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

第 20 章 蓝牙心电任务的建立

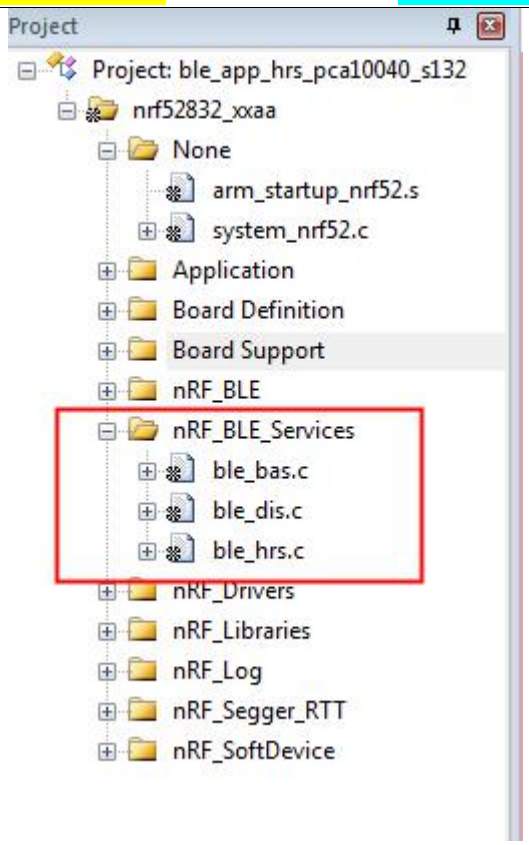
BLE 心电应用示例学习必须在你之前认真解读了前面几讲详解里的内容为基础后进行。这里特别注意。本节将结合实例代码，一步一步的通过原理分析和讲解，再次带大家深入到编 写蓝牙应用的这个过程中，学习的时候大家一定要多对照理论进行学习，从而深入理 解代码，为自己独立编写应用打下基础。

官方的心率例子实现并不是真的使用的心率传感器，而是使用模拟的，留下数据接口，同时实现手机端 app 看到的数值变化。另外该实验中还建立了另外两个服务，一个是电池服务，以及一个设备信息服务。

20.1 工程项目的建立：

本例工程对比工程搭建篇里的工程样例，对比两份工程项目，分析使用了哪些文件，哪些文件未使用，工程目录如下图所示：

对比两个工程数，蓝牙串口需要添加了三个 APP 服务应用文件，开通三个服务，分别为 1.心电服务 ble_hrs.c 文件 2.电池服务 ble_bas.c 文件 3.设备信息服务 ble_dis 文件，同时在工程设置里添加文件路径。其中电池服务在上一章内容中已经详细讲过。下面的本章重点就是来教大家在应用层里如何调用心电服务函数文件。同时大家需要具体根据工程对比本讲解进行阅读。



20.1.1 主函数的建立

nrf52832 蓝牙工程的主函数有着一定的通用性，初始化过程类似，需要修改的就是初始化过程中的子函数，先看看本列的主函数流程：

```
1.  int main(void)
2.  {       bool erase_bonds;
3.  //外设部分初始化
4.      log_init();//log 打印初始化
5.      timers_init();//定时器初始化
6.      buttons_leds_init(&erase_bonds);//外初始化
7.      ble_stack_init(); //协议栈初始化
8.      gap_params_init();//gap  参数的初始化
9.      gatt_init();//GATT 初始化
10.     advertising_init();//广播参数初始化
11.     services_init();//服务初始化，初始化上面提到的三个服务
12.     sensor_simulator_init();//函数初始化传感器模拟
13.     conn_params_init();//连接参数相关初始化
14.     peer_manager_init();//设备管理初始化
15.
16.      application_timers_start();//开启定时器
17.      advertising_start(erase_bonds);//开始广播
18. // 进入主循环待机
19.       for (;;)
20.       {
```

```
21.             idle_state_handle();
22. }
23. }
```

关于工程的主程序框架搭建在前面几章里已经讲过，对这个过程一再重复，下面我们是来看看本例有哪些不同，主函数中红色标注的三个函数需要变化的。

timers_init()函数作为软件定时器，本章内定时器开启了多个定时器。S

ervices_init()服务初始化中，由于本章有 3 个公有服务，因此这个地方也需要添加服务初始化。

sensor_simulator_init()传感器初始化，本例没有接入传感器，而是仅仅进行模拟，如果需要添加自己的传感器，需要改动本函数。下面对着几个地方一一进行讨论。

20.1.2 外设部分初始化

外设的初始化不涉及蓝牙协议，仅仅是对外设功能的初始化，在前面外设篇我们 已经把外设功能和大家学过了一篇，这里大家应该能够熟练的调用和编写相关函数。

里面还需要提一下 timers_init(void)函数，初始化外设最后调用了这个函数，这个定时器初始化里调用了函数 app_timer_create，如下图所示：

