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

2.22 蓝牙 BLE 之电池服务

由于蓝牙 BLE 是为低功耗手持设备而生存的，对应电池监测也是必须的，本文将具体讲解电池监测的使用方法。

电池服务是蓝牙兴趣小组所指定的服务，并不是用户需要自建的私有服务，因此可以直接调用官方所提供的电池服务的声明和定义。同时我们需要加入 SAADC 功能。因此我们需要 SAADC 采样的电池电量。

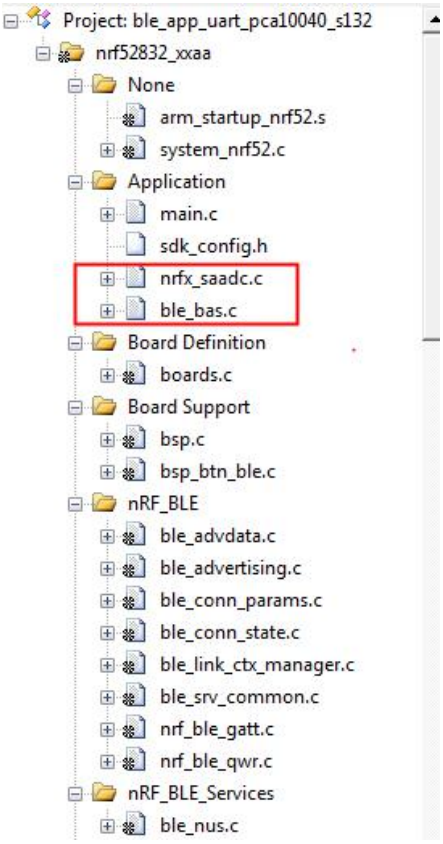
本例在匹配的 SDK15 的蓝牙串口例子基础上就行编写，使用的协议栈为：s132。

