MG127 代码解析

Application Note

Revision History:

Rev. No.	History	Issue Date	Remark
0.1	Initial issue	Jan 3, 2017	Preliminary
0.2	updata	Feb 7, 2017	Preliminary
0.3	updata	Jun 5, 2017	Preliminary
0.4	updata	Dec 2, 2017	Preliminary

目录

一、工作流程

MCU+BLE 工作的主流程是:

1） 初始化 mcu 系统（设置时钟，gpio，timer ……）
2） 初始化 BLE 芯片（初始化 BLE 芯片请在上电 30ms 后进行）
3） 用户数据处理(按键，ADC，...)
4） 设置发送和接收的次数要求，调用 BLE 收发函数

```
void main()
{
        Init_System();                          //初始化 mcu 资源

        Delay_ms(30);                           //等待 30 毫秒

        BLE_Init();                             //初始化 ble 芯片

        while(1)
        {
            //////user proc                     //用户数据处理

            //////ble rtx api
            txcnt=3; //txcnt=0 for rx only application    //设置发送次数
            rxcnt=0; //rxcnt=0 for tx only application    //设置接收次数，如果只发送，接收次数请设为 0
            BLE_TRX();                          //调用 BLE 收发函数

            Delay_ms(100);                      //根据需要设置循环的间隔
        }
}
```

txcnt、rxcnt 全局变量，表示 BLE 发射和接收的数量。示例如下：

```
txcnt = 6;
rxcnt = 3;
BLE_TRX（）;
```

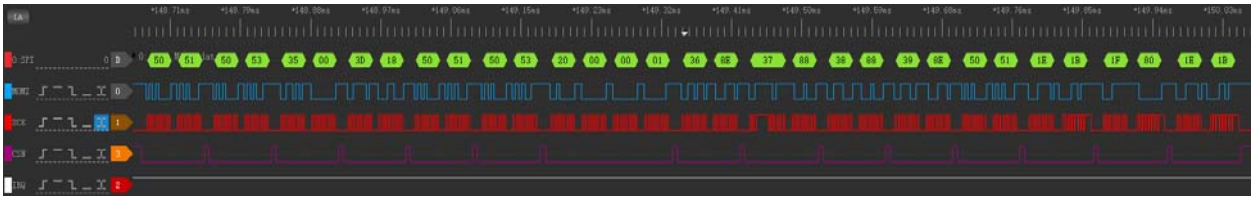
首先执行发射功能 6 次，再执行接收功能 3 次。
客户可以根据需求修改 txcnt、rxcnt 的数量。如只需发射功能，请设置 rxcnt=0;只需要接收功能，请设置 txcnt=0。

二、蓝牙代码

1、BLE 初始化函数

```
void BLE_Init()
{
        uint8_t status;
        uint8_t data_buf[4];
        uint8_t ble_Addr[6];

        SPI_Write_Reg(0x50, 0x51);          //发送 spi 命令 50 51
        SPI_Write_Reg(0x50, 0x53);          //发送 spi 命令 50 53
        SPI_Write_Reg(0x35, 0x00);          //发送 spi 命令 35 00
        SPI_Write_Reg(0x3D, 0x18);          //发送 spi 命令 3d 18
        SPI_Write_Reg(0x50, 0x51);          //发送 spi 命令 50 51

        do{
        SPI_Write_Reg(0x50, 0x53);          //发送 spi 命令 50 53

        data_buf[0] = 0;
        data_buf[1] = 0;
        data_buf[2] = 1;
        SPI_Write_Buffer(0x00, data_buf, 3);    //发送 spi 命令 20 00 00 01

        SPI_Write_Reg(0x36, 0x8e);          //发送 spi 命令 36 8e
        SPI_Write_Reg(0x37, 0x8e);          //发送 spi 命令 37 8e
        SPI_Write_Reg(0x38, 0x88);          //发送 spi 命令 38 88
        SPI_Write_Reg(0x39, 0x8e);          //发送 spi 命令 39 8e

        SPI_Write_Reg(0x50, 0x51);          //发送 spi 命令 50 51

        SPI_Read_Reg(0x1e);                 //发送 spi 命令 1e xx，读取芯片版本信息

        status = SPI_Read_Reg(CHIP_OK);     //发送 spi 命令 1f xx，读取芯片状态信息
        }while(status != 0x80);

        //read chip version
        status = SPI_Read_Reg(0x1e);        //发送 spi 命令 1e xx，读取芯片版本信息
```

