AN4968
应用笔记

STM32F72xxx和STM32F73xxx微控制器上的专有代码读取保护

前言

软件提供商开发了复杂的中间件解决方案（知识产权代码，或IP-code）， 保护它们对于微控制器而言至关重要。

为了应对这一重要需求，STM32F72xxx和STM32F73xxx微控制器具有以下功能：

- 读保护（RDP）：防止进行读取操作
- 写保护：防止进行不需要的写入或擦除操作
- 专有代码读取保护：防止进行读写操作

本应用笔记对这些闪存保护技术进行了说明，重点是STM32F72xxx和STM32F73xxx微控制器上的专有代码读取保护，并提供了PCROP保护的基本示例。

本文档随附的X-CUBE-PCROP嵌入式软件包包含了PCROP示例的源代码，以及运行示例所需的所有嵌入式软件模块。

本应用笔记必须同*STM32F72xxx和STM32F73xxx 基于ARM® 的高级32位MCU*的参考手册（RM0431）一起阅读，数据手册可以在*www.st.com*上获取。

目录

表格索引

图片目录

1 存储器保护说明

1.1 读保护（RDP）

读取保护是全局闪存读保护，可保护嵌入式软件代码，避免复制、逆向工程、使用调试工具读出或其他方式的入侵攻击。该保护必须在二进制代码载入嵌入式闪存后，由用户进行设置。

以下章节中对三个RDP级别（0，1和2）进行定义和描述。

1.1.1 读保护级别0

级别0是默认级别，闪存完全打开，可在所有引导配置（调试功能，从RAM、从系统内存引导加载程序或从闪存启动）下进行全部内存操作。这种模式下，没有保护，该模式用于开发和调试。

1.1.2 读保护级别1

激活读保护级别1时，即使是从SRAM或系统内存引导加载程序来启动，也不能使用调试功能（如串行线路或JTAG）访问（读取，擦除和编程）闪存或备用SRAM。

但是，当从闪存启动时，允许用户代码访问Flash存储器和备用SRAM。

将RDP选项字节重新编程为级别0，可禁用RDP级别1保护，这会导致批量擦除。

1.1.3 读保护级别2

激活RDP级别2时，级别1所支持的所有保护均有效，芯片受到全面保护。RDP选项字节和所有其他选项字节都会被冻结，不能再修改。JTAG、SWV（单线查看器）、ETM 和边界扫描被禁用。

从闪存启动时，用户代码可以访问内存内容。但是，不再能从SRAM或从系统内存引导加载程序启动。

这种保护是不可逆的（JTAG熔断），所以不能回到保护级别1或0。

*表 1*描述了从内部闪存启动，或在调试，或从SRAM或系统内存引导加载程序启动时，对闪存、备用SRAM、选项字节和一次性可编程字节（OTP）的不同访问。

表1. 访问状态 vs 保护级别和执行模式

存储区	保护级别	调试功能，从RAM或系统存储器自举程序启动			从 Flash 自举		
		读取	写入	擦除	读取	写入	擦除
主Flash存储器和备份SRAM	级别 1	否		否(1)	是		
	级别 2	否			是		
选项字节	级别 1	是			是		
	级别 2	否			否		
OTP	级别 1	否		NA	是		NA
	级别 2	否		NA	是		NA

1. 只有在 RDP 从级别 1 更改为级别 0 时，才会擦除主 Flash 和备份 SRAM。OTP 区域保持不变。

1.1.4 STM32F72xxx和STM32F73xxx RDP保护下的内部闪存内容更新

当RDP保护激活时（级别1或级别2），内部闪存内容不能通过调试进行更新，当从SRAM或系统内存引导加载程序启动时也不能更新。

对最终产品的一个重要要求就是，能够将内部Flash存储器中的嵌入式软件升级为新的软件版本，添加新功能并修正潜在问题。

可通过实施用户专用嵌入式软件执行内部闪存应用内编程（In-Application Programming，IAP）来解决此需求。IAP使用通信协议（如USART）来实现重编程过程。

有关IAP的更多详细信息，请参阅*www.st.com*上提供的*STM32使用USART进行应用内编程*应用笔记（AN3965）。

1.2 写保护

写保护用来保护指定扇区内容，避免代码更新或擦除。这种保护可应用于扇区。

任何写请求都会产生一个写保护错误。如果要擦除/编程的地址属于闪存中处于写保护状态的区域，则通过硬件将WRPERR标志置位。

例如，如果闪存中至少有一个扇区是写保护的，则不能对其进行批量擦除，并且WRPERR标志置位。

要激活每个闪存扇区 i 的写保护，可使用一个选项位nWRPi。当设置扇区 i（选项位nWRPi = 0）为写保护时，该扇区不能被擦除或编程。

可通过嵌入式用户代码或使用STM32 ST-Link Utility软件和调试接口，进行使能或禁用写保护管理。

1.3 专有代码读保护（Proprietary Code Read Out Protection，PCROP）

1.3.1 PCROP保护概述

PCROP是闪存中IP-code的读写保护。PCROP应用于扇区（0至7），保护专利代码不被最终用户代码、调试器工具或RAM Trojan代码修改或读取。

通过ITCM或AXI总线对PCROP扇区的任何读访问（获取操作除外）都会触发：

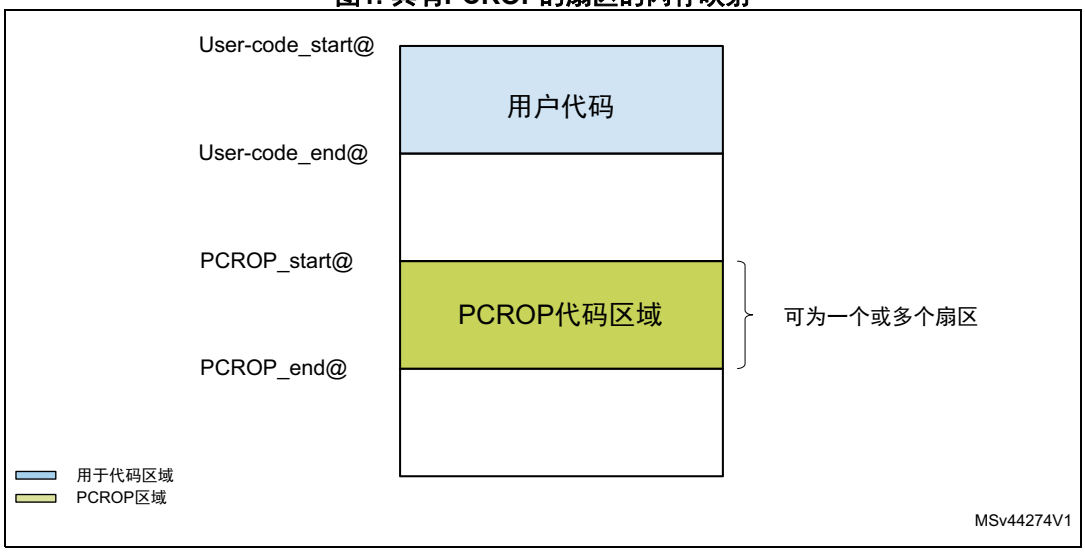
- 给定总线上出现总线错误
- FLASH_SR状态寄存器中RDERR标志置位。如果FLASH_CR寄存器中读错误中断使能位（Read Error Interrupt Enable，RDERRIE）置位，也会生成一个中断。

任何对PCROP扇区的编程/擦除操作都会触发WRPERR标志错误。

受保护的IP-code可以很容易地被最终用户应用程序所调用，并且仍能受到保护，不可直接访问IP-code本身。PCROP不会阻止执行受保护的代码。

注意： 虽然对PCROP扇区的读访问会产生总线错误，但读错误中断可被硬故障错误中断或总线故障错误中断所屏蔽。要使用读错误中断，必须将总线故障错误中断优先级使能为低于读错误中断优先级。

图1. 具有PCROP的扇区的闪存映射



1.3.2 如何使能PCROP保护

PCROP保护是逐扇区激活的，因此每个扇区可以单独作为PCROP扇区，并可以对附加扇区进行保护（当RDP设置为级别0或级别1时）。

PCROP保护通过FLASH_OPTCR2寄存器中的选项位PCROP[i]来激活：

- PCROP[i] = 0：扇区i（i = 0,...,7）上的PCROP保护不被激活
- PCROP[i] = 1：扇区i（i = 0,...,7）上的PCROP保护被激活

为了提高PCROP扇区的安全级别，在执行PCROP扇区中的代码时，所有调试事件都会被屏蔽。

另有一个选项位（PCROP_RDP = PCROP1ER [15]）可用来选择在RDP保护从级别1变为级别0时PCROP区域是否被擦除。

用来激活PCROP保护（PCROPi）的位和用来激活写保护nWRPi的其他位是独立的，因此可以同时存在一个写保护扇区和一个PCROP扇区。*表 2*显示了根据WRPi和PCROPi位的值，扇区i上会设置何种保护。

表2. 扇区i上的保护

nWRPi	PCROPi	扇区i上的保护
1	0	无保护
0	0	写保护
X	1	PCROP 保护

注： *关于实现PCROP使能的更多详细内容，用户必须参考所提供的FW包（STEP1-ST_Customer_level_n project main.c文件）中所述的PCROP_Enable()函数。*

1.3.3 如何禁用PCROP保护

根据RDP级别，如果RDP级别为1或0，则PCROP保护可被禁用，但是如果RDP设置为级别2，PCROP不能被禁用。当RDP设为级别2，所有选项字节都会被冻结，不能修改。因此，PCROP保护的扇区不能再被擦除或修改，这样就成为永久性保护。

禁用受保护扇区上PCROP的唯一方式是：

- 清除对应扇区的PCROPi位（多个扇区可同时可进行）
- 实现从级别1退回到级别0
- 将PCROP_RDP位置位

如果PCROP_RDP位被置位，则执行主闪存的批量擦除，并根据PCROPi值，对应扇区的PCROP保护保持使能或禁用。

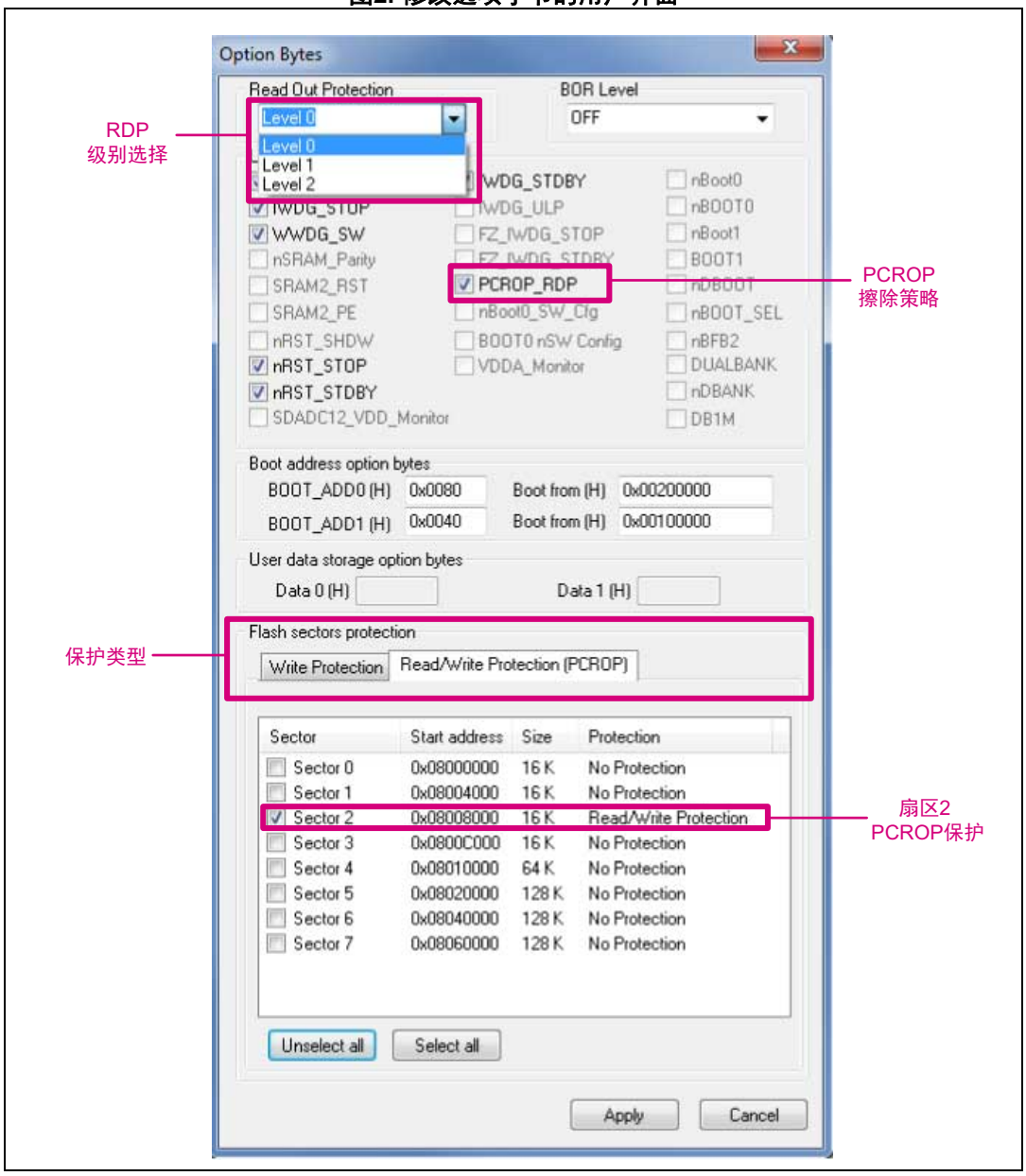
如果PCROP_RDP位被清零，则完全批量擦除由部分批量擦除所代替，该部分页面擦除是对PCROP激活的存储区内进行连续页面擦除（PCROP保护的页面除外）。这样做是为了保持PCROP代码。

注： *允许从级别1退回级别0与修改PCROP_RDP位同时进行由于完全或部分批量擦除在选项修改前启动，所以使用的是当前PCROP_RDP位，而不是被编程的新PCROP_RDP位。*