1：nRF52832 蓝牙 BLE 电池函数库：

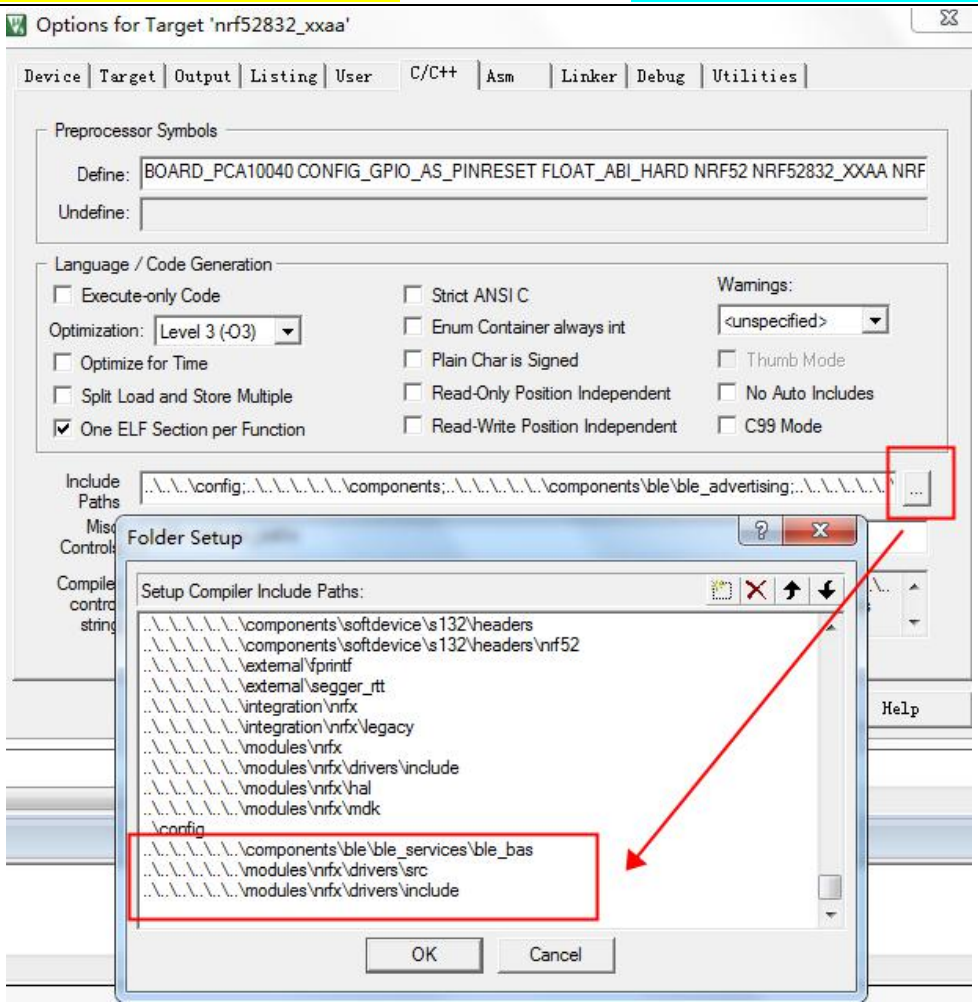
1.1 BLE 电池函数库文件的添加

由于电池服务是蓝牙兴趣小组所指定的服务，所以 nrf52832 的工程中提供了一个 ble_bas.c 的电池任务函数库，在使用电池监测任务时。还有 adc 的库函数文件 nrfx_saadc.c 文件。

在工程目录树中，如下图所示，下面工程中添加这个 ble_bas.c 函数库文件和 nrfx_saadc.c 文件，如下所示：



同时添加该函数文件的路径：



1.2 电池采样设计基本原理：

整个设计思路如下几个步骤：

首先，我们需要在服务初始化中，声明电池服务。

电池的容量值我们肯定是需要定时更新的采集，采样是给一个时间段进行一次，因此肯定需要设置一个定时器进行定时，因此在电池服务的派发函数中，通知使能情况下启动定时器定时采集。

如果初始化了 adc 之后，定时器启动了，会开始采集 adc 数据，采集一次 adc 数据，会触发一次 adc 中断，在中断里，对采集到的 adc 数值进行电压转换，然后把电压转换成电池等级。

最后把电池等级参数上传给主机，在 bas.c 中提供了一个函数：uint32_t ble_bas_battery_level_update(ble_bas_t * p_bas, uint8_t battery_level)