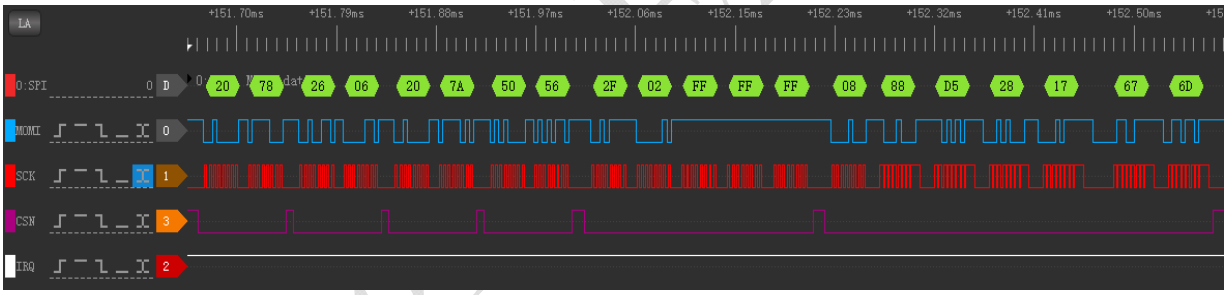
以上程序判断蓝牙芯片 SPI 通信是否正确，读取到数据 0x80 表示芯片通信 OK 。读取芯片版本号打印输出（如果不打印可以不执行最后一句）。对应的 spi 数据波形如下（用 usb 逻辑分析仪抓包）：



2、BLE 初始化函数 –读取 BLE MAC 地址

```
SPI_Write_Reg(0X20, 0x78);              //发送 spi 命令 20 78
SPI_Write_Reg(0X26, 0x06);              //发送 spi 命令 26 06
SPI_Write_Reg(0X20, 0x7a);              //发送 spi 命令 20 7a
SPI_Write_Reg(0x50, 0x56);              //发送 spi 命令 50 56

BLE_Mode_Sleep();                       //发送 spi 命令使 ble 芯片进入睡眠

SPI_Read_Buffer(0x08, ble_Addr, 6);     //发送 spi 命令读取芯片蓝牙地址
```

以上程序代码完成蓝牙发芯片速率等参数设置，并读取蓝牙 MAC 地址（6 个字节，每个蓝牙芯片 MAC 都不同）。取芯片蓝牙地址作为打印输出（如果不打印可以不执行最后一句）。对应的 spi 数据波形如下：

