STM32H750 上使用 PCROP 后导致 Hard Fault

关键字：PCROP，Hard Fault，STM32H750

1. 引言

PCROP 全称为 Proprietary code read out protection（专用代码保护），它提供了一种新的代码保护机制，在 PCROP 区域的内容只能为可执行，不能读取或写入。这种机制可以为 OEM 厂商提供保护，方便保护自己 IP 的代码。本文主要记录在使用 PCROP 上遇到的 Hardfault 问题。

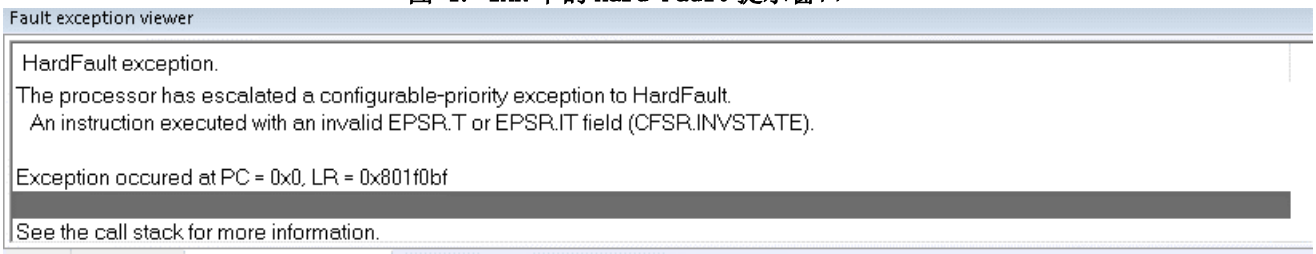
如何使用 PCROP 呢?
参考 AN4968 中 PCROP 的描述，PCROP 的使用大致有以下几个步骤：

1. 将需要保护的文件，在 IDE 中标识为仅可执行，IAR 和 KEIL，CubeIDE 都有此类标识。
2. 需要修改 Link 文件，将受保护文件中的 rw data， ro data， ro code 进行分区存放。
3. 通过修改选项字，将受保护文件中的 ro code 中的内容进行 PCROP 保护。
4. 编译工程产生保护文件的.o 文件，并把符号导出给实际应用工程使用。

2. 问题描述

在我使用 IAR 进行 PCROP 的测试时，发生了如下错误。

图 1. IAR 中的 Hard Fault 提示窗口



3. 问题分析与定位

PCROP 调试时，会遇到一个问题，那就是由于代码已经被保护起来，无法单步调试。目前知道的只是当 PCROP 打开的时候会导致 Hard Fault，而不使用 PCROP 没有问题。那么这个问题和 PCROP 强相关了。

接下来，只能通过一些简单的测试来定位发生问题的大概位置了，比如在函数入口添加一些简单的汇编代码，比如 IAR 中可以添加代码往寄存器 R5 中写入特殊测试值：__ASM(" MOVS R5,

#11")，通过这个简单的测试，发现函数并没有执行到这里来。所以大概在跳转之后，执行这一句代码之前就已经发生了问题。

Hard Fault 的错误描述为：INVSTATE，无效状态代表的是试图切换到 ARM 状态，查看文档后有以下信息：

图 2. UFSR 寄存器描述

- 发生用法错误时，可通过查看寄存器了解出错的原因
  - SCB->CFSR. Usage Fault（UFSR） @0xE000ED2A

位段	名称	类型	含义	备注
9	DIVBYZERO	可读、写→清零	企图执行除0操作（指令：SDIV、UDIV）	使能控制：SCB->CCR.DIV_0_TRP
8	UNALIGNED		企图执行非对齐访问	使能控制：SCB->CCR.UNALIGN_TRP
3	NOCP		企图执行协处理器指令	
2	INVPC		无效的异常返回码	
1	INVSTATE		试图切换到ARM状态	
0	UNDEFINSTR		企图执行未定义指令	

