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

2.29 蓝牙 FLASH 存储之 fstorage

很多朋友和客户希望能够实现内部的 FLASH 的存储，内部 FLASH 的存储官方提供两种方式，一种就是这讲将要探讨的 fstorage 方式，另外一种就是在第一次方式基础上的 fds 方式。

这里我们通过一个简单的例子：蓝牙 BLE 的 FLASH 存储，来进行一个简单的思路验证。注意本例在蓝牙串口的基础上进行修改。

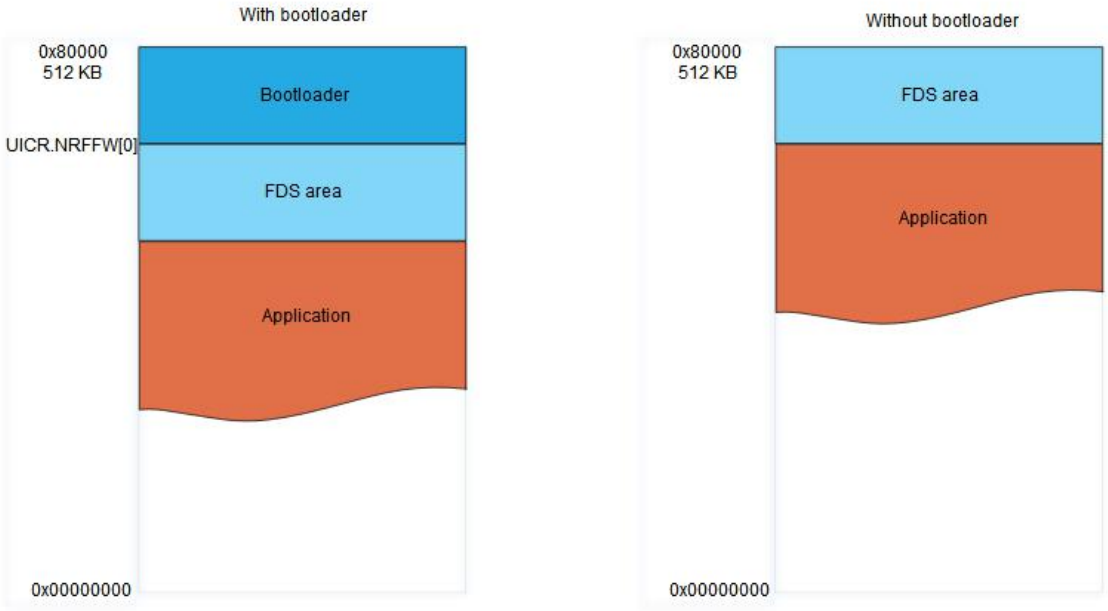
1 蓝牙 FLASH 存储介绍

fstorage 是一个用于读取、写入和擦除持久闪存中的数据的模块。该模块定义了一个异步接口来访问闪存，并使用读、写和（page）擦除操作。通过对注册事件处理程序的回调，通知应用程序的操作结果。

Fstorage 方式这是一个低级的库，旨在为闪存提供一个简单的、原始的接口。如果您需要一个具有更新/搜索功能的更高级别的 API 来存储记录和文件，这些记录和文件可以为您存储数据，如何读取数据，如何管理更新等等，可以看第二种方式：fds 数据存储方式。

任何 flash 存储器的用户都可以选择一个可用的后端，并指定允许其操作的闪存区域，以及用于报告所请求操作的状态的回调函数。因此，本讲将以这个思路定义了一个 fstorage 实例。注意： fstorage 方式不保证不同的实例在非重叠的闪存区域上运行，因此需要用户自己来确保他们对该区域的使用不会影响其他功能，也就是说自己保证不发送内容叠加。

下面是图片提供了关于在 flash 中存储数据的 SDK 模块使用了芯片上的 flash 区域。



这个后端接口与 SoftDevice flash API 在 flash 中执行操作。无论软设备是否启用，它都可以在任何时候使用软设备。然而，当运行一个协议栈时，flash 操作的成功与协议的时间限制和硬件的特性联系在一起。使用过于激进的扫描或广告间隔，或运行多个连接，会影响 flash 操作的成功。在大多数情况下，nrf_storage_sd 将在没有问题的情况下与无线电协议同时处理 flash 操作。如果可能出现超时错误，请尝试减少正在写入的数据的大小（在一个 nrf_ storage_write 调用中）。否则，尝试修改 nrf_ storage_sd_max_write_size 和增量 nrf_storage_sd_max 重试。如果问题仍然存在，请参考软设备规范文档来确定更合适的扫描和广播间隔。