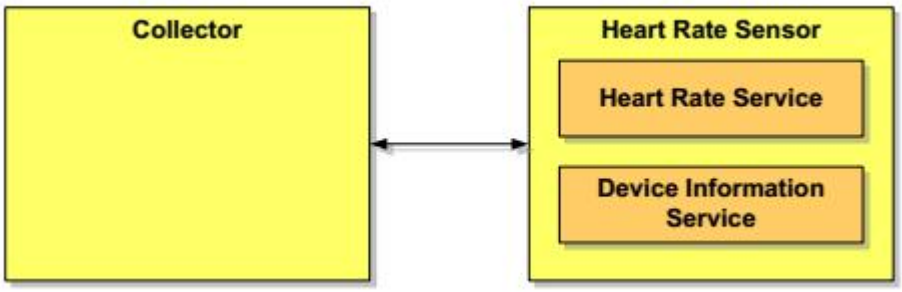
```
452  static void timers_init(void)
453  {
454      ret_code_t err_code;
455
456      // Initialize timer module.
457      err_code = app_timer_init();
458      APP_ERROR_CHECK(err_code);
459
460      // Create timers.
461      err_code = app_timer_create(&m_battery_timer_id,
462                                  APP_TIMER_MODE_REPEATED,
463                                  battery_level_meas_timeout_handler);
464      APP_ERROR_CHECK(err_code);
465
466      err_code = app_timer_create(&m_heart_rate_timer_id,
467                                  APP_TIMER_MODE_REPEATED,
468                                  heart_rate_meas_timeout_handler);
469      APP_ERROR_CHECK(err_code);
470
471      err_code = app_timer_create(&m_rr_interval_timer_id,
472                                  APP_TIMER_MODE_REPEATED,
473                                  rr_interval_timeout_handler);
474      APP_ERROR_CHECK(err_code);
475
476      err_code = app_timer_create(&m_sensor_contact_timer_id,
477                                  APP_TIMER_MODE_REPEATED,
478                                  sensor_contact_detected_timeout_handler);
479      APP_ERROR_CHECK(err_code);
480  }
481
```

app_timer_create 函数是创建一个定时器，在前面的软件定时器详解篇已经讲述过。这个定时器可以实现指定的定时功能，本例中使用 app_timer_create 函数创建了 4 个定时器，包括电池更新、心率测量、RR 间隔、传感器连接等等。其中与后面三个部分与心电服务相关。后面结合心电服务详细探讨。

20.1.3 服务初始化

协议栈的初始化流程和样例里介绍的流程基本相似，这里就不再累述，这里主要来说明一下服务初始化中的不同之处，哪里需要改变的？首先，阅读 SIG 的蓝牙心电规范，对心电基础服务定义为两个部

分，一个是心电服务，一个是设备信息服务，如下图所示：



关键的变化是进入服务初始化函数services_init()这里需要进行修改，需要自己添加SIG 服务。而这个添加的服务也就是我们需要开通的服务。本例里应该是**开通三个服务，分别为1.心电服务2.电池服务3.设备信息服务。设置如下：**

```
1.  static void services_init(void)
2.  {
3.      ret_code_t         err_code;
4.      ble_hrs_init_t     hrs_init;
5.      ble_bas_init_t     bas_init;
6.      ble_dis_init_t     dis_init;
7.      nrf_ble_qwr_init_t qwr_init = {0};
8.      uint8_t            body_sensor_location;
9.
10.     // Initialize Queued Write Module.
11.     qwr_init.error_handler = nrf_qwr_error_handler;
12.
13.     err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
14.     APP_ERROR_CHECK(err_code);
15.
16.     // Initialize Heart Rate Service.
17.     body_sensor_location = BLE_HRS_BODY_SENSOR_LOCATION_FINGER;
18.
19.     memset(&hrs_init, 0, sizeof(hrs_init));
20.
21.     hrs_init.evt_handler                 = NULL;
22.     hrs_init.is_sensor_contact_supported = true;
23.     hrs_init.p_body_sensor_location      = &body_sensor_location;
24.
25.     // Here the sec level for the Heart Rate Service can be changed/increased.
26.     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&hrs_init.hrs_hrm_attr_md.cccd_write_perm);
27.     BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&hrs_init.hrs_hrm_attr_md.read_perm);
28.     BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&hrs_init.hrs_hrm_attr_md.write_perm);
29.
30.     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&hrs_init.hrs_bsl_attr_md.read_perm);
31.     BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&hrs_init.hrs_bsl_attr_md.write_perm);
32.
33.     err_code = ble_hrs_init(&m_hrs, &hrs_init);//心电服务初始化
34.     APP_ERROR_CHECK(err_code);
35.
36.     // Initialize Battery Service.
```

```
37.     memset(&bas_init, 0, sizeof(bas_init));
38.
39.     // Here the sec level for the Battery Service can be changed/increased.
40.     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_char_attr_md.cccd_write_perm);
41.     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_char_attr_md.read_perm);
42.     BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&bas_init.battery_level_char_attr_md.write_perm);
43.
44.     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_report_read_perm);
45.
46.     bas_init.evt_handler                = NULL;
47.     bas_init.support_notification = true;
48.     bas_init.p_report_ref               = NULL;
49.     bas_init.initial_batt_level     = 100;
50.
51.     err_code = ble_bas_init(&m_bas, &bas_init);//电池服务初始化
52.     APP_ERROR_CHECK(err_code);
53.
54.     // Initialize Device Information Service.
55.     memset(&dis_init, 0, sizeof(dis_init));
56.
57.     ble_srv_ascii_to_utf8(&dis_init.manufact_name_str, (char *)MANUFACTURER_NAME);
58.
59.     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&dis_init.dis_attr_md.read_perm);
60.     BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&dis_init.dis_attr_md.write_perm);
61.
62.     err_code = ble_dis_init(&dis_init);//设备信息服务初始化
63.     APP_ERROR_CHECK(err_code);
```