把 PCROP 取消掉以后，查看汇编代码如下图 3：

图 3. 汇编窗口

```
  ASC_Task_Init();
  0x9003'70f8: 0xf7ff 0xff6e  BL        ?Veneer 23 (6) for ASC_Task_Init ; 0x9003'6fd8
  AITest_Init((RXRAWDATA)ASC_Run);
  0x9003'70fc: 0xf8df 0x0af4  LDR.W     R0, [PC, #0xaf4]          ; ASC_Run
  0x9003'7100: 0xf000 0xfec6  BL        AITest_Init               ; 0x9003'7e90
  osKernelStart();
  0x9003'7104: 0xf000 0xfde6  BL        osKernelStart             ; 0x9003'7cd4
    SENSING1_PRINTF("%d ",sum);
  0x9003'7108: 0x7828         LDRB      R0, [R5]
  0x9003'710a: 0x2802         CMP       R0, #2
```

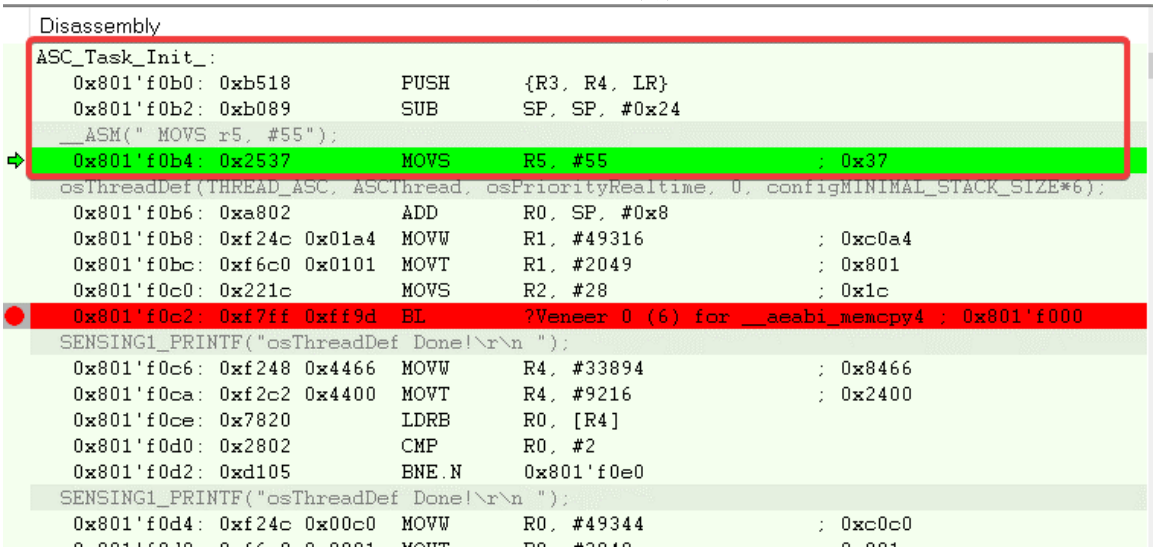
0x9003 开头的是外部 flash 地址，首先会跳转到 0x9003’6fd8 去执行，如下图 4。

图 4. 汇编窗口

```
?Veneer 22 (6) for test_sum:
  0x9003'6fd0: 0xf8df 0xf000  LDR.W     PC, [PC, #0x0]            ; test_sum
  0x9003'6fd4: 0x0801'f43d    DC32      test_sum
?Veneer 23 (6) for ASC_Task_Init:
  0x9003'6fd8: 0xf8df 0xf000  LDR.W     PC, [PC, #0x0]            ; ASC_Task_Init
  0x9003'6fdc: 0x0801'f0b1    DC32      ASC_Task_Init
?Veneer 24 (6) for ASC_Run:
  0x9003'6fe0: 0xf8df 0xf000  LDR.W     PC, [PC, #0x0]            ; ASC_Run
  0x9003'6fe4: 0x0801'f1e5    DC32      ASC_Run
```

红色框出来的部分就是 0x9003’6fd8 的内容了，这里会取下一个 PC 指针指向区域的内容，赋给 PC 指针，也就是 PC 会跳转到 0x0801’f0b0 去执行，LSB=1 代表的是下一条为 thumb 指令，如下图 5。

图 5. 汇编窗口



然后就会进行压栈操作，到此下面自己写的汇编测试指令还没有走到就已经出现 Hard Fault 了，目前看来，要么是跳转的问题，要么是压栈操作导致了 Hard Fault。

