
**Proprietary code read-out protection on microcontrollers
of the STM32F4 Series**

Introduction

Software providers are developing complex middleware solutions (Intellectual Property code, or IP-Code), protecting them is an issue of high importance for microcontrollers.

In order to respond to this important requirement, STM32F4xx MCUs feature:

- Read Protection (RDP): protection against read operations
- Write Protection: protection against undesired write or erase operations
- Proprietary Code Read Out Protection (PCROP): protection against read and write operations.

This application note provides a description of these Flash memory protection techniques, focusing on the Proprietary Code Read Out Protection (PCROP) and providing a basic example of PCROP protection.

The X-CUBE-PCROP firmware package delivered with this document contains the source code of the PCROP example with all firmware modules required to run the example.

This application note has to be read in conjunction with reference manuals as listed in [Table 1](#). These documents, as well as the corresponding datasheets, are all available at www.st.com.

Table 1. Reference documents

Product line	Reference manual
STM32F401	RM0368
STM32F410	RM0401
STM32F411	RM0383
STM32F412	RM0402
STM32F413	RM0430
STM32F42x/STM32F43x	RM0090
STM32F446	RM0390
STM32F469/479	RM0386

Contents

- 1 Read Out Protection (RDP) 6**
 - 1.1 Read protection Level 0 6
 - 1.2 Read protection Level 1 6
 - 1.3 Read protection Level 2 6
 - 1.4 Internal Flash memory content update on RDP protected
STM32F4xx microcontrollers 7

- 2 Write protection 8**

- 3 Proprietary Code Read Out Protection (PCROP) 9**
 - 3.1 PCROP protection overview 9
 - 3.2 How to enable the PCROP protection 9
 - 3.3 How to disable the PCROP protection 10
 - 3.4 Placing and executing PCROP-ed IP-Code11
 - 3.4.1 Don't PCROP vector table sector 11
 - 3.4.2 PCROP-ed IP-Code dependency 11
 - 3.5 PCROP feature availability on STM32F4 Series 12

- 4 PCROP example 13**
 - 4.1 Example requirements 13
 - 4.1.1 Hardware requirements 13
 - 4.1.2 Software requirements 13
 - 4.2 Example overview 13
 - 4.2.1 Software settings 15
 - 4.3 PCROP-ed IP-Code: FIR low pass filter 15
 - 4.4 STEP1: ST Customer level n 16
 - 4.4.1 Generating an execute-only IP-Code 16
 - 4.4.2 Placing the IP-Code 20
 - 4.4.3 Placing constants outside of PCROP-ed sector 21
 - 4.4.4 Protecting the IP-Code 22
 - 4.4.5 Step1-ST_Customer_level_n project flow 23
 - 4.4.6 Executing PCROP-ed IP-Code 25
 - 4.4.7 Creating header file and generating symbol definition file 26

4.5	STEP2: ST Customer level n+1	29
4.5.1	Creating an end user project	29
4.5.2	Including header file and adding symbol definition file	30
4.5.3	Calling PCROP-ed IP-Code functions	32
4.5.4	Step2-ST_Customer_level_n+1 project flow	32
4.5.5	Running the end user application	33
4.5.6	PCROP protection in debug mode	33
5	Conclusion	36
6	Revision history	37

List of tables

Table 1.	Reference documents.	1
Table 2.	Flash memory content access vs. RDP level	7
Table 3.	Summary of PCROP availability on STM32F4 Series	12
Table 4.	Preprogrammed STM32F429ZIT internal Flash memory map	29
Table 5.	Document revision history	37

List of figures

Figure 1.	Flash memory map with PCROP-ed sectors	9
Figure 2.	PCROP versus Write protection	10
Figure 3.	PCROP-ed code calling a function located outside the PCROP-ed region	12
Figure 4.	STM32F4 PCROP flow example	14
Figure 5.	Example of an ST Customer level n and level n+1	14
Figure 6.	FIR Lowpass Filter function block diagram	15
Figure 7.	Example of assembler code containing literal pools	16
Figure 8.	Accessing to FIR-Filter options	17
Figure 9.	Setting the Execute-Only code option	18
Figure 10.	Accessing to FIR-Filter options	19
Figure 11.	Setting option "No data reads in code memory"	19
Figure 12.	STM32F429ZIT internal Flash memory map	20
Figure 13.	Scatter file modification	21
Figure 14.	Enabling PCROP with STM32 STLink Utility	23
Figure 15.	Step1-ST_Customer_level_n project flow	24
Figure 16.	Generating symbol definition file with IAR	27
Figure 17.	Generating symbol definition file with Keil®	28
Figure 18.	Adding symbol definition file to Keil® project	30
Figure 19.	Setting symbol definition file type to Object file	31
Figure 20.	Adding symbol definition file to IAR project	31
Figure 21.	Step1-ST_Customer_level_n+1 project flow	32
Figure 22.	PCROP-ed IP-Code Assembly reading	34
Figure 23.	Filling PCROP-ed sector starting address	34
Figure 24.	PCROP-ed IP-Code Assembly reading	35

1 Read Out Protection (RDP)

The Read Out Protection is a global Flash memory read protection allowing the embedded firmware code to be protected against copy, reverse engineering, dumping using debug tools or other means of intrusive attack. This protection should be set by the user after the binary code is loaded to the embedded Flash memory.

1.1 Read protection Level 0

Level 0 is the default one, Flash memory is fully open and all memory operations are possible in all boot configurations (Debug features, Boot from RAM, from System memory bootloader or from Flash memory). In this mode there is no protection (it is mainly intended for development and debug).

1.2 Read protection Level 1

When the read protection level 1 is activated, no access (read, erase, and program) to Flash memory or backup SRAM can be performed via debug features such as Serial Wire or JTAG, even while booting from SRAM or system memory bootloader.

However, when booting from Flash memory, accesses to this memory and to backup SRAM from user code are allowed.

Any read request to the protected Flash memory generates a bus error.

Disabling RDP level 1 protection by re-programming RDP option byte to level 0 leads to a mass erase.

1.3 Read protection Level 2

When RDP level 2 is activated, all protections provided in Level1 are active and the chip is fully protected. The RDP option byte and all other option bytes are frozen and can no longer be modified. The JTAG, SWV (single-wire viewer), ETM, and boundary scan are disabled.

When booting from Flash memory, the memory content is accessible to user code. However, booting from SRAM or from system memory bootloader is no more possible.

This protection is irreversible (JTAG fuse), so it's impossible to go back to protection levels 1 or 0.

[Table 2](#) describes different accesses to Flash memory, Backup SRAM, Option bytes and One Time Programmable bytes (OTP) when booting from internal Flash memory, or in Debug or boot from SRAM or system memory bootloader.

Table 2. Flash memory content access vs. RDP level

Memory area	Protection level	Debug, boot from RAM or from system memory bootloader			Booting from Flash memory (User code)		
		Read	Write	Erase	Read	Write	Erase
Main memory and BKP	Level 1	No		No ⁽¹⁾	Yes		
	Level 2	No			Yes		
Option bytes	Level 1	Yes			Yes		
	Level 2	No			No		
OTP	Level 1	No	NA ⁽²⁾		Yes	NA ⁽²⁾	
	Level 2	No	NA ⁽²⁾		Yes	NA ⁽²⁾	

1. Erased only when RDP changes from Level 1 to Level 0. The OTP area remains unchanged.

2. NA = Not Available.

1.4 Internal Flash memory content update on RDP protected STM32F4xx microcontrollers

When RDP protection is activated (level 1 or level 2), internal Flash memory content cannot be updated through Debug or when booting from SRAM or system memory bootloader.

So an important requirement for the End product is the ability to upgrade embedded firmware in the internal flash with new firmware versions, adding new features and correcting potential issues.

This requirement can be resolved by implementing user-specific firmware to perform In-Application Programming (IAP) of the internal Flash memory by using a communication protocol such as USART for the reprogramming process.

For more details about IAP refer to the application note AN3965, available at www.st.com.

2 Write protection

The write protection is used to protect the content of specified sectors against code update or erase. This protection can be applied by sector.

Any write request generates a Write Protection Error. The WRPERR flag is set by hardware when an address to be erased/programmed belongs to a write-protected part of the Flash memory.

For example the mass erase of a Flash memory where at least one sector is write protected is not possible and the WRPERR flag is set.

Flash memory sectors can be either in Write protection or Read & Write (PCROP) protection mode, depending on the Selection of protection mode (SPRMOD option bit) of nWRP_i option bits. To activate the write protection (SPRMOD = 0) for each Flash memory sector *i*, one option bit (nWRP_i) is used. When the Write protection is set for sector *i* (option bit nWRP_i = 0), this sector cannot be erased or programmed.

Enabling or disabling write protection can be managed either by embedded user code or by using STM32 ST-Link Utility software and debug interfaces.

3 Proprietary Code Read Out Protection (PCROP)

3.1 PCROP protection overview

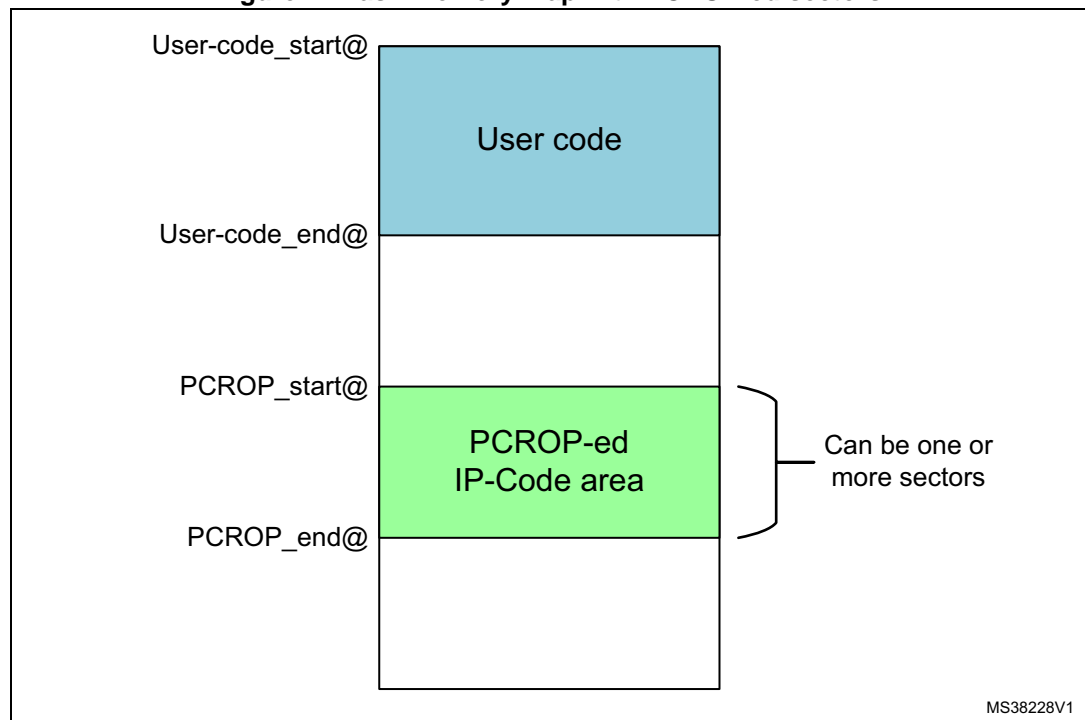
The PCROP is a read and write protection of an IP-Code in Flash memory applied by sector to protect proprietary code from possible modifications or read out by the end user code, debugger tools or RAM Trojan codes.

Any read or write request generates a Read or Write Protection Error:

- The WRPERR flag is set by hardware when an address to be erased/programmed belongs the PCROP-ed part of the Flash memory.
- The RDERR flag is set when a read access through the D-bus is performed to a PCROP-ed sector.
- A Flash memory operation error interrupt is generated and OPERR flag is set in FLASH_SR register when a read or write request from PCROP-ed sectors is detected and cannot be run. This is valid only when ERRIE bit is set in the FLASH_CR register.

The protected IP-Code can be easily called by the end user application and still be protected against direct access to the IP-Code itself. PCROP does not prevent protected codes from being executed.

