青风带你玩蓝牙 nRF51822 系列教程

----------作者：青风

出品论坛：　www.qfv8.com　青风电子社区

作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：346518370
硬件平台：青云 QY-nRF51822 开发板

2.36 蓝牙广播初始化分析

对应蓝牙广播的初始化，很多读者无法理解其设置和运行过程，不知道相关的基础知识，这一节将详细的进行讨论。

并且通过分析基本原理，设置一个固定广播模式，本例在匹配的 SDK10.0 的蓝牙样例的例子基础上就行编写，使用的协议栈为：s110_nrf51_8.0.0。

1：nRF51822 蓝牙 BLE 广播初始化：

1.1 BLE 广播初始化步骤：

```
static void advertising_init(void)
{
    uint32_t      err_code;
    ble_advdata_t advdata;

    // Build advertising data struct to pass into @ref ble_advertising_init.
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type = BLE_ADVDATA_FULL_NAME;//广播时的名称显示
    advdata.include_appearance = true;
    advdata.flags= BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;//蓝牙设备模式
    advdata.uuids_complete.uuid_cnt=sizeof(m_adv_uuids)/ sizeof(m_adv_uuids[0]);//UUID
    advdata.uuids_complete.p_uuids  = m_adv_uuids;

    ble_adv_modes_config_t options = {0};
```

```
    options.ble_adv_fast_enabled  = BLE_ADV_FAST_ENABLED;//广播类型
    options.ble_adv_fast_interval = APP_ADV_INTERVAL;//广播间隔
    options.ble_adv_fast_timeout  = APP_ADV_TIMEOUT_IN_SECONDS;//广播超时

    err_code = ble_advertising_init(&advdata, NULL, &options, on_adv_evt, NULL);//广播参数
    APP_ERROR_CHECK(err_code);
}
```

广播初始化实际上就是初始化两个结构体，一个是&advdata 广播数据，一个是&options 选择项，下面我们就具体连对广播初始化进行分析。

&advdata 广播数据定义了广播的基本参数，如下图所示，比如名称类型，最短名称长度，设备模式，发射功率等等，不是所有参数你都需要在初始化函数中进行设置。只需要选择自己需要设置的，比如我们之前讲发射功率的时候就需要添加 p_tx_power_level 参数。

```
 *       setting the advertising data. */
typedef struct
{
    ble_advdata_name_type_t      name_type;              /**< Type of device name. */
    uint8_t                      short_name_len;         /**< Length of short device name (if short type is s
    bool                         include_appearance;     /**< Determines if Appearance shall be included. */
    uint8_t                      flags;                  /**< Advertising data Flags field. */
    int8_t *                     p_tx_power_level;       /**< TX Power Level field. */
    ble_advdata_uuid_list_t      uuids_more_available;   /**< List of UUIDs in the 'More Available' list. */
    ble_advdata_uuid_list_t      uuids_complete;         /**< List of UUIDs in the 'Complete' list. */
    ble_advdata_uuid_list_t      uuids_solicited;        /**< List of solicited UUIDs. */
    ble_advdata_conn_int_t *     p_slave_conn_int;       /**< Slave Connection Interval Range. */
    ble_advdata_manuf_data_t *   p_manuf_specific_data;  /**< Manufacturer specific data. */
    ble_advdata_service_data_t * p_service_data_array;   /**< Array of Service data structures. */
    uint8_t                      service_data_count;     /**< Number of Service data structures. */
    bool                         include_ble_device_addr;/**< Determines if LE Bluetooth Device Address shall
    ble_advdata_le_role_t        le_role;                /**< LE Role field. Included when different from @re
    ble_advdata_tk_value_t *     p_tk_value;             /**< Security Manager TK value field. Included when
    uint8_t *                    p_sec_mgr_oob_flags;    /**< Security Manager Out Of Band Flags field. Inclu
} ble_advdata_t;
```

