

【经验分享】KL25 在 IAR 中将函数指定到 flash 地址

一，经验分享描述

在之前的经验分享中写了些在 KE02 下，CW，IAR 以及 keil 的编译环境中，如何定义 constant 到指定的 flash 地址。但是实际上，大家在使用我们 kinetis 的过程中，可能也希望能够灵活的将某个函数直接定义到指定的 flash 地址，这样，如果改变这个函数，实际上，只需要改变函数所在的 flash，而不需要更新所有的 flash。所以，为了方便大家操作，我们论坛里已经推出了在 CW 下，如何指定函数到具体的 flash 地址，本文就讲解下，在 IAR 的环境下如何实现指定函数到具体的 flash 地址。

二，经验分享 IAR 环境下实现

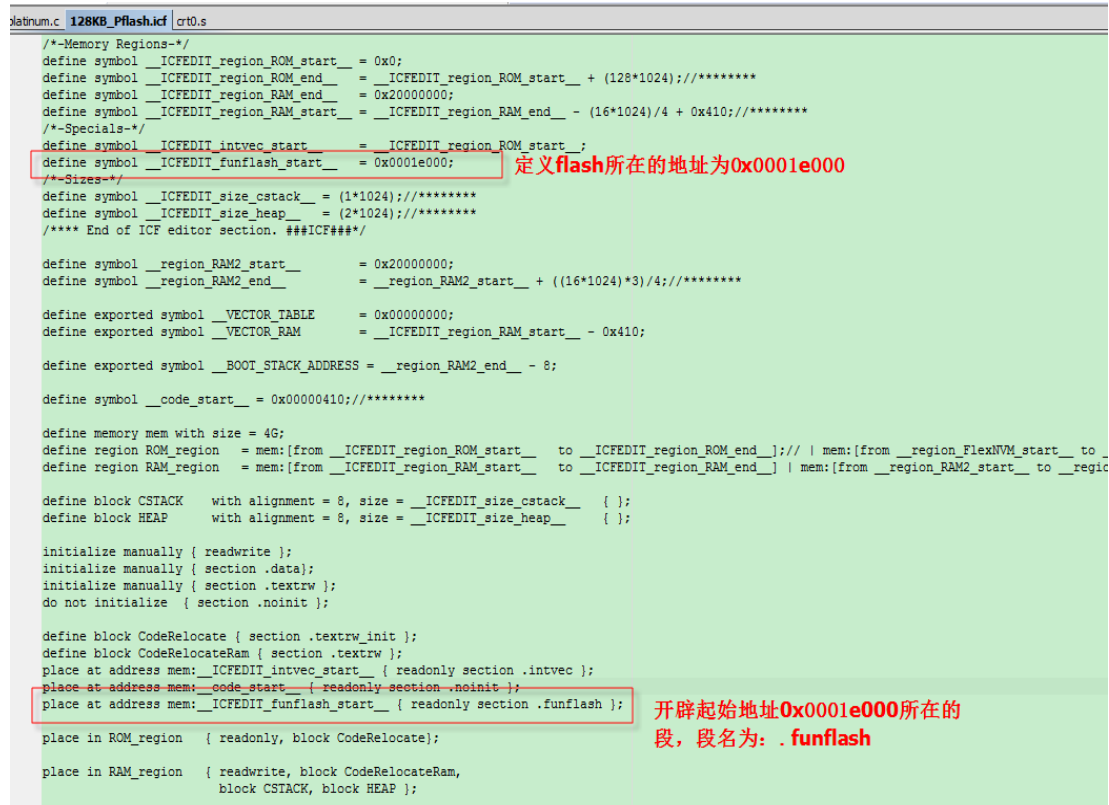
1. 在.icf 文件中定义函数要存放的地址

可以使用如下格式定义这个地址：

place at address mem: (flash 地址) { readonly section. (段名) };