为了方便继续测试，又写了一个测试函数，如下图 6：

图 6. 测试代码

```
int test_sum(int num)
{
	__ASM(" MOVS R5, #22");
	int sum=0;
	for(int i=0; i<num; i++){
		sum += i;
	}

	return sum;
}
```

在该函数加入到 PCROP 区域后，发现没有问题，和之前的汇编代码对比，该函数由于局部变量很少所以并没有压栈的操作，那么我可以尝试把局部变量加大，让编译器产生 PUSH 等压栈操作。修改后的代码如下图 7：

图 7. 测试代码

```
int test_sum(int num)
{
	__ASM(" MOVS R5, #22");
	int sum=0;
	int buf[256]={0};
	for(int i=0; i<num; i++){
		sum += i;
		buf[i] = sum;
	}
	return sum;
```

```
}
```

修改完成后，再次测试，确认已经产生了 PUSH 操作，果然 Hard Fault 又出现了。
但是 PUSH 操作不应该导致 hard fault 啊，继续查看汇编代码，查找可疑的指令：

图 8. 汇编窗口

```
int test_sum(int num)
{
test_sum:
   0x801'f00c: 0xb538          PUSH      {R3-R5, LR}
   0x801'f00e: 0xf5ad 0x6d80   SUB.W     SP, SP, #1024            ; 0x400
   0x801'f012: 0x0004          MOVS      R4, R0
    int sum=0;
   0x801'f014: 0x2500          MOVS      R5, #0
   0x801'f016: 0xf44f 0x6280   MOV.W     R2, #1024                ; 0x400
   0x801'f01a: 0x2100          MOVS      R1, #0
   0x801'f01c: 0x4668          MOV       R0, SP
   0x801'f01e: 0xf7ff 0xffef   BL        ?Veneer 0 (9) for memset ; 0x801'f000
    for(int i=0; i<num; i++){
   0x801'f022: 0x2000          MOVS      R0, #0
    for(int i=0; i<num; i++){
   0x801'f024: 0x42a0          CMP       R0, R4
   0x801'f026: 0xda05          BGE.N     0x801'f034
        sum += i;
   0x801'f028: 0x1945          ADDS      R5, R0, R5
        buf[i] = sum;
```

在 0x801’f01e 这个位置，有一段跳转指令，代码会跳转到 0x801’f000 的位置去执行，继续追踪：

图 9. 汇编窗口

```
?Veneer 0 (9) for memset:
   0x801'f000: 0xf8df 0xc004   LDR.W     R12, [PC, #0x4]          ; 0x8802'31bd
   0x801'f004: 0x44fc          ADD       R12, R12, PC
   0x801'f006: 0x4760          BX        R12
   0x801'f008: 0x8802'31bd     DC32      0x8802'31bd
```

问题来了：这个 LDR 指令需要去取 PC 指针+4 的位置的数据，而此时 PC 指针肯定是位于 PCROP 区域的，也就是说这里需要数据总线访问 PCROP 区域，这肯定是不被允许的，所以才导致了 Hard Fault。
那么，编译器为什么会产生这段跳转指令呢？查看 map 文件后，这段 flash 区域放置的是 veneer section，Veneer 又是什么呢？
查看 IAR 的帮助文档，可以发现以下信息：

