青风带你玩蓝牙 nRF52832 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-nRF52832 开发板

2.7 蓝牙 BLE 之样例的建立

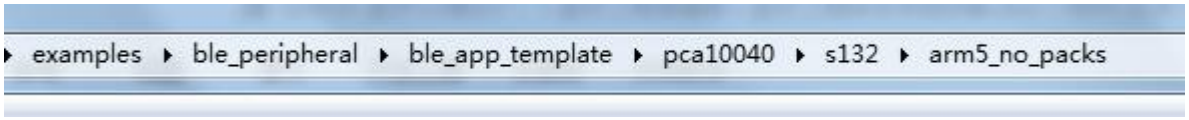
蓝牙 BLE 样例是为了让我们学习如何在 nRF52832 芯片上开发一个 BLE 应用而提供的模板工程，它是一个官方提供 BLE 的应用实例，具有通用性。为后面我们开发自己的应用实例提供参考，我们大可通过修改这个样本工程来实现我们自己应有。

蓝牙工程需要关注什么？如果搞清楚其主体框架？这几个问题我们下面就来详细讨论下：

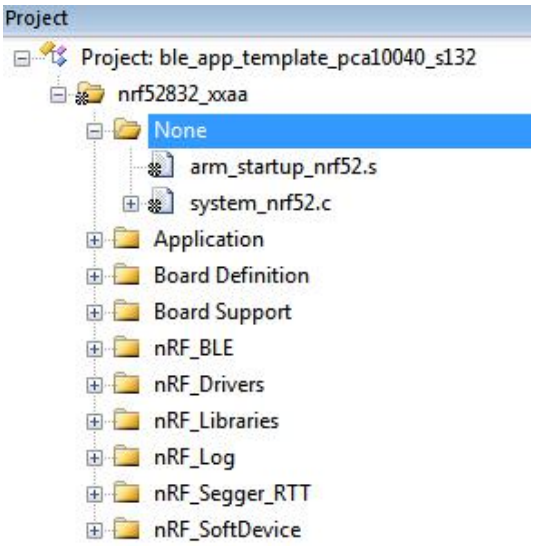
1 蓝牙样例工程结构：

蓝牙工程里包含了很多文件和工程目录，这些工程是做什么用的？那些是必须的？如何认识？那么我们首先来看看 BLE 蓝牙程序工程结构，结构如下图所示，工程建立是分层设置的，也就是应用层，协议层和硬件设备分开编译，这种方式对后面我们编写私有任务非常有力。在讲解源代码的时候我们结合蓝牙协议来说明。

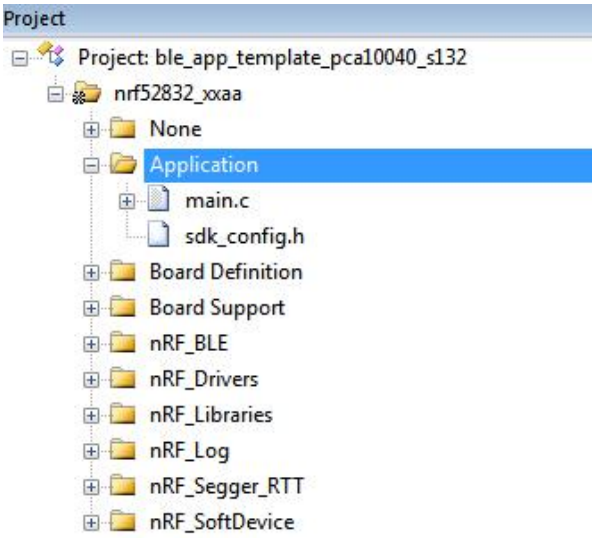
首先，打开 SDK 中如下工程目录下的工程文件，工程文件以 keil 建立：



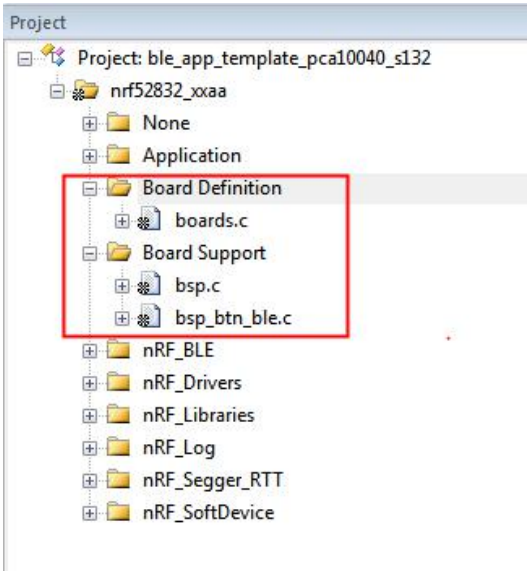
第一部分： None 工程树，该工程树中包含了两个文件，一个文件为 arm_startup_nrf52.s 是 nrf52 芯片的启动文件，该文件配置了芯片初始化启动的时候的堆栈空间，中断声明，启动循序的等参数，这些参数是芯片开机首先要运行配置的，改文件是最底层的配置文件，以汇编来书写。另外一文件为 system_nrf52.c 文件，这个文件是芯片系统文件，在运行 main 函数之前运行，配置了处理器的初始化时钟，寄存器等参数。这两个文件是工程必须的。如下图所示：



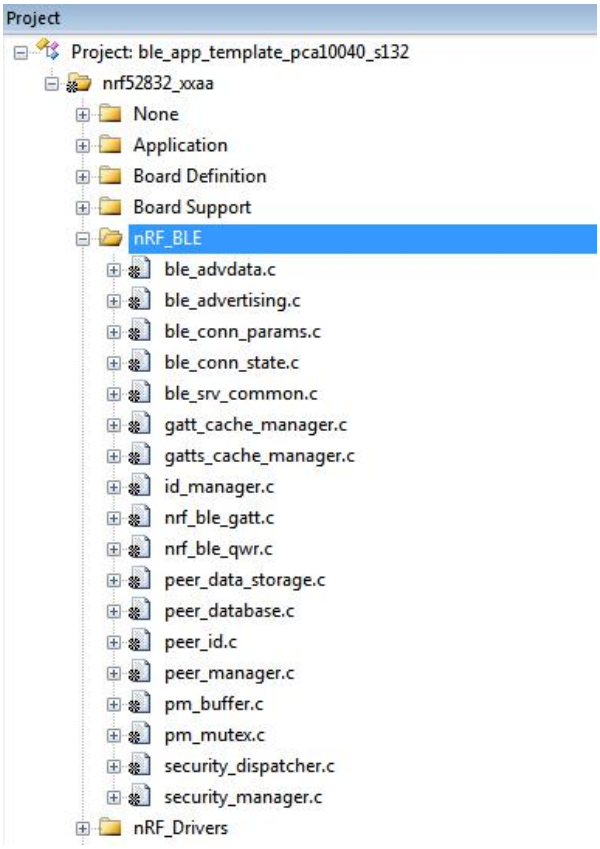
第二部分：Application 应用工程树，该工程树中主要就两个文件，一个是主函数 main.c 文件，该文件后面会拿出来具体讨论。第二个文件 sdk_config.h 配置文件，该文件也是非常重要的一个文件，各种外设或者驱动，都需要再这个文件里进行配置设置，这个文件会结合到后面的编程里具体提及：