上面这段代码最主要三个内为：

ble_hrs_init(&m_hrs, &hrs_init);　　心率服务初始化

ble_bas_init(&bas, &bas_init);　　电池服务初始化

ble_dis_init(&dis_init);　　设备信息服务初始化

这个是在样例基础上添加的三个服务，根据 SIG 心电配置规范规定如下表所示，心电服务和设备信息服务是必选的，下面我们就具体进行讨论这两个服务。

Service	Heart Rate Sensor
Heart Rate Service	M
Device Information Service	M

Table 3.1: Heart Rate Sensor Service Requirements

20.2 蓝牙服务程序设计

20.2.1 心率服务设计

心电服务是本讲的重点。我们可以在蓝牙样板框架上添加的驱动 ble_hrs .h 头文件实现了各种数据结构、应用需要实现的事件句柄和 API 函数,下面来介绍：

首先深入到心电服务初始化函数 ble_hrs_init 中进行探讨，虽然这个函数是官方已经给出的驱动函数，但是理解并且会修改它是我们后面加入传感器的关键，该服务初始化定义如下所示：

```
1.  uint32_t ble_hrs_init(ble_hrs_t * p_hrs, const ble_hrs_init_t * p_hrs_init);
```

这个函数内部实现的了如下几个功能：

1:初始化心率服务结构体并添加服务 UUID，心率服务结构体主要初始化心电服务需要的事件处理和连接的句柄，添加服务 GATT 句柄，关于 GATT 原理在前面的章节最前面详细说明了，还定义了一些需要使用的参数。

服务 UUID 使用的是 SIG 定义的公共服务的 UUID。这里面的子服务都是属于蓝牙兴趣小组规定的通用服务，分配了专用的 128BIT UUID 和 16 位 UUID。

```
2.      //初始化服务结构体
3.       //心电事件句柄定义
4.       p_hrs->evt_handler                 = p_hrs_init->evt_handler;
5.      //是否连接传感器
6.       p_hrs->is_sensor_contact_supported = p_hrs_init->is_sensor_contact_supported;
7.      //心电连接句柄定义
8.       p_hrs->conn_handle                 = BLE_CONN_HANDLE_INVALID;
9.       //是否剔除传感器连接
10.      p_hrs->is_sensor_contact_detected  = false;
11.      //RR 值的计数值
12.      p_hrs->rr_interval_count           = 0;
13.      //心电的最大数据长度
14.      p_hrs->max_hrm_len                 = MAX_HRM_LEN;
15.
16.      //添加心率服务的 UUID
17.      BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_HEART_RATE_SERVICE);
18.      err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
19.                                          &ble_uuid,
20.                                          &p_hrs->service_handle);
```

2：心率测量的特征值添加，这点后面继续展开：

```
21.       //心率测量特征值
22.    heart_rate_measurement_char_add(p_hrs, p_hrs_init);
```

3：身体位置传感特征值添加：

```
23.    if (p_hrs_init->p_body_sensor_location != NULL)
24.      {
25.           // 添加体位传感器的位置特征值
26.           err_code = body_sensor_location_char_add(p_hrs, p_hrs_init);
```

```
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }
    }
```

