	Release

目　　录

1. 概述

本文是使用 LT5926B 芯片在 C Core 集成开发环境上实现 BLE 数据双向传输应用的编程指导 ,用于 BLE 应用开发的移植和编程。

2. 软架构

软件代码的目录结构如下:

Demo

├─include

├─lib

└─src

其中蓝牙相应代码放在 src\ble.c，接口放在 include\ble.h，对应库文件在 lib\libBLE-Lib.a

3. BLE 特征值及双向数据通信操作方式

用户如果需要根据自己的需求适当调整 BLE 数据交换的特征值、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 ble.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{
   UINT16 type16;         //type2
   UINT16 handle_rec;      //handle
   UINT8  characterInfo[5];//property1 - handle2 - uuid2
   UINT8  uuid128_idx;     //0xff means uuid16,other is idx of uuid128
}BLE_CHAR;

typedef struct ble_UUID128{
   UINT8  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 16bit 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 7-15 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

3.1 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。需要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. 硬件连接和代码移植说明

请确保 GPIO 复用关系正确并正确初始化、正确初始化 SPI 接口。

GPIO 分配：

LT5926B		蓝牙
INT03	--->	BLE IRQ
SS	--->	BLE CSN
SCK	--->	BLE SCK
MOSI	--->	BLE MOSI
MISO	--->	BLE MISO

- 系统时钟设置 48MHz, SPI 时钟设置 8MHz。
- 移植与蓝牙协议相关的接口函数，具体见文件 ble.h。
- 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。
- 需要移植系统相关的函数如下：
  - ➔ 获取系统 tick 值（单位 1ms）: GetSysTickCount（ ）
- 蓝牙协议运行的入口函数为 ble_run_task()，该函数不会返回。--- 示例代码实现在 app.c
- 获取版本信息函数为 u8* GetBleLibVersion(void); --- 用于区分 lib 不同的打包版本信息（重要）
- 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）。
- 请注意配置好 StackSize，推荐配置 2KB。

5. 依赖关系

- SPI 的 GPIO 复用关系、BLE 基带芯片 IRQ 对应的 GPIO。
- SPI 的速度需要保证 6MHz 或以上（不得高于 10MHz），MCU 速度需要保证 32MHz 或以上。
- 需要 MCU 系统实现 System tick（1ms）并实现对应的调用接口。
- 蓝牙协议栈本身推荐内存需求：ROM 21KB 以上，RAM 1.4KB 以上。

6. 注意事项

- 所有接口函数不得阻塞调用。
- 函数 att_server_rd(...)每次调用发送的数据长度不得超过 20 字节。
- 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。
- UUID 支持 16bit 和 128bit 两种。