Figure 1. Flash memory map with PCROP-ed sectors



3.2 How to enable the PCROP protection

To activate the PCROP, the SPRMOD option bit must be activated, this changes the function of the nWRP option bits. The PCROP sector is selected by using the same option bytes as the write protection, but with SPRMOD option bit active. Each sector can be

independently PCROP-ed, and protecting additional sectors is possible (when RDP is set to level 0 or 1) without full chip erase unlike disabling PCROP protection.

Some precautions should be taken when activating the PCROP feature. The active values of nWRP_i bits are inverted when PCROP mode is active, so if SPRMOD = 1 and nWRP_i = 1, then the user Sector i is PCROP-ed.

To correctly activate PCROP on Sector i, the sequence detailed below must be followed:

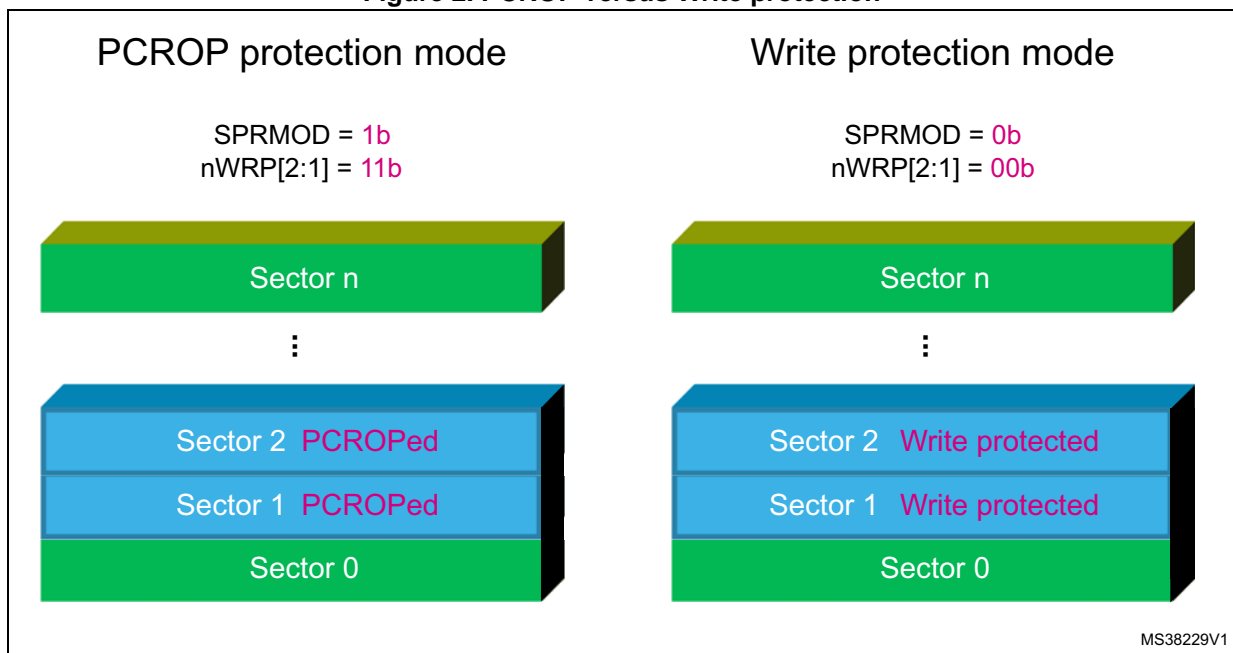
1. Clear all nWRP bits except those for already PCROP-ed sectors (for STM32F4 products with two Flash memory banks such as STM32F429ZIT, bank2 nWRP bits must be cleared too);
2. Set nWRP_i bit for Sector i to be PCROP-ed;
3. Enable PCROP Protection mode by setting SPRMOD bit

For more details on PCROP enabling implementation, refer to the *PCROP_Enable()* function described in the provided FW package (*Step1-ST_Customer_level_n project main.c* file).

It's important to note that it is not possible to have at the same time one write protected sector and another PCROP-ed sector. So user can have either write protection mode and sectors could be write protected, or PCROP mode and sectors could be PCROP-ed.

Figure 2 shows the difference between PCROP and write protection.

Figure 2. PCROP versus Write protection



3.3 How to disable the PCROP protection

Depending on the RDP level, PCROP can be disabled if RDP level is 1 or 0 but when RDP is set to level 2 PCROP can no more be disabled. When the RDP is set to Level 2, all the option bytes are frozen and cannot be modified. As a result, PCROP-ed sectors can never be erased or modified, so the protection becomes permanent.

The only way to disable PCROP on a protected sector is by changing RDP level from 1 to 0 and at the same time clearing the SPRMOD bit.

During application development user may need to disable PCROP or global RDP protection without spending time in developing and disabling protection functions. STM32 ST-LINK Utility tool can be a very simple way for disabling or enabling protection using debug interfaces as JTAG or SWD without the need for developing dedicated functions.

For more details on how to use STM32 ST-LINK Utility software user should refer to user manual UM0892, available at www.st.com.

Note: Disabling PCROP or/and Global RDP protection leads to a full chip erase.

3.4 Placing and executing PCROP-ed IP-Code

As previously mentioned, PCROP does not prevent protected IP-Code from being executed and its functions can be easily called by user code.

PCROP-ed sectors are protected against D-Code bus read accesses, so it's important to mention here that only code execution is allowed (Instruction fetch through I-Code bus) while data reading is forbidden. Therefore the protected IP-Code will be unable to access the associated data values stored in the same area (such as literal pools, branch tables or constants which are fetched from Flash memory through the D-Code bus during the execution).

PCROP-ed code must be an execute-only code and must not contain any data.

User must configure the compiler to generate an execute-only IP-Code avoiding any data read, the necessary compiler configurations are detailed in [Section 4](#).

3.4.1 Don't PCROP vector table sector

The Interrupt vector table contains entry point addresses of each interrupt handler, that are read by CPU through D-code bus. In general Interrupt vector table is located in the first sector at the first address 0x08000000 (except in some cases where it's relocated in other regions, such as SRAM).

When placing code to be protected in the Flash memory, the following rules must be respected:

- the first Flash memory sector where vector table is located must not be PCROP-ed
- the code to be PCROP-ed must not be placed in the first sector.

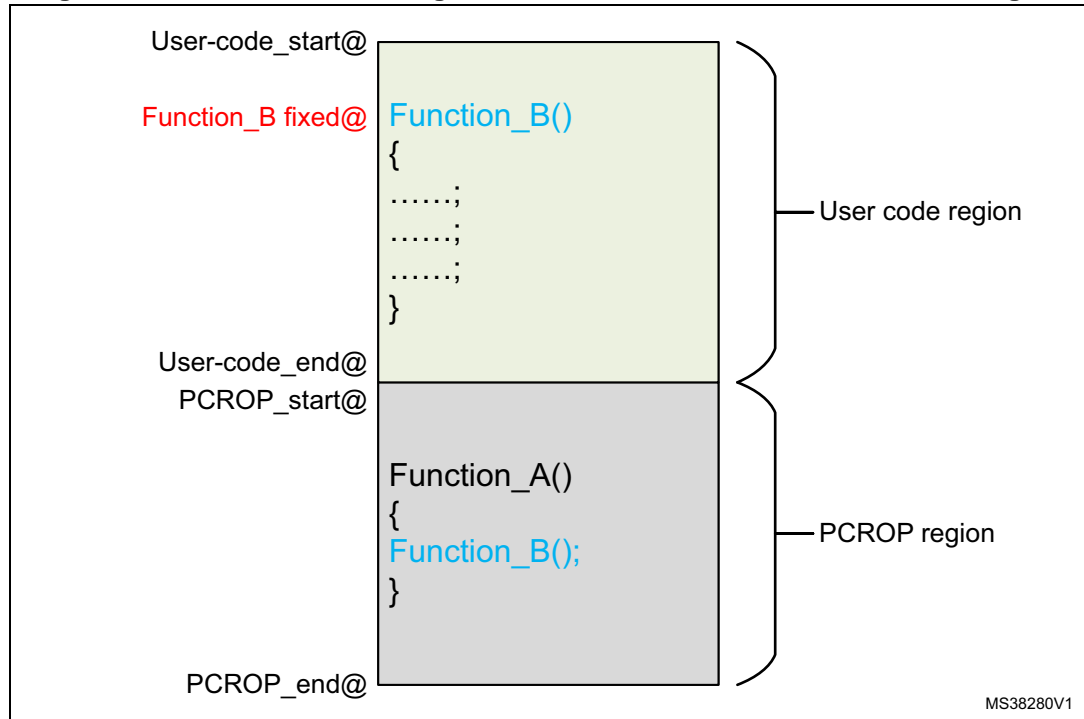
3.4.2 PCROP-ed IP-Code dependency

Protected IP-Code can call functions from libraries located in user code region and outside of PCROP-ed area. In this case the IP-Code contains the related functions addresses allowing PC (Program Counter) to jump to these functions when executing IP-Code. These addresses are unchangeable once the IP-Code is PCROP-ed. Consequently each called function must be located (outside of PCROP-ed region) at its corresponding fix address written in the PCROP-ed IP-Code else, PC jumps to an invalid address and IP-Code will not work correctly.

To be fully independent, protected IP-Code has to be placed together with all its related functions.

Figure 3 shows an example where PCROP-ed Function_A() is calling a Function_B() located at a fixed address outside the PCROP-ed region.

Figure 3. PCROP-ed code calling a function located outside the PCROP-ed region



3.5 PCROP feature availability on STM32F4 Series

Table 3 provides a summary of PCROP protection availability, Flash memory sizes and number of sectors on STM32F4 microcontrollers.

Table 3. Summary of PCROP availability on STM32F4 Series

Product lines	Flash memory (bytes)	Number of sectors	PCROP availability
STM32F401	Up to 512 K	Up to 8	Yes
STM32F405/415	Up to 1 M	Up to 12	No
STM32F407/417	Up to 1 M	Up to 12	No
STM32F411	Up to 512 K	Up to 8	Yes
STM32F427/437	Up to 2 M	Up to 24	Yes
STM32F429/439	Up to 2 M	Up to 24	Yes
STM32F446	Up to 512 K	Up to 8	Yes
STM32F469/479	Up to 2 M	Up to 24	Yes

4 PCROP example

The firmware example provided with this application note illustrates a use case of PCROP protection. All steps required to develop this firmware are detailed in this section.

4.1 Example requirements

4.1.1 Hardware requirements

The hardware required to run this example is the following:

- an STM32F429I-DISCOVERY board with embedded STM32F429ZIT MCU
- a mini-USB cable to power the board and to connect the discovery embedded STLINK for debugging and programming.

4.1.2 Software requirements

The following software tools are required:

- IAR Embedded Workbench® or Keil® µvision IDE
- STM32 STLink Utility is required mainly for enabling or disabling PCROP protection.

4.2 Example overview

This example describes a use case where an ST Customer Level n provides preprogrammed STM32F429ZIT MCUs with a critical IP-Code to an ST Customer level n+1.

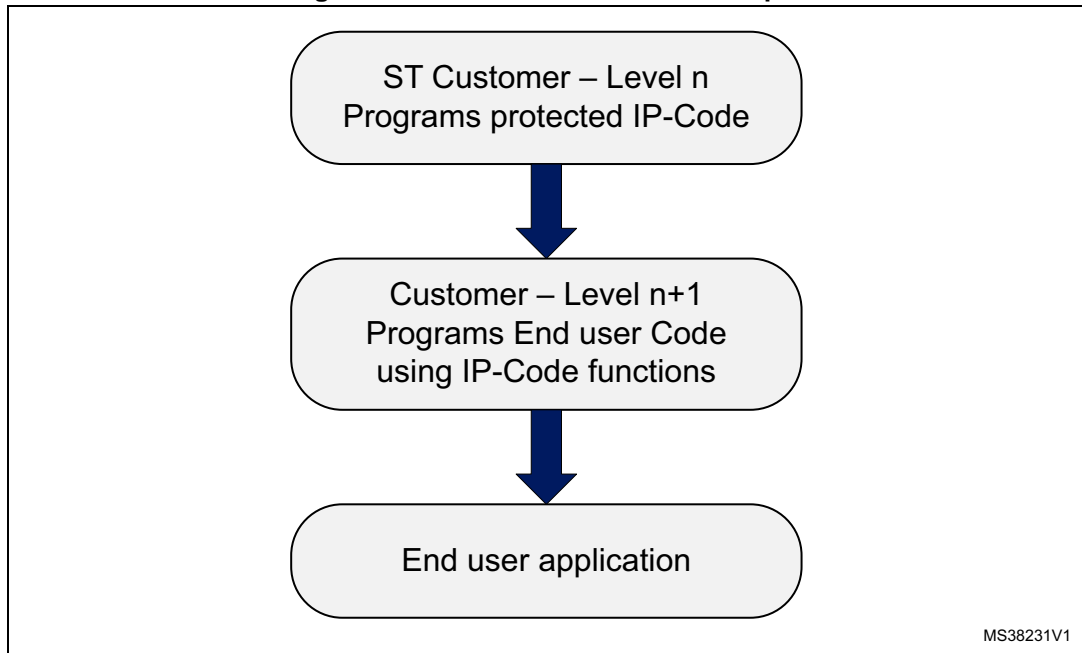
The IP-Code has to be protected by activating PCROP allowing ST Customer level n+1 to use its functions (without the ability to read or modify it) to program the end user application.