首先来看下心电测量特征的添加函数。上面 2 个子服务，是主要和传感器接口对应的接口函数，我们具体介绍心率测量的特征值添加，关于特征的添加在前面几讲服务建立里都啰嗦过一遍，这里再不重复，我们看和传感器有关的关键函数：

```
static uint32_t heart_rate_measurement_char_add(ble_hrs_t                  * p_hrs,
                                                const ble_hrs_init_t * p_hrs_init)
{
    ble_gatts_char_md_t char_md;//特征值
    ble_gatts_attr_md_t cccd_md;//CCCD 特征值
    ble_gatts_attr_t    attr_char_value;//特征值属性值
    ble_uuid_t          ble_uuid;//服务 UUID
    ble_gatts_attr_md_t attr_md;//特征值属性
    uint8_t             encoded_initial_hrm[MAX_HRM_LEN];//数据结构体

    memset(&cccd_md, 0, sizeof(cccd_md));

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
    cccd_md.write_perm = p_hrs_init->hrs_hrm_attr_md.cccd_write_perm;
    cccd_md.vloc       = BLE_GATTS_VLOC_STACK;    //设置 CCCD 写允许，可以通知使能

    memset(&char_md, 0, sizeof(char_md));//配置特征值

    char_md.char_props.notify = 1;   //特征值为通知属性
    char_md.p_char_user_desc  = NULL;
    char_md.p_char_pf         = NULL;
    char_md.p_user_desc_md    = NULL;
    char_md.p_cccd_md         = &cccd_md;
    char_md.p_sccd_md         = NULL;

    BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_HEART_RATE_MEASUREMENT_CHAR);
    //添加心电测量子服务的 UUID

    memset(&attr_md, 0, sizeof(attr_md));//设置特征值属性值

    attr_md.read_perm  = p_hrs_init->hrs_hrm_attr_md.read_perm;
    attr_md.write_perm = p_hrs_init->hrs_hrm_attr_md.write_perm;
    attr_md.vloc       = BLE_GATTS_VLOC_STACK;
    attr_md.rd_auth    = 0;
    attr_md.wr_auth    = 0;
    attr_md.vlen       = 1;

    memset(&attr_char_value, 0, sizeof(attr_char_value));//设置特征值属性

    attr_char_value.p_uuid    = &ble_uuid;//把服务 UUID 设置进去
    attr_char_value.p_attr_md = &attr_md;//特征值属性值设置进去
    attr_char_value.init_len  = hrm_encode(p_hrs, INITIAL_VALUE_HRM, encoded_initial_hrm);
```

```
74.     //数据长度
75.     attr_char_value.init_offs = 0;//偏移量
76.     attr_char_value.max_len     = MAX_HRM_LEN;//数据最大长度
77.     attr_char_value.p_value     = encoded_initial_hrm;//数据存储结构体
78.
79.     return sd_ble_gatts_characteristic_add(p_hrs->service_handle,
80.                                            &char_md,
81.                                            &attr_char_value,
82.                                            &p_hrs->hrm_handles);//添加该特征值相关参数
}
```

心电测量子服务的特征值定义空中属性为通知类型，也就是说传感器测量的数据应该和按键通知例子类似，通过通知的方式上传主机。在上 面 特征值属性定义中使用了函数：

uint8_t hrm_encode(ble_hrs_t * p_hrs, uint16_t heart_rate, uint8_t * p_encoded_buffer)就是心电的参数的编码函数，函数返回值为数据长度，这就是关键的和我们传感器相关的函数，可以添加自己的传感器进来。进入该函数来一探究竟：

```
1.  static uint8_t hrm_encode(ble_hrs_t * p_hrs, uint16_t heart_rate, uint8_t * p_encoded_buffer)
2.  {
3.      uint8_t flags = 0;
4.      uint8_t len     = 1;
5.      int     i;
6.
7.      // 设置传感器连接标志
8.      if (p_hrs->is_sensor_contact_supported)
9.      {
10.         flags |= HRM_FLAG_MASK_SENSOR_CONTACT_SUPPORTED;//提示传感器连接成功
11.     }
12.     if (p_hrs->is_sensor_contact_detected)
13.     {
14.         flags |= HRM_FLAG_MASK_SENSOR_CONTACT_DETECTED;//提示传感器剔除
15.     }
16.
17.     //编码心率测量值
18.     if (heart_rate > 0xff)//如果大于 8 位数据则转码
19.     {
20.         flags |= HRM_FLAG_MASK_HR_VALUE_16BIT;//心率数据 bit 值
21.         len     += uint16_encode(heart_rate, &p_encoded_buffer[len]);//16 位数据转 8 位数据
22.     }
23.     else
24.     {
25.         p_encoded_buffer[len++] = (uint8_t)heart_rate;//否则直接赋值
26.     }
27.
28.     //编码 rr_interval 值
29.     if (p_hrs->rr_interval_count > 0)
30.     {
31.         flags |= HRM_FLAG_MASK_RR_INTERVAL_INCLUDED;
32.     }
33.     for (i = 0; i < p_hrs->rr_interval_count; i++)
34.     {
35.         if (len + sizeof(uint16_t) > p_hrs->max_hrm_len)
```

```
            {
                // Not all stored rr_interval values can fit into the encoded hrm,
                // move the remaining values to the start of the buffer.
                memmove(&p_hrs->rr_interval[0],
                        &p_hrs->rr_interval[i],
                        (p_hrs->rr_interval_count - i) * sizeof(uint16_t));
                break;
            }
            len += uint16_encode(p_hrs->rr_interval[i], &p_encoded_buffer[len]);
        }
        p_hrs->rr_interval_count -= i;

        // Add flags
        p_encoded_buffer[0] = flags;

        return len;
    }
```

进入这个函数后，认真阅读这个函数后，你会发现这个函数的事件功能是把传感器传递后来的心电数据进行编码。函数中第二个形参 uint16_t heart_rat 是作为传感器传递的心率参数。第三个形参 uint8_t * p_encoded_buffer 作为编码后转换值存放的数据结构体。后面上传到主机的数据也是这个结构体内的数据。同时该函数还对 rr_interval 值进行了编码。

那么这里编码后的心率值以什么机制发送给手机了？这是个关键性的问题，这个可以认真回忆之前的按键通知这个实验，按键通知是单向的 notify 功能，不需要返回参数。蓝牙心电和这个实验属性很相似，只需要通知，不需要返回值。区别心率服务需要周期性的通知采样的信号量。如何周期性发送，大家很容易想到了定时器。

