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

2.31 蓝牙 MAC 地址

本节我们讲主要探讨一下蓝牙 BLE，BLE 设备有多种类型的设备地址，如 Public Device Address、Random Device Address、Static Device Address、Private Device Address 等等。如果不了解内情，大家肯定会被它们绕晕。不过存在即合理，这样看似奇怪的设计，实际上反映了 BLE 的设计思路以及所针对的应用场景。下面让我们来详细讨论一下。

这里我们通过一个简单的例子：蓝牙 BLE 蓝牙串口，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。

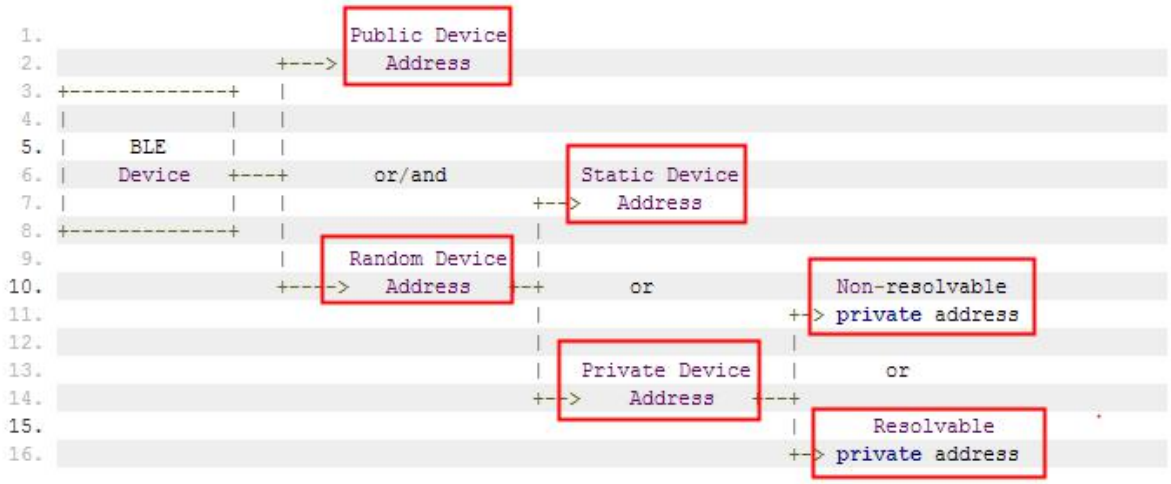
1：BLE 设备地址类型：

一个 BLE 设备，可以使用两种类型的地址（一个 BLE 设备可同时具备两种地址）：Public Device Address（公共设备地址）和 Random Device Address（随机设备地址）。而 Random Device Address 又分为 Static Device Address（静态设备地址）和 Private Device Address（私密设备地址）两类。其中 Private Device Address 又可以分为 Non-resolvable Private Address（不可解析私密地址）和 Resolvable Private Address（可解析私密地址）。它们的关系如下所示，下面就分别来详细介绍一下：

1.1 公共设备地址 Public Device Address

在通信系统中，设备地址是用来唯一识别一个物理设备的，如 TCP/IP 网络中的 MAC 地址、传统蓝牙中的蓝牙地址等。对设备地址而言，一个重要的特性，就是唯一性（或者说一定范围内的唯一），否则很有可能造成很多问题。蓝牙通信系统也不例外。

对经典蓝牙（BR/EDR）来说，其设备地址是一个 48bits 的数字，称作"48-bit universal LAN MAC addresses(和电脑的 MAC 地址一样)"。正常情况下，该地址需



要向 IEEE 申请（其实是购买，呵呵！）。企业交钱，IEEE 保证地址的唯一性，皆大欢喜。

当然，这种地址分配方式，在 BLE 中也保留下来了，就是 Public Device Address。Public Device Address 由 24-bit 的 company_id 和 24-bit 的 company_assigned 组成，如下图所示：

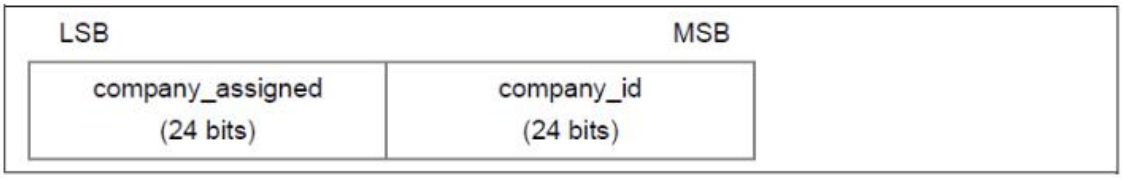


Figure 1.2: Format of public device address

高 24 位是公司标示，低 24 位公司内部自己赋值。

1.2 随机设备地址 Random Device Address

但是，在 BLE 时代，只有 Public Device Address 明显不够用了，有如下原因：

1）Public Device Address 需要向 IEEE 购买。虽然不贵，但在 BLE 时代，相比 BLE IC 的成本，还是不小的一笔开销。

2）Public Device Address 的申请与管理是相当繁琐、复杂的一件事情，再加上 BLE 设备的数量众多（和传统蓝牙设备不是一个数量级的），导致维护成本增大。

