|  |  |
| --- | --- |
| **Document Information** | |
| **Abstract** | 该文档介绍IAP的概念和用法，以及ISP与IAP的区别. |
| **Apply to** | 任何支持IAP的Cortex-m0系列. |

|  |  |
| --- | --- |
| **Table of Contents** | |
|  | [1 介绍 3](#_Toc392520256)  [1.1 寄存器 3](#_Toc392520257)  [1.2 理解IAP 3](#_Toc392520258)  [1.3 ISP与IAP的区别 4](#_Toc392520259)  [1.4 向量表 6](#_Toc392520260)  [1.5 使用方法 6](#_Toc392520261)  [1.5.1 IAP使用步骤 7](#_Toc392520262)  [1.5.2 使能IAP 7](#_Toc392520263)  [1.5.3 将应用程序编译到指定地址 14](#_Toc392520264)  [1.5.4 下载 22](#_Toc392520265)  [1.6 案例分析 22](#_Toc392520266)  [1.6.1 Vector Page Remap 失败 22](#_Toc392520267)  [1.6.2 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 22](#_Toc392520268)  [2 示例代码 23](#_Toc392520269)  [2.1 不使用复位的方式切程序 23](#_Toc392520270)  [2.2 使用复位的方式切程序 25](#_Toc392520271) |

# 介绍

新唐的m0系列大都支持IAP功能。IAP就是几块ROM中的程序/函数可以互相调用、切换。通俗一点就是APROM中的程序可以调用LDROM中的函数，LDROM中的程序可以调用APROM中的函数。对CPU来说它们都是同时可读的。同时，客户的应用程序可以放到APROM/LDROM的任何地址。下面为大家介绍一下使用的细节

## 寄存器

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指定向量表必须放到地址0的地方。如果程序放到地址非0的位置，向量表也就不在地址0的地方了，所以需要重新映射向量表。

## 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，这样用户的函数就可以放到APROM/LDROM的任何地方。
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。

Item1只要使能IAP功能，CPU就能“看”到所有的APROM和LDROM空间了。

Item2就涉及到向量表映射，就是Vector Page Remap命令。按照cotex-m0的架构，向量表必须要放到地址0的地方。如果客户将应用程序放到了地址非0的地方，例如：0x1000，则向量表就在0x1000的地方了，这时候如果发生中断，CPU从地址0的位置拿到的中断处理函数的地址就不正确了。这时候就需要用Vector Page Remap命令将地址0x1000位置的1个page映射到地址0。1个page大小就是512B。意思就是Vector Page Remap命令将0x1000 – 0x1200映射到了0x0 – 0x200。**这个动作会导致一个副作用**，就是地址0x1000 – 0x1200地方的代码除了向量表还有其他的代码，而这在IAP下是不允许的。怎样将在地址0x1000 – 0x1200的代码移到0x1200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

/\* 说明SRAM的基地址和SIZE ，并且将RW和ZI Data放到SRAM中\*/

RAM1 0x20000000 0x10000 { ; load region size\_region

RW\_IRAM1 +0 { ; RW data

.ANY (+RW +ZI)

}

}

/\* 说明应用程序基地址，也就是向量表RESET段的地址，如果不是0x00000000 ，例如：是0x1000，这里就填0x1000 \*/

ROM1 0x00001000

{

IROM1 +0 { ; load address = execution address

\*.o (RESET, +First)

\*.o (HANDLER)

\*(InRoot$$Sections)

}

}

/\* 其它程序都放到0x1200后面 \*/

FLASH1 0x00001200

{

FLASH1 0x00000200

{

.ANY (+RO)