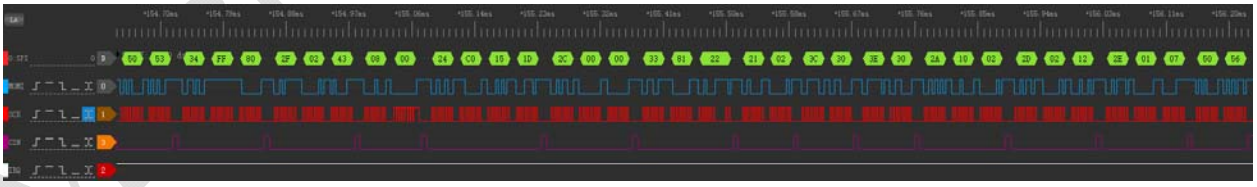


3、BLE 初始化函数 –初始化寄存器

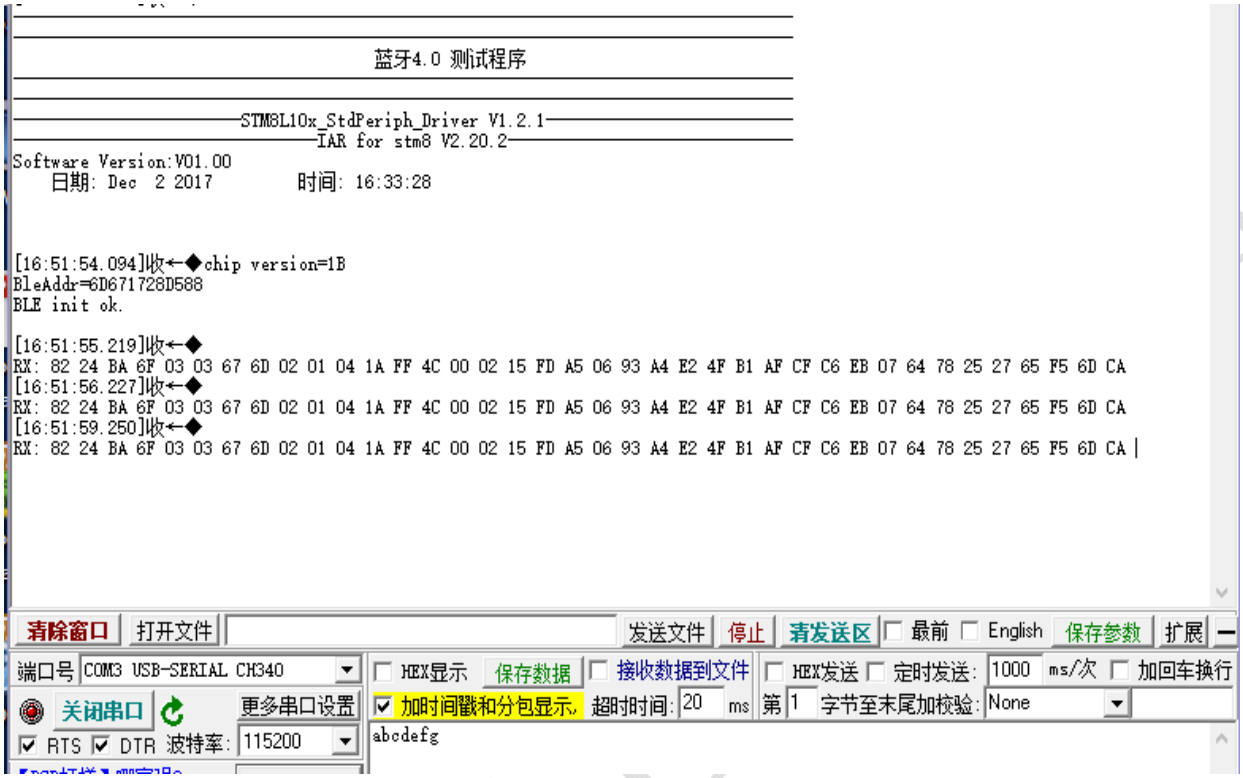
```
SPI_Write_Reg(0x50, 0x53);              //发送 spi 命令 50 53
data_buf[0] = 0xff;
data_buf[1] = 0x80;
SPI_Write_Buffer(0x14,data_buf,2);      //发送 spi 命令 34 ff 80

data_buf[0] = 0x02;
data_buf[1] = BLE_TX_POWER;
SPI_Write_Buffer(0x0f,data_buf,2);      //发送 spi 命令 2f 02 43

data_buf[1] = SPI_Read_Reg(0x08);       //发送 spi 命令 08 xx，读取寄存器值 vv
data_buf[0] = 0xc0;
data_buf[2] = 0x1D;
```

```
    SPI_Write_Buffer(0x4, data_buf, 3);              //发送 spi 命令 24 c0 vv 1d

    data_buf[0] = 0;
    data_buf[1] = 0x00;
    SPI_Write_Buffer(0x0C, data_buf, 2);             //发送 spi 命令 2c 00 00

    data_buf[0] = 0x81;
    data_buf[1] = 0x22;
    SPI_Write_Buffer(0x13, data_buf, 2);             //发送 spi 命令 33 81 22

    SPI_Write_Reg(0x3C, 0x30);                       //发送 spi 命令 3c 30
    SPI_Write_Reg(0x3E, 0x30);                       //发送 spi 命令 3e 30

    data_buf[0] = 0x10;
    data_buf[1] = 0x02;
    SPI_Write_Buffer(0xA, data_buf, 2);              //发送 spi 命令 2a 10 02

    data_buf[0] = 0x02;
    data_buf[1] = 0x12;
    SPI_Write_Buffer(0xD, data_buf, 2);              //发送 spi 命令 2d 02 12

    data_buf[0] = 0x01;
    data_buf[1] = 0x07;
    SPI_Write_Buffer(0xE, data_buf, 2);              //发送 spi 命令 2e 01 07

    SPI_Write_Reg(0x50, 0x56);                       //发送 spi 命令 50 56
}
```

以上程序代码完成射频发射功率、晶体参数等寄存器配置。对应的 spi 数据波形如下：



如使用 AS06-01TB-V3.0 开发板将通过串口显示如下信息。
ble 芯片初始化完成，串口显示（开发板连接 PC）：



注： 1)chip version:1B 芯片版本号。
2)BleAddr:6D671728D588 芯片 MAC 地址，每个芯片都不同。以上数据对应 SPI 波形内容。
3)RX 表示 MG127 接收到周边广播信号内容。

4、BLE 发送接收函数 BLE_TRX()

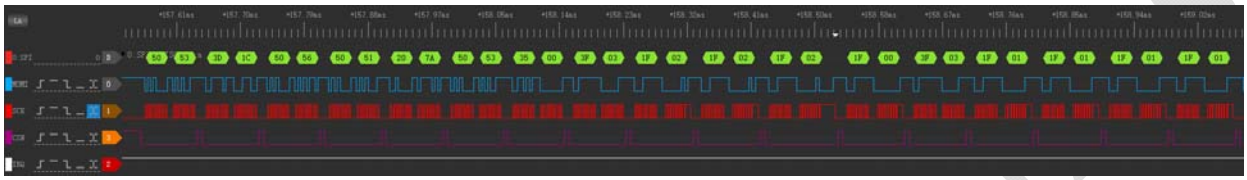
BLE 发送接收函数是关键处理函数，主要操作包括：

设置数据发射通道
设置 BLE 广播数据和包头
清除 BLE 中断标志
唤醒 BLE
设置 BLE 接收超时时间
循环读取 IRQ 电平，低电平读取读取中断标志寄存器，根据中断值进行相应的状态机处理。

```
void BLE_TRX()
{
        uint8_t status = 0;
        uint8_t ch = 37;
        uint8_t data_buf[2];
        uint8_t tmp_cnt = txcnt+rxcnt;
        uint8_t len_pdu = 0;
        uint8_t loop = 0;
```

```
if(tmp_cnt == 0) return;                //输入参数保护

BLE_Set_Xtal(1);                        //发送 spi 命令，晶体准备工作
BLE_Mode_PwrUp();                       //发送 spi 命令，BLE 芯片准备工作
```

