谷雨物联网
iotxx.com

NRF52832DK-DFU固件升级教程

谷雨文档中心
http://doc.iotxx.com

2020-04-20

NRF52832DK-DFU固件升级教程

DFU是Device Firmware Update的缩写，翻译成中文是设备固件升级。设备固件升级是现在电子设备开发过程中不可规避的问题。下面将以谷雨物联的NRF52832DK评估板为硬件基础，详细介绍DFU的流程。

1 Bootloader 与 DFU 模型

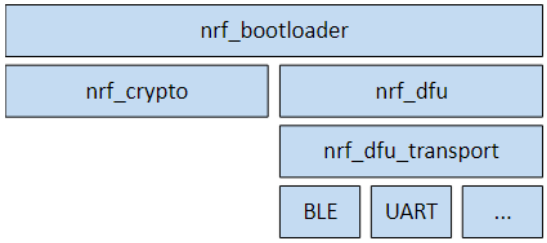
在Nordic提供的SDK中，bootloader 与DFU是其中的一部分。开发者可以在安装的SDK目录中找到。当然开发者也可以在它们的基础上，开发编译自己的bootloader。

一个基本的bootloader在运行后，将会启动指定空间的用户程序。当然可以在几个不同的用户程序间切换，或者在启动用户程序之前对设备进行初始化。但bootloader最重要的功能就是DFU。它主要有以下几个特性：

- 更新application,SoftDevice和bootloader
- 认证更新
- 降级预防
- 硬件兼容性验证
- 多种传输方式：（BLE，UART，USBD）
- 支持application 携带或不带SoftDevice
- 支持用独立于SoftDevice的固件替换依赖于SoftDevice的固件
- 支持使用依赖于SoftDevice的固件替换独立于SoftDevice的固件

注：开发者可以查看Nordic的官方原文文档说明。Bootloader and DFU modules章节。

下面是bootloader功能模块的结构框图：



bootloader modules

由上图可以看出它分为nrf_bootloader,nrf_crypto,nrf_dfu和nrf_dfu_transport功能模块。其中nrf_crypto实现安全特性，签名bootloader。在Nordic提供的SDK中，提供Secure Bootloader和Open Bootloader两种类型多种传输方式的bootloader。

注：关于nrf_bootloader,nrf_dfu,nrf_dfu_transport详细说明，可以查看Nordic的nRF5_SDK_15.2.0文档。

2 Bootloader详细说明

在DFU过程中，Bootloader占据作用的位置，这章节将详细说明DFU过程中，Bootloader所做的工用。

Bootloader主要负责的事项：

- 引导应用程序
- 激活新固件
- 进入DFU 模式，激活DFU传输，并交付接收到的新固件
- 喂狗看门狗定时器

在Nordic提供的SDK中，每个bootloader例子都包含一DFU传输方式。

2.1 Bootloader 设置页信息

这个设置页（settings page）存储在非易失性存储器（flash）中，用于保存bootloader与DFU相关的信息。这个设置页主要包含以下几方面的内容：

- 当前固件大小，CRC-32校验值
- 挂起固件大小，CRC-32校验值

- 固件更新进度
- 固件激活时度
- 当前固件版本（应用程序与bootloader）
- 一些特殊数据

2.2 固件激活（Fireware activation）

固件激活是固件更新最后一步。在启动期间，bootloader将会读取settings page里的相应信息，基于这些信息固件激活会被触发。固件激活涉及到新固件copy，以代替旧固件（如果应用程序是双块区域的，这个将在memory layout会用说明），同时会更改settings page信息，以请允许新固件启动。bootloader 要确保copy过程中断电安全。

2.3 DFU 模式（DFU mode）

在DFU模式，bootloader会打开DFU传输，以便设备接收新的固件。bootloader进入DFU模式，可以通过以下几个条件：

- 没有有效的应用程序存在
- SoftDevice与有效的应用程序都存在时，host请求bootloader进行应用程序更新
- 进入DFU模式可以通过以下几个源触发：
    - Button（NRF_BL_DFU_ENTER_METHOD_BUTON）
    - Pin reset（NRF_BL_DFU_ENTER_METHOD_PINRESET)）
    - GPREGRET寄存器设置特定的值（NRF_BL_DFU_ENTER_METHOD_GPREGRET）
    - 应用程序通过向settings page写入请求（NRF_BL_DFU_ENTER_METHOD_BUTTONLESS）

当进入DFU 模式，不活动定时器被启动。当定时器到期时，bootloader就会复位。任何DFU活动都会使不活动定时器重启。不活动定时器超时时间默认是NRF_BL_DFU_INACTIVITY_TIMEOUT_MS。如果SoftDevice激活后，设备进入DFU模式，不活动定时器超时时间被设置为NRF_BL_DFU_CONTINUATION_TIMEOUT_MS。

2.4 启动应用程序（Starting the application）

基于settings page的信息，bootloader可以确定是否有应用程序存在，处在什么位置。在启动应用程序之前，bootloader会检查程序的完整性。当然完整性检查也可以在特定的情况下跳过以减少启动时间（NRF_BL_APP_CRC_CHECK_SKIPPED_ON_GPREGRET2，NRF_BL_APP_CRC_CHECK_SKIPPED_ON_SYSTEMOFF_RESET）。只要以下之一条件发生，bootloader就会进入DFU模式：