这个函数可以对电池容量进行更新。这个函数中通过 SD 协议栈函数：

sd_ble_gatts_hvx(p_bas->conn_handle, &hvx_params)把值回传个手机。

总结如下图所示：

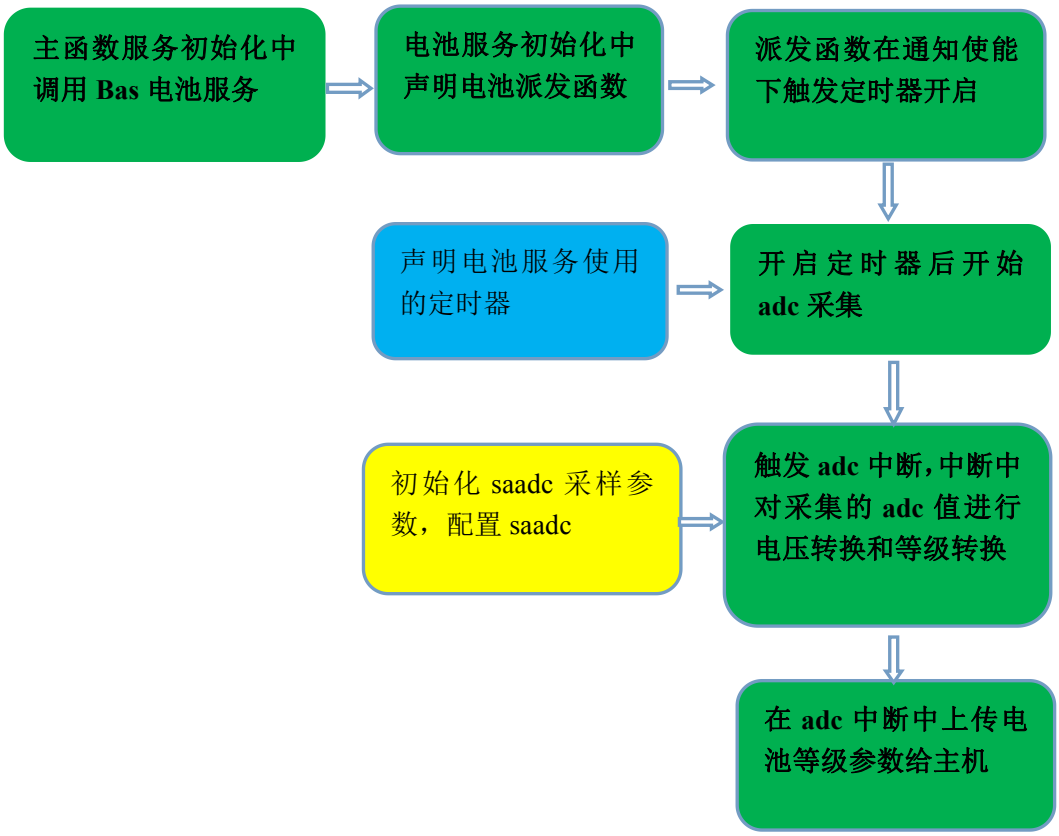


图 1 电池采集的原理

2：函数编写：

2.1 电池服务的调用

电池的任务我们不需要自己来建立了，电池服务属于已经提供的公有服务。SIG 的公有服务函数在 nrf 的 BLE 函数库里是直接提供了初始化函数 ble_bas_init，不需要再来写了，可以直接在函数声明中进行调用。调用如下，把电池电量分成 100 等级，编写电池服务的声明和初始化函数：

```
01.     static void bas_init(void)
02. {
03.     ret_code_t         err_code;
04.     ble_bas_init_t bas_init_obj;
05.     //清空电池初始化结构体
06.     memset(&bas_init_obj, 0, sizeof(bas_init_obj));
07.     //配置电池初始化函数
08.     bas_init_obj.evt_handler = on_bas_evt;//电池回调处理函数
09.     bas_init_obj.support_notification = true;//空中规范：通知使能
10.     bas_init_obj.p_report_ref = NULL;//报告无
```

```
    bas_init_obj.initial_batt_level = 100;//电池分级

    BLE_GAP_CONN_SEC_MODE_SET_OPEN//CCCD 写开放
                                  (&bas_init_obj.battery_level_char_attr_md.cccd_write_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN//读开放
                                   (&bas_init_obj.battery_level_char_attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS//不允许写
                                   (&bas_init_obj.battery_level_char_attr_md.write_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN//报告开放
                                   (&bas_init_obj.battery_level_report_read_perm);

    err_code = ble_bas_init(&m_bas, &bas_init_obj);//调用电池初始化函数
    APP_ERROR_CHECK(err_code);
}
```

由于本例在 SDK15.0 下的串口蓝牙例子下进行修改，电池初始化声明后，需要再服务初始化函数中调用，同时在广播初始化中的 UUID 中添加电池 UUID 类型，关于 UUID 的设置看专门的一讲教程《蓝牙 UUID 设置与总结》：

```
static void services_init(void)//服务初始化
{
    uint32_t           err_code;
    ble_nus_init_t     nus_init;
    nrf_ble_qwr_init_t qwr_init = {0};
    qwr_init.error_handler = nrf_qwr_error_handler;
    err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
    APP_ERROR_CHECK(err_code);

    // 初始化串口
    memset(&nus_init, 0, sizeof(nus_init));
    nus_init.data_handler = nus_data_handler;
    err_code = ble_nus_init(&m_nus, &nus_init);
    APP_ERROR_CHECK(err_code);

    bas_init();//添加电池服务声明
}

  //设置电池服务的 UUID
        static ble_uuid_t m_adv_uuids[]=
  {
    {BLE_UUID_NUS_SERVICE, NUS_SERVICE_UUID_TYPE},
    {BLE_UUID_BATTERY_SERVICE, BLE_UUID_TYPE_BLE}//电池的 UUID
  };
```