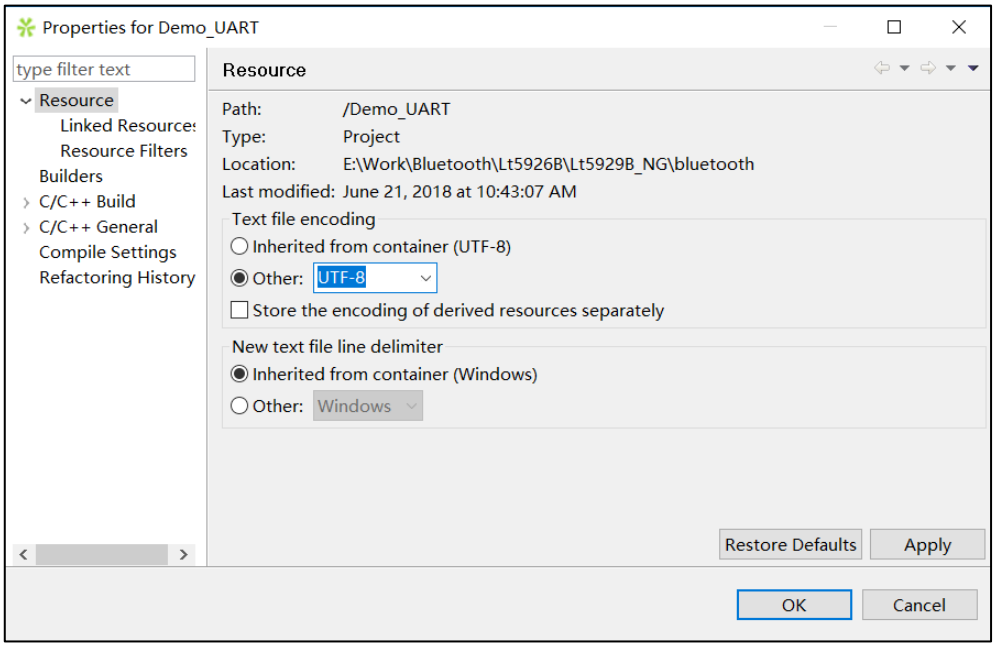
7. 应用开发 FAQ

- **如何解决中文显示乱码**

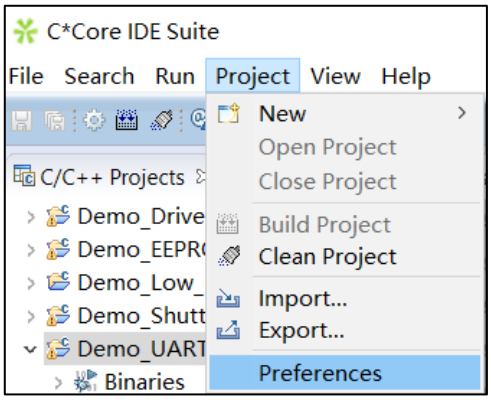
由于安装 IDE 时，字体默认使用了 Cp1252 编码，导致乱码。

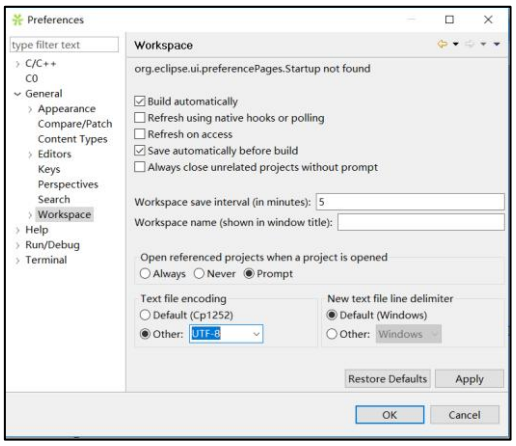
可以通过如下解决本项目乱码：

在 IDE 左边项目列表中选定指定项目，点击右键，选择 Properties 子项，出现如下界面，在 Resource 界面中，直接将 Text file encoding 选择 UTF-8。



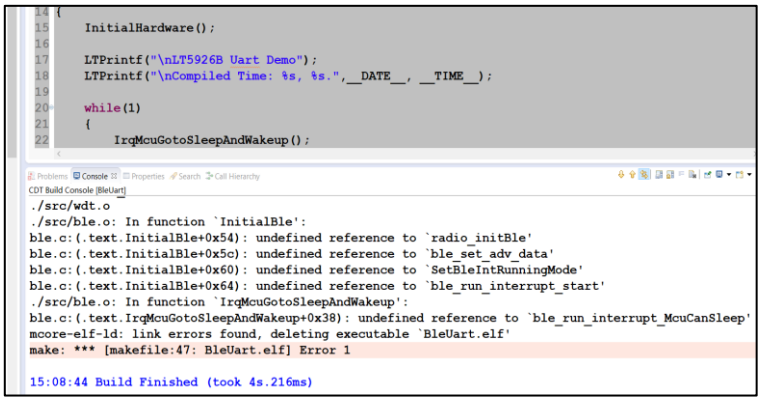
可以通过如下操作，把 IDE 默认文字编码变更为 UTF-8，在 IDE 主界面选择 Preferences 子项,在 General 的 Workspace 子项变更编码。