以上程序代码发送相关 spi 命令，使晶体和 BLE 芯片进入工作准备。对应的 spi 数据波形如下：



```
#if 1
        //set BLE TX default channel:37.38.39
        SPI_Write_Reg(CH_NO|0X20, ch);                  //发送 spi 命令 21 25，设置通道

        //BLT FIFO write adv_data . max len:31 byte
        SPI_Write_Buffer(W_TX_PAYLOAD, adv_data, LEN_DATA);     //发送 spi 命令 a0 xxxxx,设置数据

        //PDU TYPE: 2 non-connectable undirected advertising
        //set BLT PDU length:adv_data+6 mac adress.
        data_buf[0] = 0x02;
        data_buf[1] = LEN_DATA+LEN_BLE_ADDR;
        SPI_Write_Buffer(ADV_HDR_TX, data_buf, 2);          //发送 spi 命令，设置数据包头

        data_buf[0] = 0xFF;
        data_buf[1] = 0x80;
        SPI_Write_Buffer(INT_FLAG, data_buf, 2);            //发送 spi 命令 2e ff 80，准备中断
#endif
```

以上程序代码完成 4 部分功能：

1） 设置广播初始通道 21 25(37 通道)

```
SPI_Write_Reg(CH_NO|0X20, ch);
```

ch:设置通道号。广播通道有 3 个：37，38，39 一般循环执行。设置一个通道运行，缺点抗干扰能力变弱。

2） 填充发射 FIFO 数据一共 30 个字节

```
SPI_Write_Buffer(W_TX_PAYLOAD, adv_data, LEN_DATA);
```

adv_data：广播数据数组。BLE PDU 格式。 最大 31 字节，数据内容客户可以自定义。

LEN_DATA：adv_data 数组的长度。

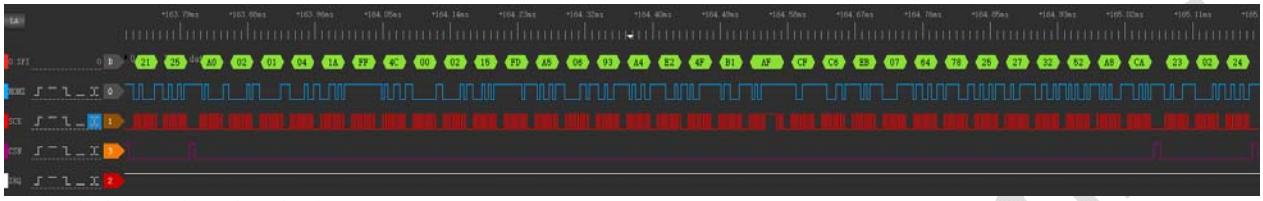
3） 设置蓝牙发送数据和标志

```
//BLE PDU 设置 非连接广播包
data_buf[0] = 0x02;
//BLE PDU 数据长度 广播数据+MAC 地址长度
```

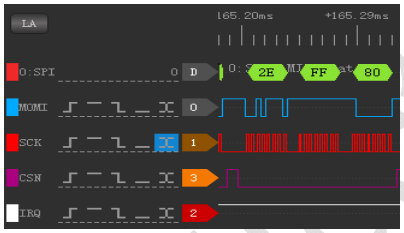
```
data_buf[1] = LEN_DATA+6;//adv_data 数组的长度+MAC 地址长度
//写入寄存器
SPI_Write_Buffer(ADV_HDR_TX, data_buf, 2);
```

此功能必须和 2）配合使用。adv_data 内容必须符合 BLE PDU 格式。

对应的 spi 数据波形如下：



4）清除蓝牙全部中断标志

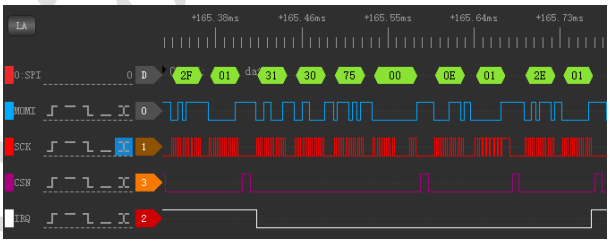


5）唤醒 BLE

```
BLE_Mode_Wakeup();                    //发送 spi 命令，唤醒 ble 芯片
BLE_Set_TimeOut(BLE_RX_TIMEOUT);      //发送 spi 命令，设置接收超时（仅用于接收）
tick = BLE_GUARD_TIME;                //设置中断保护时间，用于异常处理和代码健壮性
```