我们分析一下 BLE 样例里给的一般需要的几个设置项：

1. advdata.name_type = BLE_ADVDATA_FULL_NAME;//广播时的名称显示

设置广播里，广播名称类型，类型就三种：

```
typedef enum
{
    BLE_ADVDATA_NO_NAME,
    BLE_ADVDATA_SHORT_NAME,
    BLE_ADVDATA_FULL_NAME
} ble_advdata_name_type_t;
```

没有名称，短名称，全名。这个可以更具自己需要设置。

2.advdata.flags= BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;//蓝牙设备模式

这个设置蓝牙设备的模式,选项如下：

```
 * @{ */
#define BLE_GAP_ADV_FLAG_LE_LIMITED_DISC_MODE         (0x01)   /**< LE Limited Discoverable Mode. */
#define BLE_GAP_ADV_FLAG_LE_GENERAL_DISC_MODE         (0x02)   /**< LE General Discoverable Mode. */
#define BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED         (0x04)   /**< BR/EDR not supported. */
#define BLE_GAP_ADV_FLAG_LE_BR_EDR_CONTROLLER         (0x08)   /**< Simultaneous LE and BR/EDR, Controller. */
#define BLE_GAP_ADV_FLAG_LE_BR_EDR_HOST               (0x10)   /**< Simultaneous LE and BR/EDR, Host. */
#define BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE   (BLE_GAP_ADV_FLAG_LE_LIMITED_DISC_MODE | BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED)
#define BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE   (BLE_GAP_ADV_FLAG_LE_GENERAL_DISC_MODE | BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED)
/**@} */
```

Value	Description	Bit	Information
0x01	Flags	0	LE Limited Discoverable Mode
		1	LE General Discoverable Mode
		2	BR/EDR Not Supported (i.e. bit 37 of LMP Extended Feature bits Page 0)
		3	Simultaneous LE and BR/EDR to Same Device Capable (Controller) (i.e. bit 49 of LMP Extended Feature bits Page 0)
		4	Simultaneous LE and BR/EDR to Same Device Capable (Host) (i.e. bit 66 of LMP Extended Feature bits Page 1)
		5..7	Reserved

Table 18.1: Flags

如上表所示，蓝牙 nrf51822 实际上设置的蓝牙类型是严格限定的，BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE 为两种情况 1 和 2，**也就是说 LE 普通模式和不支持 BR/EDR 模式。**

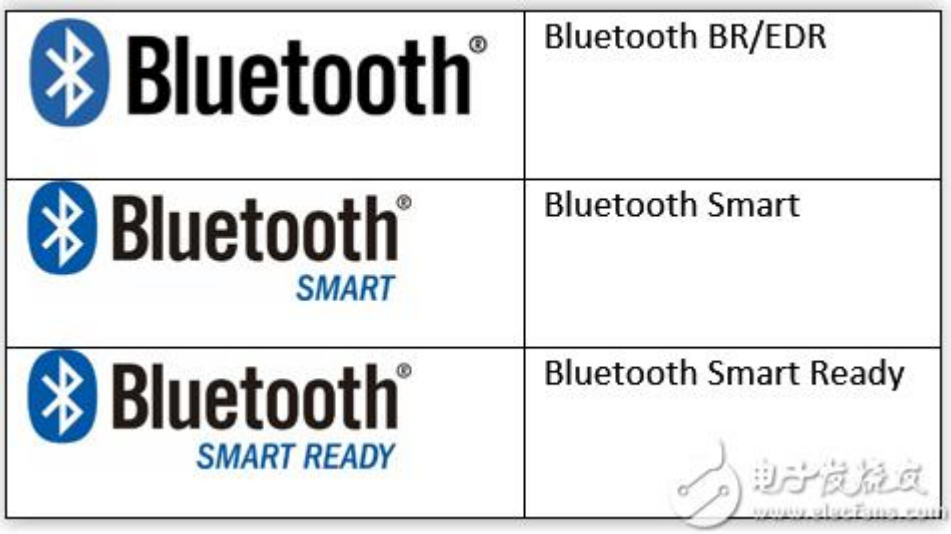
LE 和 BR/EDR 这两个英文解释一下，LE 也就是 BLE 后面两个字 LE，可以把 BLE 写成：Bluetooth Smart 也就是低功耗蓝牙。