例如，在本次试验中，希望将函数定义到 flash 地址：0x0001E000 这个地址，靠近 flash 空间的尾部。

实际我们的定义如下图：



```
/*-Memory Regions-*/
define symbol _ICFEDIT_region_ROM_start_ = 0x0;
define symbol _ICFEDIT_region_ROM_end_ = _ICFEDIT_region_ROM_start_ + (128*1024);/******
define symbol _ICFEDIT_region_RAM_end_ = 0x20000000;
define symbol _ICFEDIT_region_RAM_start_ = _ICFEDIT_region_RAM_end_ - (16*1024)/4 + 0x410;/******
/*-Specials-*/
define symbol _ICFEDIT_intvec_start_ = _ICFEDIT_region_ROM_start_;
define symbol _ICFEDIT_funflash_start_ = 0x0001e000; 定义flash所在的地址为0x0001e000
/*-Sizes-*/
define symbol _ICFEDIT_size_cstack_ = (1*1024);/******
define symbol _ICFEDIT_size_heap_ = (2*1024);/******
/***** End of ICF editor section. ###ICF###*/

define symbol __region_RAM2_start__ = 0x20000000;
define symbol __region_RAM2_end__ = __region_RAM2_start__ + ((16*1024)*3)/4;/******

define exported symbol __VECTOR_TABLE = 0x00000000;
define exported symbol __VECTOR_RAM = _ICFEDIT_region_RAM_start_ - 0x410;

define exported symbol __BOOT_STACK_ADDRESS = __region_RAM2_end_ - 8;

define symbol __code_start__ = 0x00000410;/******

define memory mem with size = 4G;
define region ROM_region = mem:[from __ICFEDIT_region_ROM_start__ to __ICFEDIT_region_ROM_end__];/* | mem:[from __region_FlexNVM_start__ to __region_FlexNVM_end__]
define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__ to __ICFEDIT_region_RAM_end__] | mem:[from __region_RAM2_start__ to __region_RAM2_end__];

define block CSTACK with alignment = 8, size = _ICFEDIT_size_cstack_ { };
define block HEAP with alignment = 8, size = _ICFEDIT_size_heap_ { };

initialize manually { readonly };
initialize manually { section .data };
initialize manually { section .text };
do not initialize { section .noinit };

define block CodeRelocate { section .text; };
define block CodeRelocateRam { section .text; };
place at address mem: _ICFEDIT_intvec_start_ { readonly section .intvec };
place at address mem: __code_start__ { readonly section .noinit };
place at address mem: _ICFEDIT_funflash_start_ { readonly section .funflash }; 开辟起始地址0x0001e000所在的段，段名为: .funflash

place in ROM_region { readonly, block CodeRelocate };

place in RAM_region { readwrite, block CodeRelocateRam, block CSTACK, block HEAP };
```

当然，实际上也可以定义为：

```
place at address mem: 0x0001e000 { readonly section .funflash };
```

2. 在主函数中定义函数

接下来要做的就是将所定义的函数放到 1 里面所定义的 flash 地址段中。

举例定义如下：

```
char funcInROM(int flag) @ ".funflash"
{
    if (flag > 0)
    {
        return 1;
    }
    return 0;
}
```

此时，这个函数就直接放到了 0x001e000 地址。

3. 测试结果

本次试验的平台是 FRDM_KL25，所使用的 IAR 平台版本为：7.20.2。代码修改是基于 KL25_SC 的 platinum 工程上修改的。

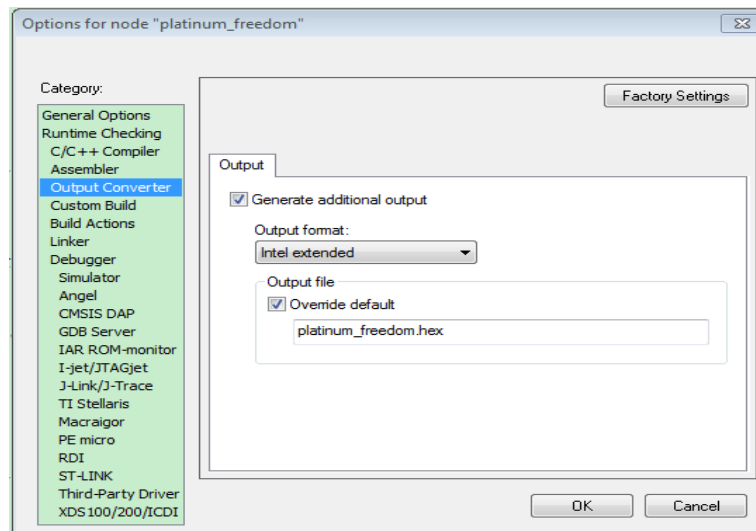
测试结果分两部分进行，第一部分是在进入 debug 的状态下，查看 memory。

第二部分是在生成的 hex 文件中查看。

在测试之前，首先讲一下 IARhex 文件生成配置，以及 hex 文件格式。

(1) IAR 环境下 hex 生成配置

在工程中选择 option-> output converter，勾选 generate additional output,选择 output format 为 intel extended，然后勾选 override default,写入你要生成的 hex 文件名，点击 ok。如下图：



(2) hex 文件格式

文件格式为:

起始符 “:”	数据长度 (1B)	存储地址 (2B)	数据类型 (1B)	数据 (16B)	校验码 (1B)
---------	-----------	-----------	-----------	----------	----------

这里举个例子:

:1000C000FFFFFFFFFFFFFFFFFFFFFFFFFFFF40

: :表示一行的开始

10: 表示这一行包含的数据长度, 一个字节

00C0: 表示存储的地址, 两个字节, 这里是在 flash 的地址 0x000000c0.