ST Customer level n should then provide, together with a preloaded STM32 MCU, the following inputs:

- Flash memory map defining the exact PCROP-ed IP-Code location and the available sectors for programming
- header file that has to be included in ST Customer level n+1 project containing IP-Code functions definition to be called in End Used Code
- symbol definition file containing PCROP-ed IP-Code function symbols.

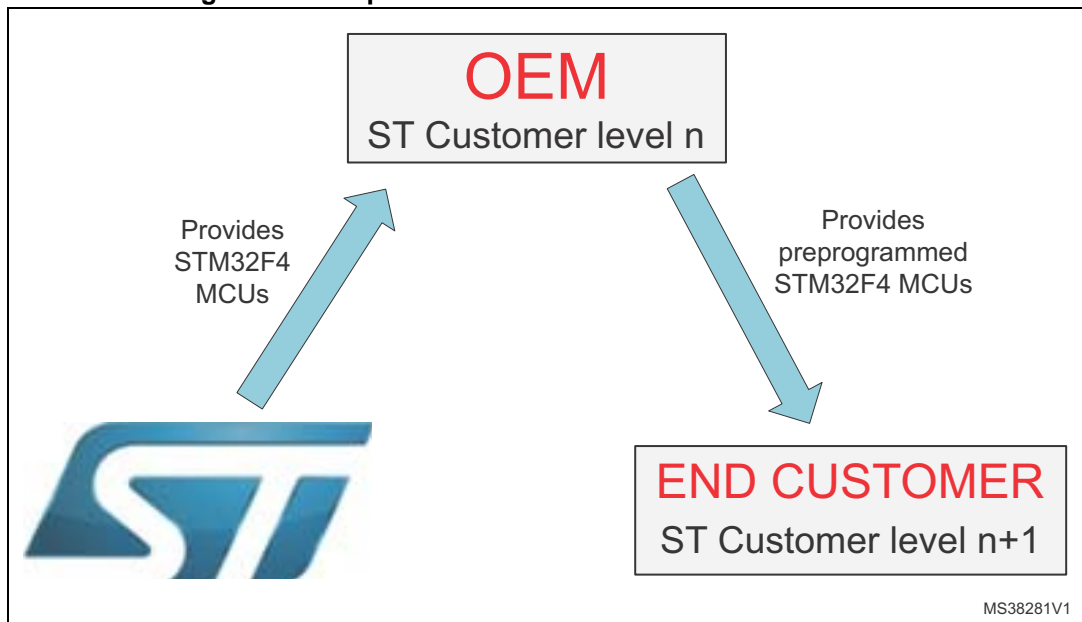
The described use case is schematically shown in [Figure 4](#).

Figure 4. STM32F4 PCROP flow example



An OEM (Original Equipment Manufacturer) can be the ST Customer level n using STM32F4 microcontrollers. Then the OEM provides preprogrammed MCUs to the ST Customer level n+1 that could be the END CUSTOMER who makes the end user product, as illustrated in [Figure 5](#).

Figure 5. Example of an ST Customer level n and level n+1



4.2.1 Software settings

This application note is provided with two projects:

- Project 1: *STEP1-ST_Customer_level_n*
 This project shows an example of how an ST Customer level n can place, protect and execute its IP-Code and how to generate IP-Code related files as header and symbol definition files to be provided to ST customer level n+1.
 This project includes two different project configurations:
 - PCROP-IP-Code-XO: in this configuration the compiler is configured to generate an execute-only IP-Code avoiding any data read from it.
 - PCROP-IP-Code: in this configuration the IP-Code is compiled without avoiding data (literal pools) generation. This configuration is dedicated to testing purposes in order to show that PCROP-ed IP-Code must be an execute-only code.
- Project 2: *STEP2-ST_Customer_level_n+1*
 This project shows an example of how an ST Customer level n+1 having a preprogrammed STM32F429ZIT with a PCROP-ed IP-Code, can create its own end user application using these protected IP-Code functions.

4.3 PCROP-ed IP-Code: FIR low pass filter

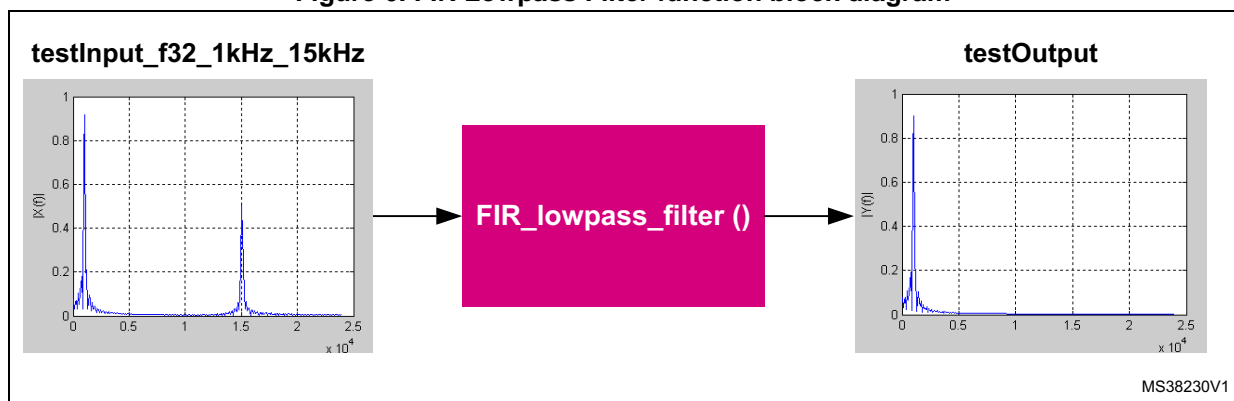
As example of IP-Code to be protected a FIR low pass filter algorithm from CMSIS-DSP will be described, focusing on how to protect and call this IP-Code function, without providing details on the functions themselves.

The FIR low pass filter removes high frequency signal components from the input.

The input signal is a sum of two sine waves having frequencies of 1 KHz and 15 KHz. The low pass filter (with its pre-configured cutoff frequency set at 6 KHz) eliminates the 15 KHz signal, leaving the 1 KHz sine wave at the output.

[Figure 6](#) shows the FIR low pass filter block diagram.

Figure 6. FIR Lowpass Filter function block diagram



Used CMSIS DSP Software Library Functions:

- arm_fir_init_f32()*: initialization function to configure the filter, described in *arm_fir_init_f32.c* file;
- arm_fir_f32()*: the elementary function representing the FIR Filter, described in *arm_fir_f32.c* file.

The following function was created using the CMSIS DSP functions described above:

- *FIR_lowpass_filter()*: the global function representing the FIR Filter, described in *fir_filter.c* file.

The FPU and DSP embedded in STM32F4 microcontrollers are used for signal processing and floating point calculation to output the correct signal.

For more details on FIR functions users should refer to CMSIS documentation in “Drivers/CMSIS/Documentation/DSP” directory included in the associated software package.

4.4 STEP1: ST Customer level n

At this stage the ST Customer level n will:

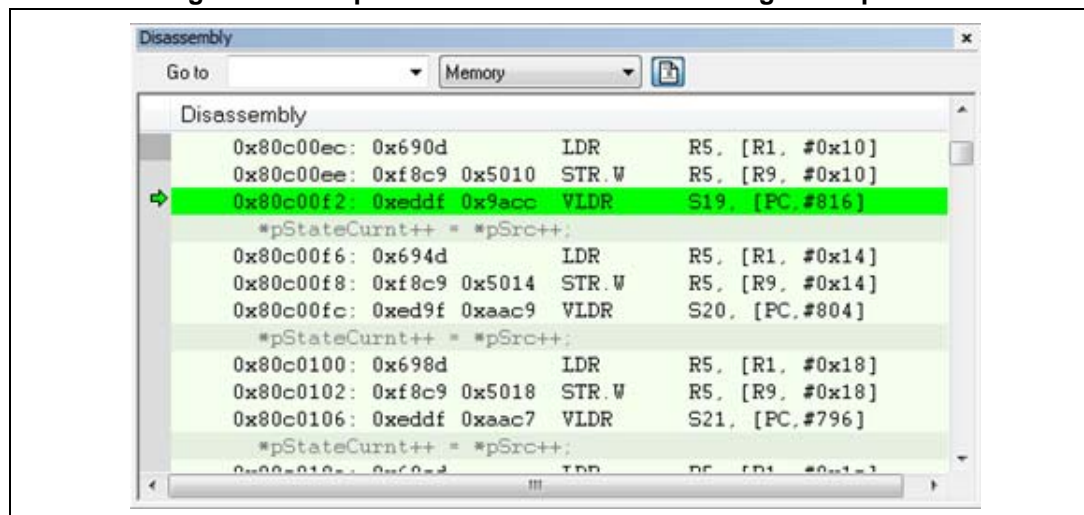
- generate an execute-only IP-Code;
- place the IP-Code in Sector 2 of internal Flash memory;
- protect the IP-Code placed in Sector 2;
- execute the IP-Code by calling its functions in main code;
- create header file and generate symbol definition file to be used in STEP2-ST_Customer_level_n+1 project.

4.4.1 Generating an execute-only IP-Code

Each toolchain has its own options to prevent compiler from generating literal pools and branch tables. For instance Keil® has the **Execute-only Code** option while IAR has the **No data reads in code memory** option.

Figure 7 shows an assembler code containing literal pools, where the instruction has the form of *VLDR <variable>, [PC + <offset>]*.

Figure 7. Example of assembler code containing literal pools



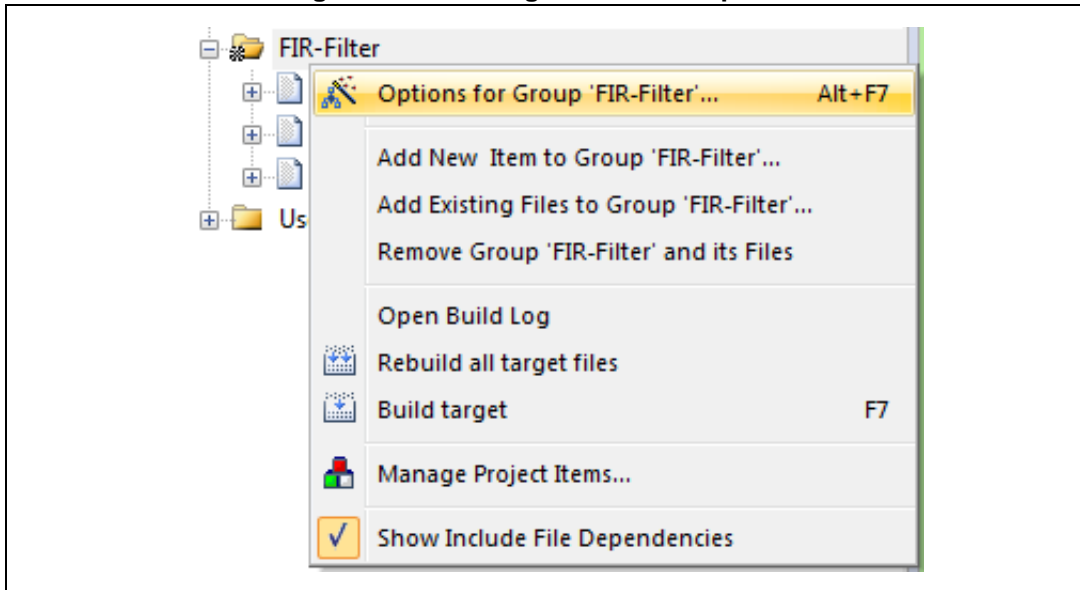
Keil®: Using *Execute-Only* command

For Keil® the command `--execute_only` must be used to generate a code without literal pools and branch tables preventing the compiler from generating any data accesses to code sections.