使用STM32-Link Utility

应用程序开发过程中，用户可能需要禁用PCROP或全局RDP保护，而不必花时间开发或禁用保护功能。STM32 ST-LINK Utility工具是一个非常简单的禁用或使能保护的方式，利用调试接口如JTAG或SWD即可实现，而无需开发专门的功能。*图 2*显示如何修改选项字节。

图2. 修改选项字节的用户界面



关于如何使用STM32 ST-LINK Utility软件的更多详细信息，用户须参考*STM32 ST-LINK实用软件说明*用户手册 （UM0892），可从*www.st.com*上获取。

1.3.4 存放和执行PCROP的IP-code

如前所述，PCROP不会阻止执行受保护的IP-code，并且其函数可方便地被用户代码所调用。

PCROP保护的扇区不被D-code总线读访问，这里重点指出，只允许进行代码执行（通过 I-code总线取指令），而禁止读取数据。因此，受保护的IP-code无法访问存储在同一区域的相关数据值（如文字池、分支表或常量，执行过程中它们通过D-code总线从闪存中取出）。

PCROP的代码必须是只可执行的代码，不能包含任何数据。

用户必须配置编译器来生成只可执行的IP-code，并且避免任何数据读取，所需编译器配置在*第 2 节*中有详细说明。

不可激活向量表扇区的PCROP保护

中断向量表包含了每个中断处理程序的进入点地址，它们可由CPU通过D-code总线进行读取。通常中断向量表位于首地址0x08000000的第一个扇区（例外是在某些情况下，它会被重新定位到其他区域，如SRAM）。

将代码置于闪存中进行保护时，必须遵守以下规则：

- 向量表所在的第一个闪存扇区不能是PCROP保护扇区
- PCROP保护的代码不能位于第一个扇区中。

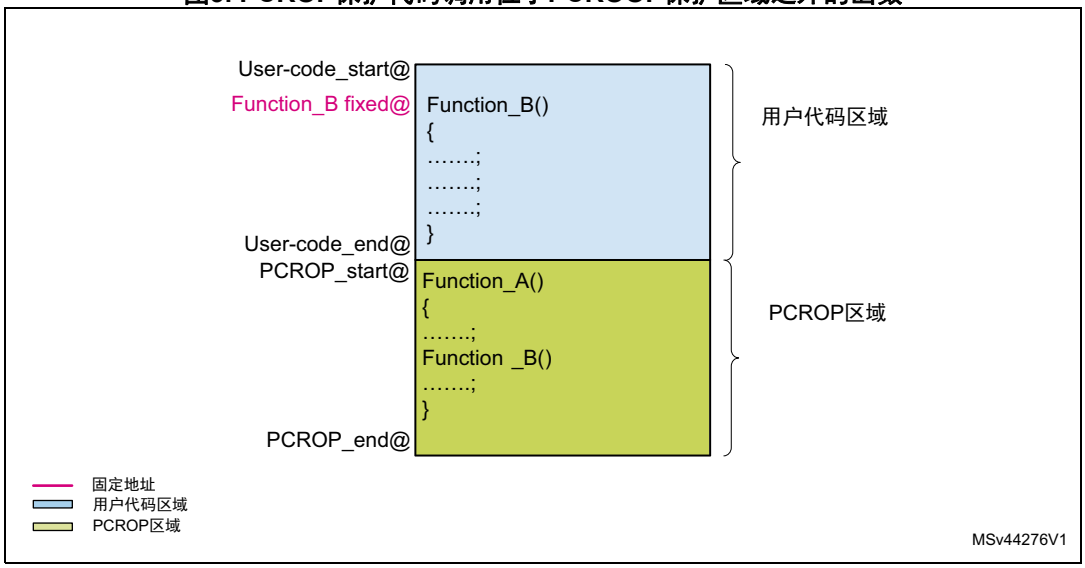
PCROP的IP-code依赖关系

受保护的IP-code可以从位于用户代码区域中以及PCROP保护区之外的库中调用函数。这种情况下，IP-code中包含了相关函数地址，允许PC（程序计数器）在执行IP-code时跳转到这些函数。一旦IP-code被PCROP保护，则这些地址不能更改。因此，每个调用函数必须位于（PCROP保护区域之外）PCROP保护IP-code中写入的相应固定地址，否则PC跳转到无效地址，IP-code无法正常工作。

要完全独立，受保护IP-code必须与其所有关联函数放在一起。

*图 3*显示一个示例，其中PCROP保护的Function_A()调用位于PCROP保护区域之外的固定地址的Function_B()。

图3. PCROP保护代码调用位于PCROOP保护区域之外的函数



2 PCROP示例

本应用笔记提供的X-CUBE-PCROP嵌入式软件示例对PCROP保护的使用情形进行了说明。开发此嵌入式软件所需的所有步骤均在本章详细说明。

2.1 示例要求

2.1.1 硬件要求

运行此示例所需的硬件如下：

- STM32F722ZE-Nucleo板（RevB）
- 微型USB数据线，用来为NUCLEO板上电，并连接嵌入式探索STLINK进行调试和编程。

2.1.2 软件要求

需要下列软件工具：

- IAR Embedded Workbench® 或 Keil® µvision IDE 或 SW4STM32
- STM32 STLink Utility主要用于使能或禁用PCROP保护。

2.2 示例概述

2.2.1 场景描述

本例描述了ST用户级别n向ST用户级别n+1提供具有关键IP-code的预编程STM32F722ZE MCU的用例。

IP-Code必须通过激活PCROP来保护。

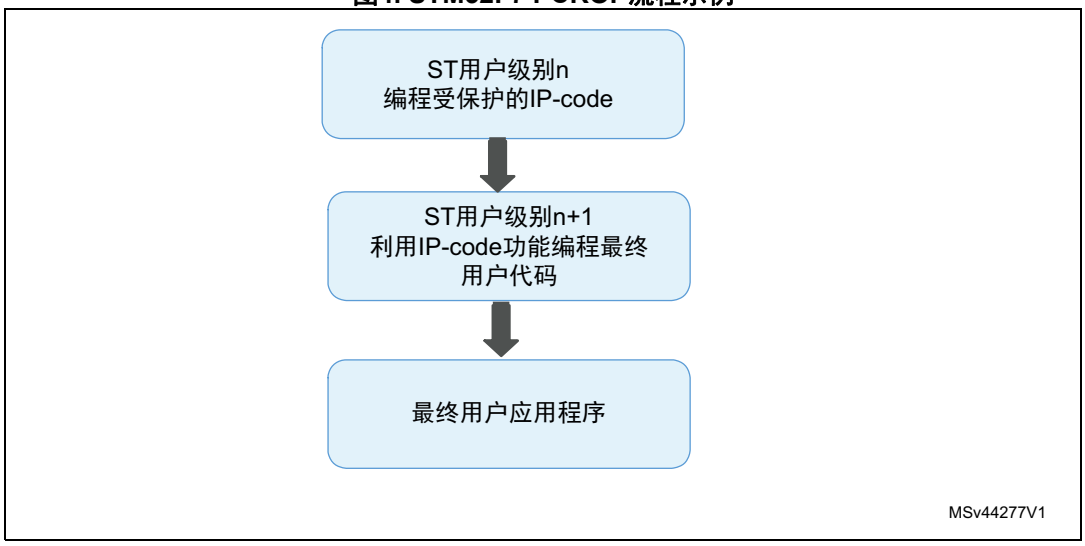
它允许ST用户级别n+1使用IP-code函数（不能读取或修改它们），并对最终用户应用程序进行编程。

ST用户级别n，ST用户级别n应将下列输入与预加载的STM32 MCU一起提供：

- 闪存和RAM内存映射，定义了准确的PCROP保护IP-code位置和可用的编程扇区
- ST用户级别n+1项目必须包含的头文件，其中含有最终用户代码中调用的IP-code函数定义
- 符号定义 文件，包含PCROP保护的IP-code函数符号。

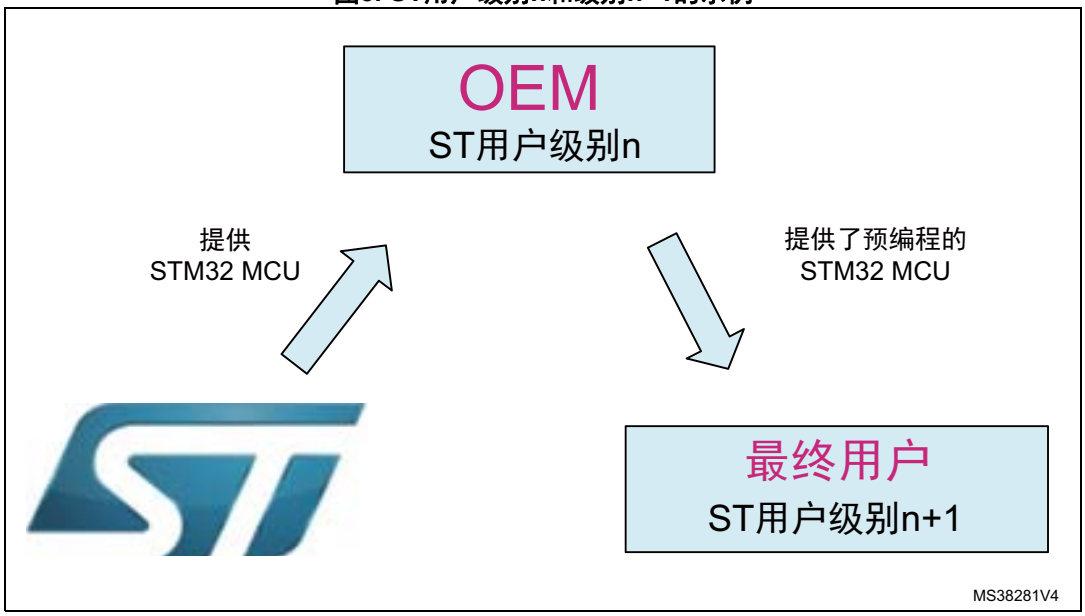
所述用例如*图 4中所示意。*

图4. STM32F7 PCROP流程示例



OEM（原始设备制造商）可以是使用STM32F7微控制器的ST用户级别n。然后OEM提供预编程MCU给ST用户级别n+1，这可能是制造最终用户产品的END CUSTOMER，如*图 5*中所示。

图5. ST用户级别n和级别n+1的示例



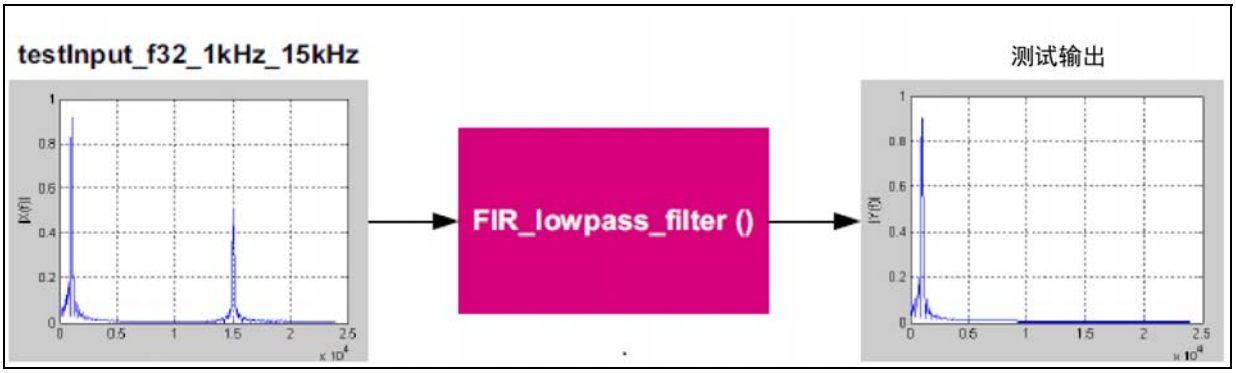
2.2.2 PCROP保护的IP-code：FIR低通滤波器

作为待保护的IP-code示例，将对来自CMSIS-DSP的FIR低通滤波器算法进行描述，重点介绍在不提供函数本身详细信息的情况下，如何保护和调用此IP-code函数。

FIR低通滤波器从输入中滤除高频信号分量。

输入信号是频率为1 KHz和15 KHz的两个正弦波之和。低通滤波器（其预配置截止频率设为6 KHz）滤除15 KHz信号，在输出端留下1 KHz正弦波。

图6. FIR低通滤波器函数框图



所用CMSIS DSP软件库函数为：

- arm_fir_init_f32()：配置滤波器的初始化函数，在arm_fir_init_f32.c文件中有描述；
- arm_fir_f32()：表示FIR滤波器的初等函数，在arm_fir_f32.c文件中有描述。

以下函数利用上述CMSIS DSP函数来创建：

- FIR_lowpass_filter()：表示FIR滤波器的全局函数，在fir_filter.c文件中有描述。

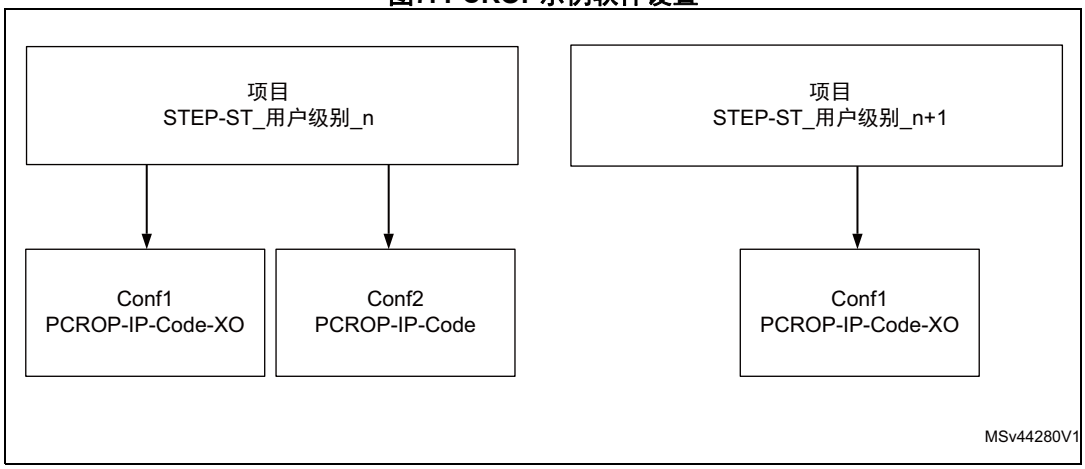
嵌入到STM32F72xxx和STM32F73xxx微控制器中的FPU和DSP用来进行信号处理和浮点计算，以输出正确信号。