- 没有application安装
- 完整性检查失败
- 没有setttings page

2.5 看门狗定时器支持（Watchdog timer support）

bootloader 检测看门狗是否活动并喂它以阻止看门狗复位。

2.6 Bootloader 依赖（Bootloader dependencies）

Bootloader模块除在传输时，不依赖SoftDevice。只有在使用BLE DFU 传输方式才依赖SoftDevice。

Bootloader也支持系统中根本不存SoftDevice的情况。

2.7 烧录bootloader（Programming the bootloader）

在系统启动期间，如果bootloader已经安装，MBR（主引导记录，Master Boot Record）负责启动bootloader。为此MBR必须知道bootloader的开始地址。bootloader开始地址存放到UICR.BOOTLOADERADDR中，UICR.BOOTLOADERADDR被映射在0x10001014（NRF_UICR_BOOTLOADER_START_ADDRESS）。因此，在烧写bootloader之前必须正确的设置UICR.BOOTLOADERADDR的值（这个操作开发者不用担心，bootloader程序已经做了）。

烧写bootloader程序要求以下几个步骤：

- 擦除设备

- 烧写SoftDevice（使用nRFgo Studio工具）
- 编译bootloader
- 烧写bootloader

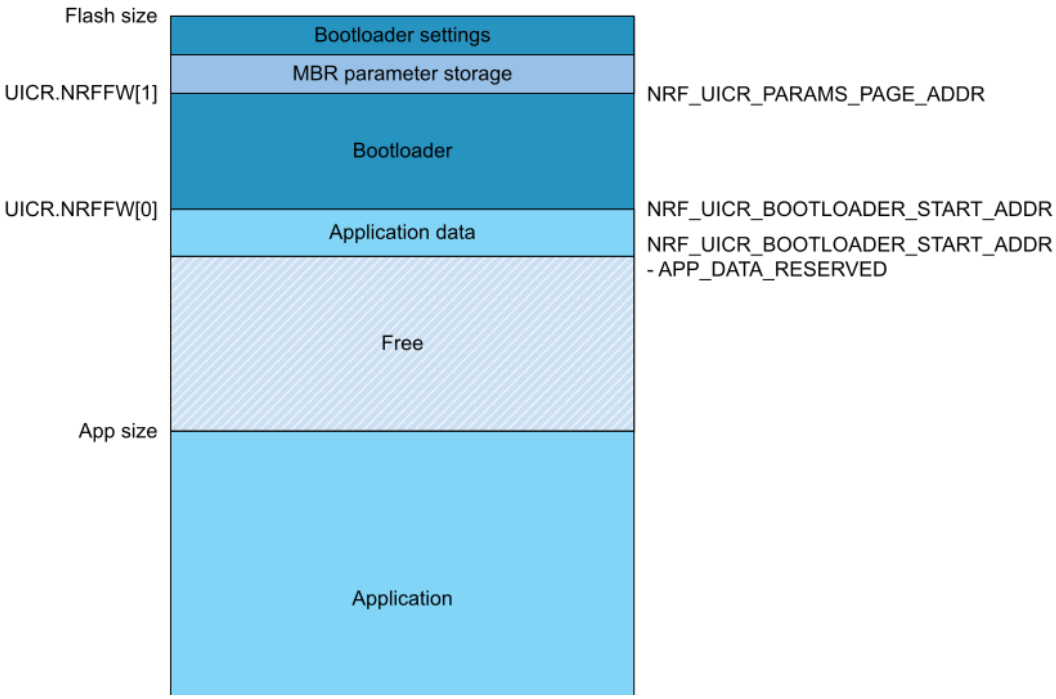
2.8 存储空间布局（Memory layout）

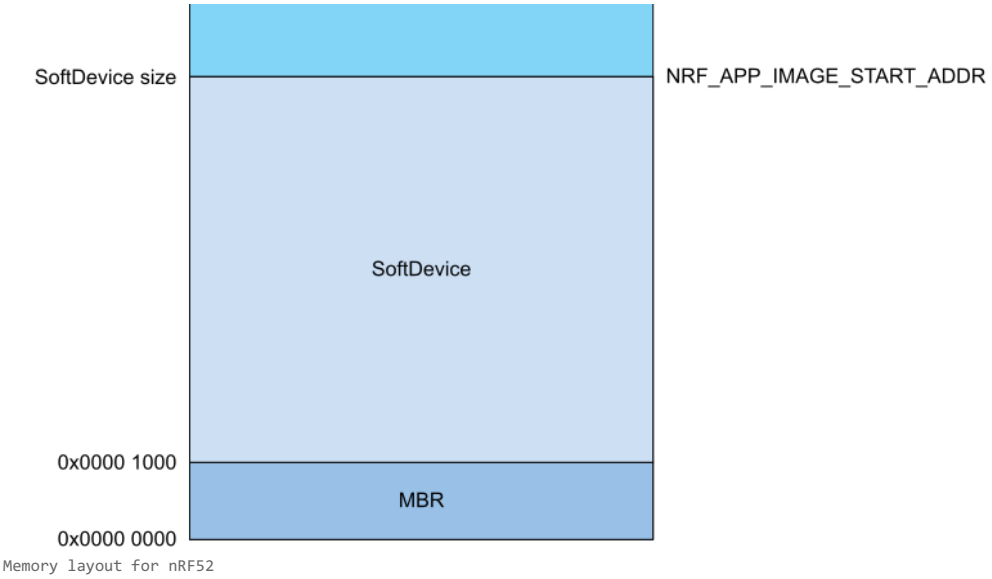
当添加bootloader到设备中时，你必须意识到不同的固件组件放到存储器哪里。

