# IAP (In-Application Programming) 介绍

**Application Note for 32-bit NuMicro® Family**

**Rev.1.24 — Jul.3, 2018**

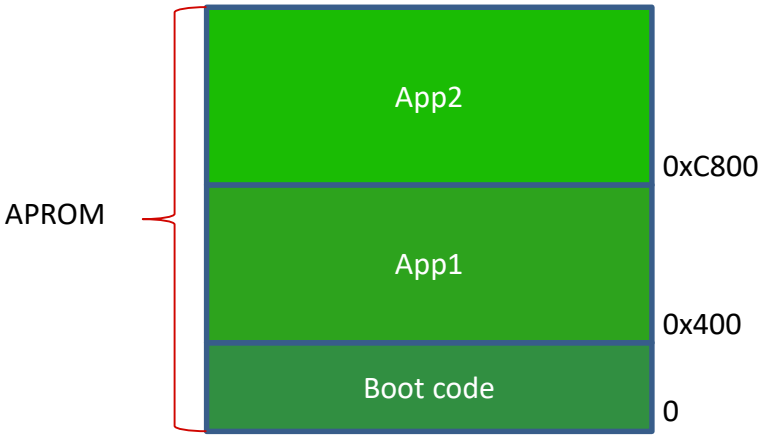
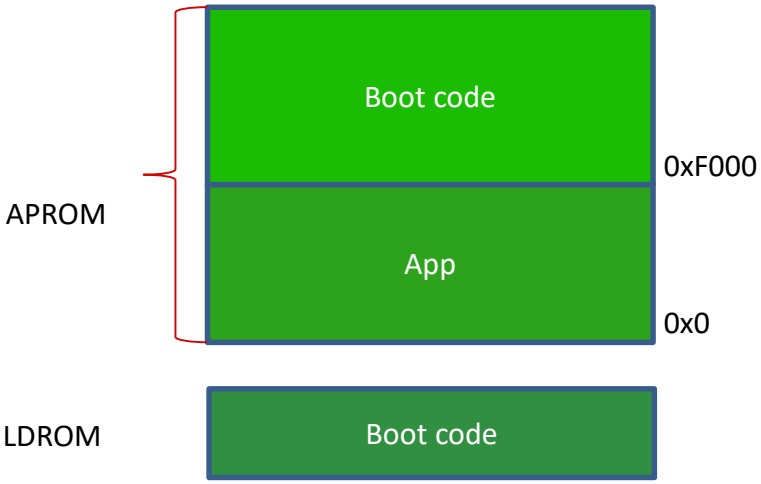
## Document Information

| | |
|---|---|
| **Abstract** | 本文档介绍IAP的概念和用法，以及ISP与IAP的区别. |
| **Apply to** | 新唐 Cortex-M MCU 系列. |

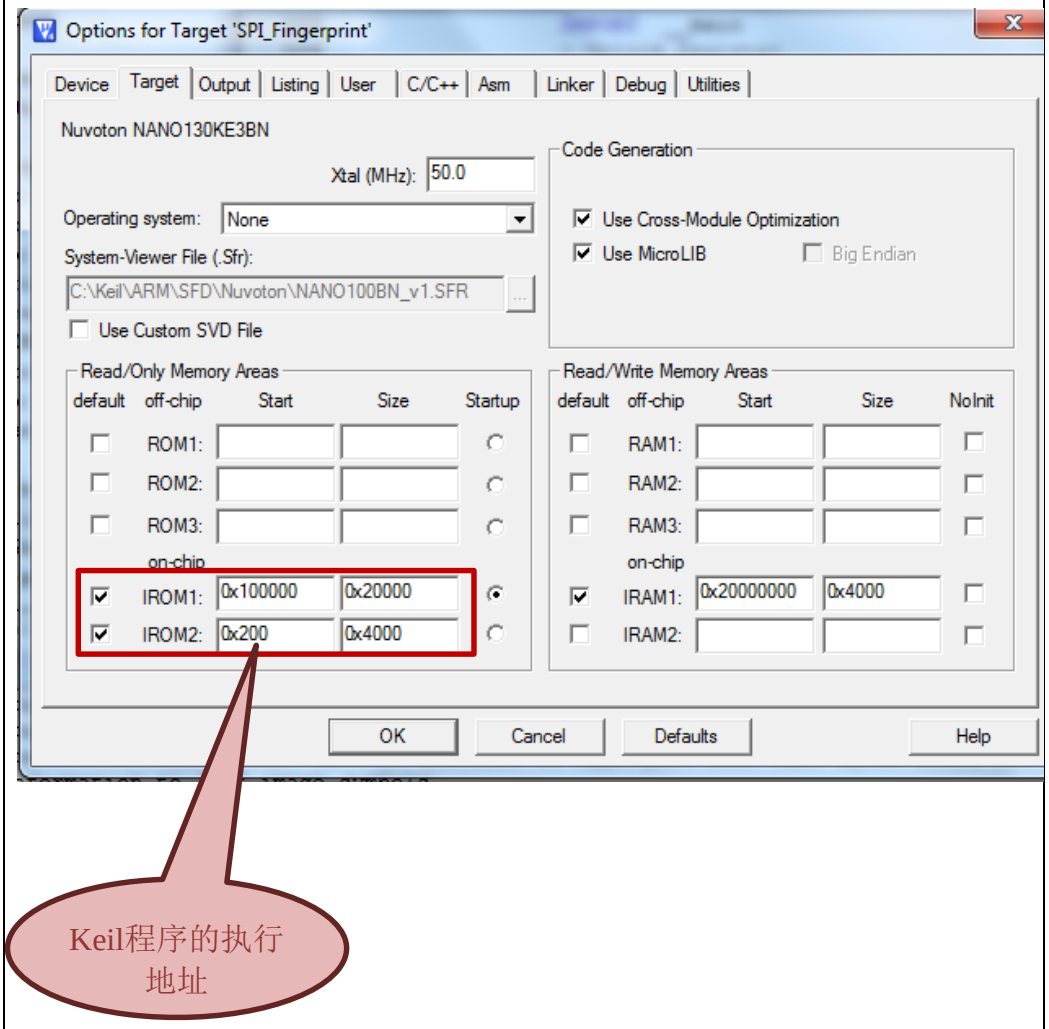
Table of Contents

# 1 名詞解釋

| | |
|---|---|
| **APROM** | 应用程序存放空间，应用程序可以存放在该 ROM 中，并在APROM中执行 |
| **LDROM** | ISP程序存放空间，类似于bootloader存放的地方，并在LDROM中执行。如果没有ISP程序，LDROM可以当作dataflash，用于存放数据 |
| **ISP** | In System Program，在系统中编程。一般放在LDROM里面，通过UART/USB等接口升级应用程序。如果IAP使能，ISP程序也可以放到APROM中。 |
| **BOOTLOADER** | 就是ISP程序，一般放在LDROM里面，如果使能了IAP，也可以放在APROM中，例如：ISP程序放在APROM地址 0 的地方，APP1 执行地址为 0x400，APP2 执行地址为0xC800。Boot code决定跳到 App1 还是App2，如果跳到 App1，执行VECTOR_PAGE_REMAP 命令，将地址0x400映射到地址0；如果跳到App2，执行 VECTOR_PAGE_REMAP 命令，将地址 0xC800 映射到地址 0。 |
| **IAP** | In Application Program，在应用中编程。它的本质就是应用程序放在APROM/LDROM 的任何地方都可以执行。因为复位或者发生中断时CPU都是从向量表拿代码执行，而向量表固定放在地址0，所以要解决程序的执行地址非0的情况，就要解决向量表的重新映射的问题。另外，该模式下 APROM 和 LDROM 中的程序可以互相调用，LDROM中程序放不下，可以将一部分放到APROM中；反之如果不需要通过 ISP升级，APROM 中的程序也可以将一部分放到 LDROM中。<br>如果IAP没有使能，程序在LDROM中执行时，CPU不能执行在APROM中的代码，APROM只能当作数据访问；程序在 APROM中执行时，CPU不能执行在 LDROM中的代码，LDROM 只能当作数据访问。这就是所谓的不可见。 |
| **CBS** | CONFIG0寄存器中 Chip Boot Select位，用于选择上电默认从APROM启动还是 LDROM 启动 |

| | |
|---|---|
| ISPCMD | ISP 命令寄存器。该 ISP 不要和上面ISP程序搞混了，该 ISP 是FMC IP 的寄存器。为了读/写/擦除ROM，需要通过下ISP command 的方式驱动 FMC IP 执行读/写/擦除等命令。ISP命令除了读/写/擦除，还有读 UID/UCID，以及向量表重新映射。 |
| **VECTOR_PAGE_REMAP** | 向量表重新映射命令。该命令用于重新映射向量表。重新映射的意思就是将某个地址映射到地址0，例如:如果应用程序的执行地址为 0x400，将地址 0x400 映射到地址 0，CPU从地址 0 拿代码其实是拿的地址 0x400 的。向量表重新映射代码如下：<br>`ISPADR = 0x400;`<br>`ISPCMD = VECTOR_PAGE_REMAP;`<br>`ISPTRG = 1;`<br>之后CPU访问地址 0，其实就是访问的地址 0x400<br>  <br>**LDROM 中 ISP 代码放不下，可以如下图放置：**<br>  |

大家记住一点，IAP的本质就是程序放到任何地址都能执行，当然程序要编译到该地址才行。对应于keil设定如下，细节后面会介绍



# 2 介绍

新唐的 Cortex-M 系列大都支持 IAP(In-Application Programming)，按字面翻译就是在应用中进行编程 (可以用来进行软件更新)，本质就是程序放到任何地址都可以执行（当然这个任何地址要与程序编译时指定的程序运行基地址相同）。有了这个特性之后，APROM 中的程序可以调用 LDROM 中的函数，LDROM 中的程序也可以调用 APROM 中的函数。对 CPU来说它们都是同时可读的，可以在里面执行代码。

使能IAP之后，因为程序可以下载到非0的地址，这就涉及程序下载地址offset设定问题。因为程序可以一部分放到LDROM一部分放到APROM，或者可以离散放到APROM中，这就涉及离散下载的问题。目前可以用keil、ICP或者ISP tool下载到ROM中

下面为大家介绍一下使用的细节。

## 2.1 寄存器

先介绍一下涉及到的寄存器：

- ✧ **CONFIG0** 中的 CBS 位。只要将该位设为’10b’或者’00b’就使能了 IAP 功能。
  - ■ 设为 ’10b’ 从 APROM 启动并使能 IAP，
  - ■ 设为 ’00b’ 从 LDROM 启动并使能 IAP。
- ✧ **ISPCMD** 寄存器。该寄存器是用来指定要执行的 ISP 命令的，这些命令大都是用来读/写/擦除 ROM 的。有一个特殊命令”VECTOR_PAGE_REMAP”，该命令用来重新映射向量表。因为 Cortex-M0/M4 指定向量表必须放到地址 0 的地方，所以需要重新映射向量表，将 应用程序的向量表映射到地址 0 的位置。

## 2.2 向量表结构

每个入口为4个字节

向量表入口 0的位置为**栈地址 (R13寄存器)，栈地址将用来初始化 R13寄存器的值**

向量表入口 1 的位置为复位中断处理函数，就是**程序的入口**

**系统复位时，Cortex-M0/M4 会自动将地址0的值赋值到 R13，然后从地址 0x04 拿到处理函数执行**

**依次往后为NMI/HardFault以及外部中断的中断处理函数**

| 异常号 | 异常 | 优先级 |
|---|---|---|
| 1 | Reset | -3 |

| | | |
|---|---|---|
| 2 | NMI | -2 |
| 3 | HardFault | -1 |
| 11 | SVCall | 由寄存器SHPR2配置 |
| 14 | PendSV | 由寄存器SHPR3配置 |
| 15 | SysTick | 由寄存器SHPR3配置 |
| 16 | 外部中断 (0) | 由寄存器NVIC_IPRx配置 |
| … | … | … |
| 16 + N | 外部中断 (N) | 由寄存器NVIC_IPRx配置 |