程序代码完成 BLE 唤醒和设置 RX 超时时间。设置接收超时时间 BLE_RX_TIMEOUT 为 30ms，此操作适用接收功能。

对应的 spi 数据波形如下：



注：1）寄存器写完 2F 01。BLE 进入唤醒状态，IRQ 将改变电平为低电平。清除中断恢复高电平。

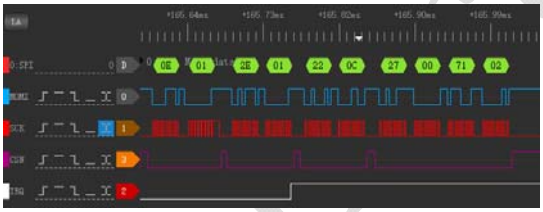
2）BLE_RX_TIMEOUT 主要使用在接收模式中。接收等待时间。最大 65535us。

5）使能接收或发射

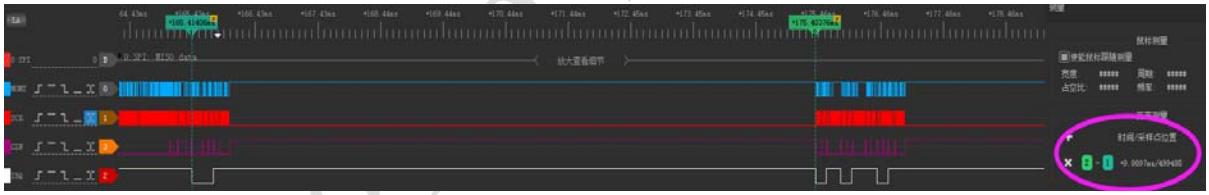
```
while(1)
{
        if (!BLE_IRQ_GET())                          //判读 IRQ 电平（低有效）
        {
          status = SPI_Read_Reg(INT_FLAG);           //发送 spi 命令 0e xx，读取中断值 vv
          SPI_Write_Reg(INT_FLAG|0X20, status);      //发送 spi 命令 2e vv，清除中断

          if(INT_TYPE_WAKEUP & status)//wakeup       //根据中断值 vv 进行判断
          {                                          //这里面是唤醒中断处理
            if(txcnt > 0){
```

```
            txcnt --;
            SPI_Write_Reg(MODE_TYPE|0X20,                    //发送 spi 命令 22 0c,进入发射模式
RADIO_MODE_ADV_TX);
            BLE_Set_StartTime(BLE_START_TIME);               //设置发射开始时间
        }else if(rxcnt > 0){
            rxcnt --;
            SPI_Write_Reg(MODE_TYPE|0X20,                    //发送 spi 命令 22 09,进入接收模式
RADIO_MODE_ADV_RX);
            BLE_Set_StartTime(BLE_START_TIME);               //设置接收开始时间
        }
        continue; //goto while(1)                            //goto while(1)
    }
```

进入状态机处理。IRQ 为低，读取寄存器 0x0E 内容，然后写入清除中断，根据读取内容判断中断类型。如果判断是“唤醒中断”，根据 txcnt/rxcnt 判断执行 BLE 发射或接收功能，后设置 starttime 时间。对应的 spi 数据波形如下：



注：starttime 设置时间长度默认 10ms，此内容和 SPI 时钟速度有关。设置太小将导致数据收/发错误。通过 usb 逻辑分析仪抓包确定合适的 starttime，如下图：

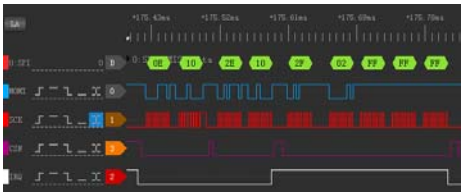


Starttime 从 IRQ 电平下降沿开始 到 第二个下降沿 一共：10ms

starttime 时间计算方法如图：

27 00 71 02:写入 starttime 10ms。Starttime 和 SCK 速度有关，其中最小时间计算就是下图两个标尺的差值。根据波形地址最小时间是：604us。如果 SCK 速度加快此处时间进一步减小。

下图完成 BLE 发射输出。可通过手机 APP ” AndroidScale_draw2.apk”或 MG127 模块接收发射数据。发射完成后 BLE 将进入睡眠模式。

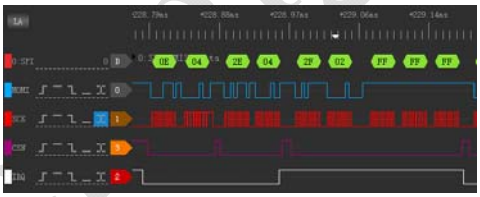


6）接收数据处理

```
BLE_Mode_Sleep();                    //对除了唤醒中断以外的中断，发送 spi 命令使芯片睡眠
if(INT_TYPE_PDU_OK & status){        //这里面是接收中断处理。对于没有接收的应用不需要这里面的处理
  BLE_Get_Pdu(rx_buf, &len_pdu);     //发送 spi 命令，读取接收数据
}
```

BLE_Get_Pdu(rx_buf, &len_pdu)；中读取接收数据 ble header 04 82 24、读取 ble mac 地址 0B BA 6F 03 03 67 6D、读取 ble adv 广播数据内容。
串口工具打印接收数据内容和逻辑分析仪抓取判断接收数据是否正确。