蓝牙 BR/EDR（蓝牙基本速率/增强数据率），低功耗是 Bluetooth Smart 的亮点之一。Bluetooth Smart 设备仅靠一颗纽扣电池就能运行数月甚至数年之久。Bluetooth Smart 灵活的配置也让应用能够更好地管理连接间隔（connection interval），以优化接收机的工作周期。对于蓝牙 BR/EDR，由于其数据吞吐量更高，功耗也会相应增加。BR/EDR 配置文件包括： 耳机（HSP）、对象交换（OBEX）、音频传输（A2DP）、视频传输（VDP） 和文件传输（FTP）。也就是一些大的数据的传输。

经常会有人问我，如果一个蓝牙耳机兼容蓝牙 4.x，是否意味着这副耳机是低功耗的呢？Bluetooth Smart 可否用于音频应用？

Bluetooth 4.x 核心规格中有一卷是低功耗控制器，还有一卷是蓝牙 BR/EDR 控制器。通常情况下，如果耳机支持 4.x，则兼容 4.x BR/EDR 规格，而不兼容低功耗规格或 Bluetooth Smart。可以通过辨认产品包装上的 Bluetooth Smart 商标确认其是否为 Bluetooth Smart 产品。**所以注意 nrf51822 是无法传输音频的。**

除了这个之外，蓝牙商标其实有三种，可用来区分产品所采用的蓝牙类型。制造商会在产品本身或其包装上使用这些商标。



最后一种称为：Bluetooth Smart Ready 设备可以接收来自其他蓝牙设备的数据、这些数据可以被 Bluetooth Smart Ready 设备上的应用转化成有用的信息，比如智能手机、个人电脑、平板电脑等都是 Bluetooth Smart Ready 设备。

3.advdata.uuids_complete.uuid_cnt=sizeof(m_adv_uuids)/sizeof(m_adv_uuids[0]);//UUID

advdata.uuids_complete.p_uuids = m_adv_uuids;

设置 UUID 类型，关于 UUID 的设置在《青风带你学蓝牙：蓝牙 UUID 设置》这篇文章里有详细的介绍，这里就不重复了。

再来看&options 选择项， options 为 ble_adv_modes_config_t 结构体，设置了如下图所示的相关参数：

```
/**@brief Options for the different advertisement modes.
 *
 * @details This structure is used to enable or disable advertising modes and to configure time-out
 *          periods and advertising intervals.
 */
typedef struct
{
    bool     ble_adv_whitelist_enabled;       /**< Enable or disable use of the whitelist. */
    bool     ble_adv_directed_enabled;        /**< Enable or disable direct advertising mode. */
    bool     ble_adv_directed_slow_enabled;   /**< Enable or disable direct advertising mode. */
    uint32_t ble_adv_directed_slow_interval;  /**< Advertising interval for directed advertising. */
    uint32_t ble_adv_directed_slow_timeout;   /**< Time-out (number of tries) for direct advertising. */
    bool     ble_adv_fast_enabled;            /**< Enable or disable fast advertising mode. */
    uint32_t ble_adv_fast_interval;           /**< Advertising interval for fast advertising. */
    uint32_t ble_adv_fast_timeout;            /**< Time-out (in seconds) for fast advertising. */
    bool     ble_adv_slow_enabled;            /**< Enable or disable slow advertising mode. */
    uint32_t ble_adv_slow_interval;           /**< Advertising interval for slow advertising. */
    uint32_t ble_adv_slow_timeout;            /**< Time-out (in seconds) for slow advertising. */
}ble_adv_modes_config_t;
```

程序设置为： options.ble_adv_fast_enabled = BLE_ADV_FAST_ENABLED;//广播类型

这个里面关键的问题就是广播类型的选择，广播类型有几种，如下图所示（英文解释具体查看官方 SDK 说明）：

Advertising modes

During initialization of the module, you decide

Advertising mode	
Direct	After disconnecting, the applic connected to one peer and seaml
Fast	The application rapidly adverti
Slow	The application advertises to s causes less traffic for other w
Idle	The application stops advertisi