图 10. IAR 帮助文档

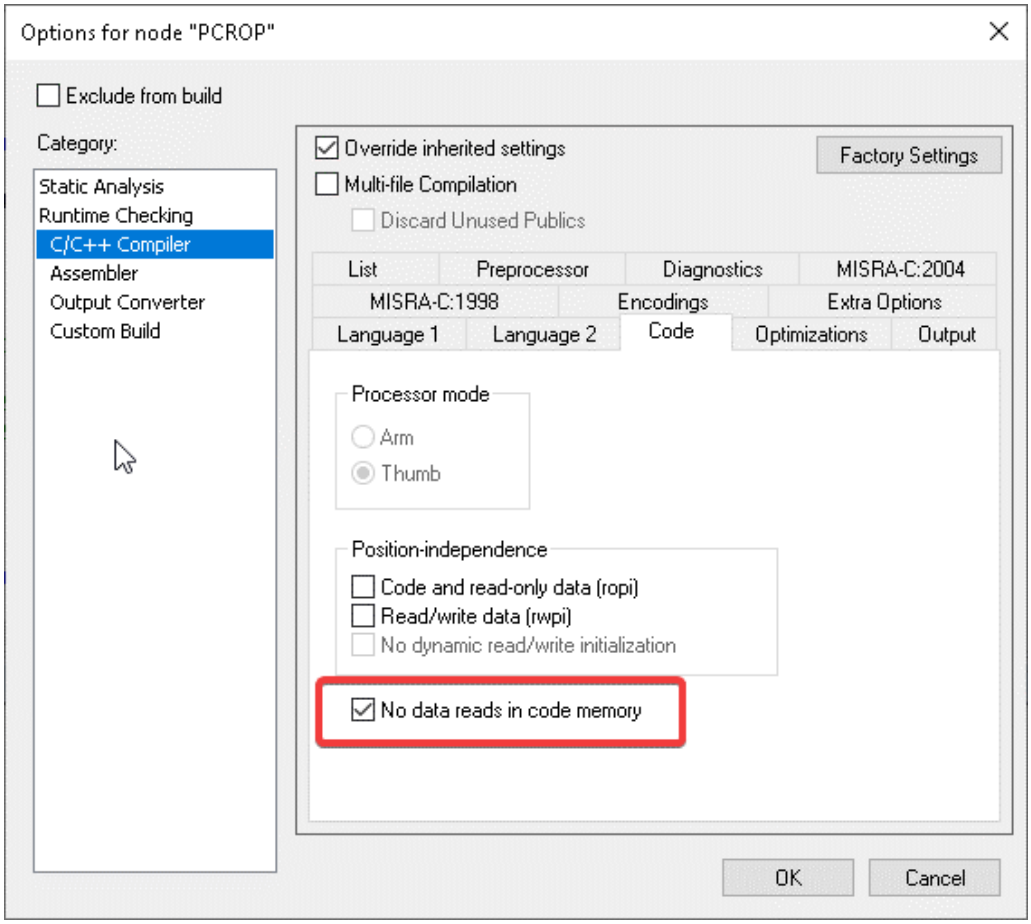
Explanation

The 3 different headings correspond to these addtions made by the IAR ILINK Linker:

Kind of section	Section type	Explanation
ro code	Veneer	These are "helpers" used in order to achive jumps in the whole address area, i.e. longer than +-4 MB / +-32 MB (Thumb/Arm mode).

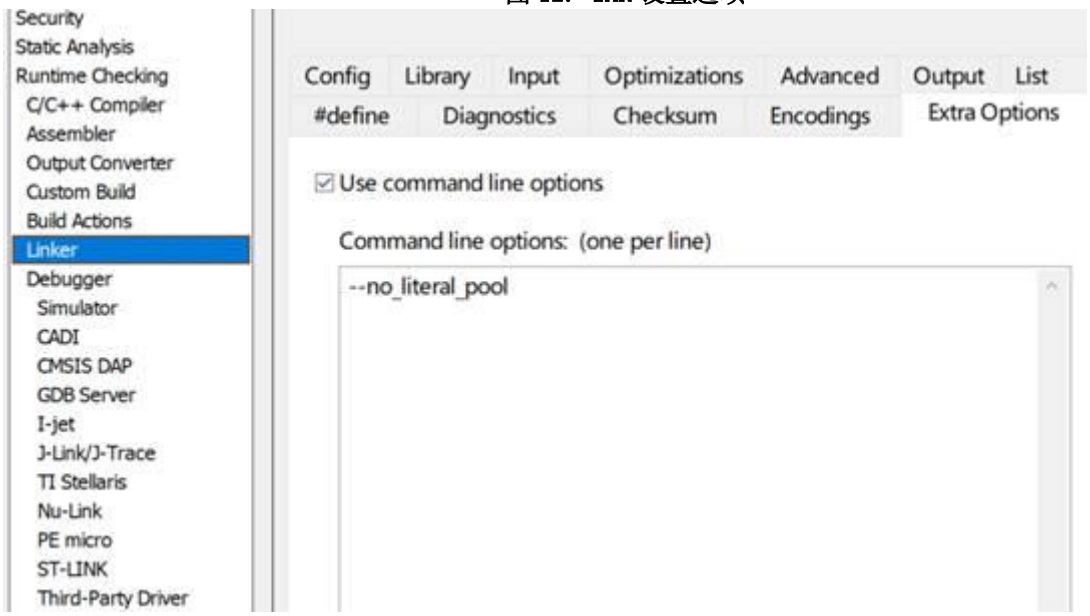
原来，编译器为了实现长跳转，会自动生成一段 Veneer 代码，并且将跳转的地址放在这段区域，而 IAR 默认将这段代码放在了 PCROP 的起始区域，所以才导致了该问题。
根据 AN4968 中关于 IAR 的配置，只有以下内容需要勾选，目前看来是无法解决这个问题的：