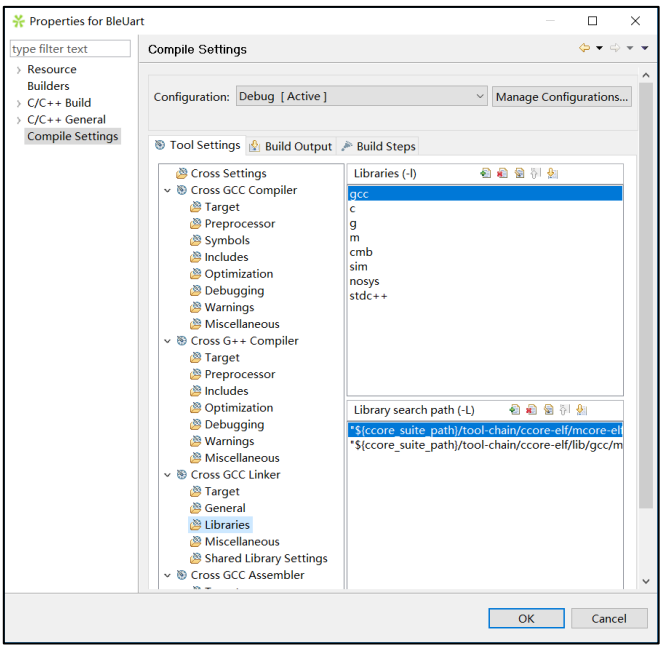


- **无法找到蓝牙库问题**

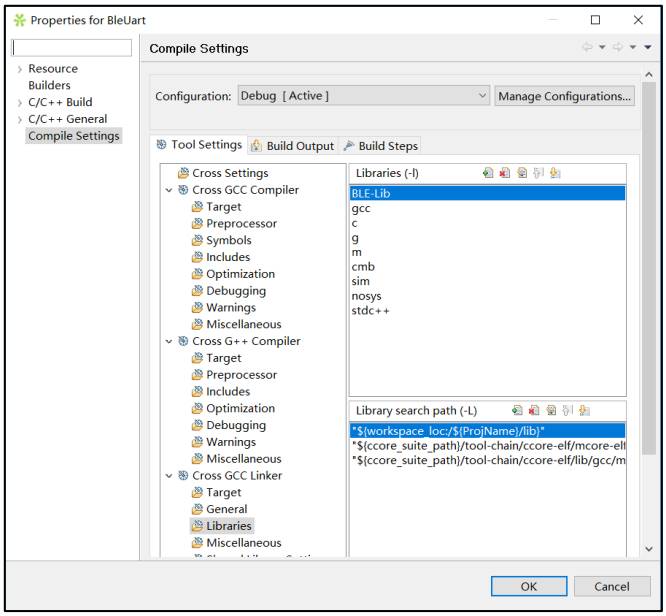
编译工程出现如下提示，是因为没有指定蓝牙库及对应路径



在 IDE 左边项目列表中选定指定项目，点击右键，进入 Properties 子界面，选择 Compile Settings,出现如下界面，



选择当前使用的工程编译模式，如：Configuration: Debug[Active]，在 Tool Settins 分页下，在 Libraries 内添加蓝牙库 BLE-Lib 及路径：${workspace_loc:/${ProjName}/lib}　如下图：



- **关于 1ms 定时器的使用**

由于蓝牙在广播模式下会出现时间累积误差，出于校准需求，蓝牙占用了 PIT0 定时器,并将其配置为 1ms 定期中断。为保证蓝牙可靠运行，在广播模式下，必须开启该定时器，连接上手机后，为避免资料冲突，建议将其关闭。