**Cortex-M中向量表固定放在地址为0的地方，复位或者发生中断时，CPU会从地址0拿code执行。如果应用程序没有放在地址0的地方，向量表也就不在0的位置了。为了支持程序放在非0的位置能够执行，必须执行VECTOR_PAGE_REMAP命令，将其它地址重新映射(remap)到地址0.**

## 2.3 理解 IAP

所谓 IAP 就是 In-Application Programming。

这个技术有2个关键点

1. 就是对 CPU 来说 APROM 和 LDROM 所有的空间都是可见的，所谓可见就是 APROM 中的程序可以调用 LDROM 中的函数，LDROM 中的程序可以调用 APROM 中函数。
2. 客户的应用程序能放到 APROM/LDROM 的任何地址，而不必非要放在头部的位置。

第一項只要使能IAP功能(參考CBS位說明)，CPU就能“看”到所有的APROM和LDROM空间了。

第二項就涉及到向量表映射，就是VECTOR_PAGE_REMAP 命令。按照Cotex-M0/M4的架构，IAP 程序使用的向量表必须要放到地址 0x0000 的地方。如果客户将 应用程序放到了地址非 0x0000 的地方，例如：0x1000，则 向量表就在0x1000的地方，这时候如果发生中断，CPU从地址 0x0000 的位置拿到的 中断处理函数的地址就不正确。这时候就需要用VECTOR_PAGE_REMAP

命令将地址0x1000位置的 1 个page 大小映射到地址 0x0000。1个page大小就是512B。这就意味着Vector Page Remap命令将0x1000 – 0x1200映射到了0x0000 – 0x0200。**这个动作会导致一个副作用**：原本地址0x0 – 0x200地方的代码除了向量表还有其他的代码，假设向量表占了256B，还是有256B是其它代码，这样一映射，向量表变成了地址0x1000处的，但是原本地址0x0100处的代码也不见了。怎样将在地址0x0 – 0x200的代码移到0x200后面呢？编译器一般都提供了强大的武器——Link Script。我们以Keil为例说明Link Script的写法，假设：Link Script文件取名 `link.ld`，整个 `link.ld` 文件内容如下：

```
;说明应用程序基地址，也就是向量表RESET段的地址为0x00000000 。如果程序执行地址是0x1000，就把0x00000000改成0x1000
ROM1 0x00000000
{
    IROM1 +0 {  ;
         *(RESET, +First)
         startup_NANO1xx.o(+RO)
         *(InRoot$$Sections)
    }
    FLASH1 0x00000200 FIXED{    ; 其它程序都放到0x200后面
         .ANY (+RO)
    }
    RAM1 0x20000000 0x10000  {    ; 说明SRAM的基地址和SIZE ，并且将RW和ZI Data放到SRAM中
          ; RW data
         .ANY (+RW +ZI)
    }
}
```

上面的Link Script 文件描述了3个段，RAM1段放在 SRAM 空间，ROM1 段放在地址 0 的地方，FLASH1段放在 0x200。将该link script 文件填入keil，这样程序在连接的时候，连接器就会按照Link Script的描述摆放程序。

***只有放在地址0x0000的程序才需要用到link script，其它地址的程序不会有代码放在地址0x0000***

## 2.4 ISP 与 IAP 的区别

| | 可以更新APROM中的应用程序 | 应用程序可以在APROM任何地址执行 | APROM中的程序可以调用LDROM中的 | LDROM中的程序可以调用APROM中的 | APROM与LDROM中的程序切 |
|---|---|---|---|---|---|

| | | | 函数 | 函数 | 换，是否需要复位系统 |
|---|---|---|---|---|---|
| **ISP** | Yes | No，只能从0开始执行 | No | No | Yes |
| **IAP** | Yes | Yes | Yes | Yes | No |

其实 ISP 是 IAP 的一个子集。它们两个都可以实现更新应用程序的功能。但是只有IAP才能实现程序放到任何地方都能执行的功能：放到0x0000，0x1000，0x4000, 0x100000 (LDROM)，都可以执行。ISP只能放到地址 0x0000 执行，如果系统从 APROM 启动，APROM 就在地址 0x0000 的地方；如果系统从LDROM启动，LDROM就在地址0x0000的地方。

但是如果 IAP 使能，APROM 的地址就固定在 0x0000，LDROM 的地址就固定在 0x100000. 然后通过 VECTOR_PAGE_REMAP 命令映射任意 ROM (APROM/LDROM) 地址的一个页到地址 0x0000（向量表）。

ROM(APROM/LDROM)中的程序更新之后，ISP 需要执行系统复位，然后系统重新启动。

IAP可以执行系统复位切换程序，也可以直接跳转。有一些应用不希望执行系统复位的动作来切换程序，此时它只要修改栈寄存器 R13，重新映射 vector table，然后直接跳去新的应用程序执行就可以。

IAP关闭，系统从APROM启动时APROM的地址就在0x0；系统从LDROM启动时LDROM的地址就在0x0

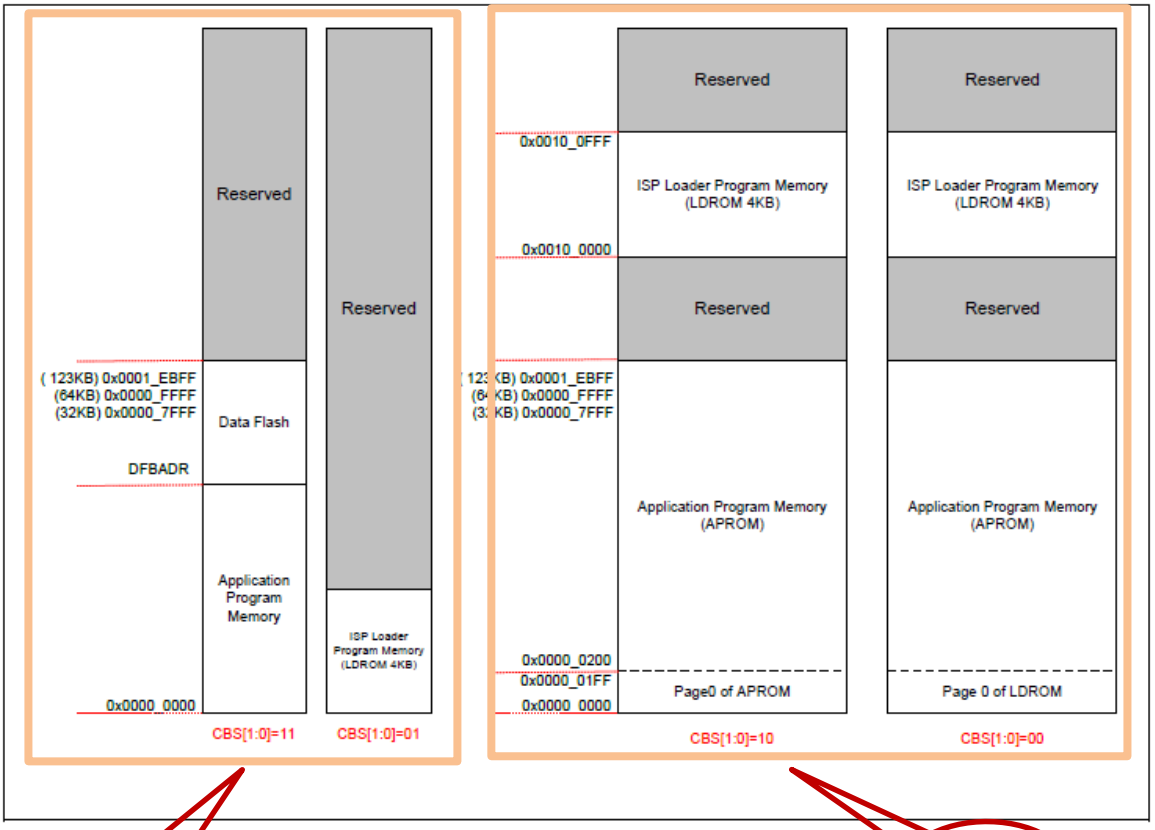



Figure 5-34 Flash Memory Mapping of CBS in CONFIG0

IAP关闭

IAP使能

IAP使能时的图说明，无论系统从APROM还是LDROM启动，APROM都在地址 0x0000 的位置，LDROM都在地址0x100000的位置。但是系统从APROM启动时，APROM的Page0 就会映射在0x0；系统从LDROM启动时LDROM的Page0就映射在0x0。注意，只有 Page0 映射到地址 0而已，其它 Pages 还在0x100200往后的位置。那原本0x100000处的代码还在吗？还在的，从0x100000和0x0000都能访问到

## 2.5 注意事项

**不能在中断里面切vector table，并跳转去另一个APP。这会导致 Cortex-M的状态机一直在中断里面，新的APP可能无法发生中断。**

## 2.6 IAP 使用方法

新唐的 M0 IAP有两种使用方法，一种支持系统复位(SystemReset)或者不复位的方式切换程序；

一种只支持不复位的方式切换程序。

早期我们做的IAP都不支持系统复位的方式切程序，但是后来发现如果使用系统复位的方式切程序，代码会简单很多。两者比较如下：

### 2.6.1 不复位方式两个 APP 之间切换的步骤

1) 为了安全起见复位所有的 IP

```
outpw(&GCR->IPRST_CTL2, 0xFFFFFFFF);
outpw(&GCR->IPRST_CTL2, 0);
```

2) 关闭所有中断

```
NVIC->ICER[0] = 0xFFFFFFFF;
```

3) 调用函数 `FMC_SetVectorPageAddr` 重新映射向量表
4) `spChange` 切栈地址
5) 跳转

### 2.6.2 复位方式两个 APP 之间切换的步骤

1) 关闭所有中断

```
NVIC->ICER[0] = 0xFFFFFFFF;
```

2) 调用函数 `FMC_SetVectorPageAddr` 重新映射向量表
3) 系统复位

```
NVIC_SystemReset();
```

大家有看出来两种切APP的方式之间的区别吗？第三章有专门列出复位方式切程序和不复位的方式切程序的代码，大家可以比较一下。

这里一直提系统复位，没有提其它的复位方式，是因为其它的复位方式不推荐。新唐的M0有3种复位方式：系统复位、CPU复位和硬件复位。

1. 因为 SystemReset 会同时帮忙把所有的 IP 都复位掉，免得影响下一个程序的稳定性。
2. 而硬件复位例如：Chip reset, BOD reset, WDT reset，/RESET 引脚等都不能用于 IAP 中程序切换。它们都是将整个芯片复位，类似于 POR，这种复位会导致 `VECTOR_PAGE_REMAP` 新映射的向量表地址失效。
3. 而 CPU 复位只是将 PC 指到新的地址，不会复位 IP。

每个型号详细支持列表如下：

| | |
|---|---|
| | 支持 SystemReset 切程序 |

| | |
|---|---|
| NUC200/NUC220 | No |
| NUC230/NUC240 | Yes |
| M051xxxDE | Yes |
| M051xxxDN | No |
| M051xxxBN | Yes |
| NUC131 | Yes |
| NUC130CN/NUC140CN | No |
| NUC123 | No |
| M058S | No |
| NANO100/NANO120/NANO130/NANO140BN | No |
| NANO100/NANO120/NANO130/NANO140AN | No |
| NANO102/NANO112 | Yes |
| Mini5xAN/Mini5xBN | No |
| Mini5xDE | Yes |
| NUC029 | No |
| NUC442/NUC470/NUC451 | Yes |
| NM1120 | Yes |
| M480 | Yes |

### 2.6.3 IAP 使用步骤

1) 使能 IAP 功能(CBS=10b 或者 CBS=00b)，可以用 ICP tool 或者在 Keil 中修改配置区使能
2) 设定 keil 确定应用程序执行地址
3) 下载程序
4) 重新上电

以上四步驟大家会说和 ISP 使用方法没有什么区别么？
详细展开就会看到区别了。

IAP 的关键就是程序放到 LDROM/ARPOM 的任何地址都可以执行，所以它们的不同是在 Link Script 或者 Keil 中的 APP 地址设定上。下面就这 4 步驟做详细说明