解释一下：

Direct 模式及直连模式，利用的就是 ble 中的直连广播，该模式是为了快速重连上刚刚断开的设备，比如利用在快速重连上意外断开的设备，已达到无缝恢复的目的。

Fast 模式：就是普通的广播，不过连接间隔我们可以设置的快一点。

Slow 模式：普通广播，连接间隔设置的慢一点

Idle 模式：停止广播。

程序里设置为 options.ble_adv_fast_enabled = BLE_ADV_FAST_ENABLED;//广播类型，为快速广播。那么开始广播就是直接进入快速广播。

那么整个广播事件就抛给了广播回调函数，如下所示：

```
353
354 /**@brief Function for dispatching a BLE stack event to all modules with a BLE stack event handler.
355  *
356  * @details This function is called from the BLE Stack event interrupt handler after a BLE stack
357  *          event has been received.
358  *
359  * @param[in] p_ble_evt  Bluetooth stack event.
360  */
361 static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
362 {
363     dm_ble_evt_handler(p_ble_evt);
364     ble_conn_params_on_ble_evt(p_ble_evt);
365     bsp_btn_ble_on_ble_evt(p_ble_evt);
366     on_ble_evt(p_ble_evt);
367     ble_advertising_on_ble_evt(p_ble_evt);
368     /*YOUR_JOB add calls to _on_ble_evt functions from each service your application is using
369     ble_xxs_on_ble_evt(&m_xxs, p_ble_evt);
370     ble_yys_on_ble_evt(&m_yys, p_ble_evt);
371     */
372 }
373
```

这个是回调函数里必须有的一个部分，ble_advertising_on_ble_evt(p_ble_evt)深入到这个函数内部详细看下，这里有一个广播超时处理，首先是蓝牙协议栈 4.1 开始分成了低速定向广播（Low Duty Cycle Directed Advertising）和普通定向广播（ Duty Cycle Directed Advertising）这里不想说，我们只看快速广播，在超时后下进慢速模式。如果是慢速广播，超时就进入无效模式：

```
            break;
        }
        // Upon time-out, the next advertising mode is started, i.e. go from fast to slow or from slow to
        case BLE_GAP_EVT_TIMEOUT:
            if (p_ble_evt->evt.gap_evt.params.timeout.src == BLE_GAP_TIMEOUT_SRC_ADVERTISING)
            {
                switch (m_adv_mode_current)
                {
                    case BLE_ADV_MODE_DIRECTED:
                        LOG("[ADV]: Timed out from directed advertising.\r\n");
                        {
                            uint32_t err_code;
                            err_code = ble_advertising_start(BLE_ADV_MODE_DIRECTED_SLOW);
                            if ((err_code != NRF_SUCCESS) && (m_error_handler != NULL))
                            {
                                m_error_handler(err_code);
                            }
                        }
                        break;
                    case BLE_ADV_MODE_DIRECTED_SLOW:
                        LOG("[ADV]: Timed out from directed slow advertising.\r\n");
                        {
                            uint32_t err_code;
                            err_code = ble_advertising_start(BLE_ADV_MODE_FAST);
                            if ((err_code != NRF_SUCCESS) && (m_error_handler != NULL))
                            {
                                m_error_handler(err_code);
                            }
                        }
                        break;
                    case BLE_ADV_MODE_FAST:
                    {
                        uint32_t err_code;
                        m_adv_evt = BLE_ADV_EVT_FAST;
                        LOG("[ADV]: Timed out from fast advertising, starting slow advertising.\r\n");
                        err_code = ble_advertising_start(BLE_ADV_MODE_SLOW);
                        if ((err_code != NRF_SUCCESS) && (m_error_handler != NULL))
                        {
                            m_error_handler(err_code);
                        }
                        break;
                    }
                    case BLE_ADV_MODE_SLOW:
                        m_adv_evt = BLE_ADV_EVT_IDLE;
                        LOG("[ADV]: Timed out from slow advertising, stopping advertising.\r\n");
                        if (m_evt_handler != NULL)
                        {
                            m_evt_handler(m_adv_evt);
                        }
                        break;
```