2.2 电池参数值的采集

电池公共服务建立后，需要对电池电量进行采集，如果采集电池电量？通过 saadc 功能，采集 vcc 的电压大小，然后进行分级，下面来看下在蓝牙 BLE 下如何启动 saadc 的采集。

对应 saadc 的使用，在外设教程里专门有一篇教程进行了讲述。我们边回忆边来看如果在 ble 下使用。首先配置 adc 的基本参数：代码如下：

```
01.      static void adc_configure(void)
02. {
03.      //adc 初始化
04.     ret_code_t err_code = nrf_drv_saadc_init(NULL, saadc_event_handler);
05.     APP_ERROR_CHECK(err_code);
01.     //adc 通道配置
06.     nrf_saadc_channel_config_t config =
07.       NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_VDD);
08.   //adc 通道初始化，带入前面的通道配置结构体
09.     err_code = nrf_drv_saadc_channel_init(0, &config);
10.     APP_ERROR_CHECK(err_code);
11.      //配置第一个缓冲
12.     err_code = nrf_drv_saadc_buffer_convert(&adc_buf[0], 1);
13.     APP_ERROR_CHECK(err_code);
14.     //配置第二个缓冲
15.     err_code = nrf_drv_saadc_buffer_convert(&adc_buf[1], 1);
16.     APP_ERROR_CHECK(err_code);
17. }
```

第 4 行：nrf_drv_saadc_init 函数对 saadc 进行初始化，第一样形参 NULL,表示默认使用 sdk_config.h 配置文件中的设置。第二个形参为 saadc_callback，设置为 saadc 回调函数，回调中断函数可以为空，什么都不执行。如果有中断任务需要执行，可以在 saadc_callback 内写中断函数，本例中，我们需要在中断函数中执行电池电量上传手机更新的操作。

第 6~10 行：配置 adc 的采样通道，电池电量采集，可以直接采用单端输入采集 vdd 的端口的电量，所有可以直接声明为 NRF_SAADC_INPUT_VDD 端口。然后通过 nrf_drv_saadc_channel_init 配置函数配置到结构体。（具体看外设教程说明），如下所示

```
01. #define NRFX_SAADC_DEFAULT_CHANNEL_CONFIG_SE(PIN_P) \
02. {                                                   \
03.     .resistor_p = NRF_SAADC_RESISTOR_DISABLED,      \
04.     .resistor_n = NRF_SAADC_RESISTOR_DISABLED,      \
05.     .gain       = NRF_SAADC_GAIN1_6,                \
06.     .reference  = NRF_SAADC_REFERENCE_INTERNAL,     \
07.     .acq_time   = NRF_SAADC_ACQTIME_10US,           \
08.     .mode       = NRF_SAADC_MODE_SINGLE_ENDED,      \
09.     .burst      = NRF_SAADC_BURST_DISABLED,         \
10.     .pin_p      = (nrf_saadc_input_t)(PIN_P),       \
11.     .pin_n      = NRF_SAADC_INPUT_DISABLED          \
12. }
```