关于FIR函数的更多详细信息，用户必须参考相关软件包里“Drivers/CMSIS/Documentation/DSP”目录中的CMSIS文档。

2.2.3 软件设置

本应用笔记提供了两个项目：

图7. PCROP示例软件设置



- **项目1：STEP1-ST_用户级别_ n**

  此项目显示了ST用户级别n如何存放、保护和执行其IP-code，以及如何生成IP-code关联文件如头文件和符号定义文件，并提供给ST用户级别n+1的示例。此项目包括两种不同的项目配置：

  – PCROP-IP-code-XO：此配置下，编译器配置为生成只执行IP-code，避免对其进行数据读取。

  – PCROP-IP-code：此配置下，编译IP-code，不阻止数据（文字池）生成。此配置专门用来进行测试，显示PCROP保护的IP-code是否为只执行代码。

- **项目2：STEP2-ST_用户级别_n+1**

  此项目显示了利用PCROP保护IP-code预编程STM32F722ZE MCU的ST用户级别n+1，如何利用这些受保护的IP-code函数来创建其自己的最终用户应用程序的示例。

2.3 STEP1：ST用户级别n

此阶段中，ST用户级别n：

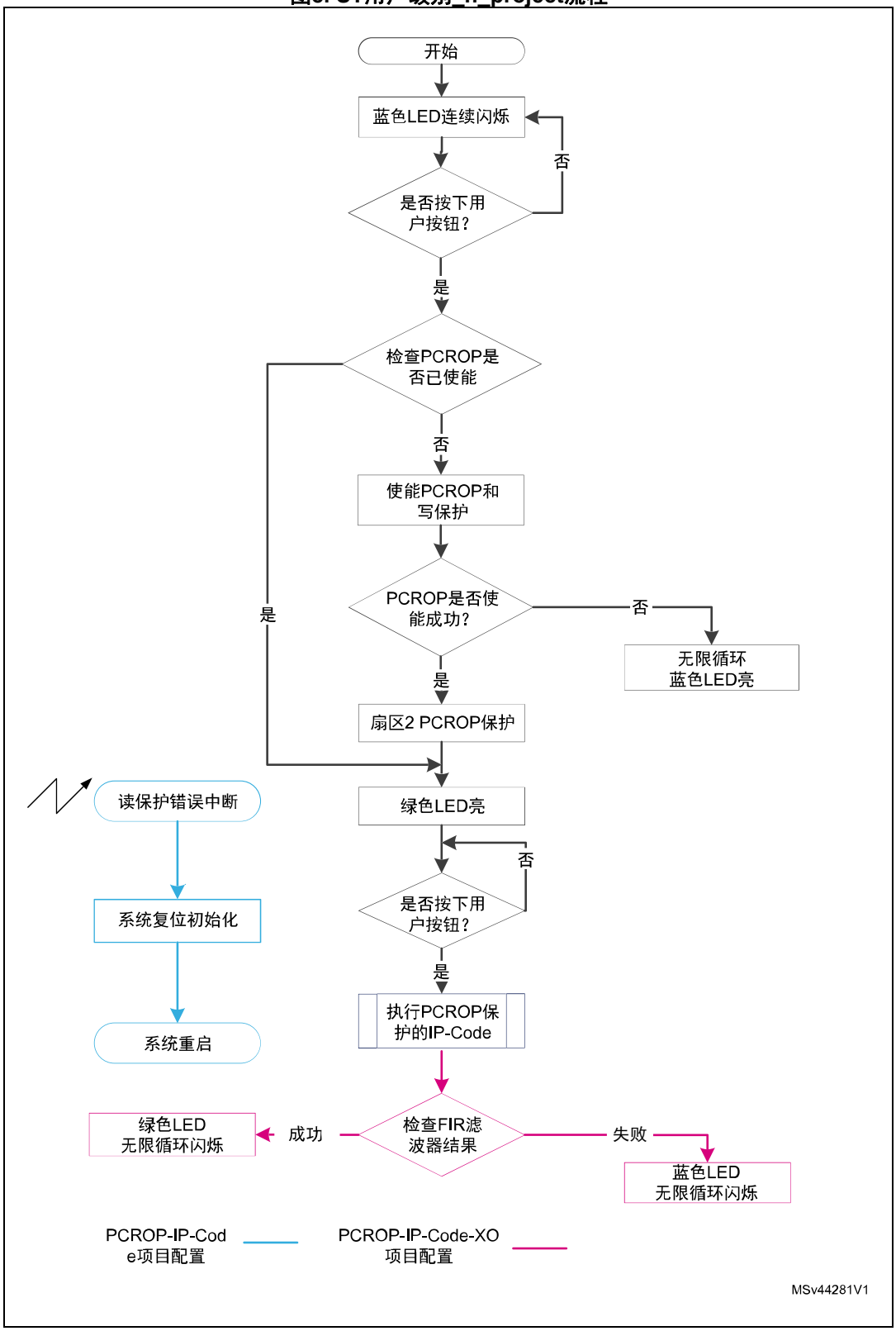
- 生成只执行IP-code
- 将IP-code和数据段放在闪存的指定位置
- 利用PCROP保护IP-code区域
- 写保护数据
- 通过在主代码中调用其函数，执行IP-code
- 创建头文件，并生成符号定义文件，用于STEP2-ST_用户级别_n+1项目。

2.3.1 项目流程

*图 8*说明了采用两种项目配置的STEP1-ST_用户级别_n项目流程：

- PCROP-IP-code-XO（粉色表示）：PCROP保护的IP-code不包含任何文字池，然后FIR滤波器算法成功运行，注意会产生一个读操作错误中断，并且如果从/向PCROP保护区域发送任何读/写请求，则启动系统复位。
- PCROP-IP-code（蓝色表示）：IP-code包含文字池，当开始执行PCROP保护的IP-code时，会产生一个读操作错误中断，然后启动系统复位，系统重新开始。

图8. ST用户级别_n_project流程

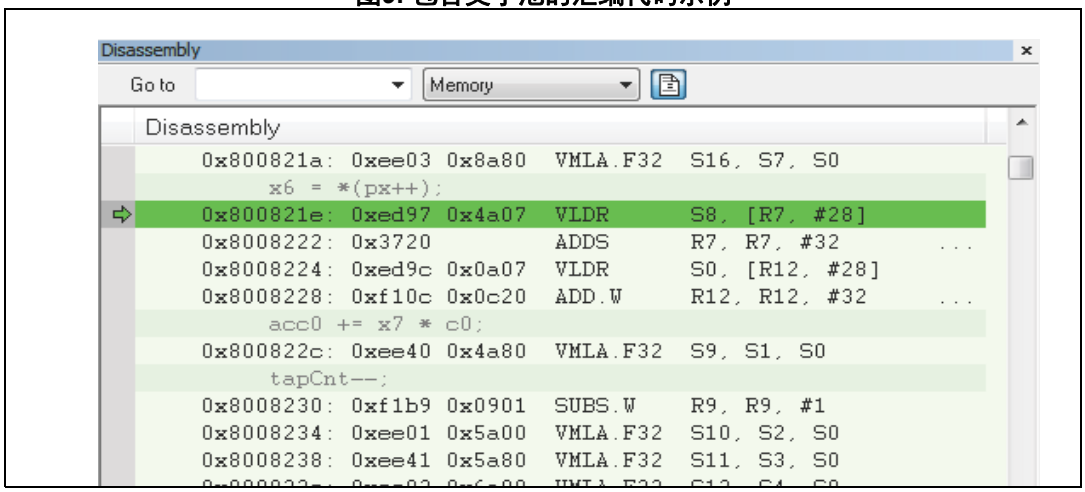


2.3.2 生成只执行IP-code

每个工具链都有其自己的选项，来防止编译器生成文字池和分支表。例如，Keil®具有“Execute-only code”选项，而IAR和SW4STM32具有“No data reads in code memory”选项。

*图 9*显示了包含文字池的汇编代码，其中指令格式为VLDR <variable>, [PC + <offset>]。

图9. 包含文字池的汇编代码示例



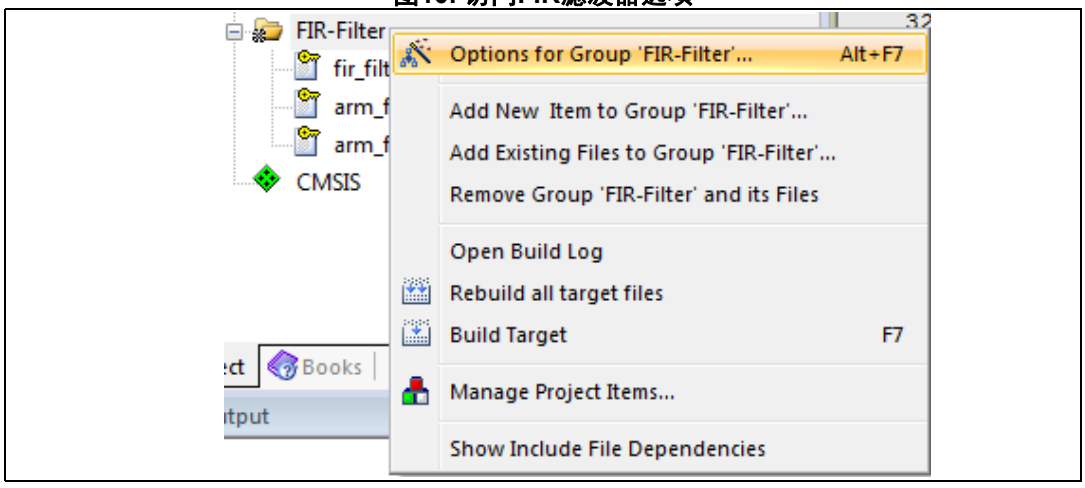
Keil®：使用Execute-only命令

必须使用编译选项“Execute-only”来生成无文字池的代码，防止编译器对代码扇区进行任何数据访问。

对于所有包含文字池的IP-code文件，必须使用该选项。

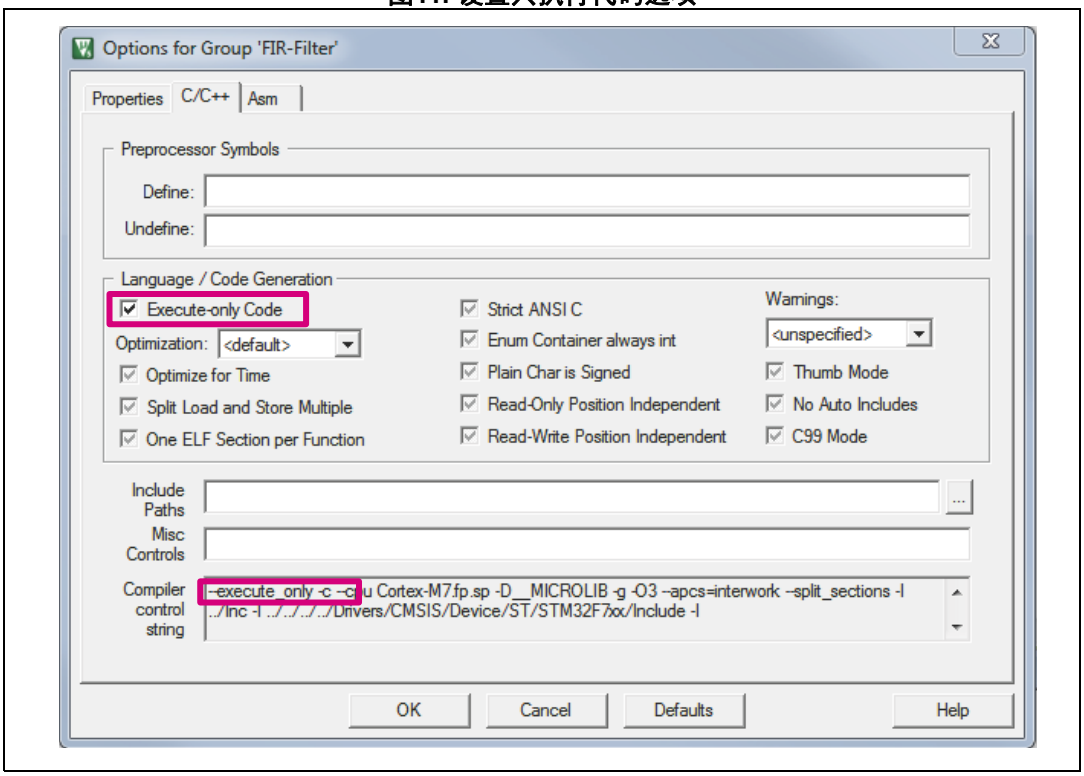
要设置此命令，请右键单击组文件或IP-code（参见*图 10*），并选择“Options for Group‘User/Fir’”。