图 11. IAR 设置选项



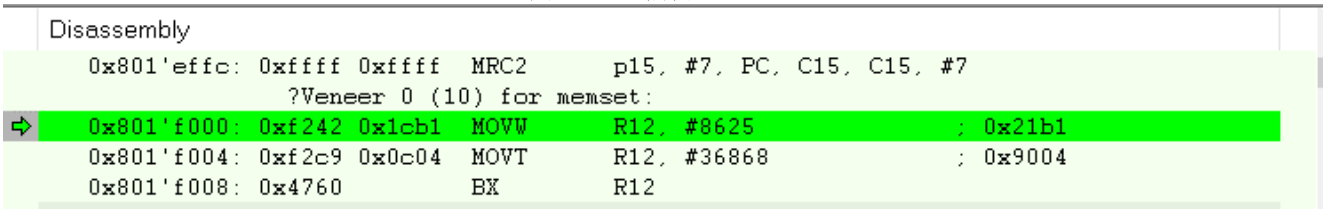
在咨询了 IAR 的支持工程师后，得到信息是可以在 Link 配置中设置：

图 12. IAR 设置选项



设置完成后，产生的跳转部分汇编代码如下：

图 13. 汇编窗口



可以看到，这里先将 0x21b1 存放到 R12 的低地址，再将 0x9004 存放到 R12 的高地址，最后 BX R12 完成跳转。和前面的跳转部分汇编相比，此时不需要访问 PCROP 保护的 flash 区域的数据了，所以不会有问题。

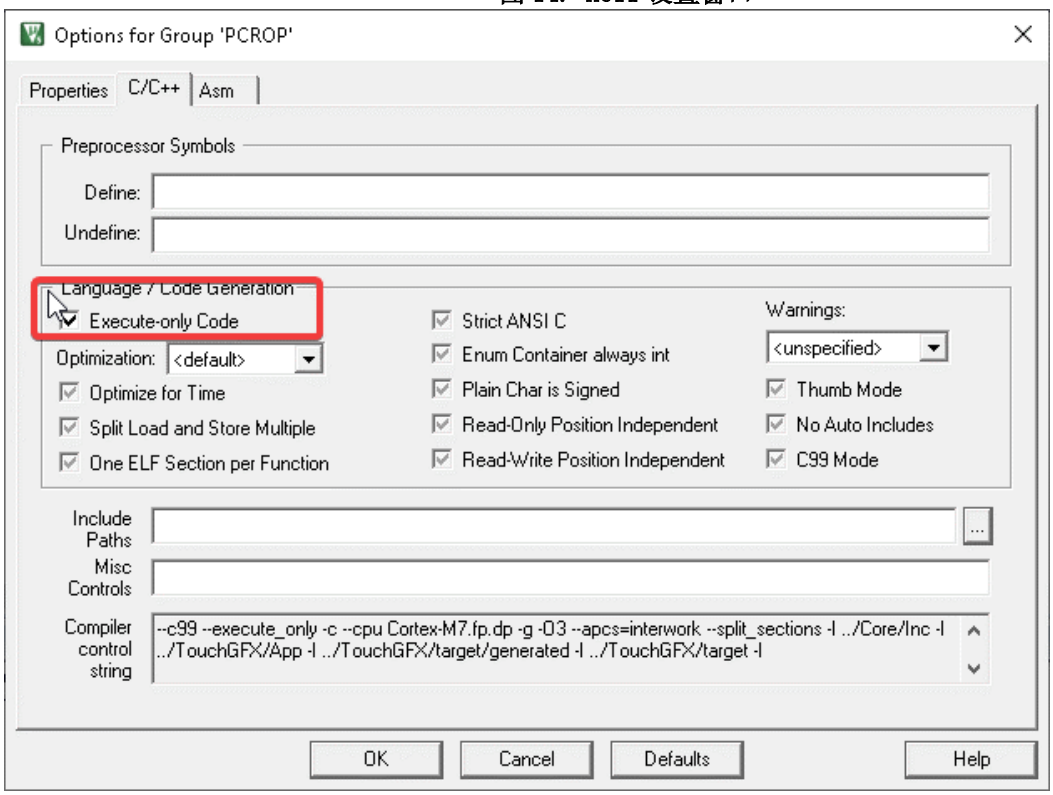
虽然这种方法可以解决，但图中 LINK 文件的配置是针对全局的，取消掉 literal pool 会对全局的代码效率产生影响，有没有更好的办法可以只针对 test.c 文件做配置呢？