The `--execute_only` command must be used for any PCROP-ed code.

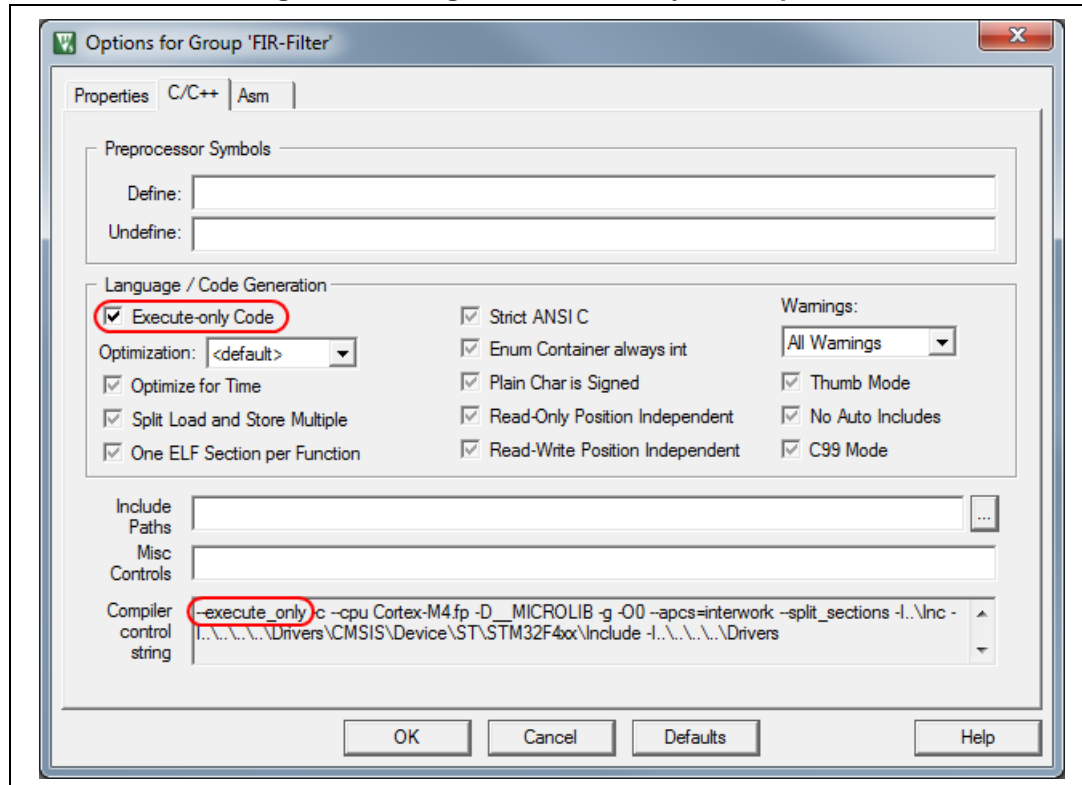
To set this command right click on the group file or the IP-Code and choose **Options for Group 'FIR-Filter'**, as indicated in [Figure 8](#).

Figure 8. Accessing to FIR-Filter options



In the following window (see [Figure 9](#)) check the option **Execute-only Code**, then the `-execute_only` command is added in compiler control string field.

Figure 9. Setting the Execute-Only code option



Then in the scatter file replace the option `read only +RO` by `execute-only +XO`.

Updating Keil® scatter file

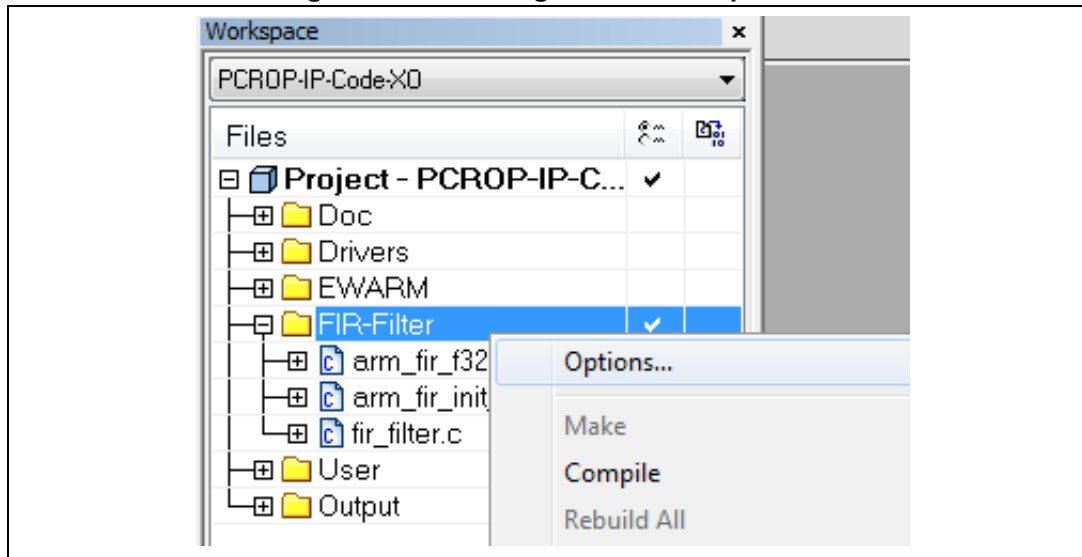
The scatter file has to be updated in order to set IP-Code files as execute-only code as shown below

```
LR_PCROP 0x08008000 0x00004000 {
  ER_PCROP 0x08008000 0x00004000 { ; load address = execution address
    arm_fir_f32.o (+XO)
    arm_fir_init_f32.o (+XO)
    FIR_Filter.o (+XO)
  }
}
```

IAR: No data reads in code memory

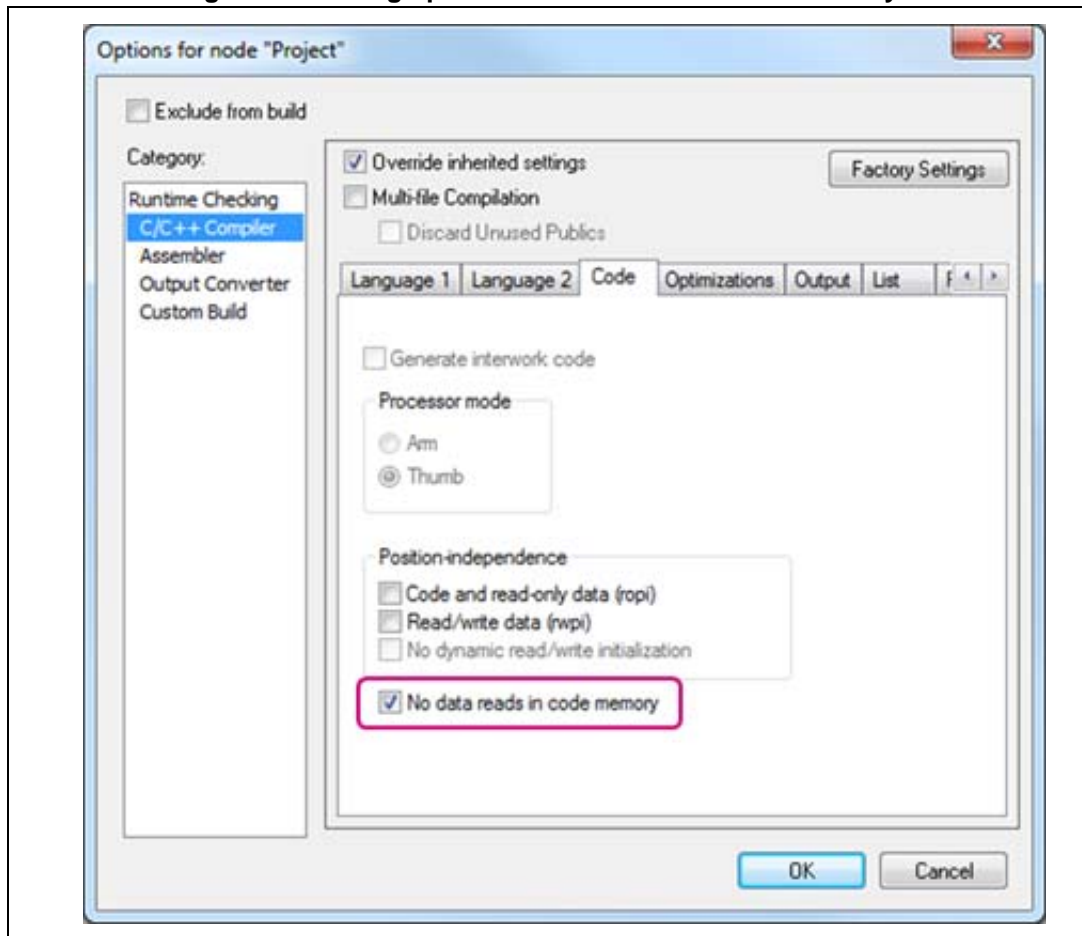
For IAR the option “No data reads in code memory” must be used to prevent the compiler from generating any data accesses to code sections. To activate this option, right click on the file group ‘FIR-Filter’ containing IP-Code source files then (see [Figure 10](#)) select *Options*.

Figure 10. Accessing to FIR-Filter options



Then, in the following window check the option "No data reads in code memory", as indicated in [Figure 11](#).

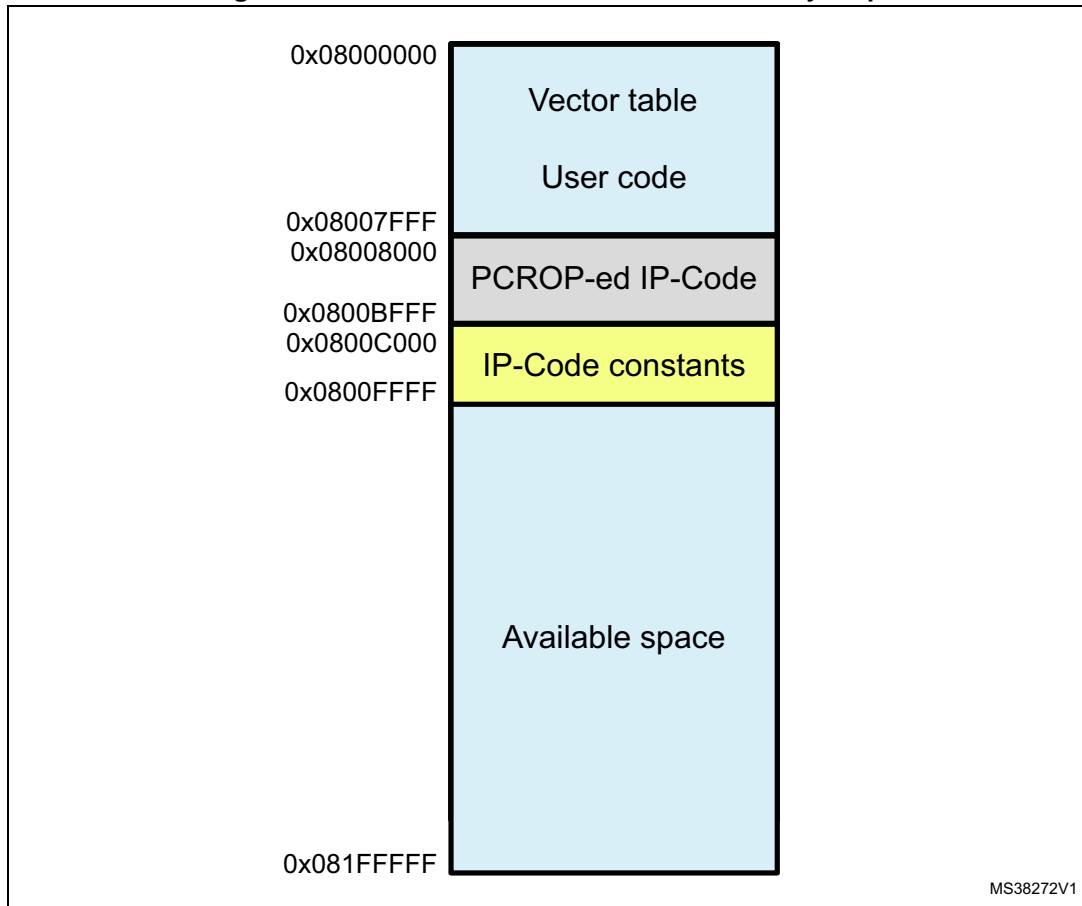
Figure 11. Setting option "No data reads in code memory"



4.4.2 Placing the IP-Code

As previously mentioned, PCROP-ed code must not be in the first sector where interrupt vector table is located. That's why we will place the IP-Code in Sector 2 BANK1 of the internal Flash memory, while the user code and the vector table are placed starting from the first sector, as indicated in [Figure 12](#).