图10. 访问FIR滤波器选项



在下面窗口中（见*图 11*），检查“Execute-only Code”选项，然后将execute_only命令添加到编译器控制字符串字段中。

图11. 设置只执行代码选项

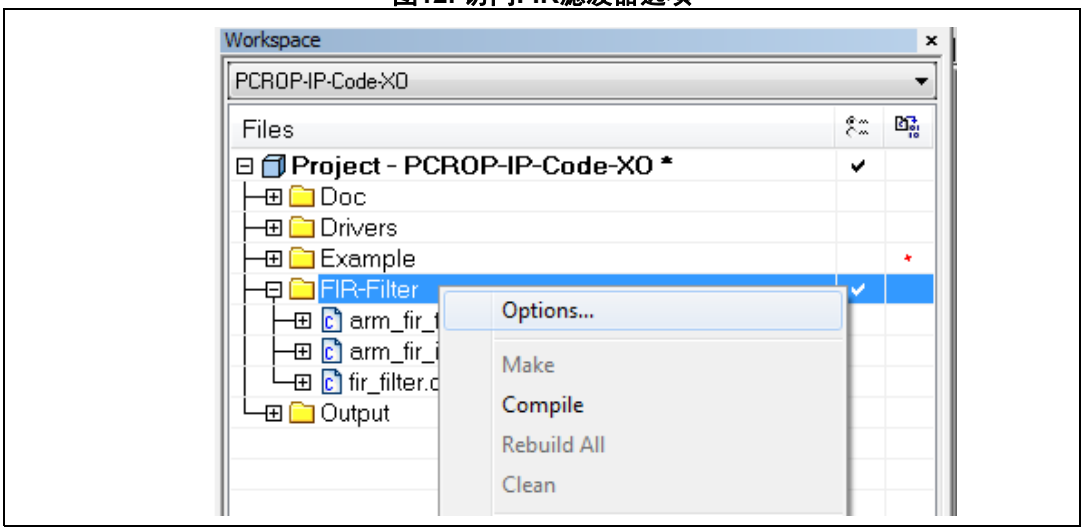


随后，定义IP-code内存区域布局的分散文件将访问类型属性设置为+XO。请参考*第 2.3.3 节：将IP-code和数据段放在Flash和RAM存储器中*。

IAR：代码内存中不进行数据读取

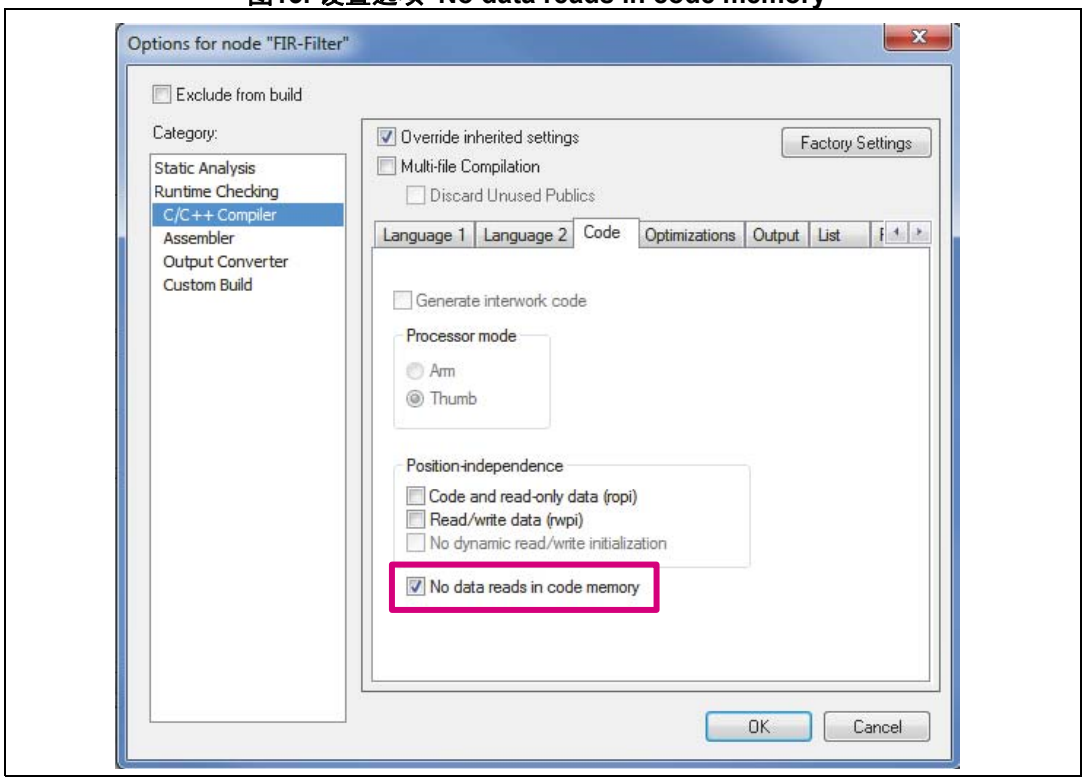
对于IAR，必须使用“No data reads in code memory”选项来防止编译器对代码扇区生成任何数据访问。要激活此选项，请右键单击包含IP-code源文件的文件组“FIR-Filter”，然后（见*图 12*）选择Options。

图12. 访问FIR滤波器选项



然后，在以下窗口中，选择选项“No data reads in code memory”，如*图 13*中所示。

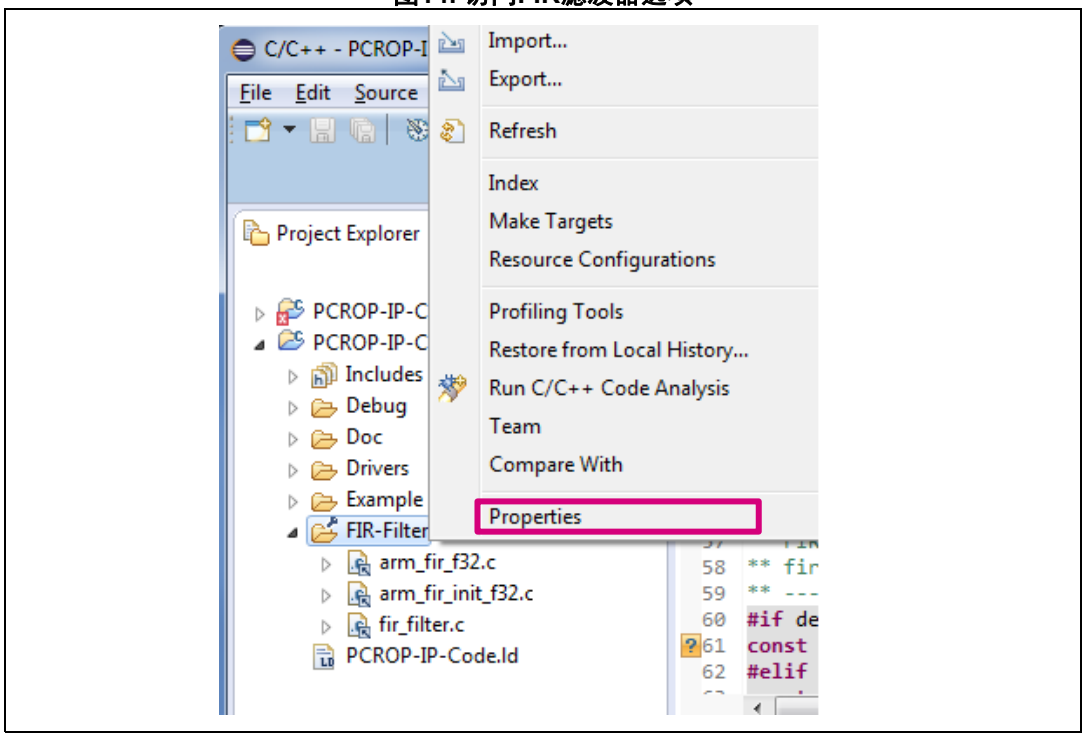
图13. 设置选项“No data reads in code memory”



SW4STM32：代码内存中不进行数据读取

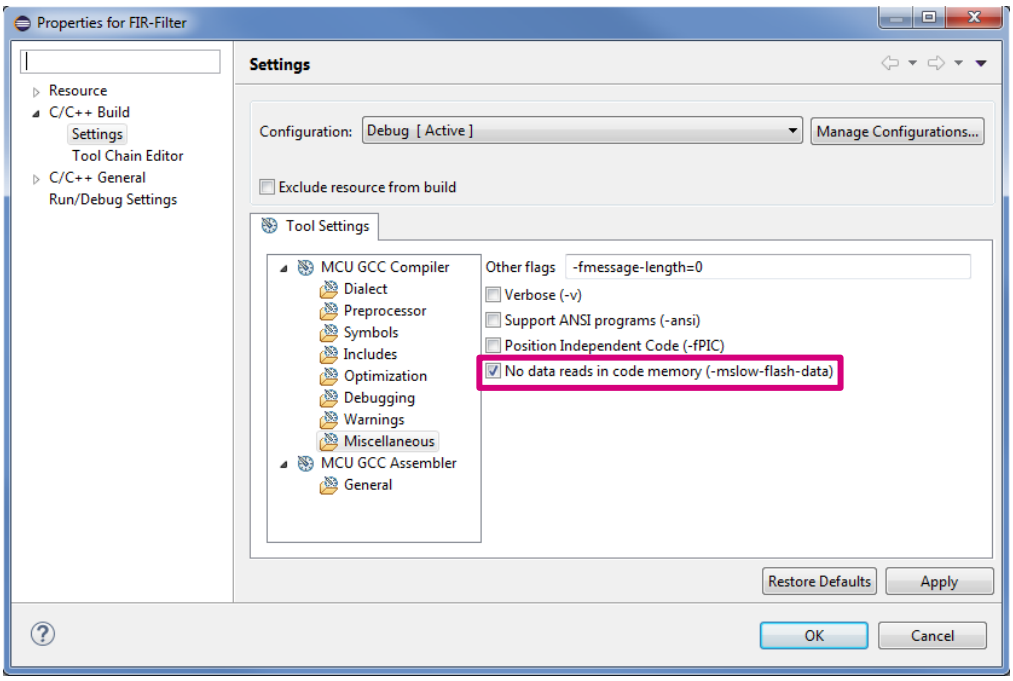
SW4STM32必须使用同一选项“No data reads in code memory”来防止编译器对代码扇区进行任何数据访问。要激活此选项，请右键单击包含IP-code源文件的文件组“FIR-Filter”，然后（见*图 14*）选择“属性”。

图14. 访问FIR滤波器选项



然后，在以下窗口中，选择选项“No data reads in code memory”，如*图 15*中所示。

图15. 设置选项“No data reads in code memory”



2.3.3 将IP-code和数据段放在Flash和RAM存储器中

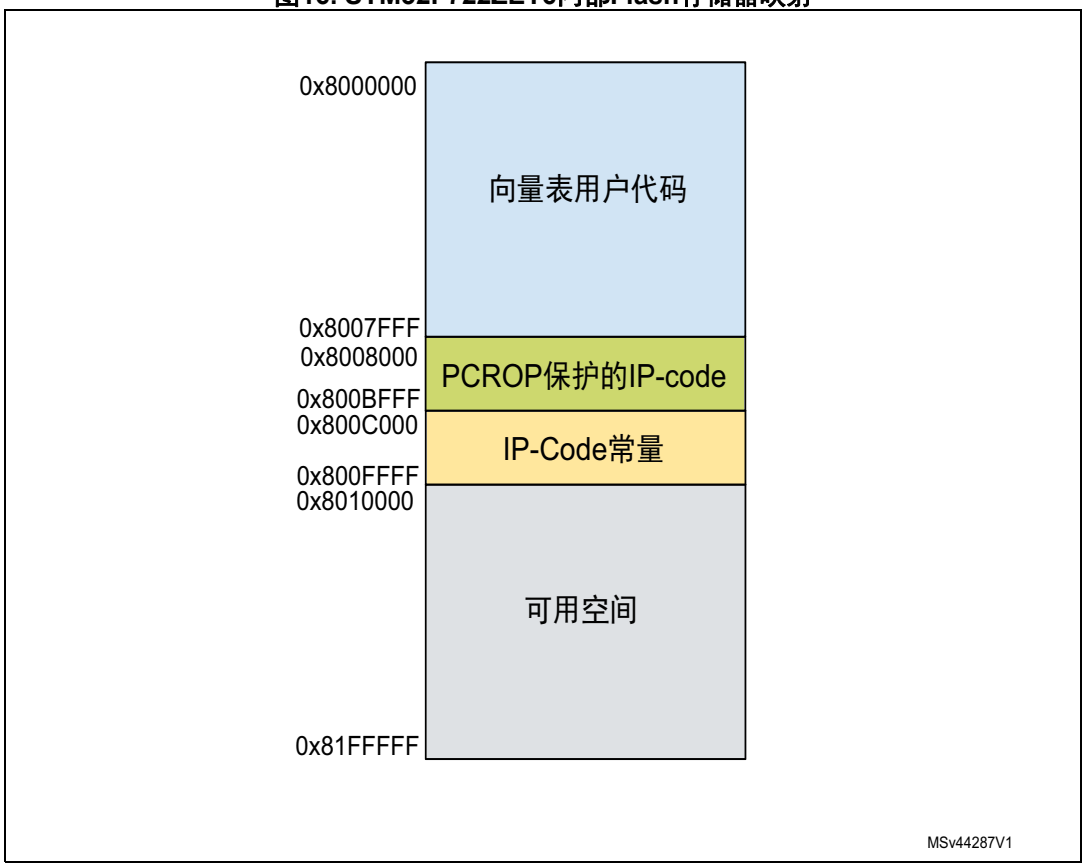
本例中，Flash存储器上定义了两个专用存储器扇区：

- IP-code段位于Flash存储区1中，地址为0x0800_8000
- 常量数据（FIR滤波器系数）位于代码扇区的后面，地址为0x0800_C000。

用户代码和向量表的主存储区域位于开端

所产生的Flash存储器映射如*图 16*所示。

图16. STM32F722ZET6内部Flash存储器映射

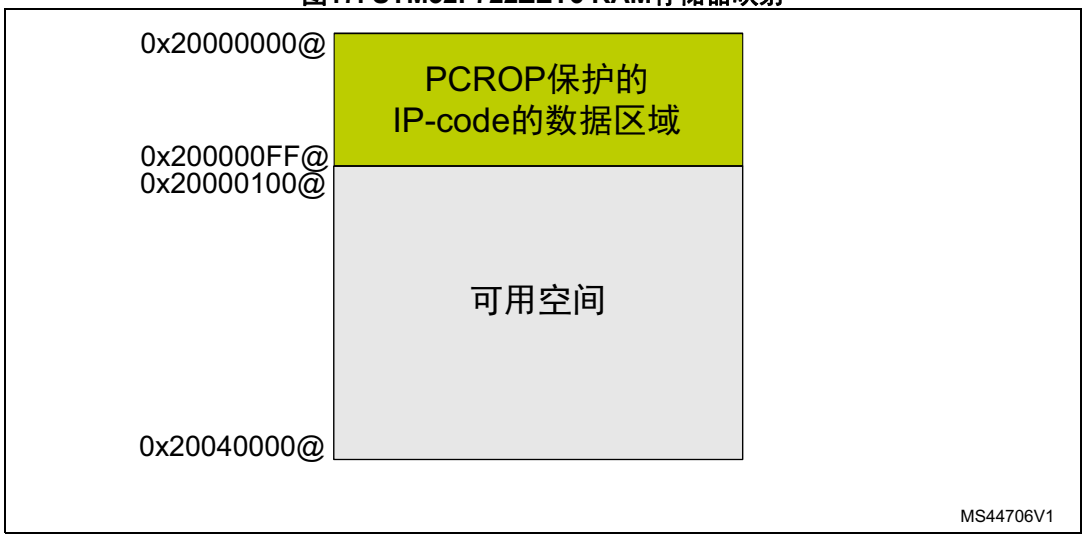


存储器布局由Keil® IDE的分散文件、IAR中的链接器配置文件（ICF）和SW4STM32中的链接器文件进行管理。

另一个存储区域必须在RAM存储器上创建，该RAM存储器专门用于PCROP保护的IP-Code的RW和零初始化数据。这些数据在级别n和级别n+1上拥有相同的地址，所以在两个级别上必须为它们保留此相同存储区域，避免重写级别n+1主程序的数据。