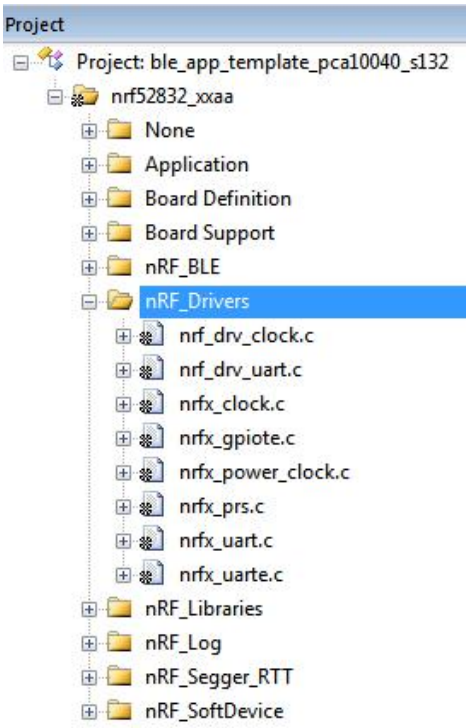
第三部分：Board Definition 板载定义和 Board Support 支持目录树，主要是按键和 LED 灯的功能的一些设置。比如协议栈下按键唤醒，按键休眠，长按与短按等配置功能。当然这两个文件在蓝牙工程里是可选的，你可以自己编写或者根据自己的硬件来进行修改。



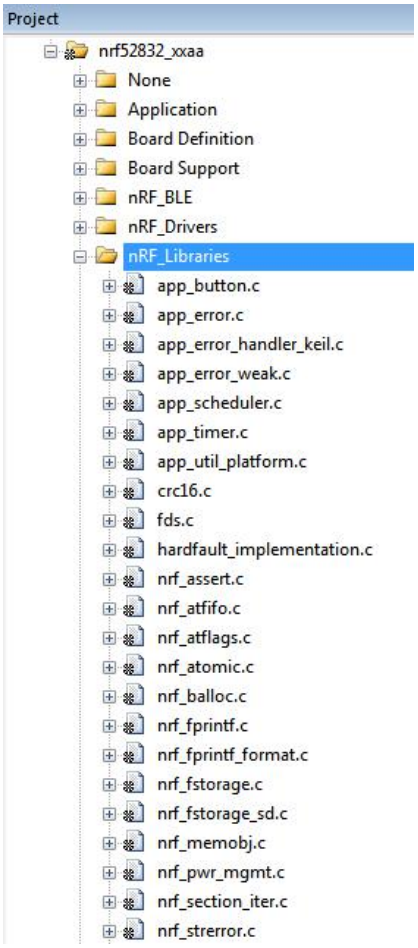
第四部分 nRF_BLE 文件夹，本文件夹提供一些蓝牙服务代码，比如广播的配置代码，连接参数的配置代码，GATT 的配置代码，还有 peer 设备匹配管理的代码，安全参数设置的代码等。



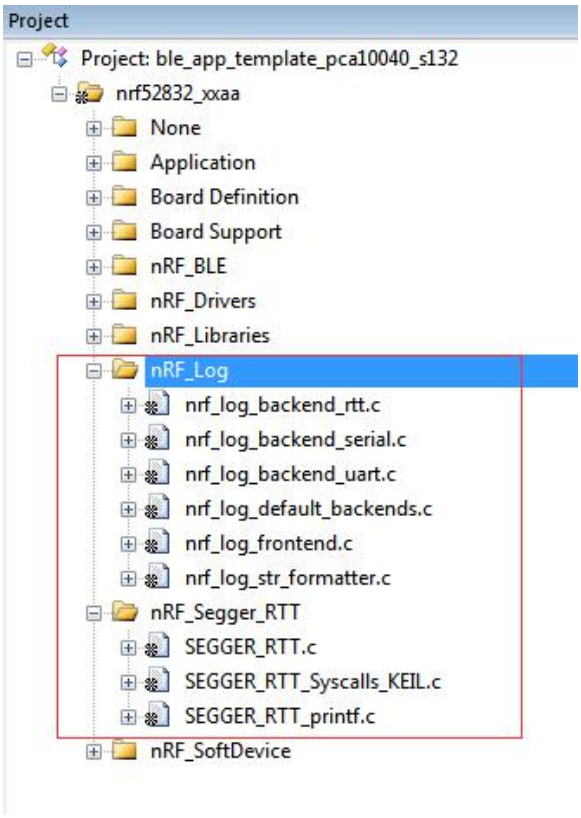
第五部分 nRF_Drivers 文件夹,提供外设驱动函，这个文件夹提供的新版本的外设驱动文件库，区别与老版本外设驱动，用 nrfx 表示新版驱动，不过 sdk15 外设任然可以兼容老板外设驱动文件库。



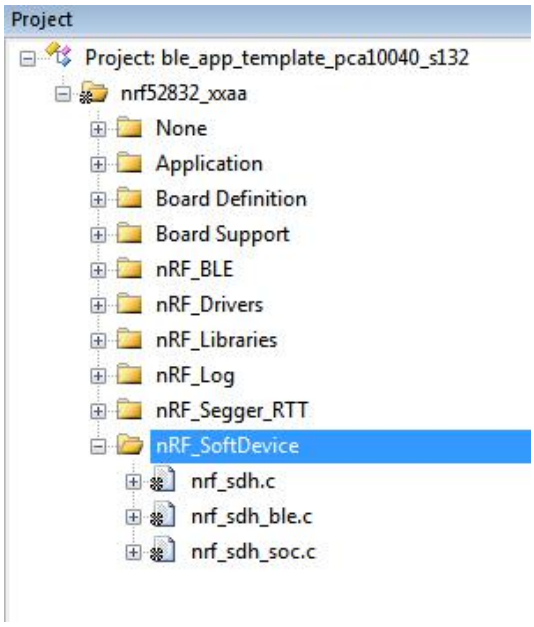
第六部分 nRF_Libraries 文件夹,提供外设驱动函数代码。nrf51822 提供的一些现成的库函数，这些函数是和硬件紧密相连的，这样带有 nrf 前缀的就是和 NRF 芯片处理相关的库函数，包含了一些内存处理，打印，缓冲，能量管理等文件。带有 app 前缀的文件是和应用有关系的库函数，是以外设驱动为基础的二级驱动文件。