下面首先来看看定时器这个函数，心率测试值发送函数，我们要决定什么时候发送这个值，代码中红色标记了测量心率值使用的定时器，这个定时器函数中，当定时时间超时溢出后，会执行 heart_rate_meas_timeout_handler 超时处理函数，这个就是心率测量超时处理函数，因此可以循环的进行定时采集：

```
static void timers_init(void)
{
    ret_code_t err_code;

    // Initialize timer module.
    err_code = app_timer_init();
    APP_ERROR_CHECK(err_code);

    // Create timers.
    err_code = app_timer_create(&m_battery_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                battery_level_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);

    err_code = app_timer_create(&m_heart_rate_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                heart_rate_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);

    err_code = app_timer_create(&m_rr_interval_timer_id,
```

```
                                          APP_TIMER_MODE_REPEATED,
                                          rr_interval_timeout_handler);
    APP_ERROR_CHECK(err_code);

    err_code = app_timer_create(&m_sensor_contact_timer_id,
                                          APP_TIMER_MODE_REPEATED,
                                sensor_contact_detected_timeout_handler);
    APP_ERROR_CHECK(err_code);
}
```

定时器超时函数如下所示，定时时间到了后，函数中主要就做两个工作，一个是启动传感器进行心电数据采集，第二个就是把测量的数据上传主机，如下代码所示：

```
static void heart_rate_meas_timeout_handler(void * p_context)
{
    static uint32_t cnt = 0;
    ret_code_t      err_code;
    uint16_t        heart_rate;

    UNUSED_PARAMETER(p_context);
    //传感器的数据测量
    heart_rate =
      (uint16_t)sensorsim_measure(&m_heart_rate_sim_state, &m_heart_rate_sim_cfg);
    //心率数据的上传
    cnt++;
    err_code = ble_hrs_heart_rate_measurement_send(&m_hrs, heart_rate);
    if ((err_code != NRF_SUCCESS) &&
        (err_code != NRF_ERROR_INVALID_STATE) &&
        (err_code != NRF_ERROR_RESOURCES) &&
        (err_code != NRF_ERROR_BUSY) &&
        (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
       )
    {
        APP_ERROR_HANDLER(err_code);
    }

    // Disable RR Interval recording every third heart rate measurement.
    // NOTE: An application will normally not do this. It is done here just for testing generation
    // of messages without RR Interval measurements.
    m_rr_interval_enabled = ((cnt % 3) != 0);
}
```

首先来看看数据上传函数，该函数和按键通知函数类似，采用 sd_ble_gatts_hvx 协议栈函数 GATT 传输 API 进行数据上传。我们关注上传数据的几个参数：

1：上传的数据是传感器采集的数据还是前面 hrm_encode 把传感器数据编码后的数据？

2：上传数据的长度？

阅读代码如下所示：

```
1.  uint32_t ble_hrs_heart_rate_measurement_send(ble_hrs_t * p_hrs, uint16_t heart_rate)
2.  {
3.      uint32_t err_code;
4.
5.      // Send value if connected and notifying
6.      if (p_hrs->conn_handle != BLE_CONN_HANDLE_INVALID)
7.      {
8.          uint8_t                  encoded_hrm[MAX_HRM_LEN];
9.          uint16_t                 len;
10.         uint16_t                 hvx_len;
11.         ble_gatts_hvx_params_t hvx_params;
12.
13.         len        = hrm_encode(p_hrs, heart_rate, encoded_hrm)
14.         hvx_len = len;;//上传数据长度
15.
16.         memset(&hvx_params, 0, sizeof(hvx_params));
17.
18.         hvx_params.handle = p_hrs->hrm_handles.value_handle;
19.         hvx_params.type     = BLE_GATT_HVX_NOTIFICATION;
20.         hvx_params.offset = 0;
21.         hvx_params.p_len    = &hvx_len;//上传的数据长度
22.         hvx_params.p_data = encoded_hrm;//上传的数据
23.
24.         err_code = sd_ble_gatts_hvx(p_hrs->conn_handle, &hvx_params);
25.         if ((err_code == NRF_SUCCESS) && (hvx_len != len))
26.         {
27.             err_code = NRF_ERROR_DATA_SIZE;
28.         }
29.     }
30.     else
31.     {
32.         err_code = NRF_ERROR_INVALID_STATE;
33.     }
34.
35.     return err_code;
36. }
```

首先代码第 22 行，对上传数据 hvx_params.p_data = encoded_hrm 进行复制，也就是说上传的数据为 encoded_hrm，这个参数是第 13 行函数 hrm_encode(p_hrs, heart_rate, encoded_hrm)的第三个形参，也就是前面讲的心率测量值编码后的数据。因此，我们上传编码后 8 位心电数据给主机。

第 21 行，hvx_len 定义数据上传长度，也就是 hrm_encode(p_hrs, heart_rate, encoded_hrm)编码函数返回的值。

最后来关心下定时器超时函数中的心电测量参数是怎么来的，在定时器超时函数中采用下面这个函数进行传感器数据采集：

```
1.    heart_rate =
2.        (uint16_t)sensorsim_measure(&m_heart_rate_sim_state, &m_heart_rate_sim_cfg);
```