所产生的存储器映射如*图 17*所示。

图17. STM32F722ZET6 RAM存储器映射

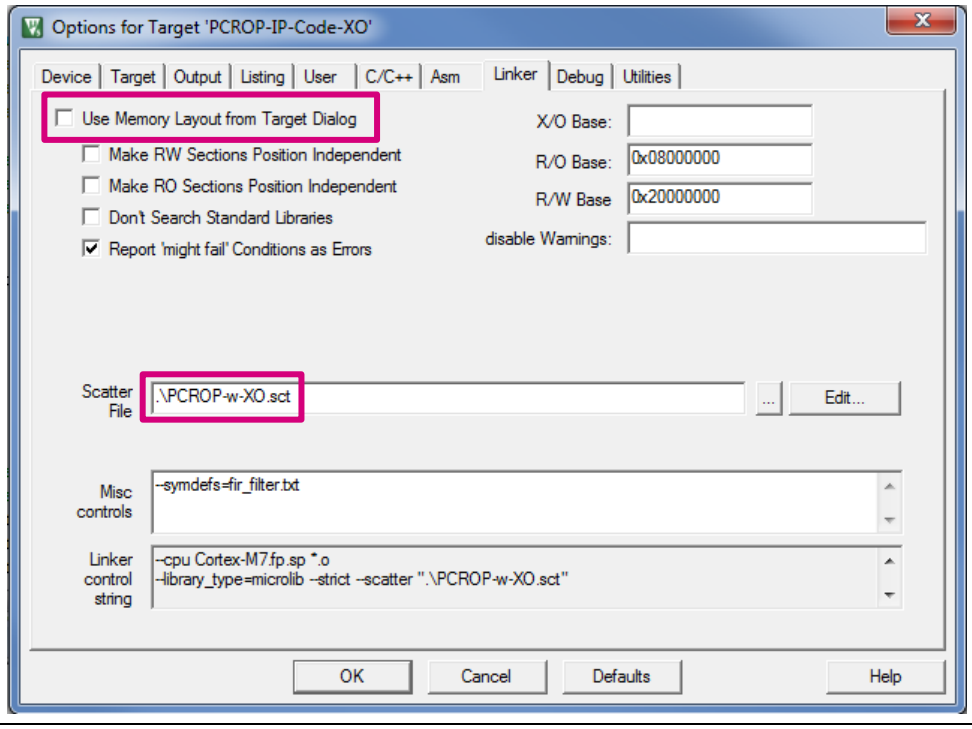


Keil®分散文件

要将IP-code放入Sector 2，需按照以下顺序：

进入Project → Options for Target，选择连接器选项卡，然后取消选中“Use Memory Layout from Target Dialog”，如*图 18*中所示，然后点击Edit按钮来修改PCROP-w-XO.sct分散文件。

图18. 分散文件修改



必须更新这两个新区域的分散文件：

- 位于闪存中的LR_PCROP，对PCROP保护的IP-code具有“只执行”访问属性
- 位于闪存中的LR_DATA，为常量数据存储区域，用于IP-code。定义一个新扇区，名为“fir_coeff_section”。
- FirBSSRAM位于RAM存储器中，用于来自PCROP保护扇区的数据

```
; *************************************************************
; *** 分散加载描述文件，由uVision生成 ***
; *************************************************************
LR_IROM1 0x08000000 0x00004000 {   ; 载入区域size_region
  ER_IROM1 0x08000000 0x00004000 { ; 载入地址 = 执行地址
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
    FirBSSRAM 0x20000000 0x100 ; fir滤波器的RW和零初始化数据
  {
     fir_filter.o (+RW +ZI)
  }
  RW_IRAM1 0x20000100 0x0003FF00 { ; 主程序的RW数据
   .ANY (+RW +ZI)
  }
}
; *********************
; *** PCROP保护区域 ***
; *********************
LR_PCROP 0x08008000 0x00004000 {
  ER_PCROP 0x08008000 0x00004000 { ; 载入地址 = 执行地址
     arm_fir_init_f32.o (+XO)
     arm_fir_f32.o (+XO)
     fir_filter.o (+XO)
  }
}
; *************************************************
; *** FIR系数的写保护数据区域 ***
; **** ***********************************************
LR_DATA 0x0800C000 0x00004000 {
  fir_coef_section 0x0800C000 0x00004000 {
     FIR_Filter.o (+RO)
```

IAR ICF文件

定义了三个存储区域

- 位于闪存中的LR_PCROP，对PCROP保护的IP-code具有“只执行”访问属性
- 位于闪存中的LR_DATA，为常量数据存储区域，用于IP-code。定义一个新扇区，名为“fir_coeff_section”。
- FirBSSRAM位于RAM存储器中，用于来自PCROP保护扇区的数据

请注意，编译对象（fir_filter.o）包含代码和全局常量数据。代码和文件之间的分隔在ICF文件中是显式的。

```
/* 定义FirBSSRAM区域起始和结束地址 */
define symbol __ICFEDIT_region_FirBSSRAM_start__  = 0x20000000;
define symbol __ICFEDIT_region_FirBSSRAM_end__   = 0x200000FF;
define region FirBSSRAM_region    = mem:[from __ICFEDIT_region_FirBSSRAM_start__   to
__ICFEDIT_region_FirBSSRAM_end__];

/* 定义PCROP区域起始和结束地址 */
define symbol __ICFEDIT_region_PCROP_start__    = 0x08008000;
define symbol __ICFEDIT_region_PCROP_end__      = 0x0800BFFF;
define region PCROP_region    = mem:[from __ICFEDIT_region_PCROP_start__  to
__ICFEDIT_region_PCROP_end__];

/* 定义fir系数常量数据的写保护区域 */
define symbol __ICFEDIT_region_CONST_start__      = 0x0800C000;
define symbol __ICFEDIT_region_CONST_end__      = 0x0800C800;
define region fir_coef_region    = mem:[from __ICFEDIT_region_CONST_start__   to
__ICFEDIT_region_CONST_end__];

/* 将待PCROP保护的IP-code放在扇区2中 */
place in PCROP_region   { ro object arm_fir_f32.o,
                          ro object arm_fir_init_f32.o,
                          ro object FIR_Filter.o};

/* 将待WP保护的常量数据（用于IP-code）放置在扇区3中 */
place in fir_coef_region { ro data section fir_coef_section object FIR_Filter.o };

/* 放置FirBSSRAM内存区域中PCROP保护扇区的RW和ZI数据*/
place in FirBSSRAM_region   {
rw object FIR_Filter.o
zi  object FIR_Filter.o};
```

SW4STM32链接器文件

定义了三个存储区域

- 位于闪存中的LR_PCROP，对PCROP保护的IP-code具有“只执行”访问属性
- 位于闪存中的LR_DATA，为常量数据存储区域，用于IP-code。定义一个新扇区，名为“fir_coeff_section”。
- FirBSSRAM位于RAM存储器中，专门用于来自PCROP保护扇区的数据

```
/* 指定存储区域 */
MEMORY
{
RAM (xrw)      : ORIGIN = 0x20000100, LENGTH = 256K - 0x100
FirBSSRAM (xrw): ORIGIN = 0x20000000, LENGTH = 0x100
FLASH (rx)     : ORIGIN = 0x08000000, LENGTH = 16K
PCROP (x)      : ORIGIN = 0x08008000, LENGTH = 16K
CONST (rx)     : ORIGIN = 0x0800C000, LENGTH = 16K
}
  /* 将待PCROP保护的IP-code放在扇区2中 */
.PCROPedCode :
{
  。= ALIGN(4);
   *arm_fir_f32.o  (.text .text*)
   *arm_fir_init_f32.o (.text .text*)
   *fir_filter.o (.text .text*)
  。= ALIGN(4);
 } >PCROP
    /* 将待WP保护的常量数据（用于IP-code）放置在扇区3中 */
  .WPedData :
{
  。= ALIGN(4);
   *fir_filter.o (.fir_coef_section .rodata*)
  。= ALIGN(4);
 } >CONST
/* 放置FirBSSRAM内存区域中PCROP保护扇区的RW和ZI数据*/
.FirBss :
{    _sFirbss = .;        /*FirrBss开始时定义一个全局符号 */
   *fir_filter.o (.bss .bss*)
    。= ALIGN(4);
    _eFirbss = .;        /*FirBss结束时定义一个全局符号 */
 } >FirBSSRAM
```

显式数据放置

由于下面的编译属性，常量数据将映射到此扇区。以下代码摘要显示了IAR、SW4STM32和Keil®的语法。

```
/* IAR IDE */
```

```
/* 采用专门地址进行放置：*/
const float32_t firCoeffs32[NUM_TAPS] @ 0x0800C000 = { …};
/* 采用ICF文件中定义的专门扇区进行放置： */
const float32_t firCoeffs32[NUM_TAPS] @ "fir_coef_section" = { ….};
/* KEIL IDE */
/* 采用专门地址进行放置： */
const float32_t firCoeffs32[NUM_TAPS] __attribute__((at(0x0800C000))) ={…};
/* 采用分散文件中定义的专门扇区进行放置： */
const float32_t firCoeffs32[NUM_TAPS] __attribute__((section("fir_coef_section"))) = {…};
/* SW4STM32 IDE */
/* 采用专门地址进行放置： */
const float32_t firCoeffs32[NUM_TAPS] __attribute__((section(".0x0800C000"))) ={…};
/* 采用ld文件中定义的专门扇区进行放置： */
const float32_t firCoeffs32[NUM_TAPS] __attribute__((section(".fir_coef_section"))) = {…};
```

2.3.4 常量写保护

所有IP-code常量（例如，整数，浮点数或字符串）必须放在PCROP保护的区域之外，因为它们不能被D-code总线进行访问。

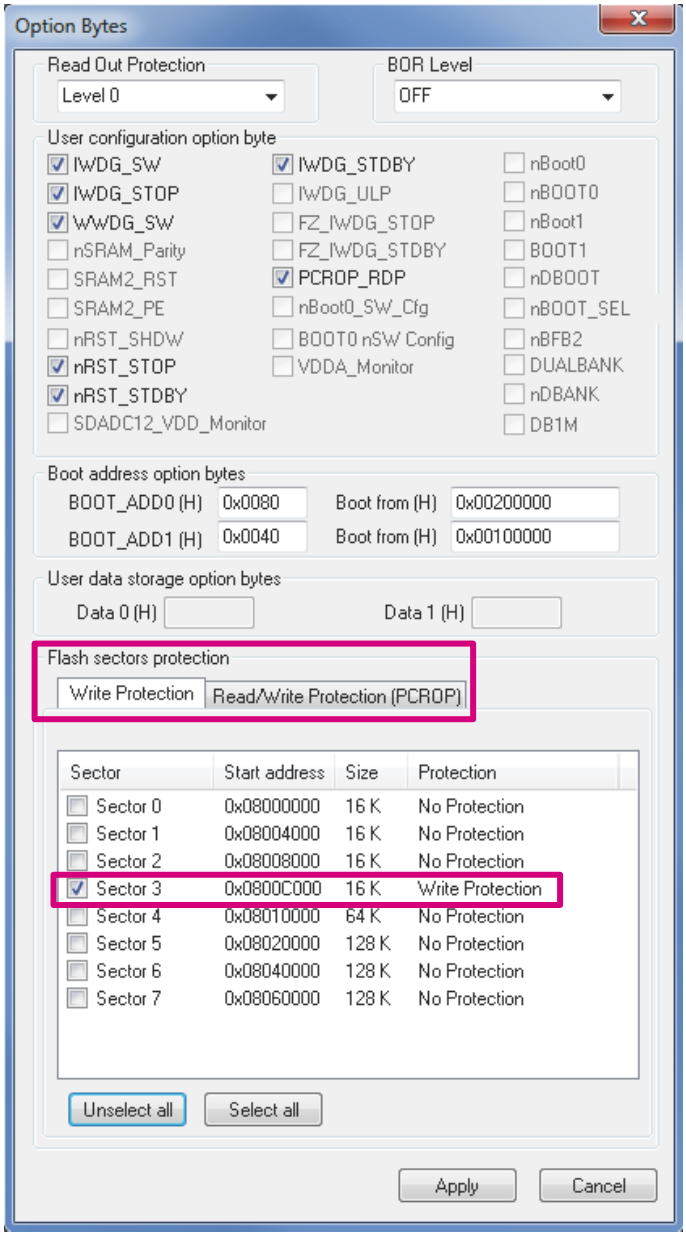
用户还必须考虑到，IP-code常量可被ST用户级别n+1所删除或修改，函数可能变成无用的，因此建议对这些常量进行保护，避免不需要的写入/擦除。

通过清除Flash选项控制寄存器（FLASH_OPTCR）的位nWRPi，可在扇区i上设置写保护区域。

示例的IP-code使用了116个字节作为常量系数。最小页大小为16 KB，因此单个页可以被保护（地址0x0800C000至0x08010000之间）。

写保护可通过专用固件代码在专用寄存器中设置正确值（参见PCROP_Enable()函数）来进行设置，或利用STLink-Utility进行设置，如*图 19*中所示。

图19. 使用STM32 STlink Utility使能写保护



2.3.5 保护IP-code

使用PCROP_Enable()函数激活PCROP

使用STEP1-ST_用户级别_n project main.c文件中定义的PCROP_Enable()函数，激活Sector 2上的PCROP，从而实现IP-code保护