第七部分 nRF_log 工程目录树和 nRF_Segger_RTT 工程目录树，这两个部分是提供一个打印输出接口，提供人家交互的一种方式。Log 打印输出可以使用两个通道，一个就是 uart 串口，在 log 驱动文件夹里有配置。另外一种就是使用 jlink 仿真器的 RTT 打印的输出方式，在串口端口被占用的时候使用。



第八部分 nRF_SoftDevice 工程目录树，这里包含的文件主要是配置协议栈初始化的时候协议栈的参数设定，由于协议栈实际上是不开源的，而是留下了配置接口，这些配置接口通过客户配置相关协议栈的参数来设置协议栈运行状态。



以上的八个部分是一个蓝牙样板工程里应该包含的内容，其文件数量众多，为了方便初学者或者使得开发者尽快的实现项目开发，不建议大家重新搭建工程，而是尽可能的在样板工程中进行修改与编程，节省大量开发时间。

2 蓝牙主函数分析：

工程目录前面已经介绍完，工程目录中大部分文件都是官方 SDK 库中提供好了的，如果独立添加文件需要注意文件目录的设置。当然推荐大家直接用 SDK 模板来编写自己的方案，加快程序开发。

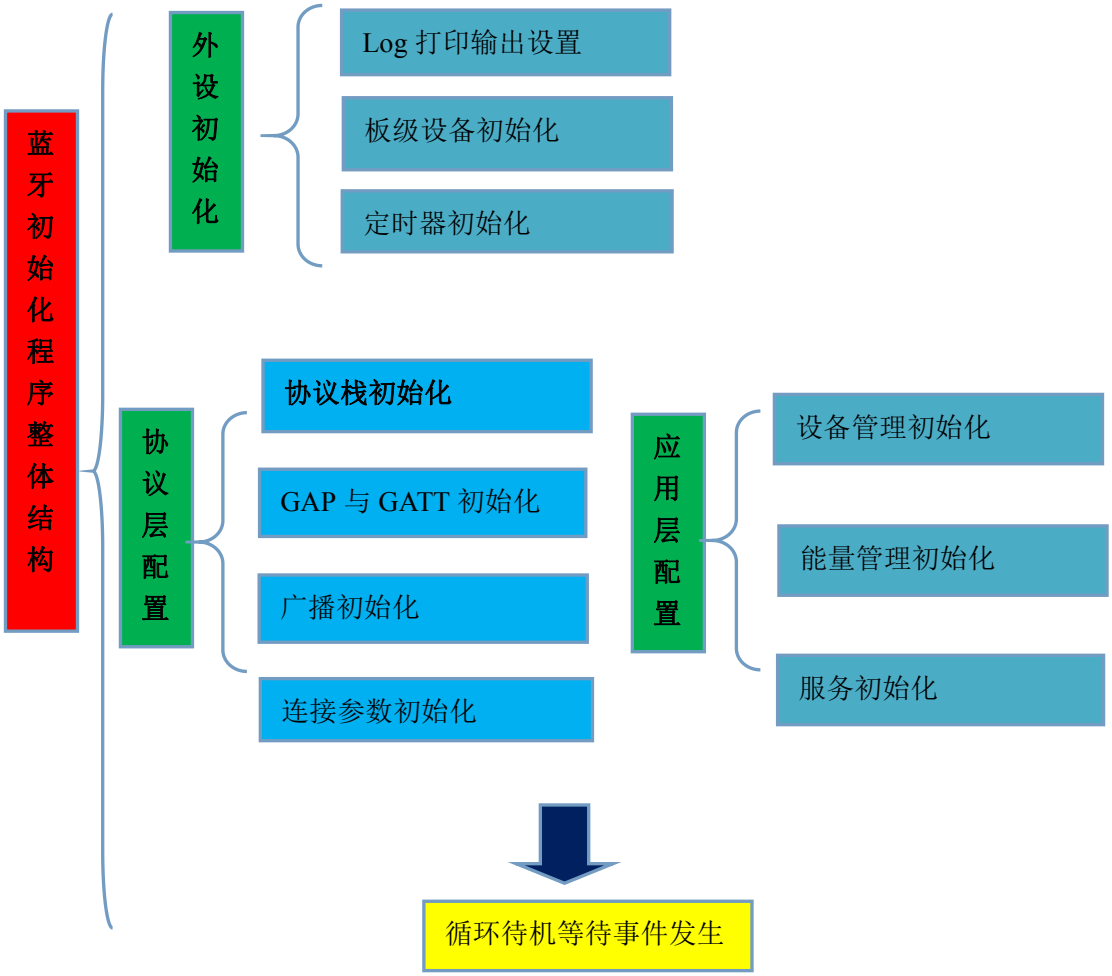
但是真正要运行，需要大家弄清楚整个蓝牙工程的基本框架，知道哪里是干嘛的？哪里需要大家编写的？ 就比如现在建楼房，需要先打一个框架，然后往框架上填东西。带着上面几个问题进行下面这节的讨论。

学过单片机的同学都知道，主函数 main 是一个程序的基本框架。而蓝牙样板工程中主函数 main 就是我们需要编写的地方，那怎么编写了？为了弄清楚这个问题，我们就必须来分析下这个模板工程的代码功能及意义，首先来看下主函数：

```
int main(void)
{
    uint32_t err_code;
    bool erase_bonds;

    // Initialize.
    log_init();//log 打印初始化
    timers_init();//初始化定时器
    buttons_leds_init(&erase_bonds);//初始化按键和 LED
    power_management_init();//能源管理函数
    ble_stack_init();//协议栈的初始化
    gap_params_init();//GAP 参数初始化
    gatt_init();//GATT 初始化
    advertising_init();//广播初始化
    services_init();//服务初始化
    conn_params_init();//更新过程初始化
     peer_manager_init();//设备管理初始化

    // Start execution.
    NRF_LOG_INFO("Template example started.");
    application_timers_start();//应用定时器开始
```

```
    advertising_start(erase_bonds);//广播开始



    // Enter main loop.
    for (;;)
    {
      idle_state_handle();
    }
}
```

主函数很长，功能很多，我们画个图来直观的表示下：



主函数结构方面主要就是三个部分，一个是外设的配置，初始化外设定时器，按键，串口等。第二个就是协议层的配置，协议栈初始化、GAP 和 GATT 初始化、广播配置、连接参数配置。第三个就是应用层配置，主要就是服务初始化，蓝牙服务的设置。下面来看看各个部分的代码：

2.1 外设初始化

外设初始化实现需要使用的一些外部设备，外部设备的使用大家可以先学习外设教程的部分。下面主要看看协议栈下外设部分如何初始化。

2.1.1 log_init()函数

```
static void log_init(void)
{
    ret_code_t err_code = NRF_LOG_INIT(NULL);
    APP_ERROR_CHECK(err_code);
    NRF_LOG_DEFAULT_BACKENDS_INIT();//初始化 log 设置，配置 log 通道
}
```