非常遗憾，在当前最新的 IAR 版本中，无法完美解决该问题。

4. 问题解决

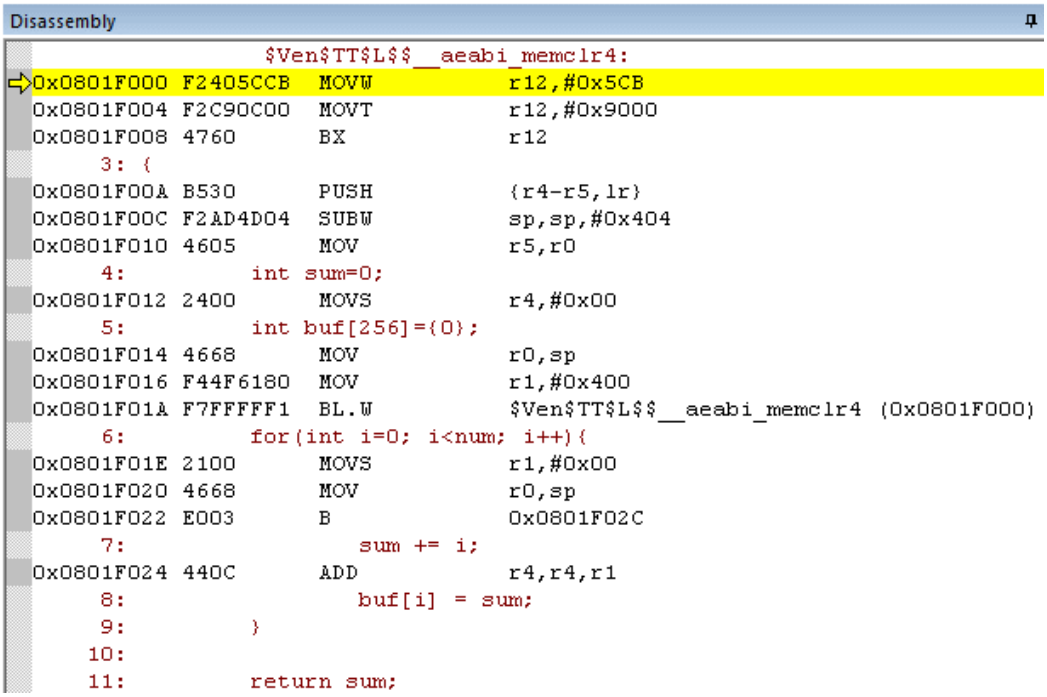
在发现 IAR 有这个问题后，尝试使用 Keil 和 CubeIDE 来测试该问题。
KEIL 中，只需要配置以下内容就可以达到目的.

图 14. Keil 设置窗口



Keil 生成的汇编代码，从图中可以看到，此处 Veneer 产生的跳转是不会有问题的.