下表展示了不同器件与SoftDevice的存储空间分配：

组件	存储空间范围nRF52832（S132 v6.1.x）	存储空间范围nRF52840（S140 v6.1.x）	存储空间范围nRF52810（S112 v6.1.x）
Bootloader setttings	0x0007 F000 - 0x0008 0000 （4kB）	0x000F F000 - 0x0010 0000 (4 kB)	0x0002 F000 - 0x0003 0000 (4 kB)
MBR parameter storage	0x0007 E000 - 0x0007 F000 (4 kB)	0x000F E000 - 0x000F F000 (4 kB)	0x0002 E000 - 0x0002 F000 (4 kB)
Bootloader	0x0007 8000 - 0x0007 E000 (24 kB)	0x000F 8000 - 0x000F E000 (24 kB)	0x0002 8000 - 0x0002 E000 (24 kB)
Application area(incl. free space)	0x0002 6000 - 0x0007 8000 (328 kB)	0x0002 6000 - 0x000F 8000 (840 kB)	0x0001 9000 - 0x0002 8000 (60 kB)
SoftDevice	0x0000 1000 - 0x0002 6000 (148 kB)	0x0000 1000 - 0x0002 6000 (148 kB)	0x0000 1000 - 0x0001 9000 (96 kB)
Master Boot Record(MBR)	0x0000 0000 - 0x0000 1000 (4 kB)	0x0000 0000 - 0x0000 1000 (4 kB)	0x0000 0000 - 0x0000 1000 (4 kB)

下图展现了nRF52系列器件的默认空间分配。nRF52832有512kB flash大小，nRF52840有1024kB flash大小，nRF52810有192kB flash大小。



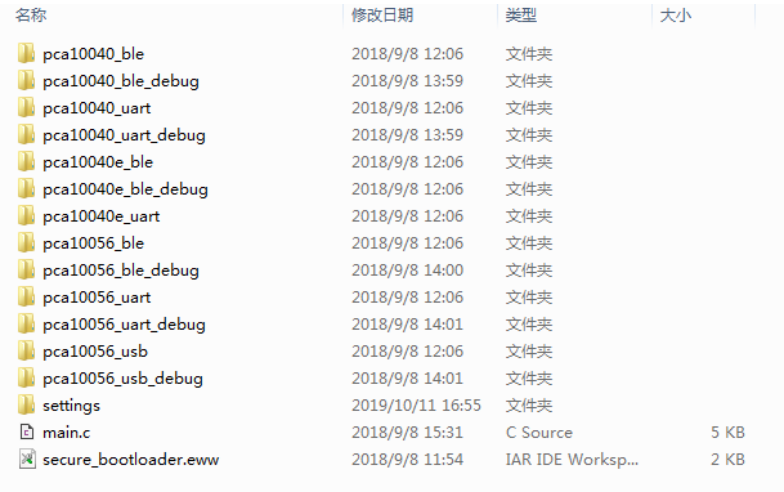


Memory layout for nRF52

3 实验前准备

关与DFU流程（nrf_dfu）与DFU协议（nrf_dfu_transport）这里不在详细说明，有兴趣的开发者可以自行查看Nordic的官方文档，可以在谷雨NRF52832DK评估板的资料中下载Noridc的SDK离线文档（推荐，目前最新为nRF5_SDK_15.2.0_offline_doc）。

接下来的说明与操作都基于DFU的secure bootloader中以ble为传输方式的bootloader。其位于Nordic SDK安装路径下/nRF5_SDK_15.2.0_9412B96/examples/dfu/secure_bootloader/pca_10040_ble。开发者安装不同版本的SDK可能会有所差异。



名称	修改日期	类型	大小
pca10040_ble	2018/9/8 12:06	文件夹	
pca10040_ble_debug	2018/9/8 13:59	文件夹	
pca10040_uart	2018/9/8 12:06	文件夹	
pca10040_uart_debug	2018/9/8 13:59	文件夹	
pca10040e_ble	2018/9/8 12:06	文件夹	
pca10040e_ble_debug	2018/9/8 12:06	文件夹	
pca10040e_uart	2018/9/8 12:06	文件夹	
pca10056_ble	2018/9/8 12:06	文件夹	
pca10056_ble_debug	2018/9/8 14:00	文件夹	
pca10056_uart	2018/9/8 12:06	文件夹	
pca10056_uart_debug	2018/9/8 14:01	文件夹	
pca10056_usb	2018/9/8 12:06	文件夹	
pca10056_usb_debug	2018/9/8 14:01	文件夹	
settings	2019/10/11 16:55	文件夹	
main.c	2018/9/8 15:31	C Source	5 KB
secure_bootloader.eww	2018/9/8 11:54	IAR IDE Worksp...	2 KB