LOG 打印功能是为了方便调试，在设备和人之间建立一个信息交换方式。选择两个通道就行信息输出，一个是串口通道，一个是仿真器 jlink 的 RTT 通道。由于 nrf5x 芯片串口端口只有一个，当串口通道被占用，RTT 的输出成为唯一的方法。具体请参看《蓝牙 RTT 输出 log 信息》详解教程。

2.1.2 static void timers_init(void)函数

```
static void timers_init(void)
{
    // 初始化定时器模块，使其用于调度
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_MAX_TIMERS,
APP_TIMER_OP_QUEUE_SIZE, true);
    /*
    err_code = app_timer_create(&m_app_timer_id,
APP_TIMER_MODE_REPEATED, timer_timeout_handler);
    APP_ERROR_CHECK(err_code); */
}
```

创建要由应用程序使用的任何定时器。并且官方给了如何创建一个计时器的实例。后面在心电，或者接近等应用实例的时候就会被使用，设置计数器 id 。对于每一个新的计时器需要，增加宏 APP_TIMER_MAX_TIMERS。如果不使用定时器，这个可以不改变，详细看《协议栈下软件定时器的建立详解》。

2.1.3 (void)buttons_leds_init(&erase_bonds);函数

```
static void buttons_leds_init(bool * p_erase_bonds)
{
    bsp_event_t startup_event;

    uint32_t err_code = bsp_init(BSP_INIT_LED | BSP_INIT_BUTTONS,
                                 APP_TIMER_TICKS(100, APP_TIMER_PRESCALER),
                                 bsp_event_handler);
    APP_ERROR_CHECK(err_code);

    err_code = bsp_btn_ble_init(NULL, &startup_event);
    APP_ERROR_CHECK(err_code);

    *p_erase_bonds = (startup_event == BSP_EVENT_CLEAR_BONDING_DATA);
}
```

初始化 LED 和按键的 操作，这个注意，直接调用的 app 文件，也就是定义了的应用库，官方提供，大家可以具体深入源代码，如果要使用中断需要修改，后面如果遇到这方面的应用再详细说明，如果不使用端口中断，这个可以不做修改。

这里定义了按键和 LED，下面给了一个需要回调的 bsp_event_handler，就是说这个函数初始化按键后，系统会自动帮你检测按键或者 LED 发生变化，如果有变化了，系统就回来调用。

同时配置了一个按键控制函数 bsp_btn_ble_init，可以用按键休眠和启动蓝牙设备广播。

```
void bsp_event_handler(bsp_event_t event)
{
    uint32_t err_code;
    switch (event)
    {
        case BSP_EVENT_SLEEP://产生休眠事件
            sleep_mode_enter();
            break;
```

```
        case BSP_EVENT_DISCONNECT://产生断开事件
            err_code = sd_ble_gap_disconnect(m_conn_handle,
BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            if (err_code != NRF_ERROR_INVALID_STATE)
            {
                APP_ERROR_CHECK(err_code);
            }
            break;
        case BSP_EVENT_WHITELIST_OFF://参数白名单丢失事件
            err_code =
                  ble_advertising_restart_without_whitelist(&m_advertising);
            if (err_code != NRF_ERROR_INVALID_STATE)
            {
                APP_ERROR_CHECK(err_code);
            }
            break;

        default:
            break;
    }
}
```

回调的 bsp_event_handler,也就是 bsp 中断服务函数如下，实现多个事件，比如 BSP_EVENT_SLEEP 进入睡眠， BSP_EVENT_DISCONNECT 蓝牙断开事件，BSP_EVENT_WHITELIST_OFF 重新广播没有白名单事件等，对应事件执行对应操作。详细看《协议栈下按键的使用》教程。

2.2 能源管理函数初始化

能源管理初始化函数，主要是实现 cortex-m4 内核 SCB 内核里低功耗管理设置的初始化，一般情况下可以不变动。

```
static void power_management_init(void)
{
```

```
    ret_code_t err_code;
    err_code = nrf_pwr_mgmt_init();
    APP_ERROR_CHECK(err_code);
}
```

```
ret_code_t nrf_pwr_mgmt_init(void)
{
    NRF_LOG_INFO("Init");

    m_shutdown_started = false;
    nrf_mtx_init(&m_sysoff_mtx);//初始化互斥量
    nrf_section_iter_init(&m_handlers_iter, &pwr_mgmt_data);
    //初始化迭代器的函数
    PWR_MGMT_SLEEP_INIT();//休眠初始化
    PWR_MGMT_DEBUG_PINS_INIT();//调试管脚初始化
    PWR_MGMT_STANDBY_TIMEOUT_INIT();//待机超时初始化
    PWR_MGMT_CPU_USAGE_MONITOR_INIT();//cpu 使用率跟踪初始化

    return PWR_MGMT_TIMER_CREATE();
}
```

2.3 协议栈初始化

ble_stack_init()协议初始化：协议栈初始化工作主要做下面几点：
1：协议栈回复使能应答，主要工作就是协议栈时钟初始化配置。
2：初始化协议栈，设置协议栈相关处理函数,使能协议栈。
3：注册蓝牙处理调度事件。
具体代码如下，详细讲解请参看**《蓝牙协议栈初始化详解》**篇教程：

```
    ret_code_t err_code;
    //协议栈回复使能应答，主要是配置协议栈时钟
```

```
    err_code = nrf_sdh_enable_request();
    APP_ERROR_CHECK(err_code);

    // 配置协议栈使用默认地址
    // 获取 RAM 的开始地址
     uint32_t ram_start = 0;
    //默认配置包括协议栈起始地址，连接配置等
    err_code = nrf_sdh_ble_default_cfg_set(APP_BLE_CONN_CFG_TAG, &ram_start);
    APP_ERROR_CHECK(err_code);

    // 使能协议栈
    err_code = nrf_sdh_ble_enable(&ram_start);
    APP_ERROR_CHECK(err_code);

    // 注册蓝牙处理事件
    NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO,
ble_evt_handler, NULL);
```

2.4 gap 初始化和 GATT 初始化

2.4.1 gap_params_init()函数 gap 初始化；

通用访问配置文件(Generic Access Profile, GAP)，该 Profile 保证不同的 Bluetooth 产品可以互相发现对方并建立连接。

（GAP）定义了蓝牙设备如何发现和建立与其他设备的安全（或不安全）连接。它处理一些一般模式的业务（如询问、命名和搜索）和一些安全性问题（如担保），同时还处理一些有关连接的业务（如链路建立、信道和连接建立）。GAP 规定的是一些一般性的运行任务。因此，它具有强制性，并作为所有其它蓝牙应用规范的基础。

GAP 是所有其他配置文件的基础,它定义了在蓝牙设备间建立基带链路的通用方法.除此之外,GAP 还定义了下列内容:

①:必须在所有蓝牙设备中实施的功能

②:发现和链接设备的通用步骤

③:基本用户界面术语.