进入这个测量函数内部，我们发现官方给出的 SDK 实例并没有接传感器，只是通

过数据进行过模拟，代码如下所示，当数据大于最小值，低于最大值的时候，自增加。达到最大值后，自减少。循环进行。主函数中定义了模拟的参数范围：

```
1.  static void sensor_simulator_init(void)
2.  {
3.      m_battery_sim_cfg.min          = MIN_BATTERY_LEVEL;
4.      m_battery_sim_cfg.max          = MAX_BATTERY_LEVEL;
5.      m_battery_sim_cfg.incr         = BATTERY_LEVEL_INCREMENT;
6.      m_battery_sim_cfg.start_at_max = true;
7.
8.      sensorsim_init(&m_battery_sim_state, &m_battery_sim_cfg);
9.
10.     m_heart_rate_sim_cfg.min          = MIN_HEART_RATE;
11.     m_heart_rate_sim_cfg.max          = MAX_HEART_RATE;
12.     m_heart_rate_sim_cfg.incr         = HEART_RATE_INCREMENT;
13.     m_heart_rate_sim_cfg.start_at_max = false;
14.
15.     sensorsim_init(&m_heart_rate_sim_state, &m_heart_rate_sim_cfg);
16.
17.     m_rr_interval_sim_cfg.min          = MIN_RR_INTERVAL;
18.     m_rr_interval_sim_cfg.max          = MAX_RR_INTERVAL;
19.     m_rr_interval_sim_cfg.incr         = RR_INTERVAL_INCREMENT;
20.     m_rr_interval_sim_cfg.start_at_max = false;
21.
22.     sensorsim_init(&m_rr_interval_sim_state, &m_rr_interval_sim_cfg);
23. }
```

实际上这种数据仅仅是进行模拟，没有任何意义，你如果加入心电传感器，数据的进入就是这个函数了。

```
1.  uint32_t sensorsim_measure(sensorsim_state_t        * p_state,
2.                             const sensorsim_cfg_t * p_cfg)
3.  {
4.      if (p_state->is_increasing)
5.      {
6.          sensorsim_increment(p_state, p_cfg);
7.      }
8.      else
9.      {
10.         sensorsim_decrement(p_state, p_cfg);
11.     }
12.     return p_state->current_val;
13. }
```

讲到这里，整个过程我们就清楚了，相当于通过软件定时器，来通过定时实现周期性采样信号后发送心率参数值给主机。可以对比之前的按键通知应用，就是在类似的通知状态下加入定时器周期发送。例子里是模拟数据，可以实际进入心电传感器进行实际的数据采集。

心电服务中，除了心跳次数数据外，还有一个 RR—interval 的心跳尖峰值参数值需要采集，方法和心跳次数数据类似，也单独分配了一个软件定时器来处理，这里就不再重复。

20.2.2 电池服务设计和设备信息服务设计

电池服务设计参考前面一讲内容，这里就不再重复。这节主要谈下设备信息服务：设备信息服务就比较简单了。因为其与手机的交互是单方向的，就是手机读取板子上的特征值。所以创建完服务添加了特征值后后续就不需要操作了。