在secure bootloader目录下有多个bootloader工程，这里我们使用pca10040_ble（S132），而pca_10056_ble是S140，RF52832器件不支持S140的SoftDevice。

3.1 辅助工具安装

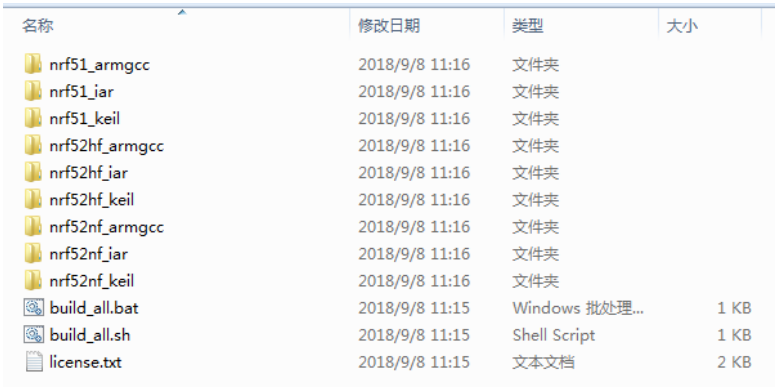
SDK12以后,DUF功能对升级文件进行了ECDSA签名加密,防止误升级未授权的程序。而Nordic使用micro-ecc开源软件实现ECDSA。

开发者初次安装SDK时，在SDK中是没有micro-ecc源码的，需要开发者去github上下载。如果开发都没有下载micro_ecc源码，则在编译bootloader时，编译器会报各种错误。主要有两个方面的如下：

错误内容	原因
Fatal Error[Pe035]: #error directive: "Debug public key not valid for production. Please see https://github.com/NordicSemiconductor/pc-nrfutil/blob/master/README.md to generate it"	没有有效的签名公钥
各种与micro-ecc的头文件：uECC.h等	没有micro-ecc源码

3.1.1 git安装

micro-ecc是外部开源软件，所以在Nordic的SDK中放到的协议栈目录下的external目录下。在external目录下，开发者会发现有一个micro-ecc目录。进入micro-ecc目录后，确定没有相应的源码，只有不同编译平台链接相关文件。



其中build_all.bat脚本文件（Windows），是用于编译micro-ecc源码的。在运行时会检查当前系统中是否安装了git。如果安装了，会进一步检查当前目录下是否有相应的源码，如果没有会用git去github上去下载。build_all.bat文件内容如下：

```
@ECHO OFF

:: This script will use git (must be in %PATH%) and arm-none-eabi tools in combination with GNU Make
:: to both fetch and compile all variants of micro-ecc for the nRF5 families

WHERE >nul 2>nul git
IF %ERRORLEVEL% NEQ 0 (
    ECHO "git is not installed. Please install and append to PATH."
)

IF NOT EXIST micro-ecc/uECC.c (
    ECHO "micro-ecc not found! Let's pull it from HEAD."
    git clone https://github.com/kmackay/micro-ecc.git
)

make -C nrf51_armgcc/armgcc
make -C nrf51_iar/armgcc
make -C nrf51_keil/armgcc
make -C nrf52hf_armgcc/armgcc
make -C nrf52hf_iar/armgcc
make -C nrf52hf_keil/armgcc
make -C nrf52nf_armgcc/armgcc
make -C nrf52nf_iar/armgcc
make -C nrf52nf_keil/armgcc
```

如果开发者的PC上没有安装git，则需要去网络找到git，并进行安装。此处安装过程不作说明。

安装git完成后，双击build_all.bat文件，此时git就会去下载micro-ecc源码。此时目录下多了一个micro-ecc文件夹，发现里面有相应的uECC.h与uECC.c等相关文件。

回到IAR工程，再次编译，发现没有报缺少micro_ecc相关的文件，只报了没有相应的公钥错误。

3.1.2 安装nrfutil

为了解决没有公钥的错误，我们需要public key与priavte key一对密钥对，使用ECDSA_P256_SHA256算法对DFU程序进行签名并加密。生成密钥对，需要使用nrfutil工具。nrfutil在nRFgo studio工具安装路径中也有，但是版本非常低（0.3.0版本）。当然，开发者如果想要使用更新版本的nrfutil，可以自己安装。

nrfutil是一个python工具，所以开发者只要安装Python就可以了（Python下载地址）。但是要注意，Python必须在2.7-3.0版本之间。Python安装完成后，在Python的路径下使用**python -m pip install nrfutil**命令安装nrfutil。

nrfutil会安装在python路径下的Scripts文件夹内。强烈建议将nrfutil配置到PC的环境变量中，这样就不需要到Scripts文件夹下执行nrfutil命令。