在 GAP 初始化里实际上只做了两个工作，一个是配置设备名称，或者生成设备图标。第二个功能就是配置 GAP 的连接参数，代码如下所，具体探讨见**《GAP 初始化设置详解》**。

```
static void gap_params_init(void)
{
    ret_code_t                    err_code;
    ble_gap_conn_params_t     gap_conn_params;
    ble_gap_conn_sec_mode_t sec_mode;
    //设置设备名称
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
    err_code = sd_ble_gap_device_name_set(&sec_mode,
                                          (const uint8_t *)DEVICE_NAME,
                                              strlen(DEVICE_NAME));
    APP_ERROR_CHECK(err_code);

    /* YOUR_JOB: Use an appearance value matching the application's use case.
       err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_);
       APP_ERROR_CHECK(err_code); */

    memset(&gap_conn_params, 0, sizeof(gap_conn_params));
   //初始化 gap 连接间隔，从机延迟，超时时间
    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency       = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout   = CONN_SUP_TIMEOUT;
    //把配置的参数设置成功
    err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
    APP_ERROR_CHECK(err_code);
}
```

2.4.2 gatt_init()的 GATT 初始化；

GATT 称为通用属性规范 Generic Attribute profile，GATT 层是传输真正数据所在的层。包括了一个数据传输和存储框架以及其基本操作。其大部分设置是在服务中进行的，在主函数中只需要初始化数据长度这个参数，代码如下所示，gatt_init 函数中调用了 nrf_ble_gatt_init 函数，改函数中定义了 gatt 中的主机，从机的最大 MTU 的长度，以及协商数据的长度。

```
static void gatt_init(void)
{
    ret_code_t err_code = nrf_ble_gatt_init(&m_gatt, NULL);
    APP_ERROR_CHECK(err_code);
}


ret_code_t
nrf_ble_gatt_init(nrf_ble_gatt_t * p_gatt, nrf_ble_gatt_evt_handler_t evt_handler)
{
    VERIFY_PARAM_NOT_NULL(p_gatt);

    p_gatt->evt_handler                = evt_handler;
    p_gatt->att_mtu_desired_periph= NRF_SDH_BLE_GATT_MAX_MTU_SIZE;
    p_gatt->att_mtu_desired_central = NRF_SDH_BLE_GATT_MAX_MTU_SIZE;
    p_gatt->data_length                = NRF_SDH_BLE_GAP_DATA_LENGTH;

    for (uint32_t i = 0; i < NRF_BLE_GATT_LINK_COUNT; i++)
    {
        link_init(&p_gatt->links[i]);
    }
    return NRF_SUCCESS;
}
```

关于 **GAP** 和 **GATT** 的详细分析请参考我们的另一篇教程《**GAP** 初始化设置详解》，同时 **GATT** 的内容会在后面的服务建立教程中详细探讨。

2.5 广播初始化

static void advertising_init(void)：函数初始化广播功能；

```
static void advertising_init(void)
{

    uint32_t          err_code;
    ble_advertising_init_t    init;
    //
    memset(&advdata, 0, sizeof(advdata));

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

    init.config.ble_adv_fast_enabled    = true;//广播类型
    init.config.ble_adv_fast_interval = APP_ADV_INTERVAL;//广播间隔
    init.config.ble_adv_fast_timeout    = APP_ADV_DURATION;//广播超时

    init.evt_handler = on_adv_evt;

    err_code = ble_advertising_init(&m_advertising, &init);//初始化广播，导入参数
    APP_ERROR_CHECK(err_code);

    ble_advertising_conn_cfg_tag_set(&m_advertising, APP_BLE_CONN_CFG_TAG);
    //设置广播识别号
}
```

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&config 选择项广播数据结构体如下，列出了一些需要初始化的广播数据：

```
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 specified). */
bool                             include_appearance;
  /**< Determines if Appearance shall be included. */
uint8_array_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 solcited 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. */
} ble_advdata_t;
```

一个广播数据实际上最多可以携带 31 字节的数据，它通常包含用户可读的名字、关于设备发送数据包的有关信息、用于表示此设备是否可被发现的标志等类似的标志，如上面结构体的定义。

当主机接收到广播包后，它可能发送请求更多数据包的请求，称为扫描回应，如果它被设置成主动扫描，从机设备将会发送一个扫描回应做为对主机请求的回应，扫描回应最多也可以携带 31 字节的数据。广播扫描回应包的数据结构类型可以和广播包一致，也如上面结构体定义。那么如何设置广播包了，具体请参考《广播初始化详解》这篇教程。

2.6 服务初始化

static void services_init(void)：服务初始化；

```
static void services_init(void)
{
    ret_code_t         err_code;
    nrf_ble_qwr_init_t qwr_init = {0};

    // Initialize Queued Write Module.
    qwr_init.error_handler = nrf_qwr_error_handler;

    err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
    APP_ERROR_CHECK(err_code);

    /* YOUR_JOB: Add code to initialize the services used by the application.
       ble_xxs_init_t                     xxs_init;
       ble_yys_init_t                     yys_init;

       // 初始化服务
       memset(&xxs_init, 0, sizeof(xxs_init));

       xxs_init.evt_handler                = NULL;
       xxs_init.is_xxx_notify_supported    = true;
       xxs_init.ble_xx_initial_value.level = 100;

       err_code = ble_bas_init(&m_xxs, &xxs_init);
       APP_ERROR_CHECK(err_code);

       // 初始化另外一个服务
       memset(&yys_init, 0, sizeof(yys_init));
       yys_init.evt_handler                  = on_yys_evt;
       yys_init.ble_yy_initial_value.counter = 0;
```

```
        err_code = ble_yy_service_init(&yys_init, &yy_init);
        APP_ERROR_CHECK(err_code);
    */
}
```

服务初始化就是建立一个服务声明，给一个 RAM 空间，专门对服务进行初始化和声明。这个代码在样例里是空的，也就是说蓝牙样例里只搭了一个框架，而没有建立蓝牙服务，那么在后面的应用实例教程里会来讲解如何添加服务。具体实例请看后面的蓝牙服务篇教程。

2.7 连接参数和安全参数初始化

conn_params_init();连接参数初始化

```
/**@brief Function for initializing the Connection Parameters module.
 */
static void conn_params_init(void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;
    memset(&cp_init, 0, sizeof(cp_init));
    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay =
                                  FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  =
                                  NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail             = false;
    cp_init.evt_handler                    = on_conn_params_evt;
    cp_init.error_handler                  = conn_params_error_handler;
    err_code = ble_conn_params_init(&cp_init);
    APP_ERROR_CHECK(err_code);
}
```

SDK 提供了一个名为 ble_conn_params 的模块用于管理连接参数更新，它通过 SoftDevice API 进行处理，包括请求的时间和第一次请求被拒绝再发送一个新的请求。SoftDevice API 函数是经过封装后的函数，无法查看源函数，大家只要通过帮助文档查找函数意义，所有带 sd 前缀的函数名就是 SoftDevice API 函数。