}

}

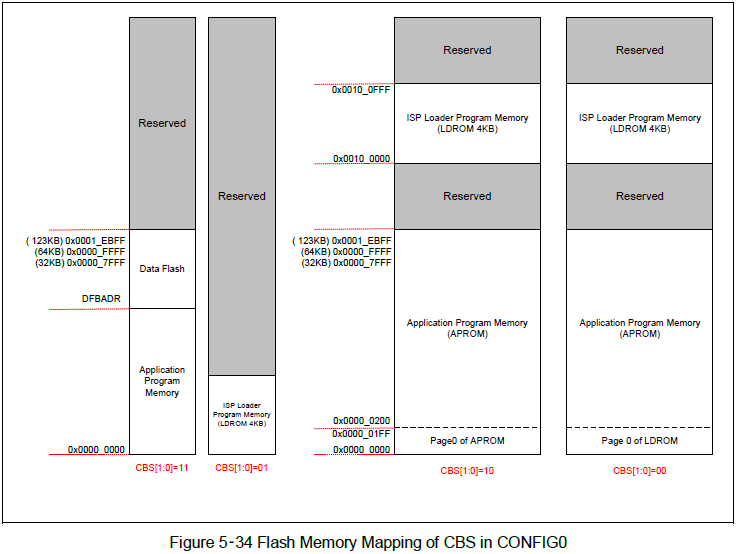
## ISP与IAP的区别

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 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中的函数 | LDROM中的程序可以调用APROM中的函数 | APROM与LDROM中的程序切换，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0x0，0x1000，0x4000, 0x100000(LDROM)。ISP只能放到地址0x0执行，如果系统从APROM启动，APROM就在地址0x0的地方；如果系统从LDROM启动，LDROM就在地址0x0的地方。但是如果IAP使能，APROM的地址就固定在0x0，LDROM的地址就固定在0x100000.

APROM中的程序更新之后，ISP需要复位系统，然系统重新启动

IAP则不需要，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了。



IAP关闭

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

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0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

## 向量表

向量表0的位置为**栈地址，栈地址将用来初始化R13寄存器的值**

向量表1的位置为复位中断处理函数，就是**程序的入口**

## 注意事项

**不能在中断里面切vector table，并跳转去另一个APP**

## 使用方法

我们的m0 IAP有两种使用方法，一种支持SystemReset或者不Reset的方式切换程序，一种只支持不Reset的方式切换程序。这里的Reset推荐使用SystemReset，而不是CPU reset。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。其它的Reset例如：Chip reset, BOD reset, WDT reset，/RESET引脚等都不能用于IAP中程序切换。

详细支持列表如下：

|  |  |  |
| --- | --- | --- |
|  | 支持SystemReset切程序 | 不支持Reset的方式切程序 |
| NUC200/NUC220/NUC230/NUC240 | Yes | Yes |
| M051xxxDN/ M051xxxDE | Yes | Yes |
| M051xxxBN | Yes | Yes |
| NUC131 | Yes | Yes |
| NUC130CN/NUC140CN | No | Yes |
| NUC123 | No | Yes |
| M058S | No | Yes |
| NANO100/NANO120/NANO130/NANO140BN | No | Yes |
| NANO100/NANO120/NANO130/NANO140AN | No | Yes |
| NANO102/NANO112 | Yes | Yes |
| Mini5xAN/Mini5xBN | No | Yes |
| Mini5xDE | Yes | Yes |
| NUC029 | No | Yes |
| NUC442/NUC470/NUC451 | Yes | Yes |

### IAP使用步骤

1. 使能IAP功能
2. 编写程序，包括Bootloader和APP
3. 分别下载Bootloader和App
4. 重新上电

**不复位方式Bootloader切APP，或者两个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);

1. 关闭所有中断

NVIC->ICER[0] = 0xFFFFFFFF;

1. 调用函数FMC\_SetVectorPageAddr重新映射向量表
2. spChange切栈地址
3. 跳转

**复位方式Bootloader切APP，或者两个APP之间切换的步骤**

1. 关闭所有中断

NVIC->ICER[0] = 0xFFFFFFFF;

1. 调用函数FMC\_SetVectorPageAddr重新映射向量表
2. 系统复位

NVIC\_SystemReset();