广播超时

普通定向广播

慢速定向广播

如果是快速广播，超时后进入慢速模式

如果是 慢速广播，超时后进入无效模式

那么如何启动广播了，看主函数里：

```
/**@brief Function for application main entry.
 */
int main(void)
{
    uint32_t err_code;
    bool erase_bonds;

    // Initialize.
    timers_init();
    buttons_leds_init(&erase_bonds);
    ble_stack_init();
    device_manager_init(erase_bonds);
    gap_params_init();
    advertising_init();
    services_init();
    conn_params_init();

    // Start execution.
    application_timers_start();
    err_code = ble_advertising_start(BLE_ADV_MODE_FAST);
    APP_ERROR_CHECK(err_code);

    // Enter main loop.
    for (;;)
    {
        power_manage();
    }
}

/**
 * @}
 */
```

详细看下 ble_advertising_start 函数内部设置，设置广播模式的时候是递进设置的，这四种模式（其实 5 种，定向广播分 2 种）是递进的，比如你设置了启动广播时选择 Direct 模式，但是如果你并未在初始化时设置 Direct 模式的相关参数，那么它就会回尝试 Fast 模式，如果初始化时 Fast 模式的相关信息也没设置，就会再尝试 Slow 模式，如果初始化时 Slow 模式相关信息也没设置最后就直接进入到 Idle 模式了。

同样的，广播超时后的超时处理就是选择下一个模式再进行广播，比如你 Fast 模式启动广播成功后，如果超时时间是 3 分钟，3 分钟后，广播超时处理中就是选择尝试 Slow 模式广播。

其实模式的定义只是给出了一个可以直接利用的模块，比如 Fast 模式和 Slow 模式并没有定义所谓的快慢是多少，只是给以一个你可以直接使用的代码模块。比如你的使用场景是希望设备上电后以 30ms 快速广播 20s，如果一直都没有被连接上，30s 后切换成 200ms 的广播 3 分钟以达到减小功耗目的。

下面代码就详细表述了这个功能：

```
uint32_t ble_advertising_start(ble_adv_mode_t advertising_mode)
{
    uint32_t             err_code;
    ble_gap_adv_params_t adv_params;

    m_adv_mode_current = advertising_mode;

    uint32_t             count = 0;

    // Verify if there are any pending flash operations. If so, delay starting advertising until
    // the flash operations are complete.
    err_code = pstorage_access_status_get(&count);
    if (err_code == NRF_ERROR_INVALID_STATE)
    {
    else if (err_code != NRF_SUCCESS)
    {
    if (count != 0)
    {
    // Fetch the peer address.
    ble_advertising_peer_address_clear();
    if (  ((m_adv_modes_config.ble_adv_directed_enabled)
          && m_adv_mode_current == BLE_ADV_MODE_DIRECTED)
        ||((m_adv_modes_config.ble_adv_directed_slow_enabled)
          && m_adv_mode_current == BLE_ADV_MODE_DIRECTED_SLOW))
    {
    // If a mode is disabled, continue to the next mode. I.e fast instead of direct, slow instead of fast, i
    if (  (m_adv_mode_current == BLE_ADV_MODE_DIRECTED)
        &&(!m_adv_modes_config.ble_adv_directed_enabled || !peer_address_exists(m_peer_address.addr)))
    {
        m_adv_mode_current = BLE_ADV_MODE_DIRECTED_SLOW;
    }
    if (  (m_adv_mode_current == BLE_ADV_MODE_DIRECTED_SLOW)
        &&(!m_adv_modes_config.ble_adv_directed_slow_enabled || !peer_address_exists(m_peer_address.addr)))
    {
        m_adv_mode_current = BLE_ADV_MODE_FAST;
    }
    if (!m_adv_modes_config.ble_adv_fast_enabled && m_adv_mode_current == BLE_ADV_MODE_FAST)
    {
        m_adv_mode_current = BLE_ADV_MODE_SLOW;
    }
    if (!m_adv_modes_config.ble_adv_slow_enabled && m_adv_mode_current == BLE_ADV_MODE_SLOW)
    {
        m_adv_mode_current = BLE_ADV_MODE_IDLE;
        m_adv_evt          = BLE_ADV_EVT_IDLE;
    }
```