3.1.3 生成密钥对

- 生成自己的私钥（private key）

```
nrfutil keys generate private.pem
```

- 根据私钥生成公钥（public key）

```
nrfutil keys display --key pk --format code private.pem --out_file public_key.c
```

此命令执行完成后，会在当前路径下生成public_key.c文件。这个文件是根据私钥生成的公钥数组。

```
/* This file was automatically generated by nrfutil on 2019-09-25 (YY-MM-DD) at 17:57:45 */

#include "stdint.h"
#include "compiler_abstraction.h"

/** @brief Public key used to verify DFU images */
__ALIGN(4) const uint8_t pk[64] =
{
    0x18, 0xd3, 0xbc, 0xdd, 0x92, 0x7a, 0xdd, 0xa7, 0x73, 0xbf, 0x11, 0xb7, 0x08, 0x59, 0x6d, 0x23, 0x52, 0xf6, 0x47, 0x01, 0xaf, 0xdb, 0xdc, 0xaf, 0x5a, 0x83, 0x03, 0x1a, 0x0a, 0xf8, 0x5b, 0xaf,
    0x9c, 0x0d, 0x37, 0x9c, 0x77, 0x69, 0xd4, 0x14, 0xab, 0x4c, 0x32, 0x02, 0x94, 0xc3, 0x15, 0x6e, 0xfd, 0x9d, 0xc9, 0xe5, 0xc3, 0x33, 0x1a, 0x69, 0xf3, 0x85, 0xa2, 0x31, 0x85, 0xc6, 0x97, 0x42
};
```

需要将uint8_t pk[64]复制到Bootloader工程的Application下的dfu_public_key.c文件中，替换#error的内容。完后如下：

```
/* This file was automatically generated by nrfutil on 2018-09-08 (YY-MM-DD) at 06:07:33 */

#include "sdk_config.h"
#include "stdint.h"
#include "compiler_abstraction.h"

#if NRF_CRYPTO_BACKEND_OBERON_ENABLED
/* Oberon backend is changing endianness thus public key must be kept in RAM. */
#define _PK_CONST
#else
#define _PK_CONST const
#endif


/* This file was generated with a throwaway private key, that is only inteded for a debug version of the DFU project.
   Please see https://github.com/NordicSemiconductor/pc-nrfutil/blob/master/README.md to generate a valid public key.
*/

#ifdef NRF_DFU_DEBUG_VERSION

/** @brief Public key used to verify DFU images */
__ALIGN(4) _PK_CONST uint8_t pk[64] =
{
    0x40, 0xe5, 0x14, 0xb4, 0x6d, 0xb9, 0x83, 0xc7, 0x1c, 0x33, 0x92, 0x17, 0x35, 0x11, 0xe2, 0x00, 0x8b, 0x52, 0x24, 0xbd, 0xbb, 0x6b, 0x6a, 0xe8, 0x68, 0x1a, 0x32, 0xfb, 0x77, 0x15, 0xe1, 0xe1,
    0xd9, 0xbc, 0x43, 0xbb, 0x55, 0x6f, 0xf6, 0x9e, 0x3d, 0x04, 0x49, 0x5b, 0xbc, 0x47, 0xa3, 0x69, 0x68, 0x24, 0x15, 0x4b, 0x5e, 0x9c, 0x9d, 0x6b, 0xf4, 0x4e, 0x62, 0x59, 0xd7, 0x24, 0xc4, 0x71
};

#else
/** @brief Public key used to verify DFU images */
__ALIGN(4) const uint8_t pk[64] =
{
    0x18, 0xd3, 0xbc, 0xdd, 0x92, 0x7a, 0xdd, 0xa7, 0x73, 0xbf, 0x11, 0xb7, 0x08, 0x59, 0x6d, 0x23, 0x52, 0xf6, 0x47, 0x01, 0xaf, 0xdb, 0xdc, 0xaf, 0x5a, 0x83, 0x03, 0x1a, 0x0a, 0xf8, 0x5b, 0xaf,
    0x9c, 0x0d, 0x37, 0x9c, 0x77, 0x69, 0xd4, 0x14, 0xab, 0x4c, 0x32, 0x02, 0x94, 0xc3, 0x15, 0x6e, 0xfd, 0x9d, 0xc9, 0xe5, 0xc3, 0x33, 0x1a, 0x69, 0xf3, 0x85, 0xa2, 0x31, 0x85, 0xc6, 0x97, 0x42
};
#endif
```

完成public_key的复制之后，再次编译bootloader工程。 此时已经没有了public key相关错误了，但是报了没有micro_ecc_lib_nrf52.a文件。具体内容如下：

```
Fatal Error[Li001]: could not open file "E:\Nordic_BLE\NRF52832_new\nRF5_SDK_15.2.0_9412b96\external\micro-ecc\nrf52hf_iar\armgcc\micro_ecc_lib_nrf52.a"
```

这个原因是上面下载了micro_ecc源码之后，没有编译生成相应的库文件造成的。

3.1.4 安装make工具

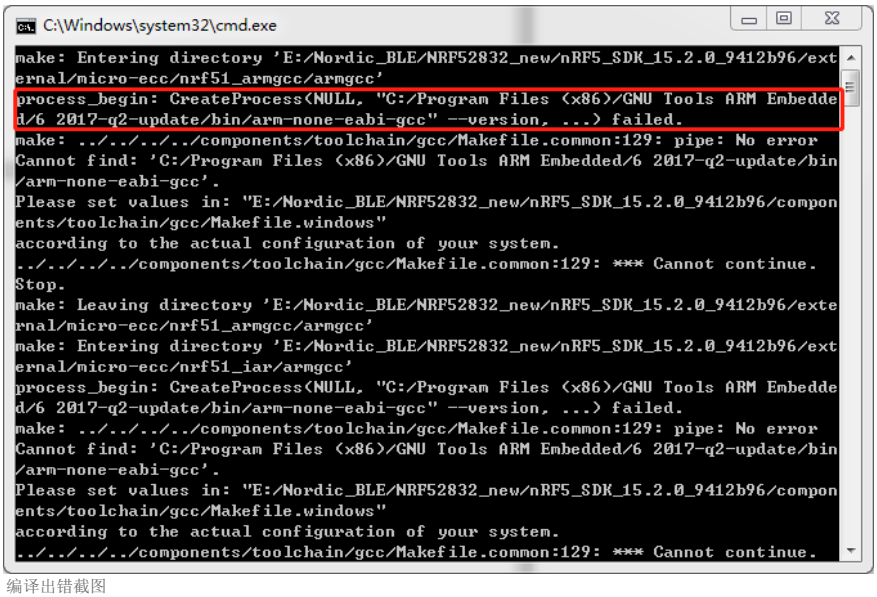
在git安装章节中，build_all.bat文件的最后有相应的make命令，这些make命令就是编译指令（所有编译相关的操作，都是通过makefile完成的，开发者不用编写makefile，SDK中已经帮用户完成这些makefile文件，开发者只要安装make工具，并双击运行build_all.bat）。

make 工具开发者可以自行安装，也可以在谷雨NRF52832DK评估板DFU相关目录下找到make工具。如果是绿色运行软件，开发者要进行PC环境变量设置，否则在运行build_all.bat时会报错。

3.1.5 安装gcc-arm编译器

运行build_all.bat文件后，再次编译bootloader工程，发现错误依旧存在。在次返回到micro-ecc目录下，发现各个平台目录下没有相应的.a库文件。难道源码有错误，导致make出错？带个这个疑问，在build_all.bat文件后，加上一个pause的bat命令，即不让windows命令窗口自行退出。再次运行build_all.bat文件。

发现命令窗口确实打印出相应的出错信息。经仔细查看PC上没有安装相应的编译器。开发者可以在在谷雨NRF52832DK评估板DFU相关目录下找到相应的gcc_arm编译器，也可以自行在下载。



编译出错截图

点击安装gcc_arm编译器，谷雨提供是（gcc-arm-none-eabi-8-2019-q3-update-win32-sha2.exe）。安装过程不做说明。

安装完成后需要修改SDK安装目录下components/toolchain/gcc/Makefile.windows文件。将编译器版本改成当前我们安装的版本。Makefile.windows默认是**GNU_INSTALL_ROOT := C:/Program Files (x86)/GNU Tools ARM Embedded/6 2017-q2-update/bin/**。这个是编译安装的路径。需要将其改成已经安装的8-2019-q3-update版本路径。最后修改完成后内容如下。

```
1 GNU_INSTALL_ROOT := C:/Program Files (x86)/GNU Tools ARM Embedded/8 2019-q3-update/bin/
2 GNU_VERSION := 6.3.1
3 GNU_PREFIX := arm-none-eabi
```

完成修改后，保存Makefile.windows文件。

回到micro-ecc目录下，运行build_all.bat文件，再次执行编译。此时命令行中正常输出编译过程。现在的micro-ecc的各个平台目录下有了相应的.a或.lib文件。

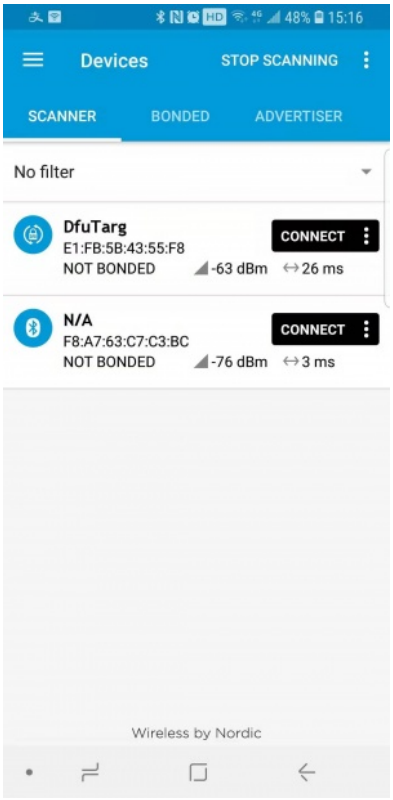
3.2 bootloader烧写

再次编译bootloader工程，此时已没有任何错误警告。工程生成secure_bootloader_ble_s132_pca10040.hex

文件，这个bootloader文件。由于此bootloader程序是基于BLE的，所以要想运行程序，还要SoftDevice。接下来所有操作都是以谷雨的NRF52832DK评估板为硬件平台进行操作。

烧写步骤开发者可以查看《烧录bootloader（Programming the bootloader）》章节。

打开nRFgo Studio，擦除器件，烧写SoftDevice，烧写生成的bootloader文件。运行程序，评估板的D1,D2 led亮点亮。打开手机端的nRF Connect，并打开Scan，便可以看到名称为DfuTarg广播设备。



nRF Connect 扫描截图

4 生成DFU .zip包

要想完成DFU升级设备程序。DFU.zip程序包是必不可少的。DFU 主机是通过.zip包将新程序发送给DFU 设备。所以.zip包要包含我们想要升级的hex文件，初始化数据和包的签名。

接下来的教程，我们只做application部分的升级。

- 准备一个应用程序

在SDK中选择一个例程编译并生成相应的hex文件。为了简单起见，选择一个led_blinky例子，路径examples/ble_peripheral/ble_app_blinky。将生成的hex文件拷到一个开发者创建的目录下，并改名ble_app_blinky.hex（主要是为后面命令输入简单点）。

- 生成.zip文件

将《生成密钥对》章节，生成的私钥拷到指定目录下（最好与ble_app_blinky.hex在同个目录下）。利用nrfutil打包生成.zip文件。

```
nrfutil pkg generate --hw-version 52 --application-version 1 --application ble_app_blinky.hex --sd-req 0xaf --key-file private.pem app_dfu_package.zip
```

解释部分参数：

- --hw-version

默认情况下，是要与你的芯片匹配。如果开发者使用的是nRF51 芯片，这里就要填写“51”；如果使用的是nRF52芯片，这里就要填写“52”。但如果想要填写自己的硬件版本号，你可以在bootloader中程序中，使用#define NRF_DFU_HW_VERSION your_hw_number 定义自己的号。

- --application-version

默认情况下，应用程序版本号是从0开始的。为了能顺利更新应用程序，应用程序的版本号必须大于等于bootloader中存在数据。因为bootloader是第一次使用，bootloader里存的版本号是0，我们在这里将application-version 写成1。

- --sd-req

这个参数是标示SofteDevice 代号（code number）。在写这个文档时，我们使用的SofteDevice是S132 V6.1.0，它的代号是0xAF。开发者可以通过**nrfutil pkg generate --help**指令查询SoftDevice的代号。如果仍没有列出代号，可以通过nRFgo Studio查看，只要用评估板烧写相应的SoftDevice。

- --applicatioin

告诉nrfutil，将要升级的应用程序。

4.1 测试DFU

现在，已经准备好了DFU .zip文件，bootloader也已经在NRF52832DK评估板上运行。接下只要将.zip文件通过DFU主机发送给设备就可以了，便完成application空中升级。由于bootloader在启动后没有发现有效的application，就会进入DFU模式（bootloader章节有说明）。

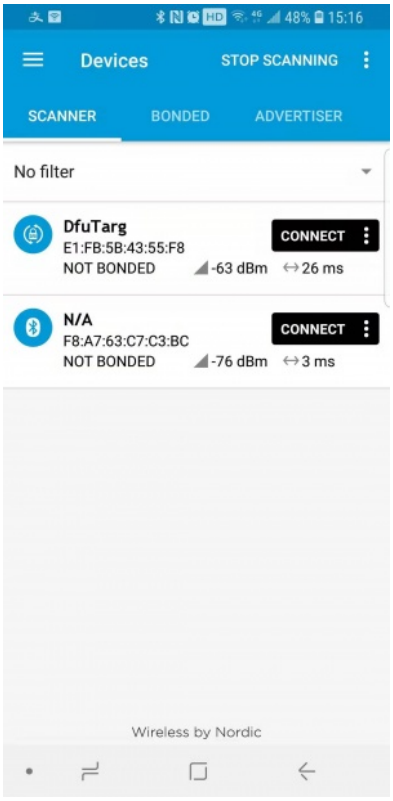
- 拷贝上述生成的app_dfu_package.zip

一般我们会使用手机作为DFU主机（因为手机方便）。并配合nRF Connect手机应用

- 使用nRF Connect完成DFU

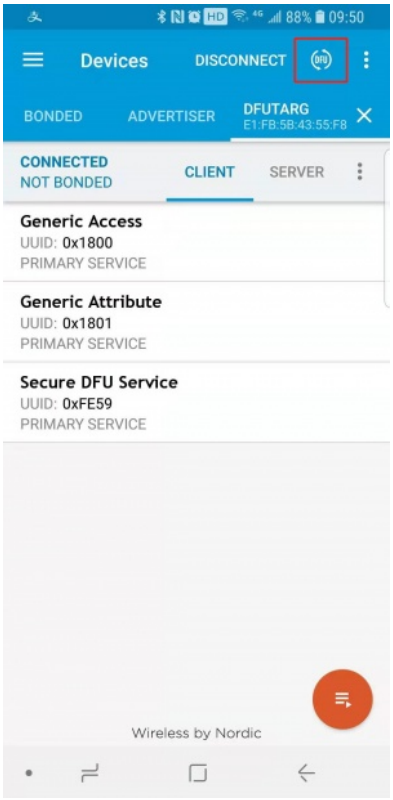
nRF Connect目前只有Android版的APK，开发者可以自行下载，也可以在谷雨的资料包里找到nRF Connect手机安装包。下面将描述手机操作步骤。

- 打开nRF Connect，并进行扫描，可以发现一个DfuTarg设备。如下图所示。

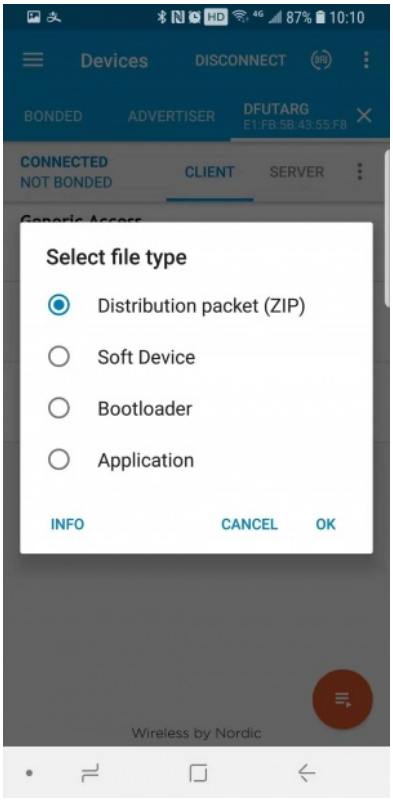


nRF Connect

- 点击旁边的**CONNECT**按钮，连接Dfutarg设备，截图如下图所示。

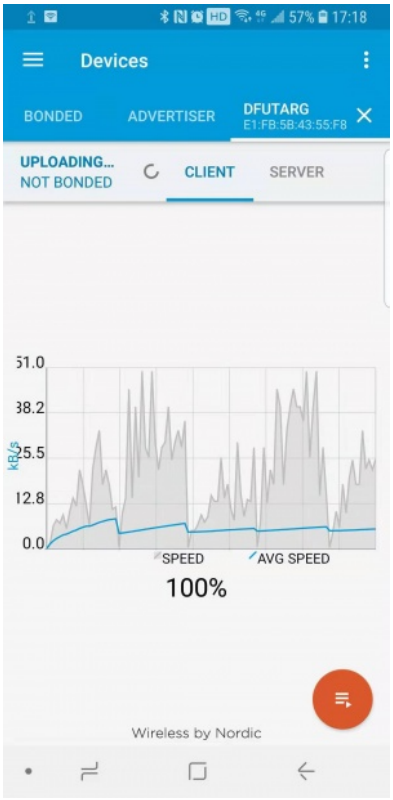


在上图的右上角，有个DFU按钮，此按钮用于选择升级文件，即上述步骤中生成的.zip。在这里我们选择Distribution packet(ZIP)。点击**OK**按钮，选择上述打包的app_dfu_package.zip文件。



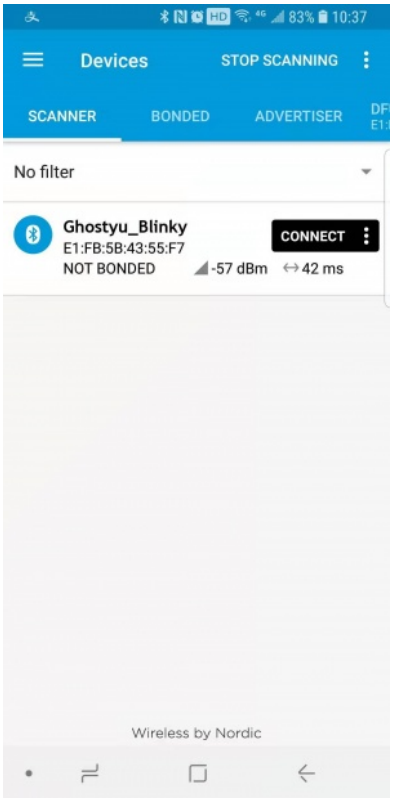
- 传输ZIP文件

选定zip文件后，nRF Connect便开始向设备传输，要升级的文件。



- 应用程序运行

传输完成后，再使用nRF Connect进行扫描，发现已经找到DfuTarg设备了，但是有个Ghostyu_Blinky设备。DFU升级应用程序已经成功，应用程序已经运行。



5 Bootloader切换DFU模式

如果升级的application中没有进入DFU模式相关的代码，可以使用bootloader中NRF_BL_DFU_ENTER_METHOD_BUTTON方法进入DFU模式。在实验部分烧写的bootloader程序中，可以通过16号引脚进入DFU模式。具体方法操作方法如下（以谷雨的NRF52832KD评估板为硬件平台）：

- 按下按钮S4，并保持（16号引脚，对应评估板按钮S4）
- 复位评估板
- 此时系统就会进入DFU模式，开发者可以通过nRF Connect自行验证

- 本页面最后编辑于2020年1月10日 (星期五) 10:48。