Figure 12. STM32F429ZIT internal Flash memory map



Keil® scatter file

To place IP-Code in Sector 2 follow the sequence below:

Go to *Project* → *Options for Target*, select the Linker tab then uncheck *Use Memory Layout* from *Target Dialog* as shown in [Figure 13](#), then click on *Edit* button to modify the PCROP-w-XO.sct scatter file.

The scatter file is then opened. IP-Code object files should be placed in a new dedicated load region named LR_PCROP as shown below:

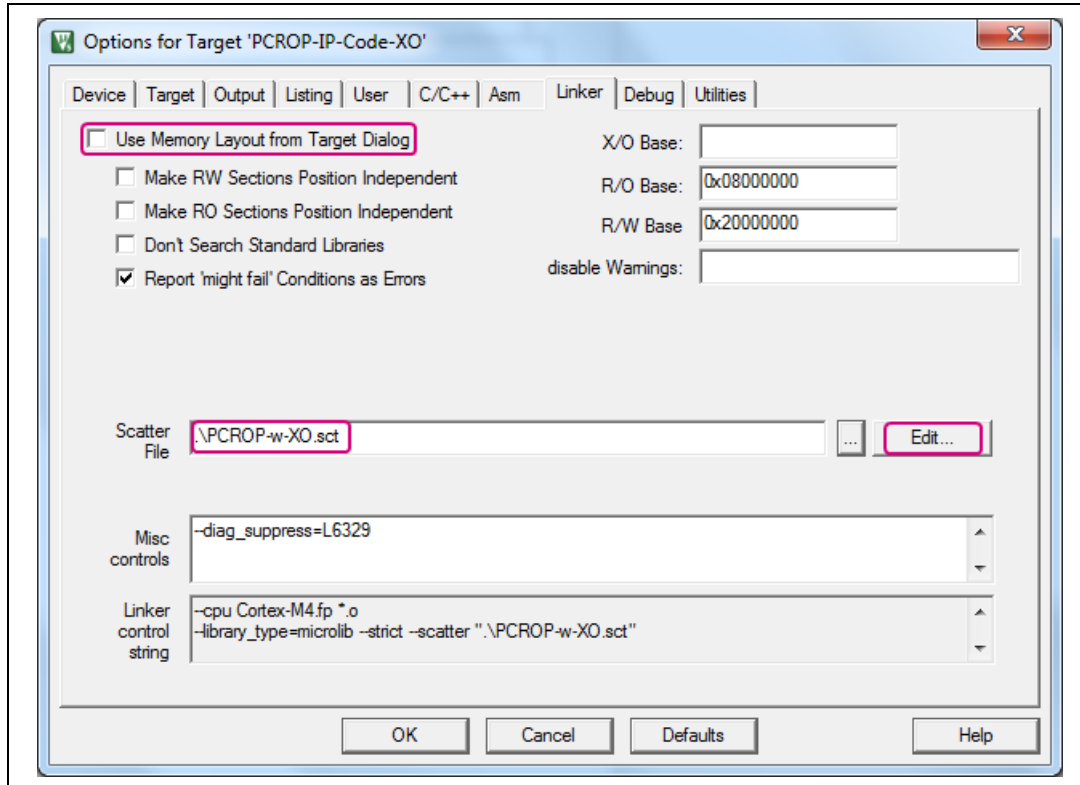
```
LR_PCROP 0x08008000 0x00004000 { ; IP Code dedicated load region
ER_PCROP 0x08008000 0x00004000 {
    arm_fir_f32.o (+XO)
    arm_fir_init_f32.o (+XO)
    FIR_Filter.o (+XO)
```

```

    }
}

```

Figure 13. Scatter file modification



IAR ICF file

To place IP-Code in Sector 2 with IAR, open ICF file and add a new load region as shown below:

```

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__];

place in PCROP_region { ro object arm_fir_f32.o,
                        ro object arm_fir_init_f32.o,
                        ro object FIR_Filter.o};

```

4.4.3 Placing constants outside of PCROP-ed sector

All IP-Code constants (e.g. integer, float or strings) have to be placed outside of PCROP-ed region as they cannot be accessed by the D-Code bus.

Linker file can be used to place constants in a specified memory region as shown in the following example.

Example: IAR ICF file:

```
define symbol __ICFEDIT_region_CONST_start__ = 0x0800C000;
define symbol __ICFEDIT_region_CONST_end__   = 0x0800FFFF;
define region CONST_region = mem:[from __ICFEDIT_region_CONST_start__
to __ICFEDIT_region_CONST_end__];
place in CONST_region {section .rodata};
```

Another way is placing constants at a fixed address directly in user code when defining the const using the *@ operator* for IAR and *__attribute__((at(address)))* for Keil®. This approach has been used in this example and constants definition were done in *fir_filter.c* file which is similar to the following example.

Placing constants at fixed address with Keil®

```
const int k __attribute__((at(0x0800C000))) = 500; /* place const k in flash
at 0x0800C000 */
```

Placing constants at fixed address with IAR

```
const int k @ 0x0800C000 = 500; /* place const k in flash at 0x0800C000 */
```

Caution: When placing IP-Code constants, user should take in account that these constants can be deleted or modified by ST Customer level $n + 1$, consequently the IP-Code functions will be useless. It's then recommended to place these constants in a dedicated sector where no user code will be programmed. Then this memory region should be highlighted in the Flash memory map provided to ST Customer level $n+1$.

4.4.4 Protecting the IP-Code

Activating PCROP using *PCROP_Enable()* function

Protecting the IP-Code is done by activating PCROP on Sector 2 using *PCROP_Enable()* function defined in the *Step1-ST_Customer_level_n project main.c* file.

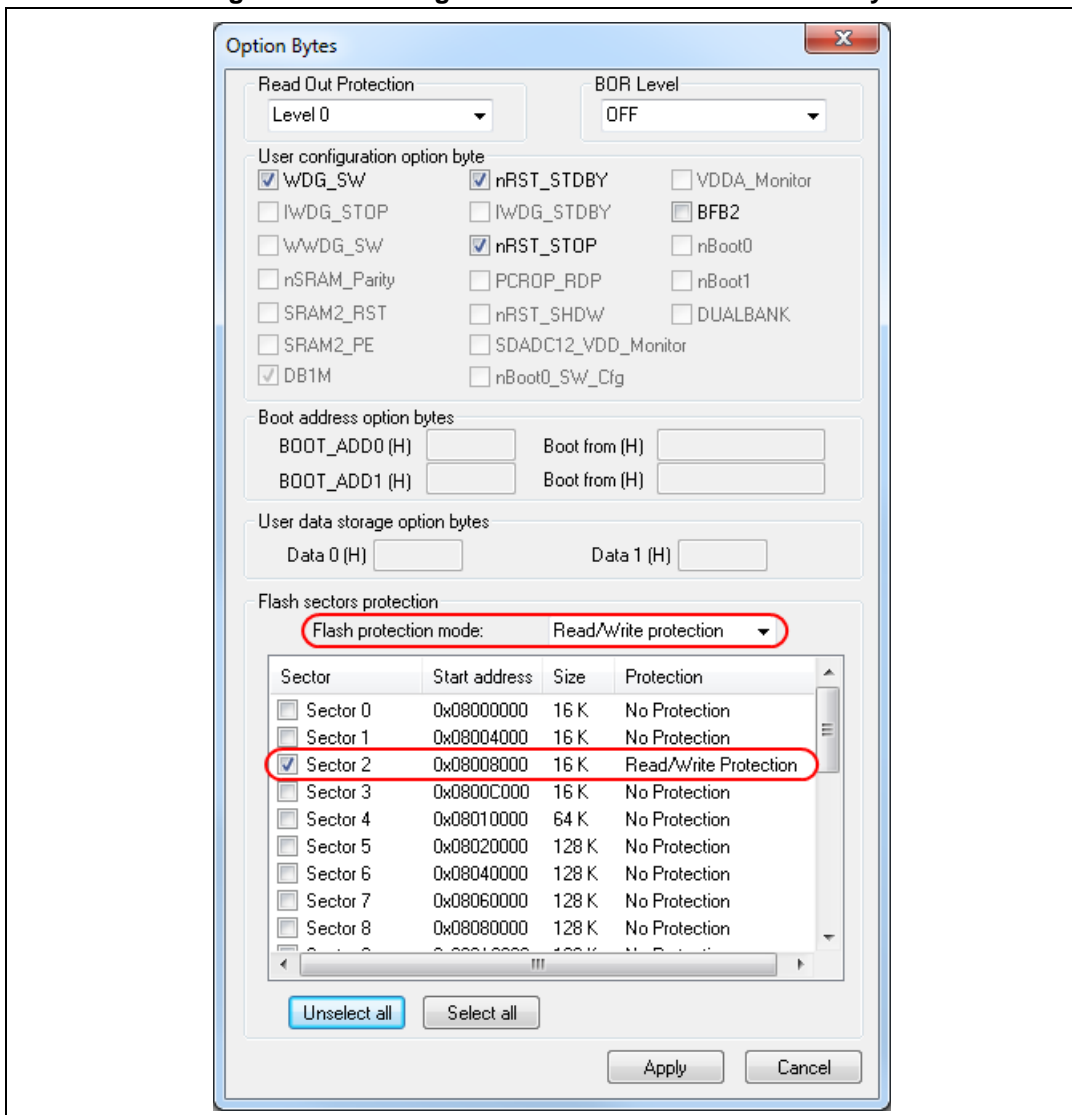
Once this function is executed at least one time, Sector 2 becomes Read/Write protected and any IP-Code modification requires disabling PCROP then full Flash memory content will be erased.

Activating PCROP using STM32 STLink Utility

To activate PCROP on Sector 2 using STM32 STLink Utility user must follow the sequence shown below:

- Power on the board, then In the STLink Utility interface go to *Target* → *Option bytes*.
- In the following window set Flash memory protection mode to Read/Write protection and enable protection on Sector 2, as shown in [Figure 14](#), then click on *Apply* button.