### 使能IAP

#### 使用ICP

1. 打开ICP tool，并连接到目标板

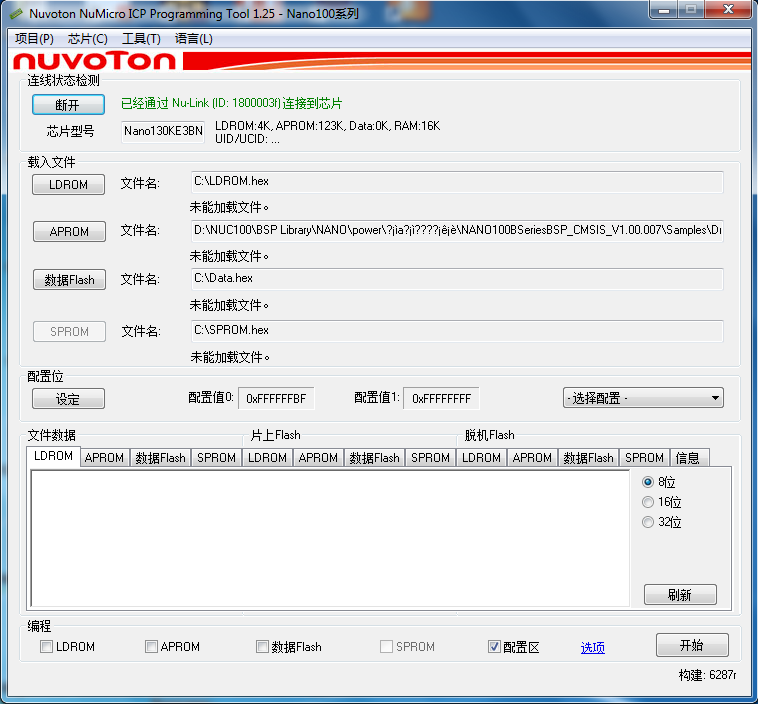
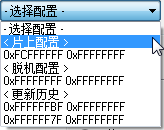