在接收模式等待 BLE_RX_TIMEOUT（30ms）时间未接收到任何数据，BLE 出发时间溢出中断 0x04，BLE 将进入 SLEEP 模式。如右图：



6）睡眠处理

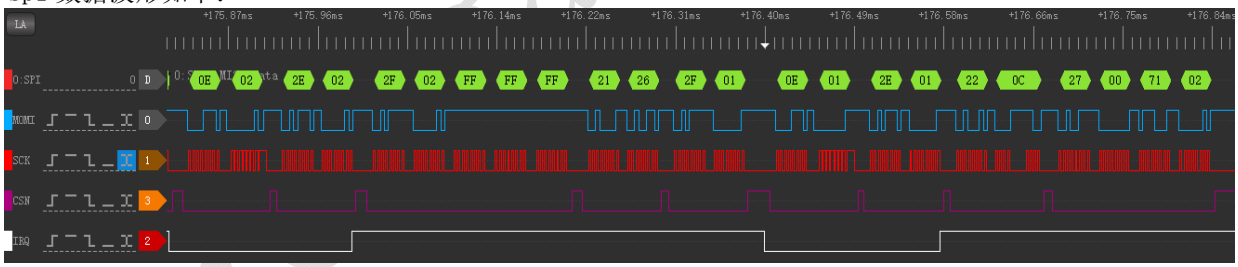
芯片处理完成以下状态后将进入睡眠模式：

- 发射完成
- 接收数据读取完成
- 接收数据超时
- 接收数据错误后

```
if(INT_TYPE_SLEEP & status)          //这里面是睡眠中断处理
{
  tick = BLE_GUARD_TIME;             //重置中断保护时间
```

```
        if (++ch > 39){
          ch = 37;
        }
        SPI_Write_Reg(CH_NO|0X20, ch);          //发送 spi 命令 21 2x，更新通道设置

        tmp_cnt --;                             //发送/接收次数递减
        if(tmp_cnt == 0){                       //发送/接收次数用完
          BLE_Set_Xtal(0);                      //发送 spi 命令，晶体关闭
          BLE_Mode_PwrDn();                     //发送 spi 命令，芯片关闭
          break; //exit from while(1)           //退出函数
        }
        else                                    //继续发送/接收
          BLE_Mode_Wakeup();                    //发送 spi 命令，唤醒芯片
        }
      }
      else if(tick == 0){                       //中断异常，触发保护机制
        BLE_Mode_Sleep();                       //发送 spi 命令，使芯片睡眠
      }
    }
}
```

以上程序代码对睡眠中断进行处理，判断发射/接收完成是否退出函数，实现异常保护。睡眠中断处理对应的 spi 数据波形如下：



为加强软件的健壮性必须异常保护。异常保护通过定时器 tick 实现。在进入发送接收函数和处理睡眠中断时设置 tick=BLE_GUARD_TIME，在发送接收函数主体里发现 tick 变为 0 则说明发生异常。异常由于晶体干扰引起。检测异常发送 spi 命令使芯片进入睡眠，芯片会重新设置晶体进入正常工作流程。

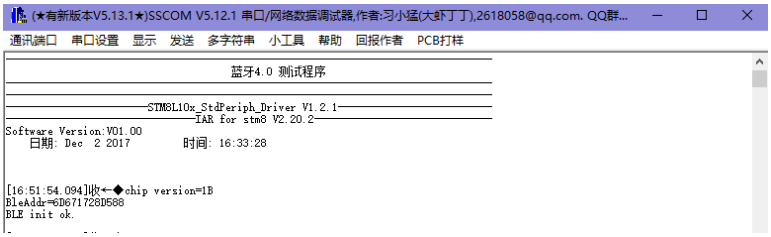
注意：IRQ 电平变化时，读取 0E 寄存器判断 BLE 工作状态，把读取状态再次写入寄存器完成清除状态标志功能。IRQ 恢复到高电平。

三、接收和发射数据验证

1、发射数据验证

安卓手机安装软件" AndroidScale_draw2.apk" 打开蓝牙接收广播数据。
发射数据内容如下：
uint8_t adv_data[30] = {0x02,0x01,0x04,