3）安全因素。BLE 很大一部分的应用场景是广播通信，这意味着只要知道设备的地址，就可以获取所有的信息，这是不安全的。因此固定的设备地址，加大了信息泄漏的风险。

为了解决上述问题，BLE 协议新增了一种地址：Random Device Address，即设备地址不是固定分配的，而是在设备设备启动后随机生成的。根据不同的目的，Random Device Address 分为 Static Device Address 和 Private Device Address 两类。

1.2.1 静态设备地址 Static Device Address

Static Device Address 是设备在上电时随机生成的地址，nrf52832 官方工程默认都是使用静态地址，其格式如下：

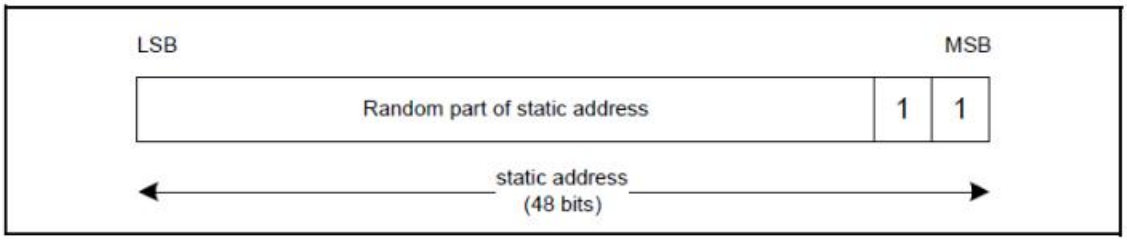


Figure 1.3: Format of static address

Static Device Address 静态设备地址的特征可总结为：

1）最高两个 bit 为“11”。

2）剩余的 46bits 是一个随机数，不能全部为 0，也不能全部为 1。

3）在一个上电周期内保持不变。

4）下一次上电的时候可以改变。但不是强制的，因此也可以保持不变。如果改变，上次保存的连接等信息，将不再有效。

Static Device Address 静态设备地址的使用场景可总结为：

1）46bits 的随机数，可以很好地解决“设备地址唯一性”的问题，因为两个地址相同的概率很小。

2）地址随机生成，可以解决 Public Device Address 申请所带来的费用和维护问题。

1.2.2 私密设备地址 Private Device Address

Static Device Address 通过地址随机生成的方式，解决了部分问题，Private Device Address 则更进一步，通过定时更新和地址加密两种方法，提高蓝牙地址的可靠性和安全性。根据地址是否加密，Private Device Address 又分为两类:

Non-resolvable private address 和 Resolvable private address。下面我们分别描述。

1.Non-resolvable private address

Non-resolvable private address 和 Static Device Address 类似，不同之处在于，Non-resolvable private address 会定时更新。更新的周期称是由 GAP 规定的，称作 T_GAP(private_addr_int) ，建议值是 15 分钟。其格式如下：

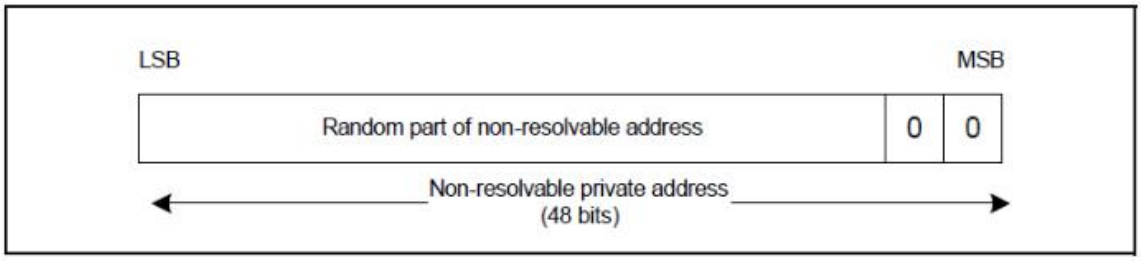


Figure 1.4: Format of non-resolvable private address

特征可总结为：

1）最高两个 bit 为“00”。

2）剩余的 46bits 是一个随机数，不能全部为 0，也不能全部为 1。

3）以 T_GAP(private_addr_int)为周期，定时更新。

注：Non-resolvable private address 有点奇怪，其应用场景并不是很清晰。地址变来变去的，确实是迷惑了敌人，但自己人不也一样被迷惑了吗？因此，实际产品中，该地址类型并不常用。