Figure 1

共5步即可实现使能IAP功能

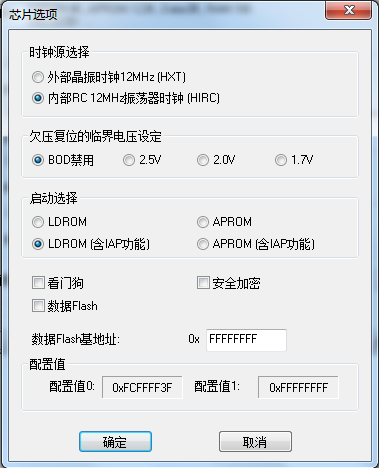
1. 点击选择配置下拉框，选择片上配置



1. 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

1. 选择配置区



1. 点击选项



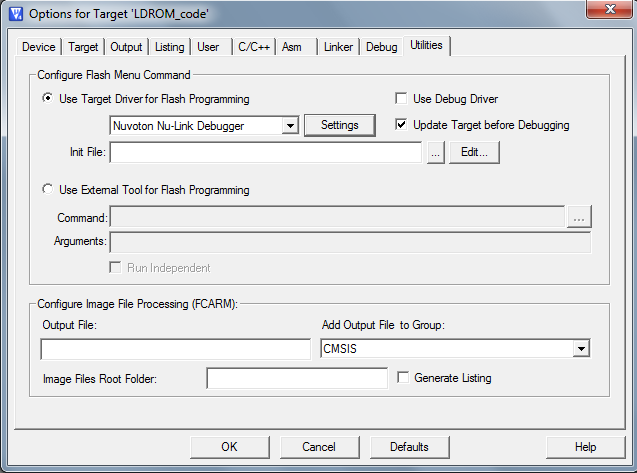
将红色方框中的相应项打勾

1. 最后点击“开始”，烧录成功之后，IAP就使能了

#### 使用keil打开IAP功能

Keil下选择Target Options ->Utilities->Settings

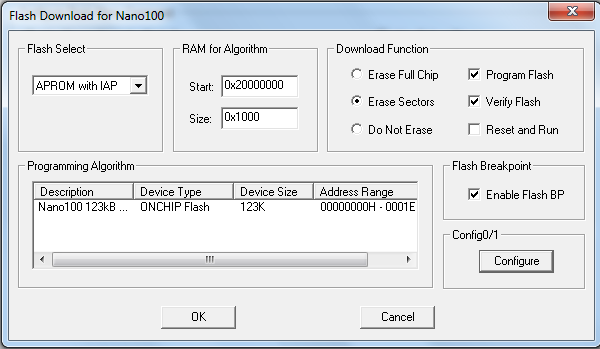
第一步：



点击“Settings”将出现下面的画面

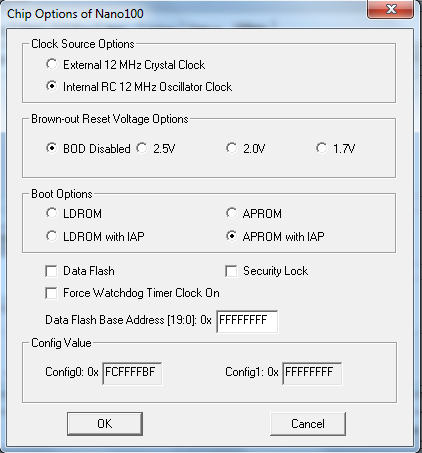
第二步：

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



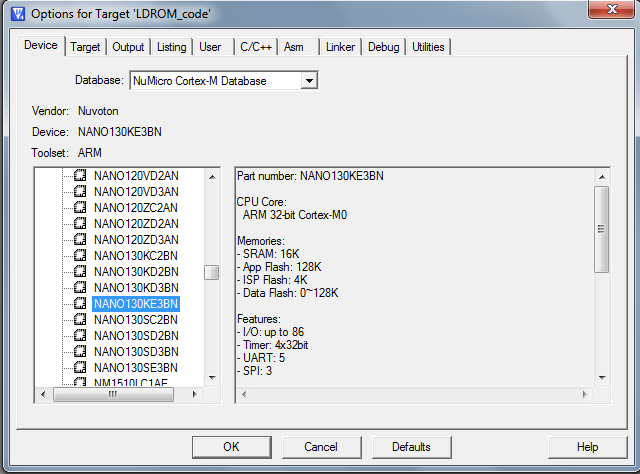
下面的画面选择“APROM with IAP”或者“LDROM with IAP”的功能，但是要跟上面的画面选择的一致

第三步：

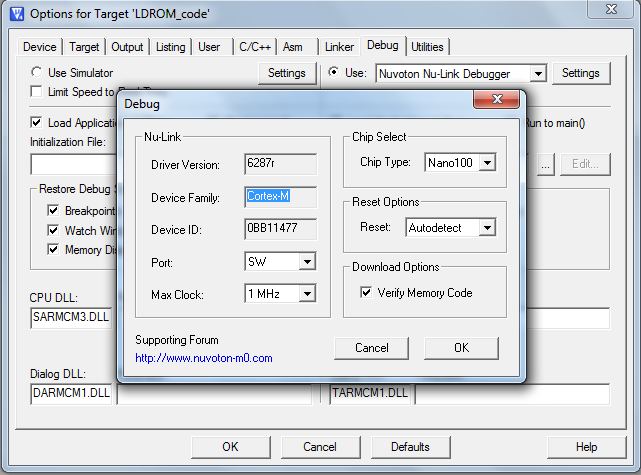


如果发现第二步和第三步没有with IAP的选项，应该是Nu-Link接的目标板和你Device中以及Debug中说明的类型不一致导致的。

例如：这里选择NANO130KE3BN



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



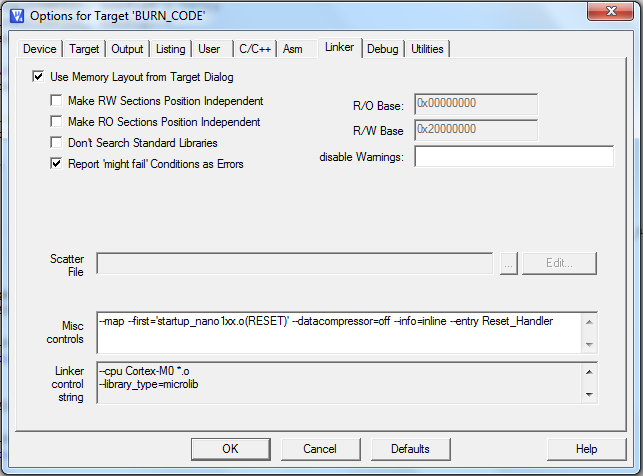
并且连接的目标板也要是NANO130KE3BN，这样以后，如果该系列支持IAP，关于IAP的选项就会出现