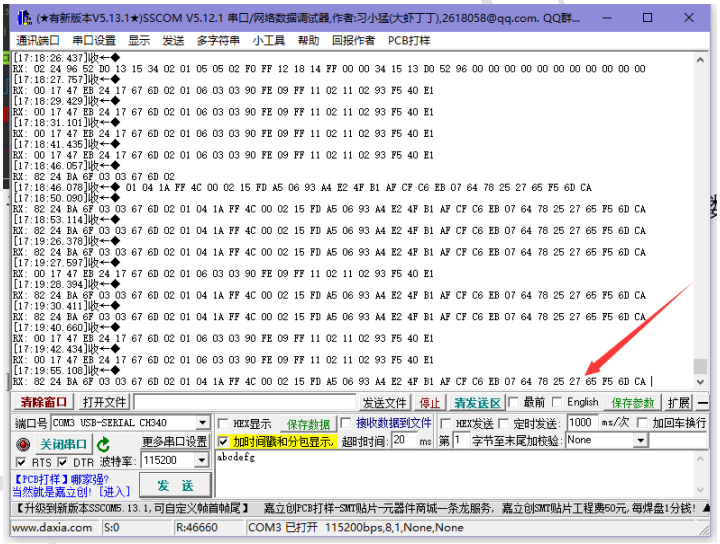
0x1a,0xff,0x4c,0x00,2,0x15,0xfd,0xa5,0x06,0x93,0xa4,0xe2,0x4f,0xb1,0xaf,0xcf,0xc6,0xeb,0x07,0x64,0x78,0x25, 0x27,0x32,0x52,0xa8, 0xCA};



手机可以获取模块 MAC 地址和发射数据内容。数据完全一致说明接收正确。

2、接收数据验证

使用另一个 MG127 或第三方 设备发射广播数据，通过 MG127 模块接收。串口工具打印接收数据内容和逻辑分析仪抓取相同。也可使用" AndroidScale_draw2.apk"软件接收的广播数据与 MG127 接收数据 SPI 对比验证。



附录:

BLE 广播数据

Preamble(1byte)	Access Address(4bytes)	PDU(2 to 39 bytes)	CRC(3bytes)

Preamble 前导码：8bit 01010101 或 10101010，Access Address 第一位决定具体内容。
Access Address 接入地址：固定值:0x8E89BED6 。确定前导码为：01010101
PDU 协议数据单元：MG127 填充或接收到的数据
CRC ：进行 PDU 部分校验

PDU 说明

Header (16bits)	Payload(37 bytes)

Header 说明

PDY Type(4bits)	RFU(2bits)	TxAdd(1bit)	RxAdd (1bit)	Length(6bits)	RFU(2bits)

PDUTYPE:广信道的数据包格式， type 指示的是广播信道的包类型。
RFU:为目前没有使用
TxAdd：为发送者的地址类型 0 表示公共地址，1 表示随机地址
RxAdd：为接收者的地址类型。
Length:指示其后的负载(payload)的长度。Payload 字节长度由它设置。
MG127 设置示例：

```
    //PDU TYPE: 2  non-connectable undirected advertising . tx add:random address
    //set BLT PDU length:adv_data+6 mac adress.
    data_buf[0] = 0x02;
    data_buf[1] = LEN_DATA+LEN_BLE_ADDR;
    SPI_Write_Buffer(ADV_HDR_TX, data_buf, 2);
```

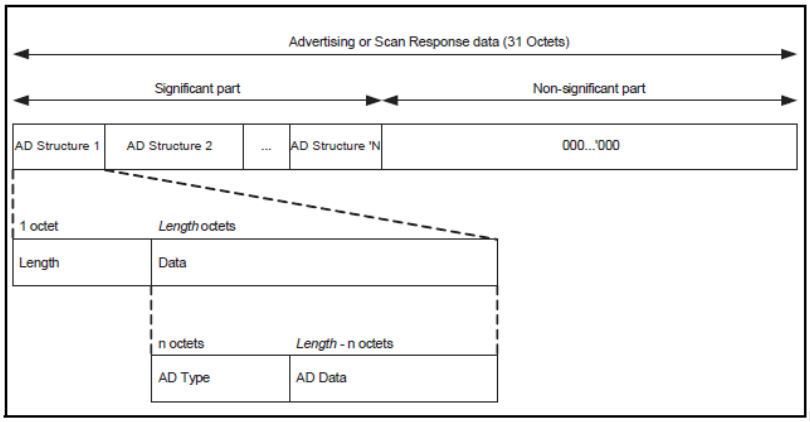
Payload

AdvA(6bytes)	AdvData(0-31bytes)

AdvA : 广播地址 MG127 MAC 地址
AdvData： 最大 31 个字节。
示例：
设置 adv data 30 字节

```
uint8_t adv_data[30] = {0x02,0x01,0x04, 0x1a,0xff,0x4c,0x00,2,0x15,
0xfd,0xa5,0x06,0x93,0xa4,0xe2,0x4f,0xb1,0xaf,0xcf,0xc6,0xeb,0x07,0x64,0x78,0x25,
0x27,0x32,0x52,0xa8, 0xCA};
```