上面的参数配置代码解释如下：

3 行：正端输入：SAADC 的旁路电阻关

4 行：负端输入：SAADC 的旁路电阻关

5 行：增益：SAADC 的增益为 1/6

6 行：参考电压值：采用芯片内部参考电压

7 行：采样时间：10us

8 行：模式：单端输入

9 行：正端输入引脚：输入管脚

10 行：负端输入引脚：输入管脚关闭。

第 11~16 行：设置双缓冲方式采集数据，同时声明两个 buff 缓冲存放数据。

上面这段配置函数基本上在外设篇里有详细讲解，我们只有注意下配置函数里的默认参数是多少就可以了，我们进去看下：

进入到第 4 行的函数 nrf_drv_saadc_init 函数里面去，我们发现这个默认配置就是 NRFX_SAADC_DEFAULT_CONFIG 结构体：

```
126 __STATIC_INLINE ret_code_t nrf_drv_saadc_init(nrf_drv_saadc_config_t const * p_config,
127                                               nrf_drv_saadc_event_handler_t  event_handler)
128 {
129     if (p_config == NULL)
130     {
131         static const nrfx_saadc_config_t default_config = NRFX_SAADC_DEFAULT_CONFIG;
132         p_config = &default_config;
133     }
134     return nrfx_saadc_init(p_config, event_handler);
135 }
136
```

打开 NRFX_SAADC_DEFAULT_CONFIG 结构体定义，定义了如下参数：

```
01. #define NRFX_SAADC_DEFAULT_CONFIG
02. {
03.         .resolution = (nrf_saadc_resolution_t)NRFX_SAADC_CONFIG_RESOLUTION,
04.         .oversample= (nrf_saadc_oversample_t)NRFX_SAADC_CONFIG_OVERSAMPLE,
05.         .interrupt_priority = NRFX_SAADC_CONFIG_IRQ_PRIORITY,
06.         .low_power_mode= NRFX_SAADC_CONFIG_LP_MODE
07. }
```

03 行：采样分辨率，默认是 10bit

04 行：设置是否是过采样，如果是过采样这设置采样周期

05 行：设置 adc 的中断优先级的级别

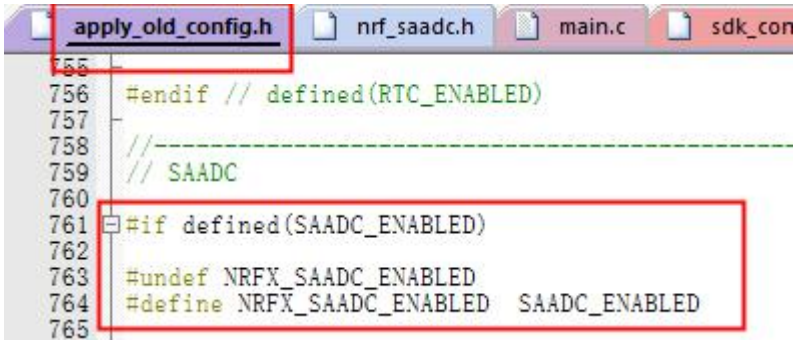
06 行：设置 adc 是否为低功耗模式

同时注意 sdk_config.h 函数中，需要对 saadc 进行使能：

```
// <e> SAADC_ENABLED - nrf_drv_saadc - SAADC peripheral driver - legacy layer
//==========================================================
#ifndef SAADC_ENABLED
#define SAADC_ENABLED 1
#endif
```

```
// <e> NRFX_SAADC_ENABLED - nrfx_saadc - SAADC peripheral driver
//==========================================================
#ifndef NRFX_SAADC_ENABLED
#define NRFX_SAADC_ENABLED 1
#endif
```