### 将应用程序编译到指定地址

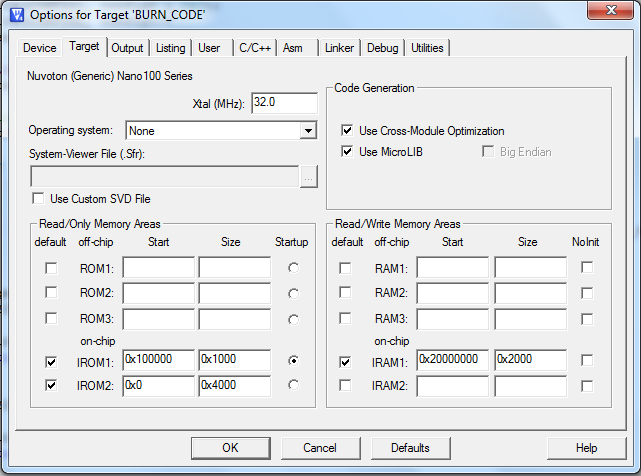
假设APROM 的地址为 0 ~ Addr1，LDROM的地址为0x100000 ~ Addr2。Addr1和Addr2将随APROM和LDROM的大小不同而不同。到目前为止LDROM大都是4K的，APROM最大到128K。

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设定方法会有所不同。IAP使能的方法，请看1.3.1节.**

1. 第一种情况：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。
2. 选择使用Target Dialog



1. 填Target Dialog



红色方框内的数据表明，共有两块ROM，一块4KB，另一块16M， 系统从0x100000启动。

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0，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。