最后一个问题，设置模式成功后执行什么操作？就是广播初始化最后一个函数：

```
err_code = ble_advertising_init(&advdata, NULL, &options, on_adv_evt, NULL);//广播参数
```

函数中 on_adv_evt 参数为广播处理函数，设置成功模式后执行对应操作，实际上只用到了两种，设置的快速广播和无效广播：

```
/**@brief Function for handling advertising events.
 *
 * @details This function will be called for advertising events which are pa
 *
 * @param[in] ble_adv_evt  Advertising event.
 */
static void on_adv_evt(ble_adv_evt_t ble_adv_evt)
{
    uint32_t err_code;

    switch (ble_adv_evt)
    {
        case BLE_ADV_EVT_FAST:
            err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
            APP_ERROR_CHECK(err_code);
            break;
        case BLE_ADV_EVT_IDLE:
            sleep_mode_enter();
            break;
        default:
            break;
    }
}
```

广播事件处理函数

快速广播模式

无效模式

无效广播模式会进入休眠模式，所以，大家会发现广播一段时间后，广播 LED 灯会停止闪烁，进入休眠模式函数，休眠模式函数如下：

```
*/
static void sleep_mode_enter(void)
{
    uint32_t err_code = bsp_indication_set(BSP_INDICATE_IDLE);
    APP_ERROR_CHECK(err_code);

    // Prepare wakeup buttons.
    err_code = bsp_btn_ble_sleep_mode_prepare();
    APP_ERROR_CHECK(err_code);

    // Go to system-off mode (this function will not return; wakeup will cause a reset).
    err_code = sd_power_system_off();
    APP_ERROR_CHECK(err_code);
}
```

设置了唤醒按键后，系统进入休眠。总结下：广播初始化中设置了 FAST 模式广播的相关参数，然后按 FAST 模式启动广播。当广播超时后，超时时间处理中判断是 FAST 模式超时，于是再启动 SLOW 模式广播，但是因为 SLOW 模式广播的相关参数并没有设置，于是切换成 IDLE 模式，并且调用了初始化时设置的回调函数。回函数中会设置唤醒按键然后设置深度睡眠。

那么广播初始化基本就讲到这里了。原理清楚了，那么大家就可以自己进行相关修改。下面了验证下，我们不进入休眠，一直进行快速广播，如下修改了？

1.2 不进入 IDLE 无效模式

通过上面的分析，广播模式的切换主要就是广播超时处理，如果无这个超时时间，那么就不会切换到下一种模式，所以，如何你需要保持一种广播模式不变，你可以注销这个超时时间，设置如下：