Figure 14. Enabling PCROP with STM32 STLink Utility

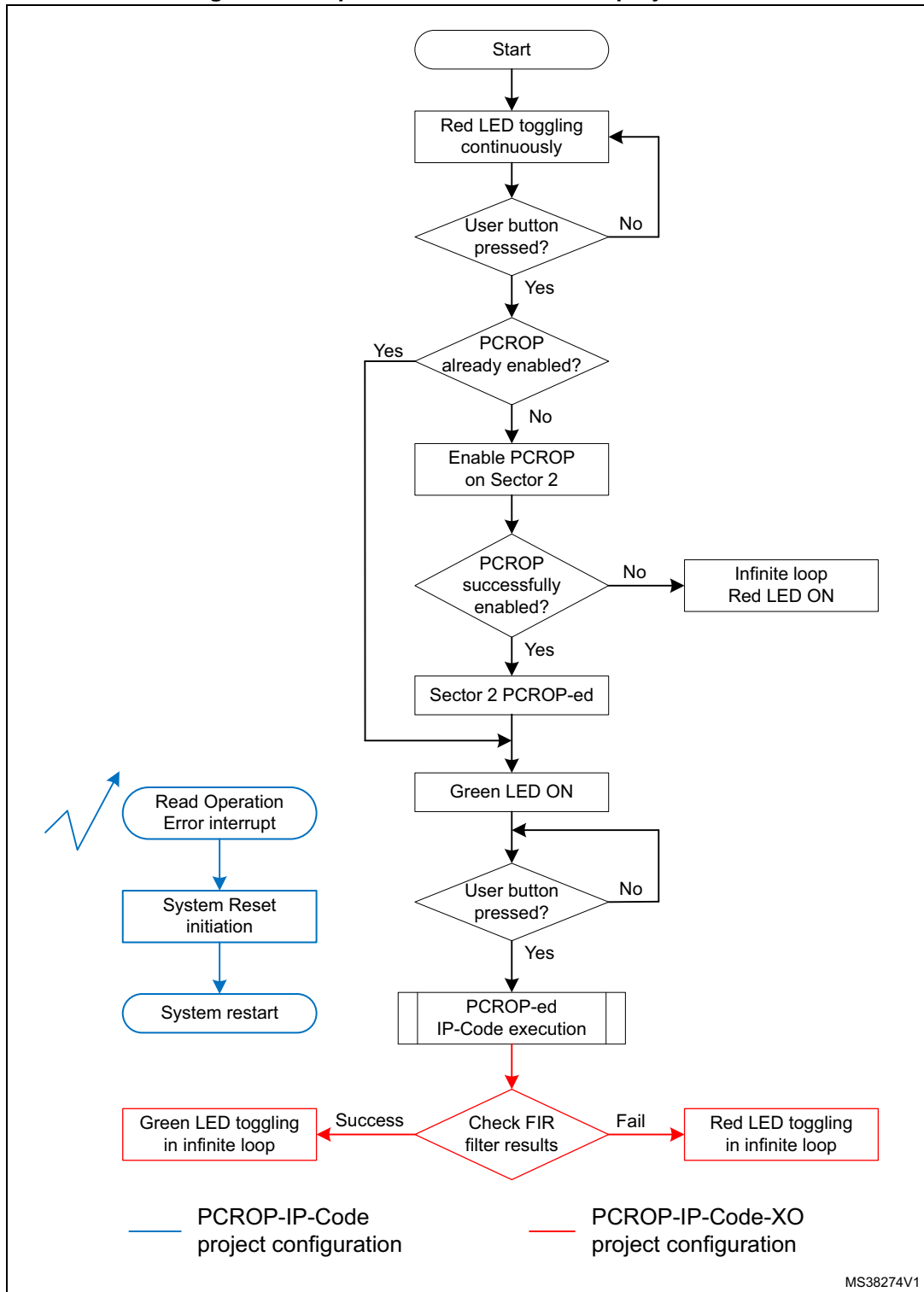


4.4.5 Step1-ST_Customer_level_n project flow

Figure 15 illustrates the Step1-ST_Customer_level_n project flow with both project configurations:

- PCROP-IP-Code-XO (in red): PCROP-ed IP-Code didn't contain any literal pool, then FIR-Filter algorithm run successfully noting that a Read Operation Error Interrupt is generated and system reset is initiated if any read/write request occurs from/to PCROP-ed region.
- PCROP-IP-Code (in blue): IP-Code containing literal pools, when starting PCROP-ed IP-Code execution, a Read Operation Error Interrupt is generated, then system reset initiated and system restarted.

Figure 15. Step1-ST_Customer_level_n project flow



4.4.6 Executing PCROP-ed IP-Code

Once IP-Code has been programmed on Sector 2 and protected, it has to be tested by calling its functions in user code.

In this section we will show how to execute the PCROP-ed IP-Code in the *Step1-ST_Customer_level_n* project with both configurations, noting that the PCROP-IP-Code configuration is only for testing purposes and must not be used for STEP2. The PCROP-IP-Code-XO is the right configuration where the compiler is set to generate an execute-only IP-Code. This is the configuration to use to run the *Step2-ST_Customer_level_n+1* project.

PCROP-IP-Code-XO (must be used before STEP2)

The compiler is configured to generate an execute-only IP-Code avoiding any data read from it (avoiding literal pools).

1. Open project located in *Step1-ST_Customer_level_n* directory and choose your preferred toolchain
2. Select PCROP-IP-Code-XO configuration
3. Rebuild all files.
4. Run the example following the sequence below:
 - a) Power on the board and before loading the code, check if there is any PCROP-ed or write protected sector. If yes disable the protection using STM32 STLink Utility then load the code. Once program has been loaded, red LED should toggle continuously;
 - b) press the user button key to activate PCROP protection, once done green LED is ON and Sector 2 is PCROP-ed, else red Led is ON and PCROP activation failed;
 - c) press the user button key to execute the PCROP-ed IP-Code called in *main.c* file, the green LED should toggle continuously.

PCROP-IP-Code (for test only and must not be used for STEP2)

No special compiler option used, just for testing purposes to show that avoiding data in code (as literal pools and branch tables) is mandatory for PCROP-ed codes.

1. In the same project located in *Step1-ST_Customer_level_n* directory select *PCROP-IP-Code* configuration
2. Rebuild all files.
3. Run the example following the sequence below:
 - a) Power on the board and before loading the code, check if there is any PCROP-ed or write protected sector. If yes disable the protection using STM32 STLink Utility, then load the code; Once program has been loaded, red LED should toggle continuously;
 - b) press the user button key to activate PCROP protection, once done green LED is ON and Sector 2 is PCROP-ed, else red LED is ON and PCROP activation failed;
 - c) press the user button key to execute the PCROP-ed IP-Code called in *main.c* file, an Error Operation Interrupt is generated, system Reset is initiated and the red LED toggles continuously.

Interpretation

The low pass filter function computes the *testInput_f32_1kHz_15kHz* input signal and should output a 1 KHz sine wave. The output data *testOutput* is then compared to the reference *refOutput* already calculated with MATLAB, if it matches, the green LED toggles continuously, else the red LED toggles continuously.

For the PCROP-IP-Code configuration where PCROP-ed IP-Code contains literal pools: when executing IP-Code (FIR_lowpass_filter() function), literal pools could not be accessed through D-Code bus then the RDERR flag is set. OPERR flag is set as well and a read operation error interrupt is generated, then a system reset is initiated in *HAL_FLASH_OperationErrorCallback()* function and red LED toggles continuously.

However for the PCROP-IP-Code-XO configuration the IP-Code is executed correctly and green LED should toggle continuously.

Note: For more details, refer to the *readme.txt* inside the firmware package.

4.4.7 Creating header file and generating symbol definition file

The header file to be provided to ST Customer level n+1 contains the definitions of IP-Code functions to be used. In this example it's the *fir_filter.h* file included in the *main.c* file.

Generating symbol definition file with IAR

In the IDE, to export PCROP-ed IP-Code symbols, choose *Project*→*Options*→*Build Actions* and specify the following command line in the Post-build command line text field:

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

where *steering_file.txt* is created to be used as option to keep only IP-Code function symbols in the symbol file to be generated.

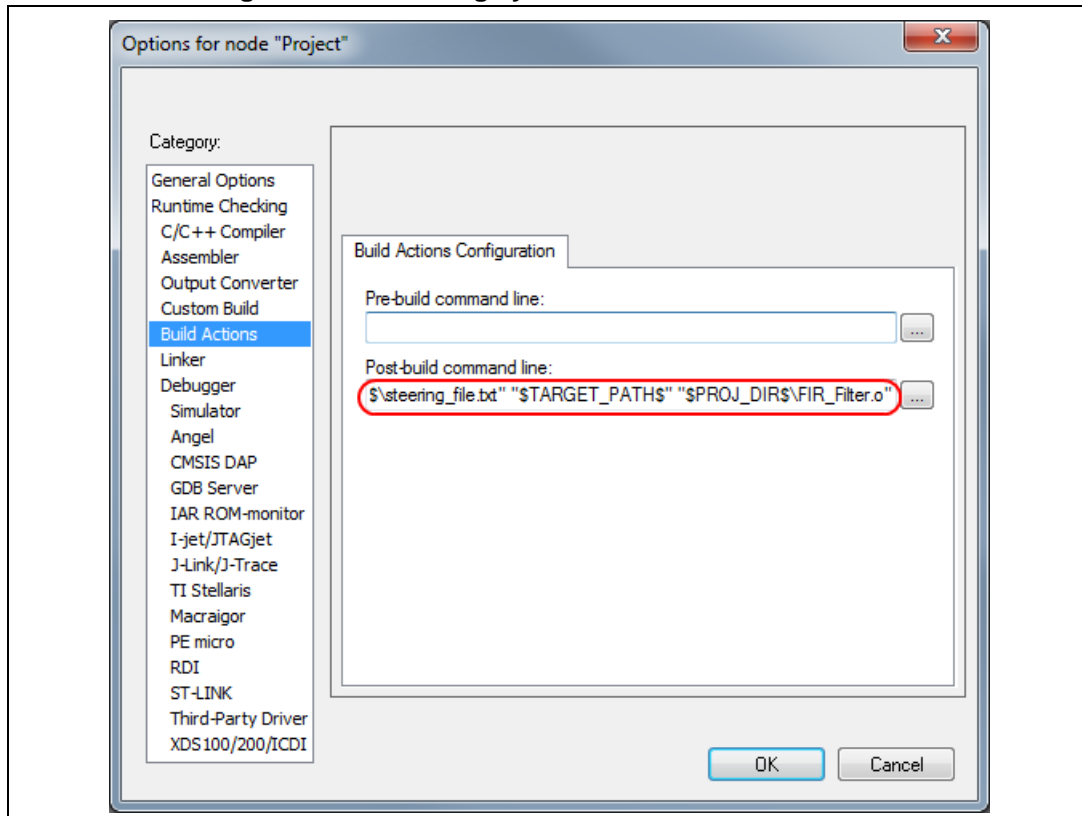
To keep only IP-Code functions in symbol file the *steering_file.txt* has to be created as shown below:

```
show arm_fir_f32  
show arm_fir_init_f32  
show FIR_lowpass_filter
```

where "show" command is used to keep the preferred functions in the symbol definition file to be generated.

The generated symbol definition file *fir_filter.o* will be used by adding it to STEP2-ST_Customer_level_n+1 IAR project.

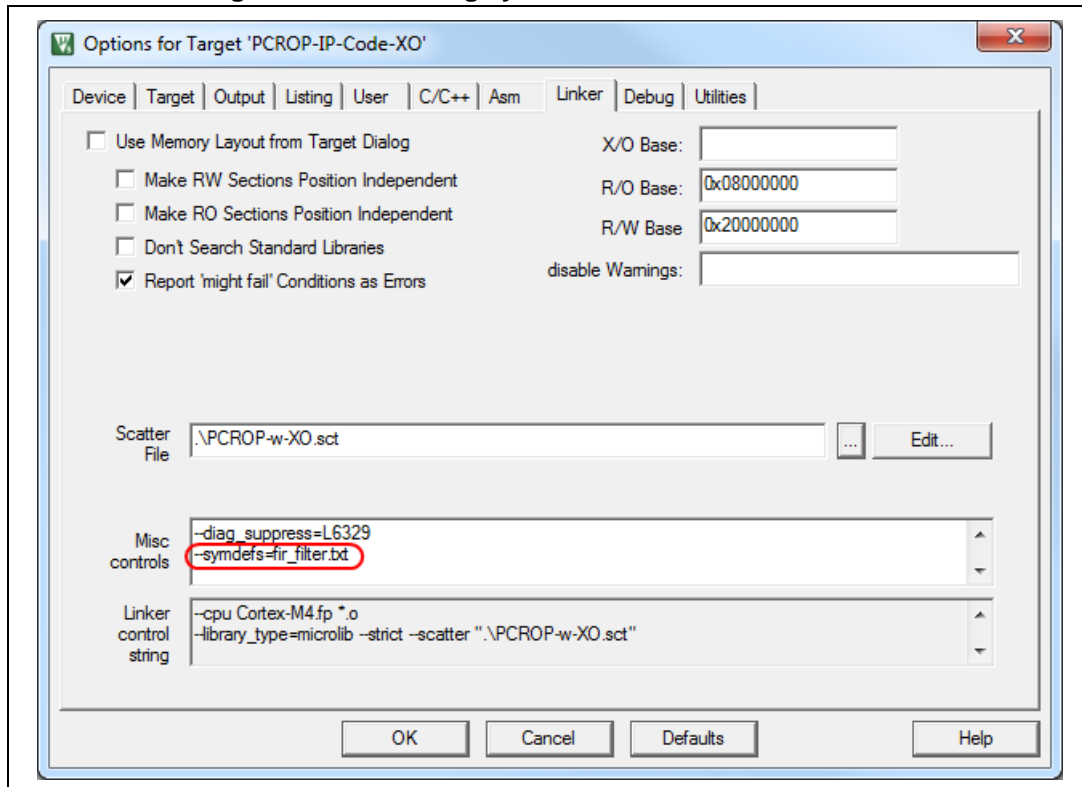
Figure 16. Generating symbol definition file with IAR



Generating symbol definition file with Keil®

To generate the symbol definition file with Keil® go to Project options in the Linker tab add the command `--symdefs=fir_filter.txt` and rebuild the project.