服务初始化中主要就是添加了几个可读的特征值。如设备厂商名字，硬件 ID 版本，软件版本等。代码如下：

```
1.      uint32_t ble_dis_init(ble_dis_init_t const * p_dis_init)
2.  {
3.      uint32_t     err_code;
4.      ble_uuid_t ble_uuid;
5.
6.      // Add service
7.      BLE_UUID_BLE_ASSIGN(ble_uuid, BLE_UUID_DEVICE_INFORMATION_SERVICE);
8.     //服务 UUID 的添加
9.      err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY, &ble_uuid,
    &service_handle);
10.     if (err_code != NRF_SUCCESS)
11.     {
12.         return err_code;
13.     }
14.
15.     // 添加特征值
16.     if (p_dis_init->manufact_name_str.length > 0)
17.     {
18.         err_code = char_add(BLE_UUID_MANUFACTURER_NAME_STRING_CHAR,
19.                             p_dis_init->manufact_name_str.p_str,
20.                             p_dis_init->manufact_name_str.length,
21.                             &p_dis_init->dis_attr_md,
22.                             &manufact_name_handles);//设备名称
23.         if (err_code != NRF_SUCCESS)
24.         {
25.             return err_code;
26.         }
27.     }
28.     if (p_dis_init->model_num_str.length > 0)
29.     {
30.         err_code = char_add(BLE_UUID_MODEL_NUMBER_STRING_CHAR,
31.                             p_dis_init->model_num_str.p_str,
32.                             p_dis_init->model_num_str.length,
33.                             &p_dis_init->dis_attr_md,
34.                             &model_num_handles);//型号
35.         if (err_code != NRF_SUCCESS)
36.         {
37.             return err_code;
38.         }
39.     }
40.     if (p_dis_init->serial_num_str.length > 0)
41.     {
```

```
        err_code = char_add(BLE_UUID_SERIAL_NUMBER_STRING_CHAR,
                            p_dis_init->serial_num_str.p_str,
                            p_dis_init->serial_num_str.length,
                            &p_dis_init->dis_attr_md,
                            &serial_num_handles);//序列号
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }
    }
    if (p_dis_init->hw_rev_str.length > 0)
    {
        err_code = char_add(BLE_UUID_HARDWARE_REVISION_STRING_CHAR,
                            p_dis_init->hw_rev_str.p_str,
                            p_dis_init->hw_rev_str.length,
                            &p_dis_init->dis_attr_md,
                            &hw_rev_handles);//硬件版本号
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }
    }
    if (p_dis_init->fw_rev_str.length > 0)
    {
        err_code = char_add(BLE_UUID_FIRMWARE_REVISION_STRING_CHAR,
                            p_dis_init->fw_rev_str.p_str,
                            p_dis_init->fw_rev_str.length,
                            &p_dis_init->dis_attr_md,
                            &fw_rev_handles);//固件号
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }
    }
    if (p_dis_init->sw_rev_str.length > 0)
    {
        err_code = char_add(BLE_UUID_SOFTWARE_REVISION_STRING_CHAR,
                            p_dis_init->sw_rev_str.p_str,
                            p_dis_init->sw_rev_str.length,
                            &p_dis_init->dis_attr_md,
                            &sw_rev_handles);//软件版本号
        if (err_code != NRF_SUCCESS)
        {
            return err_code;
        }
    }
    if (p_dis_init->p_sys_id != NULL)
    {
        uint8_t encoded_sys_id[BLE_DIS_SYS_ID_LEN];

        sys_id_encode(encoded_sys_id, p_dis_init->p_sys_id);
```

```
93.             err_code = char_add(BLE_UUID_SYSTEM_ID_CHAR,
94.                                 encoded_sys_id,
95.                                 BLE_DIS_SYS_ID_LEN,
96.                                 &p_dis_init->dis_attr_md,
97.                                 &sys_id_handles);//系统的 ID
98.             if (err_code != NRF_SUCCESS)
99.             {
100.                return err_code;
101.            }
102.    }
103.    if (p_dis_init->p_reg_cert_data_list != NULL)
104.    {
105.        err_code =
    char_add(BLE_UUID_IEEE_REGULATORY_CERTIFICATION_DATA_LIST_CHAR,
106.                                p_dis_init->p_reg_cert_data_list->p_list,
107.                                p_dis_init->p_reg_cert_data_list->list_len,
108.                                &p_dis_init->dis_attr_md,
109.                                &reg_cert_data_list_handles);//IEEE 的认证
110.            if (err_code != NRF_SUCCESS)
111.            {
112.                return err_code;
113.            }
114.    }
115.    if (p_dis_init->p_pnp_id != NULL)
116.    {
117.        uint8_t encoded_pnp_id[BLE_DIS_PNP_ID_LEN];
118.
119.        pnp_id_encode(encoded_pnp_id, p_dis_init->p_pnp_id);
120.        err_code = char_add(BLE_UUID_PNP_ID_CHAR,
121.                                encoded_pnp_id,
122.                                BLE_DIS_PNP_ID_LEN,
123.                                &p_dis_init->dis_attr_md,
124.                                &pnp_id_handles);//PNP 的 ID 号
125.            if (err_code != NRF_SUCCESS)
126.            {
127.                return err_code;
128.            }
129.    }
130.
131.    return NRF_SUCCESS;
132.}
```

这些数据相当于是确定给定的数据，都是一些供手机查看的信息数据，所以特征属性为只读属性。设置完成后，与手机端也只需要一个读取的交互，并且不需要板子这端的做处理。

```
133.typedef struct
134.{
135.    ble_srv_utf8_str_t                  manufact_name_str;
136.  /**< Manufacturer Name String. */
137.    ble_srv_utf8_str_t                  model_num_str;
138./**< Model Number String. */
```

```
139.      ble_srv_utf8_str_t                  serial_num_str;
140./**< Serial Number String. */
141.      ble_srv_utf8_str_t                  hw_rev_str;
142./**< Hardware Revision String. */
143.      ble_srv_utf8_str_t                  fw_rev_str;
144.  /**< Firmware Revision String. */
145.      ble_srv_utf8_str_t                  sw_rev_str;
146./**< Software Revision String. */
147.      ble_dis_sys_id_t *                  p_sys_id;
148.  /**< System ID. */
149.      ble_dis_reg_cert_data_list_t * p_reg_cert_data_list;
150./**< IEEE 11073-20601 Regulatory Certification Data List. */
151.      ble_dis_pnp_id_t *                   p_pnp_id;
152.  /**< PnP ID. */
153.      ble_srv_security_mode_t            dis_attr_md;
154.  /**< Initial Security Setting for Device Information Characteristics. */
155.} ble_dis_init_t;
```

设备参数专门提供一个结构体 ble_dis_init_t 进行定义，定义了所以的设备可以添加的参数，包含设备名称、型号、序列号、硬件版本号、固件版本号、软件版本号、系统 ID 号、IEEE 认证、PNP 的 ID 号，这些参数通过 char_add 函数添加特征值的时候全部为只读属性，所以方便用一个函数批量添加。