在初始化结构体 ble_conn_params_init_t 中，定义了更新过程的有关参数，例如，是否开始连接，什么情况开始写入一个特定的 CCCD，是否使用连接参数，发送更新请求的延时等等。大家可以查看源码，在 BLE_CONN_PARAMS.H 文件内

在初始化函数 ble_conn_params_init()中，使用封装了初始化连接参数(ble_gap_conn_params_t）的结构体 ble_conn_params_init_t 作为输入参数进行连接参数初始化结构体。(注意红色标注的概率名称)

```
err_code = ble_conn_params_init(&cp_init);
```

ble_conn_params 在 SDK 模块确保与主机（集中器）的连接参数相适应，如果不适应，外围设备将要求更改连接参数，超过设定的更新次数都没有更新成功后，它就会断开连接或者根据设置返回一个事件到应用层。**详细分析请参考我们的另一篇教程《连接参数更新详解》。**

2.8 设备管理初始化

2.9 广播开始

advertising_start(erase_bonds);

```
static void advertising_start(bool erase_bonds)
{
    if (erase_bonds == true)
    {
        delete_bonds();
        // Advertising is started by PM_EVT_PEERS_DELETED_SUCEEDED event
    }
    else
    {
        ret_code_t err_code = ble_advertising_start(&m_advertising,
```

```
BLE_ADV_MODE_FAST);


        APP_ERROR_CHECK(err_code);
    }
}
```

广播数据的设置在 main.c 的 **advertising_start** 中，安装之安全广播初始化中设置的广播数据结构开始广播，主函数中，首先启动的为快速广播，关于广播的类型具体参见《广播初始化详细》中的讲解。

2.10 电源待机

idle_state_handle（）；电源待机

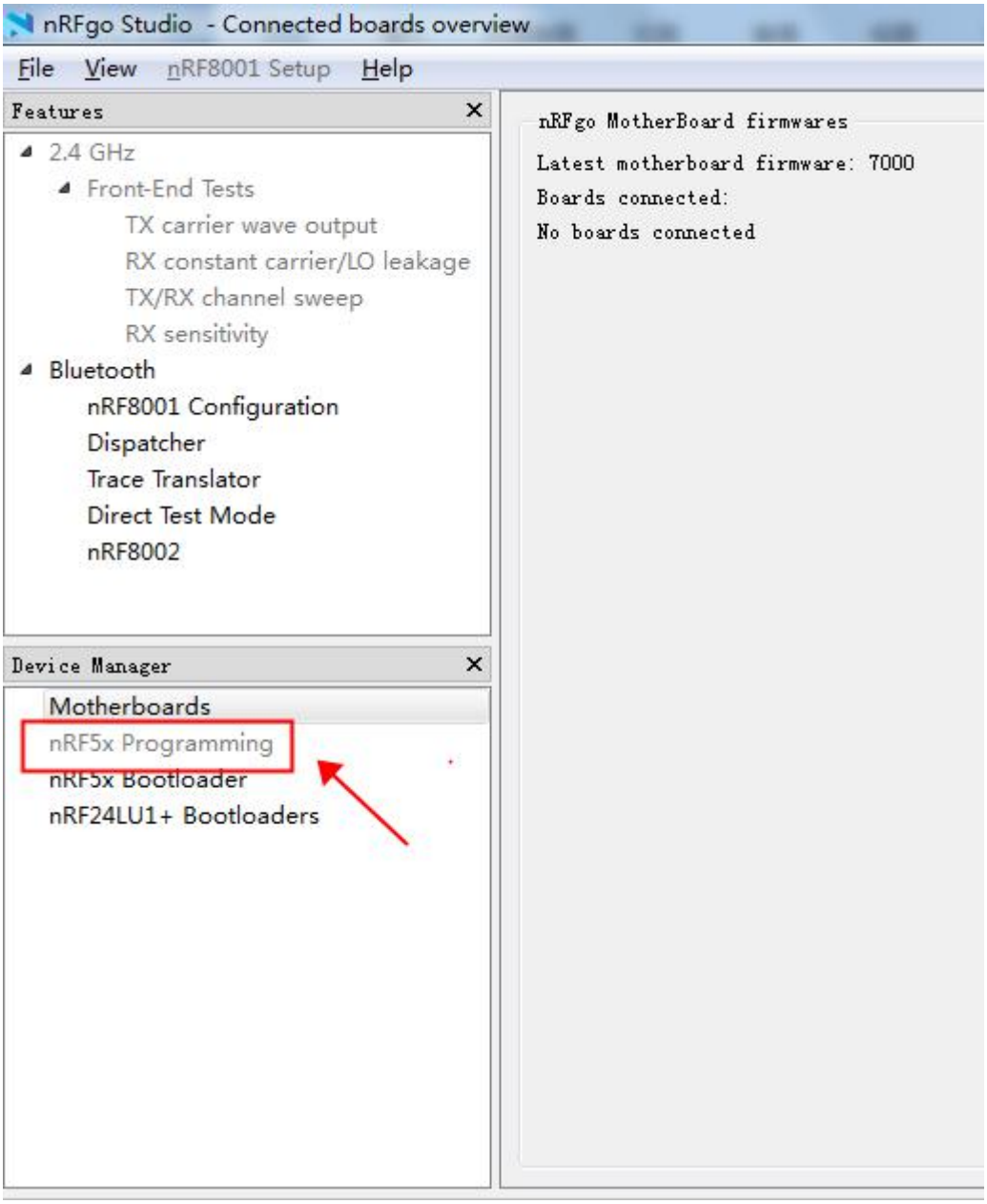
主函数中，最后一个循环等待，调用了 idle_state_handle（）函数，这个函数字面意思为无效状态操作，也就是说在没有蓝牙事件或者芯片处理事件的时候，设备调用这个函数，可以使设备处于 system on 状态，也就是待机状态。idle_state_handle（）函数中调用了 nrf_pwr_mgmt_run 函数中，这个函数中使用了 SDK 提供了一个协议栈函数名为 sd_app_evt_wait 的模块用于管理电源，可以直接调用，当中断事件发生后，设备被唤醒工作。

```
static void idle_state_handle(void)
{
    if (NRF_LOG_PROCESS() == false)
    {
        nrf_pwr_mgmt_run();
    }
}
```