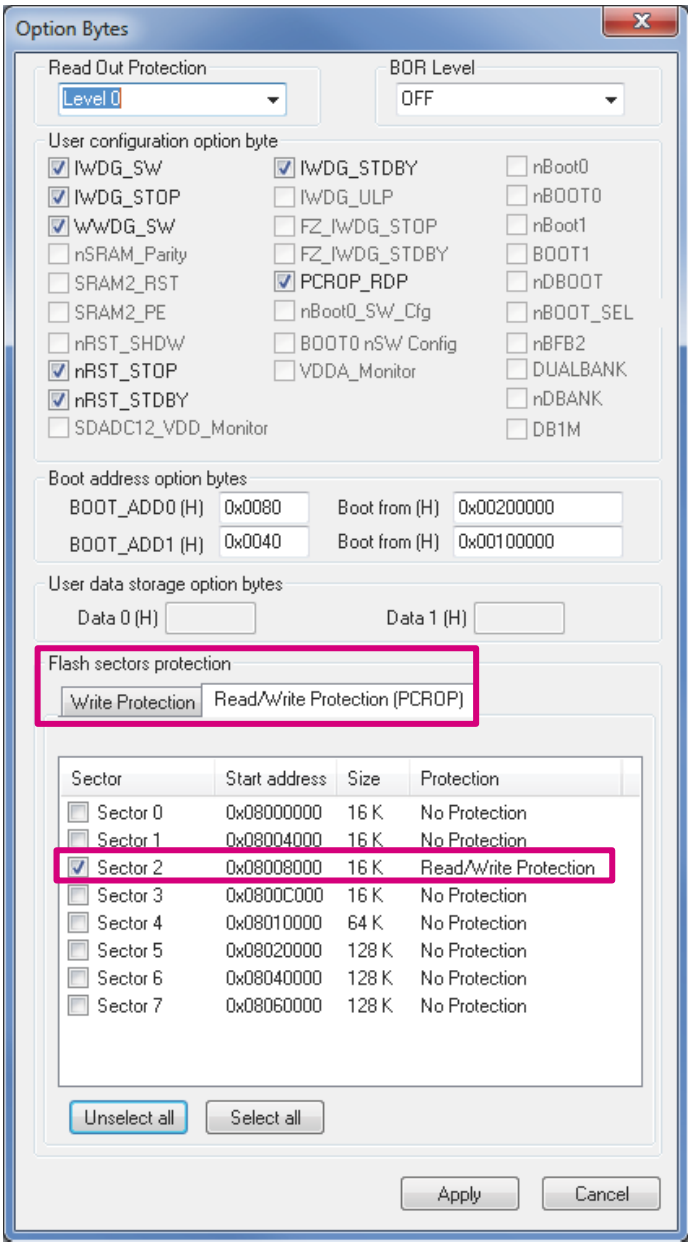
当此函数至少执行了一次时，Sector 2变为读/写保护的，任何IP-code修改都需要禁用PCROP，然后全部闪存内容都会被擦除。

使用STM32 STLink Utility激活PCROP

要利用STM32 STLink激活Sector 2上的PCROP，用户必须按照下面所示的顺序：

- 板子上电，然后在STLink Utility接口进入Target → Option bytes。
- 下面的窗口中，将闪存保护模式设置为读/写保护，并使能Sector 2上的保护，如*图 20*中所示。
- 点击Apply按钮。

图20. 使用STM32 STlink Utility使能PCROP



2.3.6 执行PCROP保护的IP-code

一旦IP-code被编程在Sector 2上并且受到保护，则必须通过调用用户代码中的函数来对其进行测试。

本章介绍如何在具有两种配置的STEP1-ST_用户级别_n项目中执行PCROP保护的IP-code，注意PCROP-IP-code配置仅用于测试，不能用于STEP2。
PCROP-IP-code-XO才是正确的配置，这里编译器被设置为产生只执行IP-code。这是用来运行STEP2-ST_用户级别_n+1项目的配置。

PCROP-IP-code-XO（必须在STEP2之前使用）

编译器配置为生成只执行IP-code，避免对其进行数据读取（避免文字池）。

1. 打开位于STEP1-ST_用户级别_n目录中的项目，选择您首选的工具链
2. 选择PCROP-IP-code-XO配置
3. 重建所有文件。
4. 按照以下顺序运行示例：
   a) 为板子上电，在载入代码之前，检查是否存在任何PCROP保护或写保护的扇区。如果有，则使用STM32 STLink Utility禁用该保护，然后载入代码。程序加载完毕后，蓝色LED应连续闪烁；
   b) 按下用户按钮键来激活PCROP保护，当此完成时，绿色LED会亮起，Sector 2为PCROP保护扇区，否则蓝色LED亮起，PCROP激活失败；
   c) 按下用户按钮键来执行在main.c文件中调用的PCROP保护IP-code，绿色LED应连续闪烁。

PCROP-IP-code（仅用于测试，不能用于STEP2）

没有使用特殊的编译器选项，仅用于测试目的，说明对于PCROP保护的代码，必须避免代码中的数据（如文字池和分支表）。

1. 在位于STEP1-ST_用户级别_n目录的同一个项目中，选择PCROP-IP-code配置
2. 重建所有文件。
3. 按照以下顺序运行示例：
   a) 为板子上电，在载入代码之前，检查是否存在任何PCROP保护或写保护的扇区。如果有，则利用STM32 STLink Utility禁用该保护，然后载入代码；当程序加载完毕时，蓝色LED应连续闪烁；
   b) 按下用户按钮键来激活PCROP保护，当此完成时，绿色LED会亮起，Sector 2被PCROP保护，否则红色LED亮起，PCROP激活失败；
   c) 按下用户按钮键来执行在main.c文件中调用的PCROP保护IP-code，会产生一个错误操作中断，启动系统复位，并且蓝色LED连续闪烁。

解释

低通滤波器函数计算testInput_f32_1kHz_15kHz输入信号，必须输出1 KHz正弦波。输出数据（testOutput）随后与MATLAB已计算出的参考（refOutput）进行比较，如果二者匹配，则绿色LED连续闪烁，否则蓝色LED连续闪烁。

对于PCROP-IP-code配置，其中PCROP保护的IP-code包含文字池：执行IP-code（FIR_lowpass_filter()函数）时，不能通过D-code总线访问文字池，然后RDERR标志置位。OPERR标志也被置位，并产生一个读操作错误中断，然后在HAL_FLASH_OperationErrorCallback()函数中启动系统复位，蓝色LED连续闪烁。

但是对于PCROP-IP-code-XO配置，IP-code可正确执行，绿色LED必须连续闪烁。

注： *更多详细内容，请参考嵌入式软件版中的readme.txt。*

2.3.7 创建头文件并生成符号定义文件

要提供给ST用户级别n+1的头文件包含要使用的IP-Code函数定义。
本例中包含在main.c文件中的是fir_filter.h文件。

利用IAR产生符号定义文件

IDE中，要导出PCROP保护的IP-code符号，请选择Project→Options→Build Actions，并在Post-build命令行文本字段中指定以下命令行，如*图 21*所示：

$TOOLKIT_DIR$\bin\isymexport.exe --edit "$PROJ_DIR$\steering_file.txt" "$TARGET_PATH$" "$PROJ_DIR$\fir_filter.o"

这里，创建steering_file.txt用作选项，使生成的符号文件中只有IP-code函数符号。

要使符号文件中只有IP-code函数，则必须创建steering_file.txt，如下所示：

show arm_fir_f32

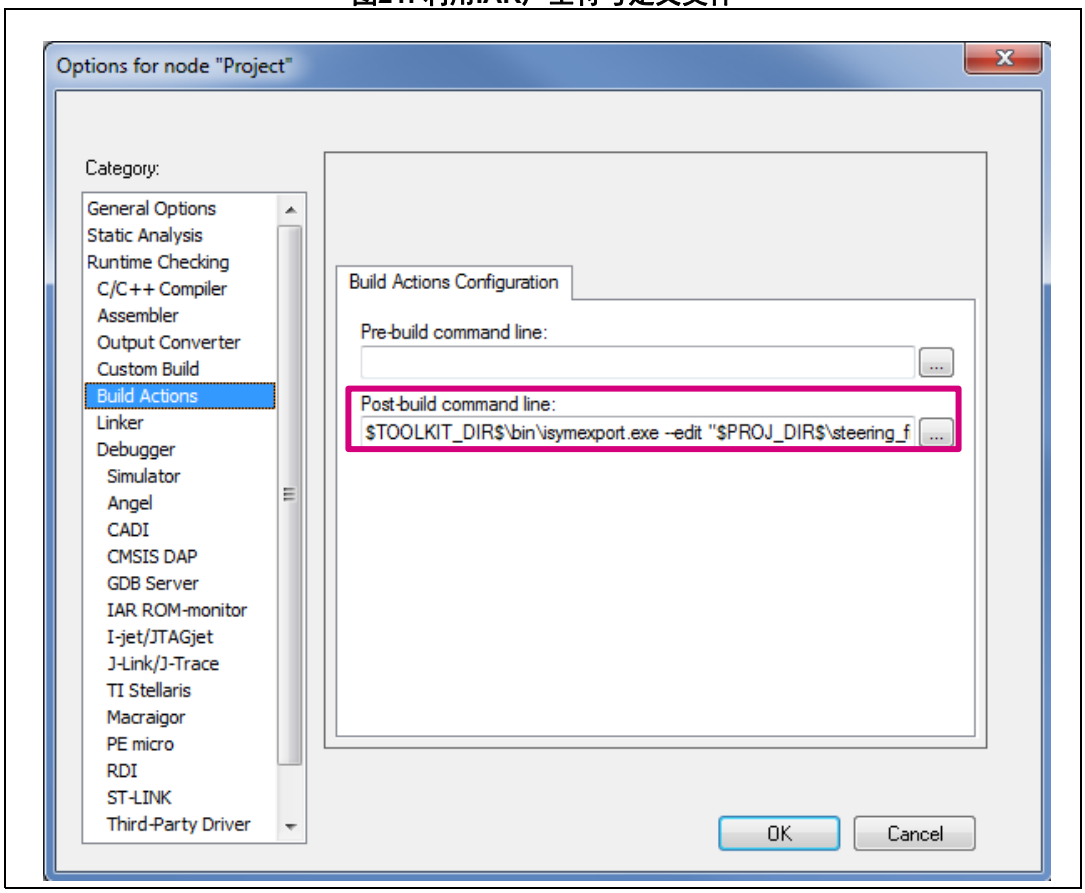
show arm_fir_init_f32

show FIR_lowpass_filter

这里“show”命令用来保留所生成符号定义文件中的首选函数。

所生成符号定义文件（fir_filter.o）可通过将其添加到STEP2-ST_用户级别_n+1 IAR项目中来使用。

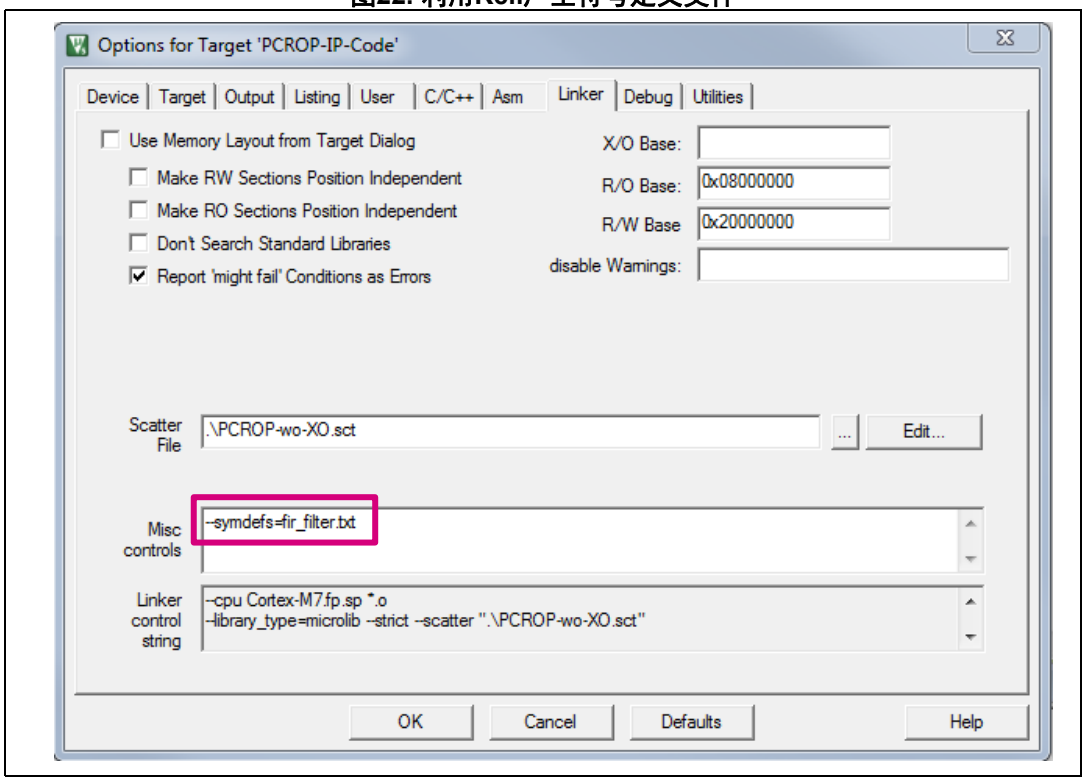
图21. 利用IAR产生符号定义文件



利用Keil®产生符号定义文件

要利用Keil®生成符号定义文件，请进入Linker选项卡中的Project选项，添加命令--symdefs=fir_filter.txt并重新构建项目（参见*图 22*）。

图22. 利用Keil产生符号定义文件



名为fir_filter.txt的符号定义文件随后会在STEP1-ST_用户级别_n\MDK-ARM\PCROP-IP-code-XO目录中创建。

该文件包含项目的所有符号，因此只能保留最终用户所调用的那些IP-code函数。所有其他函数都必须被删除，产生的符号定义文件为：

符号定义文件fir_filter.txt：

#<SYMDEFS># ARM Linker, 5060300: Last Updated: Wed Dec 07 11:38:09 2016

0x08008001 T FIR_lowpass_filter

0x08008049 T arm_fir_f32

0x0800836d T arm_fir_init_f32

利用SW4STM32产生符号定义文件

IDE中，要导出PCROP保护的IP-code符号，请选择Project→Propities→C/C++ Build→Settings→Build Steps，并在Post-build步命令行文本字段中指定以下命令行，如*图 23*所示：

arm-none-eabi-objcopy -O ihex "${BuildArtifactFileBaseName}.elf" "${BuildArtifactFileBaseName}.hex" && arm-none-eabi-size "${BuildArtifactFileName}" && arm-none-eabi-objcopy -K arm_fir_f32 -K FIR_lowpass_filter -K arm_fir_init_f32 --only-section=.PCROPedCode --only-section=.WPedData --only-section=.FirBss "${BuildArtifactFileBaseName}.elf" fir_filter.o