1. 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。Mapping如下：

Boot code

APROM

0

App1

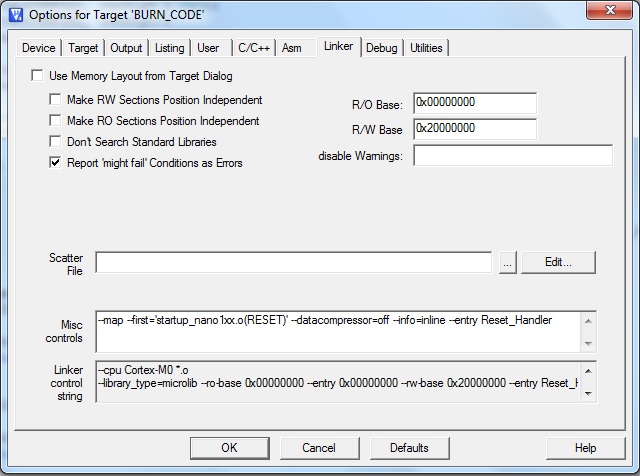
App2

0x400

0xC800

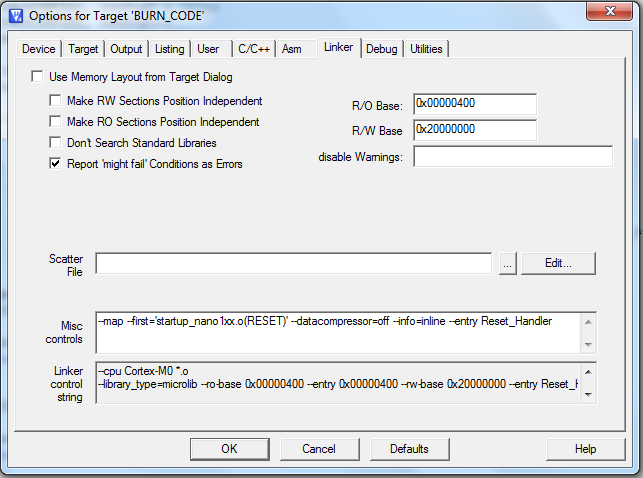
Keil设定如下：

1. Bootloader的设定



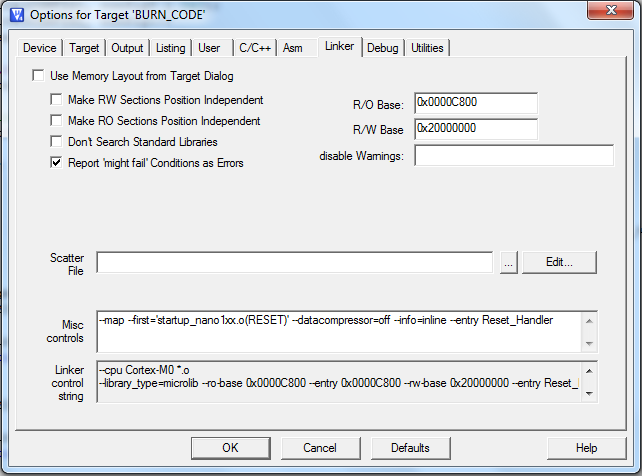
程序执行地址为0x0

1. App1的设定



程序执行地址为0x400

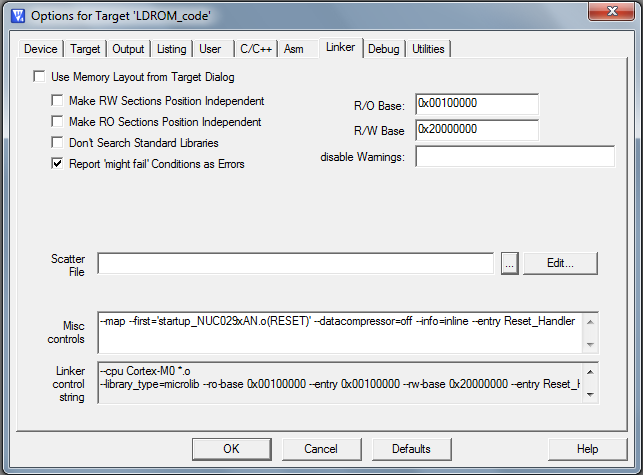
1. App2的设定



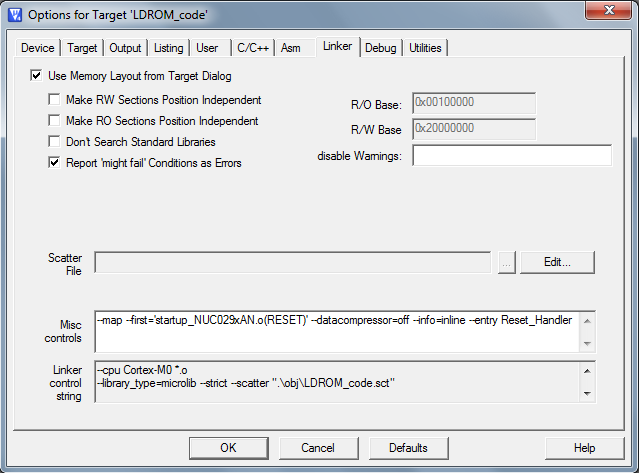
程序执行地址为0xC800

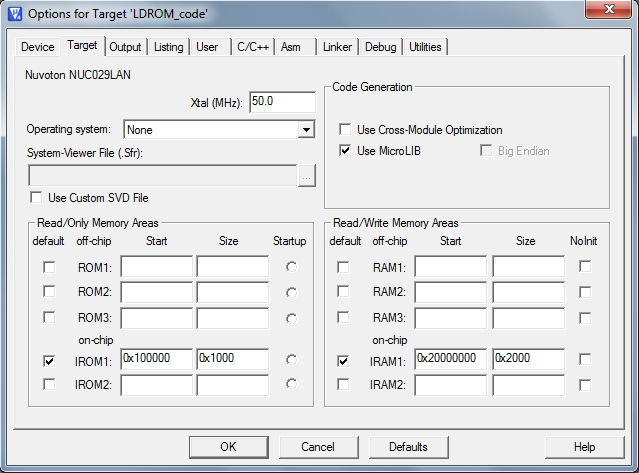
Bootloader起来之后(向量表由硬件自动映射到APROM中，也就是地址0x00的地方)如果要跳去执行用户程序，需要先执行Vector Page Remap 命令，将vector table地址切到App1/App2的地址，也就是地址0x400/0xC800，然后设定堆栈地址，之后就可以跳转了。示例代码可以见文档最后。

1. 第三种情况：程序从LDROM启动，bootloader程序小于4KB，正好放到LDROM中。这种情况keil有2种设定方法
2. 比较简单，直接使用Linker frame中的R/O base和R/W base设定



1. 跟第一种情况有些类似





### 下载

Keil中点击“LOAD”将Bootloader和APP分别下载就可以了

## 案例分析

### 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

### 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(page0)之间，这就导致0~0x200之间被Vector 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：IAP的程序需要Link Script文件，详见1.2节。只把vector table 放到0~0x200之间，其它的程序都放到0x200之后。

# 示例代码

## 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 \*/

}

## 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 |
|  |  |  |

**Important Notice**

**Nuvoton products are not designed, intended, authorized or warranted for use as components in systems or equipment intended for surgical implantation, atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications wherein failure of Nuvoton products could result or lead to a situation wherein personal injury, death or severe property or environmental damage could occur.**

**Nuvoton customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from such improper use or sales.**

**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.**