2 程序 API 使用与编写

本来我们通过编写一个 flash_test 对设计思路的验证。采用 fstorage 方式测试 flash 的思路比较简单，就是向两个指定的地址写入两个数据，然后读取两个地址的数据，对比写入和读出的数据是否正确。具体代码如下所示，下面来进行详细分析：

```
/*设置写入数据. */
static uint32_t m_data                 = 0xBADC0FFE;
static uint32_t m_data2;

void flash_test(void)
{
    ret_code_t rc;
    NRF_LOG_INFO("fstorage example started.\r\n");//开始演示

    nrf_fstorage_api_t * p_fs_api;
```

```
08.         p_fs_api = &nrf_fstorage_sd;//处理操作类型设置
09.
10.         rc = nrf_fstorage_init(&fstorage, p_fs_api, NULL);//flash 处理的开始地址和结束地址初始化
11.        APP_ERROR_CHECK(rc);
12.        (void) nrf5_flash_end_addr_get();//获取地址，判断为可写地址大小
13.
14.        //向 0x3e000 写入数据 m_data 数据
15.        rc = nrf_fstorage_write(&fstorage, 0x3e000, &m_data, sizeof(m_data), NULL);
16.        APP_ERROR_CHECK(rc);
17.
18.        wait_for_flash_ready(&fstorage);//等待写完
19.         NRF_LOG_INFO("Writing \"%x\" to flash.\r\n", m_data);//打印第一次写入的值
20.        NRF_LOG_INFO("Done.\r\n");
21.
22.         m_data = 0xDEADBEEF;//重新向 m_data 赋值
23.        //再把数据写到 0x3e100 地址去
24.        rc = nrf_fstorage_write(&fstorage, 0x3e100, &m_data, sizeof(m_data), NULL);
25.        APP_ERROR_CHECK(rc);
26.
27.        wait_for_flash_ready(&fstorage);//等待完成
28.        NRF_LOG_INFO("Writing \"%x\" to flash.\r\n", m_data);//打印第二次写入的值
29.         NRF_LOG_INFO("Done.\r\n");
30.
31.        nrf_fstorage_read(&fstorage, 0x3e000,
32.                                              &m_data2,
33.                                             sizeof(m_data2));//从 0x3e000 地址读取数据
34.         printf("Read \"%x\" to flash.\r\n", m_data2);//打印读取的值
35.
36.         nrf_fstorage_read(&fstorage, 0x3e100,
37.                                              &m_data2,
38.                                             sizeof(m_data2));//从 0x3e100 地址读取数据
39.         printf("Read \"%x\" to flash.\r\n", m_data2); //打印读取的值
40. }
```

代码分析如下：

◎第 8 行：首先需要使用结构体 &nrf_fstorage_sd 定义了 FLASH 的操作处理类型，用在后面初始化 FLASH 函数的第二个形参中进行调用，具体定义如下所示：

```
41. nrf_fstorage_api_t nrf_fstorage_sd =
42. {
43.         .init       = init,
44.         .uninit     = uninit,
45.         .read        = read,
46.         .write       = write,
```

```
47.     .erase    = erase,
48.     .rmap     = rmap,
49.     .wmap     = wmap,
50.     .is_busy = is_busy
51. };
```

◎ 第 10 行：初始化 FLASH 区域，来实现 flash 处理的开始地址和结束地址初始化。使用官方 API 函数 nrf_fstorage_init 来实现需要处理的 FLASH 区域的初始化。该 API 函数解释如下：

```
* 参数    p_fs          初始化 fstorage 实例区域
* 参数    p_api         使用的操作 API.
* 参数    p_param       可选的参数,调用于实现特定操作的 API
 * 返回值  NRF_SUCCESS          如果初始化成功，则返回成功.
 * 返回值  NRF_ERROR_NULL       参数 p_fs 或者 p_api 在参数 p_fs 区域中为空
 * 返回值   NRF_ERROR_INTERNAL  发生其他错误.


ret_code_t nrf_fstorage_init(nrf_fstorage_t       * p_fs,
                             nrf_fstorage_api_t * p_api,
                             void               * p_param);
```