注意：要避免在该定时器中断内做耗时任务，比如串口输出等

- **关于芯片各外设中断优先级设置**

为保证蓝牙通信时序不给打乱，必须将蓝牙中断 INT03 的优先级设为最高，在初始化芯片时对其修改，参考具体如下：

```
EIC->U8PLSR[2].EICPLSR=1;  //SCI0
EIC->U8PLSR[9].EICPLSR=1;  //PIT0
EIC->U8PLSR[19].EICPLSR=0;//INT03
```

注意：默认中断优先级由高到低是 SCI0> PIT0> INT03

- **如何处理芯片低功耗**

为保证芯片可靠运行，需要按照以下 3 点处理：

- 关闭外设，特别是包含中断处理的
- 关闭所有外设时钟，进入休眠
- 恢复需要使用外设时钟，接着使能对应外设

参考如下：

```
void DisablePeripheral(void)
{
   //关闭 PIT0 定时器
   PIT0->U16PCSR.EN=0;
   //关闭串口
   UART0->U8SCICR2.RE=0;
   UART0->U8SCICR2.TE=0;
}
void EnablePeripheral(void)
{
   //打开需要使用外设时钟
   CLOCK->U32MSCR.MS&=~(1<<3);  //enable ADC
   CLOCK->U32MSCR.MS&=~(1<<4);  //enable PIT0
   CLOCK->U32MSCR.MS&=~(1<<9);  //enable SPI
   CLOCK->U32MSCR.MS&=~(1<<10); //enable PWM0
   CLOCK->U32MSCR.MS&=~(1<<11); //enable PWM1
   CLOCK->U32MSCR.MS&=~(1<<12); //enable EPORT0
   CLOCK->U32MSCR.MS&=~(1<<13); //enable EPORT1
   CLOCK->U32MSCR.MS&=~(1<<15); //enable EFM
   CLOCK->U32MSCR.MS&=~(1<<16); //enable RESET
   CLOCK->U32MSCR.MS&=~(1<<17); //enable WDT
   CLOCK->U32MSCR.MS&=~(1<<18); //enable SCI0
   CLOCK->U32MSCR.MS&=~(1<<19); //enable CCM

   //使能 PIT0 定时器
   PIT0->U16PCSR.EN=1;
  //使能串口
   UART0->U8SCICR2.RE=1;
   UART0->U8SCICR2.TE=1;
}
```

```
void Sleep(void)
{
   DisablePeripheral();
   CLOCK->MSCR=0xffffffff;    //disable all modules
   CLOCK->U32SYNCR.STBYMD=3;
   CLOCK->U32SYNCR.SLEEP=1;  //sleep
   EnablePeripheral();
  }
```

- **如何修改蓝牙设备名称**

在头文件对蓝牙设备名称直接指定，是可以随意改动，例如：

```
#define DeviceInfo      "BleUart"   /*max len is 24 bytes*/
```

- **该在哪个地方处理耗时任务**

考虑到耗时任务容易打断蓝牙事件，导致异常，如果该任务包含对蓝牙操作，只能在函数 void UsrProcCallback(void)内处理。没有的话，也可以放在主循环执行。

- **如何主动断开蓝牙连接，返回广播模式**

只需要在函数 void UsrProcCallback(void)内调用 void ble_disconnect(void)即可。

- **如何上传 Notify 消息**

推荐在函数 void gatt_user_send_notify_data_callback(void) 内处理上传的 Notify 消息 ,注意该函数不得阻塞！！也可以在函数 void UsrProcCallback(void)内处理。上传分两步：

A.指定变量 cur_notifyhandle 为当前 Notify 消息操作句柄，
例如：cur_notifyhandle = 0x0006;

B.填充上传数据，并递交，通过调用函数 unsigned char sconn_notifydata(unsigned char* data, unsigned char len)

- **如何修改蓝牙广播间隔时间**

初始化蓝牙时，调用函数 void ble_run_interrupt_start(unsigned short interv_adv); 通过参数 interv_adv 指定。注意：interv_adv 单位为 0.625ms
例如：interv_adv=320，对应 200ms