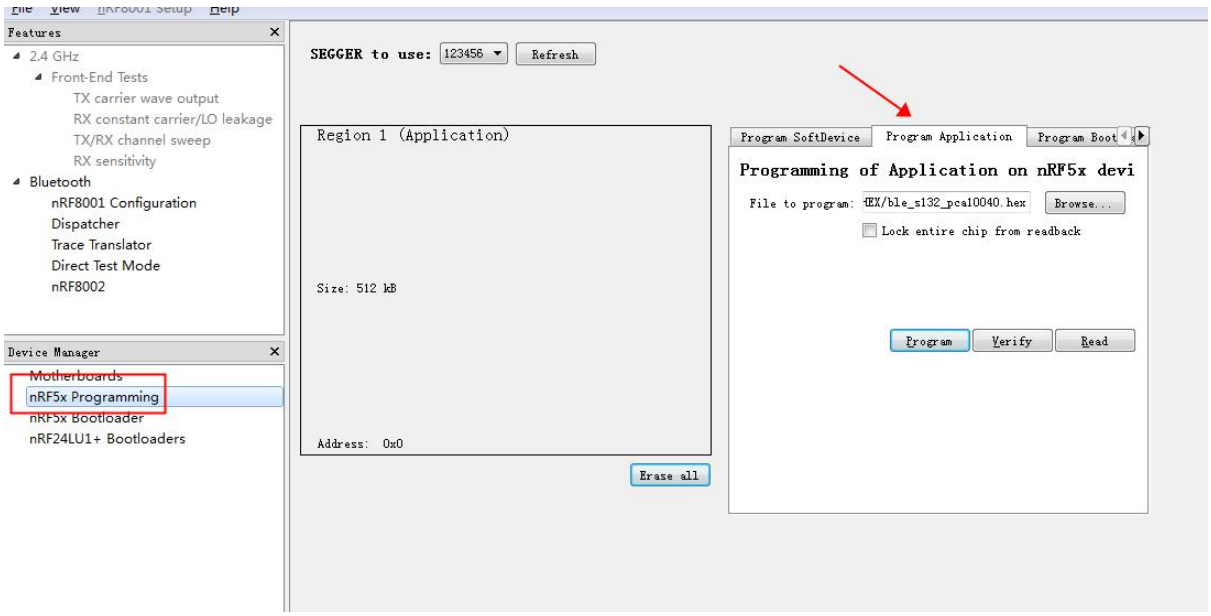
3 下载验证：

之前下载的都是外设带， 外设代码不带协议栈，从本章后讲解的都将上蓝牙程序，蓝牙程序带协议栈，下载过程和外设有区别，下面详解进行介绍，后面的蓝牙程序下载方法类似。

1. 首先使用 nrfgo 下载协议栈，打开协议栈下载软件 nrfgo,如下图所示，如果仿真器没有连接，nRF5x Programming 显示为灰色：

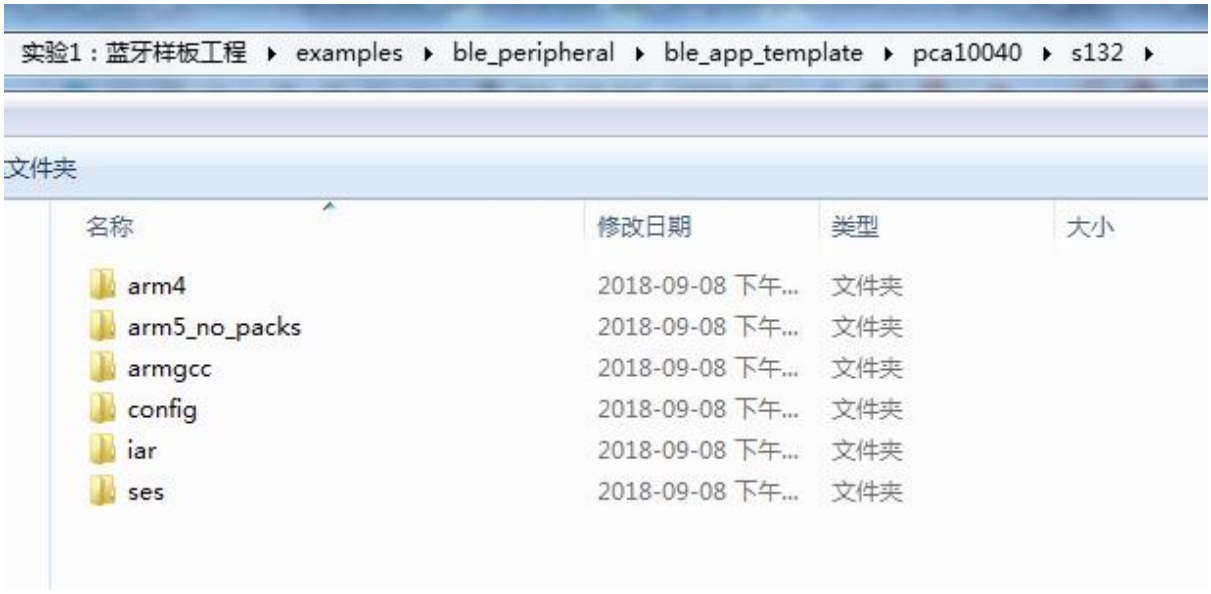


2. 按照《下载错误解决办法里的方法》文档里的方法，把仿真器驱动安装好后，把仿真器和开发连接上，会如下图所示：

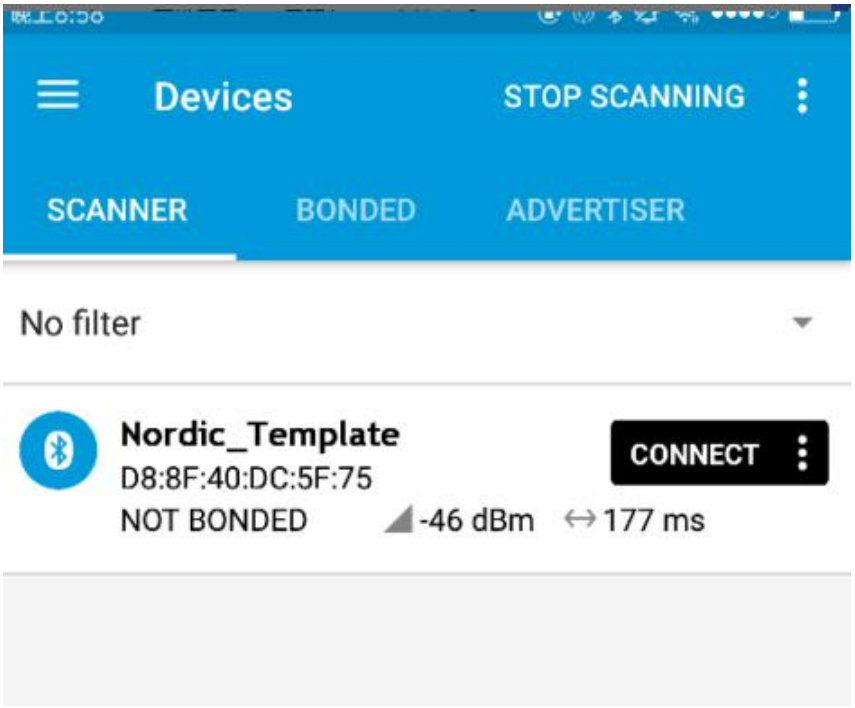


3 选择 Program Application ，点击 Browse 选择添加你要下载的协议版本，然后点击 Program 进行下载：

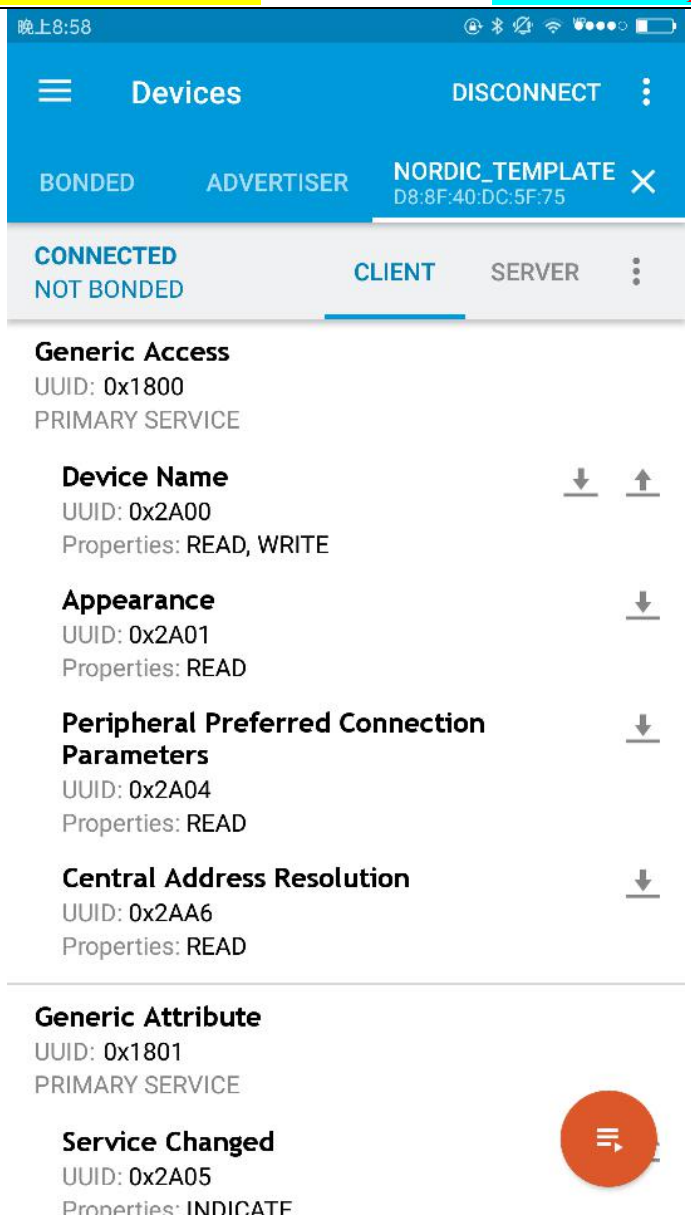
4.下载完协议栈，然后再用 keil 下载工程项目，工程项目目录如下图所示：



5. 安装手机 APP nrf connect,或者应用商店直接下载，安装后打开手机 APP nrf connect 观察如下现象，发现广播名称为 Nordic_Template 的广播信号：



6.点击设备，显示连接成功，LED1 灯熄灭，LED2 灯点亮，查看设备属性，观察相关参数：



蓝牙技术联盟所用的基本 UUID 为 16bit UUID，本例中蓝牙服务为空，我们观察到的是 GAP 基础 UUID 和 GATT 基础 UUID，再搭建一个空服务中，基础 GAP 和 GATT UUID 是底层自带的，在文件 ble_types.h 文件中，对这个 GATT specific UUID 和 GAP specific UUID 有列表出来，如下图所示：

main.c　ble_types.h

```
69  /* Generic UUIDs, applicable to all services */
70  #define BLE_UUID_UNKNOWN                          0x0000 /**< Reserved UUID. */
71  #define BLE_UUID_SERVICE_PRIMARY                  0x2800 /**< Primary Service. */
72  #define BLE_UUID_SERVICE_SECONDARY                0x2801 /**< Secondary Service. */
73  #define BLE_UUID_SERVICE_INCLUDE                  0x2802 /**< Include. */
74  #define BLE_UUID_CHARACTERISTIC                   0x2803 /**< Characteristic. */
75  #define BLE_UUID_DESCRIPTOR_CHAR_EXT_PROP         0x2900 /**< Characteristic Extended Properties Descriptor. */
76  #define BLE_UUID_DESCRIPTOR_CHAR_USER_DESC        0x2901 /**< Characteristic User Description Descriptor. */
