【经验分享】KL25 在 MDK 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 以及 IAR 下，如何指定函数到具体的 flash 地址，本文就讲解下，在 MDK 的环境下如何实现指定函数到具体的 flash 地址。

二， 经验分享在 MDK 环境下实现

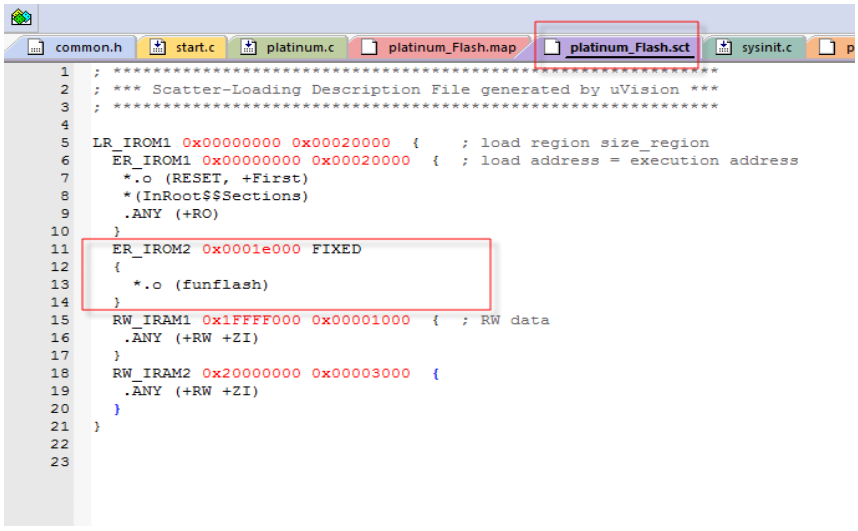
该经验分享要在 MDK 中实现函数的绝对地址指定，一共有两点需要注意：scatter 文件中函数绝对地址的开辟以及程序中将函数定义到定义的绝对地址处。下面来仔细讲解这两点

1．在.sct 文件中定义函数要存放的地址

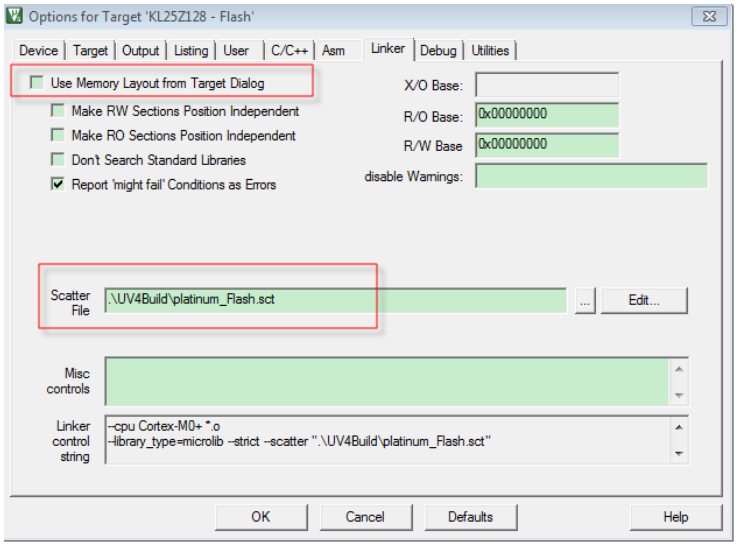
.sct 文件可以在工程的 UV4Build 文件夹中找到，找到后可以拖进 MDK 界面中即可以打开，可以使用如下格式定义这个地址：

```
ER_IROM2 绝对地址 FIXED
{
    *.o (段地址名)
}
```

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：



另外，每次编译之后，编译器都会重新生成.sct 文件，所以为了避免写的代码被刷掉，需要做一些配置，禁止使用编译器每次生成的.sct 文件，而是使用指定的.sct 文件。修改方法：project->options for target->linker, 不勾选 use memory layout from target dialog. 如下图所示：