2.Resolvable private address

Resolvable private address 比较有用，它通过一个随机数和一个称作 identity resolving key (IRK) 的密码生成，因此只能被拥有相同 IPK 的设备扫描到，可以防止被未知设备扫描和追踪。其格式如下：

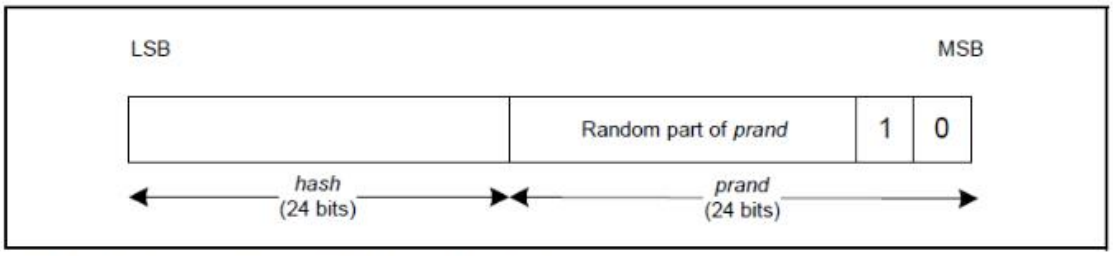


Figure 1.5: Format of resolvable private address

特征如下：

1）由两部分组成：
高位 24bits 是随机数部分，其中最高两个 bit 为“10”，用于标识地址类型；

低位 24bits 是随机数和 IRK 经过 hash 运算得到的 hash 值，运算的公式为 hash = ah(IRK, prand)。

2）当对端 BLE 设备扫描到该类型的蓝牙地址后，会使用保存在本机的 IRK，和该地址中的 prand，进行同样的 hash 运算，并将运算结果和地址中的 hash 字段比较，相同的时候，才进行后续的操作。这个过程称作 resolve（解析），这也是 Non-resolvable private address/Resolvable private address 命名的由来。如果不同则继续 用下一个 IRK 做上面的过程，知道找到一个关联的 IRK 或者一个没找到。

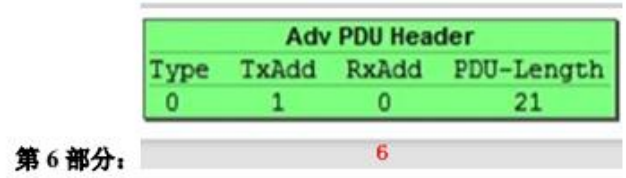
3）以 T_GAP(private_addr_int)为周期，定时更新。哪怕在广播、扫描、已连接等过程中，也可能改变。

4）Resolvable private address 不能单独使用，因此需要使用该类型的地址的话，设备要同时具备 Public Device Address 或者 Static Device Address 中的一种。

2：nrf52832 地址配置：

通过上面对地址的分析，我们基本清楚了 BLE 的地址类型，下面来看看蓝牙 nrf52832 的设备地址如何设置的。

其实在教程《青风 5.0 教程 28：蓝牙广播包与数据包分析》里有提到过蓝牙设备地址的问题，数据包第六部分：



如上图，其中的 TxAdd 表示发送方的地址类型 (0 为 public，1 为 random) RxAdd 表示接收方的地址类型。比如对于普通广播来说，只有 TxAdd 的指示是有效的，表示广播发送者的第一类型。 而对于定向广播来说，TxAdd 和 RxAdd 都是有效的，TxAdd 表示广播发送者的地址类型，RxAdd 表示广播接受者的地址类型。通过抓取 nrf52832 官方例子的广播包，我们发现其地址属于 Random Device Address（随机设备地址），那么属于随机设置地址的哪一种了？我们继续看：

如果是随机设备地址，则查看地址的最高两位，如果是 11 就是静态随机地址，地址类型确定。

如果是 00 表示为不可解析隐私地址，类型确定。

如果是 01 表示为可解析的私有地址，并执行上面说过的 ah 方法进行解析。

0XE11D2123261C 实际上就是 11，是静态随机地址。这个实际上是 nrf 官方代码默认的设置。

那么如果改变设置自己需要的地址方式？实际上芯片给了两个地址进行设备地址存储：

6.2.13 DEVICEADDR[0]

Bit number	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ID (Field ID)	A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
Value after erase	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

ID	RW	Field	Value ID	Value	Description
A	R	ADDR			Device address bit 31-0.