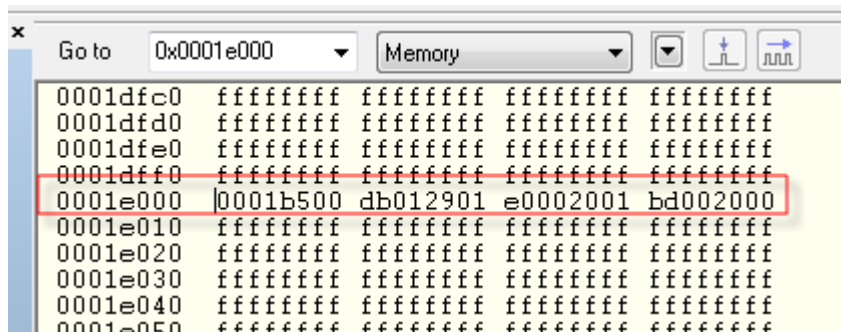
00: 表示数据类型, 一个字节, 00 为数据记录; 01 文件结束记录; 02 扩展段地址记录; 04 扩展线性地址记录。

FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: 代表具体的数据, 这里数据都是 FF, 16 个字节。数据的存储是高位地址在前, 低位地址在后。

40: 表示校验码, 一个字节。校验码是 0x100-这一行数据总和的低字节, 然后取低一个字节。

(3) debug 的 memory 查看测试

进入 debug 后, 调出 memory 窗口, 方法为: view->memory, 然后在 goto 中输入要查询的 flash 地址: 0x0001e000, 然后按下 enter 键。结果如下:



Goto	0x0001e000	Memory			
0001dfc0	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
0001dfd0	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
0001dfe0	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
0001dff0	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
0001e000	0001b500	db012901	e0002001	bd002000	
0001e010	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
0001e020	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
0001e030	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
0001e040	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff
0001e050	ffffffff	ffffffff	ffffffff	ffffffff	ffffffff

可以看到在 0x0001e000 这个地址的数据为:

0001b500 db012901 e0002001 bd002000

(4) 查看对应的 hex 文件。

编译工程之后, Hex 文件在工程的 build\iar\platinum\FLASH_128KB\Exe 中, 打开后可以看到具体的 hex 内容, 找到地址为 E000 的地方, 可以看到结果如下:

```

451 :101C20003634204B420A0D0053696C69636F6E2095
452 :101C30007265762025640A0D200000000A0D4D44CF
453 :101C40004D2D415020526573657400003136706926
454 :101C50006E202020202020200000000032347069F7
455 :101C60006E20202020202020000000003130307025
456 :101C7000696E202020202020000000004C4C5320C2
457 :101C80006578697420000000564C4C53302065780C
458 :101C900069742000564C4C533120657869742000DB
459 :101CA000564C4C533220657869740000564C4C53A6
460 :101CB0003320657869742000333270696E202020EB
461 :101CC00020202000343870696E2020202020200041
462 :101CD000363470696E202020202020003830706952
463 :101CE0006E20202020202000C046C046C046C046AE
464 :101CF000FFF7EEFE0A0D0A0D000000000A0D4B4C26
465 :101D0000300000000A0D4B4C310000000A0D4B4C16
466 :101D1000320000000A0D4B4C330000000A0D4B4C02
467 :041D20003400000008B
468 :020000040001F9
469 :10E000000B50100012901DB012000E0002000BD76
470 :0400000500001CE9F2
471 :00000001FF
472

```

即结果为：

00B50100 012901DB 012000E0 002000BD

和窗口 memory 中的数据：

0001b500 db012901 e0002001 bd002000

相比 hex 的数据正好是高位地址在前，低位地址在后。

其实这一串数据就是上面定义的 `char funcInROM(int flag)` 的十六进制码，可以看到确实是在我们指定的 flash 地址中，实现了 IAR 环境下，函数的 flash 地址指定。

三， 附件

附件提供了本文的 PDF，以及测试代码。