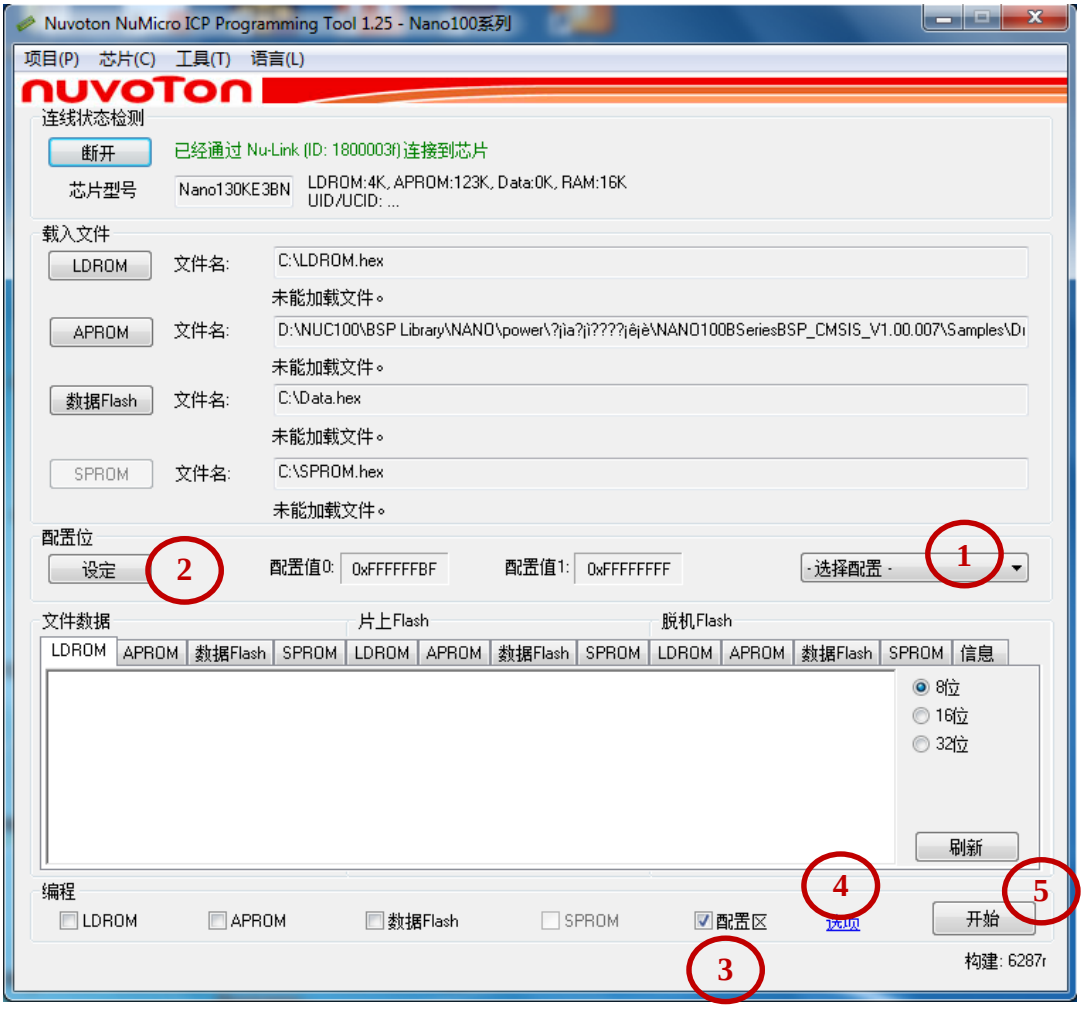
### 2.6.4 IAP 参考程序

1) **新唐在 BSP 中提供** Vector Page Remap 参考程序:FMC_VECMAP **供大家理解 vector remap**
2) **使能 IAP 的 ISP demo 程序：**IAPISP 。大家可以在论坛中(www.nuvoton-m0.com)下载到。该程序 demo 的是 ISP 程序超过 4KB，将一部分放到 APROM。这样 ISP 程序就分成两部分，一部分烧入 LDROM 中，一部分烧入 APROM 中。生成一个 hex 文件通过 ICP tool 下载比较方便。这样大家就可以用 ISP 程序，通过 UART/USB 更新应用程序。
**如果更新的程序执行地址非 0，例如需要烧入地址 0x400，有两种方法：**ICP tool 和 ISP tool 详细参考 2.6.7 节

### 2.6.5 使能 IAP

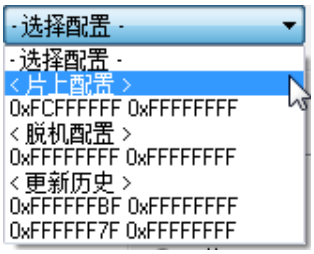
#### 2.6.5.1 使用ICP

打开 ICP tool，并连接到目标板




共5步即可实现使能IAP功能，下面详细介绍这5步具体怎么做

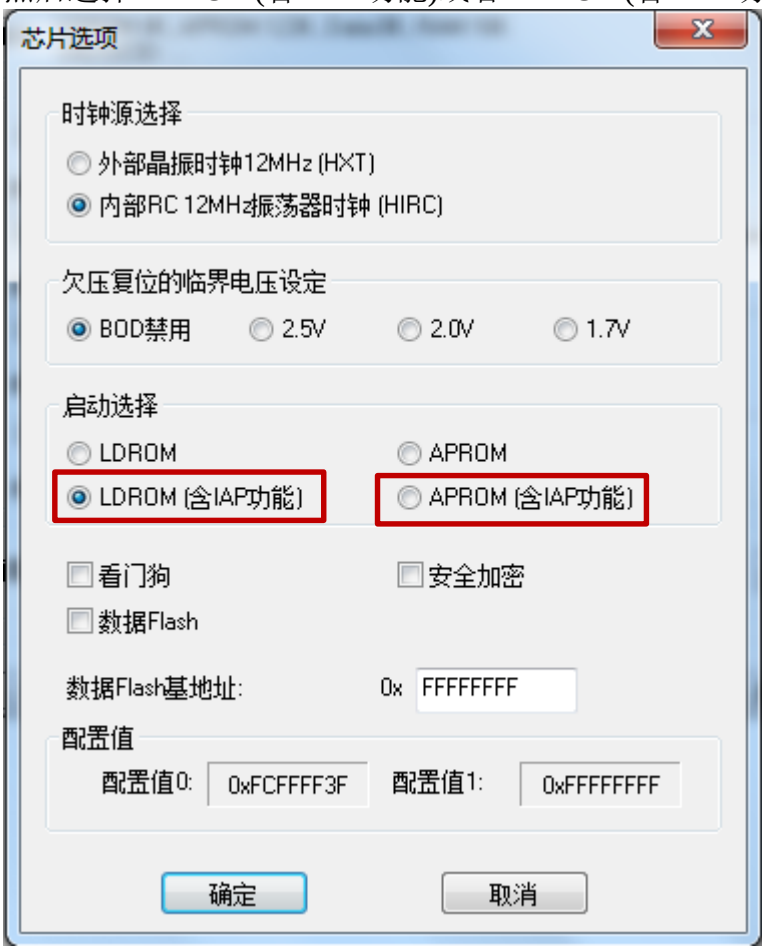
一、 点击选择配置下拉框，选择“片上配置”




二、 进入配置画面


点击设定 进入配置画面

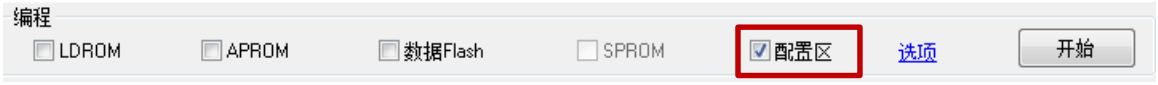
然后选择 LDROM(含 IAP 功能)或者 APROM(含 IAP 功能)