6.2.14 DEVICEADDR[1]

Bit number	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ID (Field ID)	- - - - - - - - - - - - - - - - A A A A A A A A A A A A A A A A
Value after erase	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

ID	RW	Field	Value ID	Value	Description
A	R	ADDR			Device address bit 47-32.

同时协议栈提供了一个封装函数进行修改：

uint32_t sd_ble_gap_address_set (uint8_t addr_cycle_mode, const ble_gap_addr_t * p_addr)

大家打开协议栈函数说明文档，找到相应解释：

参数 addr_cycle_mode 为：

1：BLE_GAP_ADDR_CYCLE_MODE_AUTO：

该模式下，设置为 Non-resolvable private address 和 Resolvable private address 类型，会忽略第二个地址参数 p_addr 中给的地址 addr。协议栈内部会自动周期性根据 p_addr 中 addr_type 指定的地址类型来生成可解析的或者不可解析的私密地址。

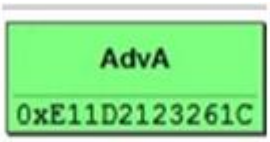
2：BLE_GAP_ADDR_CYCLE_MODE_NONE：

该模式下，可以使用 public 地址和自己设置的 static random 地址。

如果第二个参数 P_addr 中的 addr_type 为 staic random 类型，则 addr 中存放的地址的最高 2 位必须为 1，不然这个地址会被认为无效并自动替换，52832 会自动替换为蓝牙 mac 地址。跟没有调用 sd_ble_gap_address_set 函数的效果是一样的，下面就来讲讲没有调用时默认状态是什么效果：

在官方的提供的工程中，都是没有主动调用过 sd_ble_gap_address_set 函数来设置设备地址的。所以 工程中使用的都是默认静态设备地址。这个默认地址就是芯片出厂是设置的，出厂直接时烧写在设备地址寄存器中的。

所以蓝牙芯片启动后，如果你没有主动调用 sd_ble_gap_address_set 函数来设置地址的话， 协议栈就会使用寄存器 DEVICEADDR 中的值来设置 BLE 地址。但是并不是直接用 DEVICEADDR 中的值，而是把地址的最高两位必须要是设置为 1，1，所以协议栈会使用该寄存器中的地址，但是会将最高两 bit 的值都设置为 1.比如当我们抓包抓到的设备地址如下图所示：

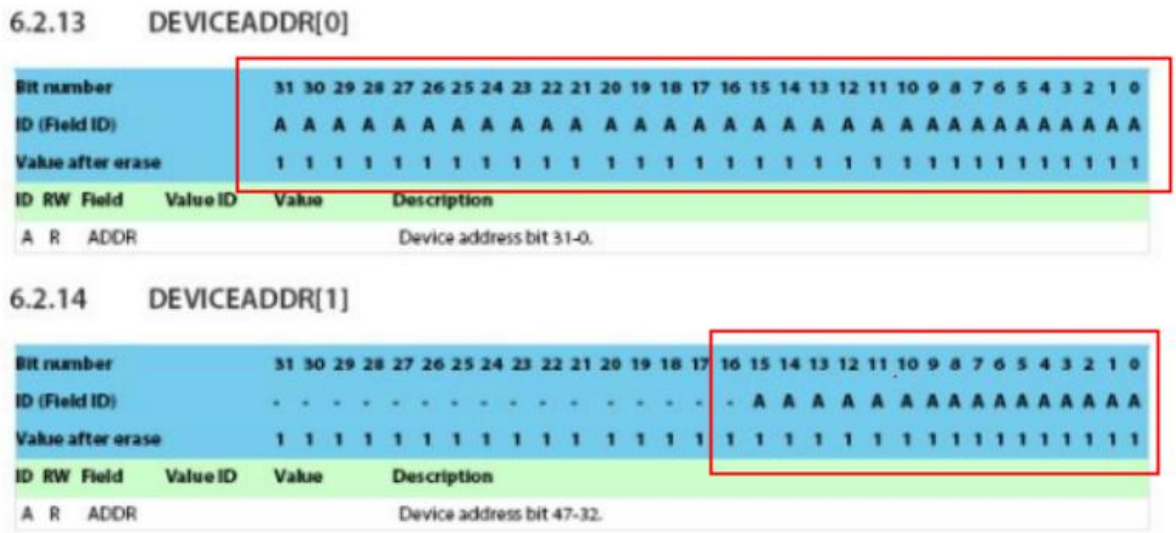


第 7 部分： 7 蓝牙设备地址

而读取寄存器 DEVICEADDR 中的值分别为：

DEVICEADDR[0] = 0x2123261C

DEVICEADDR[1] = 0x691CA11D，有用部分如下图所示：



6.2.13 DEVICEADDR[0]

Bit number	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ID (Field ID)	A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
Value after erase	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

ID	RW	Field	Value ID	Value	Description
A	R	ADDR			Device address bit 31-0.