Figure 17. Generating symbol definition file with Keil®



The symbol definition file named *fir_filter.txt* is then created in Step1-ST_Customer_level_n\MDK-ARM\PCROP-IP-Code-XO directory.

The file contains all symbols of the project, so only those of the IP-Code functions to be called by the end user should be kept. All other ones should be removed, the resulting symbol definition file will then be

Symbol definition file *fir_filter.txt*:

```
#<SYMDEFS># ARM Linker, 5050106: Last Updated: Sat May 16 20:25:52 2015
0x08008001 T FIR_lowpass_filter
0x0800804d T arm_fir_f32
0x08008483 T arm_fir_init_f32
```

4.5 STEP2: ST Customer level n+1

ST Customer level n+1 have the preloaded STM32F429ZIT MCUs with the PCROP-ed IP-Code and the provided symbol definition file and the header file from ST Customer level n.

Then referring to the Flash memory map provided by the ST Customer level n, ST Customer level n+1 should:

- create end project;
- include header file and add symbol definition file provided by ST Customer level n to its project;
- call PCROP-ed IP-Code functions;
- execute and debug the end user application.

Caution: The ST Customer level n+1 must use exactly the same toolchain and compiler version used by ST Customer level n to develop and program the IP-Code, else ST Customer level n+1 could never use the IP-Code.

As in the provided example, user must use the same toolchain and compiler version for both projects STEP1-ST_Customer_level_n and STEP2-ST_Customer_level_n+1.

For example if the MDK-ARM toolchain V5.14 is used to run STEP1-ST_Customer_level_n project, then it has to be used to run STEP2-ST_Customer_level_n+1 project as well.

4.5.1 Creating an end user project

ST Customer level n+1 will place its code based on Flash memory map described in [Table 4](#).

Sector 2 (gray line) is unavailable as it is PCROP-ed while Sector 3 (light yellow line) shouldn't be used as it contains IP-Code constants. So user must take care when implementing linker file to avoid placing any code at these regions.

Table 4. Preprogrammed STM32F429ZIT internal Flash memory map

Sector number	Sector size	Start address	Content
Sector 0	16 KBytes	0x08000000	Available
Sector 1	16 KBytes	0x08004000	
Sector 2	16 KBytes	0x08008000	PCROP-ed IP-Code
Sector 3	16 KBytes	0x0800C000	IP-Code constants
Sector 4	16 KBytes	0x08008000	Available
...	
...	
Sector 11	128 KBytes	0x080E0000	
Sector 12	16 KBytes	0x08100000	
...	
...	
Sector 22	128 KBytes	0x081C0000	
Sector 23	128 KBytes	0x081E0000	

PCROP-ed IP-Code functions are then used to create end user application. The project located in Step2-ST_Customer_level_n+1 directory is an example where PCROP-ed FIR Filter functions are called in *main.c* file.

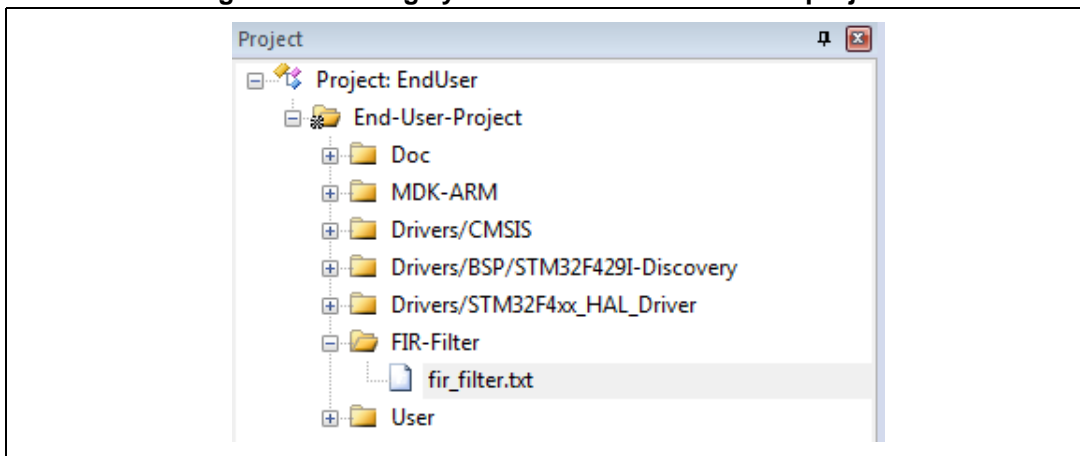
4.5.2 Including header file and adding symbol definition file

The symbol definition file provided from ST Customer level n should be added as a classic source file which is necessary to link PCROP-ed IP-Code in the Step2-ST_Customer_level_n+1 project.

Adding symbol definition file in Keil® project

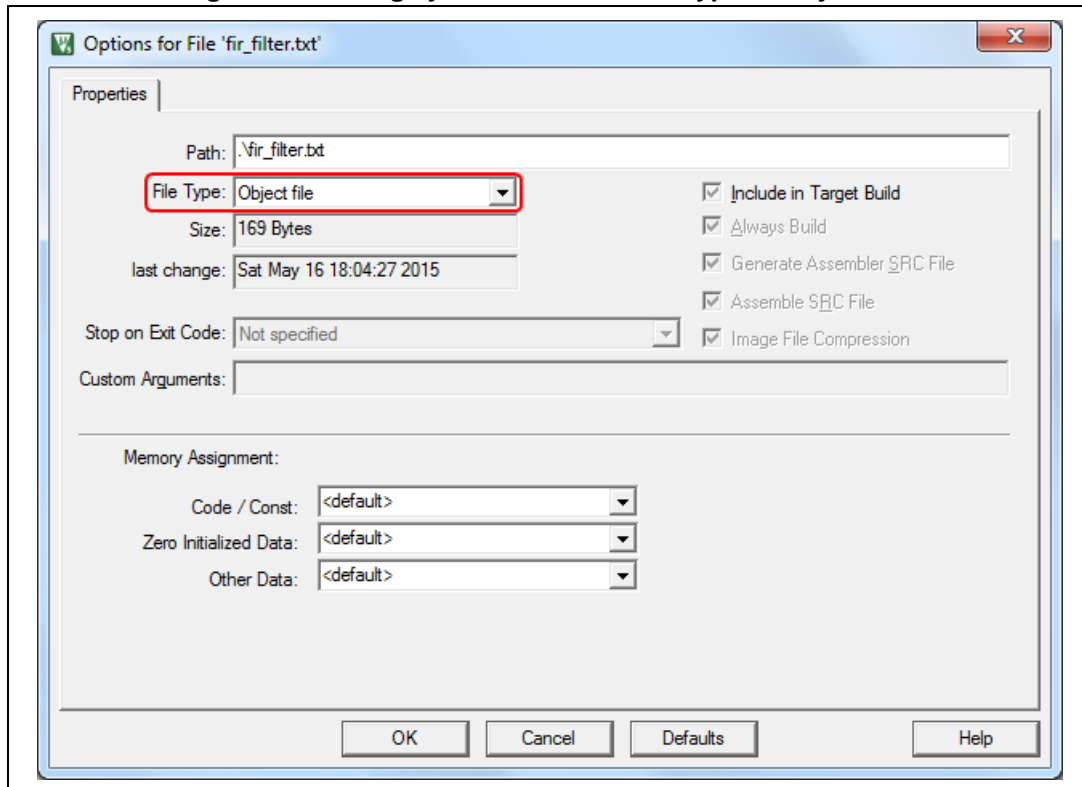
The symbol definition file provided from ST Customer level n described in [Section 4.4.7](#) and named *fir_filter.txt* in this example must be added to FIR-Filter group as described in [Figure 18](#).

Figure 18. Adding symbol definition file to Keil® project



The added *fir_filter.txt* file type has to be changed to Object file instead of text document file, as indicated in [Figure 19](#).

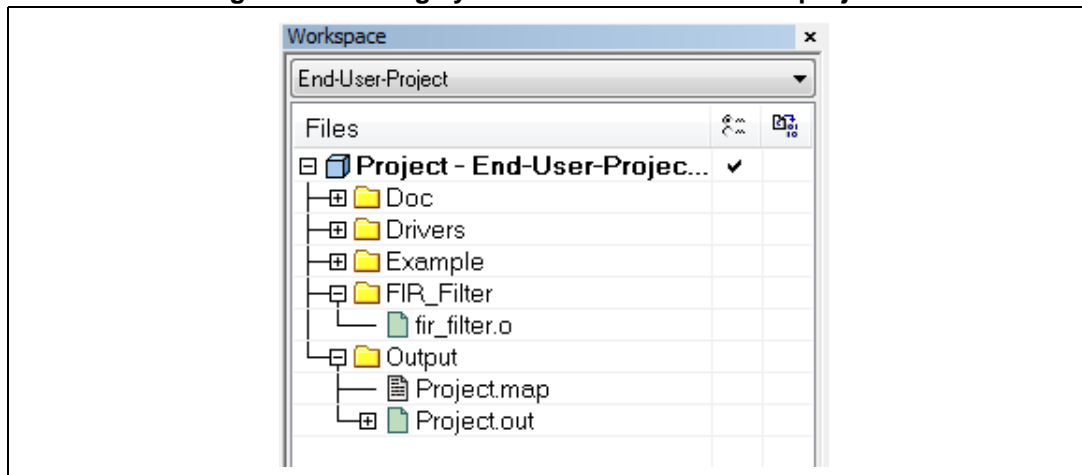
Figure 19. Setting symbol definition file type to Object file



Adding symbol definition file in IAR project

The *fir_filter.o* file must be added as an object file to the group *FIR_Filter* as shown in [Figure 20](#).

Figure 20. Adding symbol definition file to IAR project



Including header file

The header file *fir_filter.h* is included in *main.c* file then user can call FIR Filter functions.

4.5.3 Calling PCROP-ed IP-Code functions

Once *fir_filter.h* header file is included in *main.c* file and symbol definition file added to project, PCROP-ed IP-Code functions can be called.