AdvData 格式 （MG127 数据内容就是 AdvData）



AD type 如图：

```
#define BLE_GAP_AD_TYPE_FLAGS                                0x01 /**< Flags for discoverability. */
#define BLE_GAP_AD_TYPE_16BIT_SERVICE_UUID_MORE_AVAILABLE    0x02 /**< Partial list of 16 bit service UUIDs. */
#define BLE_GAP_AD_TYPE_16BIT_SERVICE_UUID_COMPLETE          0x03 /**< Complete list of 16 bit service UUIDs. */
#define BLE_GAP_AD_TYPE_32BIT_SERVICE_UUID_MORE_AVAILABLE    0x04 /**< Partial list of 32 bit service UUIDs. */
#define BLE_GAP_AD_TYPE_32BIT_SERVICE_UUID_COMPLETE          0x05 /**< Complete list of 32 bit service UUIDs. */
#define BLE_GAP_AD_TYPE_128BIT_SERVICE_UUID_MORE_AVAILABLE   0x06 /**< Partial list of 128 bit service UUIDs. */
#define BLE_GAP_AD_TYPE_128BIT_SERVICE_UUID_COMPLETE         0x07 /**< Complete list of 128 bit service UUIDs. */
#define BLE_GAP_AD_TYPE_SHORT_LOCAL_NAME                     0x08 /**< Short local device name. */
#define BLE_GAP_AD_TYPE_COMPLETE_LOCAL_NAME                  0x09 /**< Complete local device name. */
#define BLE_GAP_AD_TYPE_TX_POWER_LEVEL                       0x0A /**< Transmit power level. */
#define BLE_GAP_AD_TYPE_CLASS_OF_DEVICE                      0x0D /**< Class of device. */
#define BLE_GAP_AD_TYPE_SIMPLE_PAIRING_HASH_C                0x0E /**< Simple Pairing Hash C. */
#define BLE_GAP_AD_TYPE_SIMPLE_PAIRING_RANDOMIZER_R          0x0F /**< Simple Pairing Randomizer R. */
#define BLE_GAP_AD_TYPE_SECURITY_MANAGER_TK_VALUE            0x10 /**< Security Manager TK Value. */
#define BLE_GAP_AD_TYPE_SECURITY_MANAGER_OOB_FLAGS           0x11 /**< Security Manager Out Of Band Flags. */
#define BLE_GAP_AD_TYPE_SLAVE_CONNECTION_INTERVAL_RANGE      0x12 /**< Slave Connection Interval Range. */
#define BLE_GAP_AD_TYPE_SOLICITED_SERVICE_UUIDS_16BIT        0x14 /**< List of 16-bit Service Solicitation UUIDs. */
#define BLE_GAP_AD_TYPE_SOLICITED_SERVICE_UUIDS_128BIT       0x15 /**< List of 128-bit Service Solicitation UUIDs. */
#define BLE_GAP_AD_TYPE_SERVICE_DATA                         0x16 /**< Service Data - 16-bit UUID. */
#define BLE_GAP_AD_TYPE_PUBLIC_TARGET_ADDRESS                0x17 /**< Public Target Address. */
#define BLE_GAP_AD_TYPE_RANDOM_TARGET_ADDRESS                0x18 /**< Random Target Address. */
#define BLE_GAP_AD_TYPE_APPEARANCE                           0x19 /**< Appearance. */
#define BLE_GAP_AD_TYPE_ADVERTISING_INTERVAL                 0x1A /**< Advertising Interval. */
#define BLE_GAP_AD_TYPE_LE_BLUETOOTH_DEVICE_ADDRESS          0x1B /**< LE Bluetooth Device Address. */
#define BLE_GAP_AD_TYPE_LE_ROLE                              0x1C /**< LE Role. */
#define BLE_GAP_AD_TYPE_SIMPLE_PAIRING_HASH_C256             0x1D /**< Simple Pairing Hash C-256. */
#define BLE_GAP_AD_TYPE_SIMPLE_PAIRING_RANDOMIZER_R256       0x1E /**< Simple Pairing Randomizer R-256. */
#define BLE_GAP_AD_TYPE_SERVICE_DATA_32BIT_UUID              0x20 /**< Service Data - 32-bit UUID. */
#define BLE_GAP_AD_TYPE_SERVICE_DATA_128BIT_UUID             0x21 /**< Service Data - 128-bit UUID. */
#define BLE_GAP_AD_TYPE_3D_INFORMATION_DATA                  0x3D /**< 3D Information Data. */
#define BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA           0xFF /**< Manufacturer Specific Data. */
```

MG127 发射数据分析：

Len	AD TYPE	data
0x02	0x01（flag）	0x04 #define GAP_ADTYPE_FLAGS_LIMITED 0x01 //!< Discovery Mode: LE Limited Discoverable Mode #define GAP_ADTYPE_FLAGS_GENERAL 0x02 //!< Discovery Mode: LE General Discoverable Mode #define GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED 0x04 //!< Discovery Mode: BR/EDR Not Supported
0x1A	0xFF（Manufacturer Specific Data）	0x4c,0x00 :apple 公司标识 0x4C00 2,0x15, :ibceacon 标识 0xfd,0xa5,0x06,0x93,0xa4,0xe2,0x4f,0xb1,0xaf,0xcf,0xc6,0xeb,0x07,0x64,0x78,0x25, :128bits UUID 0x27,0x32, :MAJOR 0x52,0xa8, :MINOR 0xCA :Measured Power

使用专门抓包工具抓取数据和 MG127 发射数据一致。如图：