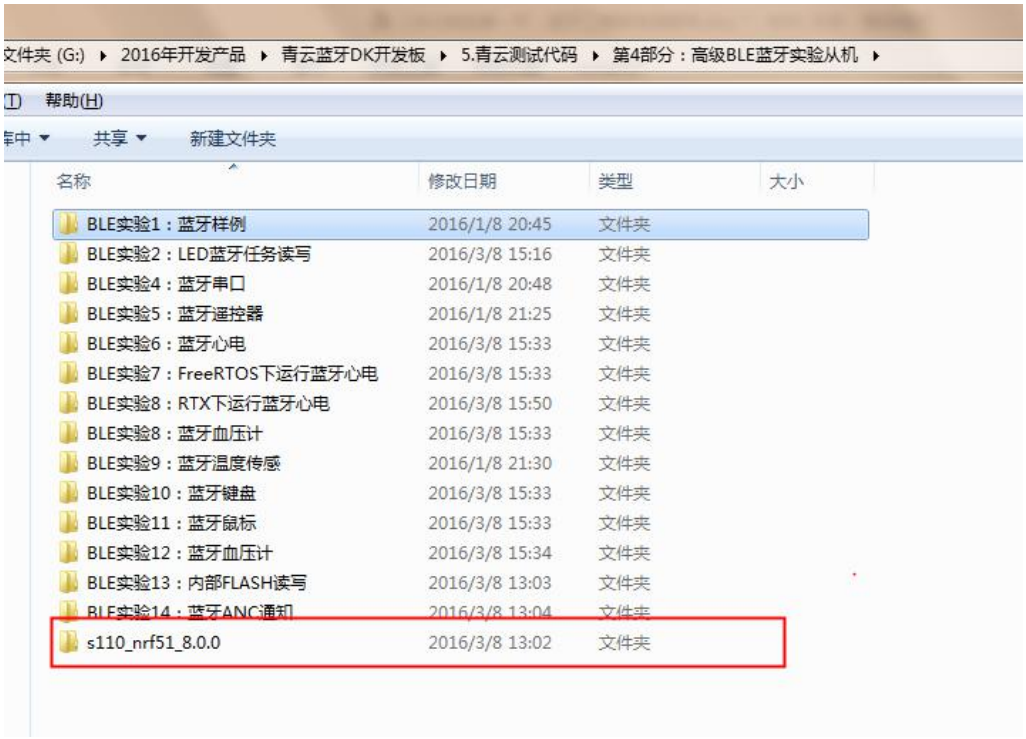
```
static void advertising_init(void)
{
    uint32_t          err_code;
    ble_advdata_t advdata;

    // Build advertising data struct to pass into @ref ble_advertising_init.
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type                 = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance        = true;
    advdata.flags                                                                    =
BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
    advdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) / sizeof(m_adv_uuids[0]);
    advdata.uuids_complete.p_uuids  = m_adv_uuids;

    ble_adv_modes_config_t options = {0};
    options.ble_adv_fast_enabled  = BLE_ADV_FAST_ENABLED;
    options.ble_adv_fast_interval = APP_ADV_INTERVAL;
    options.ble_adv_fast_timeout  = APP_ADV_TIMEOUT_IN_SECONDS;

    err_code = ble_advertising_init(&advdata, NULL, &options, on_adv_evt, NULL);
    APP_ERROR_CHECK(err_code);
}
```