这样，你每次修改后的 platinum_flash.sct 在编译之后就不会被改变掉了。

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

此时，这个函数就直接放到了 0x001e000 地址。

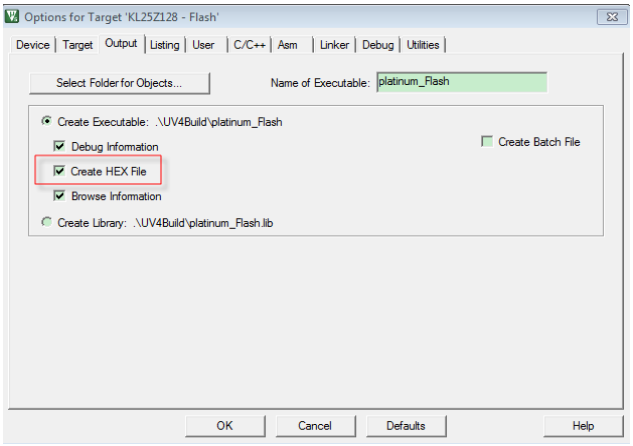
3，测试结果

本次试验的平台是 FRDM_KL25，所使用的 MDK 平台版本为：V5.10.0.2。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 MDK hex 文件生成配置，以及 hex 文件格式。

（1） MDK 环境下 hex 生成配置

在工程中选择 project->options for target->output,勾选 Create HEX file,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：

:1000C000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF40

: :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.

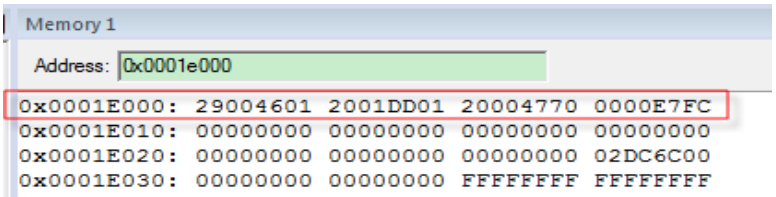
00：表示数据类型，一个字节，00 为数据记录；01 文件结束记录；02 扩展段地址记录；04 扩展线性地址记录。

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

编译后，进入 debug，调出 memory 窗口，方法为：view->memory windows->memory 1, 然后在 address 中输入要查询的 flash 地址：0x0001e000，然后按下 enter 键。结果如下：



可以看到在 0x0001e000 这个地址的数据为：

29004601 2001DD01 20004770 0000E7FC

（4）查看对应的 hex 文件。

编译工程之后，Hex 文件在工程的 build\keil\platinum\UV4Build 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：

```
:10DFC000000000000000000000000000000000000051
:10DFD000000000000000000000000000000000000041
:10DFE000000000000000000000000000000000000031
:10DFF000000000000000000000000000000000000021
:10E000000146002901DD012070470020FCE70000E7
:10E010000000000000000000000000000000000000
:10E020000000000000000000000000006CDC02A6
:08E03000000000000000E8
:0400000500000411E2
:00000001FF
```

即结果为：

01460029 01DD0120 70470020 FCE70000

和窗口 memory 中的数据：

29004601 2001DD01 20004770 0000E7FC

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。

如果屏蔽掉主函数的地址指定：

char funcInROM(int flag)__attribute__((section("funflash")));

会发现，实际上这个函数会被编译器分配到 hex 的 0x0424 处，而 0XE000 处没有相关的函数代码。

```
64 :1003E000000000000000000000000000000000000D
65 :1003F000000000000000000000000000000000000FD
66 :10040000FFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFD
67 :1004100003488546 01F0AAFB00480047330400006A
68 :10042000F82F0020 0146002901DD012070470020 3F
69 :10043000FCE700F0 56F907A001F0CCFB6420FFF7C1
70 :10044000F1FF05E000F018F80446204600F019F826
71 :10045000F8E700000A0D52756E6E696E67207468C9
72 :10046000 6520706C6174696E756D2070726F6A655D
73 :1004700063742E0A0D00000010B5084801F0D5FA8B
```

三， 附件

附件提供了本文的 PDF，以及测试代码。