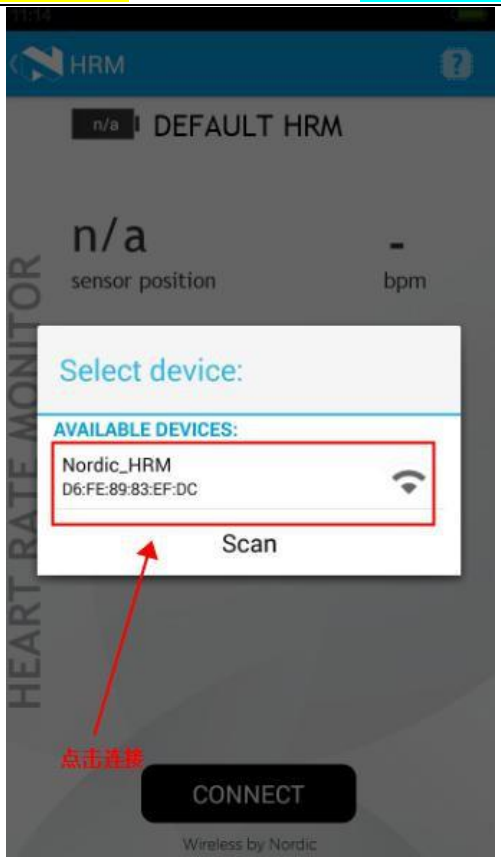
在主函数中的服务初始化中，这些参数只定义了一个公司设备名称号。如下所示：

```
156.  ble_srv_ascii_to_utf8(&dis_init.manufact_name_str, (char *)MANUFACTURER_NAME);
```

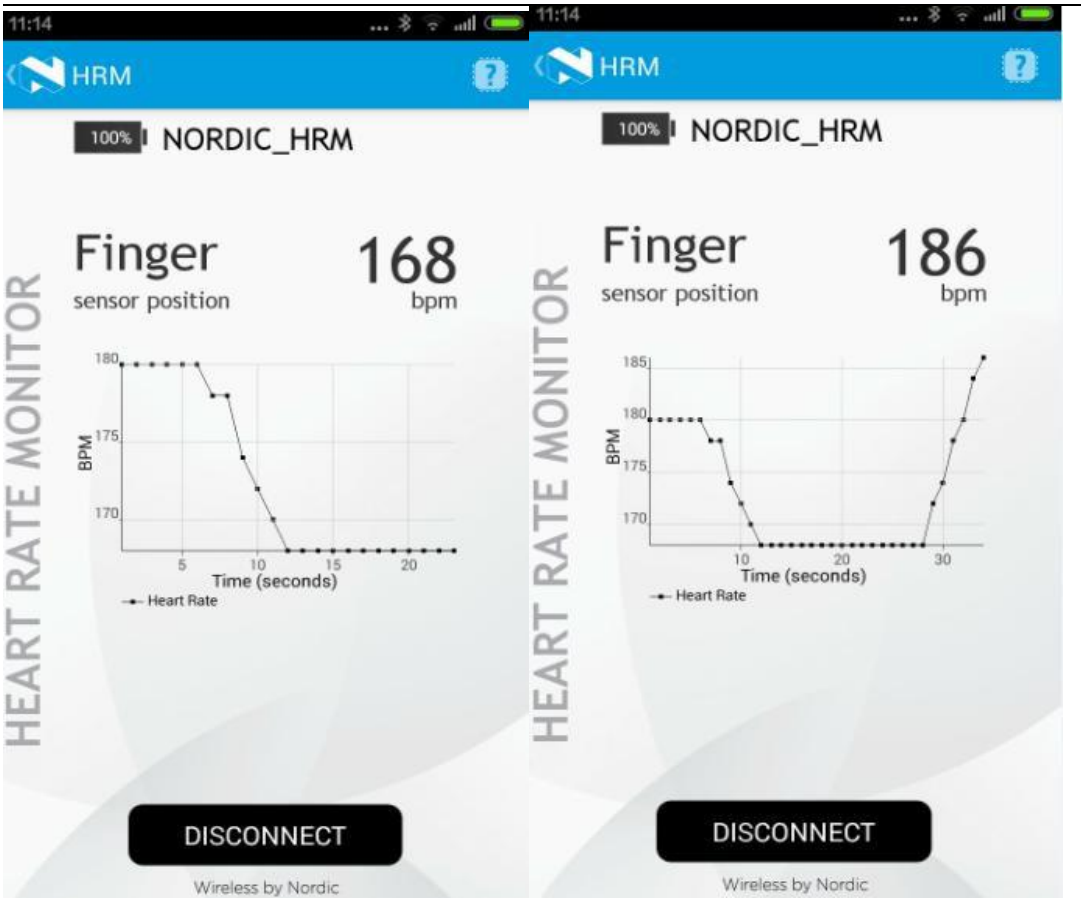
当完成 3 个服务的搭建后。主函数中添加服务声明，整个心电工程就建立成功了。

20.3 下载验证

首先下载协议栈，然后下载应用程序，下载过程请参考之前的章节。下载完成应用代码后，程序开始运行程序，广播 LED 灯开始闪烁。然后打开 nrf tool app 中的 HRM， 如下图所示，点击扫描 SCAN，发现广播后点击连接：



连接成功如下图所示，心电 BPM 的值开始按照模拟的方式变化：



然后断开