6.2.14 DEVICEADDR[1]

Bit number	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ID (Field ID)	- - - - - - - - - - - - - - - - A A A A A A A A A A A A A A A A
Value after erase	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

ID	RW	Field	Value ID	Value	Description
A	R	ADDR			Device address bit 47-32.

厂家存放的地址静态设备地址就是：A11D2123261C,然后将最高两位都设置为 1，那么就变成了 E11D2123261C,和抓包的效果一致。

3 设置自己的 MAC 地址：

在实际应用中，使指定设备连接，如何设置自己的 MAC 地址？通过上面的分析，直接调用官方的协议栈封装函数就可以实现 MAC 地址的配置，我们下面简单的验证下如何设置：

打开蓝牙串口工程，下载协议栈后，下载程序，打开 MCP APP，找到服务，查看设置的 MAC 地址如图所示：

然后我们打开工程 BLE 实验 25：蓝牙 MAC 地址设置。这个工程在蓝牙串口基础上进行修改，加入如下代码：

```
void mac_set(void)
{
  ble_gap_addr_t addr;
 uint32_t   err_code = sd_ble_gap_address_get(&addr);
  APP_ERROR_CHECK(err_code);
  addr.addr[0] += 1;

   err_code = sd_ble_gap_addr_set(&addr);
  APP_ERROR_CHECK(err_code);
}
```

代码实现了两个功能，首先采用 sd_ble_gap_address_get 读取官方默认的 MAC 地址，然后再默认地址+1，再用 sd_ble_gap_address_set 写入到设备中，来验证我们设备写入 MAC 是否成功。

主函数中进行调用，注意，一定要在广播开始后设置，下次广播后新的 MAC 地址就设置成功。

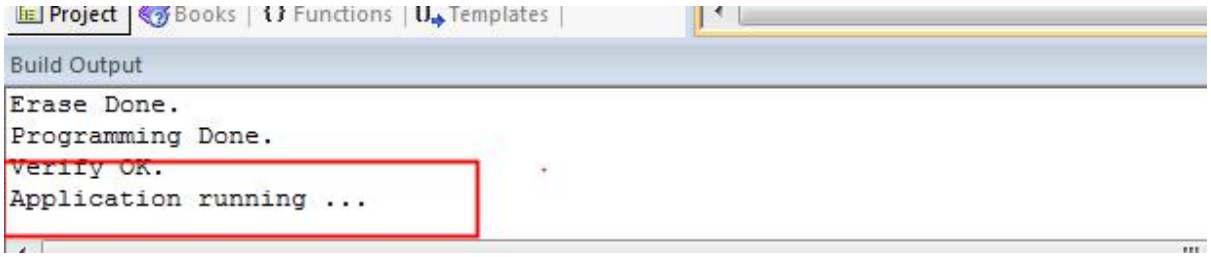
```
int main(void)
{
    uint32_t err_code;
    bool erase_bonds;
    uint8_t  start_string[] = START_STRING;

    // Initialize.
    APP_TIMER_INIT(APP_TIMER_PRESCALER,
APP_TIMER_OP_QUEUE_SIZE, false);
    uart_init();
    buttons_leds_init(&erase_bonds);
    ble_stack_init();
    gap_params_init();
    services_init();
    advertising_init();
    mac_set();
    conn_params_init();
    printf("%s",start_string);
    err_code = ble_advertising_start(BLE_ADV_MODE_FAST);
    APP_ERROR_CHECK(err_code);
    // Enter main loop.
    for (;;)
    {
        power_manage();
    }
}
```

编译后下载，打开 APP MCP 后发现 MAC 发生变化，如下图所示,新的 MAC 地址加 1，设置成功：

4 应用与调试

4.1 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 nrfgo 软件介绍篇。首先整片擦除，后下载协议栈。下载完后可以通过 keil 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：



4.2 测试

打开 APP MCP 后发现 MAC 发生变化，如下图所示，新的 MAC 地址加 1，设置成功：