



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**

28 空中升级 DFU 功能详解

我们在产品批量后, 后期如果出现 BUG 或者需要需要增加产品功能, 需要对固件进行修复或者升级, 如果这时我们重新连接仿真器进行下载, 则会破坏产品的完整性, 并且相当的费时费力, 这对开发者是相当麻烦的问题。因此这种情况下我们可以采取空中升级或者串口升级。

那么如何实现了? 什么叫 BLE OTA? 什么叫 DFU? 如何通过 UART 实现固件升级? 又如何通过 USB 实现固件升级? 怎么保证升级的安全性? 什么叫双备份(dual bank)DFU? 什么叫单备份(single bank)DFU? 本文将对上述问题进行探讨。

这一讲就主要讲解下如何实现空中升级, 并且对以上的专业术语进行解释:

28.1: DFU 的功能介绍

28.1.1 DFU 的原理

所谓 DFU (Device Firmware Update), 就是设备固件升级的英文解释, 而 OTA 是 DFU 的一种类型, 是(Over the Air)的英文简称, OTA 的全称应该是 OTA DFU, 只不过大家为了方便起见, 直接用 OTA 来指代固件空中升级。DFU 除了可以通过无线方式(OTA)进行升级, 也可以通过有线方式进行升级, 比如通过 UART, USB 或者 SPI 通信接口来升级设备固件。

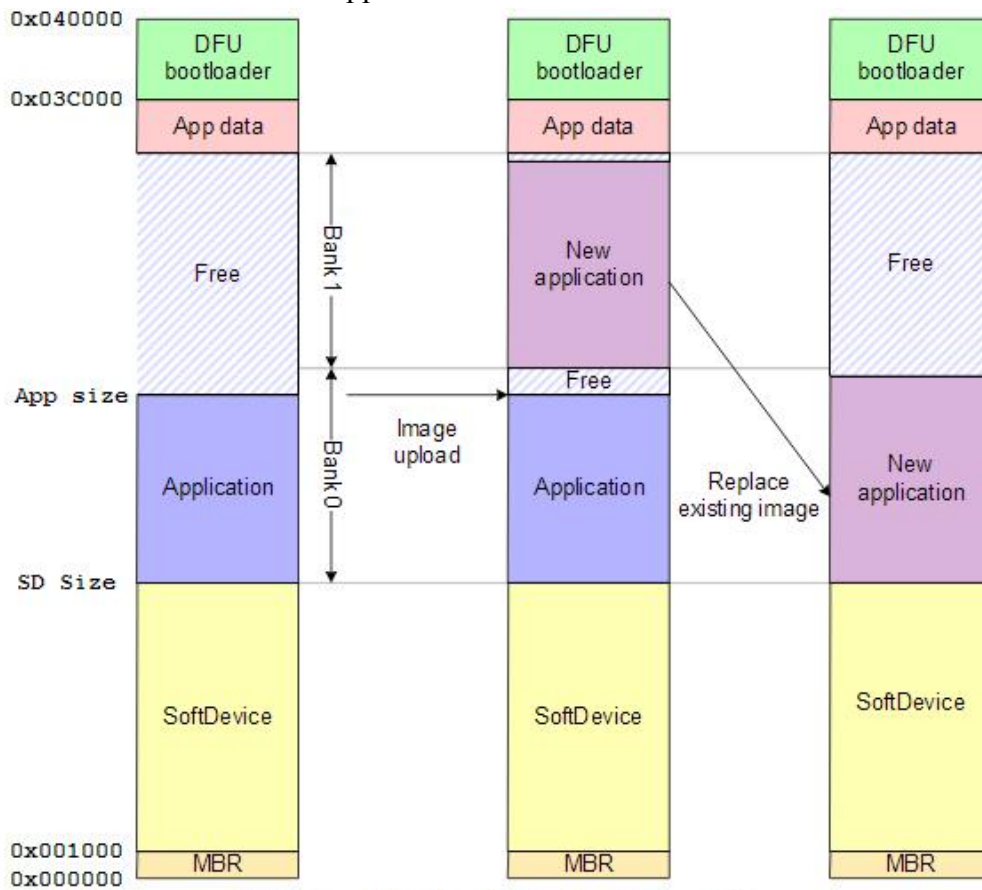
DFU 可以采用 dual bank 或者 single bank 模式, dual bank 的做法是升级时系统先进入 bootloader, 然后把新系统(新固件)下载下来并校验成功, 然后再擦除老系

统（老固件）并升级新系统，dual bank 方式虽然牺牲了很多存储空间，但是换来了更好的升级体验。

Single bank 的做法是升级时系统也是先进入 bootloader，然后立马把老系统擦除，然后直接把新系统下载到老系统区域，跟 dual bank 相比，single bank 将大大节省 Flash 存储区域，在系统资源比较紧张的时候，推荐使用 single bank 方式。不管是 single bank 还是 dual bank，升级过程出现问题后，都可以进行二次升级，都不会出现“变砖”情况。不过 dual bank 有一个好处，如果升级过程中出现问题或者新固件有问题，它还可以选择之前的老系统继续执行而不受其影响。而 single bank 碰到这种情况就只能一直待在 bootloader 中，然后等待二次或者多次升级，此时设备的正常功能就无法使用了，从用户使用这个角度来说，你也可以认为此时设备已经“变砖”了。

1: Dual Bank Flash 布局

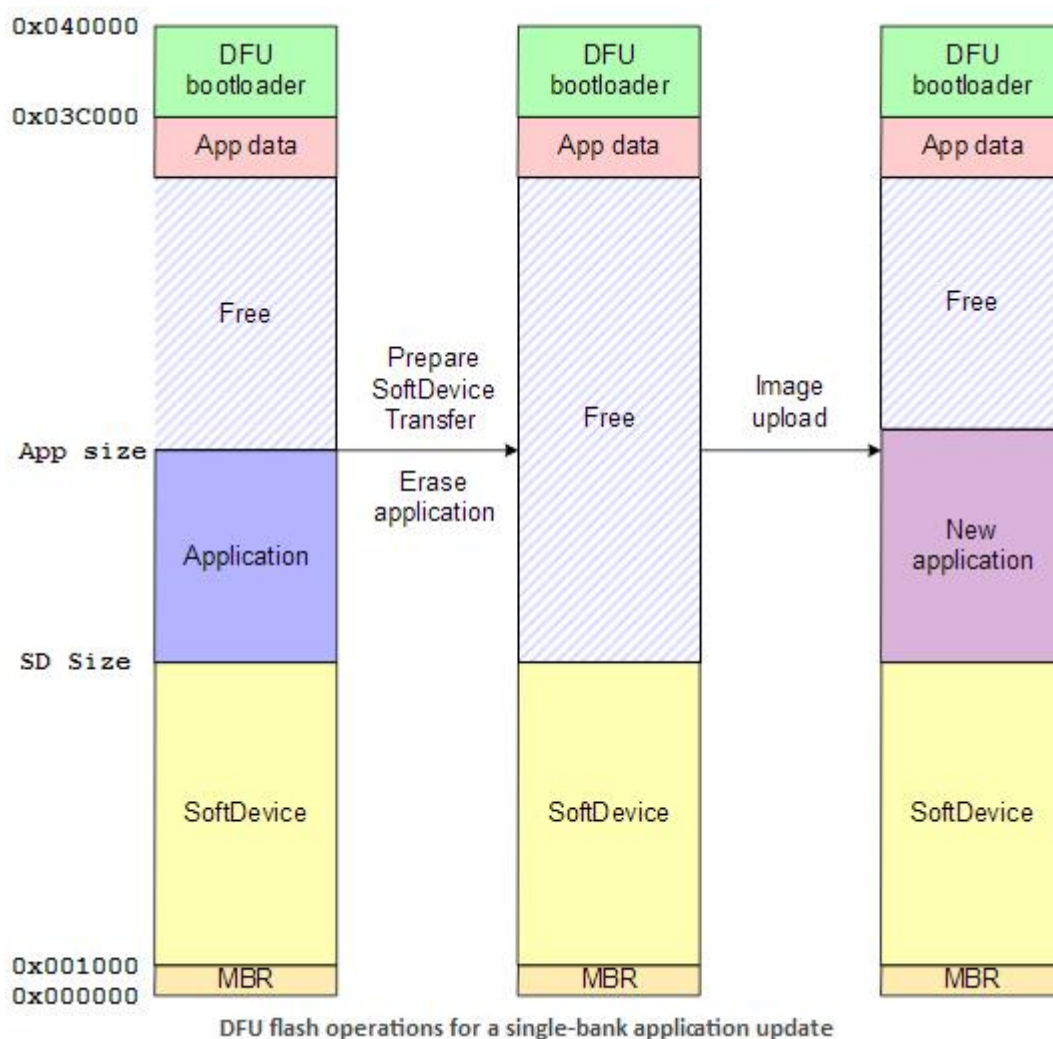
如图所示：Dual-bank 在更新过程中会先将新的 app 放在 flash 中，等全部就收完了。然后会检验收到的 app 是否有效，如果有效就替换掉现有的 app。这样即使新的 app 是无效的，至少还有旧的 app 可以运行。



DFU flash operations for a dual-bank application update

2. Single Bank Flash 布局

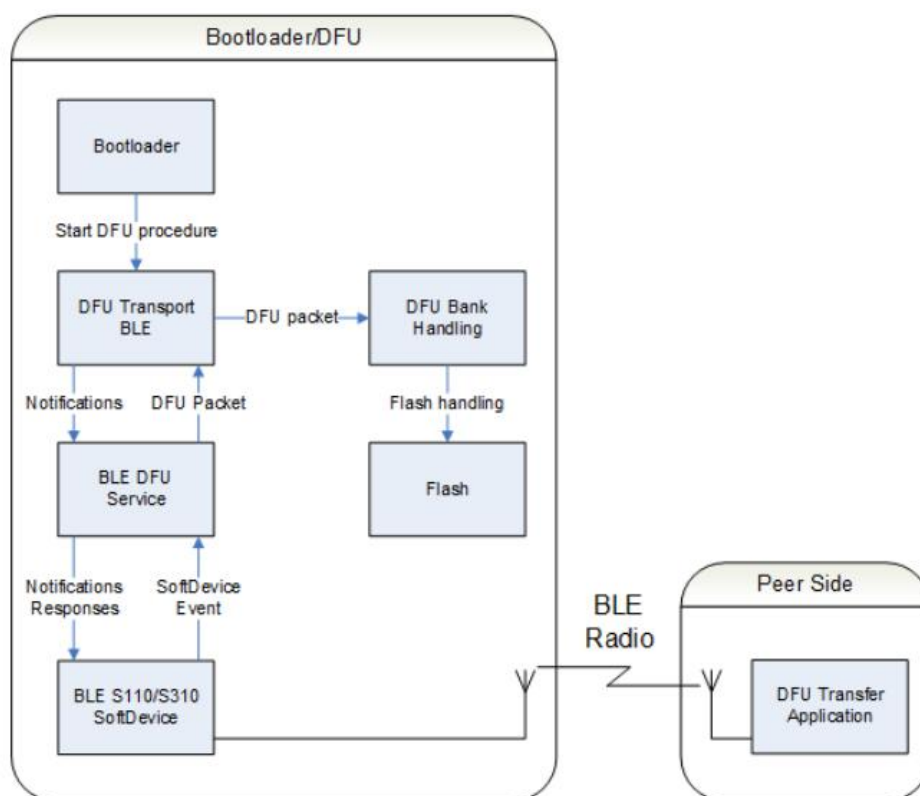
Single-bank 模式是在更新过程中就直接开始用新的 app 覆盖旧的，因为不需要在 flash 中放两个 app，这种情况可以更新更大的 app，缺点是一旦传输错误。会导致新旧 app 都无法使用了。



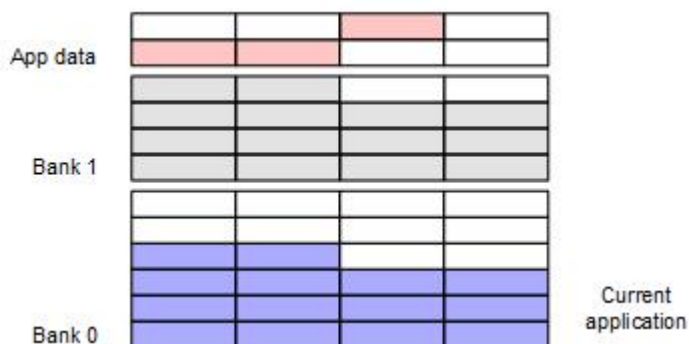
可以对比下前面的 bootloader 和 sd 的更新只能用 dual-bank 模式, 因为使用 single_bank 模式, 一旦传输错误设备就无法启动了, 就只能通过 flash 工具重新烧写了。而应用程序更新 dual 或 single 模式都可以。即使出错设备因为 bootloader 还在就可以再下载 app。

在实际给出的 nRF52832 例子中, 我们确定使用 Dual bank 的方式进行 OTG 升级。设备烧录了 bootloader 程序后, 设备会工作在 DFU 模式, 这时可以使用手机 DFU 工具或者 PC 端的 Master Control Panel 软件 (配合 dongle) 对设备进行 DFU 操作。

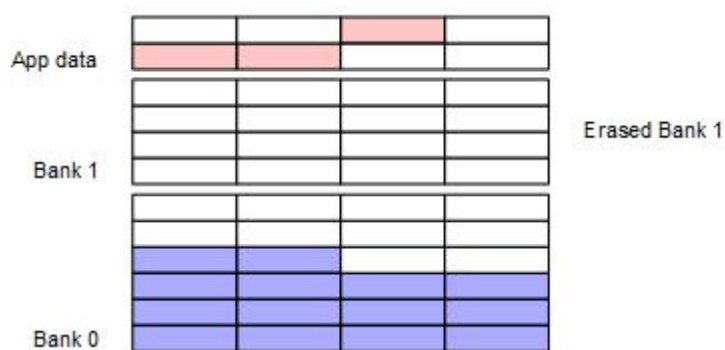
设备上电完成后, 系统会先运行 bootloader, bootloader 会判断在 bank0、是否有应用程序。如果在 bank0 中有应用程序, bootloader 会去执行应用程序, 否则系统会一直处在 DFU 模式, 等待应用程序更新。系统的执行流程框架图如下图所示:



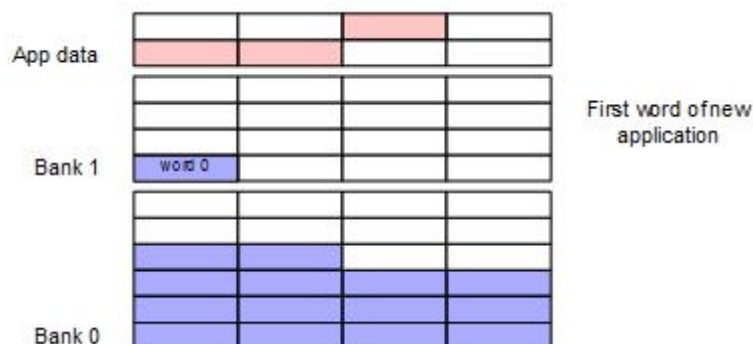
1.在升级程序之前，当前的应用程序存放在 Bank 0，此时 Bank 1 的存储空间未被使用。



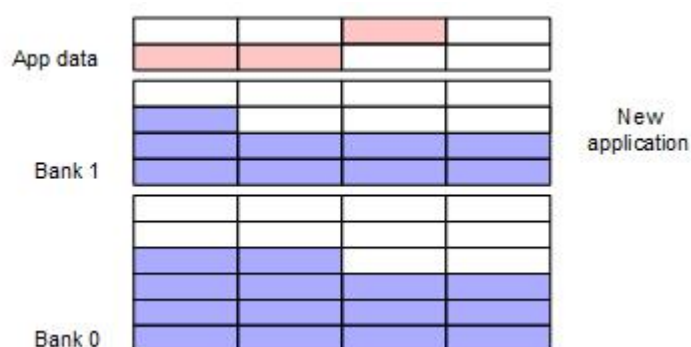
2.Bootloader 进入 DFU 模式会将 Bank 1 区域擦除，用于存放将要升级的应用程序数据，只有接收的数据校验成功，才会去擦除 Bank0 的程序，这可以确保当升级程序失败时，旧的应用程序还可以正常运行，不至于系统停止运行。



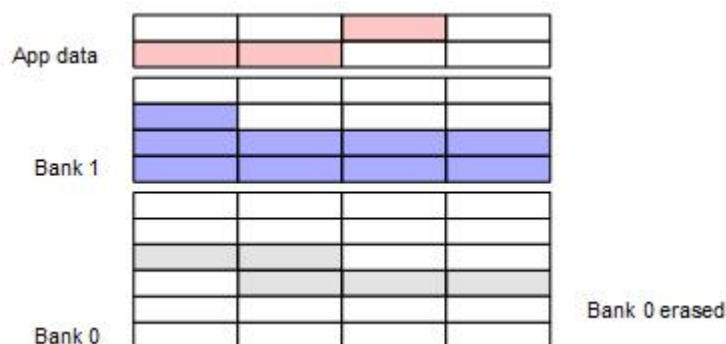
3.将接收到的新应用程序的数据包写入 Bank 1。



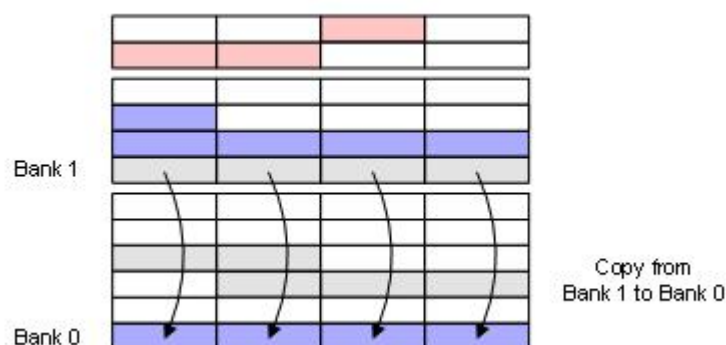
4.完成将待升级的应用程序完成写入 Bank 1 中后,新应用程序和旧的应用程序都会存在 Flash 存储空间中,这样可以确保当新的应用程序无法启动时,还可以运行旧的应用程序。



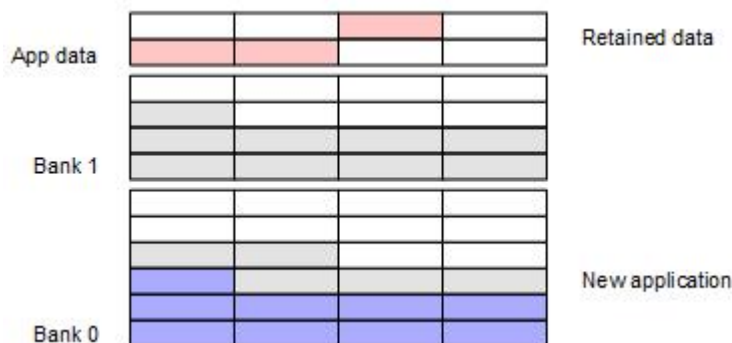
5.待新接收的数据检验成功后, Bank 0 中的旧应用程序会被擦除。



6.将 Bank 1 中的数据拷贝到 Bank 0 也是激活新应用程序的一部分。



7.完成 Bank 1 到 Bank 0 数据的拷贝后,会开始运行 Bank0 的应用程序。Bank 1 中数据不会被擦除,等待再次进入 bootloaderDFU 模式才擦除 Bank1 数据。



8. 如果设置了将旧应用程序的数据保留, 新的应用程序会将其数据在原有数据存储空间上叠加存储, 不会覆盖。这就是 single Bank 和 Dual Bank 的区别了。

28.1.2 DFU 升级工具

Nordic 提供的 SDK 就提供了 OTA (BLE) DFU, UART DFU (后面会专门做一篇教程), 以及 USB DFU (nrf52840 独有) 例程, 大家可以直接参考 Nordic 例程来实现自己的 DFU。由于 Nordic SDK 版本很多, 而且每个版本之间都或多或少有些差异, 前面出过 2 个版本, SDK6.0 和 SDK11 版本的 DFU 教程, 那么本文将以 SDK15 版本来阐述 Nordic 的空中升级详细讲解。

之前的 SDK9/10/11 版本 Nordic 只有明文 DFU 的演示代码。从 SDK12 到 SDK14 开始, Nordic 开始支持安全 DFU (secure DFU) 的演示代码。SDK15 版本同时提供明文 DFU 和安全 DFU (secure DFU) 的演示代码。

所谓 secure DFU, 不是指升级时固件是加密的, 而是指升级之前 bootloader 会先验证新固件的签名, 只有验签通过后, 才允许后续升级, 此时的升级方式仍然是明文; 验签失败, 则拒绝后续升级。Secure DFU 的方式可以防止任意第三方接入升级, 大大提高系统的安全性。

secure DFU 需要比较多的工具, 下面我们一一罗列出来:

- 1: gcc-arm-none-eabi 编译环境: GCC 编译环境
- 2: mingw 平台:
- 3: micro-ecc-master 源码:
- 4: python 安装文件:
- 5: pc-nrfutil 工具:
- 6: nrfgostudie 环境:
- 7: nrf connect app:

软件可以通过下面网址进行更新下载:

gcc-arm-none-eabi 编译环:

<https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads>

mingw 平台: <https://sourceforge.net/projects/mingw/files/latest/download?source>

micro-ecc-master 源码: <https://github.com/kmackay/micro-ecc>

python 安装文件: <https://www.python.org/downloads/>

pc-nrfutil 工具: <https://github.com/NordicSemiconductor/pc-nrfutil/>

或者通过我们网盘下载好的 DFU 软件直接安装使用

注意: 同时根据升级时如何跳转到 Bootloader, nRF5 SDK 不同的版本又将 DFU 分为按键式 DFU 和非按键式 (Buttonless) DFU, 所谓按键式 DFU, 就是上电时长按某个按键以进入 bootloader 模式。而 buttonless DFU, 就是整个 DFU 过程中设备端无任何人工干预, 通过 BLE 指令方式让设备进入 bootloader 模式。

SDK11 之前版本采用按键式 DFU, 按键式 DFU 比较简单, 你只需将 softdevice 和 bootloader image 烧入到设备中 (application 可烧可不烧), 按住 button4 然后上电, 设备就会自动进入 bootloader 模式, 然后就可以通过 nRF Connect 或者 nRF Toolbox 对设备进行 OTA 了。

SDK12 后的版本采用非按键式 (Buttonless) DFU, 一旦 buttonless DFU 例子从 app 跳到了 bootloader, 后续 DFU 升级过程就跟按键式 DFU 一模一样, 所以如果你对按键式 DFU 操作过程中有什么不明白的地方, 可以参考后面的 buttonless DFU 的说明, 这里就不单独对按键式 DFU 操作过程进行说明了。下面的升级步骤将以对 buttonless DFU 的为例子进行讲述:

28.2 DFU 文件制作步骤

28.2.1 GCC 编译环境的安装

1. 安装 gcc-arm-none-eabi, 双击安装包

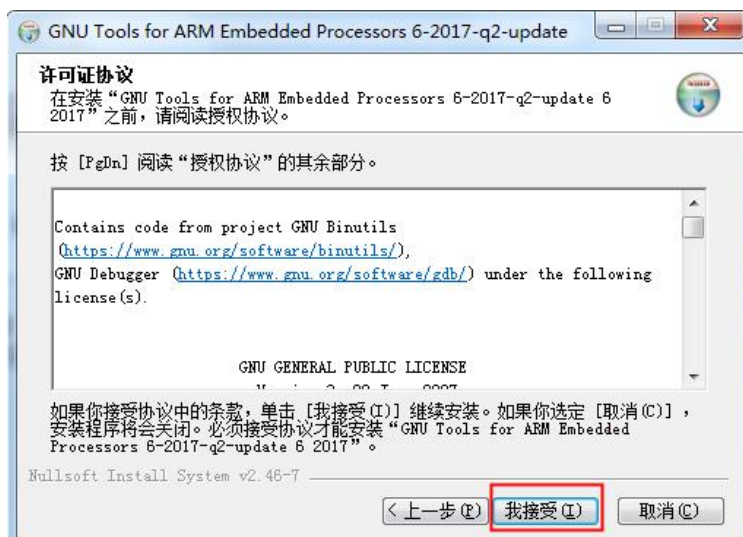
“gcc-arm-none-eabi-6-2017-q2-update-win32.exe” 后弹出语言选择框, 点击 OK:



2. 弹出 gcc-arm-none-eabi 的安装向导, 点击下一步继续安装:



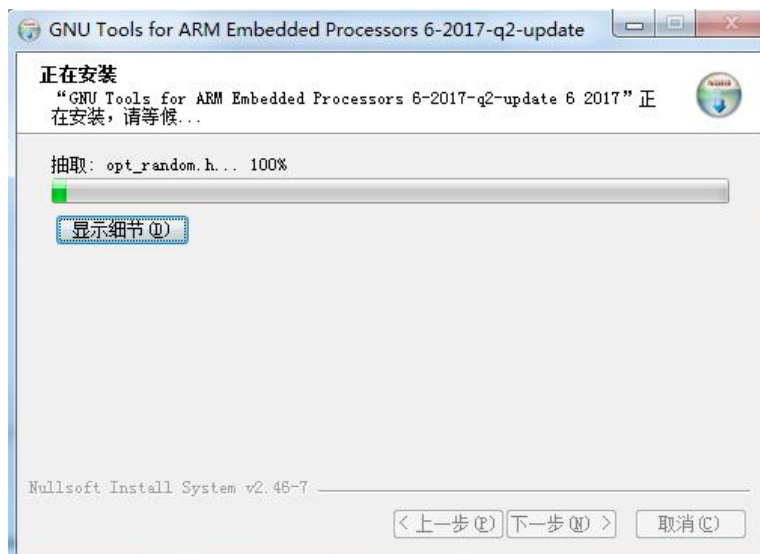
3. 提示许可证协议条款，同意条款，点击“我接受”：



4. 弹出选择安装路径选项，如下图所示，使用默认的地址路径，不要修改，然后点击安装：



5. 点击安装后, 会出现安装进度条, 如下图所示, 等待安装完成:



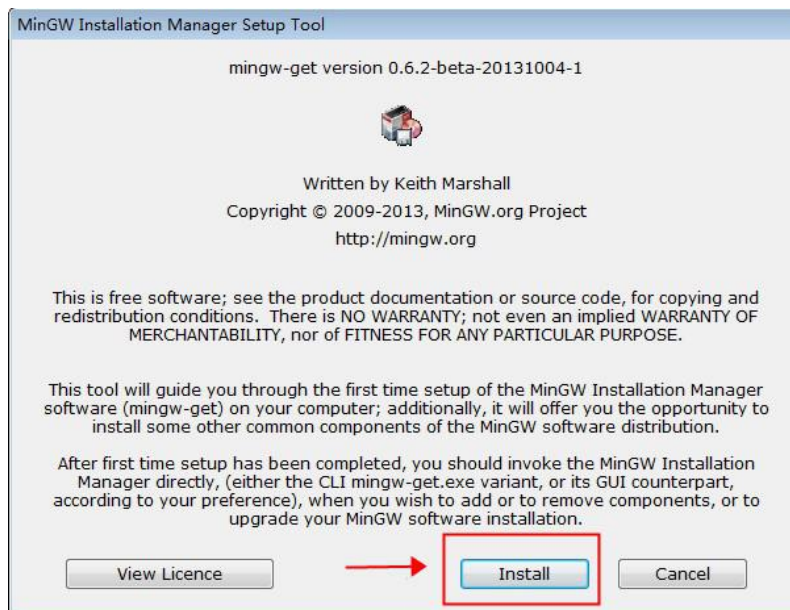
6. 安装完成, 点击完成退出界面, 如下图所示



28.2.2 mingw 平台的安装

28.2.2.1 mingw 平台的安装步骤:

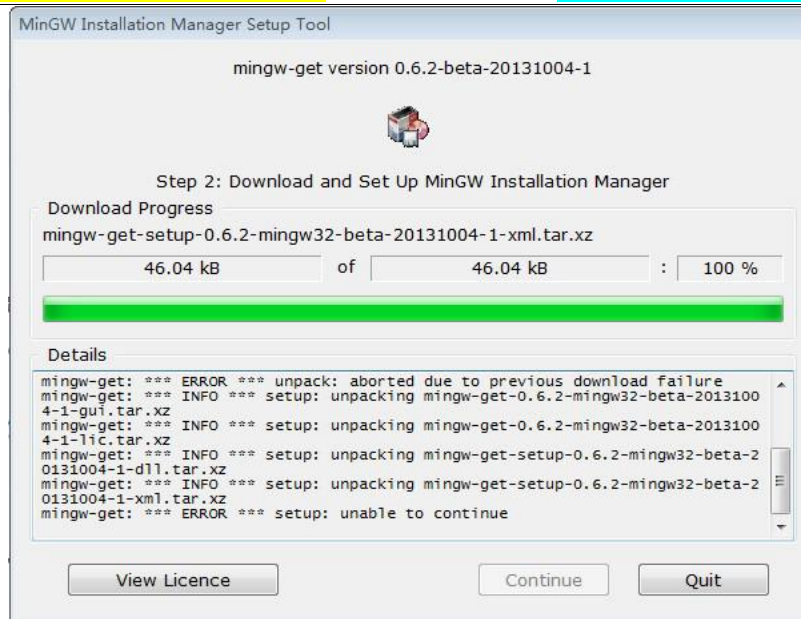
1. 双击 mingw-get-setup.exe 图标, 点击 Install 进行安装, 如下图所示:



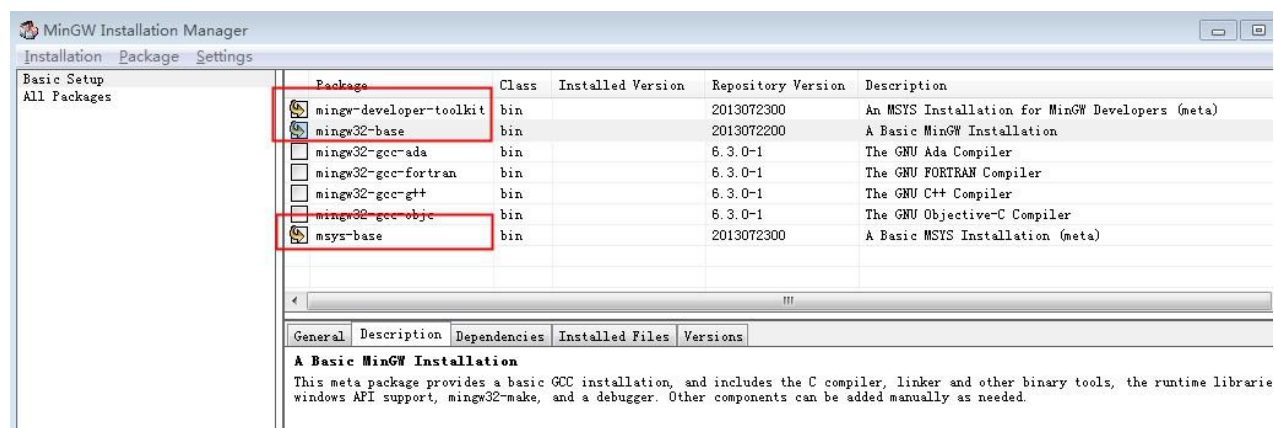
2. 选择安装路径，如下图所示，选择默认路径不要改变，然后点击 Continue 继续：



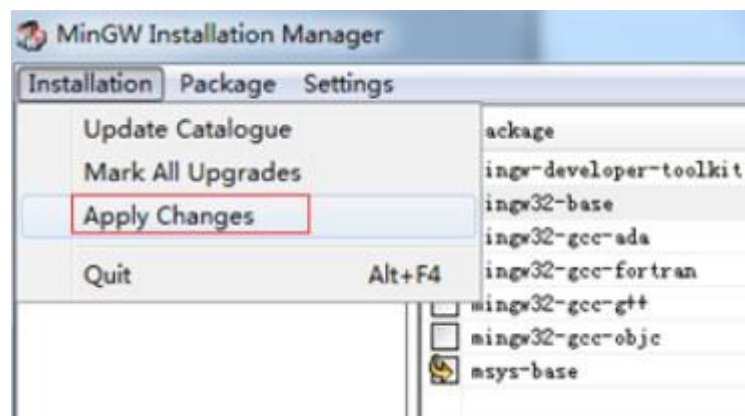
3. 点击继续后，会出现安装进度条，如下图所示，等待安装完成：

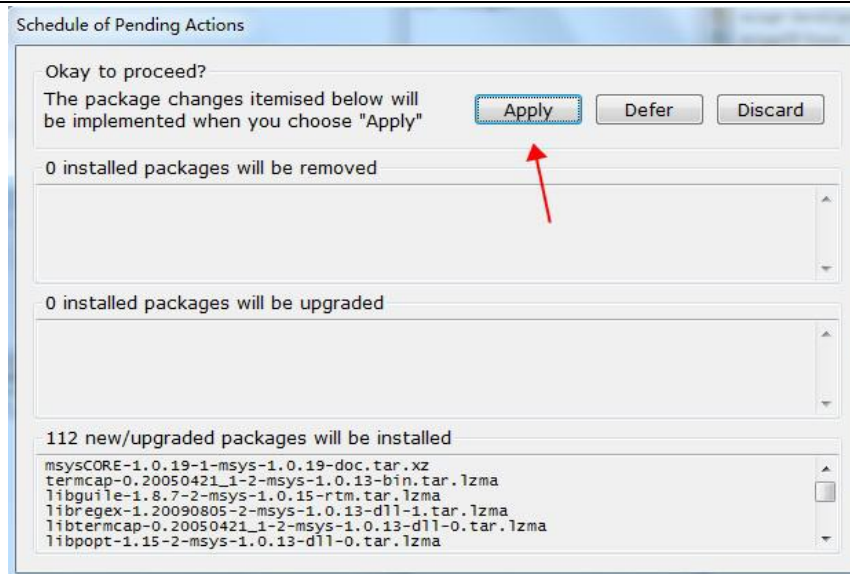


4. 安装好后会弹出 Package 包安装界面，这里需要选择三个项目，如下图所示，点击选项后选择“make for Installation”



5. 选择选项后，点击“Installation”-->“Apply Change”选项，选择安装前面选择的安装选项，如图所示：





28.2.2.2 环境变量的配置

安装好 MinGW 后, 需要在系统的环境变量的 path 中添加路径, 具体步骤如下所示:

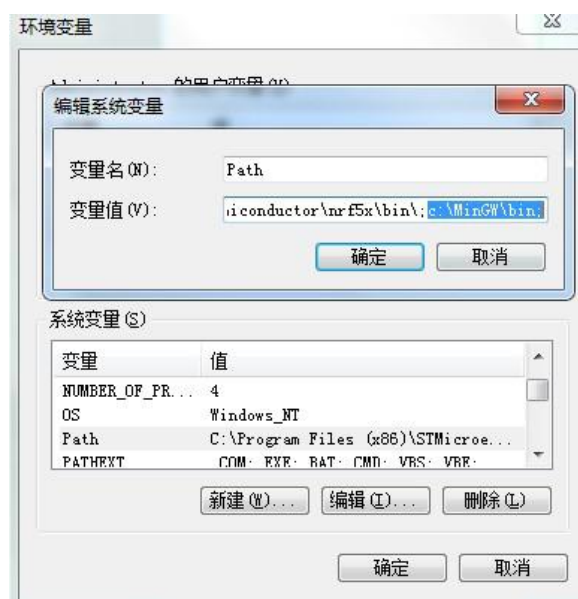
1: 鼠标放到“我的计算机”上, 点击右键, 选择属性, 弹出如下图编辑界面, 然后选择高级系统设置:



2. 选择高级系统设置后, 弹出如下图系统属性框, 点击下方的环境变量按钮:



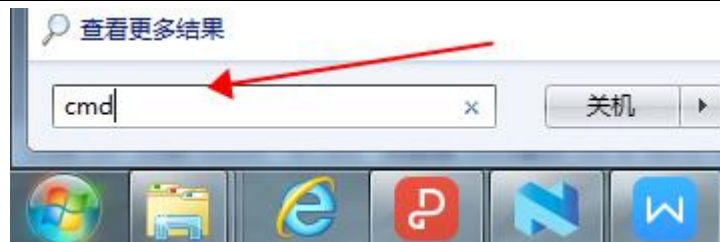
3. 在系统变量中, 找到 Path 变量, 点击打开, 弹出编辑系统变量框, 在框中添加 MinGw 的路径, 输入 `c:\MinGW\bin`; 如下图所示, 注意一定要用 ; 号结束。



28.2.2.3 环境安装验证及常见错误解决

有的客户安装后, 软件无法使用, 也就是缺少文件之类, 所有安装后需要验证是否安装成功。

1. 打开电脑开始, 在命令框内输入 `cmd` 命令, 进入 dos, 如下图所示:



2. 在命令行内输入“gcc -V”指令，查询 GCC 版本，注意 V 一定要大小，如下图所示，如果出现版本说明，表示安装成功：

```
管理员: C:\windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=c:/mingw/bin/./libexec/gcc/mingw32/6.3.0/lto-wrapper.exe
Target: mingw32
Configured with: ../src/gcc-6.3.0/configure --build=x86_64-pc-linux-gnu --host=m
ingw32 --target=mingw32 --with-gmp=/mingw --with-mpfr --with-mpc=/mingw --with-i
sl=/mingw --prefix=/mingw --disable-win32-registry --with-arch=i586 --with-tune=
generic --enable-languages=c,c++,objc,obj-c++,fortran,ada --with-pkgversion='Min
GW.org GCC-6.3.0-1' --enable-static --enable-shared --enable-threads --with-dwar
f2 --disable-sjlj-exceptions --enable-version-specific-runtime-libs --with-libic
onv-prefix=/mingw --with-libintl-prefix=/mingw --enable-libstdcxx-debug --enable
-libgomp --disable-libvtv --enable-nls
Thread model: win32
gcc version 6.3.0 (MinGW.org GCC-6.3.0-1)

C:\Users\Administrator>
```

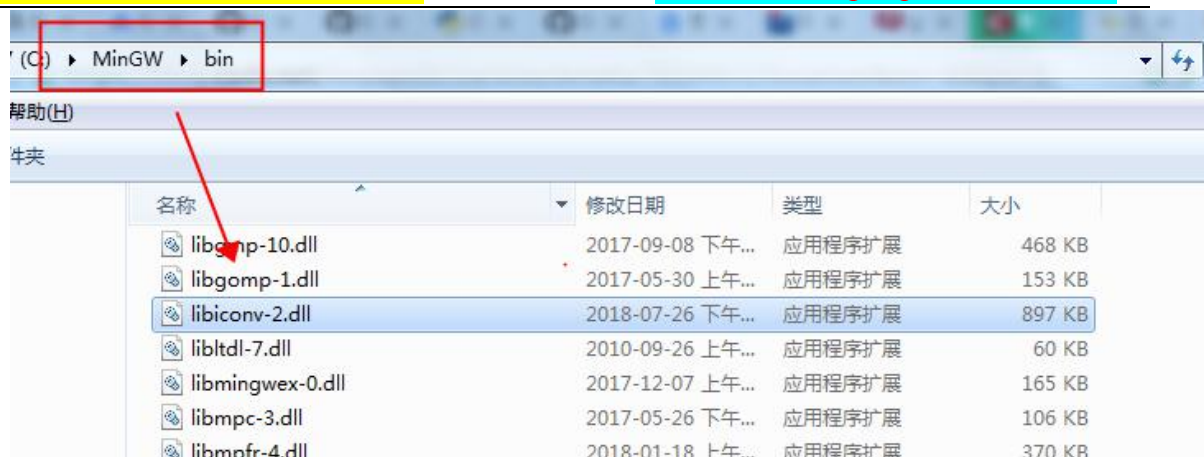
3. 如果输入指令弹出如图下错误，提示缺少文件：



4: 重新单独下载一个 dll 文件，

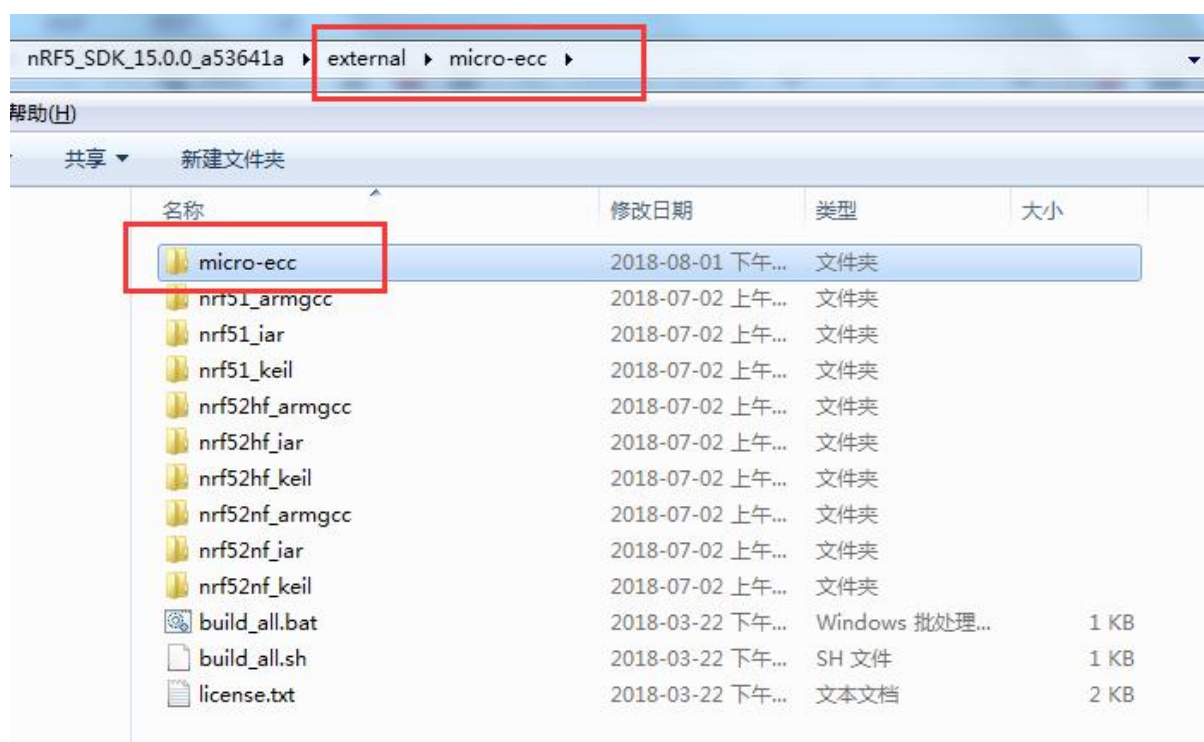
下载一个 dll 文件地址: <http://dl.pconline.com.cn/download/401180.html>

复制到 MinGW/bin 文件夹内，如下图所示：



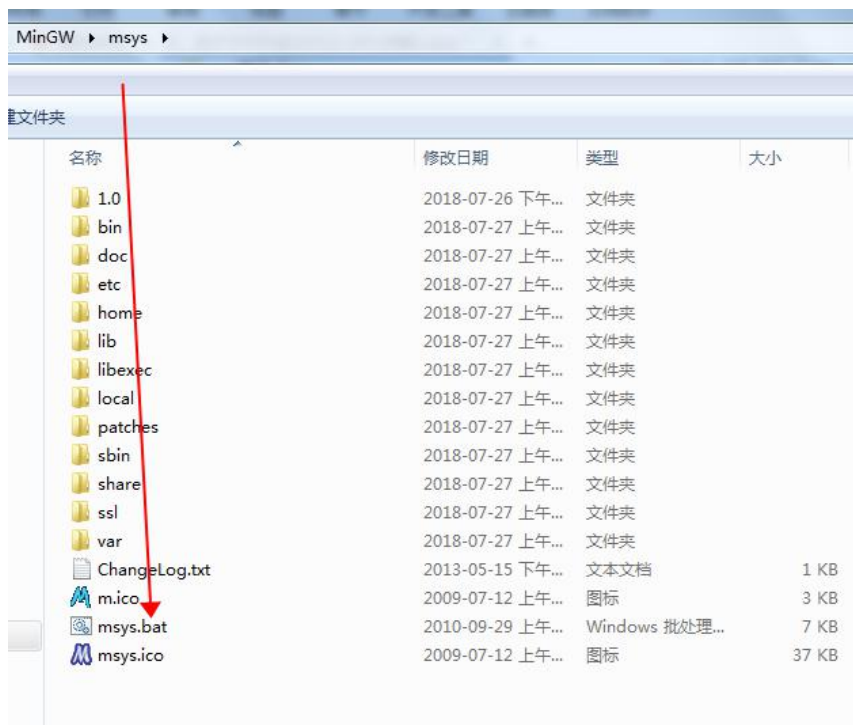
28.2.3 micro-ecc-master 源码的添加:

把下载的 micro-ecc-master.zip 解压, 解压后拷贝到 nRF5_SDK_15.0.0_a53641a /external/micro-ecc 文件中, 重新名为 micro-ecc, 如下图所示:



28.2.4 micro_ecc_lib_nrf52.lib 文件的生成:

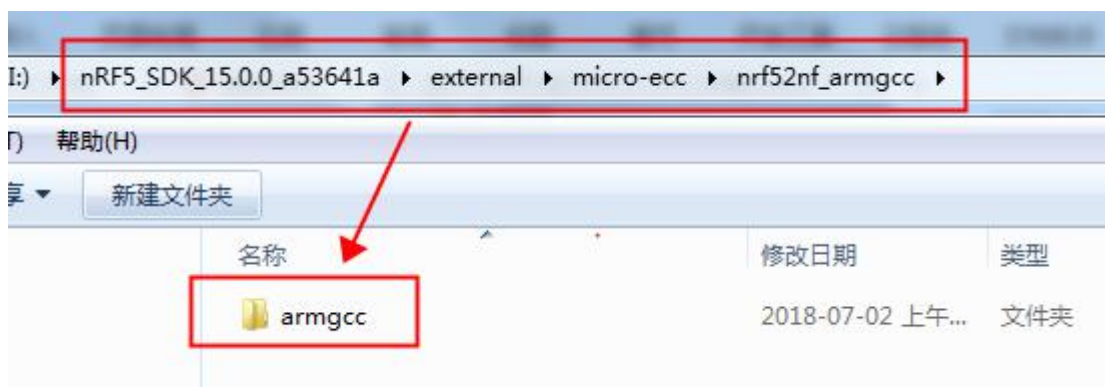
1: 首先打开 MinGW/msys 文件中的批处理文件 msys.bat, 点击打开, 如下图所示:



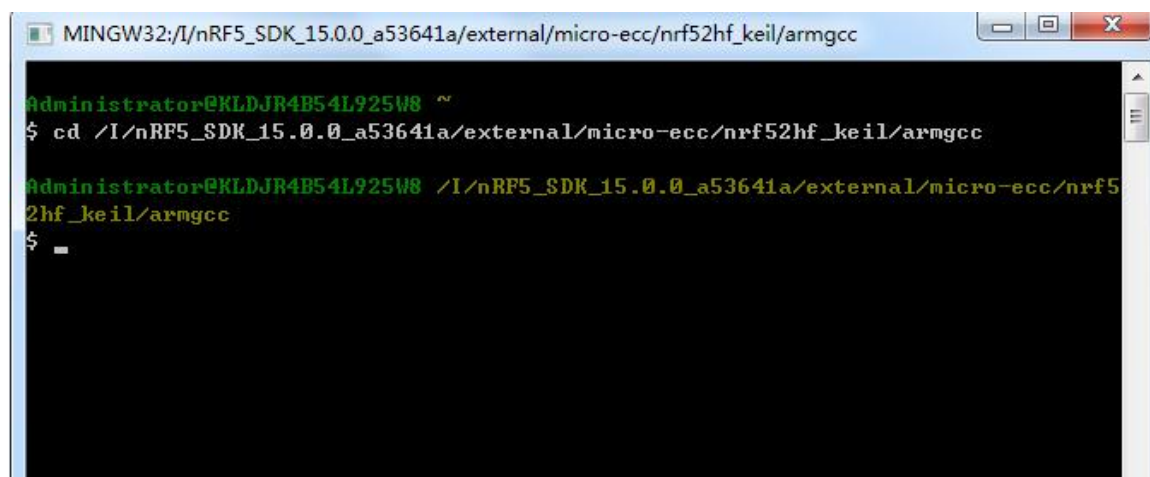
2. 弹出一个批处理文件框，在框内输入如下命令：

```
cd /I/nRF5_SDK_15.0.0_a53641a/external/micro-ecc/nrf52hf_keil/armgcc
```

后面 I 开始这段，实际上是指的 armgcc 的算法路径。也没有直接把文件包 armgcc 拖入到批处理框中：

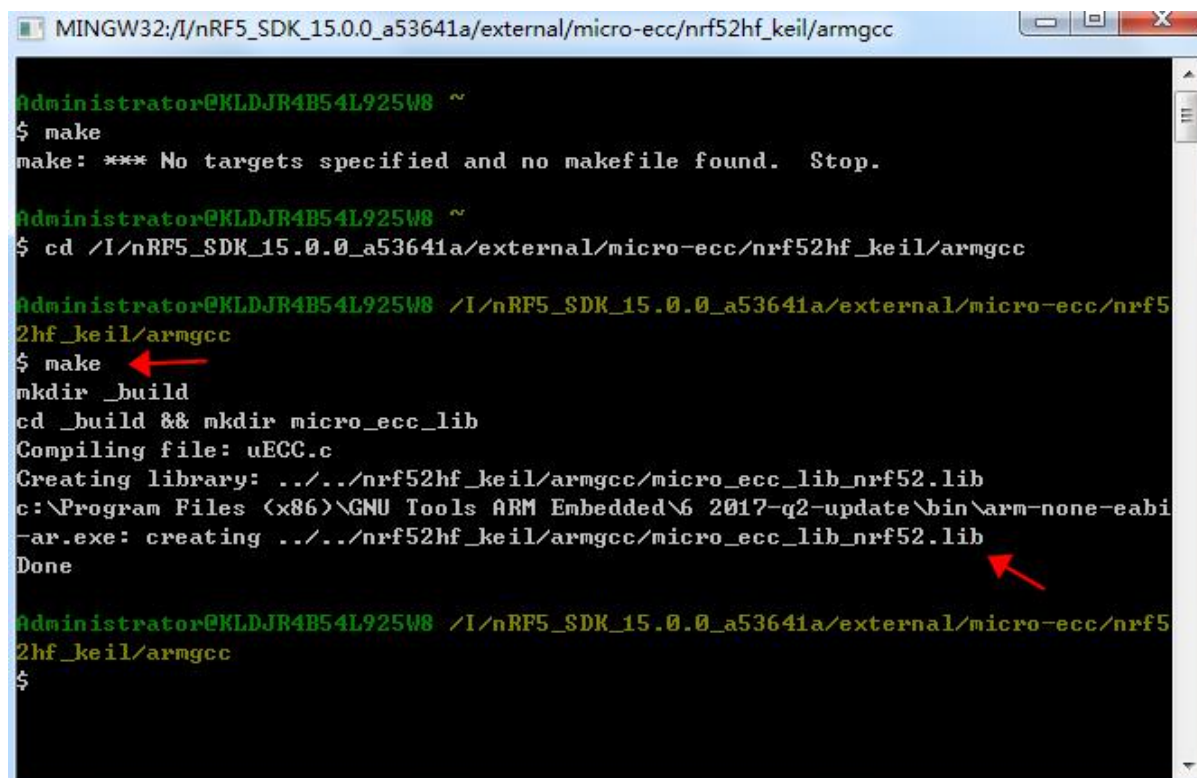


上图文件夹拖入批处理框中，然后把命令修改成指定格式，如下图所示，点击回车：



```
MINGW32:/I/nRF5_SDK_15.0.0_a53641a/external/micro-ecc/nrf52hf_keil/armgcc
Administrator@KLDJR4B54L925W8 ~
$ cd /I/nRF5_SDK_15.0.0_a53641a/external/micro-ecc/nrf52hf_keil/armgcc
Administrator@KLDJR4B54L925W8 /I/nRF5_SDK_15.0.0_a53641a/external/micro-ecc/nrf52hf_keil/armgcc
$
```

3. 找到算法后, 继续输入指令 make, 生成 lib 文件, 如下图所示:




```
MINGW32:/I/nRF5_SDK_15.0.0_a53641a/external/micro-ecc/nrf52hf_keil/armgcc
Administrator@KLDJR4B54L925W8 ~
$ make
make: *** No targets specified and no makefile found. Stop.

Administrator@KLDJR4B54L925W8 ~
$ cd /I/nRF5_SDK_15.0.0_a53641a/external/micro-ecc/nrf52hf_keil/armgcc
Administrator@KLDJR4B54L925W8 /I/nRF5_SDK_15.0.0_a53641a/external/micro-ecc/nrf52hf_keil/armgcc
$ make
mkdir _build
cd _build && mkdir micro_ecc_lib
Compiling file: uECC.c
Creating library: ../../nrf52hf_keil/armgcc/micro_ecc_lib_nrf52.lib
c:\Program Files (x86)\GNU Tools ARM Embedded\6 2017-q2-update\bin\arm-none-eabi-ar.exe: creating ../../nrf52hf_keil/armgcc/micro_ecc_lib_nrf52.lib
Done
```

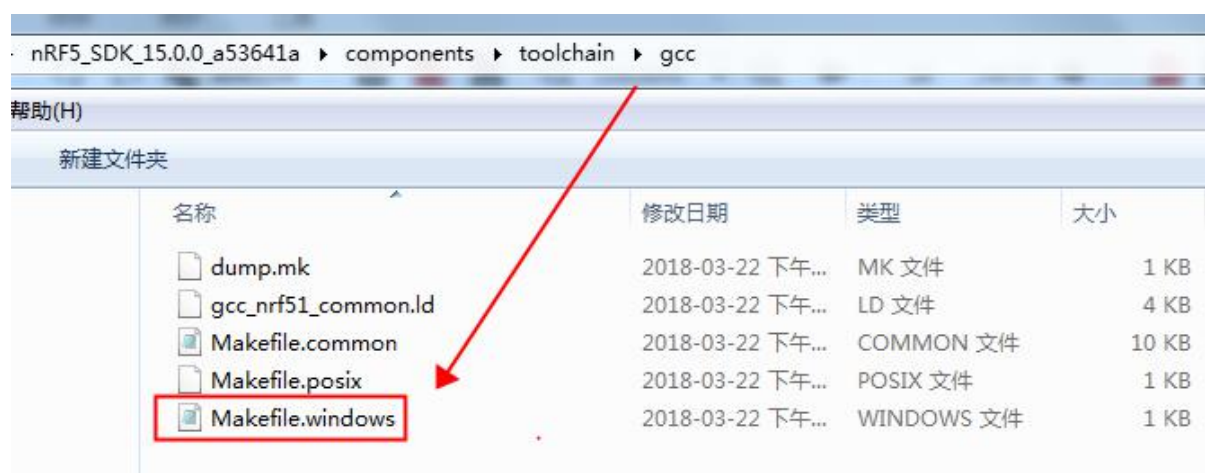
3. 常见错误解决:

当 make 后出现未发现文件, 如下图所示:

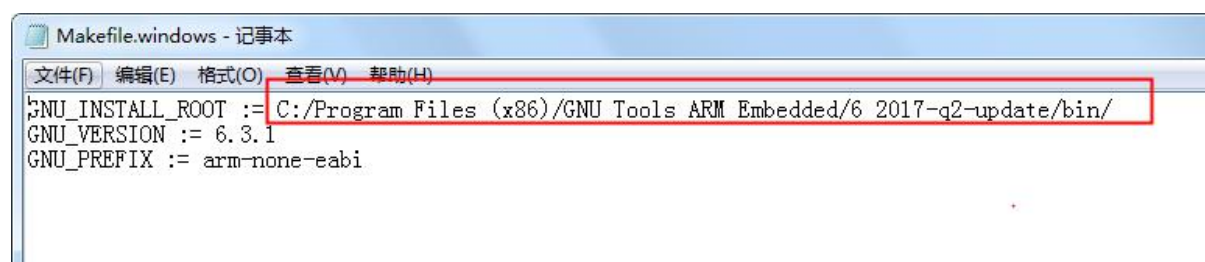


```
$ make
/bin/sh: C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc: No such file or directory
Cannot find: "C:/Program Files (x86)/GNU Tools ARM Embedded/4.9 2015q3/bin/arm-none-eabi-gcc".
Please set values in: "/D/temp/DFU123/nRF5/components/toolchain/gcc/Makefile.windows"
```

也就是没有发现 GNU Tools 的安装路径,也就是前面安装的 gcc-arm-none-eabi 路径和 sdk 里的设置路径和版本不同。这时打开 nRF5_SDK_15.0.0_a53641a/components/toolchain/gcc 文件夹,找到 Makefile.windows 文件,用记事本打开,如下图所示:

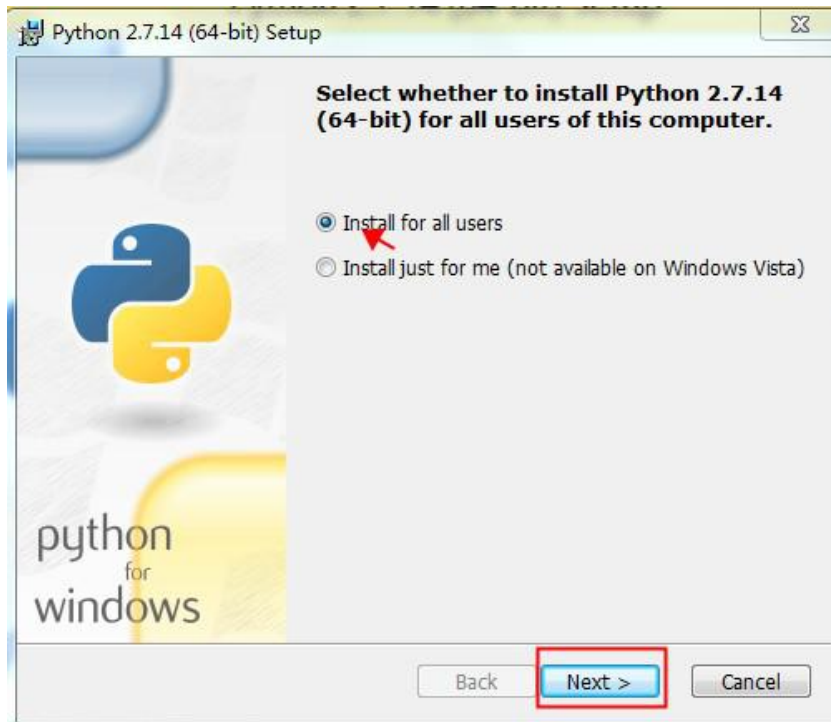


打开文件后显示了一个安装路径和版本号,如果这个与实际的安装路径和版本号不符合,请修改第一步 gcc-arm-none-eabi 软件安装的路径和软件版本:

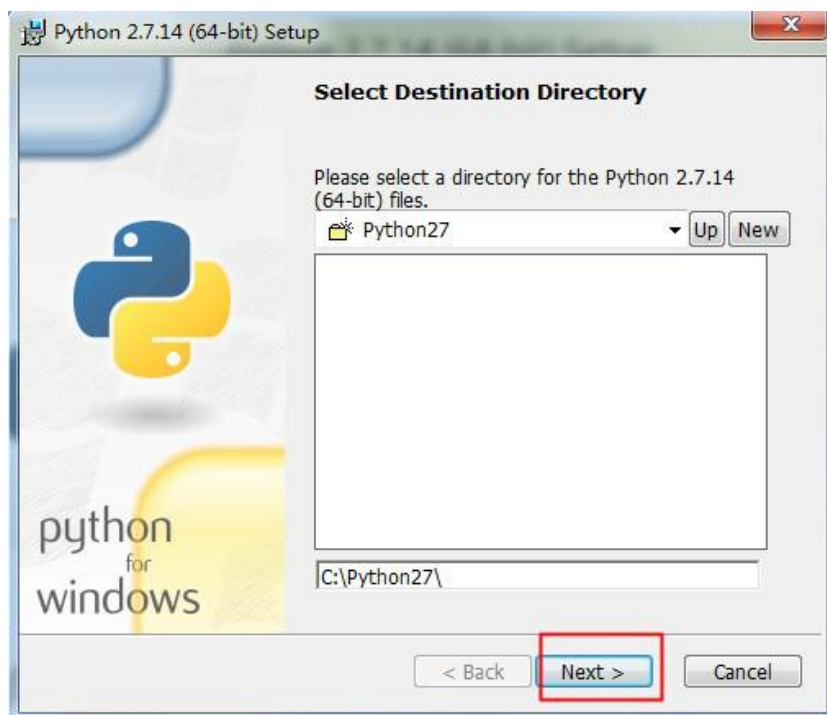


28.2.5 python 软件的安装

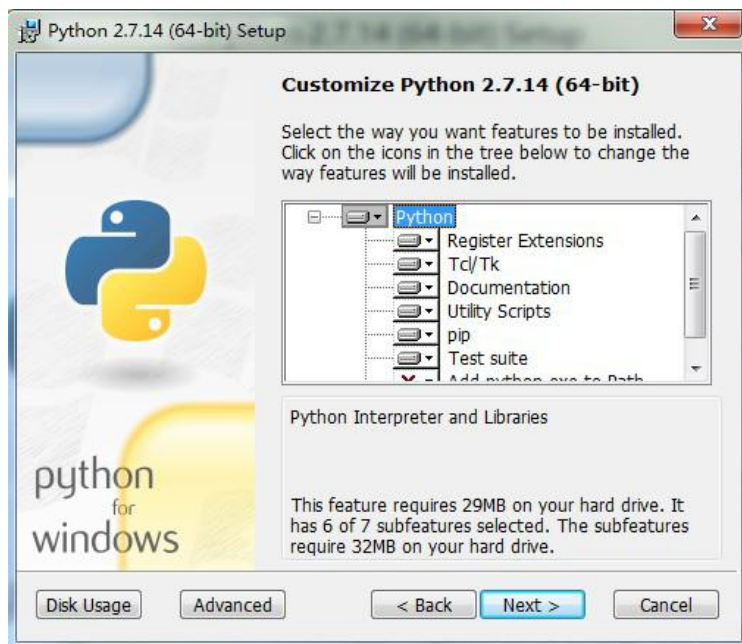
1. 双击 python-2.7.14.amd64.msi 图标 (32 位电脑安装 python-2.7.14.msi), 点击 Install 进行安装, 如下图所示:



2. 选择安装路径，如下图所示，选择默认路径不要改变，然后点击 next 继续：



3. 选择安装的插件，如下图所示，选择默认插件不要改变，然后点击 Continue 继续：



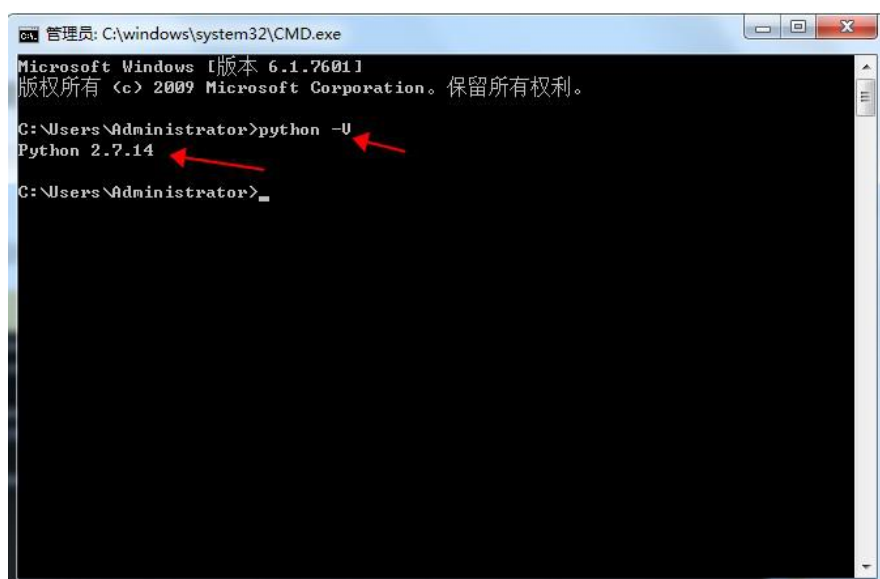
4. 安装完成，点击完成退出界面，如下图所示



5. 在系统变量中，找到 Path 变量，点击打开，弹出编辑系统变量框，在框中添加 MinGw 的路径，输入;c:\Python27;c:\Python27\Scripts; 如下图所示，注意一定要用;号结束。

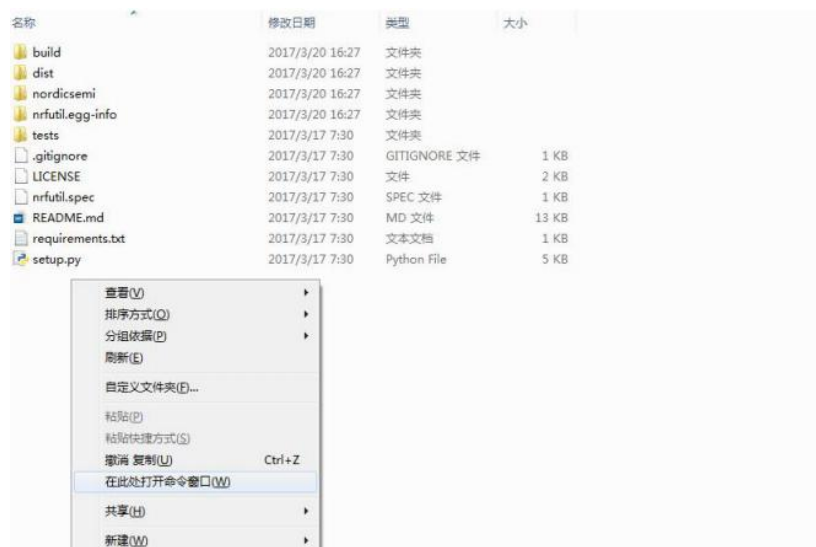


6. 打开电脑开始，在命令框内输入 cmd 命令，进入 dos，在命令行内输入“python -V”指令，查询 python 版本，注意 V 一定要大小，如下图所示，如果出现版本说明，表示安装成功：

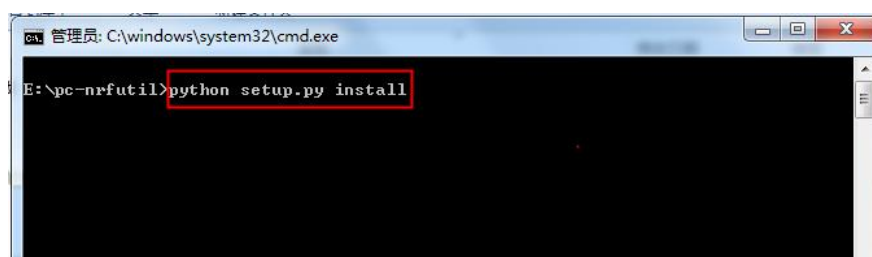


28.2.6 pc-nrfutil 的安装与秘钥的生成

1. 把 pc-nrfutil 压缩包解压。如图所示打开有 setup.py 文件的目录，在此处打开 dos 命令(shift+右键打开)。



2. 输入 `python setup.py install` 命令。这步骤需要有网络的时候运行, 安装可能需要等待几分钟时间。



3. 在 DOS 命令下输入 `nrfutil` 可以获取更多关于 `nrfutil` 的信息, 则表示 `nrfutil` 安装成功:



4. 在 I 盘新建一个文件夹, 命名为 `key`, 在 `cmd` 命令中输入以下内容: `nrfutil.exe keys generate I:\key\private.key` 指令, 如图会在 I 盘的 `key` 文件夹中会生成 `private.key` 文件, 如下图所示:

```
管理员: C:\windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>nrfutil
Usage: nrfutil [OPTIONS] COMMAND [ARGS]...

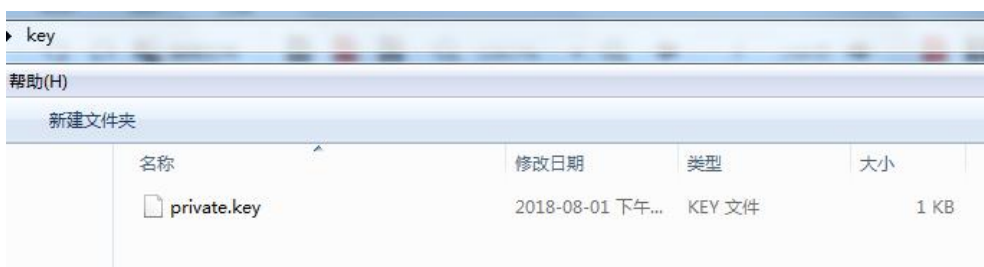
Options:
  -v, --verbose          Show verbose information.
  -o, --output <filename> Log output to file
  --help                Show this message and exit.

Commands:
  dfu      Perform a Device Firmware Update over, BLE, Thread, or serial
           transport given a DFU package <zip file>.
  keys     Generate and display private and public keys.
  pkg      Display or generate a DFU package <zip file>.
  settings Generate and display Bootloader DFU settings.
  version  Display nrfutil version.

C:\Users\Administrator>nrfutil.exe keys generate I:\key\private.key
Generated private key and stored it in: I:\key\private.key

C:\Users\Administrator>
```

打卡 key 文件夹，会发现生成了一个 private.key 文件，如下图所示：

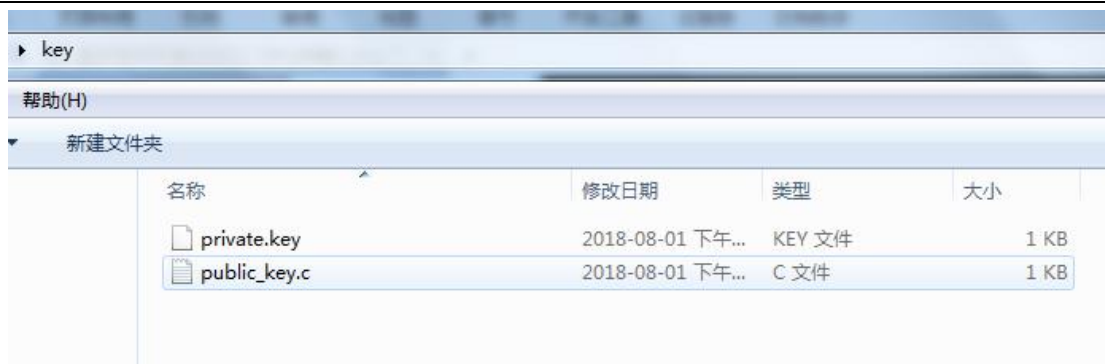


5. 然后使用这个文件生成一个 C 文件，这里的私钥 (private.key) 和公钥 (public_key.c)。大家务必要保存好私钥 private.key，以后每个新 image 要升级时，都会先通过这个私钥对它进行签名，一旦 private.key 丢失或者被暴露，DFU 将无法进行或者变得不安全。

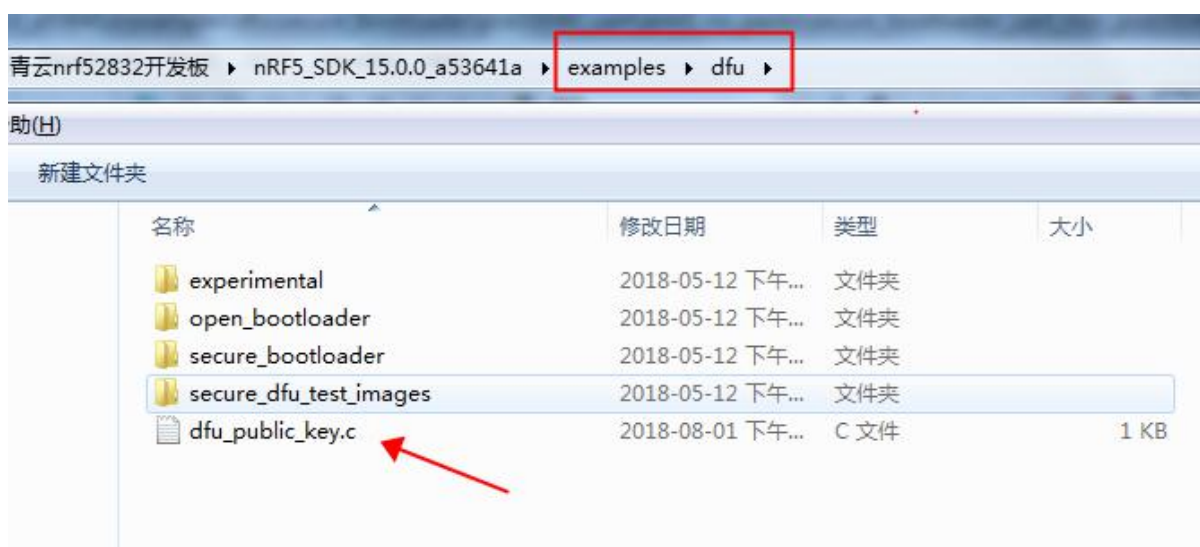
生成公钥的指令如下：nrfutil keys display --key pk --format code I:\key\private.key --out_file I:\key\public_key.c

```
C:\Users\Administrator>nrfutil keys display --key pk --format code I:\key\privat
e.key --out_file I:\key\public_key.c
C:\Users\Administrator>
```

打卡 key 文件夹，会发现生成了一个 public_key.c 文件，如下图所示，这时候私钥 (private.key) 和公钥 (public_key.c) 都生成成功了：



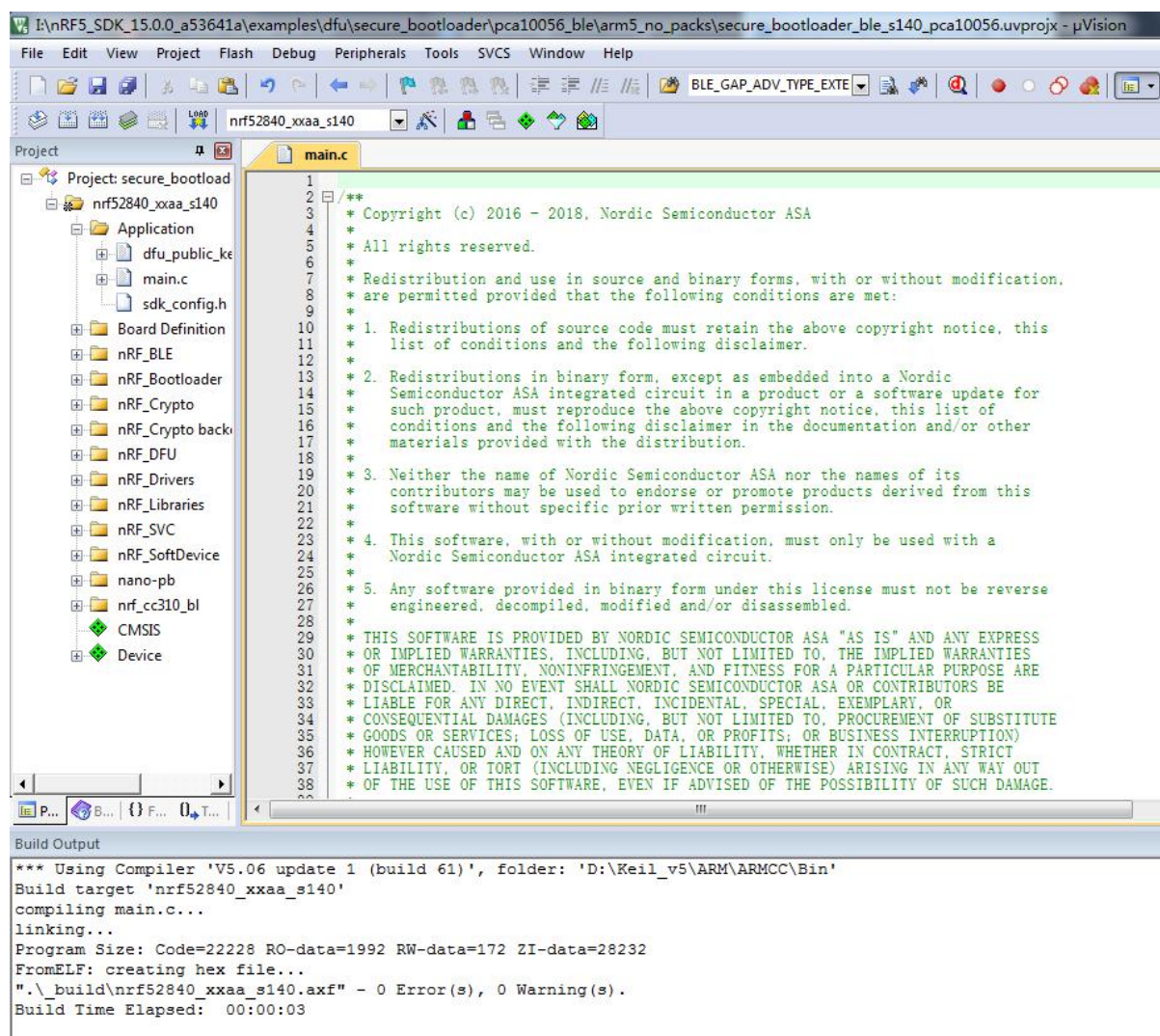
6. 将公钥 `public_key.c` 这个文件改名为 `dfu_public_key.c`，将该文件替换目录 `\examples\dfu\` 下的 `dfu_public_key.c` 文件，如下图所示：



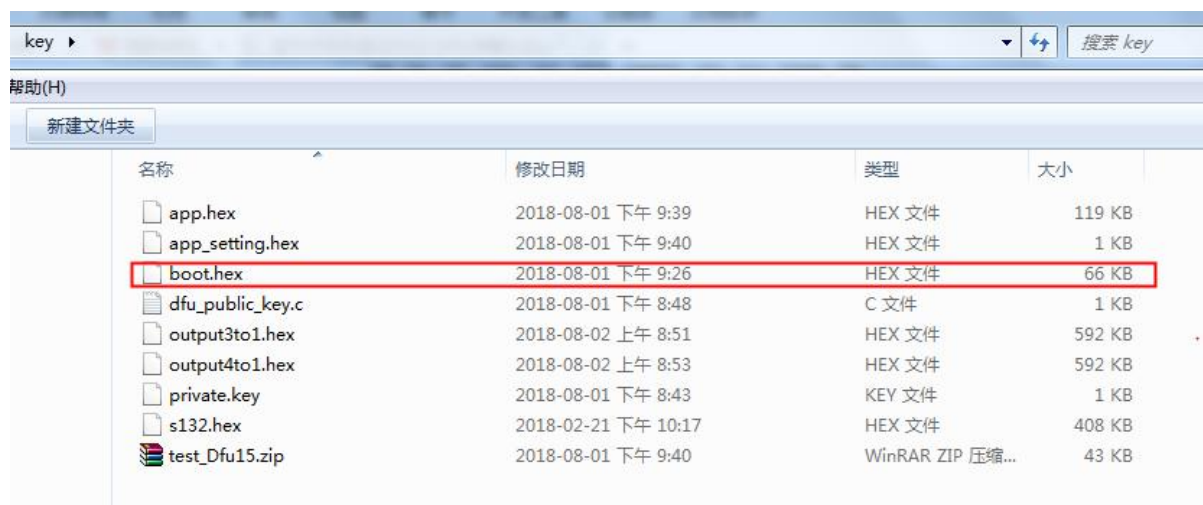
28.2.7 boot 工程的 hex 和应用的 hex 生成

28.2.7.1 boot 工程的 hex 生成：

1. 打开工程目录：`nRF5_SDK_15.0.0_a53641a\examples\dfu\secure_bootloader\pca10056_ble\arm5_no_packs` 下的 keil 工程。如果之前的工作操作正确的话，编译该工程会提示 0 错误，如下图所示，并且生成默认名字为 `nrf52840_xxaa_s132.hex` 的文件：



2. 把 nrf52840_xxaa_s132.hex 的文件改名为 boot.hex, 放到之前新建的 key 文件夹中, 如下图所示:



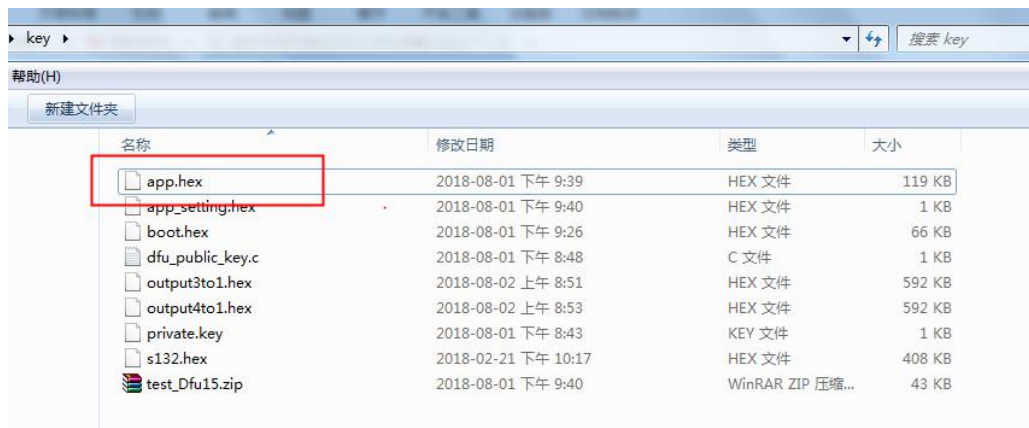
28.2.7.2 应用工程的 hex 生成:

1. 打开工程目录:

nRF5_SDK_15.0.0_a53641a\examples\ble_peripheral\ble_app_buttonless_dfu

\pca10056\S132\arm5_no_packs 下的 keil 工程。如果之前的工作操作正确的话, 编译该工程会提示 0 错误, 如下图所示, 并且生成默认名字为 nrf52832_xxaa.hex 的文件。

2. 把 nrf52840_xxaa.hex 的文件改名为 app.hex, 放到之前新建的 key 文件夹中, 如下图所示:



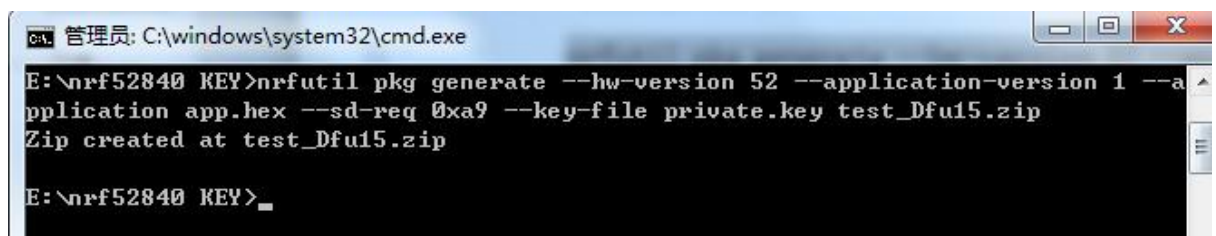
28.2.7.3 应用工程的 zip 生成:

SDK9 版本后, DFU 紧支持 ZIP 升级, 因此, 升级的固件必须做成 ZIP 格式, 下面主要通过 DOS 输入 nrfutil 来实现。

打开刚才存放 app.hex 文件的 key 文件夹, 在此处打开 dos 命令(shift+右键打开)。然后输入命令:

```
nrfutil pkg generate --hw-version 52 --application-version 1 --application app.hex --sd-req 0xa9 --key-file private.key test_Dfu15.zip
```

如下图所示:



对这指令进行简单说明:

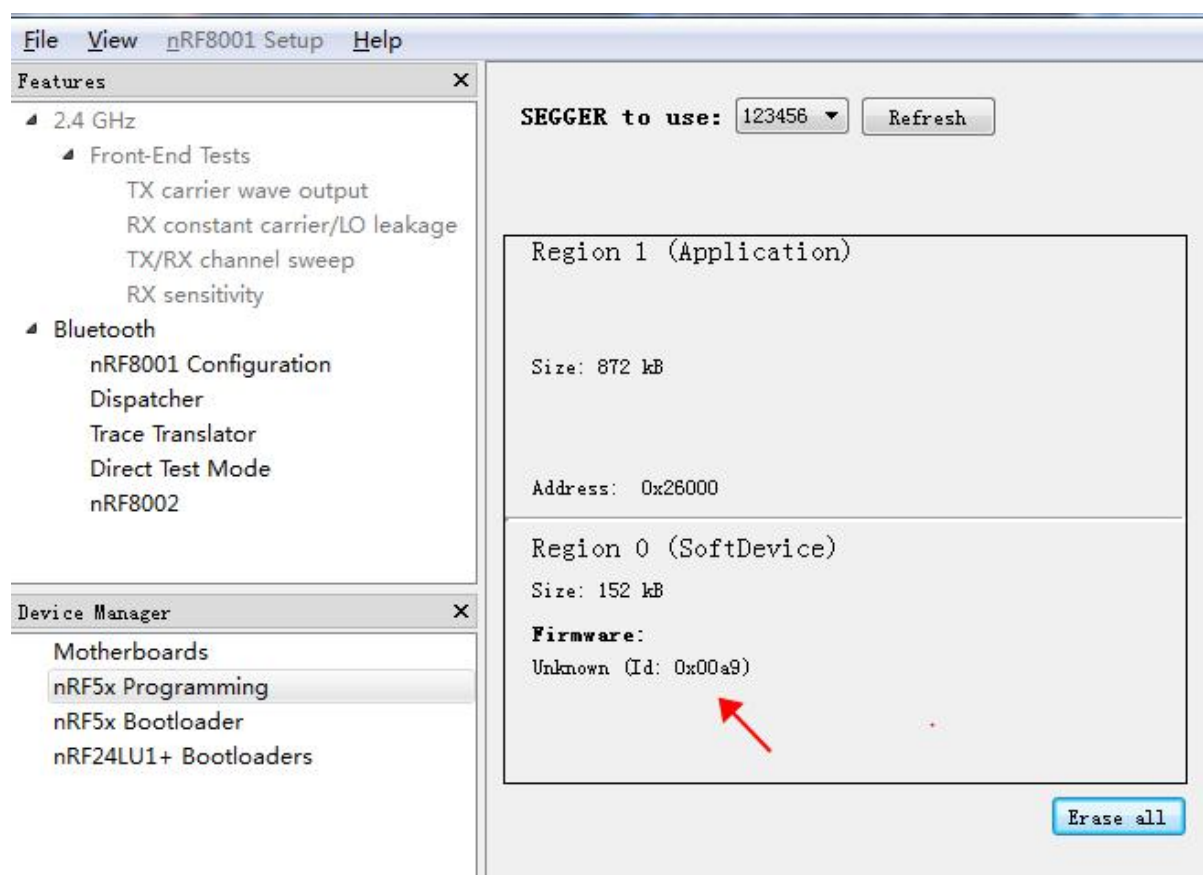
hw-version 52: 硬件版本。如果是 nrf51822, 设置为 51, 如果是 nrf52832 或者 nrf52840 设置为 52。你也可以通过修改代码里的 sdk_config.h 文件中的定义 #define NRF_DFU_HW_VERSION your_hw_number 定义自己的硬件版本号。

application-version 1 : 应用版本号, 这个可以自己设置, 用户标志第几个版本的固件。

req 0xa8: 协议栈版本 ID 号, 官方给出来各个协议栈版本的 ID 号如下:

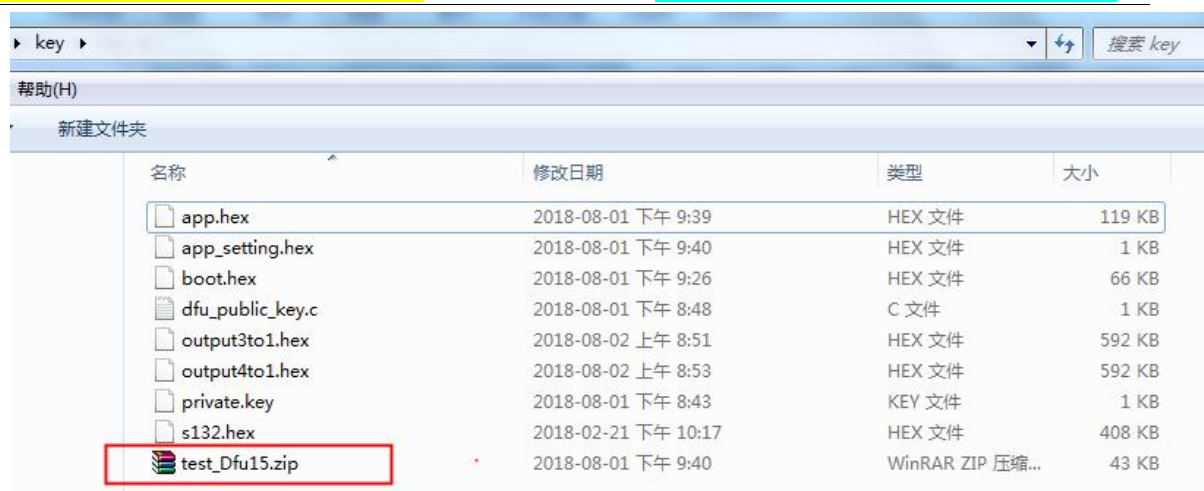
- .S132_nrf52_2.0.0:0x81
- .S132_nrf52_2.0.1:0x87
- .S132_nrf52_3.0.0:0x8c
- .S132_nrf52_4.0.2:0x98
- .S132_nrf52_5.0.0:0x9D
- .S132_nrf52_6.0.0:0xA8
- .S140_nrf52_6.0.0:0xA9

也可以通过 nrfgo 软件进行观察, 如下图所示:



test_Dful5.zip: 输出的应用 zip 文件名称。

输入命令后, 回车, 会在 key 文件夹内生成 zip 应为文件:



28.3 程序烧录与升级

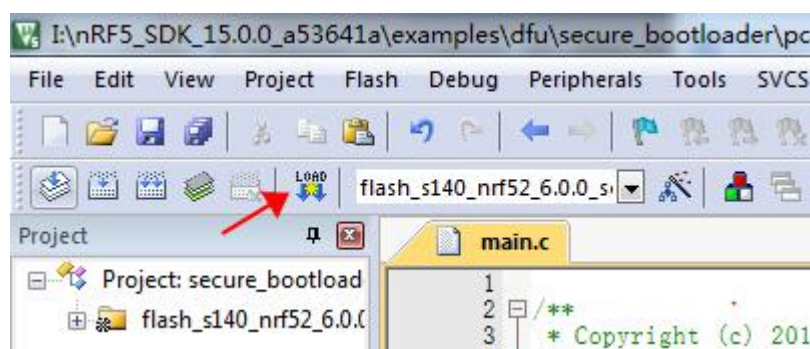
28.3.1 程序的烧录

因为 nrf52840 的 boot 是不支持 nrfgo 进行烧写的, 因此可以采用两种方式, 一种方式是 KEIL 直接下载, 另外一种方式是 nrfjprog 命令行直接烧写。

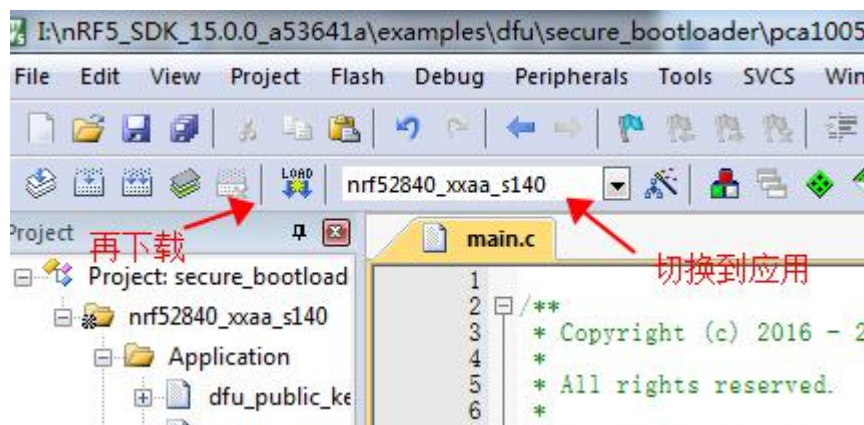
方法一: 打开 nRF5_SDK_15.0.0_a53641a\examples\dfu\secure_bootloader\pca10056_ble\arm5_no_packs 下的 keil 工程, 在工程上选择框, 首先选择协议栈:



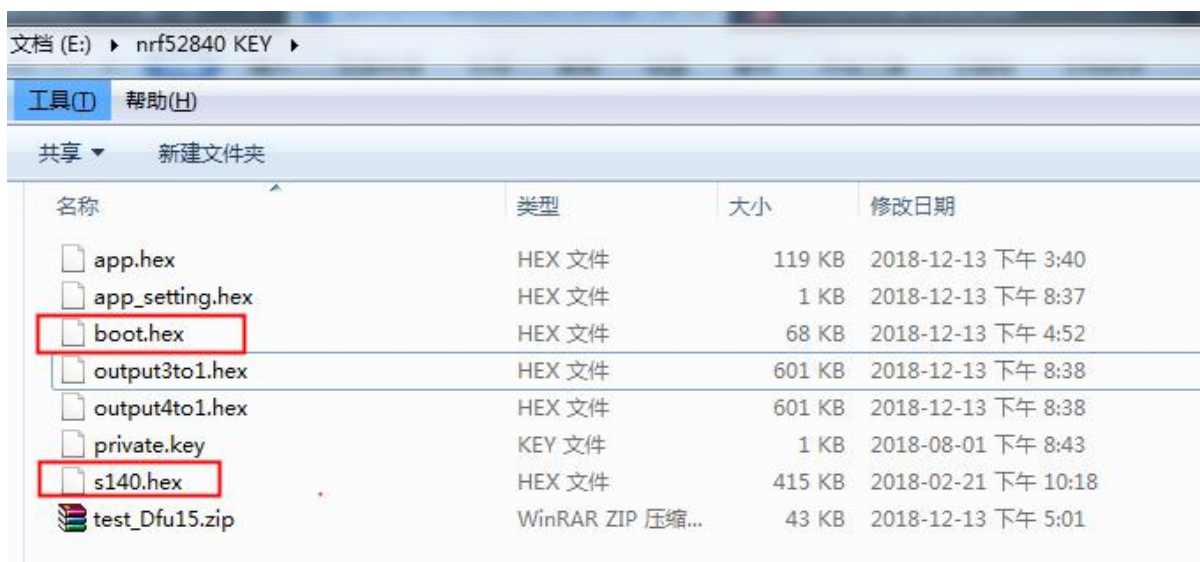
选择后, 点击 downloader 下载:



再切换到应用, , 然后点击 downloader 下载, 如下图所示:



方法二: nrfjprog 命令行烧写, 操作如下所示, 首先把协议栈改成 S140.hex, 拷贝到 boot.hex 的同一个文件夹, 如下图所示:



在此处打开 dos 命令(shift+右键打开)。然后输入命令:

nrfjprog -f NRF52 --program s140.hex 烧写协议栈。

nrfjprog -f NRF52 --program boot.hex 烧写 boot 引导程序。

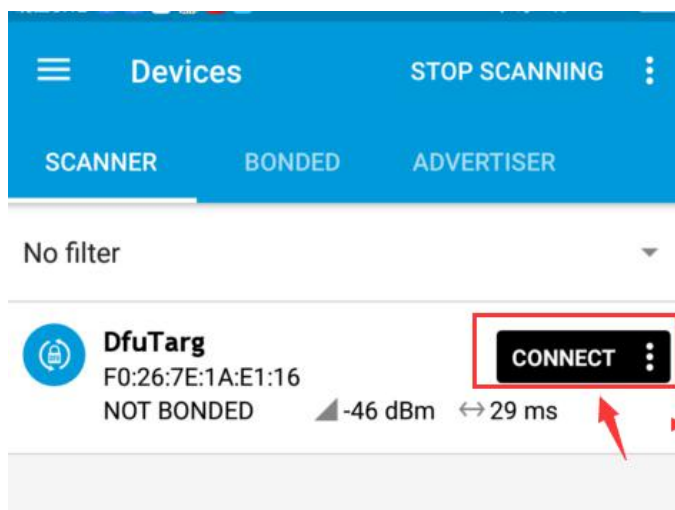
如下图所示:

```
管理员: C:\windows\system32\cmd.exe
E:\nrf52840 KEY>nrfjprog -f NRF52 --program s140.hex
Parsing hex file.
Reading flash area to program to guarantee it is erased.
Checking that the area to write is not protected.
Programing device.

E:\nrf52840 KEY>nrfjprog -f NRF52 --program boot.hex
Parsing hex file.
Reading flash area to program to guarantee it is erased.
Checking that the area to write is not protected.
Programing device.

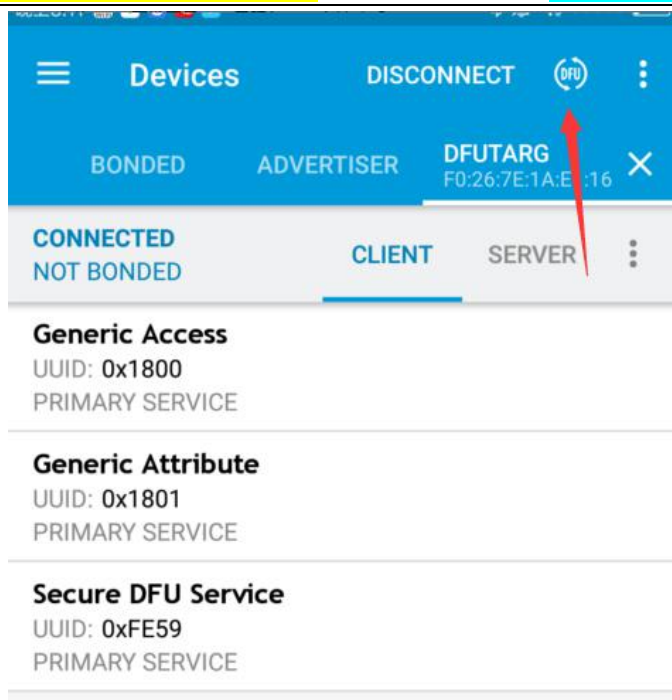
E:\nrf52840 KEY>_
```

烧录完成后, 开发板重新上电, **注意一定要重新上电**。重新上电后开发板上 LED1 和 LED2 灯会被点亮。这时使用 nrf connect app 搜索到广播信号名称为: DfuTarg, 也就是说广播还没有进入应用广播, 如下图所示:

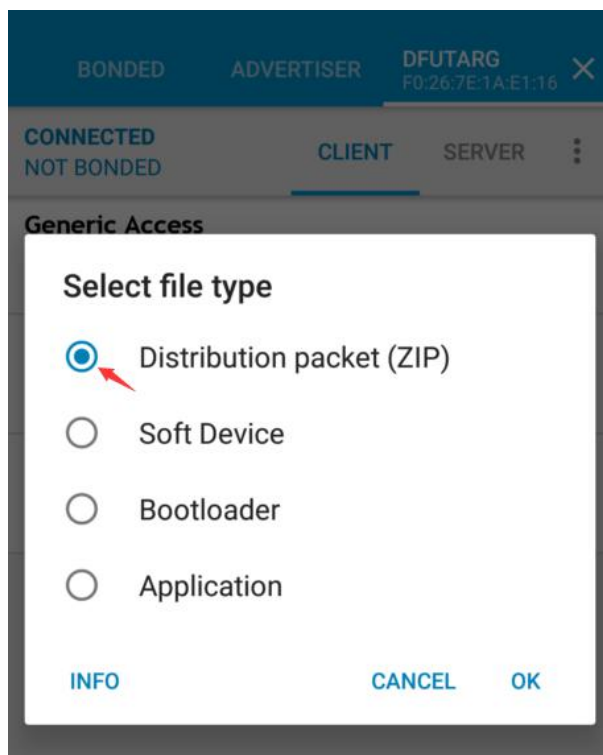


28.3.2 DFU 升级:

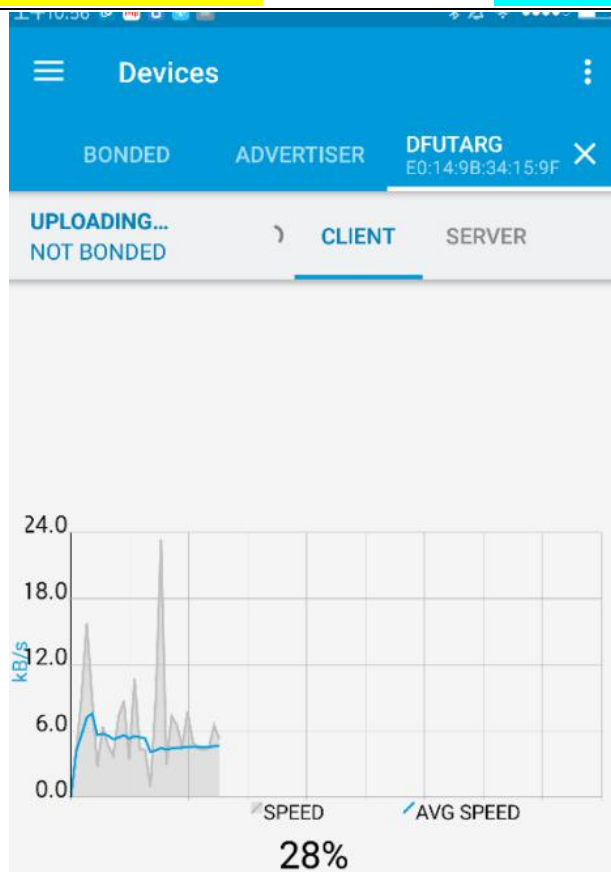
1. 那么我们首先需要升级一次应用, 点击 DfuTarg 广播的连接 connect, 进入到服务, 点击复位右上角的 DFU 图标, 如下图所示:



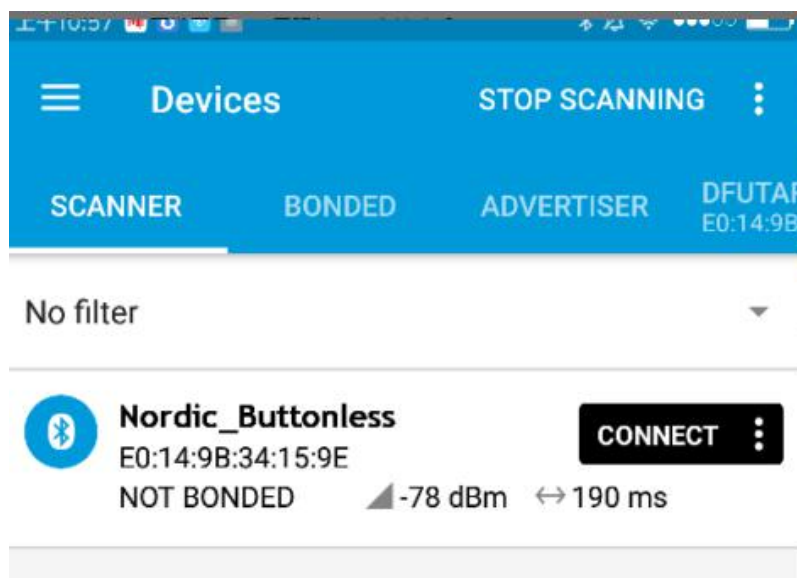
2.弹出来选择文件框，由于 SDK10 之后的版本只支持 ZIP 方式升级，因此这里选择第一项，如下图所示，然后把之前传到手机上的之前制作的应用 ZIP 文件加载上。



3. 加载 zip 文件后，升级开始，下图显示的是升级速度和升级进度：

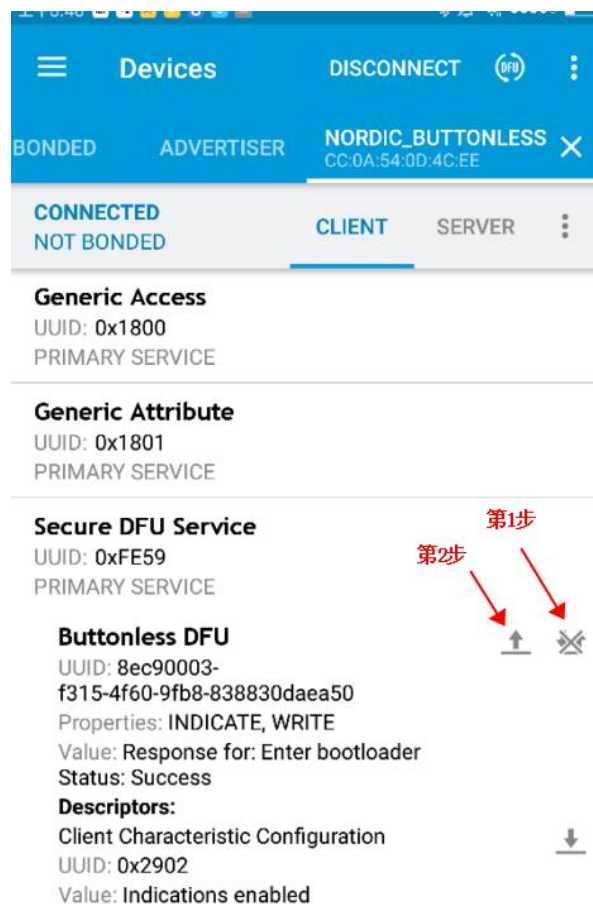


4. 升级完成后, 重新用 nrf connect app 进行扫描, 扫描到的广播名称变成了 Nordic_Buttonless, 表明已经进入应用服务:

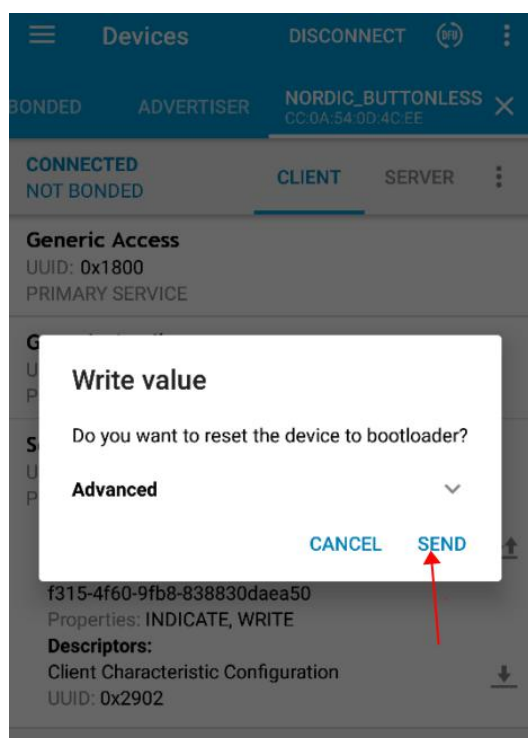


根据前面的分析, SDK12 后的 sdk 版本采用非按键式 (Buttonless) DFU, 一旦 buttonless DFU 例子从 bootloader 跳到了 app 应用后, 后续如何还要升级 DFU, 就不需要通过按键来切换竟然到 boot 了, 只需要通过 app 发送切换指令就可以了, 因为大部分设备上没有按键的, 这样就大大扩展了应用场景。那么如何发送切换指令了?

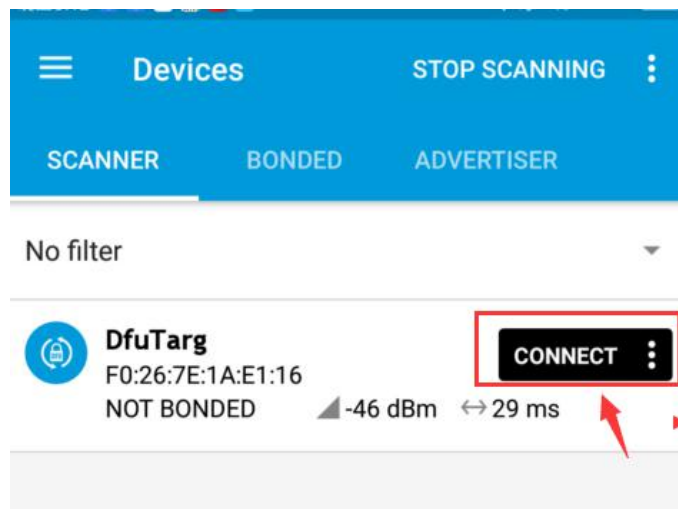
我们连接应用，进入应用后，发现存在一个按键服务，这个按键复位有个指示功能和写功能。首先第一步需要使能指示功能，如下图所示：



第二步：点击写功能，弹出来写命令框，选择发送指令，如下图所示：



发送完切换指令后,退出原来的连接,重新用 app 扫描,会发现广播信号名称有变成了 DfuTarg,这个时候有可以重新开始进行空中升级了:



28.3.3 setting 文件的生成与使用

批量生产的时候,我们往往希望直接下载后就可以运行到应用程序,而不再空中升级一次才运行。如何希望到底这种状态,则需要生成 setting 文件。

还是在刚才我们生成的 key 文件夹中,存放了 app.hex 文件的 key 文件夹,在此处打开 dos 命令(shift+右键打开)。然后输入命令:

```
nrfutil settings generate --family NRF52840 --application app.hex
--application-version 1 --bootloader-version 1 --bl-settings-version 1
app_setting.hex 如下图所示:
```

```
管理员: C:\windows\system32\cmd.exe
I:\key>nrfutil pkg generate --hw-version 52 --application-version 1 --application
n app.hex --sd-req 0xa8 --key-file private.key test_Dfu15.zip
Zip created at test_Dfu15.zip

I:\key>nrfutil settings generate --family NRF52 --application app.hex --applicat
ion-version 1 --bootloader-version 1 --bl-settings-version 1 app_setting.hex

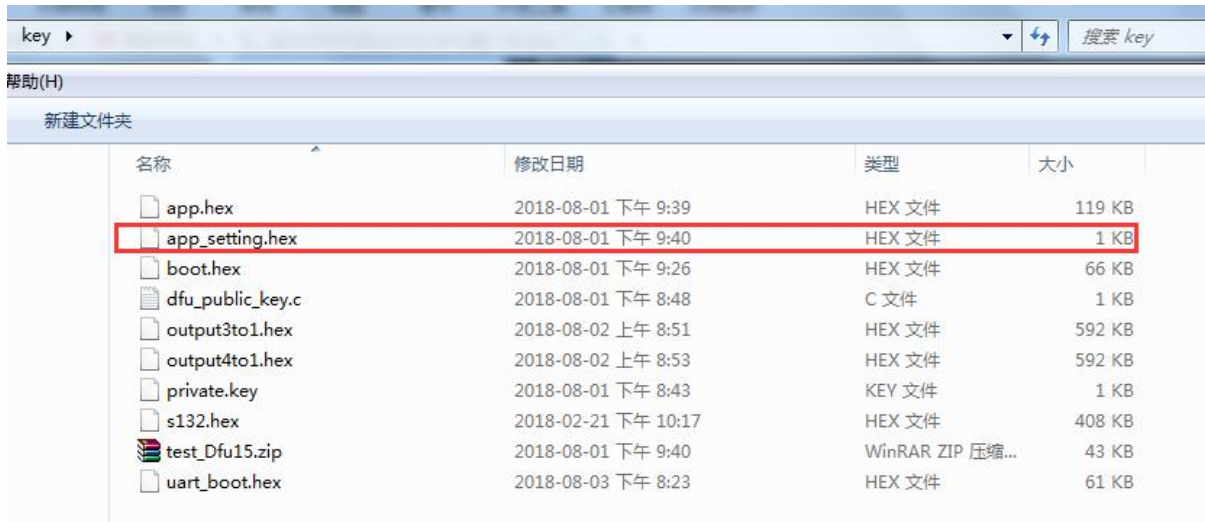
Generated Bootloader DFU settings .hex file and stored it in: app_setting.hex

Bootloader DFU Settings:
* File: app_setting.hex
* Family: nRF52
* CRC: 0x484F88AD
* Settings Version: 0x00000001 <1>
* App Version: 0x00000001 <1>
* Bootloader Version: 0x00000001 <1>
* Bank Layout: 0x00000000
* Current Bank: 0x00000000
* Application Size: 0x00005CC8 <23752 bytes>
* Application CRC: 0x21C76A56
* Bank0 Bank Code: 0x00000001

I:\key>
I:\key>
```

注意: --application-version 和 --bootloader-version:为用户选择的 application 和 bootloader 最初的版本,这里我都填 1。

如果生成成功后, 打开 key 文件夹, 会发现生成了 app_setting.hex 文件, 如下图所示:



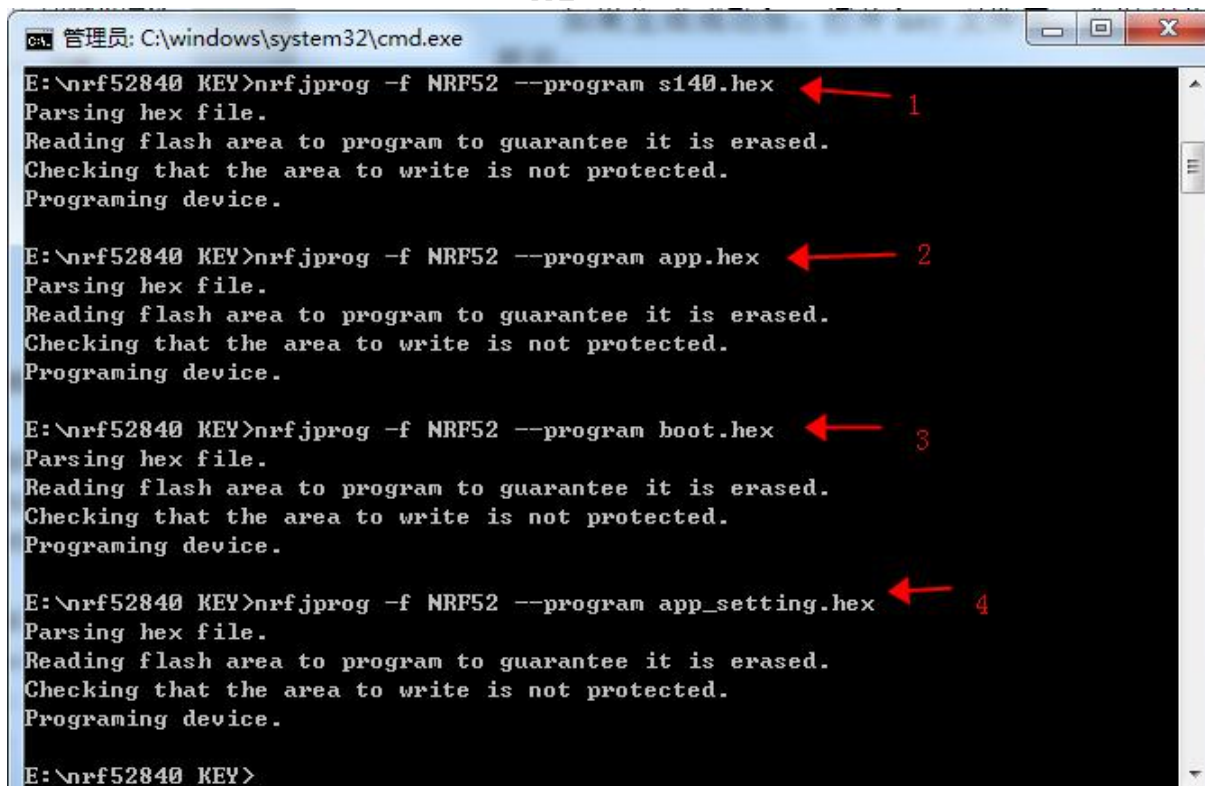
然后依次烧录如下文件, 如下图所示:

nrfjprog -f NRF52 --program s140.hex 烧写协议栈。

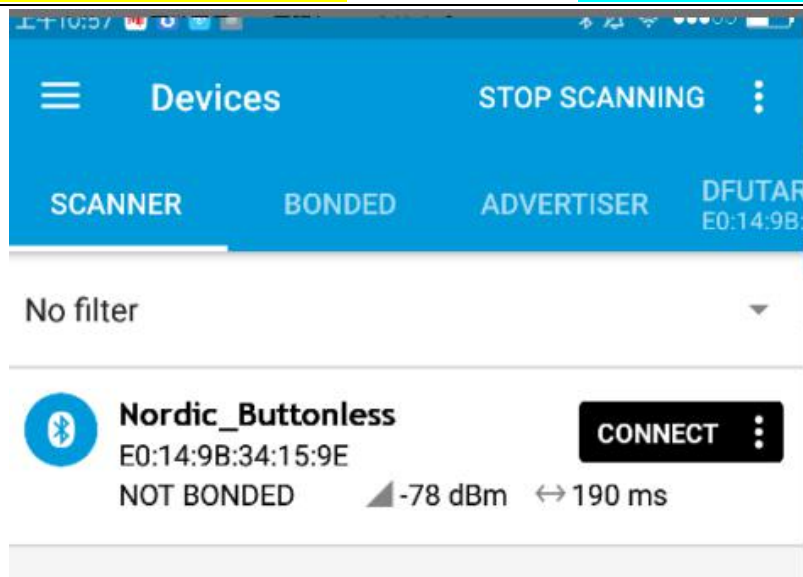
nrfjprog -f NRF52 --program app.hex 烧写应用。

nrfjprog -f NRF52 --program boot.hex 烧写 boot 引导程序

nrfjprog -f NRF52 --program app_setting.hex 烧写 app 设置程序。



烧录完后, 重新上电, app 搜索, 直接显示应用:



28.3.4 hex 的烧录与合并

为了进一步简化批量生产的步骤，我们可以把所有需要烧录的文件打包，形成一个单独的 hex，方便批量烧写，这里可以用到 nrf5x command line tool 里的 mergehex 指令，改指令一次最多可以合并 3 个 hex 文件。一个 4 个 hex，我们分两步来进行合并：

1. 合并协议栈 hex, boot 的 hex, 应用 hex。

在刚才我们生成的 key 文件夹中，存放了三个 hex 文件的 key 文件夹，在此处打开 dos 命令(shift+右键打开)。然后输入命令：

```
mergehex --merge s140.hex boot.hex app.hex --output output3to1.hex
```

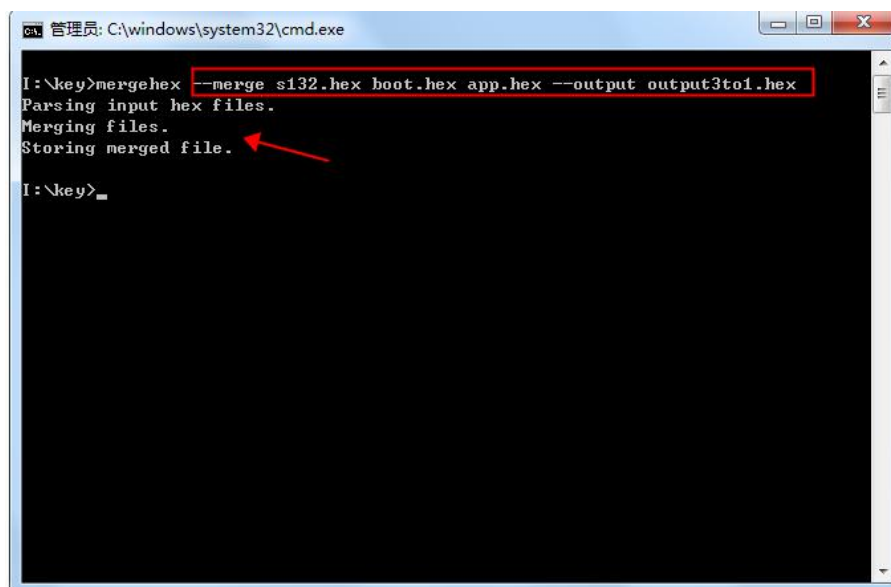
注：s140.hex 为协议栈 hex 文件，

boot.hex 为 boot 的 hex 文件，

app.hex 为应用的 hex 文件，

output3to1.hex 为需要输出的 hex 文件。如下图所示：

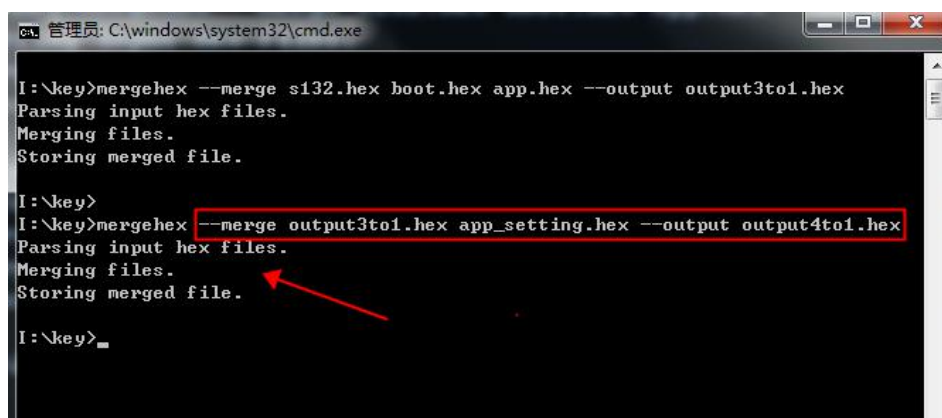
输入命令后回车，生成了 3 个文件的合并 hex 文件：



```
管理员: C:\windows\system32\cmd.exe
I:\key>mergehex --merge s132.hex boot.hex app.hex --output output3to1.hex
Parsing input hex files.
Merging files.
Storing merged file.
I:\key>
```

继续合并, 把 3 合一的 hex 和 app_setting.hex 进行合并, 指令如下, 输出 output4to1.hex:

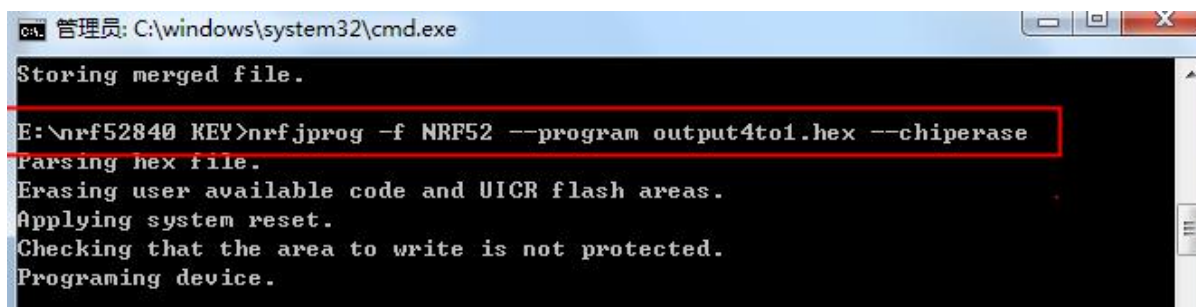
mergehex --merge output3to1.hex app_setting.hex --output output4to1.hex 输入命令后回车, 生成了 4 个 hex 的合并文件



```
管理员: C:\windows\system32\cmd.exe
I:\key>mergehex --merge s132.hex boot.hex app.hex --output output3to1.hex
Parsing input hex files.
Merging files.
Storing merged file.
I:\key>
I:\key>mergehex --merge output3to1.hex app_setting.hex --output output4to1.hex
Parsing input hex files.
Merging files.
Storing merged file.
I:\key>
```

合并的 hex, 就可以直接用 nrfgo 的应用进行烧录:

nrfjprog -f NRF52 --program output4to1.hex --chiperase
烧录完后重新上电就可以跑到应用了:



```
管理员: C:\windows\system32\cmd.exe
Storing merged file.
E:\nrf52840 KEY>nrfjprog -f NRF52 --program output4to1.hex --chiperase
Parsing hex file.
Erasing user available code and UICR flash areas.
Applying system reset.
Checking that the area to write is not protected.
Programming device.
```



这里我们就实现了整个 DFU 的升级过程。那么下一讲，我们将讲下如何把应用程序改造成带 DFU 功能的例子。