该命令仅将以下IP-code函数保存在符号文件中

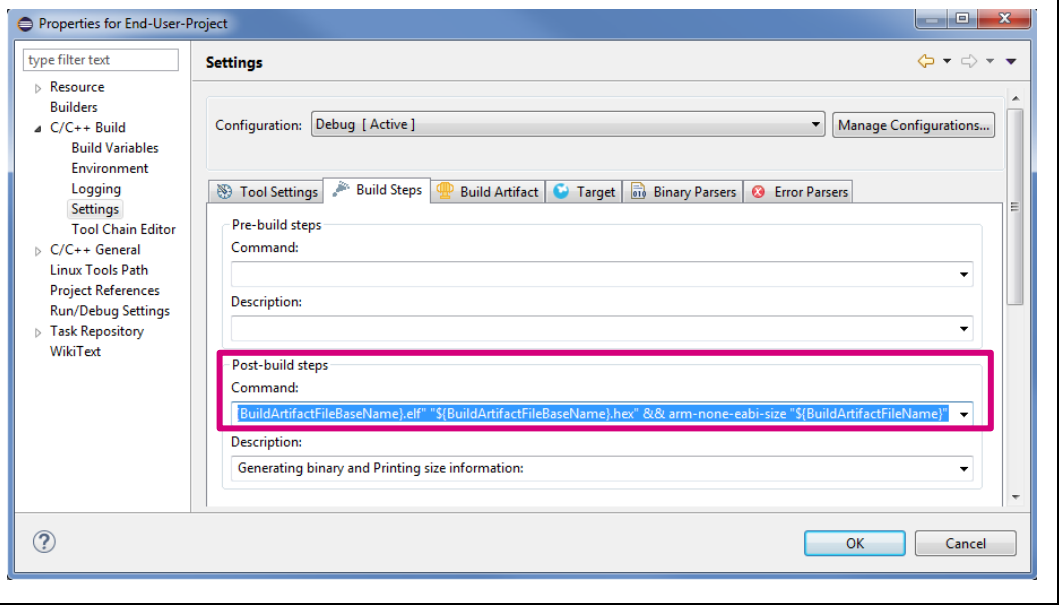
arm_fir_f32

arm_fir_init_f32

FIR_lowpass_filter

所生成符号定义文件（fir_filter.o）可通过将其添加到STEP2-ST_用户级别_n+1 SW4STM32项目中来使用。

图23. 利用SW4STM32产生符号定义文件



2.4 STEP2：ST用户级别n+1

ST用户级别n+1具有来自用户级别n的，预加载PCROP保护的IP Code的STM32F722ZE MCU，提供的符号定义文件以及头文件。

然后参考由ST用户级别n提供的Flash和RAM存储器映射，ST用户级别n+1必须：

- 创建最终项目，
- 在其项目中包含ST用户级别n提供的的头文件以及添加其提供的符号定义文件，
- 调用PCROP保护的IP-code函数，
- 执行并调试最终用户应用程序。

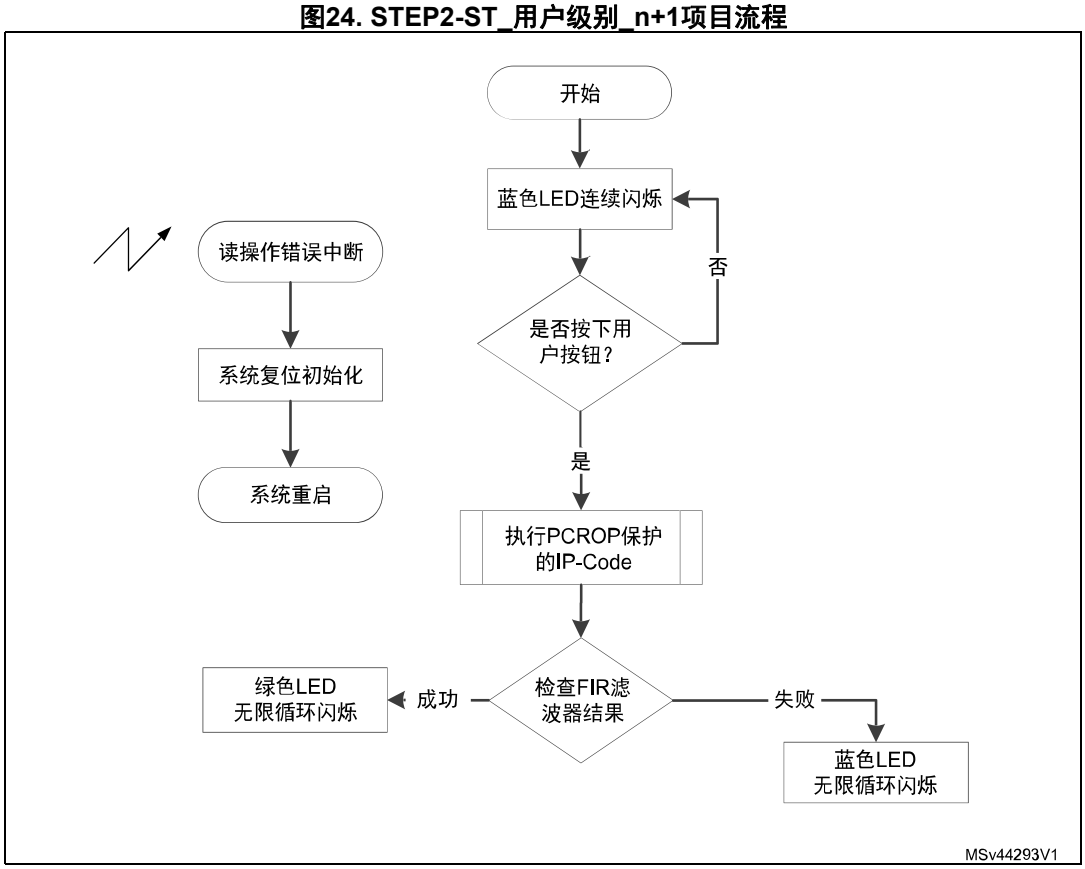
注意： ST用户级别n+1必须使用与ST用户级别n完全相同的工具链和编译器版本，来开发和编程IP-code，否则ST用户级别n+1不能使用该IP-code。

在所提供的示例中，对于两个项目STEP1-ST_用户级别_n和STEP2-ST_用户级别_n+1，用户必须使用同样的工具链和编译器版本。例如，如果使用了MDK-ARM工具链V5.21a来运行STEP1-ST_用户级别_n项目，必须也使用它来运行STEP2-ST_用户级别_n+1项目。

2.4.1 项目流程

*图 24*说明了STEP2-ST_用户级别_n+1项目流程。如果从/向PCROP保护的IP-code发送任何读/写请求，则会产生一个读操作错误中断，启动系统复位，然后系统重启，蓝色LED连续闪烁。

图24. STEP2-ST_用户级别_n+1项目流程



2.4.2 创建一个最终用户项目

受保护的代码和写保护数据段以及为来自PCROP扇区保留的RAM区域位于存储器映射（*图 16*和*图 17*）的已知位置，因此用户必须注意，在实现链接器文件时，应避免将代码放置在这些区域。

PCROP保护的IP-code函数随后用来创建最终用户应用程序。位于STEP2-ST_用户级别_n+1目录中的项目是一个示例，其中PCROP保护的FIR滤波器函数在main.c文件中被调用。

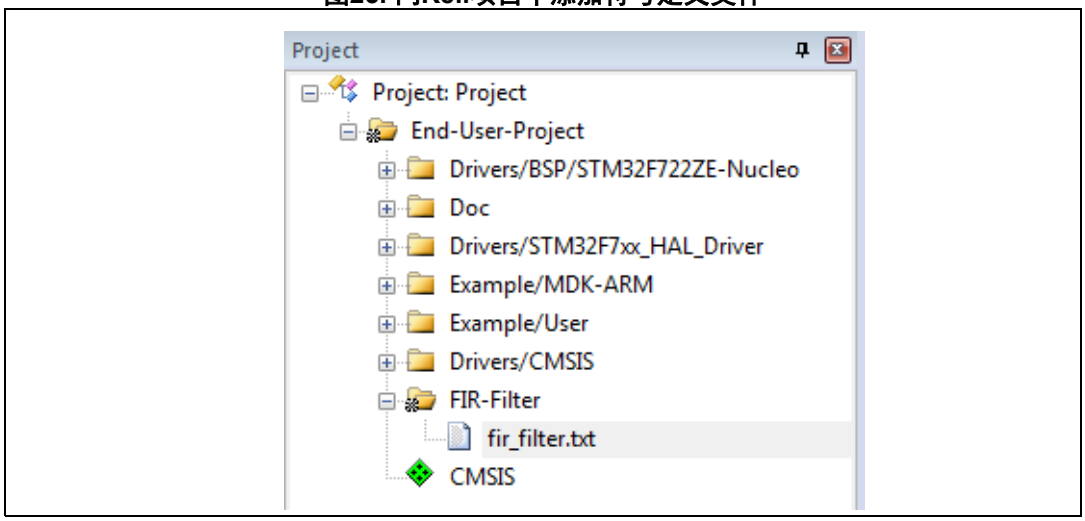
2.4.3 包含头文件并添加符号定义文件

ST用户级别n所提供的符号定义文件必须添加为传统源文件，这是在链接STEP2-ST_用户级别_n+1项目中PCROP保护的IP-code所必需的。

向Keil®项目中添加符号定义文件

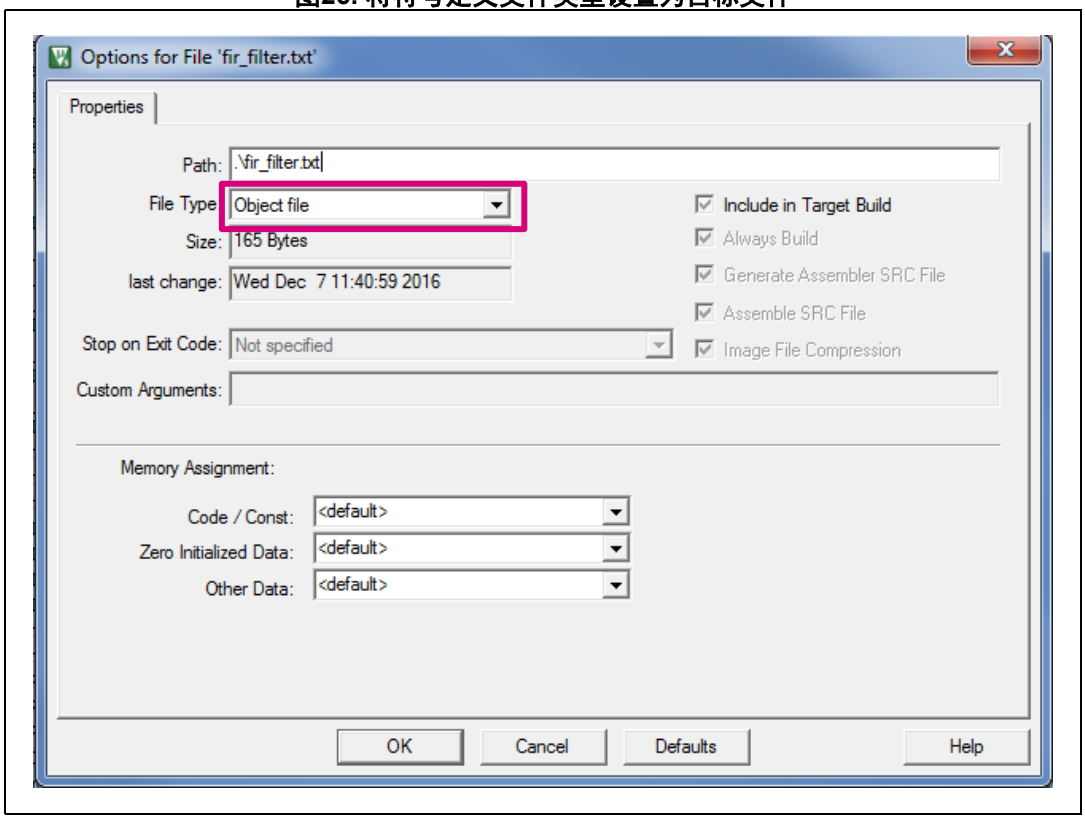
符号定义文件由*第 2.3.7节：创建头文件并生成符号定义文件*中所述的ST用户级别n提供，本例中名为fir_filter.txt，它必须添加到FIR-Filter组中，如*图 25*中所述。