图 15. Keil 汇编窗口



另外，在 Keil 的官方文档中，有以下提示，这表明 Keil 已经注意到了类似的问题，只需要勾选 XO 选项就能避免：

图 16. Keil 说明文档

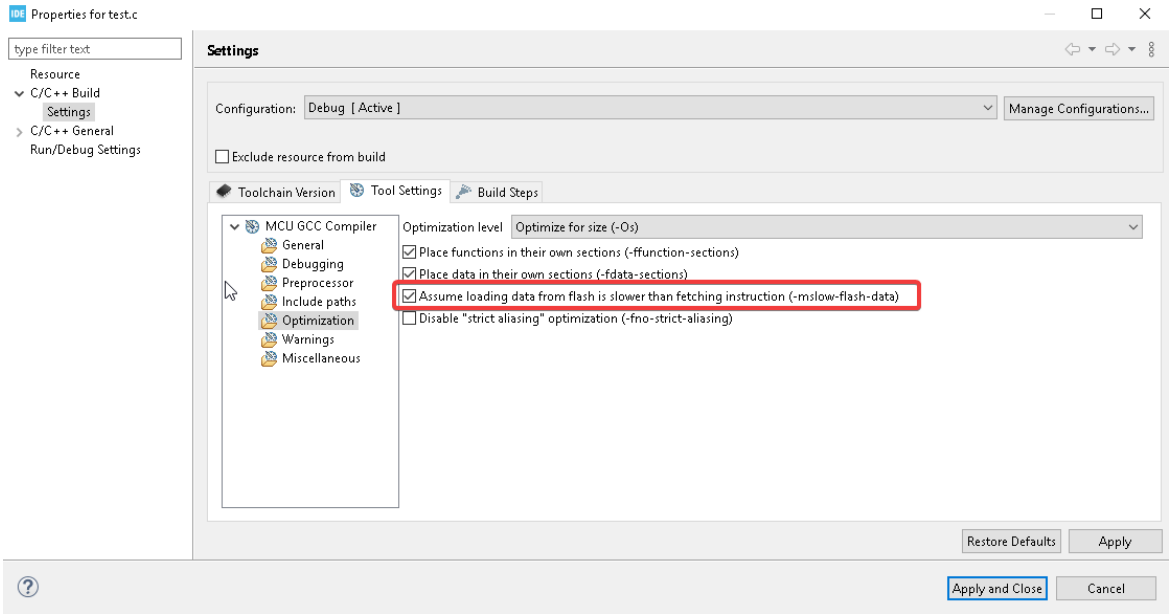
Note

If *execute-only* (XO) sections are present, only XO-compliant veneer code is created in XO regions.