这两个配置其实就是为了兼容老版本点给的一个补丁定义，最好两个全部使能：



2.3 电池电量的更新：

前面设置了 Adc 采集后会触发 adc 中断，也就是 saadc_callback 参数，该参数被命名为 saadc_event_handler 中断处理，这个处理函数主要是实现采集的电压转换为电量，并且上传给主机的功能，代码如下：

```
01. void saadc_event_handler(nrf_drv_saadc_evt_t const * p_event)
02. {
03.     if (p_event->type == NRF_DRV_SAADC_EVT_DONE)
04.     {
05.         nrf_saadc_value_t adc_result;
06.         uint16_t          batt_lvl_in_milli_volts;
07.         uint8_t           percentage_batt_lvl;
08.         uint32_t          err_code;
09.         //设置好缓存，为下次转换预备缓冲，并且把导入到缓冲的值提取出来
10.         adc_result = p_event->data.done.p_buffer[0];
11.         err_code = nrf_drv_saadc_buffer_convert(p_event->data.done.p_buffer, 1);
12.         APP_ERROR_CHECK(err_code);
13.         //电池电量转换计算
14.         batt_lvl_in_milli_volts = ADC_RESULT_IN_MILLI_VOLTS(adc_result) +
15.                                   DIODE_FWD_VOLT_DROP_MILLIVOLTS;
16.         percentage_batt_lvl = battery_level_in_percent(batt_lvl_in_milli_volts);
17.         //电池电量上传主机
18.         err_code =
19.         ble_bas_battery_level_update(&m_bas, percentage_batt_lvl, BLE_CONN_HANDLE_ALL);
20.          if ((err_code != NRF_SUCCESS) &&
21.              (err_code != NRF_ERROR_INVALID_STATE) &&
22.              (err_code != NRF_ERROR_RESOURCES) &&
23.              (err_code != NRF_ERROR_BUSY) &&
24.              (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
25.             )
26.          {
27.              APP_ERROR_HANDLER(err_code);
28.          }
29.     }
```

```
30. }
```

对上面代码进行如下解释：

第 10~12 行：设置好缓存，为下次转换预备缓冲。并且把导入到缓冲的值提取出来。

第 14~16 行：对 adc 采样到的电量进行转换。

ADC_RESULT_IN_MILLI_VOLTS 函数如下所示：

```
01. #define ADC_RESULT_IN_MILLI_VOLTS(ADC_VALUE)\
02.         ((((ADC_VALUE) * ADC_REF_VOLTAGE_IN_MILLIVOLTS) / ADC_RES_10BIT) *
    ADC_PRE_SCALING_COMPENSATION)
```

根据外设部分的讲解，我们可以根据前面的计算公式计算出来电压，

$$RESULT = [V(P) - V(N)] * GAIN/REFERENCE * 2^{(RESOLUTION - m)}$$

VP 为 要正端采样的电压，VN 为负向端采样电压为 0，GAIN 为 1/6，REFERENCE 为内部参考电压 0.6，RESOLUTION 为 10，m 为 0，RESULT 是 saadc_val 结果。

因为是单端输入，那么 VP=(RESULT*REFERENCE/2^{10})*6 为实际电压。

测量的电池电量为采样值+偏移量，公式如下：

```
batt_lvl_in_milli_volts = ADC_RESULT_IN_MILLI_VOLTS(adc_result) +
                                        DIODE_FWD_VOLT_DROP_MILLIVOLTS;
```

然后把电压划分成等级，等级划分函数如下：

```
03. static __INLINE uint8_t battery_level_in_percent(const uint16_t mvolts)
04. {
05.     uint8_t battery_level;
06.     //当电压大于 3v 的时候，认为电量是满的
07.     if (mvolts >= 3000)
08.     {
09.         battery_level = 100;
10.     } //对电压进行等级划分
11.     else if (mvolts > 2900)
12.     {
13.         battery_level = 100 - ((3000 - mvolts) * 58) / 100;
14.     }
15.     else if (mvolts > 2740)
16.     {
17.         battery_level = 42 - ((2900 - mvolts) * 24) / 160;
18.     }
19.     else if (mvolts > 2440)
20.     {
21.         battery_level = 18 - ((2740 - mvolts) * 12) / 300;
22.     }
23.     else if (mvolts > 2100)
24.     {
25.         battery_level = 6 - ((2440 - mvolts) * 6) / 340;
26.     } //小于 2.1v 的认为电压为 0
27.     else
```

```
28.         {
29.             battery_level = 0;
30.         }
31.         return battery_level;
32.  }
```