- **如何修改蓝牙连接后的间隔时间**

可以在函数 void UsrProcCallback(void)中，调用函数 sconn_GetConnInterval()，来获取当前由手机指定的间隔时间，如果不符合时间要求，通过函数 void SIG_ConnParaUpdateReq(unsigned short IntervalMin, unsigned short IntervalMax, unsigned short slaveLatency, unsigned short TimeoutMultiplier) ,将期望修改的间隔时间提交给手机，手机会在若干周期(视具体手机而定)后，才会修改。

iOS 设备对连接参数设置是有要求的，不符合 iOS 设备连接参数定义的数值是不被接受也就不会变更。参数定义必须满足以下要求：

Interval Max * (Slave Latency + 1) <= 2 s
Interval Max >= 20 ms
Interval Min + 20 ms <= Interval Max
Slave Latency <= 4
ConnSupervisionTimeout <= 6 s
Interval Max * (Slave Latency + 1) * 3 < ConnSupervisionTimeout

注意：
函数 sconn_GetConnInterval(),sconn_GetConnInterval()的间隔时间单位都是 1.25ms，超时参数 TimeoutMultiplier 单位是 10ms。由于不能保证每次都能成功修改间隔时间，可以定期查询再修改，参考如下：

```
if((time200ms%50)==0)//每隔 10S 操作一次
{
 if(runOnce==TRUE)
 {
  if (gConnectedFlag)//已经处于连接状态
  {
   if(sconn_GetConnInterval()<48)//1.25*48=60ms
   {
    //建议不要超过 1s，否则相应会慢，时钟误差积累也会容易断开
    //Interval:(160,176)*1.25ms=(200,220)ms
    //Latency:0
    //Timeout:600*10ms=6s
    SIG_ConnParaUpdateReq(160, 176, 0, 600);
   }
  }
```



```
    runOnce=FALSE;
   }
  }else
  {
   runOnce=TRUE;
  }
```

- **在没有 RTC 外设情况下，如何获取时钟源**

可以利用蓝牙定期广播及通信机制获取，获取到的时间精度依赖设定的广播间隔及修改后的连接间隔，其中，刚连接上的连接间隔时间，手机间存在差异，不能保证时间的准确，需要成功修改后，才认为是可靠的。该过程在函数 void UsrProcCallback(void)内执行，可获得广播累计时间和连接后累计时间，具体参考如下：(其中，广播间隔时间默认使用 200 ms，具体值可以根据项目需要修改。)

```
#define intervalAdvMs 200 //200 ms
volatile UINT32 time200msAdv=0;
volatile UINT32 time200ms=0;
void TimeHandle(void)
{
 static UINT16 intervalMs=0;
 if(gConnectedFlag)//已经处于连接状态
 {
  intervalMs=sconn_GetConnInterval()*5/4;
  timeMs+=intervalMs;  //累加连接时间
  time200msAdv=0;   //清空广播时间
  if(timeMs>=200)
  {
   time200ms++;
   timeMs-=200;
  }
 }
 else
 {
  timeMs+=intervalAdvMs;  //累加广播时间
  time200ms=0;   //清空连接时间
  if(timeMs>=200)
  {
   time200msAdv++;
   timeMs-=200;
```

```
    }
  }
}
```

- **如何关闭蓝牙及重启蓝牙**

为避免打乱蓝牙事件处理，关闭动作只能在函数 void UsrProcCallback(void)内执行，

关闭蓝牙：

```
radio_initBle(TXPWR_0DBM, &ble_mac_addr);
radio_standby();
```

注意：关闭后，蓝牙处于休眠状态，必须要重新开启才能使用

开启蓝牙：

```
radio_initBle(TXPWR_0DBM, &ble_mac_addr);
ble_run_interrupt_start(320); //320*0.625=200 ms
```

注意：如果开启后，Lt5926 全程不需要执行休眠，需要对变量 WakeupDelay=0; 否则不能对其赋值。

8. 参考文档

- LT32A01 技术参考手册 V2.3
- Bluetooth Specification Version 4.0