把上面的 APP_ADV_TIMEOUT_IN_SECONDS 设置为 0 就可以了，然后修改后编译通过，提示 OK。

2 应用与调试

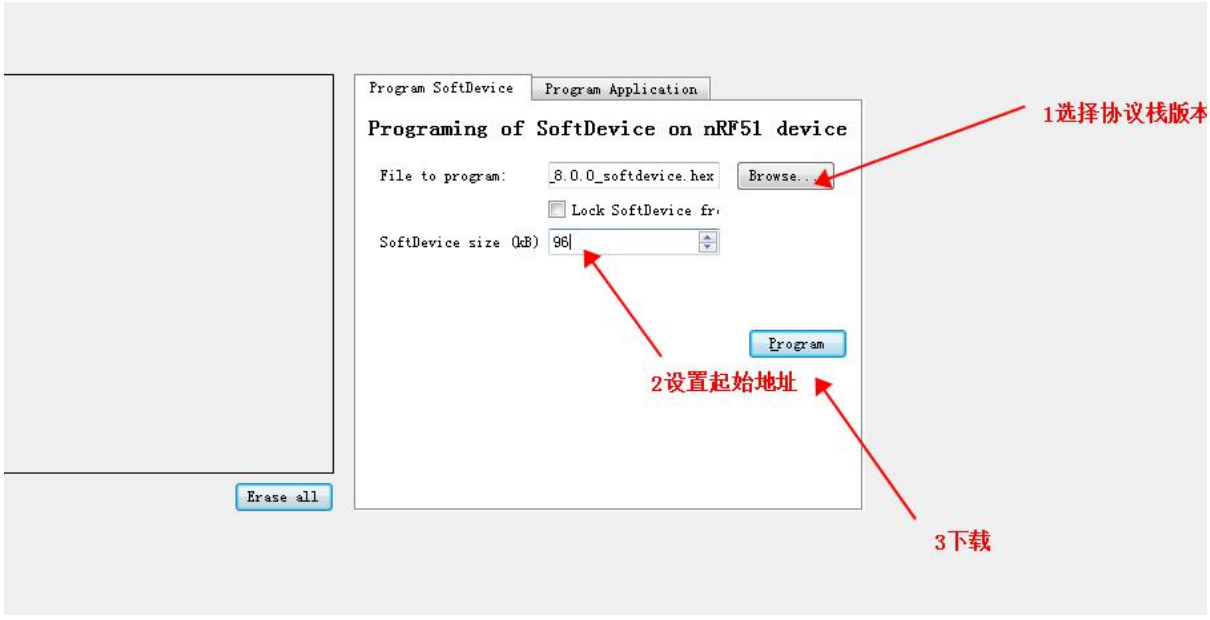
2.1 下载

连接好后首先下载协议栈，本例使用的协议栈为 S110.8.0 版本，位于文件夹第 4 部分这个位置：

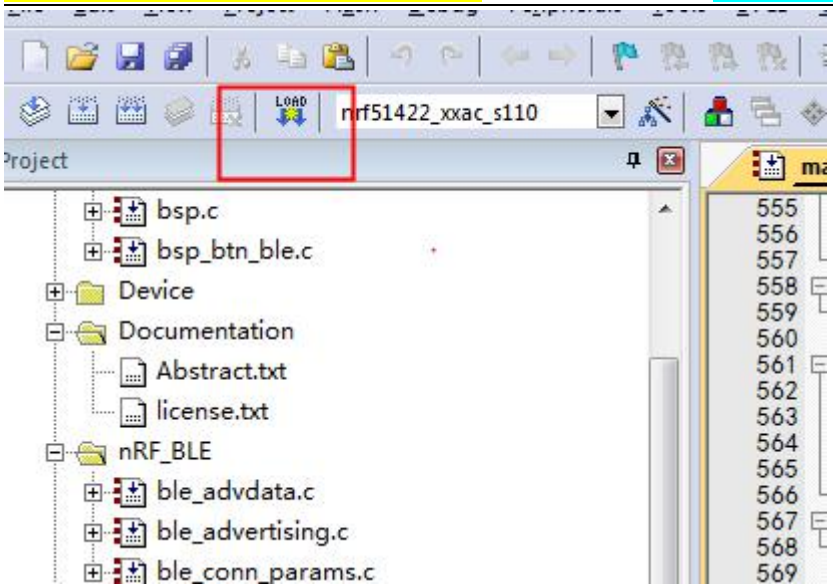


打开 NRFgo 

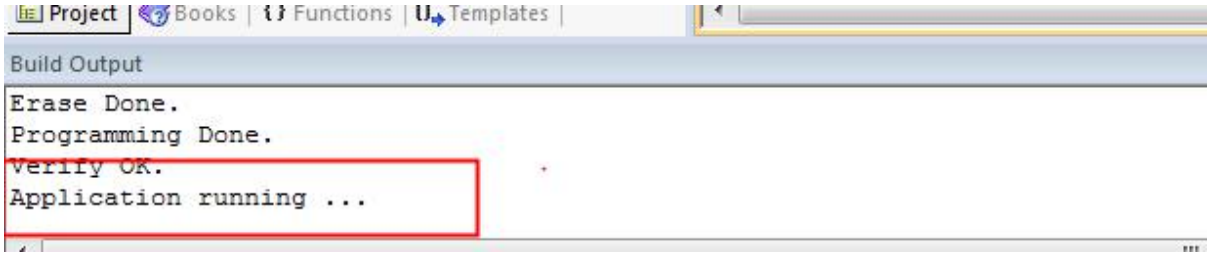
进行下载，可以参考软件篇介绍。下载如图所示，起始地址位 96K，首先整片擦除，后下载协议栈:



下载完后可以下载工程，首先把工程编译一下，通过后点击 KEIL 上的下载按键：



下载成功后提示如图，程序开始运行，同时开发板上广播 LED 开始广播：



2.2 测试

本实验需要使用抓包器，观察广播信号，广播信号一直广播，不切换模式。