最后，把划分好的电池等级参数值，上传到主机，在 bas.c 的服务函数中，提供一个电池上传函数：

```
ble_bas_battery_level_update(ble_bas_t * p_bas,    uint8_t         battery_level,
                                                   uint16_t        conn_handle)
```

该函数和之前的蓝牙按键例子里的按键参数通知上传函数类似。

2.4 定时器设计

为了节省功耗，我们不是总是进行 adc 采样，我们需要设定一个时间，定时进行采样，采样后触发中断，进行电量转换和电量级别上传。这里就可以用到软件定时器。设计定时器了，其实定时器在协议栈下的使用，我们在之前的蓝牙心电里已经接触过，这里面再来讲一讲，首先设置一个定时器，函数如下：

```
static void timers_init(void)
{
    uint32_t err_code;
    // Initialize timer module, making it use the scheduler.
    APP_TIMER_INIT(APP_TIMER_PRESCALER,     APP_TIMER_OP_QUEUE_SIZE,
false);

    err_code = app_timer_create(&m_battery_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                battery_level_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);
}
```

上面主要关注 app_timer_create 函数，这个函数是创建一个定时器超时中断处理函数，需要处理的是里只需要开启 adc 的采样：

```
static void battery_level_meas_timeout_handler(void * p_context)
{
    UNUSED_PARAMETER(p_context);

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

声明完定时器后需要开始运行定时器，设置定时器运行更新的时间间隔，函数如下所示。

```
static void on_bas_evt(ble_bas_t * p_bas, ble_bas_evt_t * p_evt)
{
    ret_code_t err_code;

    switch (p_evt->evt_type)
    {
        case BLE_BAS_EVT_NOTIFICATION_ENABLED:
            // 启动定时器
            err_code=
    app_timer_start(m_battery_timer_id, BATTERY_LEVEL_MEAS_INTERVAL, NULL);
            APP_ERROR_CHECK(err_code);
            break; // BLE_BAS_EVT_NOTIFICATION_ENABLED

        case BLE_BAS_EVT_NOTIFICATION_DISABLED:
            err_code = app_timer_stop(m_battery_timer_id);
            APP_ERROR_CHECK(err_code);
            break; //关掉定时器
        default:
            // No implementation needed.
            break;
    }
}
```

上面我们在派发函数中启动定时器，on_bas_evt 派发函数在前面初始化电池服务中已经声明，派发回调函数表示产生定时事件后，我们执行对应的操作。

当蓝牙服务发生 BLE_BAS_EVT_NOTIFICATION_ENABLED 通知使能事件，我们就开启定时器。

当发生 BLE_BAS_EVT_NOTIFICATION_DISABLED 使能关闭事件后，我们就停止定时器，定时器的时间间隔为 **BATTERY_LEVEL_MEAS_INTERVAL，也就是说开启定时器后，在这个时间间隔内进行采样，直到定时器关闭。**

2.5 主函数编写

主函数写一个测试函数，用于把我们写入电池服务函数，主要是 adc 初始化需要声明，编写代码如下：

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

    // Initialize.
    uart_init();
    log_init();
    timers_init();
    buttons_leds_init(&erase_bonds);
```