CubeIDE 中，需要在下图两处地方进行配置：

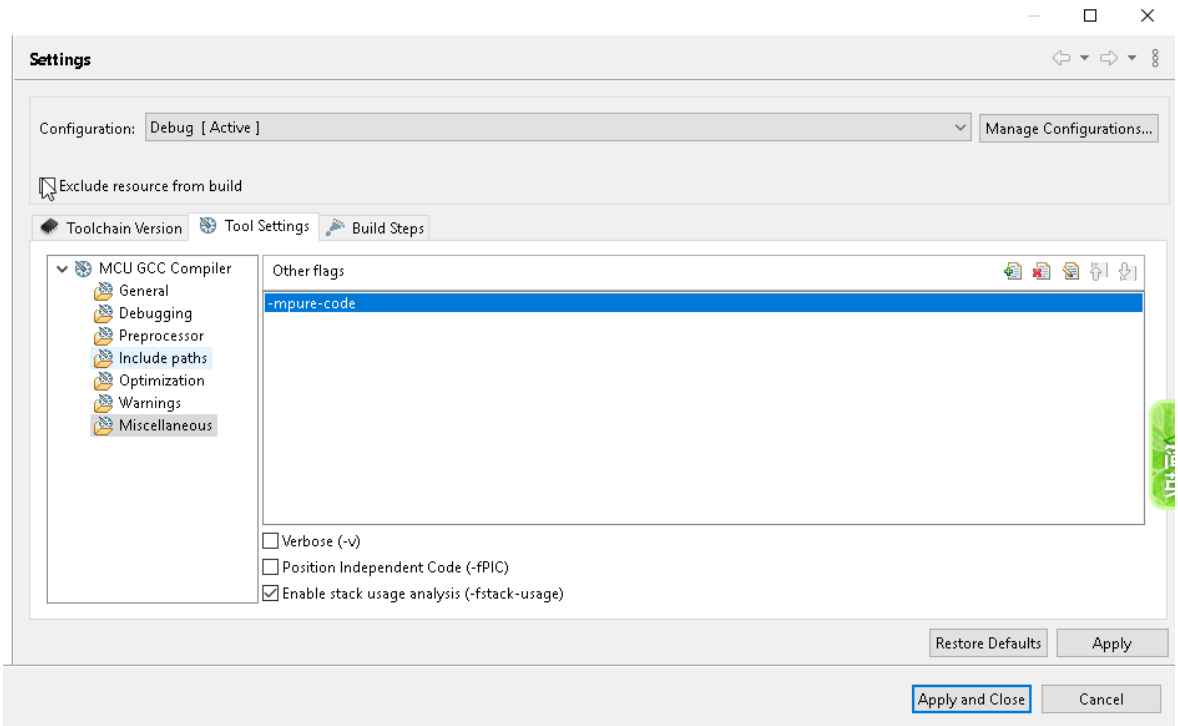
1.勾选-mslow-falsh-data 选项

图 17. Cube IDE 选项设置



2.在 GCC 编译器中，添加-mpure-code 选项

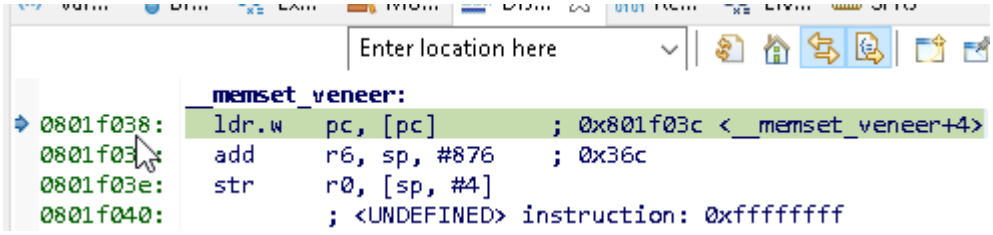
图 18. Cube IDE 选项设置



对比下 CubeIDE 配置前和配置后的汇编代码如下：

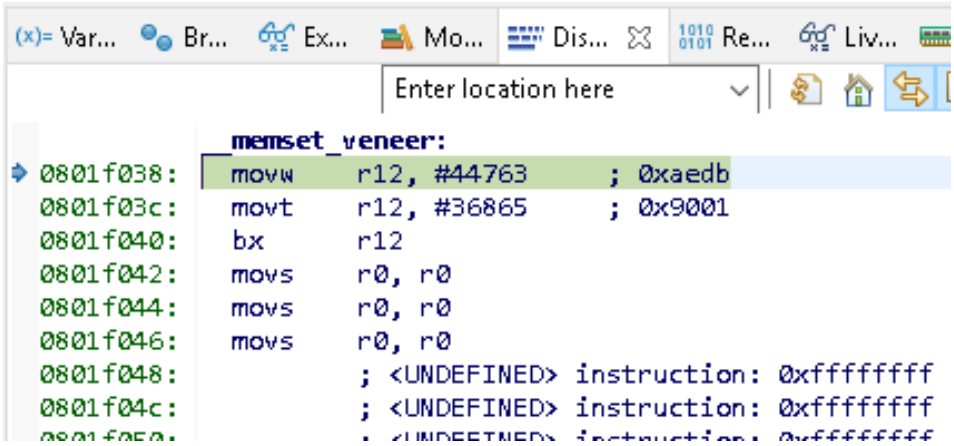
配置前：访问了 PCROP 区域的 flash 内容进行跳转。

图 19. Cube IDE 汇编窗口



配置后，直接修改 R12 进行跳转。

图 20. Cube IDE 汇编窗口



5. 总结

PCROP 只能保护片内 flash 区域，无法保护片外 flash，在使用 PCROP 进行保护时，不仅需要配置好 Link 文件，还需要配置好 IDE，注意片外 Flash 和片内 Flash 区域相互跳转的地方。

参考文献

Keil Veneer 说明：

- https://www.keil.com/support/man/docs/armlink/armlink_pge1406301797482.htm

IAR Veneer 说明：

- https://www.iar.com/support/tech-notes/linker/what-is-linker-created-and-lcgbwk-in-the-.map-file/
- AN4968: Proprietary code read out protection (PCROP) on STM32F72xxx and STM32F73xxx microcontrollers
-

文档中所用到的工具及版本

测试工具版本信息：

- IAR：8.50.1
- KEIL：5.24.2.0
- CubeIDE：1.5.1

版本历史

日期	版本	变更
2021 年 10 月 29 日	1.0	首版发布