77  #define BLE_UUID_DESCRIPTOR_CLIENT_CHAR_CONFIG    0x2902 /**< Client Characteristic Configuration Descriptor. */
78  #define BLE_UUID_DESCRIPTOR_SERVER_CHAR_CONFIG    0x2903 /**< Server Characteristic Configuration Descriptor. */
79  #define BLE_UUID_DESCRIPTOR_CHAR_PRESENTATION_FORMAT  0x2904 /**< Characteristic Presentation Format Descriptor. */
80  #define BLE_UUID_DESCRIPTOR_CHAR_AGGREGATE_FORMAT     0x2905 /**< Characteristic Aggregate Format Descriptor. */
81  /* GATT specific UUIDs */
82  #define BLE_UUID_GATT                             0x1801 /**< Generic Attribute Profile. */
83  #define BLE_UUID_GATT_CHARACTERISTIC_SERVICE_CHANGED  0x2A05 /**< Service Changed Characteristic. */
84  /* GAP specific UUIDs */
85  #define BLE_UUID_GAP                              0x1800 /**< Generic Access Profile. */
86  #define BLE_UUID_GAP_CHARACTERISTIC_DEVICE_NAME       0x2A00 /**< Device Name Characteristic. */
87  #define BLE_UUID_GAP_CHARACTERISTIC_APPEARANCE        0x2A01 /**< Appearance Characteristic. */
88  #define BLE_UUID_GAP_CHARACTERISTIC_RECONN_ADDR       0x2A03 /**< Reconnection Address Characteristic. */
89  #define BLE_UUID_GAP_CHARACTERISTIC_PPCP              0x2A04 /**< Peripheral Preferred Connection Parameters Characteristic. */
90  #define BLE_UUID_GAP_CHARACTERISTIC_CAR               0x2AA6 /**< Central Address Resolution Characteristic. */
91  #define BLE_UUID_GAP_CHARACTERISTIC_RPA_ONLY          0x2AC9 /**< Resolvable Private Address Only Characteristic. */
92  /** @} */
93
```

在 APP 连接广播后，服务中第一部分就是 GAP specific UUID 以及其的特征值，第二部分就是 GATT specific UUID 以及其的特征值，对比 APP 连接参数和上图的定义类表。