图25. 向Keil项目中添加符号定义文件



所添加的fir_filter.txt文件类型必须改为Object文件，而不是文本文档文件，如*图 26*中所示。

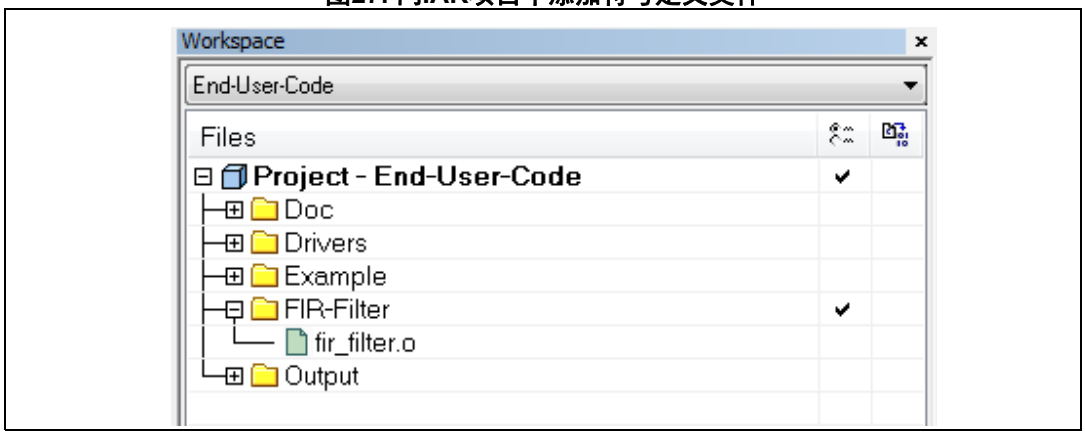
图26. 将符号定义文件类型设置为目标文件



向IAR项目中添加符号定义文件

必须将fir_filter.o文件作为目标文件添加到组FIR_Filter中，如*图 27*中所示。

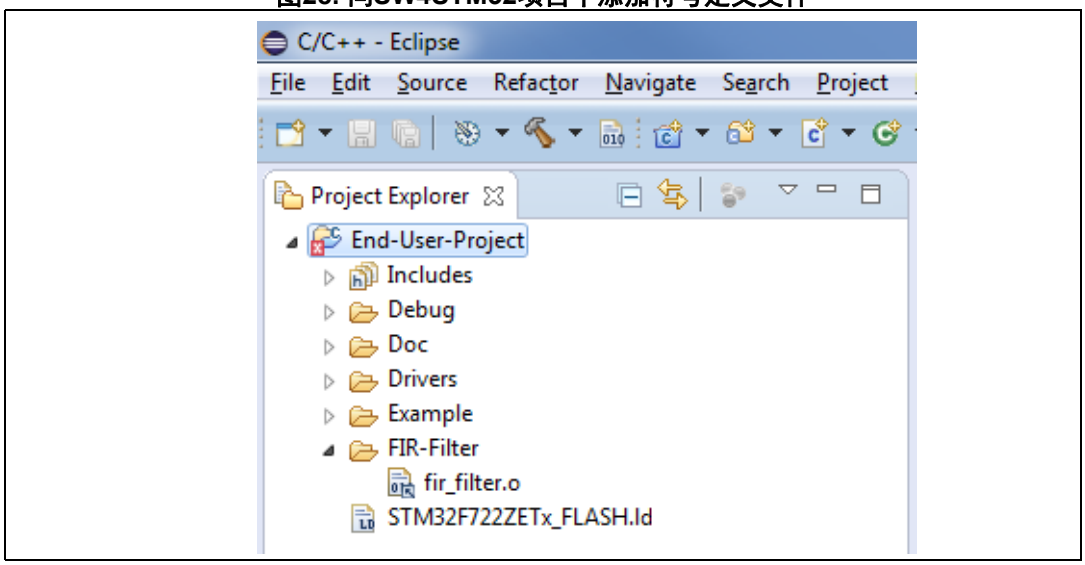
图27. 向IAR项目中添加符号定义文件



向SW4STM32项目中添加符号定义文件

必须将fir_filter.o文件作为目标文件添加到组FIR_Filter中，如*图 28*中所示。

图28. 向SW4STM32项目中添加符号定义文件



包含头文件

头文件（fir_filter.h）包含在main.c文件中，然后用户可以调用FIR Filter函数。

为来自PCROP保护扇区的数据保留RAM存储器区域

由于来自PCROP扇区的数据在项目级别n和n+1的RAM存储器上具有相同的地址，因此级别n的RAM存储器上为这些数据所创建的区域，在级别n+1上也必须为其保留。

Keil®分散文件

必须更新FirBSSRAM存储区域的分散文件：

```
; *************************************************************
；*** 分散加载描述文件，由uVision生成 ***
; *************************************************************
LR_IROM1 0x08000000 0x00004000 {   ；载入区域size_region
  ER_IROM1 0x08000000 0x00004000 {  ；载入地址 = 执行地址
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
    FirBSSRAM  0x20000000 EMPTY 0x100  {  ；为fir滤波器的RW和零初始化数据保留
  }
  RW_IRAM1 0x20000100 0x0003FF00  {  ；主程序的RW数据
   .ANY (+RW +ZI)
  }
}
```

IAR ICF文件

必须更新FirBSSRAM存储区域的icf文件：

```
/* 定义FirBSSRAM区域起始和结束地址 */
define symbol __ICFEDIT_region_FirBSSRAM_start__  = 0x20000000;
define symbol __ICFEDIT_region_FirBSSRAM_end__   = 0x200000FF;
define region FirBSSRAM_region    = mem:[from __ICFEDIT_region_FirBSSRAM_start__   to
__ICFEDIT_region_FirBSSRAM_end__];

/* 放置FirBSSRAM内存区域中PCROP保护扇区的RW和ZI数据*/
place in FirBSSRAM_region  {
rw object FIR_Filter.o
zi  object FIR_Filter.o};
```

SW4STM32链接器文件

使用STM4STM32，　必须更新FirBSSRAM存储区域的链接器文件，最好也指定PCROP保护和写保护扇区：

```
/* 指定存储区域 */
MEMORY
{
RAM (xrw)     : ORIGIN = 0x20000100, LENGTH = 256K - 0x100
FirBSSRAM (xrw): ORIGIN = 0x20000000, LENGTH = 0x100
FLASH (rx)    : ORIGIN = 0x08000000, LENGTH = 16K
PCROP (x)     : ORIGIN = 0x08008000, LENGTH = 16K
CONST (rx)    : ORIGIN = 0x0800C000, LENGTH = 16K
}
.PCROPedCode (NOLOAD): { *(.PCROPedCode)   } >PCROP
.WPedData (NOLOAD):   { KEEP(*(.WPedData)) } >CONST
```

```
.FirBss :          { KEEP(*(.FirBss)) } >FirBSSRAM
```

2.4.4 调用PCROP保护的IP-code函数

当fir_filter.h头文件包含在main.c文件中，并且符号定义文件已添加到项目中时，可调用PCROP保护的IP-code函数。

FIR_lowpass_filter()函数（在fir_filter.c文件中有描述）在主文件中被调用，如下所示：

FIR_lowpass_filter(inputF32, outputF32, TEST_LENGTH_SAMPLES)

其中：

- inputF32：指针，指向包含输入信号数据的表；
- outputF32：指针，指向已处理输出信号数据所填充的表；
- TEST_LENGTH_SAMPLES：待处理输入中的采样数。

2.4.5 运行最终用户应用程序

在此阶段IP-code已经由STM32F722ZE MCU进行预加载并PCROP保护，因此STEP1-ST_用户级别_n项目（PCROP-IP-code-XO配置）必须在运行该项目前进行加载和执行。

要使程序工作，请按照下述步骤操作：

1. 打开位于STEP2-ST_用户级别_n+1目录中的项目，选择与STEP1中相同的工具链。
2. 重建所有文件。
3. 按照以下顺序运行示例
   a) 为板子上电，并加载代码（这里仅加载用户代码）
   b) 按下用户按钮键来执行在main.c文件中调用的PCROP保护IP-code，绿色LED必须连续闪烁。

注： *更多详细内容，请参考嵌入式软件版中的readme.txt。*

2.4.6 调试模式中的PCROP保护

在开发其最终用户应用程序时，ST用户级别n+1需要调试其代码。因此，在调试最终用户应用程序时，PCROP保护必须阻止任何对ST用户级别n所开发的IP-code的读访问。

本章中，我们介绍在调试用户代码时，PCROP如何保护预编程IP-code不被读取。

下述调试示例在Keil®项目上完成。

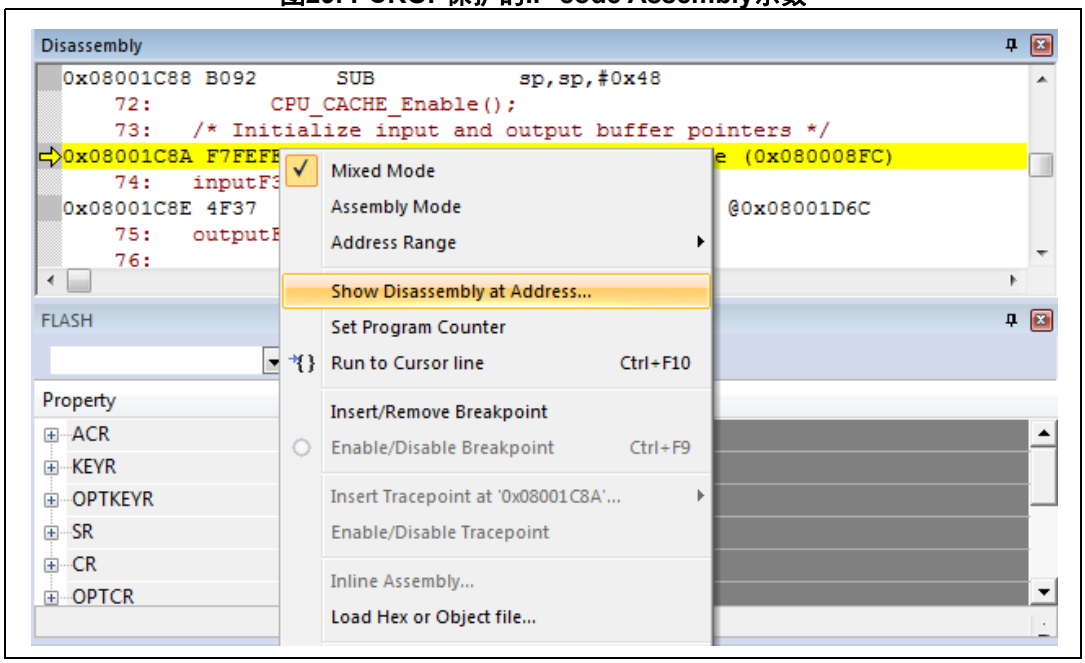
调试最终用户应用程序

调试最终用户项目之前，必须加载并执行STEP1-ST_用户级别_n项目（PCROP-IP-code-XO工作空间），以使STM32F722ZE MCU具有预编程且PCROP保护的IP-code。

要调试STEP2-ST_用户级别_n+1项目，请按照下述步骤操作：

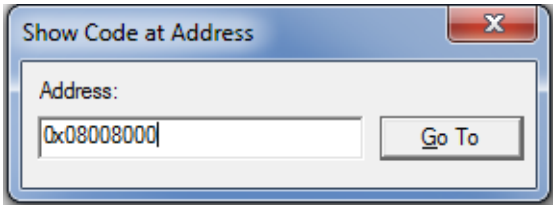
1. 打开位于STEP2-ST_用户级别_n+1目录中的项目，选择与STEP1中相同的工具链。
2. 重建所有文件。
3. 点击Start/Stop Debug会话来开始调试。
4. 点击Run来运行程序，蓝色LED会连续闪烁。
5. 按下User按钮来执行IP-code，绿色LED会连续闪烁。
6. 要访问PCROP保护的IP-code，右键点击Disassembly窗口，然后选择Show Disassembly at Address，如*图 29*中所示。

图29. PCROP保护的IP-code Assembly示数



7. 在地址栏填写PCROP保护扇区起始地址，并单击“Go To按钮”，如*图 30*中所示。

图30. 填写PCROP保护扇区起始地址

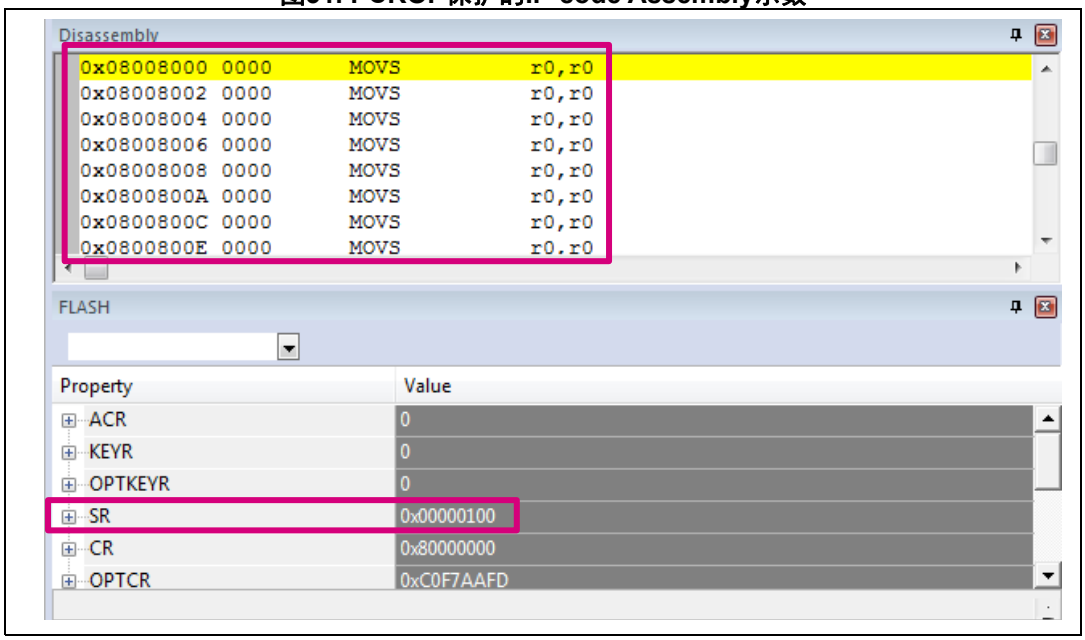


如*图 31*中所示，载入到Sector 2中的PCROP保护IP-code是不可读的，而位于0x08008000地址之前的代码可以被读取。读取PCROP保护扇区会将FLASH_SR寄存器中的RDERR和OPERR标志置位。

由于调试时通过D-code总线进行了闪存读操作，因此会产生一个闪存操作错误中断。

然后在HAL_FLASH_OperationErrorCallback()函数（在main.c文件中有描述）中启动软件复位，蓝色LED会连续闪烁。

图31. PCROP保护的IP-code Assembly示数



3 结论

STM32F72xxx和STM32F73xxx微控制器提供了灵活有用的读和/或写保护功能，可用于需要保护的应用。本应用笔记描述了如何使用这些读、写和PCROP保护功能。

4 版本历史

表3. 文档版本历史

日期	版本	变更
2017 年 2 月 15 日	1	初始版本。

表4. 中文文档版本历史

日期	版本	变更
2017 年8月25日	1	中文初始版本。