The *FIR_lowpass_filter()* function (described in *fir_filter.c* file) is called in the main file as shown below:

```
FIR_lowpass_filter(inputF32, outputF32, TEST_LENGTH_SAMPLES)
```

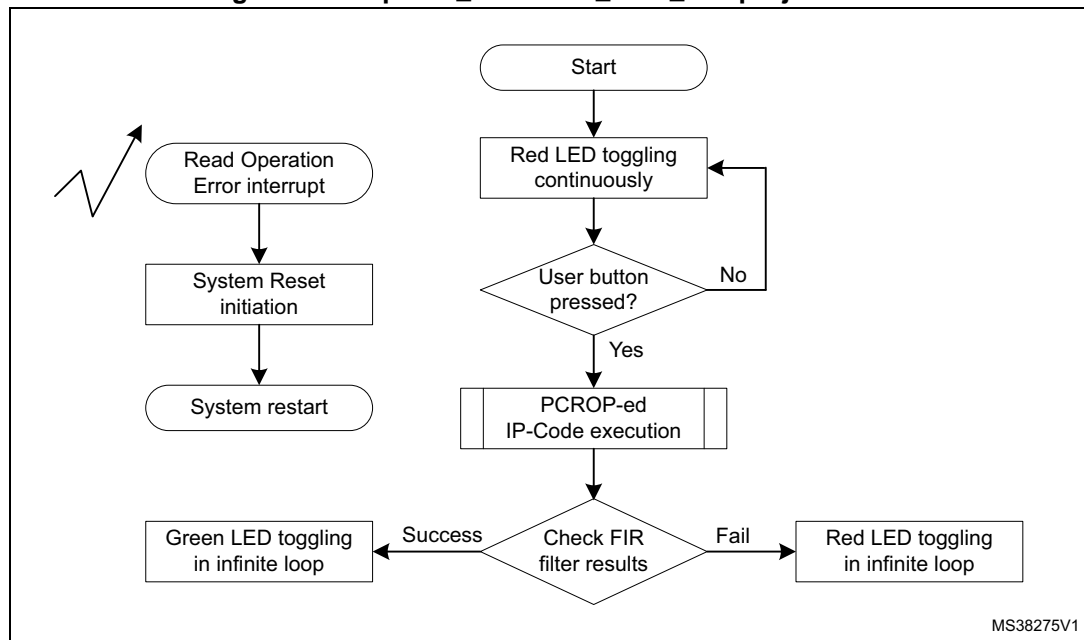
where:

- *inputF32*: is the pointer to the table containing input signal data;
- *outputF32*: is the pointer to the table to be filled with processed output signal data;
- *TEST_LENGTH_SAMPLES*: is the number of samples in the input to be processed.

4.5.4 Step2-ST_Customer_level_n+1 project flow

Figure 21 illustrates the Step2-ST_Customer_level_n+1 project flow. If any read/write operation request from/to PCROP-ed IP-Code occurs, a Read Operation Error Interrupt is generated, a system reset is initiated then system restarts and red LED toggles continuously.

Figure 21. Step1-ST_Customer_level_n+1 project flow



MS38275V1

4.5.5 Running the end user application

At this stage the IP-Code must be already preloaded and PCROPed in the STM32F429ZIT MCU, so Step1-ST_Customer_level_n project (PCROP-IP-Code-XO configuration) must be loaded and executed before running the project.

To make the program work, follow the steps described below:

1. Open project located in Step2-ST_Customer_level_n+1 directory and choose the same toolchain used in STEP1.
2. Rebuild all files.
3. Run the example following the sequence below
 - a) Power on the board and load the code (here only user code is loaded)
 - b) Press the user button key to execute the PCROP-ed IP-Code called in *main.c* file, the green LED should toggle continuously.

Note: For more details, refer to the *readme.txt* inside the firmware package.

4.5.6 PCROP protection in debug mode

When developing its end user application, ST Customer level n+1 needs to debug its code. Therefore, when debugging the end user application, PCROP protection should prevent any read access to the IP-Code developed by ST Customer level n.

In this section we will show how the PCROP protects the preprogrammed IP-Code against reading when debugging user code.

The debugging example described below is done on Keil® project.

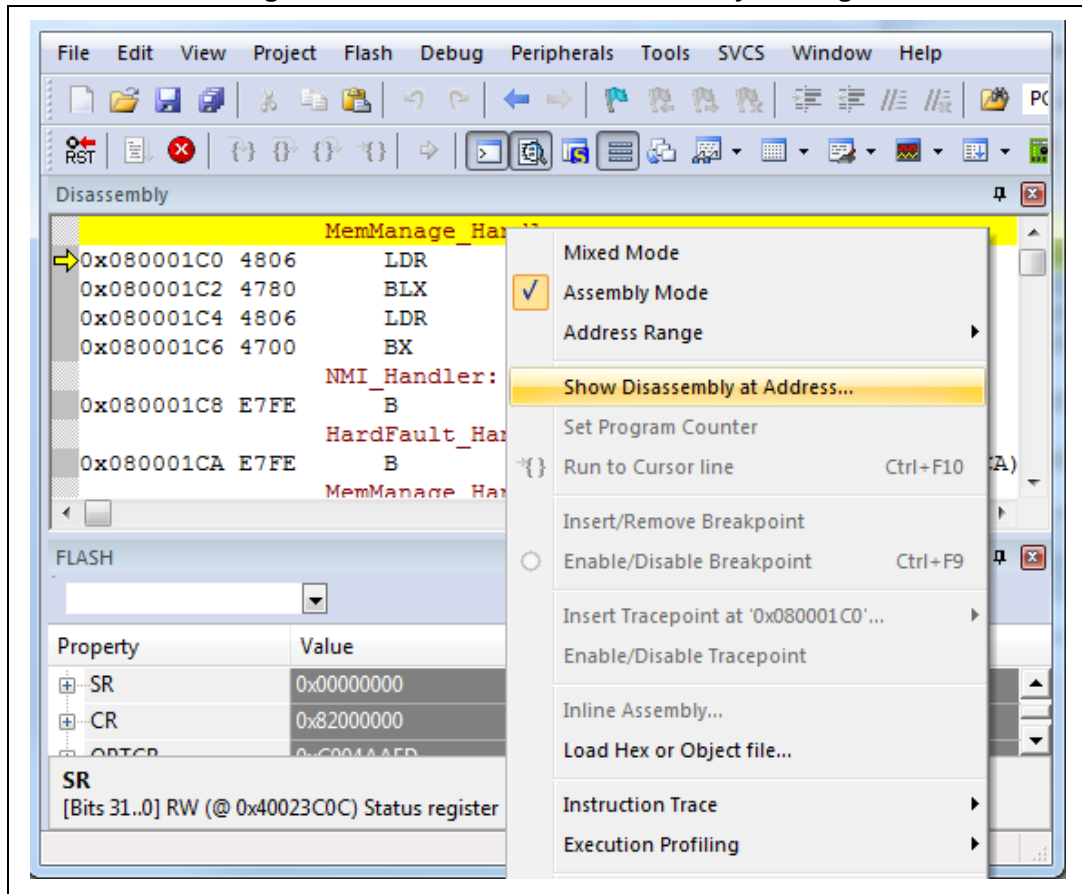
Debugging the end user application

Before debugging end user project, Step1-ST_Customer_level_n project (PCROP-IP-Code-XO workspace) must be loaded and executed in order to get STM32F429ZIT with preprogrammed and PCROP-ed IP-Code.

To debug Step2-ST_Customer_level_n+1 project, follow the steps described below:

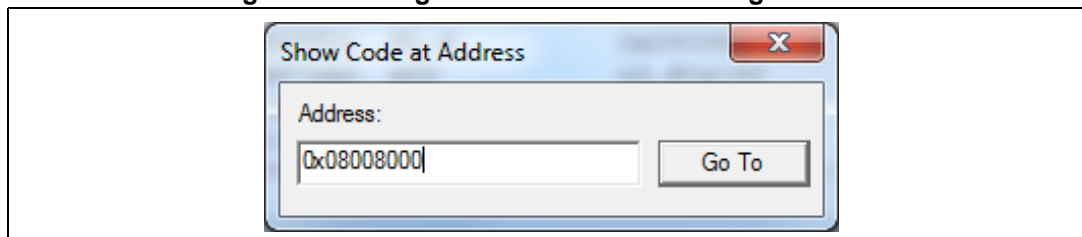
1. Open project located in Step2-ST_Customer_level_n+1 directory and choose the same toolchain used in STEP1.
2. Rebuild all files.
3. Click on Start/Stop Debug session to start debug.
4. Run the program by clicking on Run, red LED should toggle continuously.
5. Press User button to execute IP-Code, the green LED should toggle continuously.
6. To access to the PCROP-ed IP-Code, right click on the Disassembly window then select Show Disassembly at Address as shown in [Figure 22](#).

Figure 22. PCROP-ed IP-Code Assembly reading



- 7. Fill in the address field the PCROP-ed sector starting address and click on **Go To** button as shown in [Figure 23](#).

Figure 23. Filling PCROP-ed sector starting address

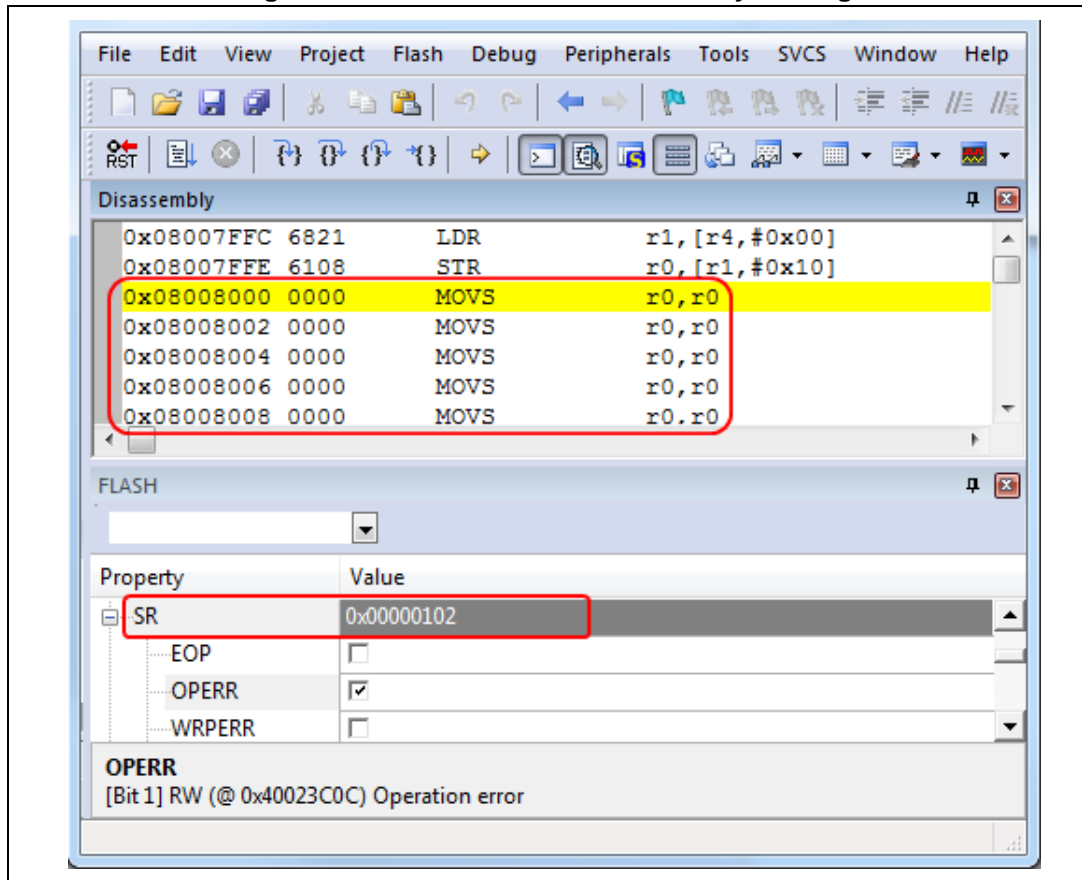


As shown in [Figure 24](#), the PCROP-ed IP-Code loaded in Sector 2 is unreadable while code located just before 0x08008000 address can be read. Reading PCROP-ed sector sets RDERR and OPERR flags in FLASH_SR register.

A Flash operation error interrupt is generated due to Flash memory read operation request through D-Code bus when debugging.

Then a software Reset is initiated in the *HAL_FLASH_OperationErrorCallback()* function (described in *main.c* file) and Red LED4 toggles continuously.

Figure 24. PCROP-ed IP-Code Assembly reading



5 Conclusion

Microcontrollers of the STM32F4 Series provide very flexible and useful Read and/or Write protection features that can be used in applications where protection is required.

This application note shows how Read, Write and PCROP protection features provided on STM32F4xx MCUs can be used.

6 Revision history

Table 5. Document revision history

Date	Revision	Changes
28-Jul-2015	1	Initial release.
12-Oct-2015	2	Added STM32F469/479 products, hence updated Introduction . Updated Keil® scatter file and Debugging the end user application .
29-Nov-2016	3	Added STM32F410, STM32F412 and STM32F413 products, hence updated Introduction and added Table 1: Reference documents .

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved