# AN4758
# Application note

# Introduction to proprietary code read-out protection for STM32L4, STM32L4+, STM32G4 and STM32WB MCUs

## Introduction

Software providers are developing complex middleware solutions known as IP (intellectual propriety) code, whose protection is a high importance issue for microcontrollers.

To respond to this important requirement, STM32L4, STM32L4+, STM32G4 and STM32WB Series MCUs are provided with the following protection features:

- Read-out 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 on Flash and SRAM memories.
- Firewall: access protection to sensitive code and data against external processes.

This application note provides a description of the Flash memory protection features, focusing on the PCROP technique, and providing a basic example of it. Firewall protection (available on STM32L4 and STM32L4+ Series) is described in AN4729 "*STM32L0/L4 FIREWALL overview*", available on *www.st.com*).

The X-CUBE-PCROP firmware package is delivered with this document. It contains the source code of the PCROP example with all firmware modules required to run the example based on STM32L4 Series microcontrollers, and easily portable to STM32L4+, STM32G4 and STM32WB Series microcontrollers.

This application note must be read in conjunction with product datasheets and the following reference manuals, available on *www.st.com*:

- RM0351 (STM32L4x5xx, STM32L4x6xx)
- RM0392 (STM32L4x1xx)
- RM0394 (STM32L43xxx, STM32L44xxx, STM32L45xxx, STM32L46xxx)
- RM0432 (STM32L4Rxxx and STM32L4Sxxx)
- RM0440 (STM32G4xx)
- RM0434 (STM32WB55xx and STM32WB35xx)
- RM0471 (STM32WB50CG and STM32WB30CE)
- RM0473 (STM32WB15CC)
- RM0478 (STM32WB10CC)

# Contents


1 General information . . . . . 6

2 Memory protection description . . . . . 7

2.1 Read-out protection (RDP) . . . . . 7

2.1.1 Read protection Level 0 . . . . . 7

2.1.2 Read protection Level 1 . . . . . 7

2.1.3 Read protection Level 2 . . . . . 8

2.1.4 Internal Flash memory content updating on an RDP protected STM32 . 8

2.2 Write protection . . . . . 9

2.2.1 Flash memory write protection . . . . . 9

2.2.2 SRAM2 CCM-SRAM write protection . . . . . 10

2.3 Proprietary code readout protection (PCROP) . . . . . 10

2.3.1 PCROP protection overview . . . . . 10

2.3.2 How to enable PCROP protection? . . . . . 11

2.3.3 How to disable PCROP protection? . . . . . 12

2.3.4 PCROP-ed IP code compilation . . . . . 13

2.3.5 PCROP-ed IP code dependency . . . . . 13

2.4 Other protections . . . . . 14

2.4.1 Firewall . . . . . 14

2.4.2 Securable memory area for the STM32G4 Series . . . . . 14

3 PCROP example . . . . . 15

3.1 Requirements . . . . . 15

3.1.1 Hardware requirements . . . . . 15

3.1.2 Software requirements . . . . . 15

3.2 Description . . . . . 15

3.2.1 Scenario overview . . . . . 15

3.2.2 PCROP-ed IP code: FIR low-pass filter . . . . . 17

3.2.3 Software settings . . . . . 18

3.3 Development step 1: ST customer - Level n . . . . . 18

3.3.1 Project flow . . . . . 19

3.3.2 Generating an execute-only IP code . . . . . 20

3.3.3 Placing IP code and data segments in Flash memory . . . . . 25

3.3.4 Write protection of constants . . . . . 29

3.3.5 Protecting IP code . . . . . 29
3.3.6 Executing PCROPed IP code . . . . . 30
3.3.7 Creating header file and generating symbol definition file . . . . . 32
3.4 Development step 2: ST customer - Level n+1 . . . . . 34
3.4.1 Project flow . . . . . 35
3.4.2 Creating end user project . . . . . 35
3.4.3 Including header file and adding symbol definition file . . . . . 35
3.4.4 Running the end user application . . . . . 37
3.4.5 PCROP protection in debug mode . . . . . 37

**4 Conclusion . . . . . 41**

**Revision history . . . . . 42**

# List of tables

Table 1. Access status versus protection level and execution modes . . . . . 8
Table 2. Protection area vs. register values . . . . . 11
Table 3. Document revision history . . . . . 42

# List of figures

Figure 1. Two write protected regions per Flash bank . . . 9
Figure 2. Flash memory map with PCROP-ed area. . . . 11
Figure 3. User interface to modify the RDP level . . . 12
Figure 4. User interface to activate PCROP_RDP and activate a PCROP area . . . 12
Figure 5. User interface to program a Write protection area . . . 13
Figure 6. PCROP-ed code calling a function located outside the PCROP-ed region . . . 14
Figure 7. STM32L4 PCROP flow example . . . 16
Figure 8. Example of an ST customer - Level n and level n+1 . . . 16
Figure 9. FIR low-pass filter function block diagram . . . 17
Figure 10. PCROP example software settings . . . 18
Figure 11. Step1-ST_Customer_level_n project flow . . . 19
Figure 12. Example of assembler code containing literal pools . . . 20
Figure 13. Accessing the FIR filter options . . . 21
Figure 14. Setting the Execute-Only code option . . . 21
Figure 15. Accessing FIR-Filter options . . . 22
Figure 16. Setting option “No data reads in code memory” . . . 23
Figure 17. “Assume loading data from flash is slower than fetching” instruction . . . 24
Figure 18. STM32L476VG internal Flash memory map . . . 25
Figure 19. Scatter file modification . . . 26
Figure 20. Enabling PCROP with STM32CubeProgrammer . . . 29
Figure 21. Activating PCROP with STM32 STM32CubeProgrammer . . . 30
Figure 22. Generating symbol definition file with Keil . . . 32
Figure 23. Generating symbol definition file with IAR . . . 33
Figure 24. Generating symbol definition file with STM32CubeIDE . . . 34
Figure 25. ST_Customer_level_n+1 project flow . . . 35
Figure 26. Adding symbol definition file to Keil project . . . 36
Figure 27. Setting symbol definition file type to “Object file” . . . 36
Figure 28. Adding symbol definition file to Keil project . . . 37
Figure 29. PCROP-ed IP code Assembly reading . . . 39
Figure 30. Filling PCROP-ed area starting address . . . 39
Figure 31. PCROP-ed IP code Assembly reading . . . 40
Figure 32. Reading PCROP-ed area sets RDERR flags in FLASH_SR register (bit[14]) . . . 40

# 1 General information

Microcontrollers in the STM32L4, STM32L4+, STM32G4, and STM32WB series use Arm®[a] cores.

arm

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

# 2 Memory protection description

STM32L4, STM32L4+, STM32G4 and STM32WB Series microcontrollers include several mechanisms for read and write protection of the entire memory or specific segments.

Readout protection is used to protect code from extraction by external access (software intellectual property protection), while write protection is used to prevent code or data from being erased unintentionally. In addition to flash memory, these protections also apply to SRAM2 for the STM32L4 and STM32L4+ Series, and to core-coupled memory (CCM) SRAM for the STM32G4 Series.

The STM32L4xx MCUs also feature a firewall mechanism to establish trusted execution areas in memory.

## 2.1 Read-out protection (RDP)

RDP 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 intrusive attacks. This protection must be set by the user after the binary code is loaded into the embedded Flash memory.

The read-out protection applies to:

- main Flash memory
- backup registers in the RTC (real-time clock)
- SRAM2 (STM32L4/L4+) or CCM-SRAM (STM32G4)
- option bytes (Level 2 only).

Three RDP levels (0, 1 and 2) are defined, they are described in the following sections.

### 2.1.1 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, this suits development and debug needs.

### 2.1.2 Read protection Level 1

When the read protection Level 1 is activated, no access (read, erase, and program) to Flash memory or to SRAM2 and CCM-SRAM, respectively, for STM32L4/L4+ and STM32G4 Series can be performed via debug features such as Serial Wire or JTAG even while booting from SRAM or system memory bootloader. In these cases, any read request to the protected region generates a bus error.

When booting from Flash memory, however, accesses to both the Flash memory and to the SRAM2 (STM32L4/L4+ and STM32WB) or CCM-SRAM (STM32G4) from user code are allowed.

Disabling RDP Level 1 protection by re-programming RDP option byte to Level 0 leads to a Flash memory mass erase; SRAM2 (STM32L4/L4+ and STM32WB) or CCM-SRAM (STM32G4) and backup registers are reset as well.

### 2.1.3 Read protection Level 2

When RDP Level 2 is activated, all protections provided in Level 1 are active and the MCU 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 all disabled.

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

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

*Table 1* summarizes read access permission depending upon protection level and execution modes.

**Table 1. Access status versus protection level and execution modes**

| Memory area | Protection level | User execution (Boot from Flash) | | | Debug / Boot from RAM / Boot from loader | | |
|---|---|---|---|---|---|---|---|
| | | Read | Write | Erase | Read | Write | Erase |
| Main Flash memory | Level 1 | Yes | | | No | | |
| | Level 2 | Yes | | | NA(1) | | |
| System memory | Level 1 | Yes | No | | Yes | No | |
| | Level 2 | Yes | No | | NA(1) | | |
| Option bytes | Level 1 | Yes | | | Yes | | |
| | Level 2 | Yes | No | | NA(1) | | |
| Backup registers | Level 1 | Yes | | NA(1) | No | | |
| | Level 2 | Yes | | NA(1) | NA(1) | | |
| SRAM2 or CCM-SRAM | Level 1 | Yes | | NA(1) | No | | |
| | Level 2 | Yes | | NA(1) | NA(1) | | |

1. NA: Not available

### 2.1.4 Internal Flash memory content updating on an RDP protected STM32

When Flash RDP protection is activated (Level 1 or Level 2), the internal memory content cannot be updated any longer through Debug or when booting from SRAM or from System memory bootloader. So an important requirement for the end product is the ability to upgrade embedded firmware in the internal Flash memory with new firmware versions, adding new features and correcting potential issues. This requirement can be resolved by implementing user-specific firmware to perform IAP (in-application programming) of the internal Flash memory by using a communication protocol such as USART for the reprogramming process.

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

## 2.2 Write protection

The write protection is used to protect the content of specified memory area against code segment or non-volatile data update or erase.

### 2.2.1 Flash memory write protection

The number of write protected areas depends upon the Flash memory architecture.

For STM32L4 and STM32L4+ Series, up to two areas can be defined in each Flash memory bank with 2-Kbyte granularity.

STM32G4 Cat3 devices can work in single or dual bank.

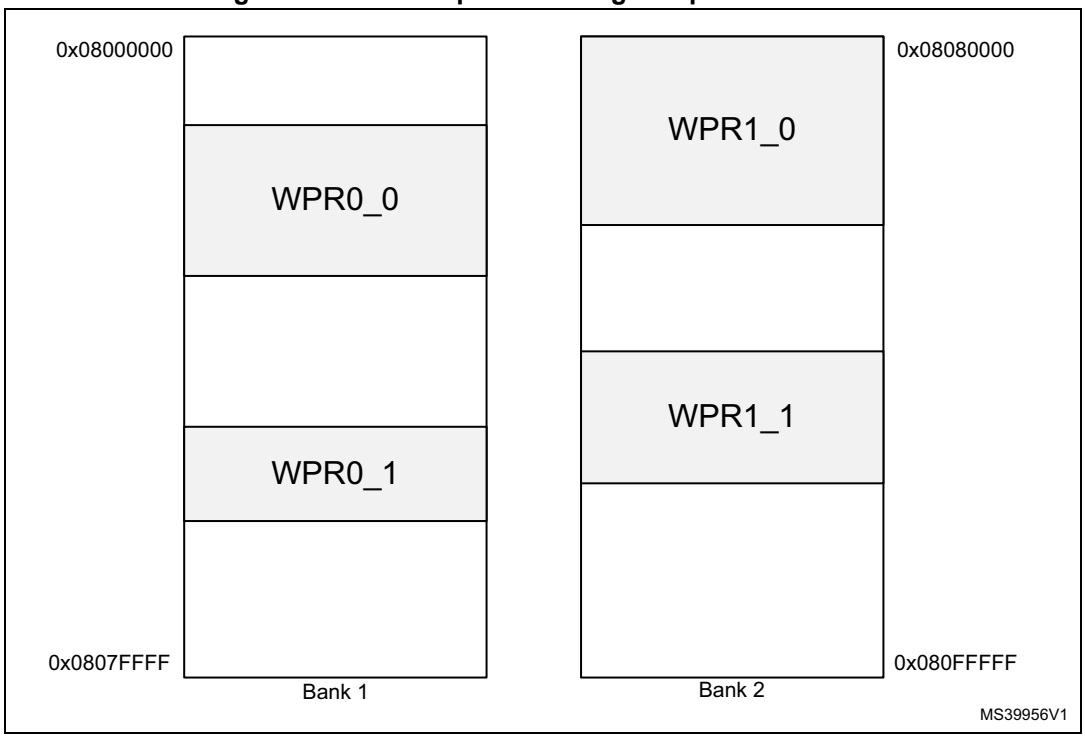
- In single bank mode (DBANK=0), up to four write-protected areas can be defined with 4-Kbyte granularity.
- In dual bank mode (DBANK=1), up to two write-protected areas can be defined with 2-Kbyte granularity in each bank.

STM32G4 Cat2 devices can only work in single Flash memory bank. Two write-protected areas can be defined with 2-Kbyte granularity.

For STM32WB Series, up to two write-protected areas can be defined, with 4-Kbyte granularity.

The gray areas in *Figure 1* are an example of a dual bank organization with two write protected (WRP) regions with 2-Kbyte granularity.

**Figure 1. Two write protected regions per Flash bank**




The protected area cannot be neither erased nor programmed, 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. As an example, the mass erase of a Flash memory where at least one page is write protected is not possible and the WRPERR flag is set.

Enabling or disabling write protection can be managed either by embedded user code or by using STM32CubeProgrammer software and debug interfaces.

### 2.2.2 SRAM2 CCM-SRAM write protection

The 32 Kbytes of SRAM2 in STM32L4/L4+ can be write-protected independently by 1-Kbyte pages. The setting of this protection is controlled by a 32-bit system configuration register and, once enabled, only a system reset can disable it.

In STM32G4 the CCM-SRAM can be write-protected by 1-Kbyte segments (32 Kbytes for Cat3 device, 10 Kbytes for Cat2 device).

For STM32WB the 32 Kbytes of SRAM2a and 32 Kbytes of SRAM2b can be independently write-protected by 1-Kbyte pages, controlled by two 32-bit system configuration registers.

## 2.3 Proprietary code readout protection (PCROP)

### 2.3.1 PCROP protection overview

The PCROP is a read and write protection of an IP code in Flash memory. It prevents proprietary code from possible modification or readout by the end-user code, debugger tools or RAM trojan code.

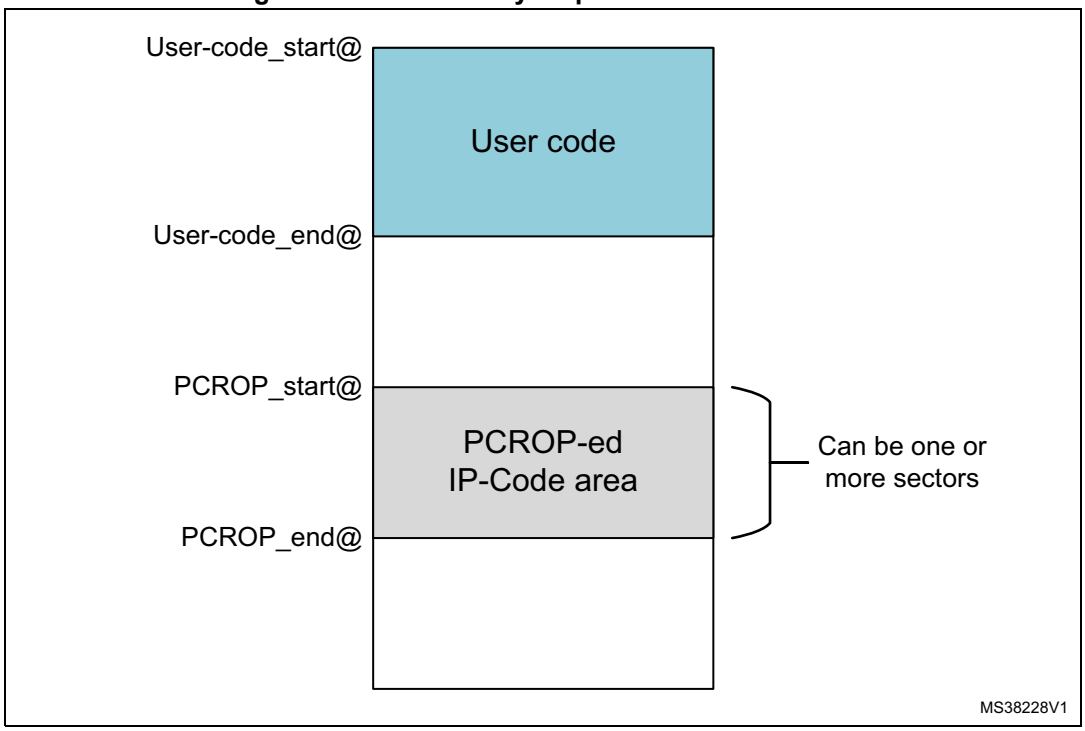
Any read or write request generates a Read or Write protection error:

- 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 area.
- Along with these flags, an interrupt can be raised if enabled by ERRIE bit 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. Then PCROP does not prevent protected codes from being executed.

The PCROP area is set with a fine granularity of 8 bytes, so that no Flash memory is wasted.

**Figure 2. Flash memory map with PCROP-ed area**




### 2.3.2 How to enable PCROP protection?

On STM32L4/L4+ Series, one area per bank can be selected with 64-bit granularity.

In STM32G4 Series, depending of the DBANK mode, one PCROP zone per bank (in dual bank mode) or two PCROP zones (for all memory) can be defined.

In STM32WB Series, up to two PCROP areas can be selected with a 2-Kbyte granularity.

Each PCROP area is defined by a start page offset and an end page offset related to the physical Flash memory bank base address.

To activate the PCROP, start and end address of the protected area shall be programmed in the Flash memory option bytes registers:

- PCROP1SR: PCROP area start address bank 1
- PCROP1ER: PCROP area end address bank 1
- PCROP2SR: PCROP area start address bank 2
- PCROP2ER: PCROP area end address bank 2

*Table 2* specifies how registers values determine the read-out protection area:

**Table 2. Protection area vs. register values**

| PCROP register values | PCROP protection area |
|---|---|
| PCROPxSR = PCROPxER | Memory bank fully protected |
| PCROPxSR > PCROPxER | No PCROP area (no protection) |
| PCROPxSR < PCROPxER | Area between PCROPxSR and PCROPxER is protected |

An additional option bit (PCROP_RDP = PCROP1ER [31]) allows to select if the PCROP area is erased or not when the RDP protection is changed from Level 1 to Level 0. This option is set for both memory banks.

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

### 2.3.3 How to disable PCROP protection?

PCROP can only be disabled if RDP level is 1 or 0. If RDP is set to Level 2, PCROP can no more be disabled; all the option bytes are frozen and can no longer be modified. As a result, PCROPed regions can never be erased or modified, so the protection becomes permanent.

The only way to disable PCROP on a protected region, is by decreasing RDP from Level 1 to Level 0 and deactivate the programmed area at the same time (in embedded SW, set PCROPxSR > PCROPxER).

If PCROP_RDP is set, a mass erase of the Flash main memory is performed. The backup registers in the RTC and the SRAM2 / CCM-SRAM are also erased.

If the bit PCROP_RDP is cleared, the full mass erase is replaced by a partial mass erase that does successive page erases in the bank where PCROP is active, except for the pages protected by PCROP. This is done in order to keep the PCROP code.

#### Use STM32CubeProgrammer

During application development user may need to disable PCROP or global RDP protection using alternatives to embedded Flash memory code. STM32CubeProgrammer tool is a simple method for disabling or enabling protection using debug interfaces as JTAG or SWD without the need for developing dedicated functions. Figures *3* to *5* show how to modify option bytes.

**Figure 3. User interface to modify the RDP level**

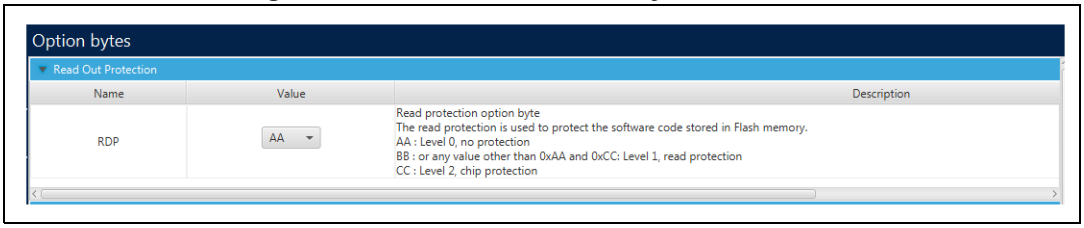



**Figure 4. User interface to activate PCROP_RDP and activate a PCROP area**

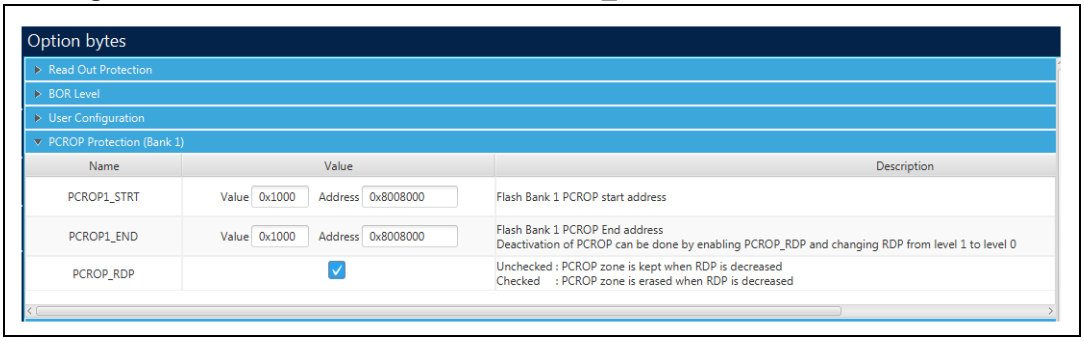
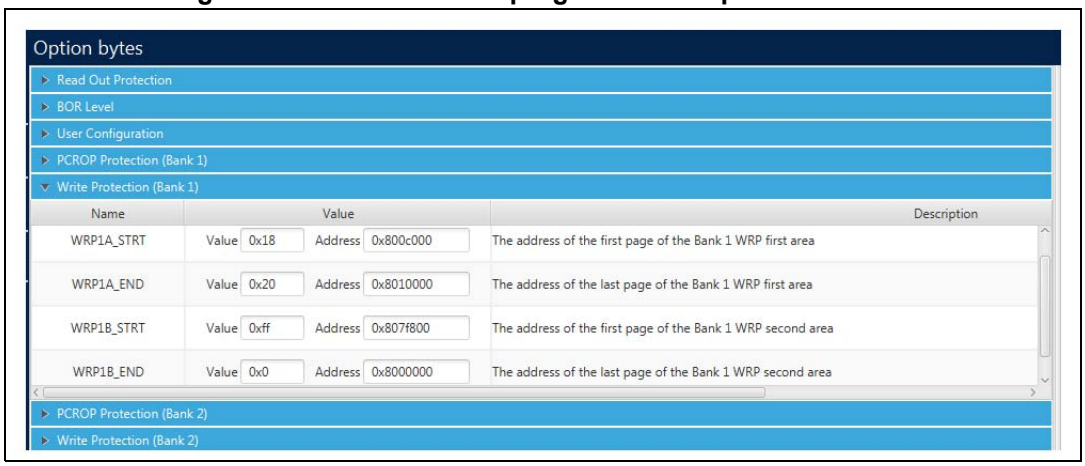


**Figure 5. User interface to program a Write protection area**




For more details on how to use STM32CubeProgrammer software refer to user manual UM2237 available on the Help section of the tool or at *www.st.com*.

### 2.3.4 PCROP-ed IP code compilation

PCROP-ed regions are protected against data bus (D-Code) read accesses, so only code execution is allowed (instruction fetch through I-Code bus) while data reading is not possible. The protected IP code is then unable to access the associated data values stored in the same area such as literal pools or constants fetched from Flash memory through the D-Code bus during the execution.

#### Literal pools

Compilation option must be used to avoid the use of literal pools that would be placed in the code section otherwise. Compilation option is further detailed in *Section 3*.

#### Constant data

Non-volatile data used by the IP code, must be placed in a specific memory region outside the PCROP-ed area. The IP code developer must provide the memory map with these constant data regions. It is advised to write-protect these sections.

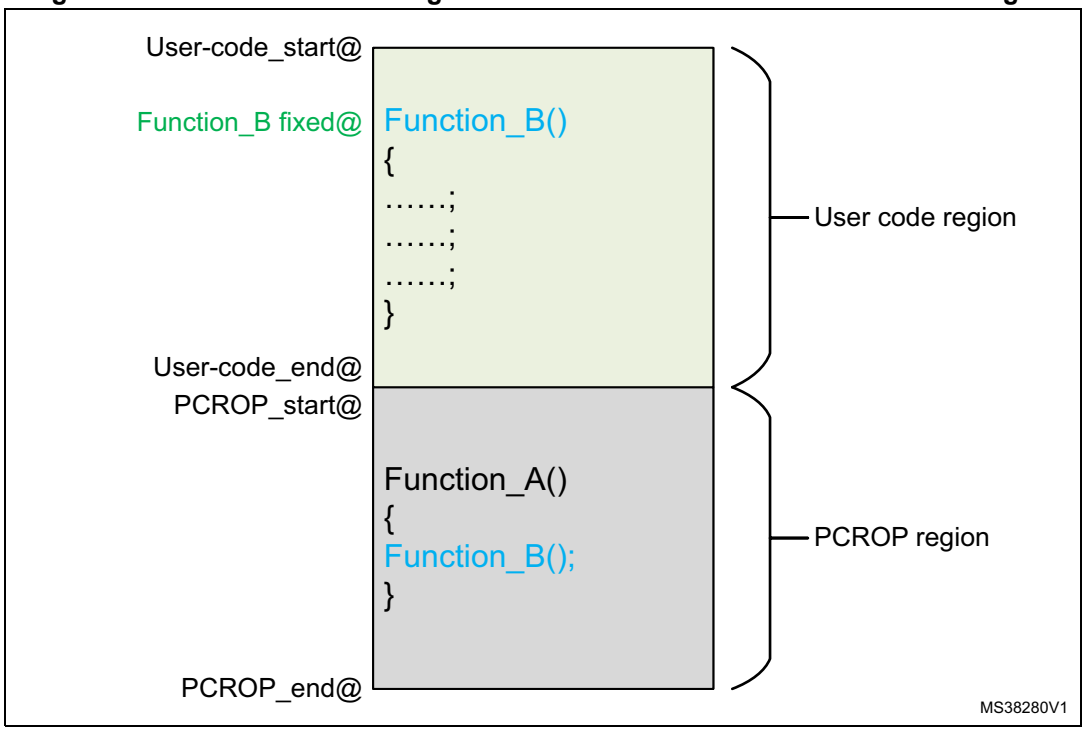
### 2.3.5 PCROP-ed IP code dependency

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 6* shows an example where PCROP-ed Function_A() is calling a Function_B() (light blue) located at a fixed address (in green) outside the PCROP-ed region.

**Figure 6. PCROP-ed code calling a function located outside the PCROP-ed region**




## 2.4 Other protections

### 2.4.1 Firewall

The Firewall is a protection feature available on the STM32L4/L4+ Series. Associated to the other Flash memory protections, it provides an increased level of protection for part of code and data coming from a third party.

The user may want to protect some sensitive algorithms using confidential associated data (e.g. cryptographic keys, secured algorithms and associated variables) somewhere in the memory mapping, and/or to manage exactly when the user code is going to access this trusted area and when this secure area is going to jump back to the non-protected user-code execution. The firewall fills up exactly this role of accesses monitoring (instruction fetches, read, write operations) and generates reset if some non-expected accesses are detected during the code execution to kill immediately any intrusive action inside the protected areas.

Refer to AN4729 for mode detailed description of the feature.

### 2.4.2 Securable memory area for the STM32G4 Series

The securable memory area defines an area of code that can be executed only once at boot, and never again unless a new reset occurs.

The main purpose of this memory area is to protect a specific part of Flash memory against undesired access. It is dedicated to executing trusted code such as secure key storage or safe boot.

# 3 PCROP example

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

This example has been developed for the STM32L4 Series but it is easily portable to the STM32L4+, STM32G4 and STM32WB Series.

## 3.1 Requirements

### 3.1.1 Hardware requirements

The hardware required to run this example is the following:

- an STM32L4-discovery board (RevB or RevC) with embedded STM32L476VG MCU
- a mini-USB cable to power the board and to connect the discovery embedded STLINK for debugging and programming.

### 3.1.2 Software requirements

The following software tools are required:

- IAR Embedded Workbench® (v7.40.3) or Keil® µvision IDE(v5.14.0)
- STM32CubeProgrammer (v2.5.0), to enable or disable PCROP protection.

## 3.2 Description

### 3.2.1 Scenario overview

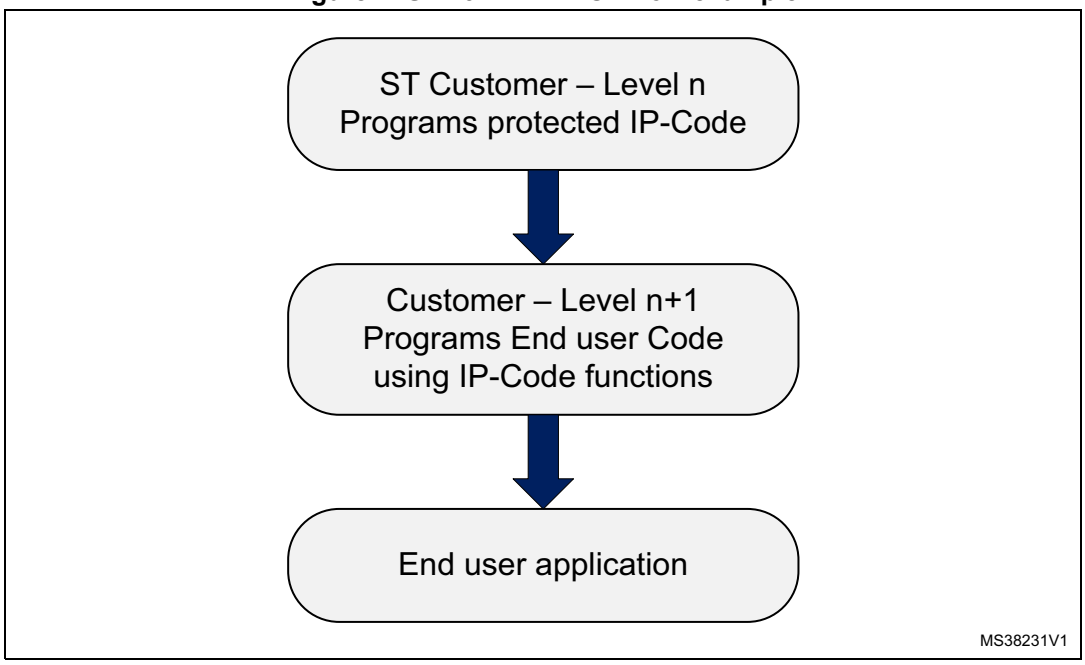
This example describes a use case where an ST customer - Level n provides preprogrammed STM32L476VG MCUs with a critical IP code to an ST customer - Level n+1. The IP code has to be protected by activating PCROP, allowing ST customer - Level n+1 to use its functions (without the ability to read or modify it) to program the end user application.

ST customer - Level n must provide with preloaded STM32 MCUs the following inputs:

- Flash memory map defining the exact protected IP code location, as well as constant data location.
- Header file that has to be included in ST customer - Level n+1 project containing IP code functions definition to be called in End User code.
- Symbol definition file containing IP code function symbols.

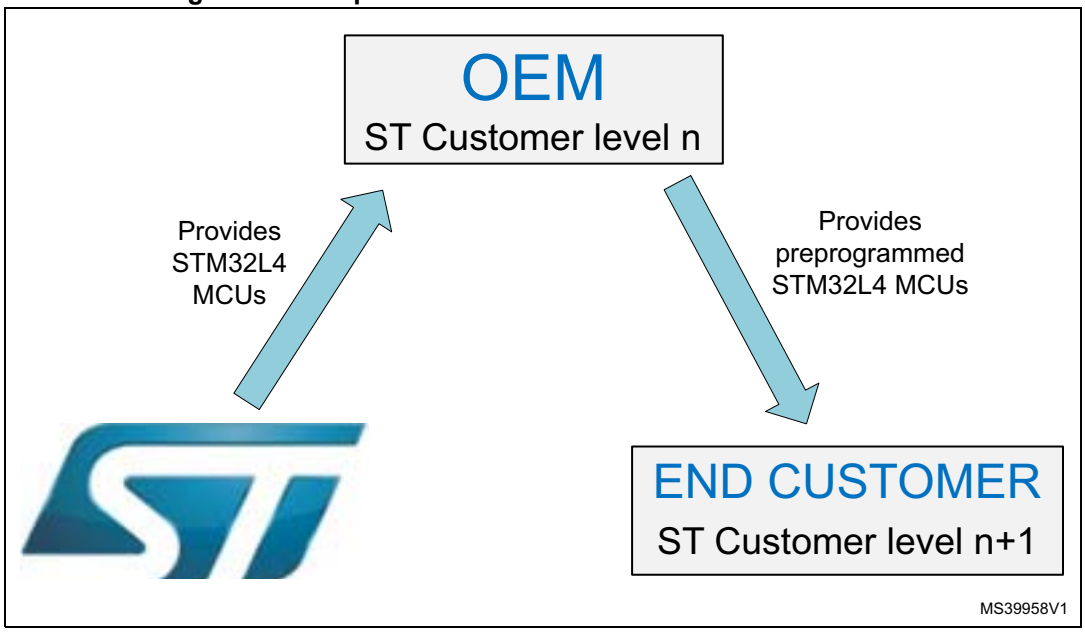
The described use case is schematized in *Figure 7*.

**Figure 7. STM32L4 PCROP flow example**




An OEM (original equipment manufacturer) can be the ST customer - Level n using STM32L4 microcontrollers. The OEM provides preprogrammed MCUs to the ST customer - Level n+1, which can be the one making the end user product, as in *Figure 8*.

**Figure 8. Example of an ST customer - Level n and level n+1**



### 3.2.2 PCROP-ed IP code: FIR low-pass filter

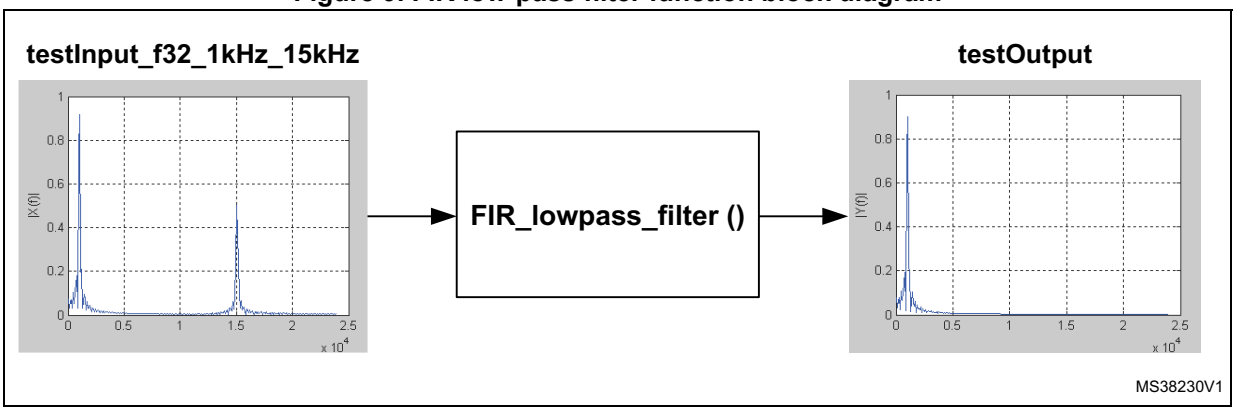
As an example FIR let us consider a low-pass filter algorithm from CMSIS-DSP library as the IP code to be protected, focusing on detailing how to protect and call this IP code example, without providing any details on the functions.

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

The input signal is a sum of two sine waves: 1 kHz and 15 kHz. The low-pass filter (with a preconfigured cutoff frequency of 6 kHz) eliminates the 15 kHz signal leaving the 1 kHz sine wave at the output.

*Figure 9* shows the FIR low-pass filter block diagram.

**Figure 9. FIR low-pass 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 has been created using the CMSIS DSP functions described above:

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

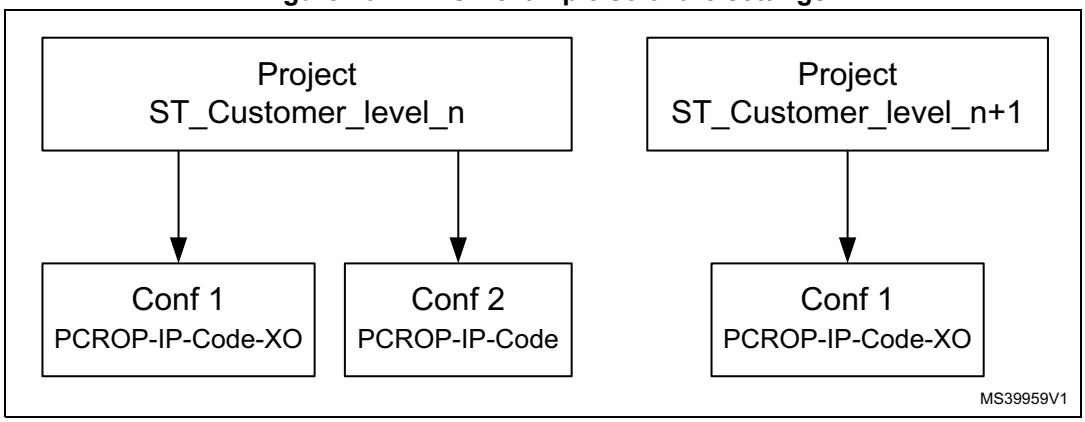
The FPU and DSP embedded in STM32L4 microcontrollers are used for signal processing and floating point calculation to output the correct signal.

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

### 3.2.3 Software settings

This application note describes two projects (*Figure 10*).

**Figure 10. PCROP example software settings**




#### 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 with a preprogrammed STM32L476VG with a PCROP-ed IP code can create its own end user application using the protected IP code functions.

## 3.3 Development step 1: ST customer - Level n

At this stage the ST customer - Level n:

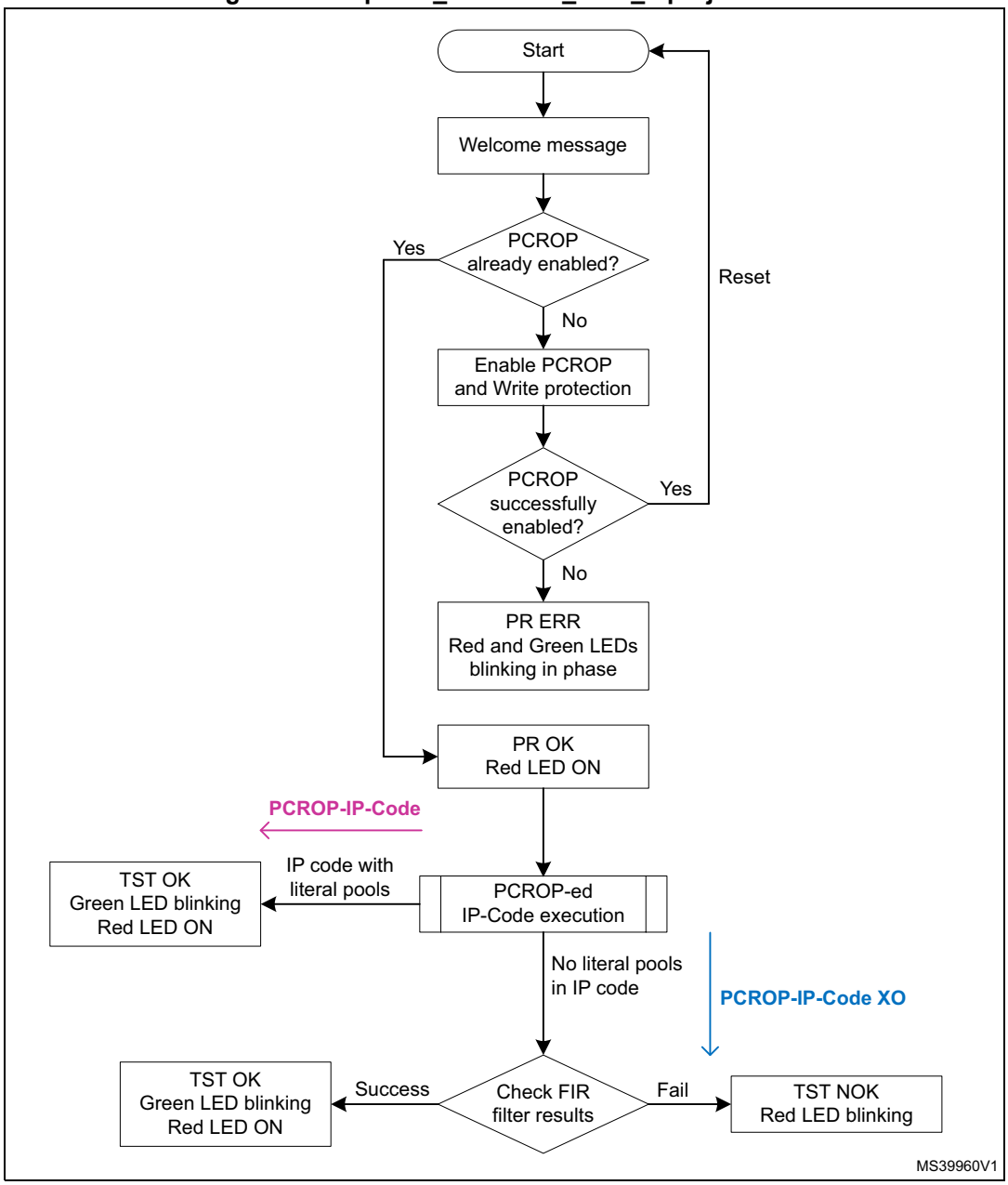
- generates an execute-only IP code
- places IP code and data segments at specified location in Flash memory
- write-protects data segment
- protects the IP code area with PCROP
- executes the IP code by calling its functions in main code
- creates header file and generates symbol definition file to be used in STEP2-ST_Customer_level_n+1 project.

### 3.3.1 Project flow

*Figure 11* illustrates the Step1-ST_Customer_level_n project flow with both project configurations. End of test differs according to the chosen configuration:

- PCROP-IP-Code-XO (blue arrow): PCROP-ed IP code does not contain any literal pool, FIR filter algorithm runs successfully
- PCROP-IP-Code (red arrow): IP code containing literal pools, when starting PCROP-ed IP code execution, a Read Operation Error Interrupt is generated.

**Figure 11. Step1-ST_Customer_level_n project flow**

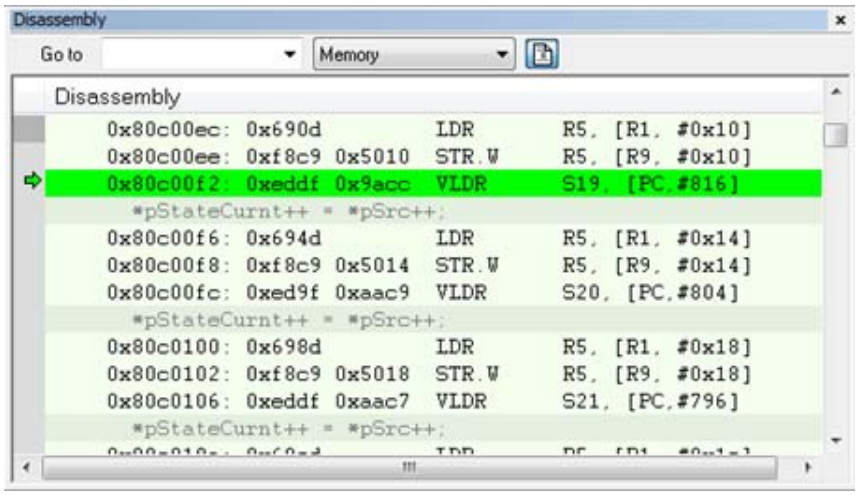


### 3.3.2 Generating an execute-only IP code

Each tool chain 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. For STM32CubeIDE the option is called “Assume loading data from flash is slower than fetching instruction”.

*Figure 12* shows an assembler code containing literal pools, where the instruction has the form of VLDR <variable>, [ PC + <offset> ].

**Figure 12. Example of assembler code containing literal pools**

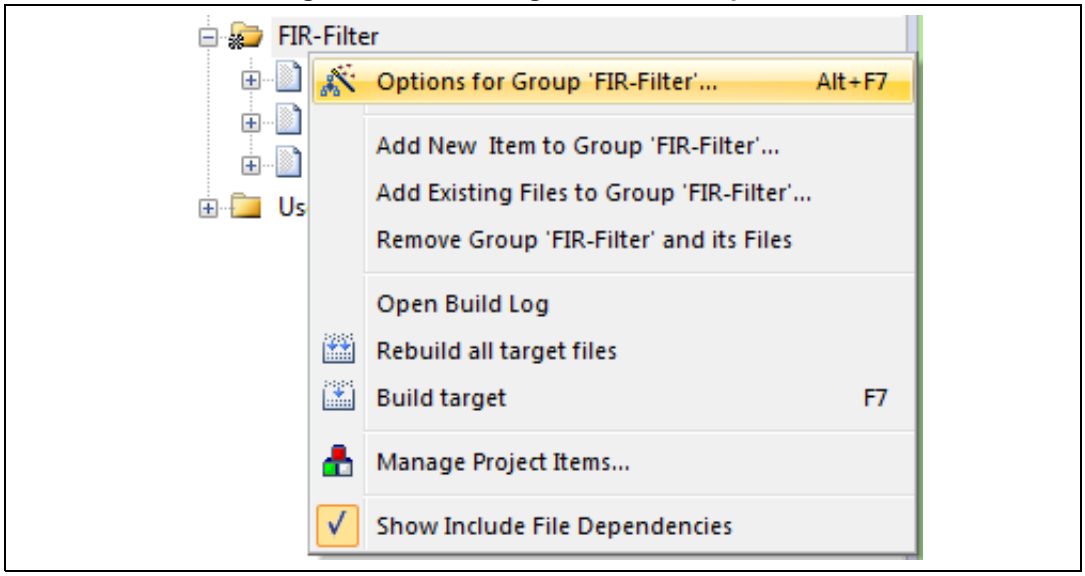



#### Keil - Execute-only compile option

Compilation option *-execute_only* must be used to generate a code without literal pools, preventing the compiler from generating any data accesses to code sections. This option must be used for all IP code files containing literal pools.

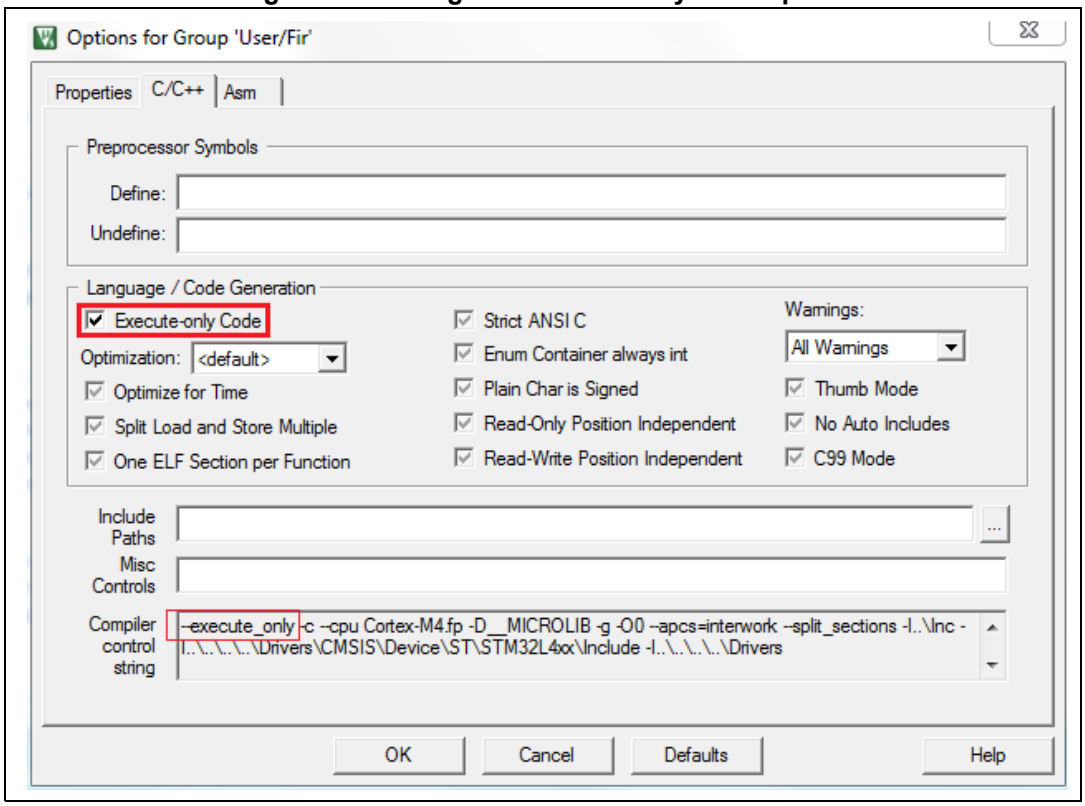
To set this command right click on the group file or the IP code and (see *Figure 13*) choose Options for Group ‘User/Fir:

**Figure 13. Accessing the FIR filter options**




In the following window check the option Execute-only Code, as indicated in *Figure 14*: *-execute_only* option is added in compiler control string field:

**Figure 14. Setting the Execute-Only code option**



The scatter file defining the memory region placement of the IP code sets the access type attribute to +XO. Refer to *Section 3.3.3*.

### IAR: No data reads in code memory

Option “No data reads in code memory” must be used to generate a code without literal pools. To activate it, right click on the file group ‘Fir’ containing IP code source files (see *Figure 15*), then select Options, and then, in the following window check the option “No data reads in code memory”, as indicated in *Figure 16*.

**Figure 15. Accessing FIR-Filter options**

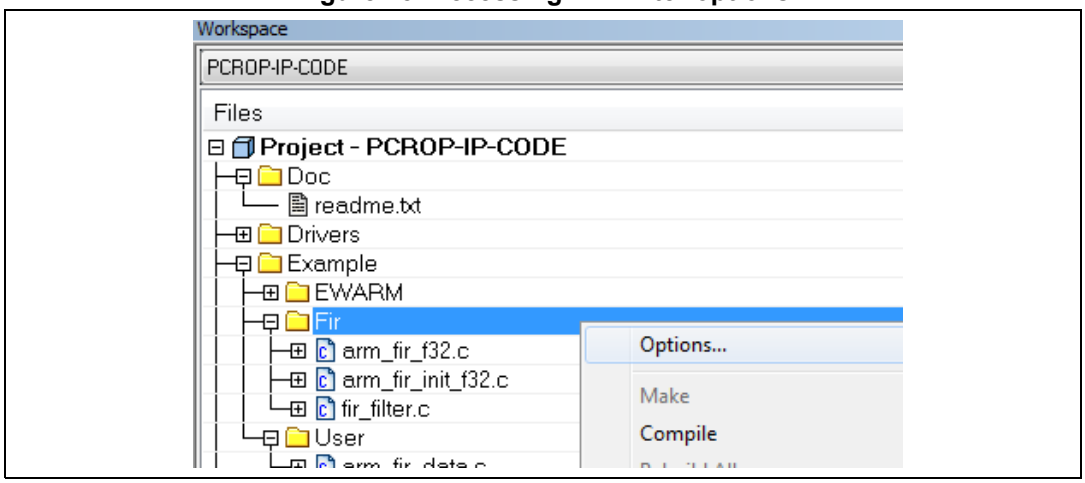
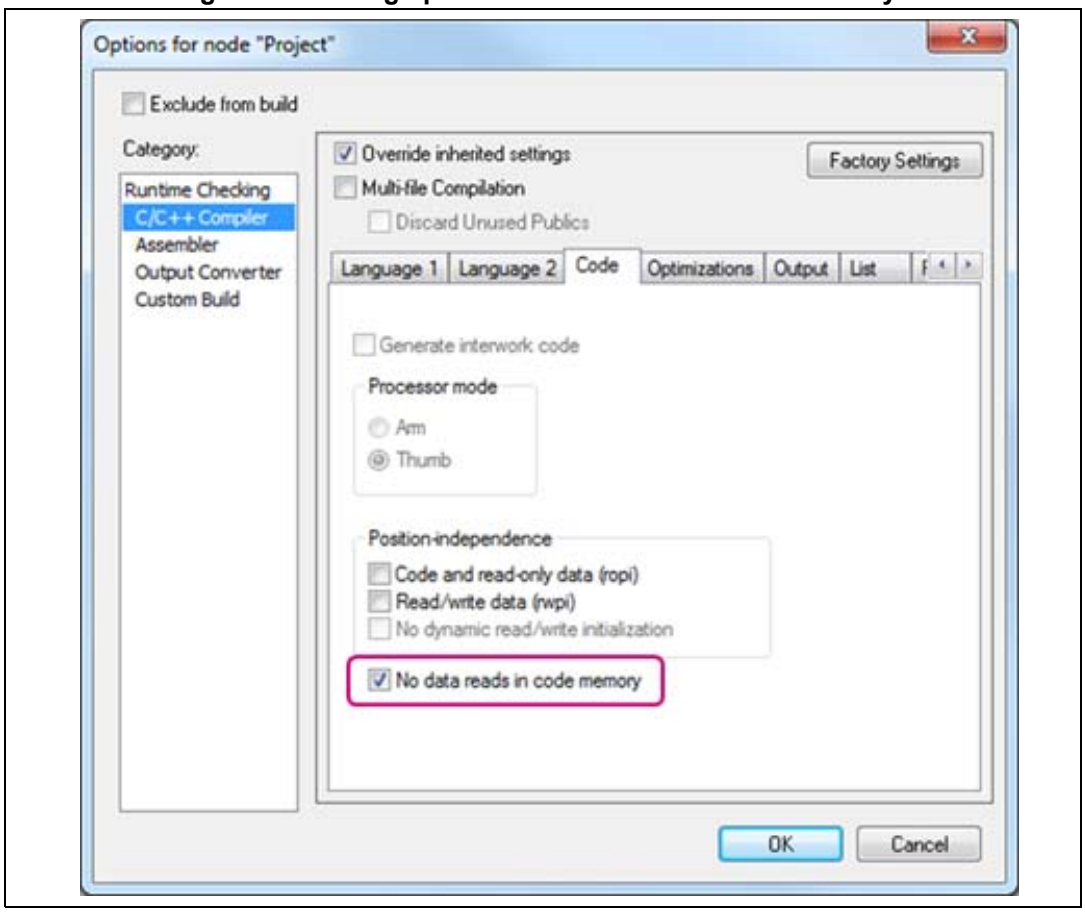


**Figure 16. Setting option “No data reads in code memory”**

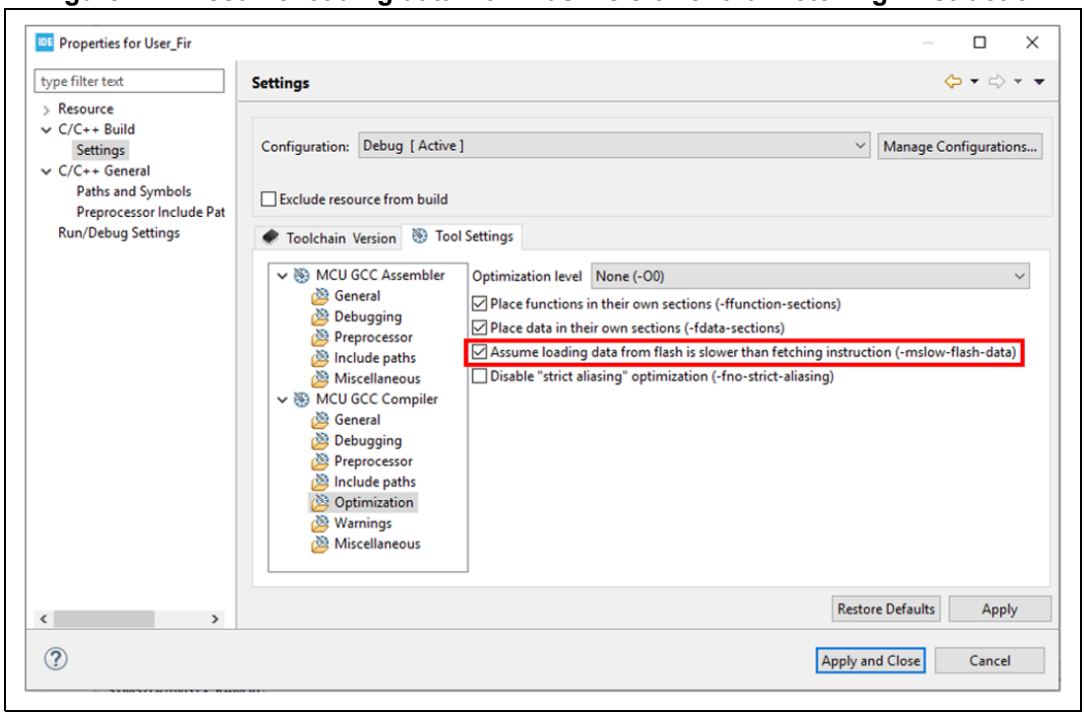


### STM32Cube IDE: “Assume loading data from flash is slower than fetching”

The same option must be used with STM32CubeIDE to prevent the compiler from generating any data accesses to code sections. To activate this option, right click on the file group “User_Fir” containing IP-code source files then select “Properties”.

Then, in the following window check the option “Assume loading data from flash is slower than fetching” instruction, as indicated in *Figure 17*.

**Figure 17. “Assume loading data from flash is slower than fetching” instruction**



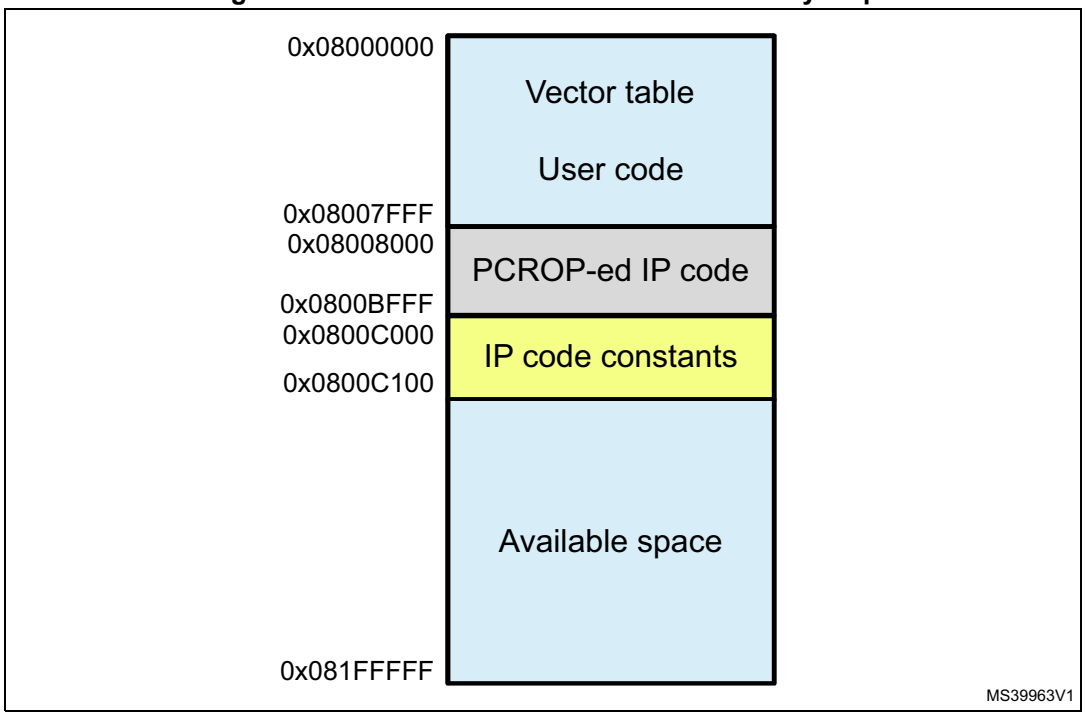
### 3.3.3 Placing IP code and data segments in Flash memory

In this application note example, two specifics memory sections are defined for IP code segment and its associated constant data (FIR filter coefficients).

- IP code segment is located in Flash memory bank 1 at address 0x0800_8000
- Constant data located just after the code section at address 0x0800_C000.

Main memory region for user code and vector table is located at the beginning. The resulting Flash memory map is shown in *Figure 18*.

**Figure 18. STM32L476VG internal Flash memory map**




Memory placement are handled by scatter file for Keil IDE, Linker configuration file (ICF) in IAR and the linker file in STM32CubeIDE.

#### Keil scatter file

The scatter file has to be updated with the two new regions:

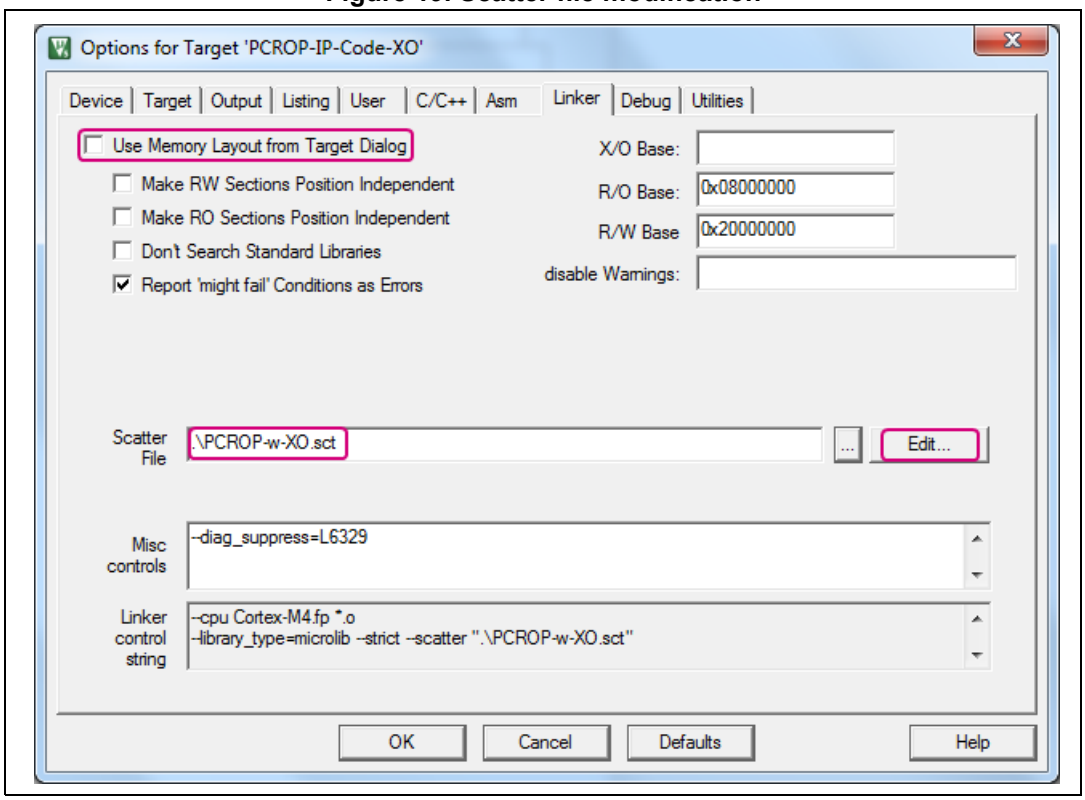
- LR_PCROP is the IP code region with execute-only access attribute
- LR_DATA is the constant data memory region used by the IP code. A new section is defined, namely “fir_coeff_section”.

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************
; here stands ROM and sram memory regions
..
..
```

```
; *******************************
; PCROP-ed area
; *******************************
LR_PCROP 0x08008000 0x00004000  {
  ER_PCROP 0x08008000 0x00004000  {  ; load address = execution address
   arm_fir_init_f32.o (+XO) ; +XO is for execute only access
   arm_fir_f32.o (+XO)
   FIR_Filter.o (+XO)
  }
}
; Define a section of 2KB for FIR coefficients
; This section correspond to Flash page number = 24
LR_DATA 0x0800C000 0x00000800  {
  fir_coef_section 0x0800C000 0x00000800  {
   FIR_Filter.o (+RO)
  }
}
```

Select this new scatter file to the linker in Project → Options for Target. Select the Linker tab then uncheck Use Memory Layout from Target Dialog as shown in *Figure 19*.

**Figure 19. Scatter file modification**



### IAR ICF file

Two memory regions are defined

- PCROP_region is the IP code region.
- Fir_coef_region is memory area that will host the fir coefficients thanks to the right attribute in the array declaration (see *Explicit data placement*).

Note that the compiled object fir_filter.o contains code and global constant data. A split between code and data is explicit in the ICF file.

```
/* define PCROP area start and end address */
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__];

/* define 2KB page of constant data for fir coefficients */
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__];

/* Place IP-CODE in PCROP memory region*/
place in PCROP_region { ro object arm_fir_f32.o,
ro object arm_fir_init_f32.o,
ro code object FIR_Filter.o};
/* Place IP-CODE in PCROP memory region*/
place in fir_coef_region { ro data section fir_coef_section object
FIR_Filter.o };
```

### STM32Cube IDE

Two memory regions are defined:

- PCROP_region is the IP code region
- Fir_coef_region is memory area hosting the fir coefficients thanks to the correct attribute in the array declaration (see *Explicit data placement*).

```
/* Memories definition */

MEMORY
{
  RAM    (xrw)     : ORIGIN = 0x20000000,   LENGTH = 96K
  RAM2    (xrw)     : ORIGIN = 0x10000000,   LENGTH = 32K
  FLASH    (rx)     : ORIGIN = 0x8000000,   LENGTH = 1024K
  PCROP (x)       : ORIGIN = 0x08008000, LENGTH = 16K
  CONST (rx)      : ORIGIN = 0x0800C000, LENGTH = 2K
}
```

```
/* Place IP-code in sector 16 which will be PCROP-ed */

  .PCROPedCode :
  {
  . = ALIGN(4);
   *arm_fir_f32.o  (.text .text*)
   *arm_fir_init_f32.o (.text .text*)
   *fir_filter.o (.text .text*)
  . = ALIGN(4);
  } >PCROP

/* Place IP Data in sector 24 which will be WP-ed */

  .WPedData :
  {
  . = ALIGN(4);
   *fir_filter.o (.fir_coef_section .rodata*)
  . = ALIGN(4);
  } >CONST
```

### Explicit data placement

Constant data are mapped in this section thanks to the next compilation attribute. The following code extract shows the syntax for IAR and Keil.

```
/* IAR IDE */
/* Placement with specific address: */
const float32_t firCoeffs32[NUM_TAPS] @ 0x0800C000 = { …};
/* Placement with specific section defined in ICF file: */
const float32_t firCoeffs32[NUM_TAPS] @ "fir_coef_section" = { ….};

/* KEIL IDE */
/* Placement with specific address: */
const float32_t firCoeffs32[NUM_TAPS] __attribute__((at(0x0800C000))) ={
…};
/* Placement with specific section defined in scatter file: */
const float32_t firCoeffs32[NUM_TAPS]
__attribute__((section("fir_coef_section"))) = {…};

/* STM32CUBEIDE */
/* Placement with specific address: */
const float32_t firCoeffs32[NUM_TAPS] attribute ((section(".0x0800C000")))
={…};
```

```
/* Placement with specific section defined in scatter file: */
const float32_t firCoeffs32[NUM_TAPS]
  attribute ((section("fir_coef_section"))) = {…};
```

### 3.3.4 Write protection of constants

User must take into account that IP code constants can be deleted or modified by ST customer - Level n + 1 and functions can become useless, so it is recommended to protect these constants from unwanted write/erase operations.

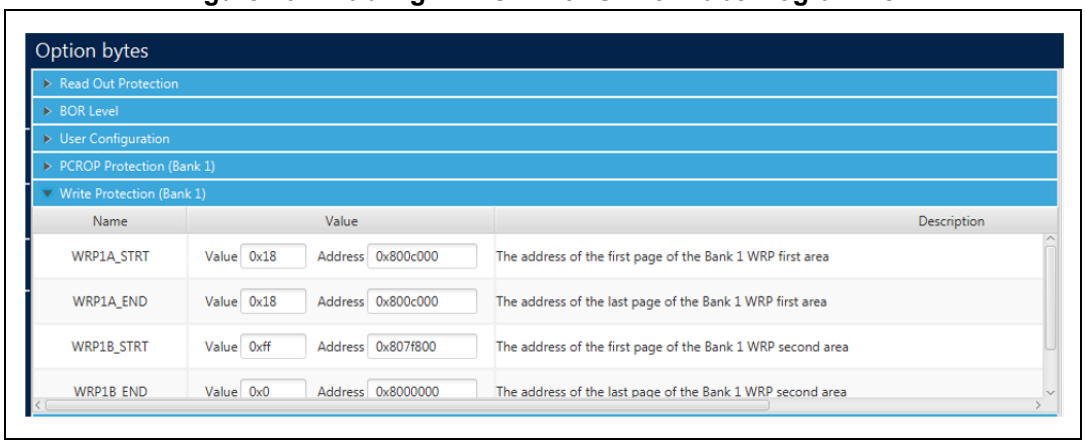
Write protection area is set through the following option bytes registers:

- FLASH_WRP1AR: First write-protected area of bank 1 (start and end page number)
- FLASH_WRP1BR: Second write-protected area of bank 1
- FLASH_WRP2AR: First write-protected area of bank 2
- FLASH_WRP2BR: Second write-protected area of bank 2

The IP code of the example uses 116 bytes as constant coefficients. The minimum page size is 2 Kbytes, so a single page is protected (between addresses 0x0800C000 and 0x0800C100).

Write protection can be set either by dedicated FW code by setting the right values in dedicated registers (see the *PCROP_Enable()* function) or by using STCubeProgrammer, as shown in *Figure 20*.

**Figure 20. Enabling PCROP with STM32CubeProgrammer**




If WRPxy_STRT = WRPxy_END, page WRPxy is protected. To protect Sector 24 of the Flash memory write 0x18 to WRPxy_STRT and WRPxy_END (the whole sector becomes protected).

### 3.3.5 Protecting IP code

As for Write protection, PCROP area can be set either by dedicated FW code or by using STCubeProgrammer.

### Activating PCROP using FW

*PCROP_Enable ()* function sets the protected area at the first run. Once the area is defined, a system reset is generated to take the Flash memory protection into account. At the second run, code continues up to the end of test.

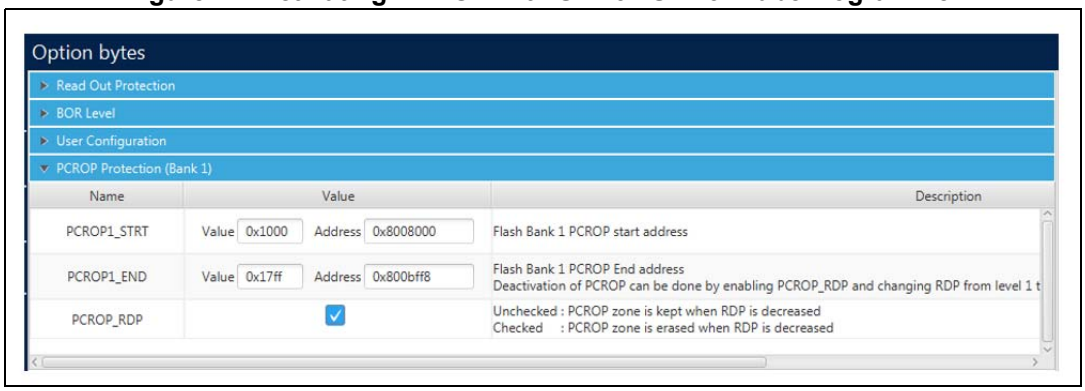
This function is defined in the Step1-ST_Customer_level_n project main.c file.

### Activating PCROP using STM32CubeProgrammer

To activate PCROP using STM32 STM32CubeProgrammer user must follow the following sequence:

- Click on the Connect button
- In Option Bytes tab click on the PCROP Protection (Bank 1) Menu, and enable protection on bank A by specifying start and end address of area as shown in *Figure 21*.
- Click on Apply button.

**Figure 21. Activating PCROP with STM32 STM32CubeProgrammer**




## 3.3.6 Executing PCROPed IP code

Once IP code has been programmed on specified area and protected, it has to be tested by calling its functions in user code. This section shows 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 be used before running 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 the 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 area. If yes, disable the protection using STM32CubeProgrammer, then load the code.
   b) Press the reset button to start the test:

      A welcome message is displayed “WELCOME PCROP STEP 1”

      If PCROP is enabled the red light is set, else the PCROP area is programmed and a system reset occurs looping on welcome message[a]

      Application test is running and result is checked. If test is passed, the green light toggles infinitely.

### PCROP-IP-Code (for test only, not be used for STEP2)

No special compiler option is 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 area. If yes, disable the protection using STM32CubeProgrammer, then load the code.
   b) Press the reset button to start the test:

      A welcome message is displayed “WELCOME PCROP STEP 1”

      If PCROP is enabled the red light is set, else the PCROP area is programmed and a system reset occurs looping on welcome message[a]

      Application test is running but an interrupt is raised and the board displays the “IT MEM” error message. Both LEDs are ON.

### Interpretation

The low-pass filter function computes the *testInput_f32_1kHz_15kHz* input signal and outputs a 1 kHz sine wave. The output data *testOutput* is then compared to the reference *refOutput* already calculated with MATLAB, if it matches, the test is passed (“TST OK” is displayed and green 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 cannot be accessed through D-Code bus, then the RDERR flag is set. An interrupt is generated and *HAL_FLASH_OperationErrorCallback()* function is called.

For the PCROP-IP-Code-XO configuration the IP code is executed and test ends correctly.

a. This system reset disconnects the device from a possible debug session.

### 3.3.7 Creating header file and generating symbol definition file

Once the IP code development is over, header file and symbol definition file shall be given to the following developer / customer.

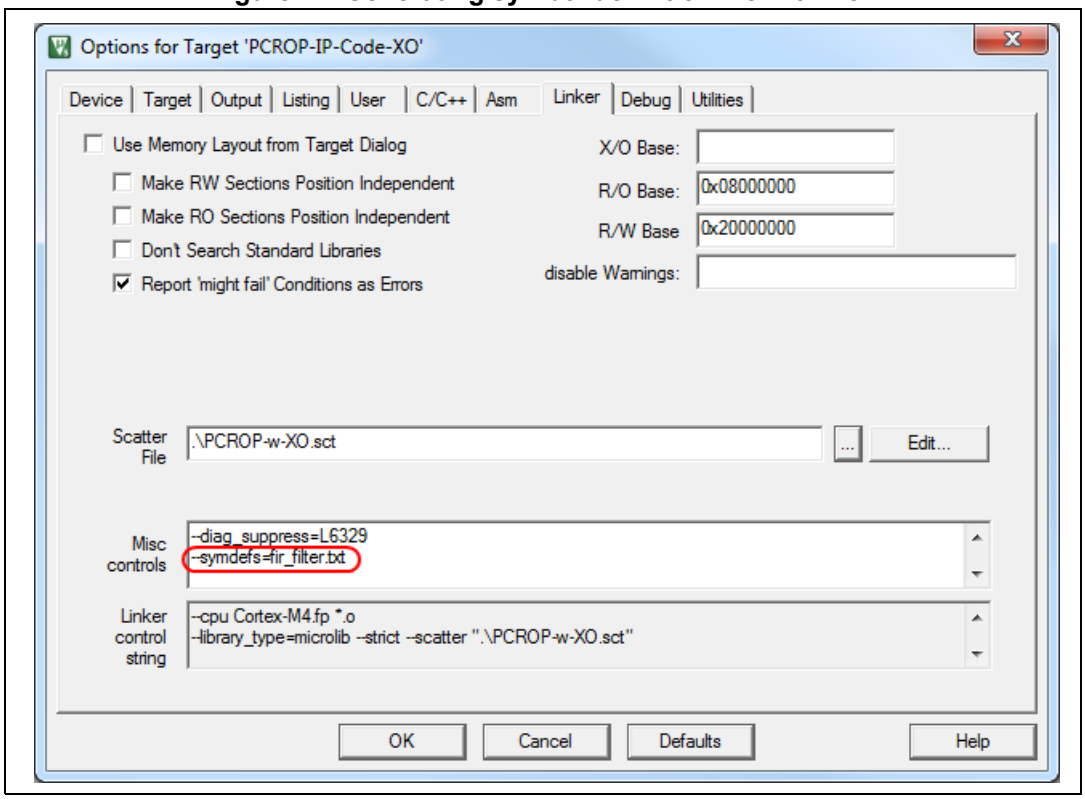
#### Header file

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 is the *fir_filter.h* file included in the *main.c* file.

#### Symbol definition file with Keil

To generate a symbol definition file with Keil go to Project options in the Linker tab and add the command *-symdefs=fir_filtre.txt* (see *Figure 22*). Rebuild the project.

**Figure 22. Generating symbol definition file with Keil**




The symbol definition file named fir_filtre.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 user only must keep those of the IP code functions to be called by the end user, all other ones must be removed.

The resulting symbol definition file is:

```
#<SYMDEFS># ARM Linker, 5050106: Last Updated: Tue Sep 01 14:22:05 2015
0x08008001 T FIR_lowpass_filter
0x08008051 T arm_fir_f32
0x0800871b T arm_fir_init_f32
```

### Symbol definition file with IAR

The IAR Absolute Symbol Exporter, *isymexport*, is used to export absolute symbols from a ROM image file.

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

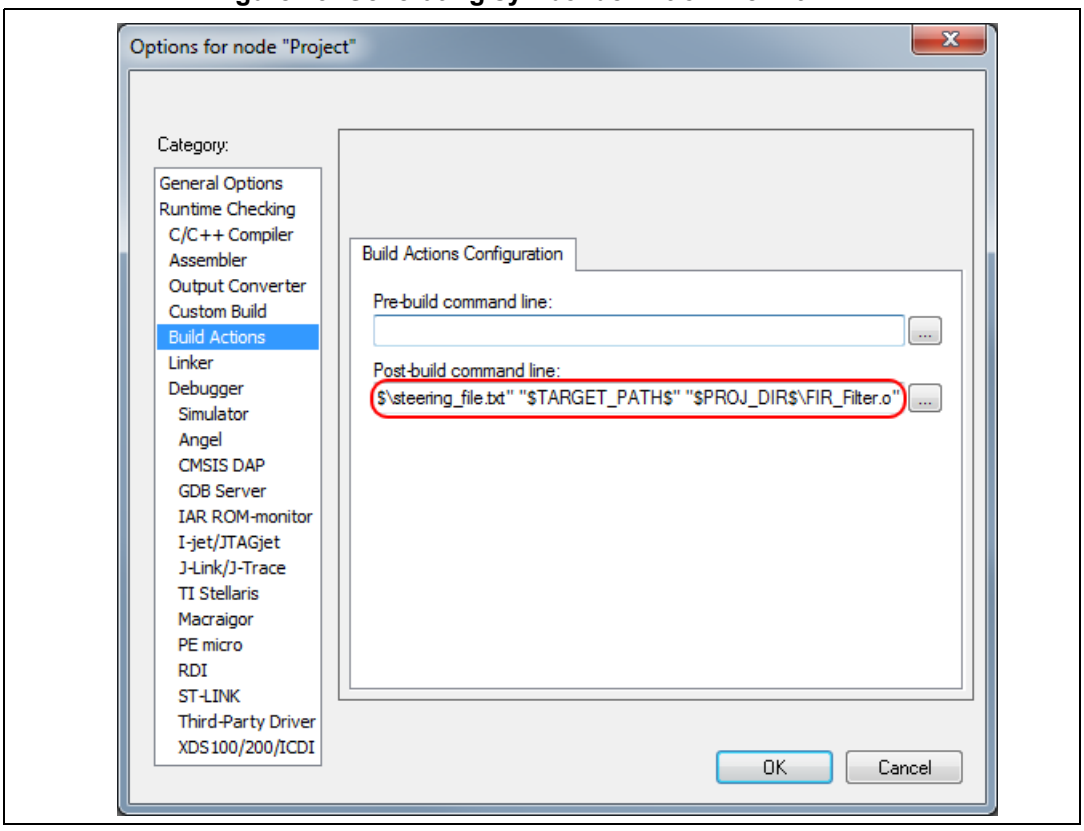
```
$TOOLKIT_DIR$\bin\isymexport.exe --edit "$PROJ_DIR$\steering_file.txt"
"$TARGET_PATH$" "$PROJ_DIR$\fir_filter.o"
```

The *–edit steering_file.txt* option is used to keep only symbols for functions to be called by End User Code. In the *steering_file.txt* the command “show” is used to keep only selected functions:

- *show arm_fir_f32*
- *show arm_fir_init_f32*
- *show FIR_lowpass_filter*

The output file *fir_filter.o* is used by end user by adding it to the project.

**Figure 23. Generating symbol definition file with IAR**



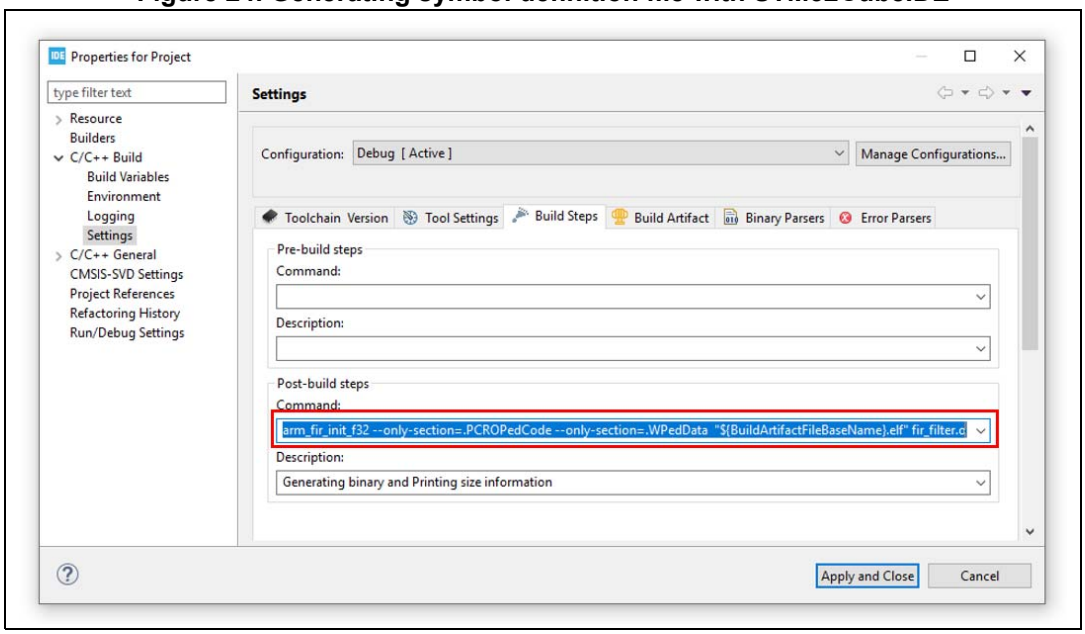
### Symbol definition file with STM32CubeIDE

In the IDE, to export the PCROP-ed IP-code symbols, choose
Project →Properties →C/C++ Build →Settings →Build Steps and specify the following command line in the Post-build steps command line text field, as shown in *Figure 24*.

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

This command keeps only the following IP-code functions in the symbol file arm_fir_f32 arm_fir_init_f32 FIR_lowpass_filter. The generated symbol definition file (fir_filter.o) is used by adding it to the STEP2- ST_Customer_level_n+1 STM32CubeIDE project.

**Figure 24. Generating symbol definition file with STM32CubeIDE**




## 3.4 Development step 2: ST customer - Level n+1

ST customer - Level n+1 gets the preloaded STM32L476VG MCUs with the PCROP-ed IP code, the provided symbol definition and header files from ST customer - Level n.

Referring to the Flash memory map provided by the ST customer - Level n,
ST customer - Level n+1 must:

- create the 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:** 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, or it cannot 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.

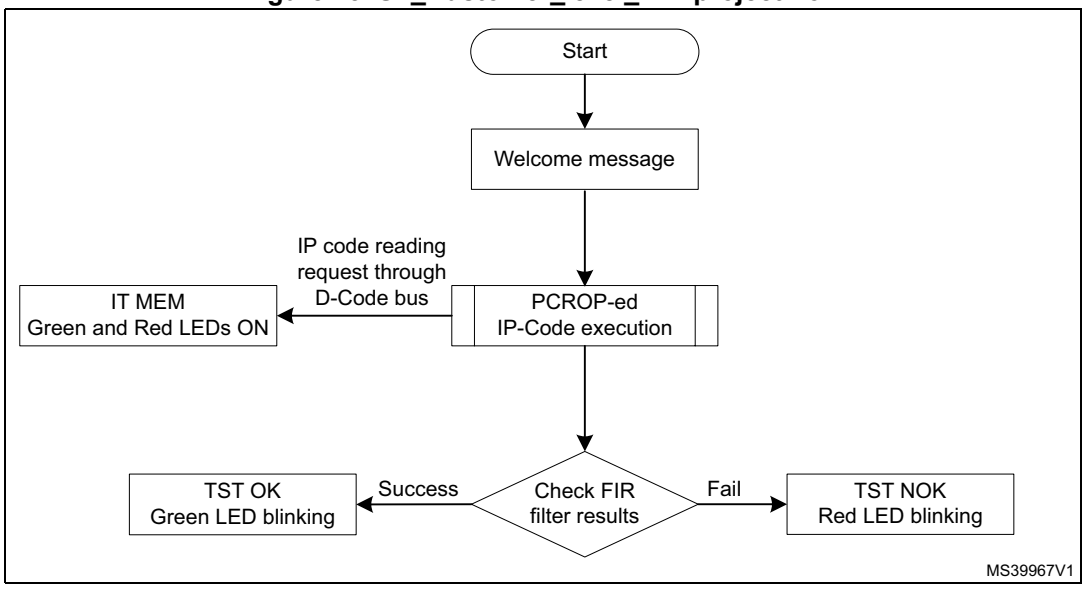
### 3.4.1 Project flow

*Figure 25* illustrates the ST_Customer_level_n+1 project flow. The code executes with success unless a read operation (a debug access or a memory dump) occurs on the protected area.

The interrupt is signaled by the specific message “IT MEM”.

The main test may fail and end up with the message “TSTNOK”. This can happen if constant data (here, filter coefficients) are corrupted or have been erased.

**Figure 25. ST_Customer_level_n+1 project flow**




### 3.4.2 Creating end user project

Protected code and write-protected data segments are located at known position of memory map (see *Figure 18*), hence the user must take care when implementing linker file to avoid placing code in these regions.

PCROPed IP code functions are then used to create the end user application. The project located in Step2-ST_Customer_level_n+1 directory is an example where PCROPed FIR filter functions are called in *main.c* file.

### 3.4.3 Including header file and adding symbol definition file

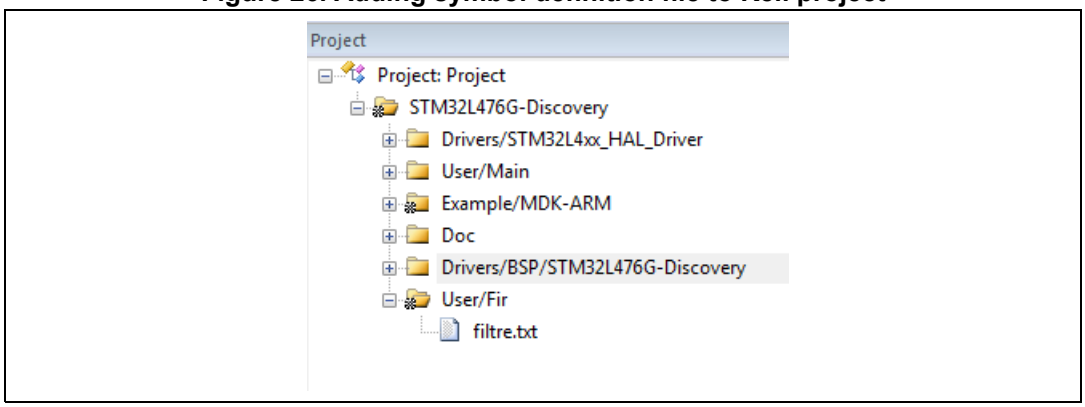
The header file *fir_filtre.h* is included in *main.c* file so that user can call FIR filter functions. The file must be added in the include search path of the compiler settings.

The symbol definition file provided from ST customer - Level n must be added as a classic source file, necessary to link properly. User must follow the method depending on the chosen IDE.

### Adding symbol definition file in Keil project

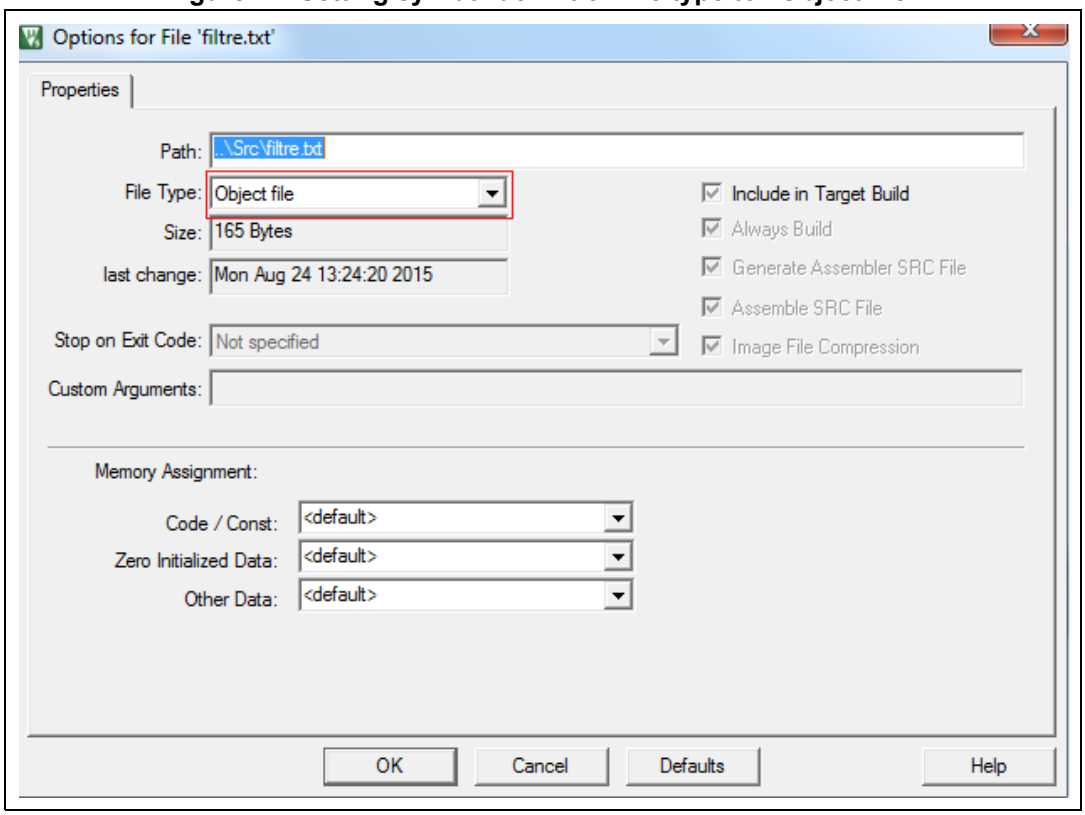
The symbol definition file described in *Section 3.3.7* and named *filtre.txt* in this example is added to User/Fir group as described in *Figure 26*.

**Figure 26. Adding symbol definition file to Keil project**




The added filtre.txt file type must be changed to Object file (see *Figure 27*) instead of text document file.

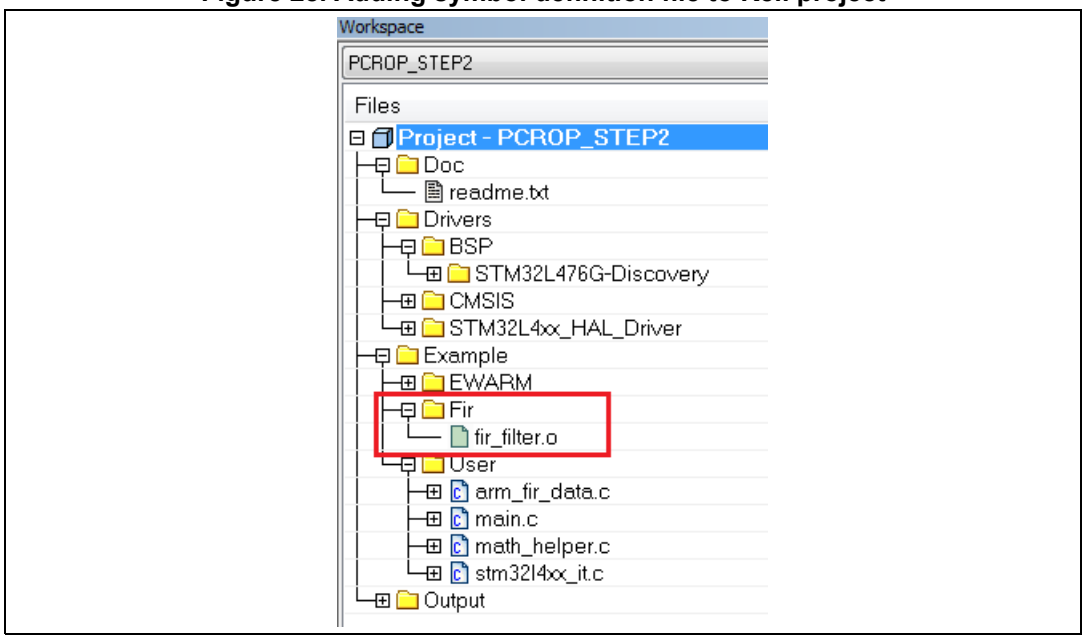
**Figure 27. Setting symbol definition file type to “Object file”**



### Adding symbol definition file in IAR project

The *fir_filter.o* file generated in *Section 3.3.7* is added as an object file to the group “Fir” as shown in *Figure 28*.

**Figure 28. Adding symbol definition file to Keil project**




## 3.4.4 Running the end user application

At this stage the IP code must be already preloaded and PCROPed in the STM32L476VG 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, user must follow the steps described below:

1. Open project located in Step2-ST_Customer_level_n+1 directory and choose the same toolchain used in step 1
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 reset button

   A welcome message is displayed “WELCOME PCROP STEP 2”

   Application test is running and result is checked. If test is passed, the green light toggles infinitely.

## 3.4.5 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 must prevent any read access to the IP code developed by ST customer - Level n.

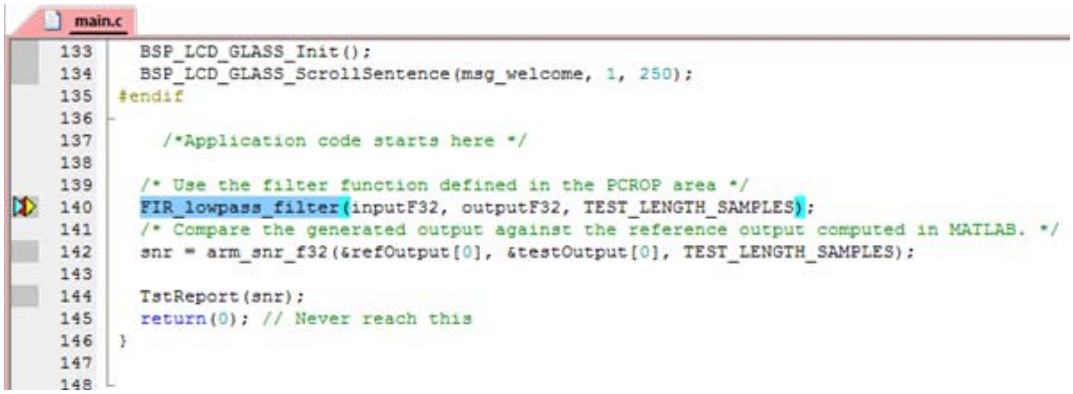
This section shows how the PCROP protects the preprogrammed IP code against reading when debugging user code. The debugging example described below is carried out on Keil IDE.

### Debugging end user application

Before debugging end user project, Step1-ST_Customer_level_n project (PCROP-IP-Code-XO workspace) must be loaded and executed to get STM32L476 with preprogrammed and PCROP-ed IP code.

To debug Step1-ST_Customer_level_n+1 project, user must follow the steps described below:

1. Open project located in Step1-ST_Customer_level_n+1 directory and choose the same toolchain used in step 1
2. Rebuild all files.
3. Click on Start/Stop Debug session to start debug.
4. Place a breakpoint on FIR_lowpass_filter (protected function).




5. Run the program by clicking on Run, the welcome message “WELCOME PCROP STEP 2” is displayed on the board.
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 29*.
7. Fill in the address field the PCROP-ed area starting address and click on Go To button as shown in *Figure 30*: as shown in *Figure 31*, the PCROP-ed IP code is unreadable.
8. Click on next step. A Flash memory operation error interrupt is generated due to Flash memory read operation request through D-Code bus. The “IT MEM” message is displayed (*Figure 32*).

Figure 29. PCROP-ed IP code Assembly reading

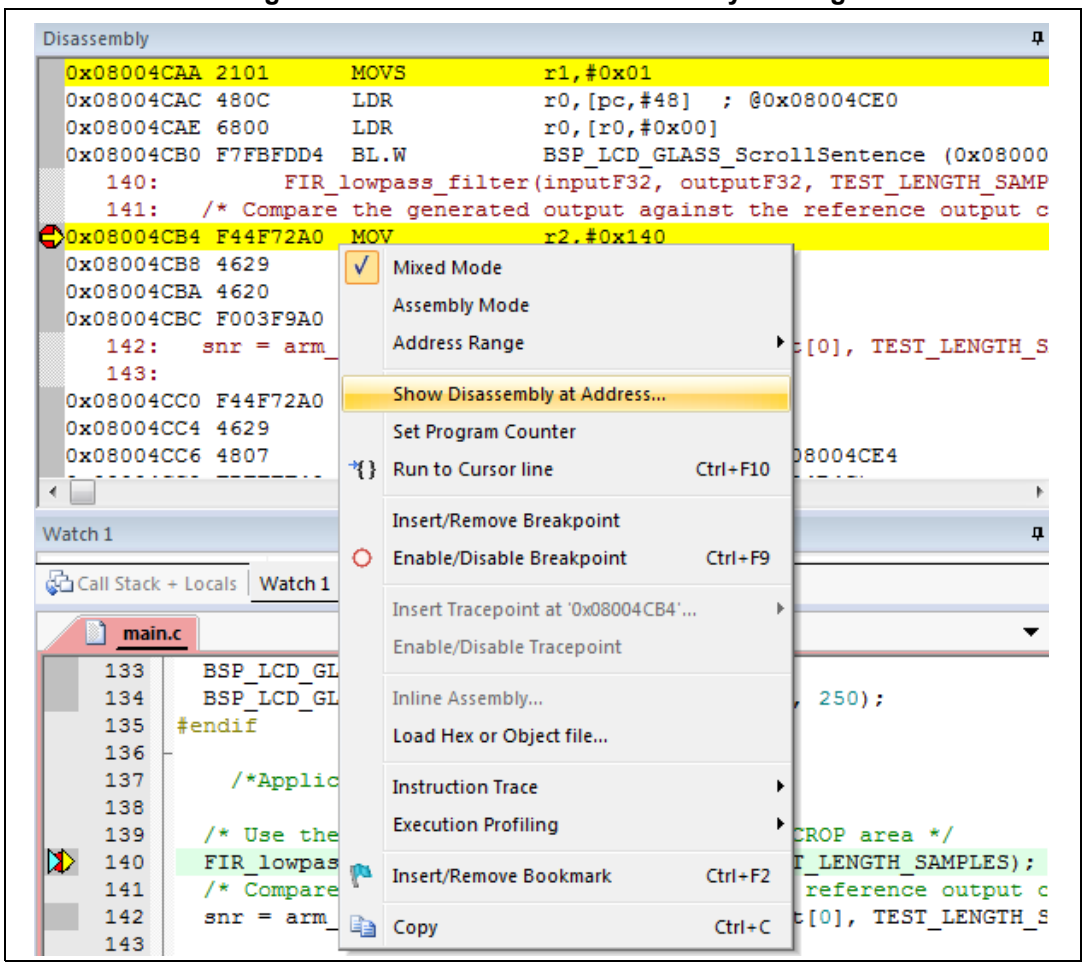



Figure 30. Filling PCROP-ed area starting address

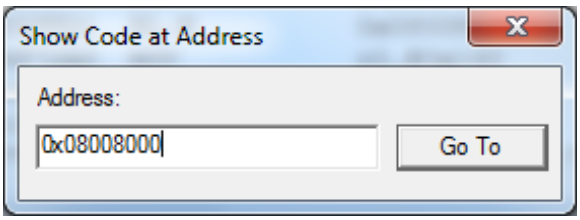


Figure 31. PCROP-ed IP code Assembly reading

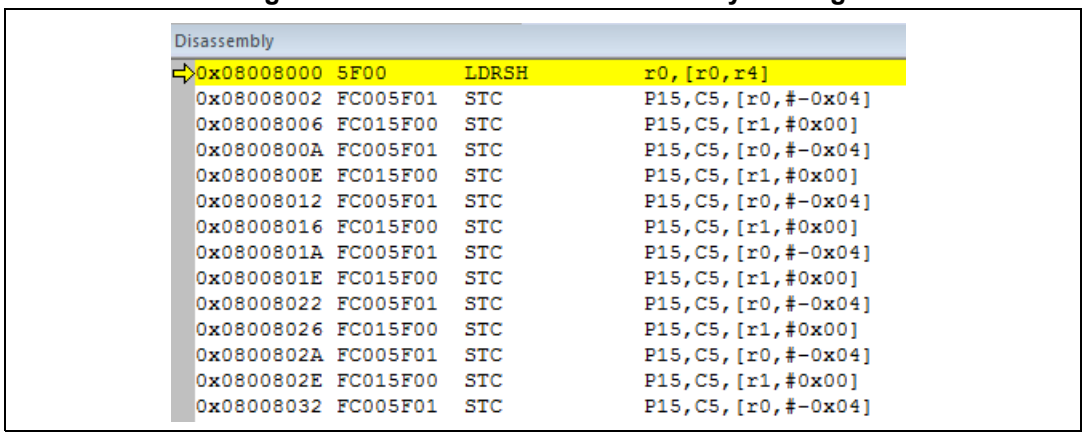
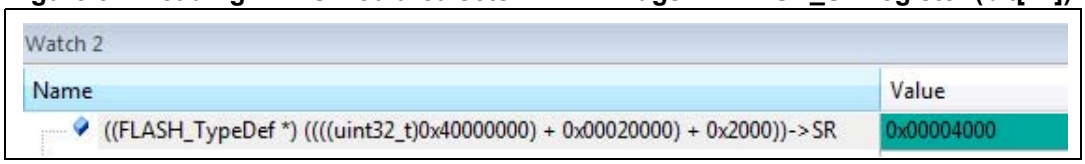



Figure 32. Reading PCROP-ed area sets RDERR flags in FLASH_SR register (bit[14]).



# 4 Conclusion

Microcontrollers of the STM32L4, STM32L4+, STM32G4 and STM32WB 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 can be used.

# Revision history

**Table 3. Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 26-Oct-2015 | 1 | Initial release. |
| 08-Sep-2017 | 2 | Added references to RM0392 and RM0394 on the cover page. |
| 05-Sep-2019 | 3 | Introduced STM32G4 Series.<br>Updated:<br>Document title, *Introduction*, *Section 1: General information*, *Section 2.1: Read-out protection (RDP)*, *Section 2.1.2: Read protection Level 1*, *Section 2.2.1: Flash memory write protection*, *Section 2.2.2: SRAM2 CCM-SRAM write protection*, *Section 2.3.2: How to enable PCROP protection?*, *Section 2.3.3: How to disable PCROP protection?*, *Section 2.4.1: Firewall*, *Section 3: PCROP example*, *Section 4: Conclusion* and *Table 1: Access status versus protection level and execution modes*.<br>Added *Section 2.4.2: Securable memory area for the STM32G4 Series*. |
| 18-Oct-2019 | 4 | Introduced STM32L4+ Series.<br>Updated:<br>Document title, *Introduction*, *Section 1: General information*, *Section 2.1: Read-out protection (RDP)*, *Section 2.1.2: Read protection Level 1*, *Section 2.2.1: Flash memory write protection*, *Section 2.2.2: SRAM2 CCM-SRAM write protection*, *Section 2.3.2: How to enable PCROP protection?*, *Section 2.4.1: Firewall*, *Section 3: PCROP example* and *Section 4: Conclusion*. |
| 17-Nov-2020 | 5 | Introduced STM32WB Series.<br>Updated:<br>Document title, *Introduction*, *Section 2.1.2: Read protection Level 1*, *Section 2.2.1: Flash memory write protection*, *Section 2.2.2: SRAM2 CCM-SRAM write protection*, *Section 2.3.2: How to enable PCROP protection?*, *Section 3.1.2: Software requirements*, *Section 3.3.2: Generating an execute-only IP code*, *Section 3.3.3: Placing IP code and data segments in Flash memory*, *Section 3.3.4: Write protection of constants*, *Section 3.3.5: Protecting IP code*, *Section 3.3.6: Executing PCROPed IP code*, *Figure 20: Enabling PCROP with STM32CubeProgrammer* and *Figure 21: Activating PCROP with STM32 STM32CubeProgrammer*.<br>Added *STM32Cube IDE: "Assume loading data from flash is slower than fetching"*, *STM32Cube IDE* and *Symbol definition file with STM32CubeIDE*.<br>Replaced former *Use STM32-Link Utility* and *Activating PCROP using STM32-Link Utility* with, respectively, *Use STM32CubeProgrammer* and *Activating PCROP using STM32CubeProgrammer*.<br>Updated Minor text edits across the whole document. |
| 02-Jul-2021 | 6 | Updated *Introduction*. |
| 05-Feb-2026 | 7 | Updated document title.<br>Added *Chapter 1: General information* |

**IMPORTANT NOTICE – 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.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The 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 acknowledgment.

The 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 the 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.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers' market segment, the purchasers shall contact ST for more information.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. 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.

© 2026 STMicroelectronics – All rights reserved