本函数的第一个参数是分配给 fstorage 实例的闪存 FLASH 空间的边界。您必须手动设置这些，每次在运行前，首先调用 nrf_fstorage_init（）函数。

```
52.  NRF_FSTORAGE_DEF(nrf_fstorage_t fstorage) =
53. {
54.     /* 设置 fstorage 处理事件 */
55.     .evt_handler = fstorage_evt_handler,
56.     .start_addr = 0x3e000,//设置开始地址
57.     .end_addr   = 0x3ffff,//设置结束地址
58. };
```

然后需要使用函数 nrf5_flash_end_addr_get()用来检索 FLASH 区域上可用于编写数据的在最后一页最后一个地址。因此在初始化 FLASH 操作区域后，第 12 行，需要马上调用 nrf5_flash_end_addr_get()函数，代码如下所示：

```
59. static uint32_t nrf5_flash_end_addr_get()
60. {
61.     uint32_t const bootloader_addr = NRF_UICR->NRFFW[0];
62.     uint32_t const page_sz         = NRF_FICR->CODEPAGESIZE;
```

```
63.     uint32_t const code_sz              = NRF_FICR->CODESIZE;
64.
65.     return (bootloader_addr != 0xFFFFFFFF ?
66.             bootloader_addr : (code_sz * page_sz));
67. }
```

◎第 15 行：开始向指定的地址内写入数据，首先定义一个指针赋值，然后写入对应地址内，使用函数 nrf_fstorage_write，对应这个 API 函数，详细进行介绍如下所示：

```
/*功能描述： 功能是向 FLASH 内写数据.从 p_src 中写一个长度为 len 的 bytes 数据到 dest 的地址内。
 *当数据长度超过 NRF_FSTORAGE_SD_MAX_WRITE_SIZE 定义的最大字节数的时候，数据需要通过 sd_flash_write 函数多次写入。完成后只发送一个事件。
 *
 * 注意： 要写入 flash 的数据必须保存在内存中，直到操作终止，并接收到事件为止。
 *
 * 参数    p_fs          定义的 FLASH 区域块.
 * 参数    dest          写入数据的 flash 地址.
 * 参数    p_src         需要写入的数据指针.
 * 参数    len           数据长度 (以 bytes 为单位).
 * 参数    p_param       传递给事件处理程序的已定义参数(可能为 NULL).
 *
 * 返回值  NRF_SUCCESS                   如果操作被接受.
 * 返回值  NRF_ERROR_NULL                如果 p_fs 或者 p_src 是空的.
 * 返回值  NRF_ERROR_INVALID_STATE       如果 flash 模块没有被初始化
 * 返回值  NRF_ERROR_INVALID_LENGTH      如果参数 len 为零或者不是程序单位的倍数，或者是无效的
 * 返回值  NRF_ERROR_INVALID_ADDR        如果 flash 写入地址参数 dest 超出了参数 p_fs 定义的 flash 存储空间范围，或者不是字对齐的。
 * 返回值  NRF_ERROR_NO_MEM              如果没有存储空间接受在结构体 nrf_fstorage_sd 中定义的操作，这个错误表明操作的内部队列是满的
 */
```

代码写入如下所示,向 0x3e000 写入&m_data 中存的数据：

```
ret_code_t nrf_fstorage_write(nrf_fstorage_t const * p_fs,
                              uint32_t               dest,
                              void           const * p_src,
                              uint32_t               len,
                              void                 * p_param);
```

```
68.     rc = nrf_fstorage_write(&fstorage, 0x3e000, &m_data, sizeof(m_data), NULL);
69.         APP_ERROR_CHECK(rc);
```

◎第 18 行：FLASH 的写入速度比较慢，因此，每次写入的时候，必须等待写入完成，调用 nrf_fstorage_is_busy(p_fstorage)判断是否空闲，如果还在写入，蓝牙设备使用 sd_app_evt_wait 待机，不进行蓝牙通信，代码如下所示：

```
70. void wait_for_flash_ready(nrf_fstorage_t const * p_fstorage)
71. {
72.         /* While fstorage is busy, sleep and wait for an event. */
73.         while (nrf_fstorage_is_busy(p_fstorage))
74.         {
75.             sd_app_evt_wait();
76.         }
77. }
```

◎第 32 行--第 36 行：写入数据完成后，需要从指定地址读取数据，把数据进行对比，因此需要使用 API 读取函数 nrf_fstorage_read 进行读取，读取代码如下：

```
78.     nrf_fstorage_read(&fstorage, 0x3e000,
79.                                          &m_data2,
80.                                         sizeof(m_data2));//从 0x3e000 地址读取数据
81.         printf("Read \"%x\" to flash.\r\n", m_data2);//打印读取的值
82.
83.         nrf_fstorage_read(&fstorage, 0x3e100,
84.                                          &m_data2,
85.                                         sizeof(m_data2));//从 0x3e100 地址读取数据
86.         printf("Read \"%x\" to flash.\r\n", m_data2); //打印读取的值
```

对该读取函数函数 API，详细解释如下：

```
/**功能    读取 FLASH 对应地址的内容.复制 len 长度的数据 bytes 从参数地址 addr 到参数指针 p_dest.
 *
 * 参数    p_fs      The fstorage instance.
 * 参数    addr      Address in flash where to read from.
 * 参数    p_dest    Buffer where the data should be copied.
 * 参数    len       Length of the data to be copied (in bytes).
 *
 * 返回值   NRF_SUCCESS                  如果操作成功了.
 * 返回值   NRF_ERROR_NULL               如果参数 p_fs 或者 p_dest 为空
 * 返回值   NRF_ERROR_INVALID_STATE      如果 flash 模块没有被初始化
 * 返回值   NRF_ERROR_INVALID_LENGTH     如果参数 len 为零或者不是程序单位的倍数，或者是无效的
 * 返回值   NRF_ERROR_INVALID_ADDR       如果 flash 写入地址参数 dest 超出了参数 p_fs 定义的 flash 存储空间范围，或者不是字对齐的。
 */
ret_code_t nrf_fstorage_read(nrf_fstorage_t const * p_fs,
                             uint32_t             addr,
                             void               * p_dest,
                             uint32_t             len);
```

那么主函数就比较简单了，直接调用 flash_test()函数进行测试，具体代码如下：

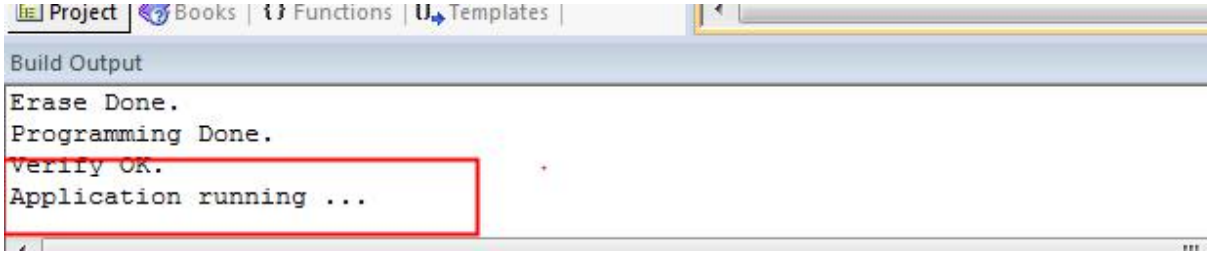
```
04. int main(void)
05. {
06.     bool erase_bonds;
07.
08.     // Initialize.
09.     uart_init();
10.     log_init();
11.     timers_init();
12.     buttons_leds_init(&erase_bonds);
13.      power_management_init();
14.     ble_stack_init();
15.     gap_params_init();
```

```
16.     gatt_init();
17.     services_init();
18.     advertising_init();
19.     conn_params_init();
20.     // Start execution.
21.      flash_test();
22.     printf("\r\nUART started.\r\n");
23.     NRF_LOG_INFO("Debug logging for UART over RTT started.");
24.     advertising_start();
25.
26.
27.     // Enter main loop.
28.     for (;;)
29.     {
30.         idle_state_handle();
31.     }
32. }
```

3 应用与调试

3.1 下载

本例使用的协议栈为 S132 版本， 打开 NRFgo 进行下载，可以参考前面的章节。首先整片擦除，后下载协议栈，下载完协议栈后可以下载工程，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：



3.2 测试

打开 jlink-RTT 观察窗口，可以观察写和读的结果对比，如下图所示，如图两次写入和读的内容相同，则表明写入成功：

```
0> <info> app: fstorage example started.
0>
0> <info> app: --> Event received: wrote 4 bytes at address 0x3E000.
0> <info> app: Writing "BADCOFFE" to flash.
0>
0> <info> app: Done.
0>
0> <info> app: --> Event received: wrote 4 bytes at address 0x3E100.
0> <info> app: Writing "DEADBEEF" to flash.
0>
0> <info> app: Done.
0>
0> <info> app: Read "BADCOFFE" to flash.
0>
0> <info> app: Read "DEADBEEF" to flash.
0>
0> <info> app: Debug logging for UART over RTT started.
```