配置值 0 的值发生改变，特别 bit[7:6]从 11b 变成了 00b，表示启动模式从”APROM”变成了”LDROM 并且使能 IAP”的模式

之后点击确定

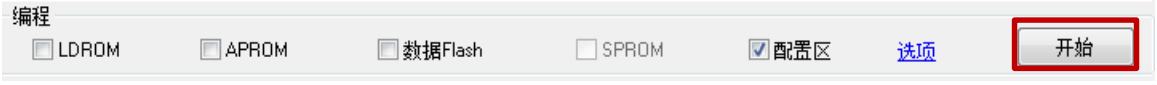
三、 勾选配置区




四、 点击选项


将红色方框中的相应项打勾

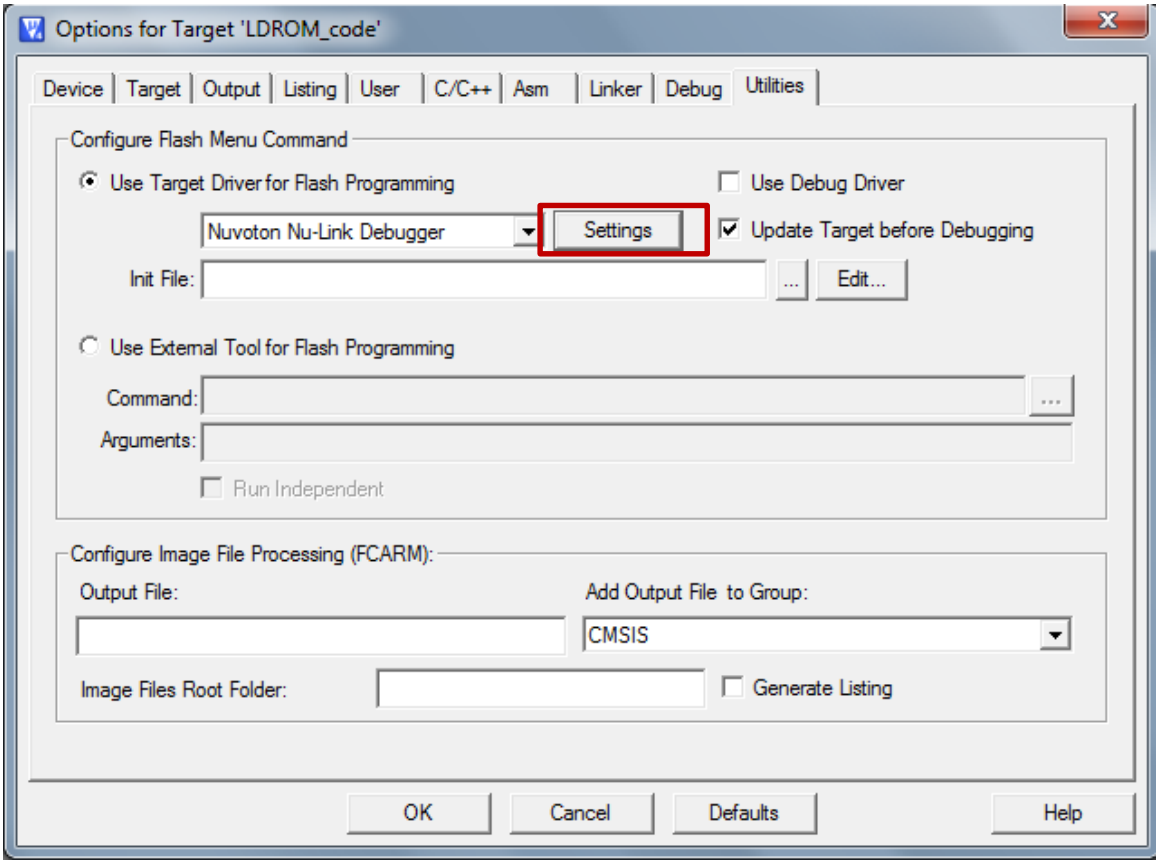
五、 最后点击“开始”，烧录成功之后，IAP 就使能了




### 2.6.5.2 使用keil打开IAP功能

Keil下选择Target Options ->Utilities->Settings

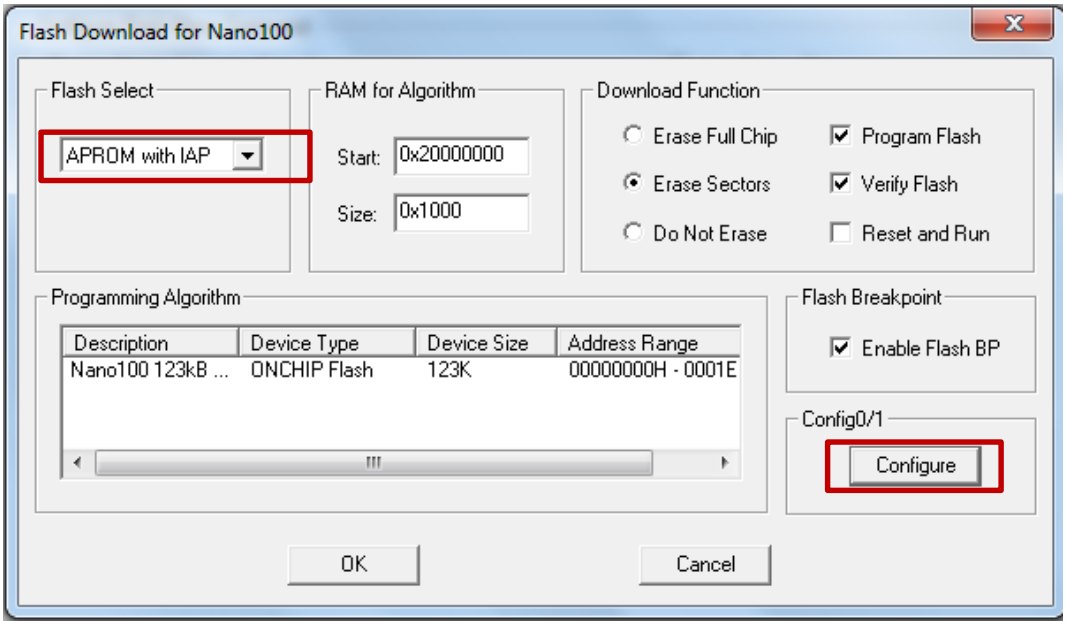
第一步：




点击“Settings”将出现第二步的画面

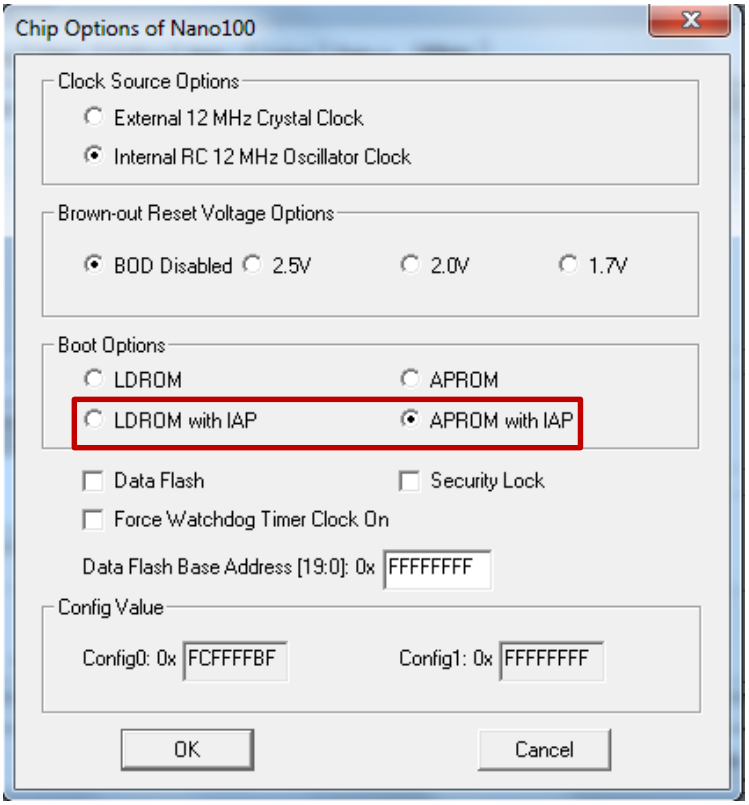
第二步：

Flash Select选择“APROM with IAP”或者“LDROM with IAP”。之后点击“Configure”



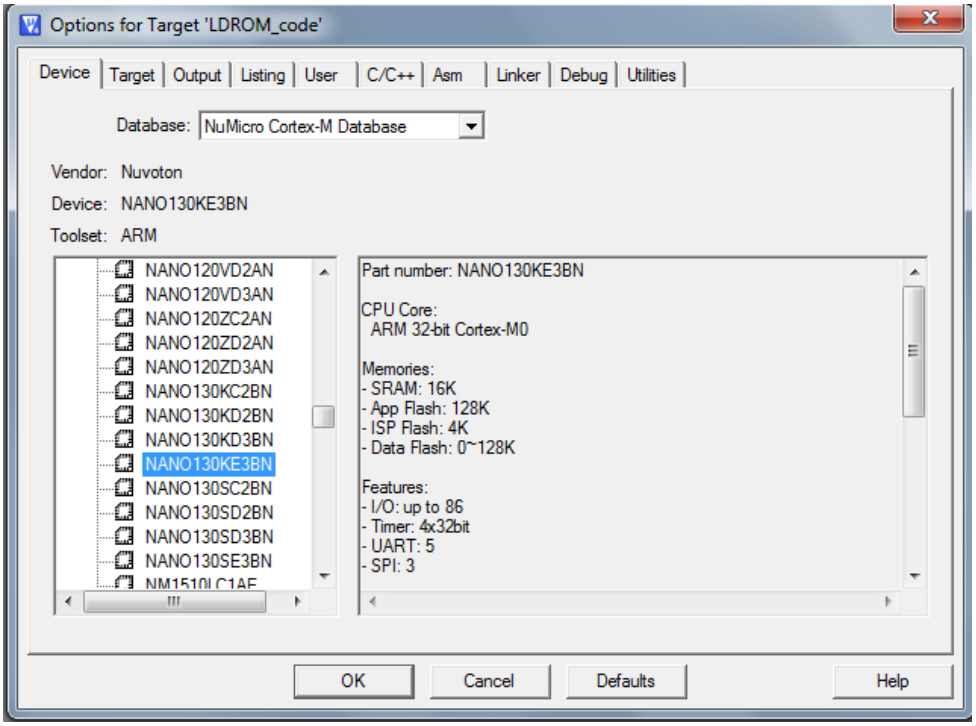

下面的画面选择“APROM with IAP”或者“LDROM with IAP”的功能，但是要跟上面的画面选择的一致

第三步：

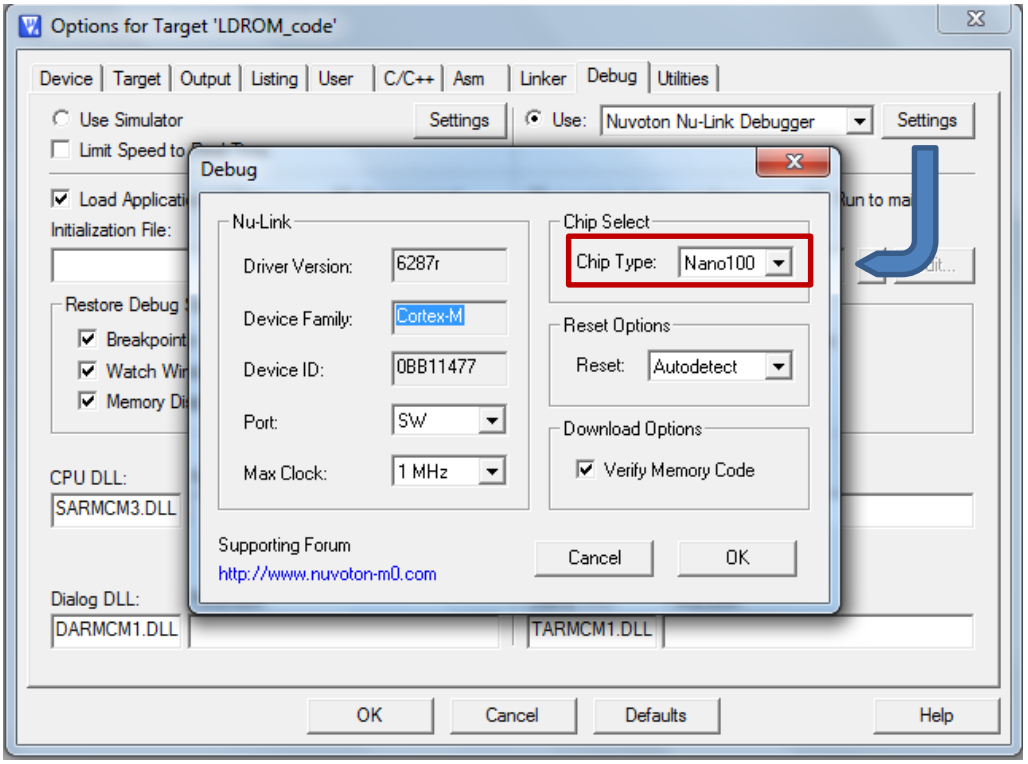


如果发现第二步和第三步没有with IAP的选项，应该是Nu-Link接的目标板和你Options->Device中以及Options->Debug中说明的类型不一致导致的。

例如：这里选择NANO130KE3BN




Debug中点击“Setting”，”Chip Select”下Chip Type也选择Nano100




并且连接的目标板也要是NANO130KE3BN，这样以后，如果该系列支持IAP，关于IAP的选项就会出现

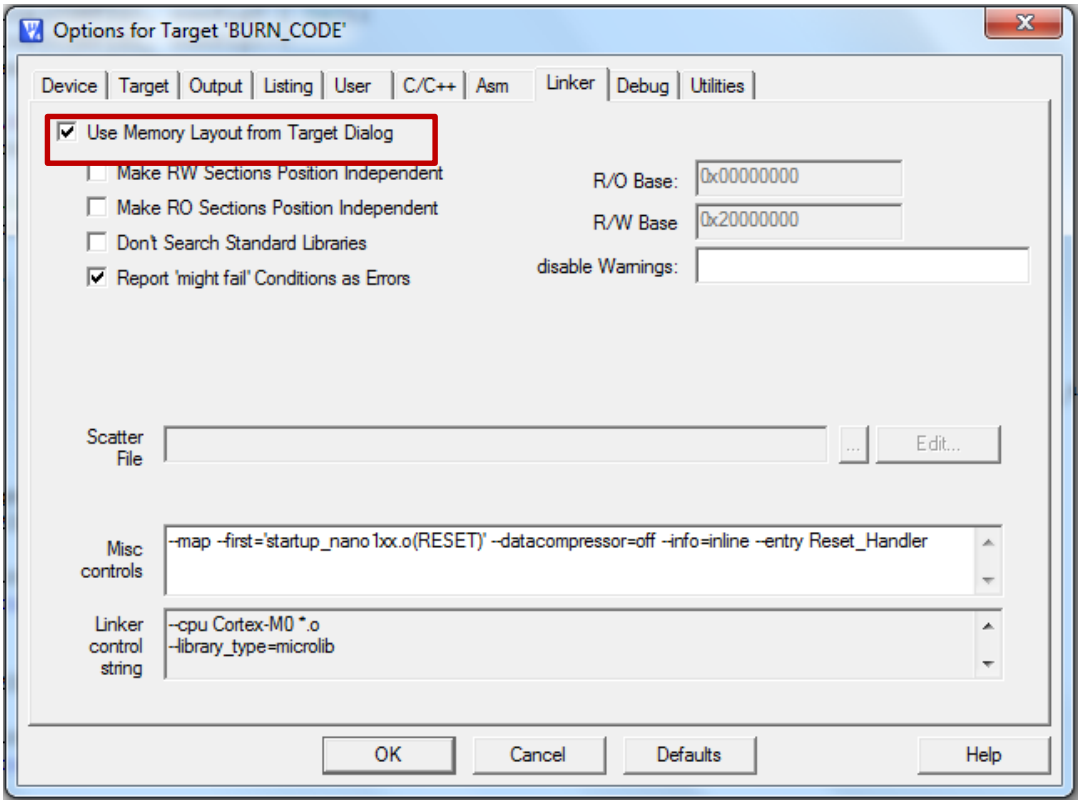
### 2.6.6 将应用程序编译到指定地址

假设APROM 的地址为 0 ~ Addr1，LDROM的地址为0x100000 ~ Addr2。Addr1和Addr2将随APROM和LDROM的大小不同而不同。到目前为止LDROM大都是4K的，APROM最大到128K。