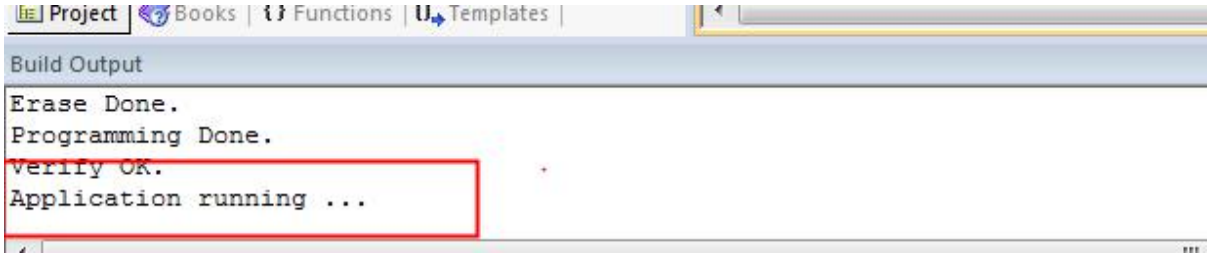
```
    adc_configure();
    power_management_init();
    ble_stack_init();
    gap_params_init();
    gatt_init();
    services_init();
    advertising_init();
    conn_params_init();

    // Start execution.
    printf("\r\nUART started.\r\n");
    NRF_LOG_INFO("Debug logging for UART over RTT started.");
    advertising_start();
    // Enter main loop.
    for (;;)
    {
        idle_state_handle();
    }
}
```

修改后编译通过，提示 OK

3 应用与调试

3.1 下载

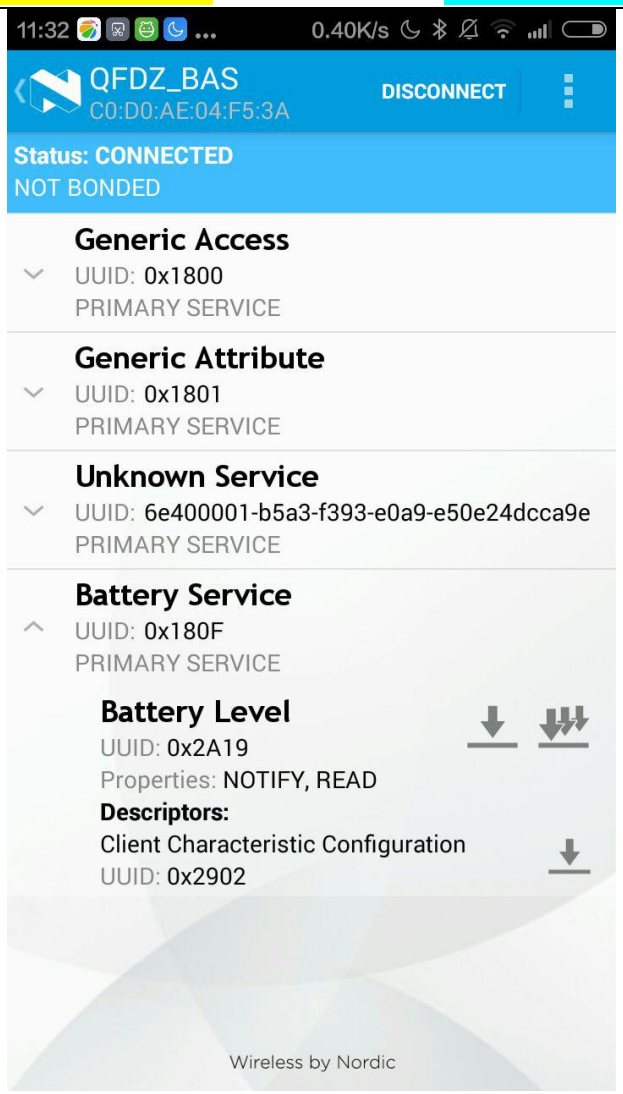
本例使用的协议栈为 S132 版本，首先采用 nrfgo 整片擦除，后下载协议栈。下载完后可以下载工程，首先把工程编译一下，通过后点击 KEIL 上的下载。下载成功后提示如图，程序开始运行，同时开发板上广播 LED 开始广播：



3.2 测试

本实验采用手机写 mcp app 软件，返现电池服务应用名 QFDZ_BAS,如下图所示：

点击打开，找到电池应用 Battey Service，如下图所示：



特性：可读，通知类型：点击通知：

通知刷新，如果电量有变化，可以比较变化值：