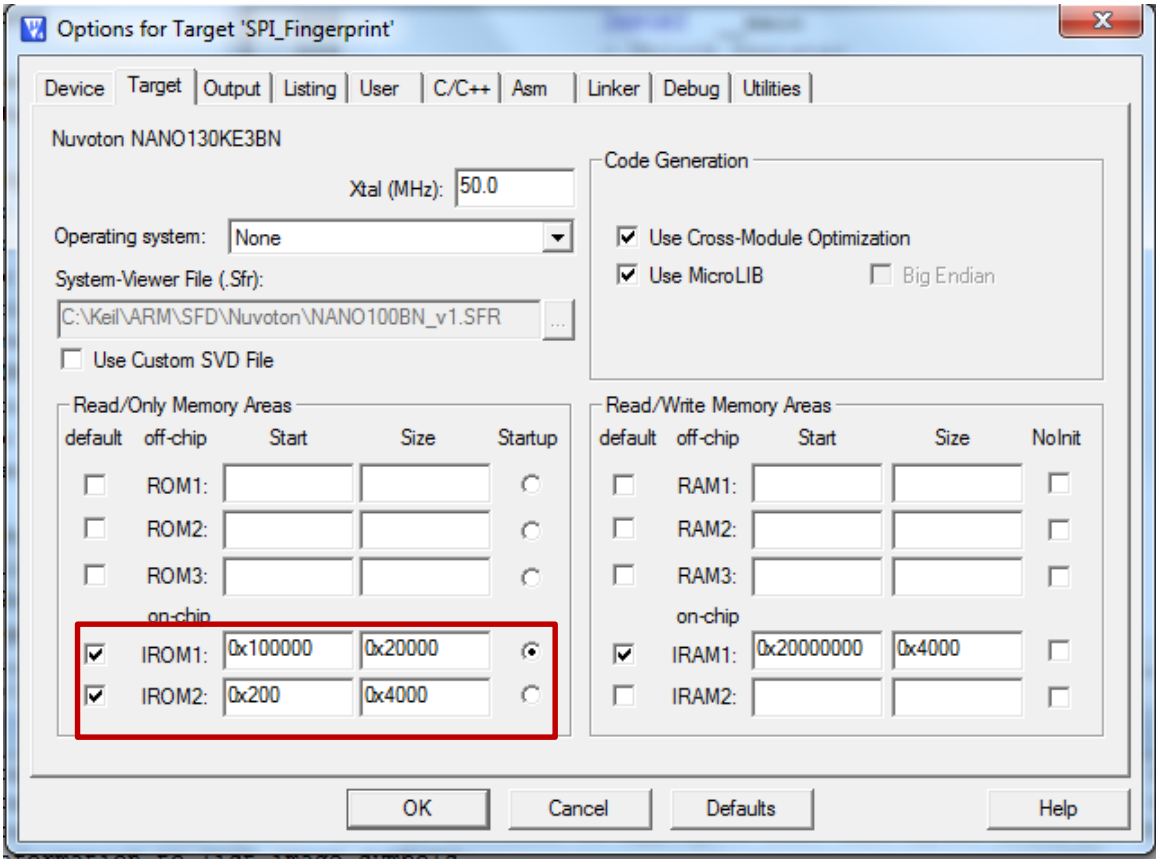
地址有了之后，怎么把程序编译到这个地址执行呢？这就是编译器的事了，下面以Keil为例说明设定方法。**下面的设定都以客户已经使能IAP为前提，如果没有使能IAP，APROM和LDROM的地址会有所不同。IAP使能的方法，请看1.3.1节.**

一、 情景一：程序从 LDROM 启动，bootloader 程序太大 4KB 放不下，所以有一部分放在 APROM 中。

a) Option->Linker 选择使用 Target Dialog



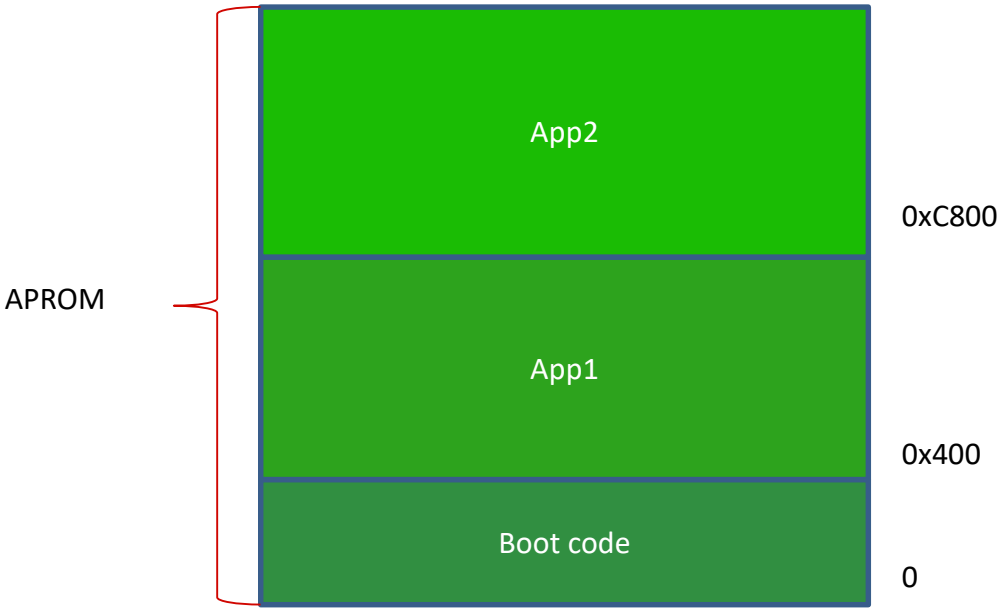

b) 填 Target Dialog




红色方框内的数据表明，共有两块ROM，一块4KB(基地址为0x100000)，另一块16K(基地址为0x200)， 系统从0x100000启动。因為LDROM的第一個page映射到0x0~0x200，所以APROM 地址0x0~0x200就不能用了。

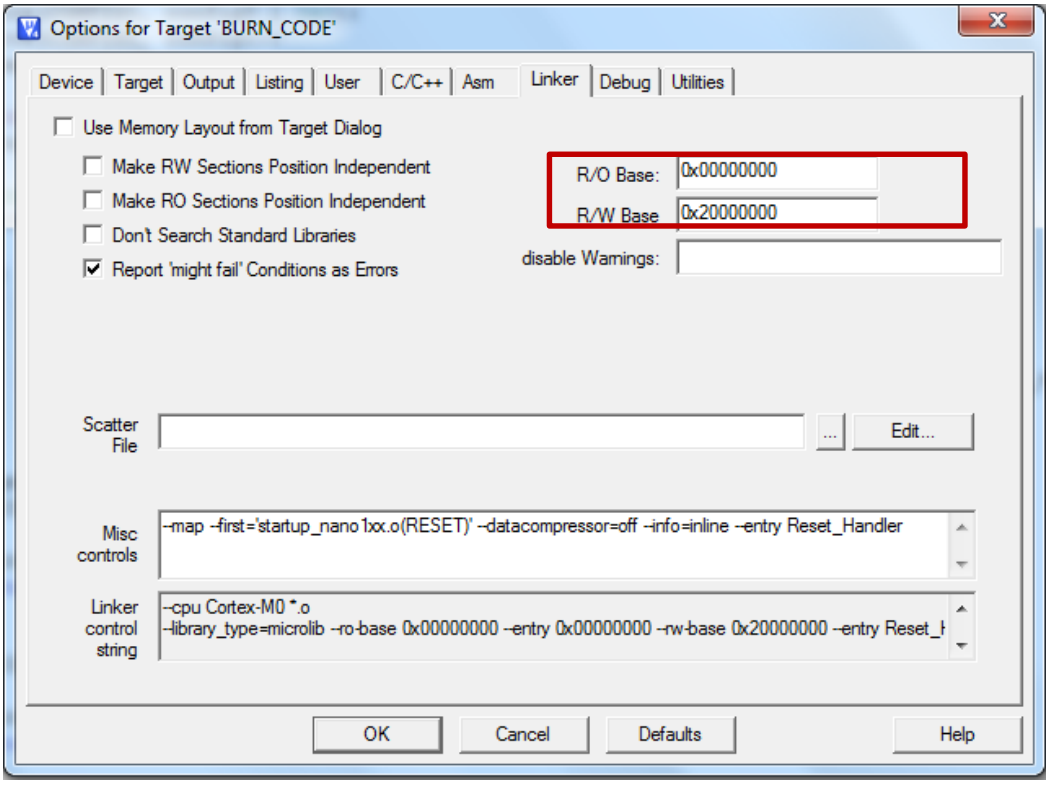
Bootloader起来之后(向量表由硬件自动映射到地址0x100000的地方)如果要跳去执行用户程序，需要先执行VECTOR_PAGE_REMAP 命令，将vector table地址切到APROM的地址，也就是地址0x0000，然后切堆栈地址，之后就可以跳转了。示例代码可以见文档最后。

二、 情景二：系统从 APROM 启动，将 bootloader 全部放到 APROM 里面，应用程序放在 bootloader 后面。如果有 2 个 App，可以依次往后面放。2 个 App 的情景一般是为了防止升级失败，一个 App 是另一个 App 的备份，Mapping 如下：



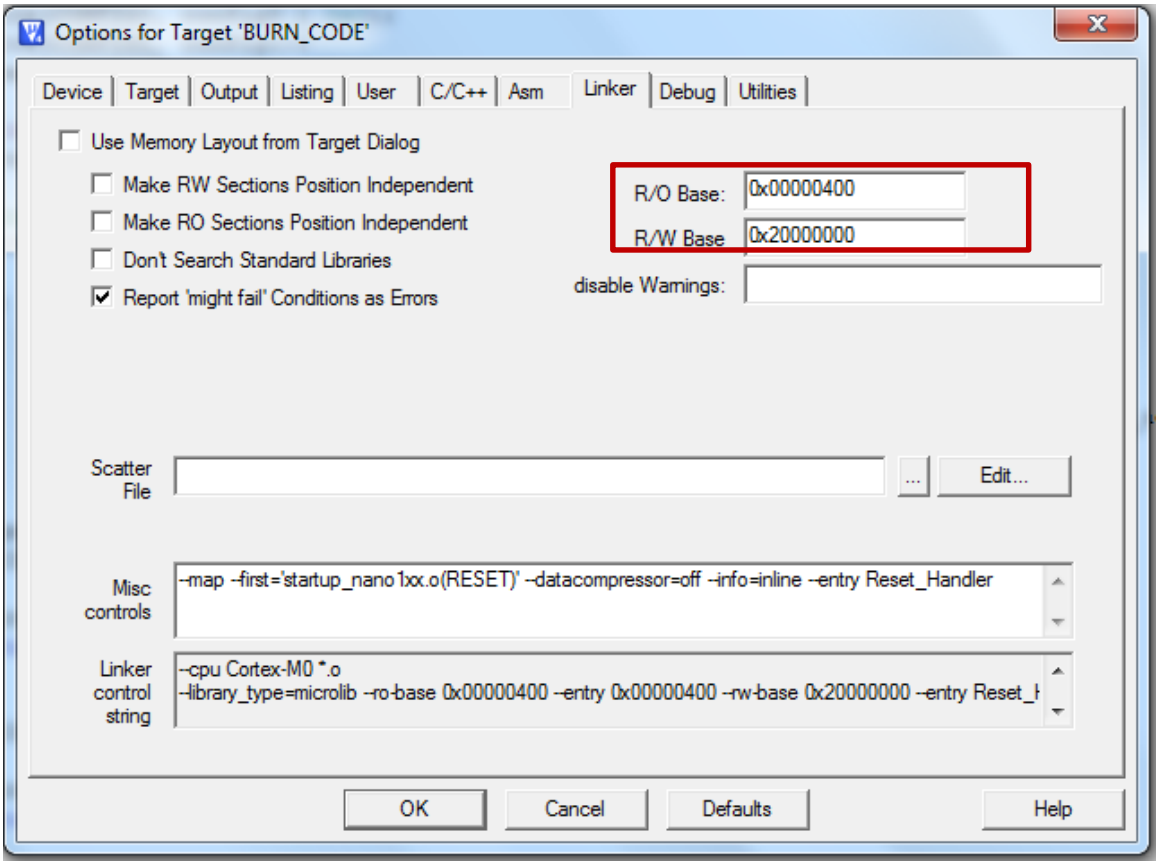

Keil设定如下：

a) Bootloader 的设定



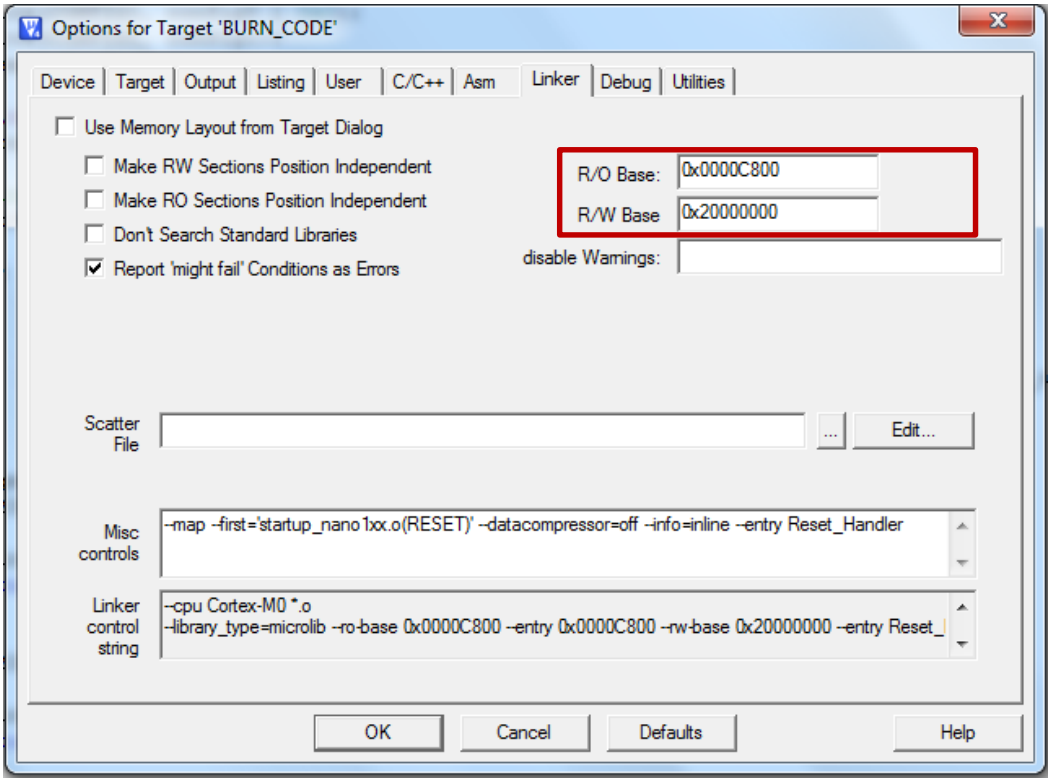

程序执行地址为 0x0
有人说不是 0x00000000~ 0x00000200 之间只能放 vector table，其他程序只能放 0x200 之后吗？哎，说明前面说的你记住了，恭喜恭喜！你怀疑的是对的，这里最好使用 link.ld

b) App1 的设定




程序执行地址为 0x400

c) App2 的设定

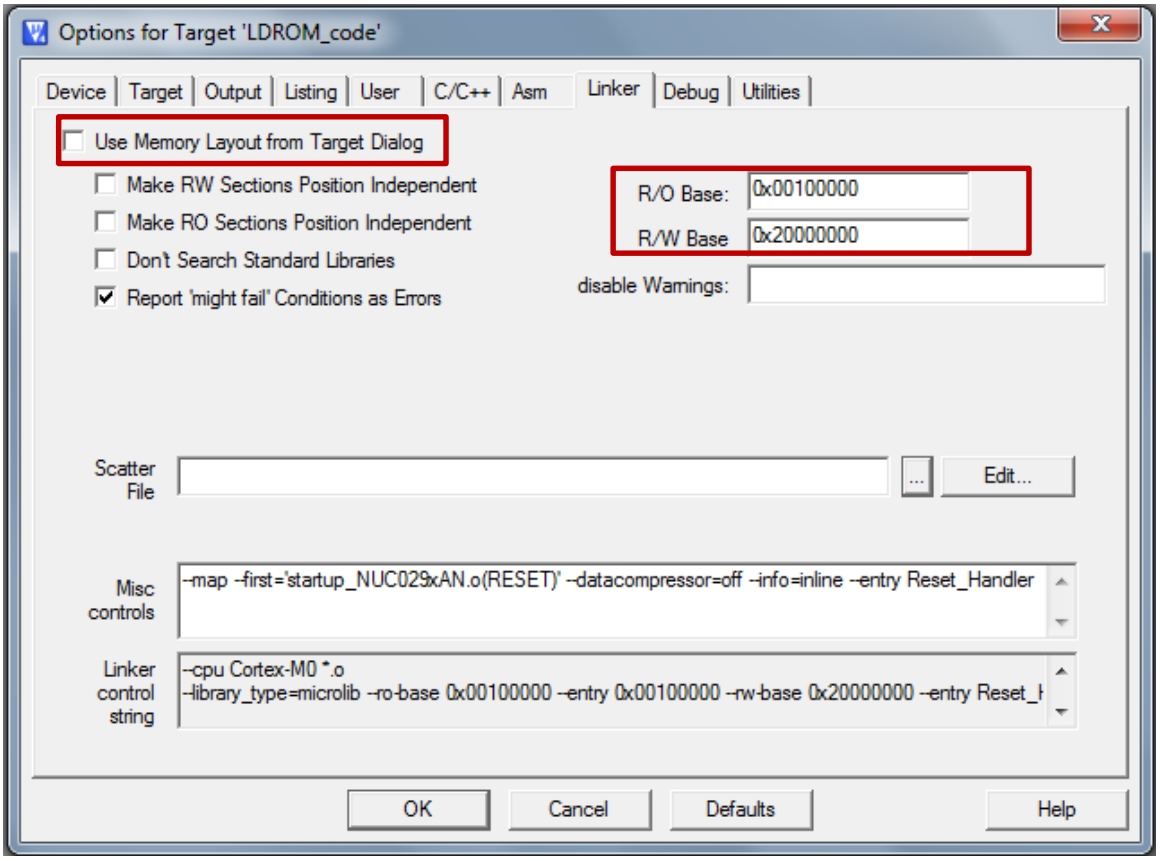



程序执行地址为 0xC800

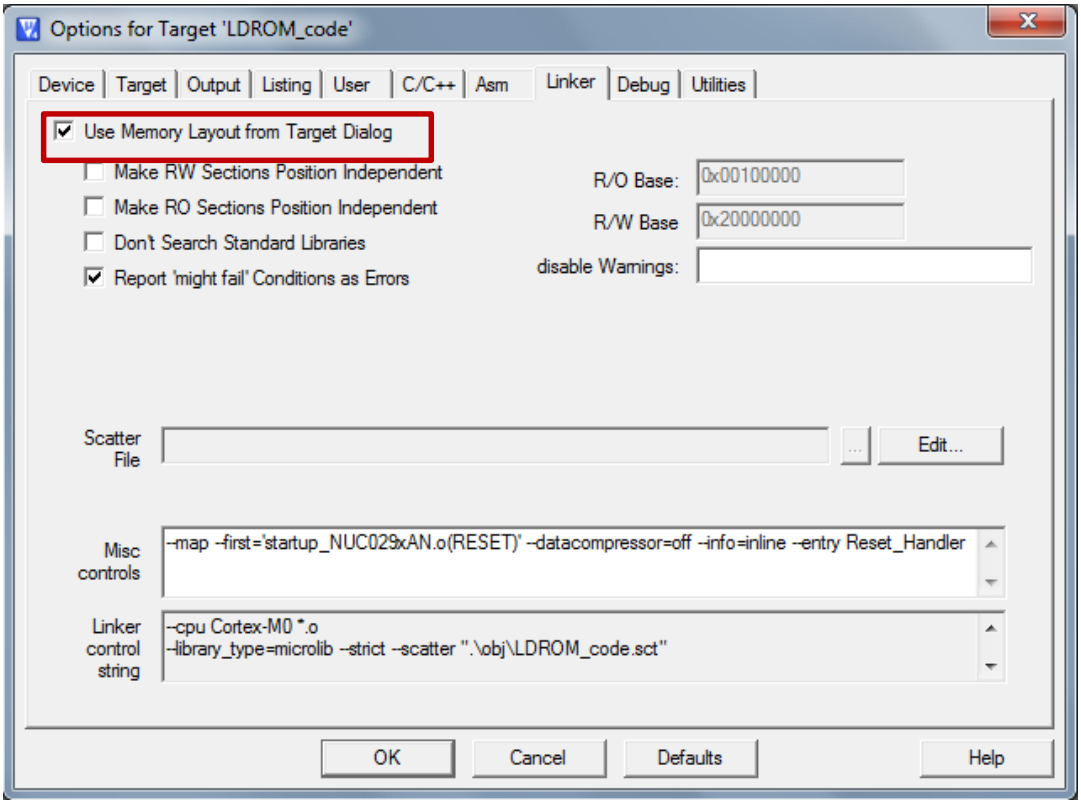
Bootloader起来之后(向量表由硬件自动映射到APROM中，也就是地址0x00的地方)如果要跳去执行用户程序，需要先执行VECTOR_PAGE_REMAP 命令，将vector table地址切到App1/App2的地址，也就是地址0x400/0xC800，然后设定堆栈地址，之后就可以跳转了。示例代码可以见文档最后。

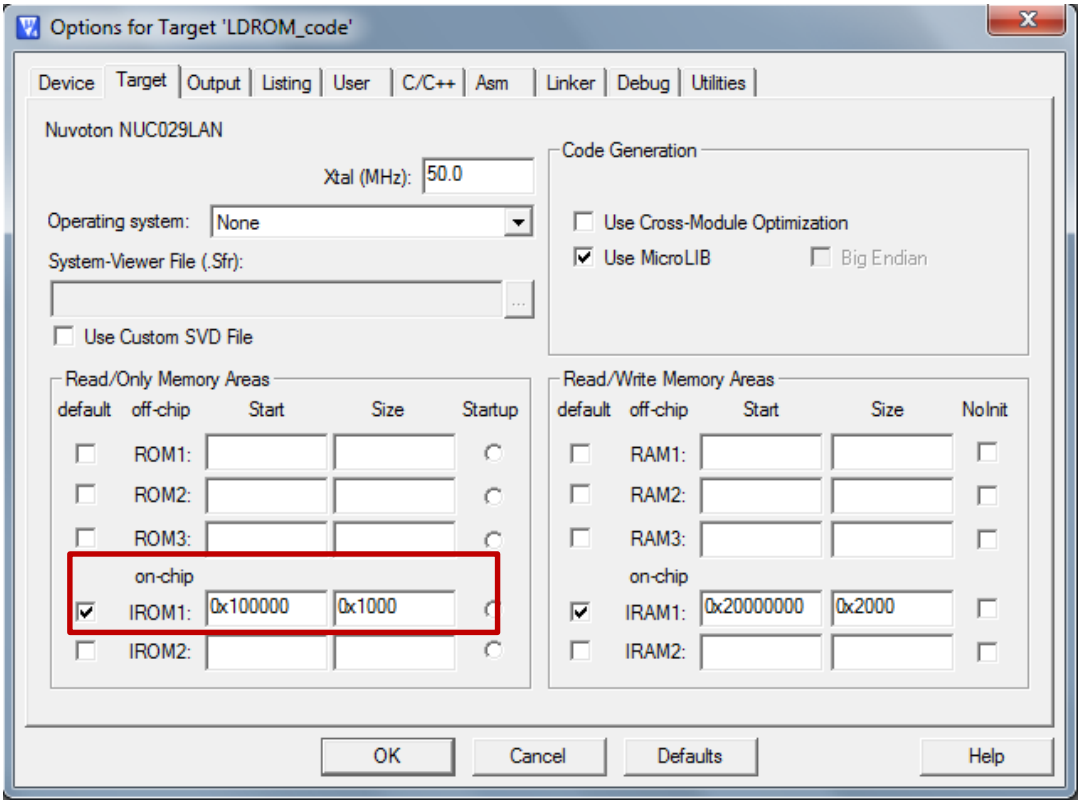
三、 情景三：程序从 LDROM 启动，bootloader 程序小于 4KB，正好放到 LDROM 中。这种情景和 ISP 有点类似，这种情况 keil 有 2 种设定方法：

a) 比较简单，直接使用 Linker frame 中的 R/O base 和 R/W base 设定




b) 跟第一种情况有些类似



### 2.6.7 使能 IAP 后程序下载方法

#### 2.6.7.1 Keil中点击“LOAD”下载应用程序就可以了

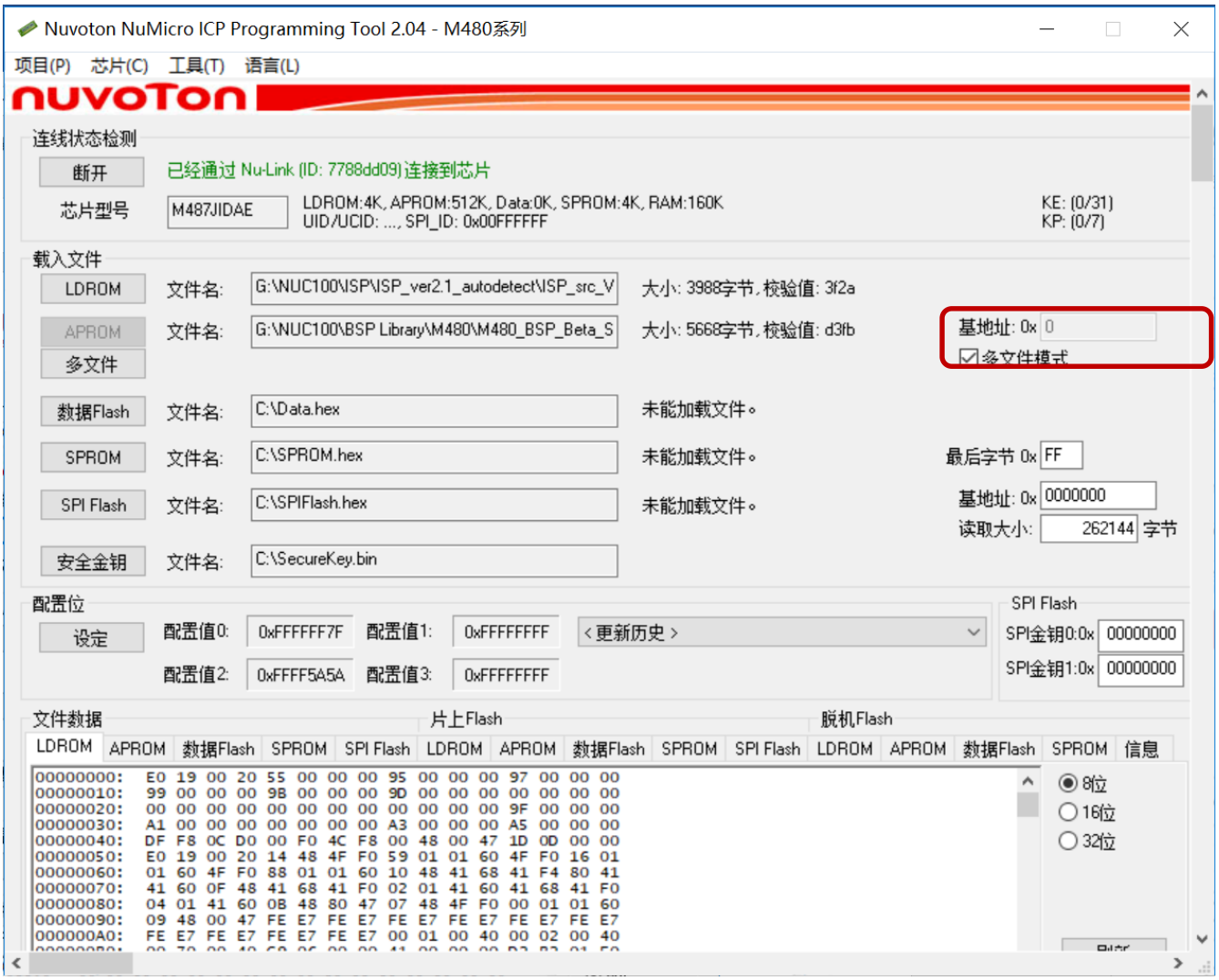
#### 2.6.7.2 通过上位机ISP Tool更新应用程序

下载程序需要填offset地址，这个功能官方ISP Tool上面不知为何没有加。但是新唐ISP 上位机和下位机在github.com/opennuvoton下面全部source code都开放的，大家感兴趣可以自己添加。

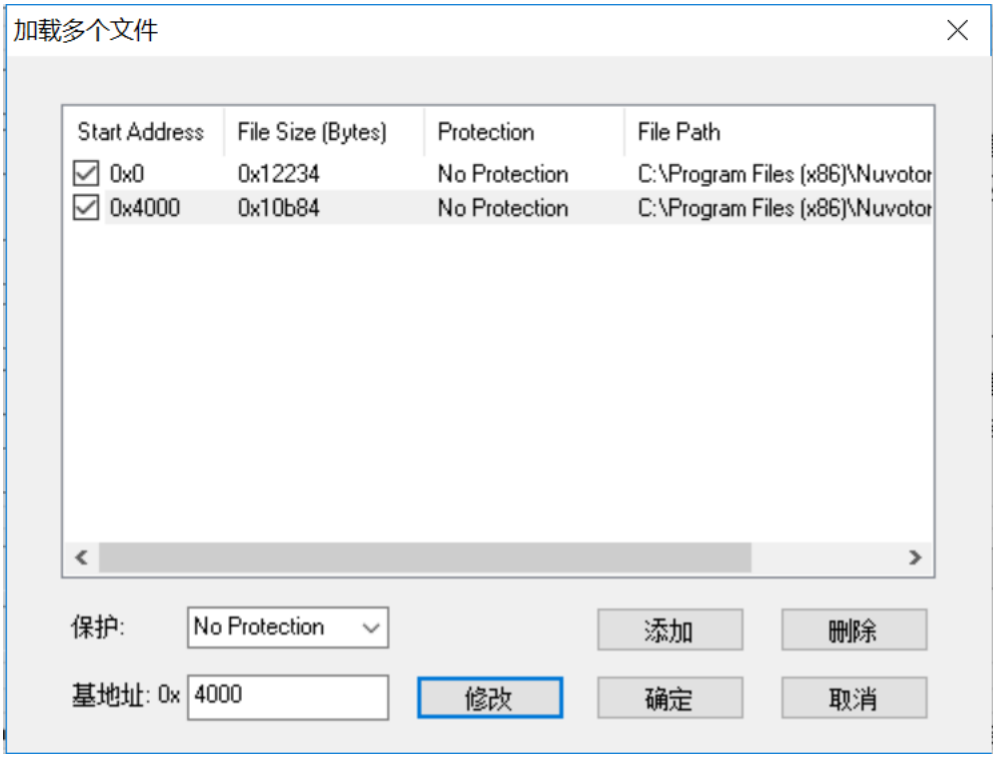
或者用NuMerFileUI.exe转成从0开始的bin档，NuMerFileUI.exe在2.6.7.3节可以找到。

#### 2.6.7.3 通过上位机ICP tool更新应用程序

基地址这里填程序烧入偏移地址。如果APROM中需要烧入多个文件，需要将这个几个文件先链接成一个文件：选中“多文件模式”




点击“多文件”之后，画面如下：




点击“添加”，可以添加多个文件，之后点击“确定”，就可以整合成一个.bin文件。或者提前用下面exe工具，先合并多个bin档成1个bin档

NuMerFileUI.exe

## 2.7 使能 IAP 之后的调试方法

使能IAP之后，运行地址不在0的APP将不能再source level debug,可以用如下的方法解决。

第一步写一个SRAM.ini，内容如下

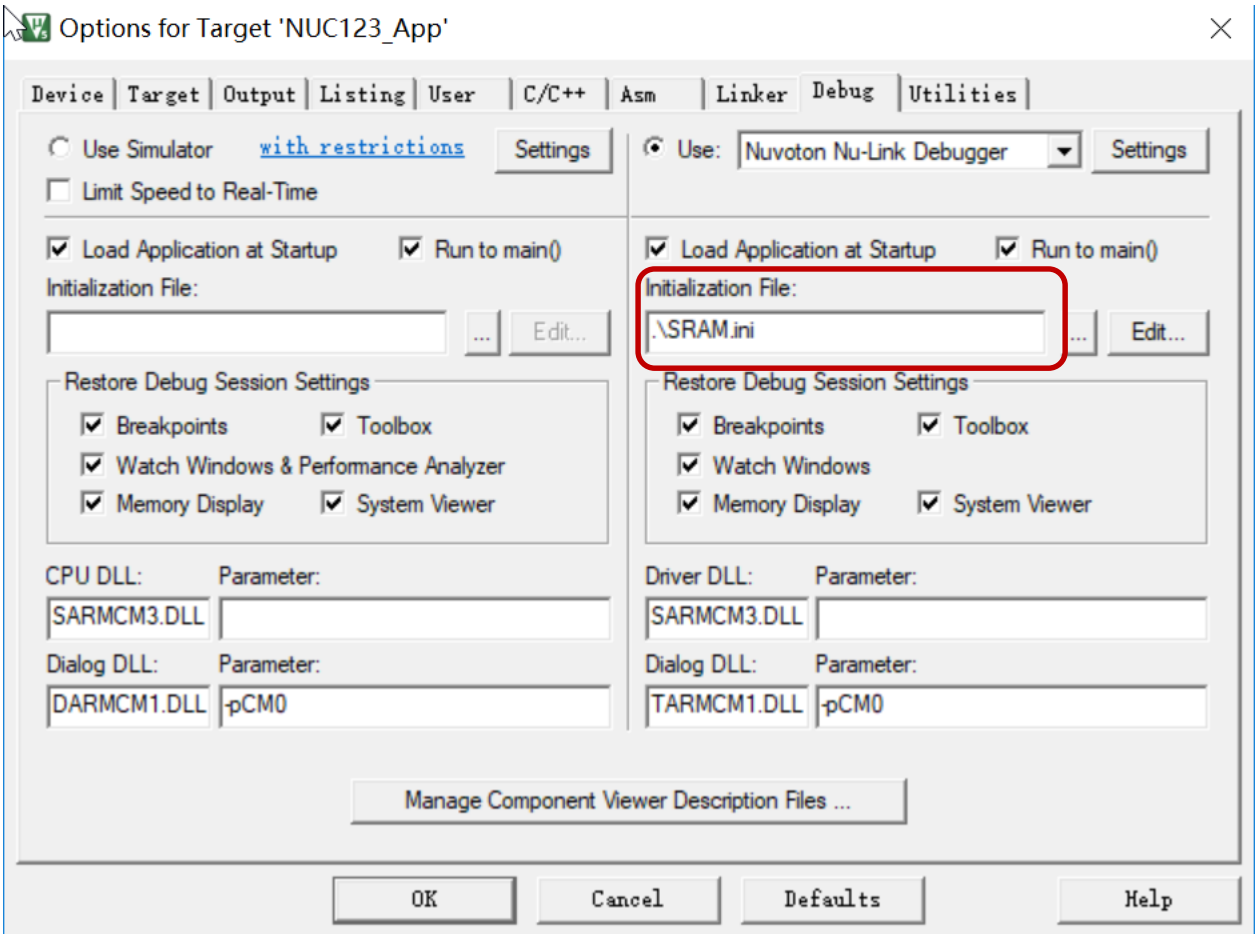
```
/*****************************************************************************/
/* VECMAP function is used to map the specified start address to memory address 0x0000-
0000.        */
/* This initialize file maps the APROM (0x0000-4000) to address 0x0000_0000 through VECMAP
function.
*/*****************************************************************************/
```

```
FUNC void IAPMap(void)
{
 _WDWORD(0xE000ED08, 0x00004000);          /* Specify the load VECMAP address   (reg :
SYS_LVMPADDR) */
 R13 = _RDWORD(0x4000);
 R15 = _RDWORD(0x4004);


}

IAPMap();
```

之后填入下面位置：Options->Debug




然后点击“d”进入debug，就能调试了

## 2.8 案例分析

### 2.8.1 Vector Page Remap 失败

使用FMC下ISP命令Vector remap 总是失败。

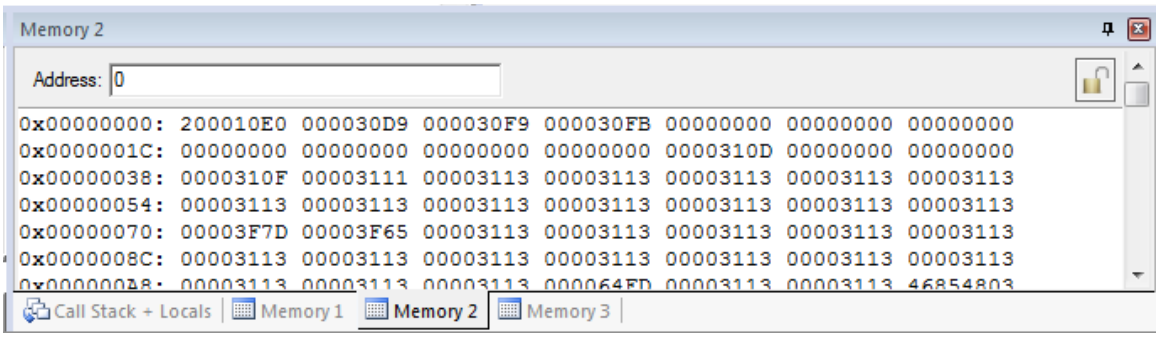
原因：CONFIG0 中CBS的值没有设定。有人可能会反驳：我设了，就在应用程序中，main 函数的前面，修改了CONFIG0 的CBS值。 大家要注意的是，CONFIG0的值需要芯片复位才能起作用，所以在程序中修改之后需要复位才行。所以这个方法一般不推荐，而是推荐大家使用ICP tool或者keil先把CONFIG0的值修改好。详见1.3.1节描述。

检测IAP是否使能的方法为：假设要重新映射到0x3000的位置，调用如下函数重新映射

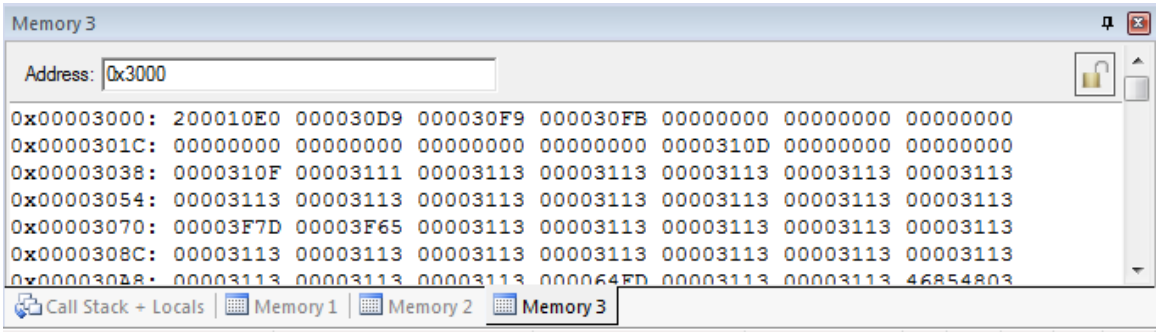
FMC_SetVectorPageAddr(0x00003000);

之后，在keil里面，可以看一下地址0x0和0x3000处的内容是否一致。如果一致，说明IAP使能了，否则就没有使能。

地址0出内容如下：




地址0x3000处内容如下：




大家可以看到内容完全一样。

### 2.8.2 调试发现 Vector Remap 之后，本来正确的下一条要执行的指令，忽然变成其它指令了

原因：这是因为下一条要执行的指令地址在0 ~ 0x200(Page 0)之间，这就导致 0~0x200之间的指令在VECTOR_PAGE_REMAPVECTOR_PAGE_REMAP 命令重新映射之后，原本的指令也不见了，变成了新映射的page中的内容。

解决方法：这就需要Link Script文件了，详见2.3节理解IAP。只把vector table 放到0~0x200之间，其它的程序都放到0x200之后。

# 3 示例代码

## 3.1 不使用复位的方式切程序

```
#include <string.h>
#include "NANO1xx.h"

#define FISPCON          (FMC_BASE+0x000)
#define ISPADR          (FMC_BASE+0x004)
#define ISPDAT          (FMC_BASE+0x008)
#define ISPCMD          (FMC_BASE+0x00C)
#define ISPTRG          (FMC_BASE+0x010)

#define ISPGO            0x01
#define ISPFF            0x00000040

#define ISP_Read         0x00
#define ISP_VecRemap  0x2E           /*!<ISP Command - Vector Page Re-Map */

#define Config0          0x00300000
#define Config1          0x00300004

typedef void (FUNC_PTR)(void);

/* 切堆栈地址 */
__asm spChange(uint32_t _sp)
{
    MSR MSP, r0
    BX lr
}

/* 设定向量表地址 */
int32_t FMC_SetVectorPage(uint32_t u32addr)
{
    unsigned int Reg;

    outp32(ISPCMD, ISP_VecRemap);
    outp32(ISPADR, u32addr);
    outp32(ISPTRG, ISPGO);

     __ISB();

    Reg = inp32(FISPCON);
```

```
        if (Reg & ISPFF)
        {
            outp32(FISPCON, Reg);
            return -1;
        }

    return 0;
}

/* 读 Flash */
int FMC_Read(unsigned int address, unsigned int * data)
{
   unsigned int Reg;

   outp32(ISPCMD, ISP_Read);
   outp32(ISPADR, address);
   outp32(ISPDAT, 0x00000000);
   outp32(ISPTRG, ISPGO);

   __ISB();

   Reg = inp32(FISPCON);
   if (Reg & ISPFF)
   {
       outp32(FISPCON, Reg);
       return -1;
   }

    *data = inp32(ISPDAT);

   return 0;
}

int32_t main()
{
   volatile uint32_t a, b, _sp;
    FUNC_PTR       *func;
    uint32_t uData;

    UNLOCKREG();
    FMC->ISPCON_BITS.ISPEN = 1;

    FMC_Read(Config0, &uData);
```

```
    /*如果没有使能 data flash，默认一直执行 APP1*/
    if(uData & 0x1)
        b = 0x400;
    else
    {
        /*如果使能了 data flash，从 data flash 中读出有效的 APPx 程序*/
        FMC_Read(Config1, &uData);
        FMC_Read(uData, &a); /* 决定执行 App1 还是 App2 */

        /* 取得 App1/App2 的起始地址 */
        if((a == 0xFFFFFFFF) || (a < 0x400))
            b = 0x400;
        else
            b = a;
    }

    //reset all IPs
    outpw(&GCR->IPRST_CTL2, 0xFFFFFFFF);
    outpw(&GCR->IPRST_CTL2, 0);

    /*关闭所有中断*/
    NVIC->ICER[0] = 0xFFFFFFFF;

    /* 设定向量表到地址 b */
    FMC_SetVectorPage(b);
    /* 程序起始地址处存放的就是堆栈的地址 */
    _sp = *(volatile uint32_t *) b;  /* 取得堆栈地址 */

    spChange(_sp); /* 设定堆栈指针 */

    func = (FUNC_PTR *)(*(uint32_t *)(b + 4)); /* 取得复位处理函数地址 */
    func();/* 跳到复位处理函数 */

}
```

## 3.2 使用复位的方式切程序

```
    #include <string.h>
#include "NANO1xx.h"

#define FISPCON         (FMC_BASE+0x000)
#define ISPADR         (FMC_BASE+0x004)
#define ISPDAT         (FMC_BASE+0x008)
```

```
#define ISPCMD          (FMC_BASE+0x00C)
#define ISPTRG          (FMC_BASE+0x010)

#define ISPGO            0x01
#define ISPFF            0x00000040

#define ISP_Read         0x00
#define ISP_VecRemap  0x2E          /*!<ISP Command - Vector Page Re-Map */

#define Config0          0x00300000
#define Config1          0x00300004

typedef void (FUNC_PTR)(void);

/* 设定向量表地址 */
int32_t FMC_SetVectorPage(uint32_t u32addr)
{
    unsigned int Reg;

    outp32(ISPCMD, ISP_VecRemap);
    outp32(ISPADR, u32addr);
    outp32(ISPTRG, ISPGO);

     __ISB();

    Reg = inp32(FISPCON);
    if (Reg & ISPFF)
    {
       outp32(FISPCON, Reg);
       return -1;
    }

    return 0;
}

/* 读 Flash */
int FMC_Read(unsigned int address, unsigned int * data)
{
   unsigned int Reg;

   outp32(ISPCMD, ISP_Read);
   outp32(ISPADR, address);
   outp32(ISPDAT, 0x00000000);
    outp32(ISPTRG, ISPGO);
```

```
    __ISB();

    Reg = inp32(FISPCON);
    if (Reg & ISPFF)
    {
        outp32(FISPCON, Reg);
        return -1;
    }

    *data = inp32(ISPDAT);

    return 0;
}

int32_t main()
{
    volatile uint32_t a, b, _sp;
    FUNC_PTR       *func;
    uint32_t uData;

    UNLOCKREG();
    FMC->ISPCON_BITS.ISPEN = 1;

    FMC_Read(Config0, &uData);

    /*如果没有使能 data flash，默认一直执行 APP1*/
    if(uData & 0x1)
        b = 0x400;
    else
    {
        /*如果使能了 data flash，从 data flash 中读出有效的 APPx 程序*/
        FMC_Read(Config1, &uData);
        FMC_Read(uData, &a); /* 决定执行 App1 还是 App2 */

        /* 取得 App1/App2 的起始地址 */
        if((a == 0xFFFFFFFF) || (a < 0x400))
            b = 0x400;
        else
            b = a;
    }

    /*关闭所有中断*/
    NVIC->ICER[0] = 0xFFFFFFFF;
```

```
    /* 设定向量表到地址 b */
    FMC_SetVectorPage(b);
    /*系统复位 */
     NVIC_SystemReset();
}
```

## Revision History

| Rev. | Date | Description |
|---|---|---|
| 1.00 | 3-21-2014 | Initially issued. |
| 1.10 | 7-7-2014 | 添加详细说明；ISP与IAP的区别；支持SystemReset方式切程序的Chip列表；案列分析 |
| 1.11 | 11-15-2014 | 重新整理章节，让大家更容易懂 |
| 1.20 | 2-15-2015 | 添加名词解释一章 |

## Important Notice

**Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".**
**Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.**
**All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.**

**Please note that all data and specifications are subject to change without notice. All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.**