AN4758
Application note

Proprietary code read-out protection on microcontrollers of the STM32L4 Series

Introduction

Software providers are developing complex middleware solutions (Intellectual Propriety code), protecting these IP Codes is a high importance issue concerning the microcontrollers.

In order to respond to this important requirement, microcontrollers of the STM32L4 Series 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 these Flash memory protection features, focusing on the PCROP technique, and providing a basic example of it. Firewall protection is described in a dedicated application note (AN4729).

The X-CUBE-PCROP firmware package is delivered with this document containing 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 product datasheets and the following reference manuals:

- RM0351 (STM32L4x5xx, STM32L4x6xx)
- RM0394 (STM32L43xxx, STM32L44xxx, STM32L45xxx, STM32L46xxx)
- RM0392 (STM32L4x1xx)

These documents are available at *www.st.com*.

Contents

List of tables

List of figures

1 Memory protection description

Microcontrollers of the STM32L4 Series feature several mechanisms for read and write protection of the full memory or of specific segments. Read protection is used to protect code from dumping by external access (SW IP protection) while Write protection is needed to protect code or data from unwanted erasing. In addition to Flash memory, STM32L4 Series now extends these protections to the SRAM2 segment.

The STM32L4xx MCUs introduce also a new Firewall mechanism to establish trusted execution areas in memory.

1.1 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.

In the STM32L4 Series, the read-out protection applies to:

- the main Flash memory;
- the backup registers in the Real-Time Clock (RTC);
- the SRAM2 (new feature);
- the option bytes (Level 2 only).

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

1.1.1 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.

1.1.2 Read protection Level 1

When the read protection Level 1 is activated, no access (read, erase, and program) to Flash memory or to SRAM2 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 will generate a bus error.

When booting from Flash memory, however, accesses to both the Flash memory and to the SRAM2 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 and backup registers are reset as well.

1.1.3 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's impossible to go back to protection Level 1 or 0.

Table 1 summarizes read access permission depending on 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	Level 1	Yes		NA(1)	No		
	Level 2	Yes		NA(1)	NA(1)		

1. NA: Not available

1.1.4 Internal Flash memory content updating on RDP protected STM32

When Flash RDP protection is activated (Level 1 or Level 2), internal Flash memory content cannot any longer be updated 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 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 please refer to application note AN3965, available at *www.st.com*.

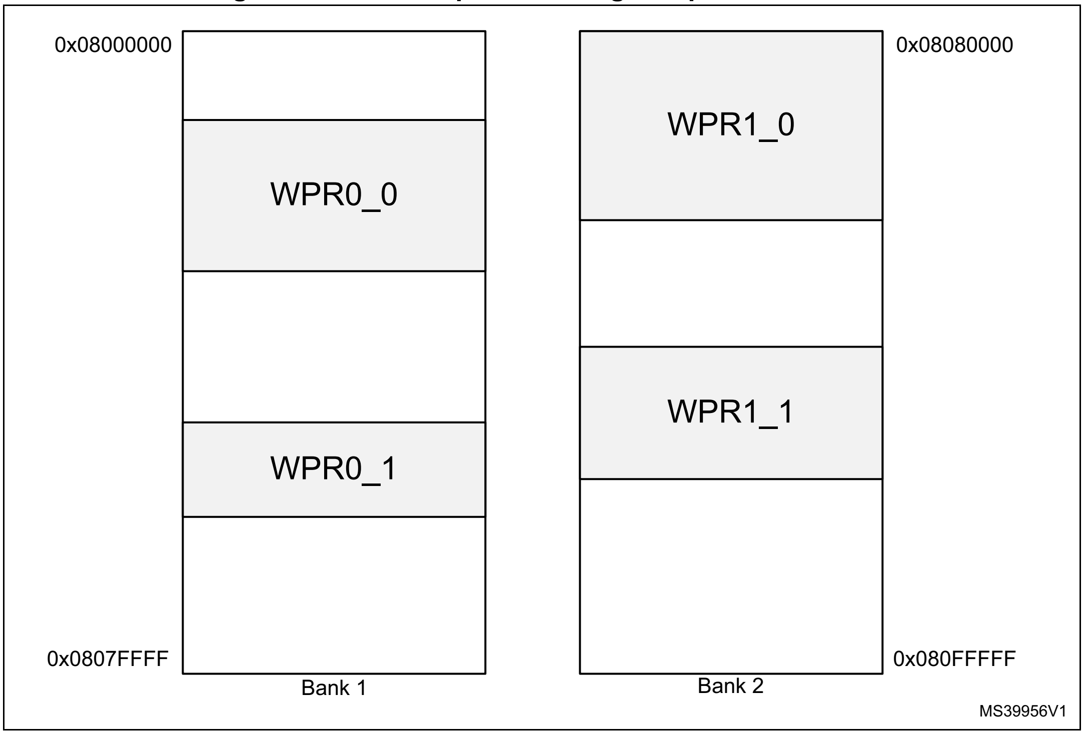
1.2 Write protection

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

1.2.1 Flash memory write protection

In STM32L4 microcontrollers, two write protected (WRP) regions can be defined in each bank, with 2-Kbytes pages granularity (gray areas in *Figure 1*).

Figure 1. Two write protected regions per Flash bank



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.

For 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 STM32 ST-Link Utility software and debug interfaces.

1.2.2 SRAM2 write protection

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

SRAM2 Erase

Automatic erase of SRAM2 can be triggered by:

- regressing from RDP Level 1 to Level 0;
- setting the SRAM2ER bit in system configuration control status register;
- after a system reset in case the SRAM2_RST option bit is active.

1.3 Proprietary code readout protection (PCROP)

1.3.1 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.

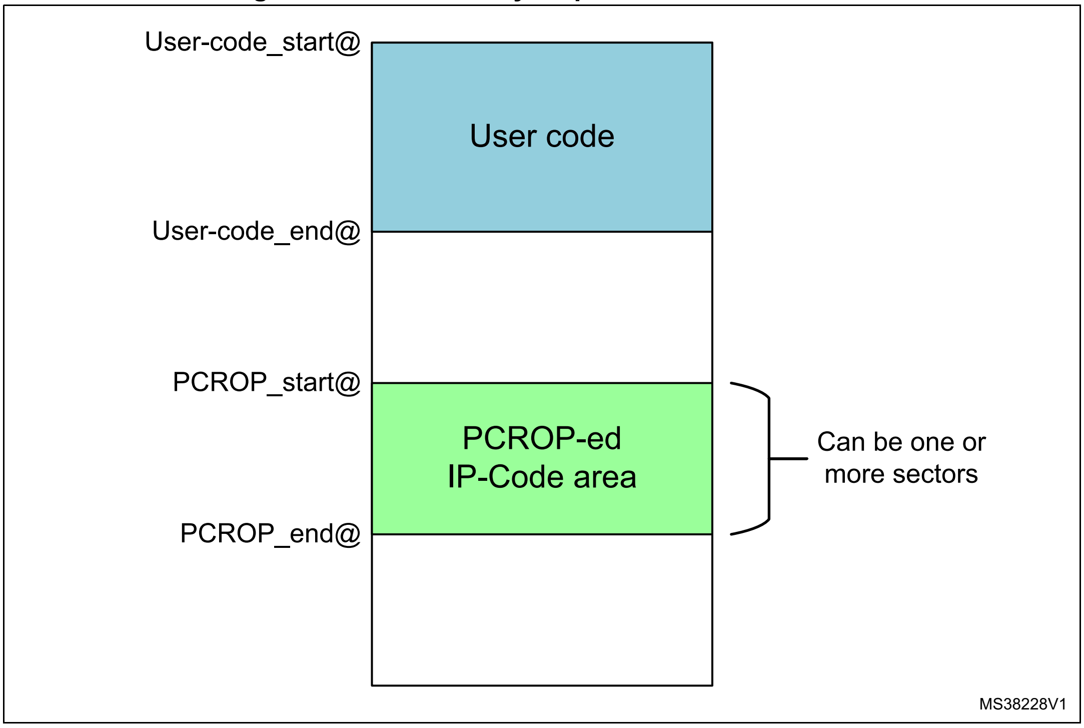
Any read or write request generates a Read or Write protection error:

- 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 for end-user development.

Figure 2. Flash memory map with PCROP-ed area



1.3.2 How to enable PCROP protection?

One area per bank can be selected, with double word (64-bit) 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, please refer to the example PCROP_ENABLE() function described in the provided FW package (Step1-ST_Customer_level_n project main.c file).

1.3.3 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).

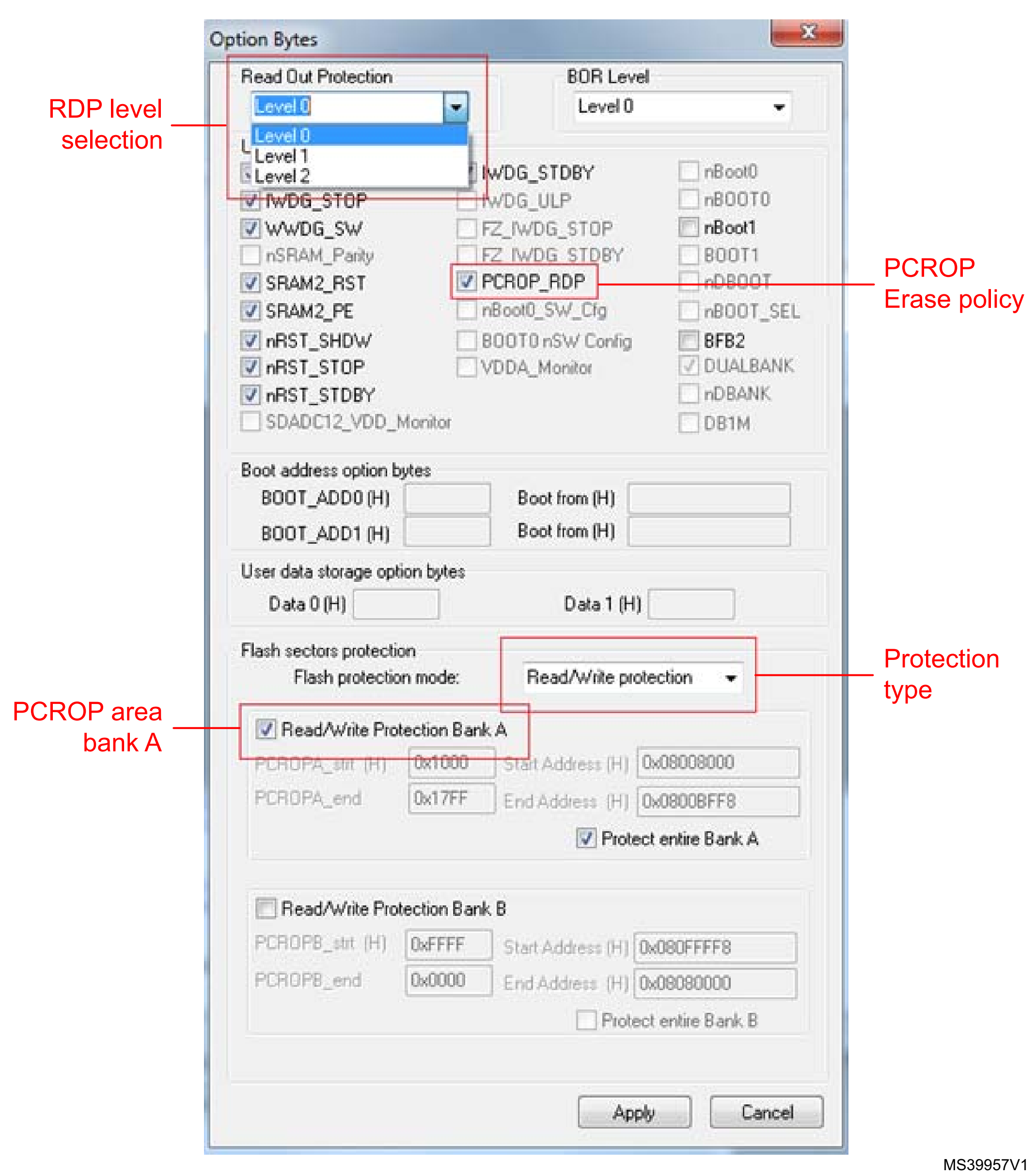
If PCROP_RDP is set, a mass erase of the Flash main memory is performed. The backup registers in the RTC and the SRAM2 are also erased.

If the bit PCROP_RDP is cleared, the full mass erase is replaced by a partial mass erase that is 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 STM32-Link Utility

During application development user may need to disable PCROP or global RDP protection using another alternative than embedded Flash memory code. 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. *Figure 3* shows how to modify option bytes.

Figure 3. User interface for modification of option bytes



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

1.3.4 PCROP-ed IP code compilation

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

Literal pools

Compilation option shall be used to avoid the use of literal pools that would be placed in the code section otherwise. Compilation option will be further detailed in *Section 2*.

Constant data

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

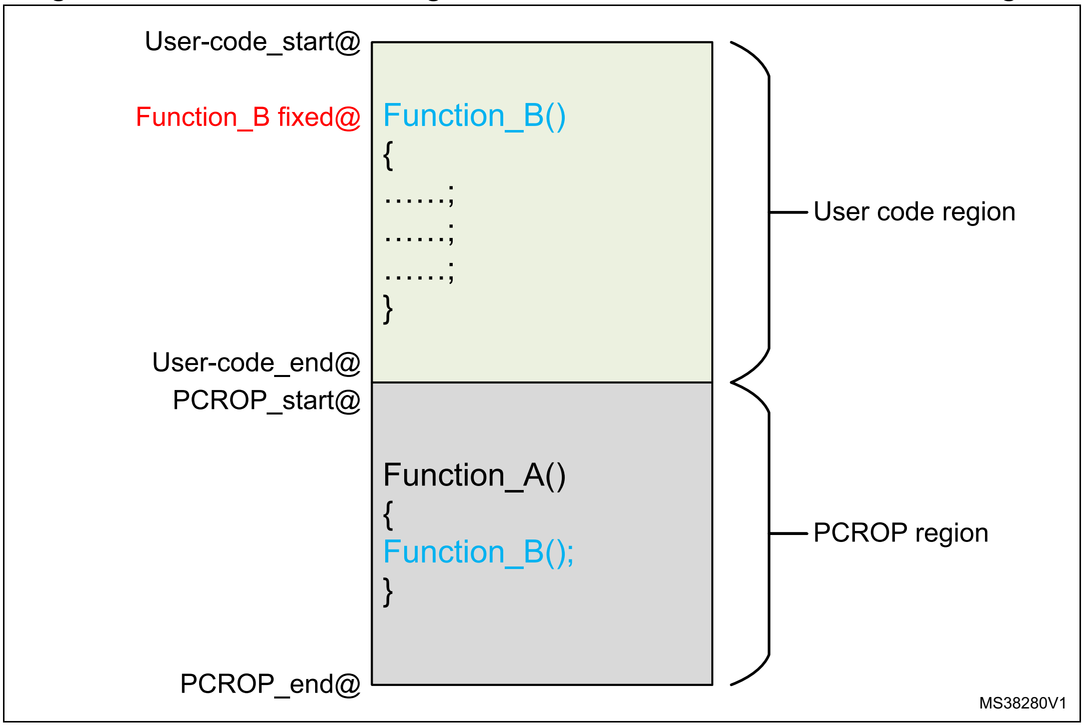
1.3.5 PCROP-ed IP code dependency

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

Figure 4. PCROP-ed code calling a function located outside the PCROP-ed region



1.4 Firewall

The Firewall is a new protection feature introduced in the STM32L4 Series. Associated to the other Flash protections, it provides an increased level of protection for part of code and data coming from third party.

The user may want to protect some sensitive algorithms using confidential associated data (cryptographic keys, secured algorithms and associated variables...) somewhere in the memory mapping. He may want to manage exactly when the user code is going to access to 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 dedicated application notes AN4729 for mode detailed description of the feature.

2 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.

2.1 Requirements

2.1.1 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.

2.1.2 Software requirements

The following software tools are required:

- IAR Embedded Workbench (v7.40.3) or Keil® µvision IDE(v5.14.0)
- STM32 STLink Utility (v3.7.0) is required mainly for enabling or disabling PCROP protection.

2.2 Description

2.2.1 Scenario overview

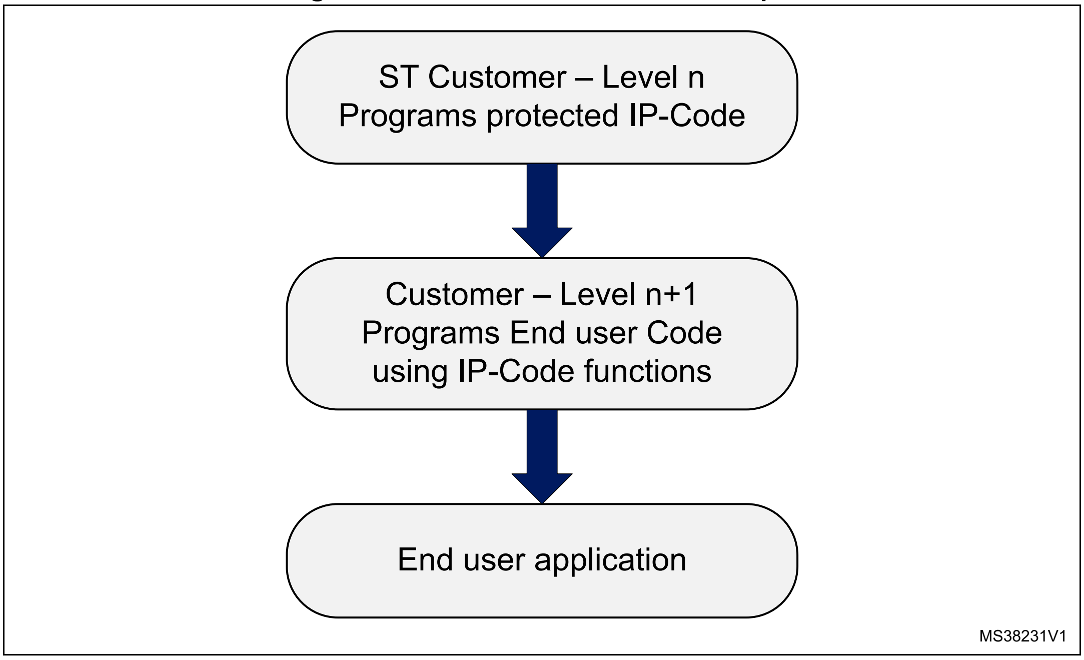
This example describes a use case where an ST Customer Level n provides preprogrammed STM32L476VG MCUs with a critical IP code to an ST Customer level n+1. The IP code has to be protected by activating PCROP allowing ST Customer level n+1 to use its functions (without the ability to read or modify it) to program the End User Application.

ST Customer level n should then 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 functions' symbols.

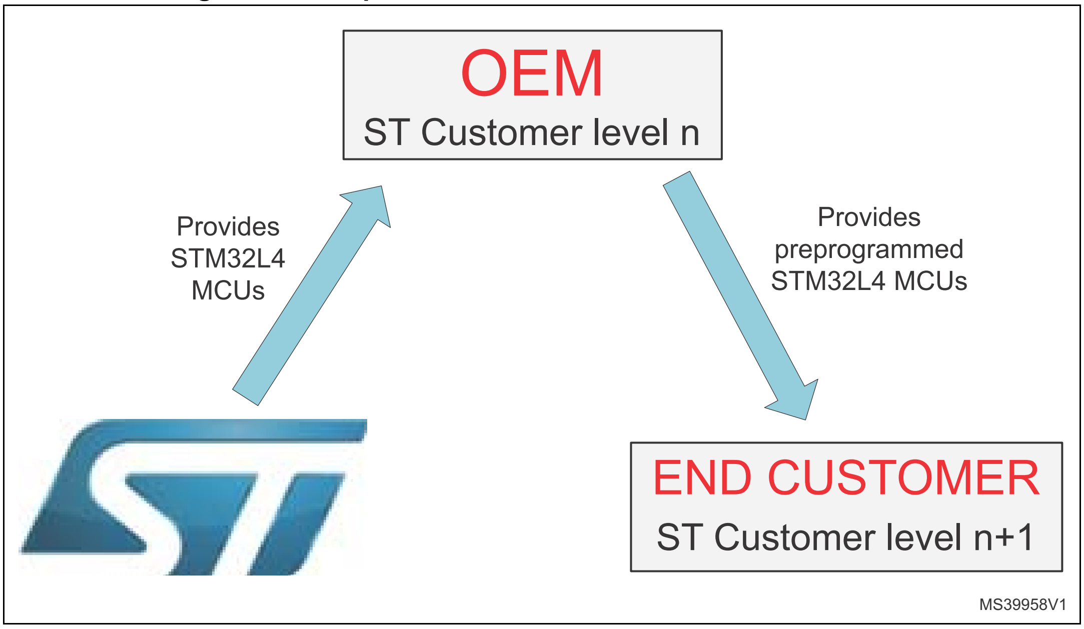
The described use case is schematized in *Figure 5*.

Figure 5. STM32L4 PCROP flow example



An OEM (Original Equipment Manufacturer) can be the ST Customer level n using STM32L4 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 6*.

Figure 6. Example of an ST Customer level n and level n+1



2.2.2 PCROP-ed IP code: FIR low pass filter

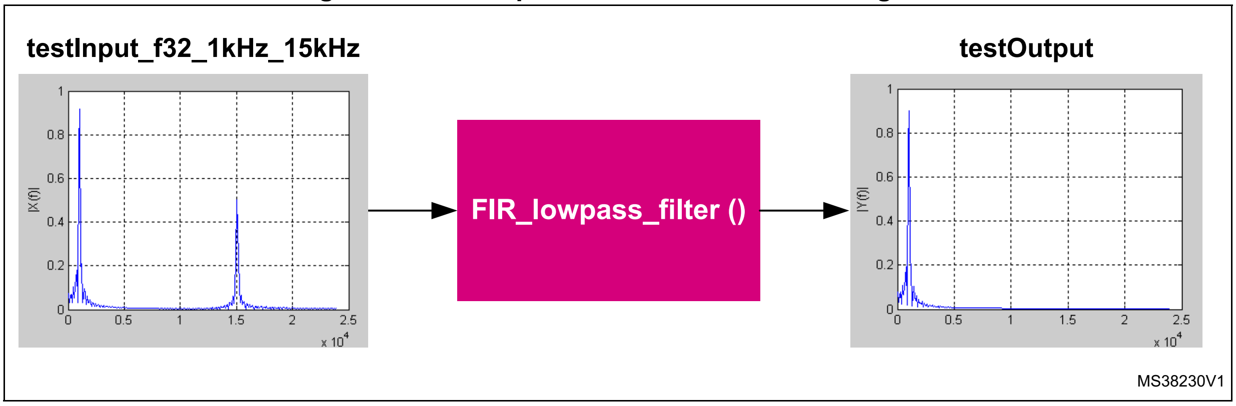
We can take as an example FIR low pass filter algorithm from CMSIS-DSP library as the IP code to be protected. Here we will focus only on detailing how to protect and call this IP code example without providing any details on the functions it-selves.

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 its preconfigured cutoff frequency 6 KHz) eliminates the 15 KHz signal leaving the 1 KHz sine wave at the output.

Figure 7 shows the FIR low pass filter block diagram.

Figure 7. 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 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.

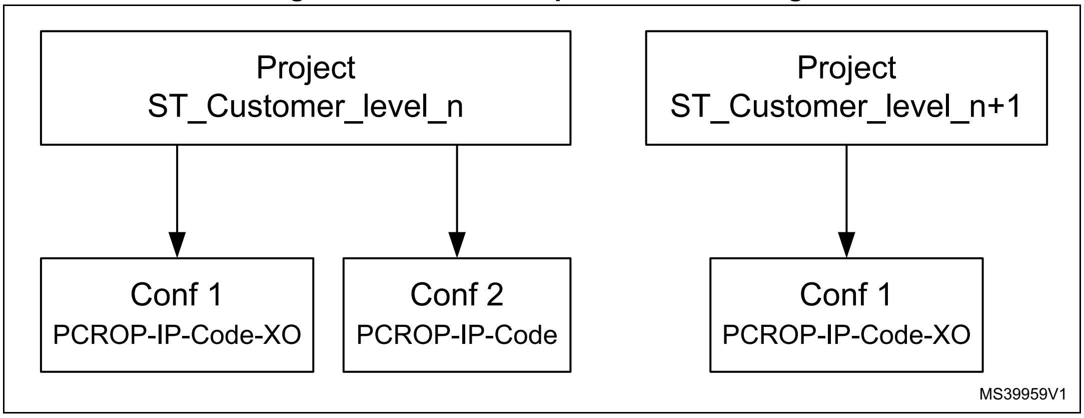
The FPU and DSP embedded in STM32L4 microcontrollers are used for signal processing and floating point calculation to output the correct signal.

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

2.2.3 Software settings

This application note describes two projects (*Figure 8*).

Figure 8. 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 having a preprogrammed STM32L476VG with a PCROP-ed IP code can create its own end user application using these protected IP code functions.

2.3 Development step 1: ST Customer level n

At this stage the ST Customer level n will:

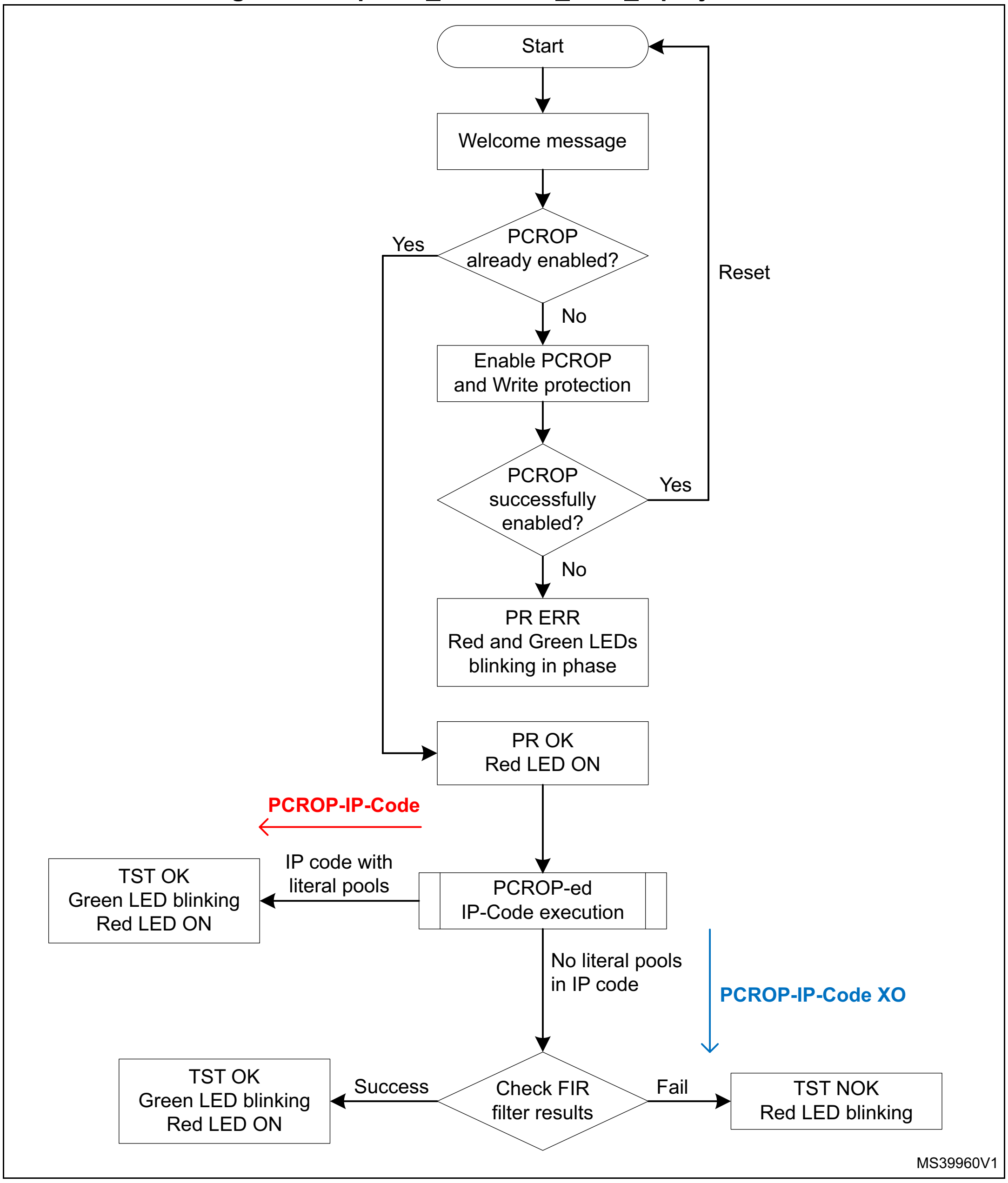
- generate an execute-only IP code;
- place IP code and data segments at specified location in Flash memory;
- write-protect data segment;
- protect the IP code area with PCROP;
- execute the IP code by calling its functions in main code;
- create header file and generate symbol definition file to be used in STEP2-ST_Customer_level_n+1 project.

2.3.1 Project flow

Figure 9 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 didn't contain any literal pool, FIR filter algorithm run 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 9. Step1-ST_Customer_level_n project flow

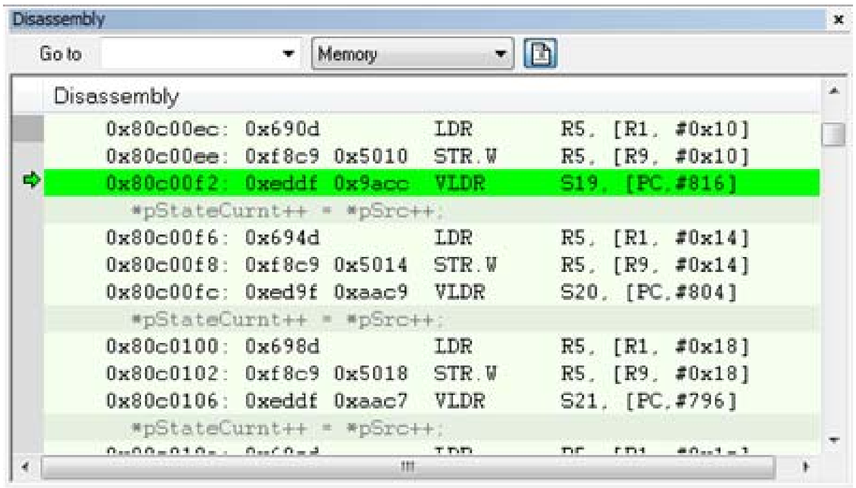


2.3.2 Generating an execute-only IP code

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.

Figure 10 shows an assembler code containing literal pools, where the instruction has the form of VLDR <variable>, [PC + <offset>].

Figure 10. 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.

The option have to 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 11*) choose Options for Group ‘User/Fir:

In the following window check the option Execute-only Code, as indicated in *Figure 12*: *-execute_only* option is added in compiler control string field:

Figure 11. Accessing the FIR filter options

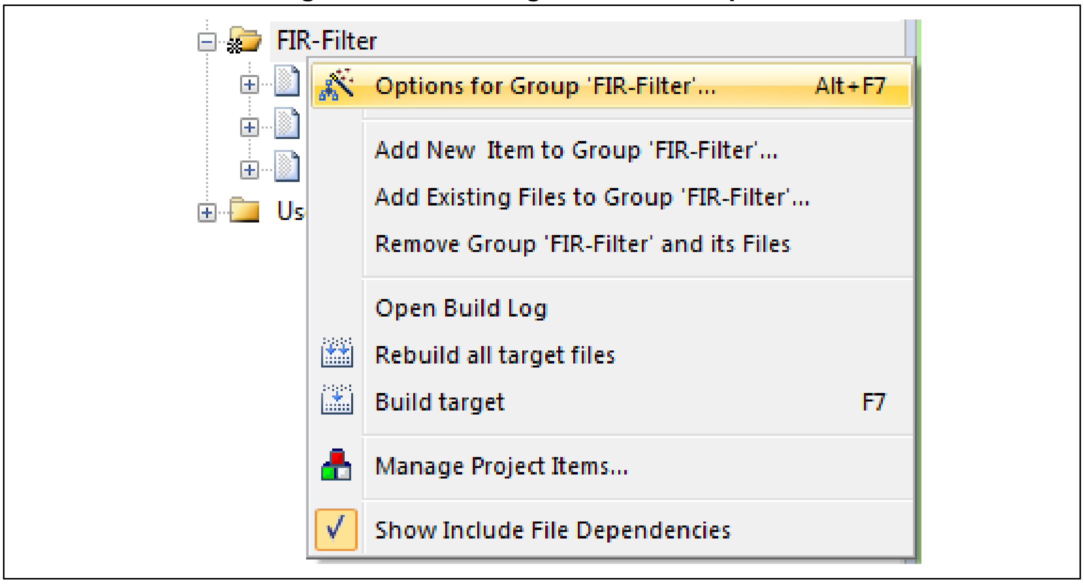
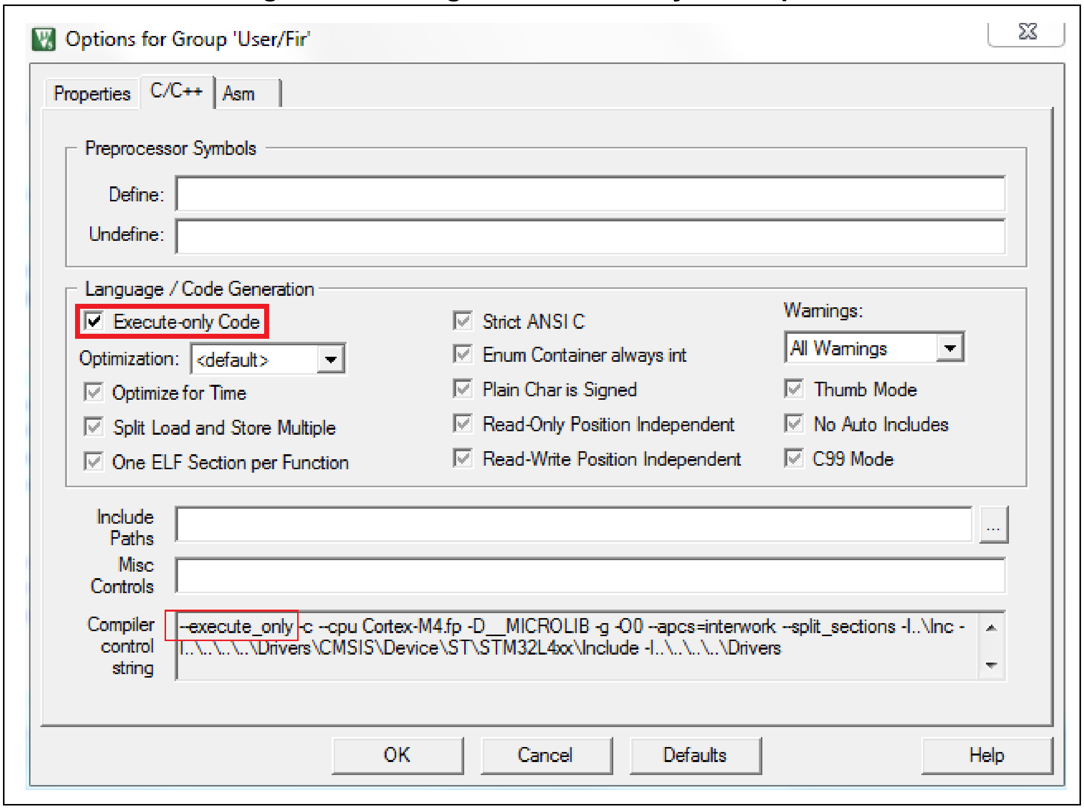


Figure 12. Setting the Execute-Only code option



The scatter file that will define the memory region placement of the IP code will set the access type attribute to +XO. Refer to *Section 2.3.3*.

IAR: No data reads in code memory

Option “No data reads in code memory” must be used in IAR to generate a code without literal pools. To activate this option, user must right click on the file group ‘Fir’ containing IP code source files (see *Figure 13*), then select Options, and then, in the following window check the option “*No data reads in code memory*”, as indicated in *Figure 14*.

Figure 13. Accessing to FIR-Filter options

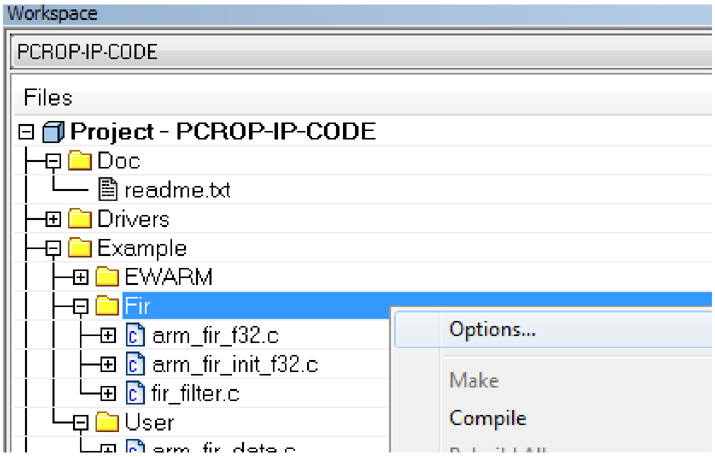
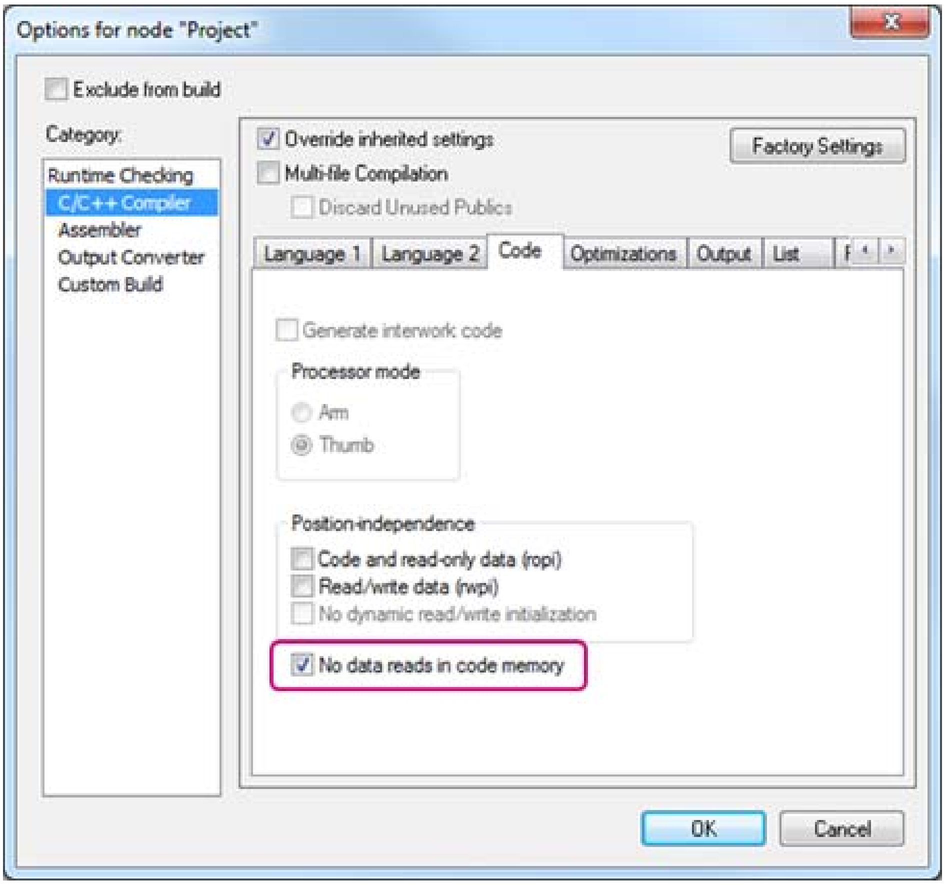


Figure 14. Setting option “No data reads in code memory”



2.3.3 Placing IP code and data segments in Flash memory

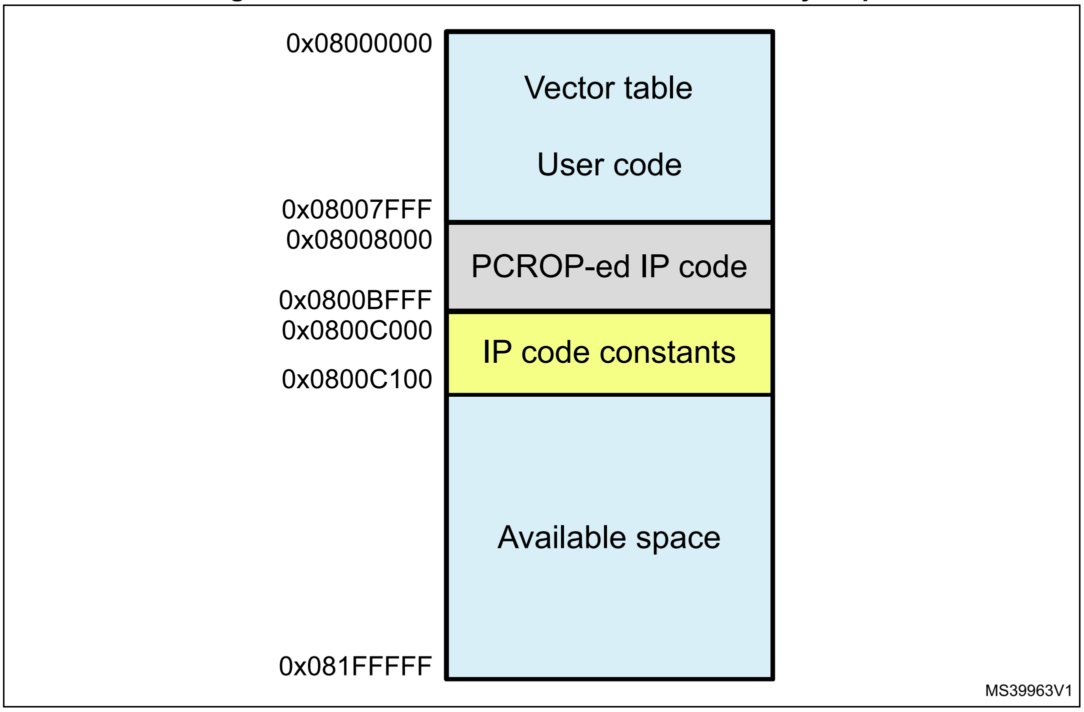
In this application note example, two specifics memory sections will be defined for IP code segment and its associated constant data (FIR filter coefficients).

- IP code segment will be 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 will be located at the beginning

The resulting Flash memory map is shown in *Figure 15*.

Figure 15. STM32L476VG internal Flash memory map



Memory placement are handled by scatter file for Keil® IDE and Linker configuration file (ICF) in IAR.

Keil® scatter file

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

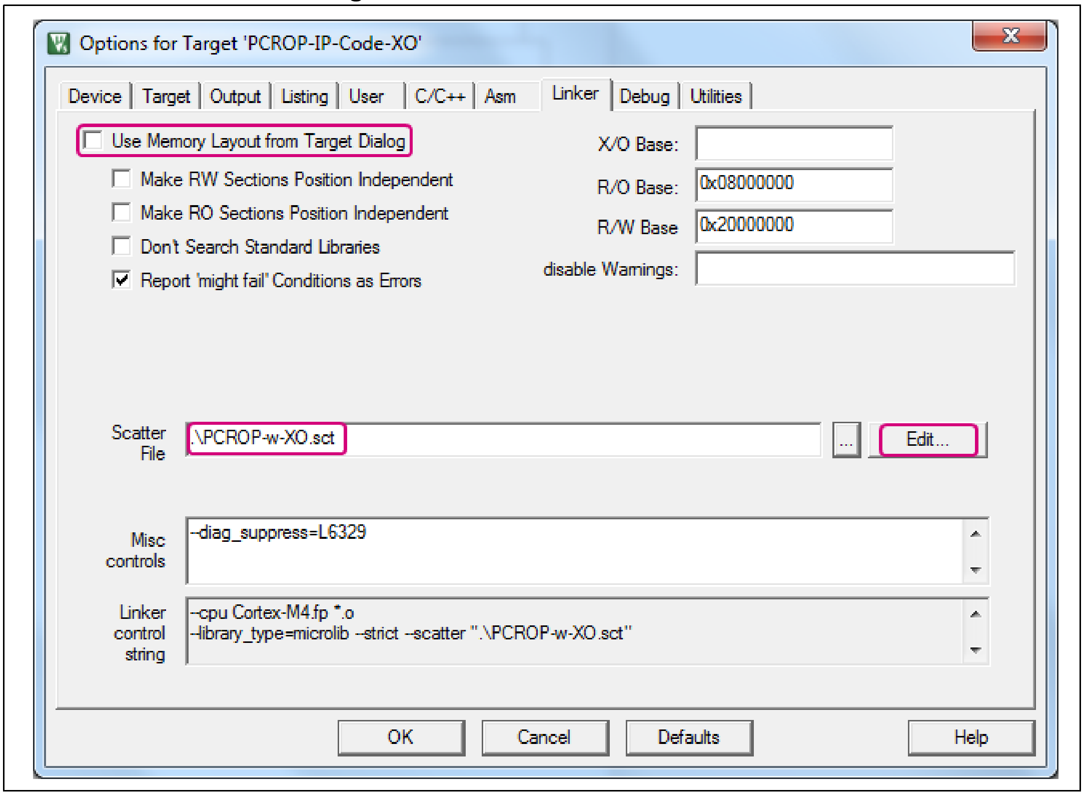
- LR_PCROP is the IP code region with execute-only access attribute
- LR_DATA is the constant data memory region used by the IP code. A new section is defined, namely “fir_coeff_section”.

```
; *************************************************************
; *** 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 16*.

Figure 16. 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 };
```

Explicit data placement

Constant data will be mapped on 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"))) = {…};
```

2.3.4 Write protection of constants

User should take into account that IP code's constants can be deleted or modified by ST Customer level n + 1 and functions could become useless, so it's recommended to protect these constants in from unwanted write/erase.

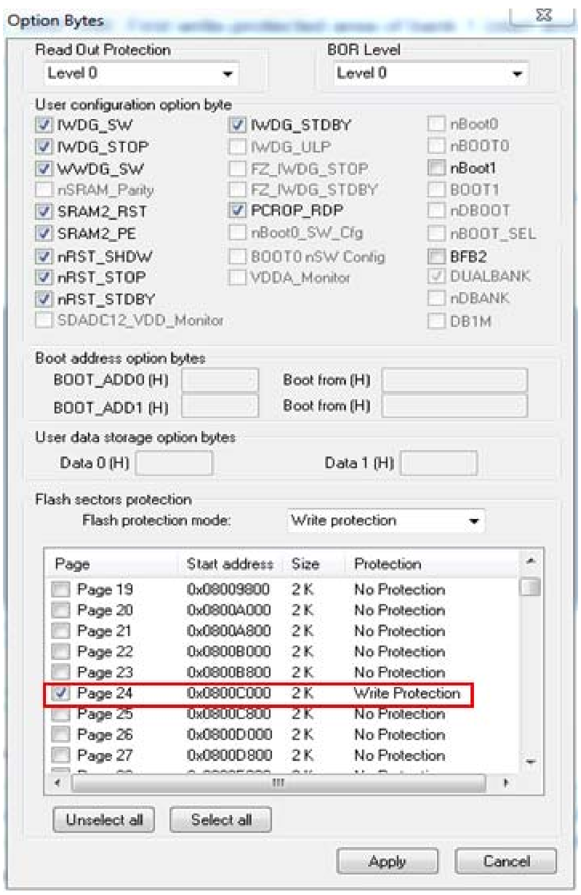
Write protection area is set through the following option bytes registers:

- 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 2Kbytes so a single page will be 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 it can be set by using STLink-Utility as shown in *Figure 17*.

Figure 17. Enabling PCROP with STM32 STLink Utility



2.3.5 Protecting IP code

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

Activating PCROP using FW

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

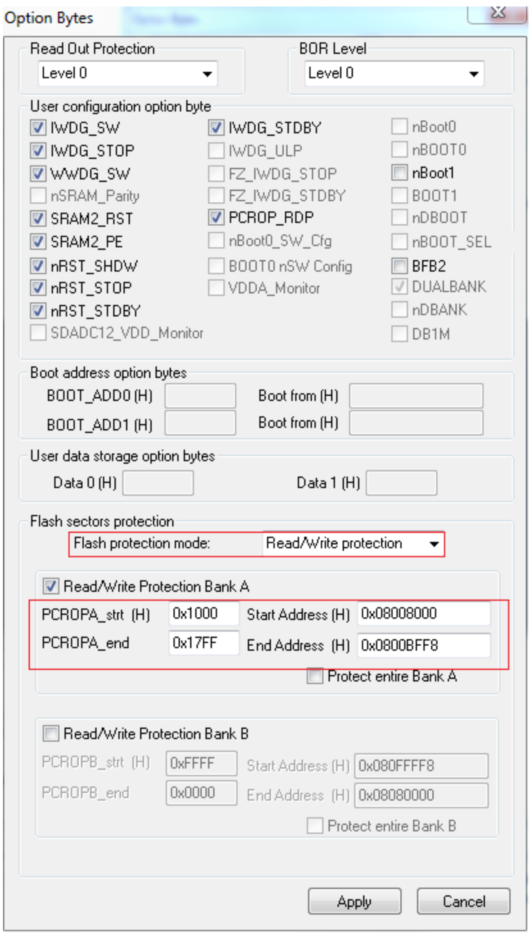
This function is defined in the Step1-ST_Customer_level_n project main.c file.

Activating PCROP using STM32 STLink Utility

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

- 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, enable protection on bank A and specify start and end address of area as shown in *Figure 18*.
- Click on Apply button.

Figure 18. Activating PCROP with STM32 STLink Utility



2.3.6 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.

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 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 STM32 STLink Utility; 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.

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

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 STM32 STLink Utility, then load the code;
   b) Press the reset button to start the test:

      awelcome 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 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 test is passed (“TST OK “message 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 could not 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.

2.3.7 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

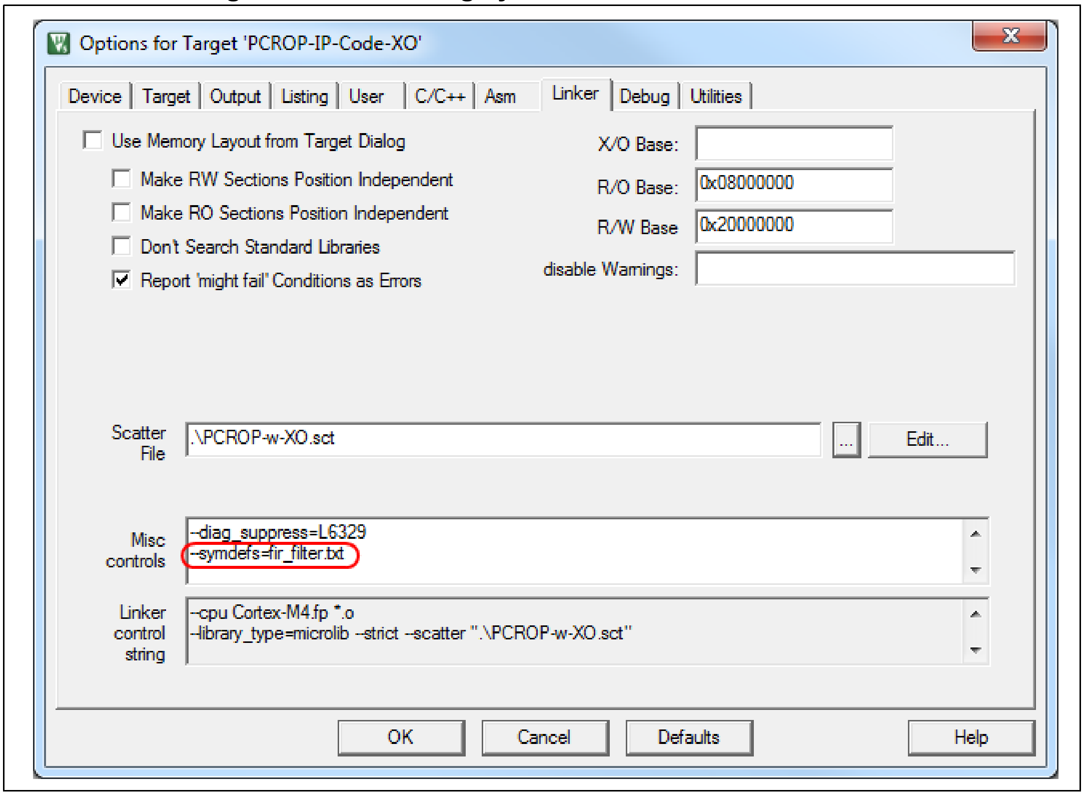
The header file to be provided to ST Customer level n+1 contains the definitions of IP code functions to be used. In this example it’s the *fir_filter.h* file included in the *main.c* file.

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 19*). Rebuild the project.

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

Figure 19. 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 should be removed.

The resulting symbol definition file will then be:

```
#<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:

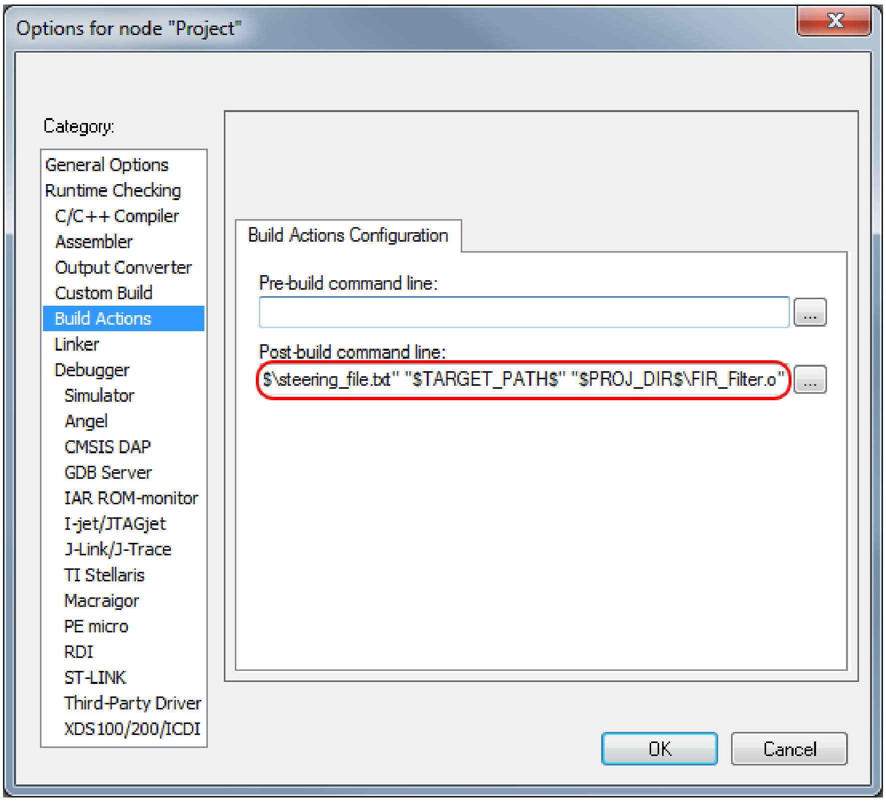
show arm_fir_f32

show arm_fir_init_f32

show FIR_lowpass_filter

The output file *fir_filter.o* will be used by End User by adding it to project.

Figure 20. Generating symbol definition file with IAR



2.4 Development step 2: ST Customer level n+1

ST Customer level n+1 have 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 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.

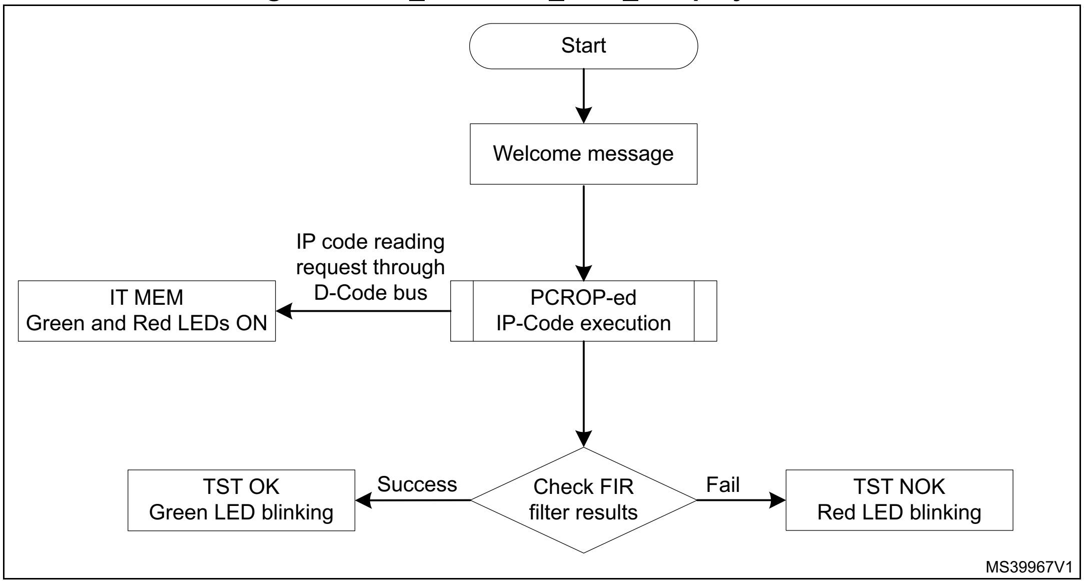
2.4.1 Project flow

Figure 21 illustrates the ST_Customer_level_n+1 project flow. The code shall execute with success unless a read operation occurs on the protected area. This read operation can be a debug access or a memory dump.

The interrupt is signaled by the specific message "IT MEM".

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

Figure 21. ST_Customer_level_n+1 project flow



2.4.2 Creating End User Project

Protected code and write-protected data segments are located at known position of memory map (see *Figure 15*), hence the user must take care when implementing linker file to avoid placing any 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.

2.4.3 Including header file and adding symbol definition file

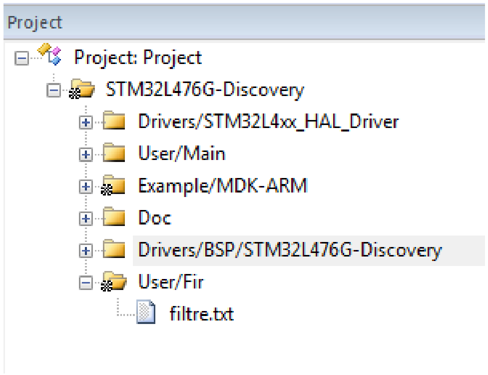
The header file *fir_filtre.h* is included in *main.c* file so that user can call FIR filter functions. The file shall be added in the include search path of the compiler settings.

The symbol definition file provided from ST Customer level n should be added as a classic source file which is necessary to link properly. User must follow the method depending on the chosen IDE.

Adding symbol definition file in Keil® project

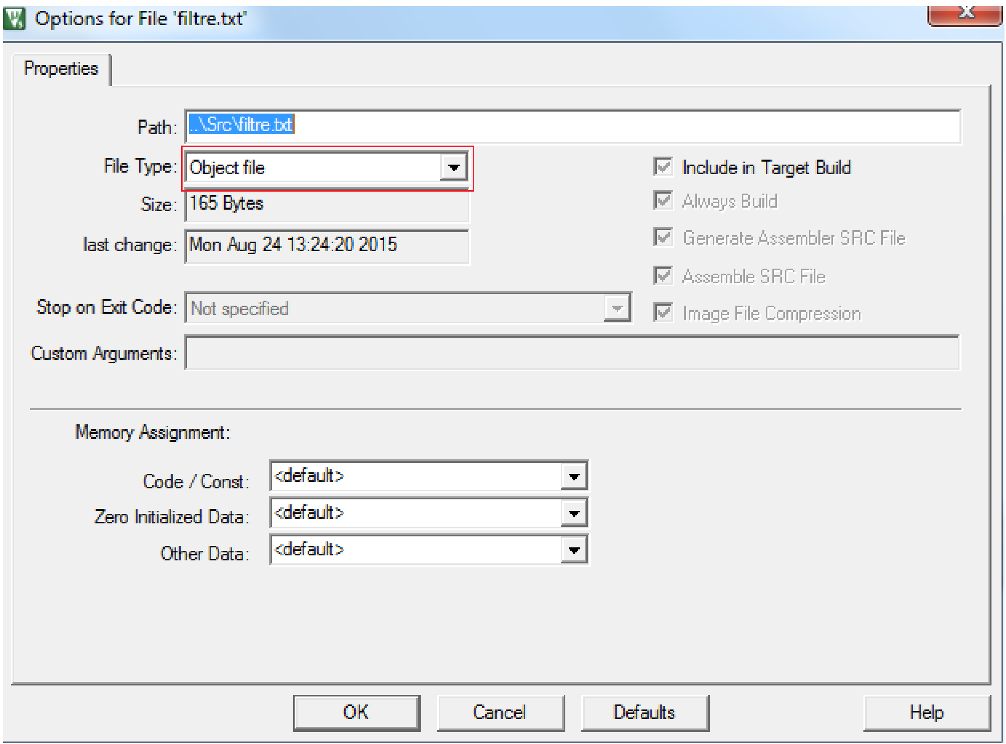
The symbol definition file described in *Section 2.3.7* and named *filtre.txt* in this example is added to User/Fir group as described in *Figure 22*.

Figure 22. Adding symbol definition file to Keil® project



The added filtre.txt file's type must be changed to Object file (see *Figure 23*) instead of text document file.

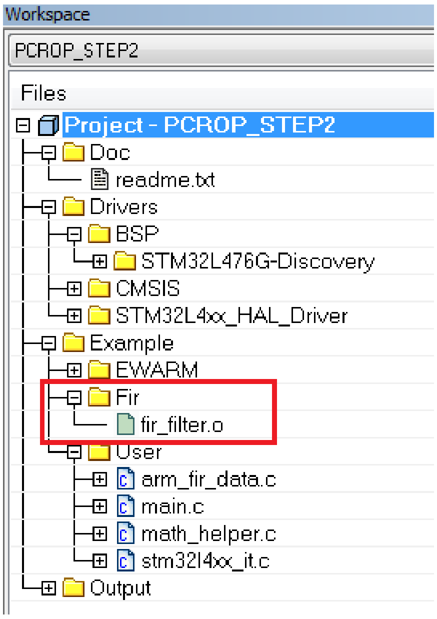
Figure 23. Setting symbol definition file type to "Object file"



Adding symbol definition file in IAR project

The *fir_filter.o* file generated in *Section 2.3.7* is added as an object file to the group “Fir” as shown in *Figure 24*.

Figure 24. Adding symbol definition file to Keil project



2.4.4 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.

2.4.5 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 should prevent any read access to the IP code developed by ST Customer level n.

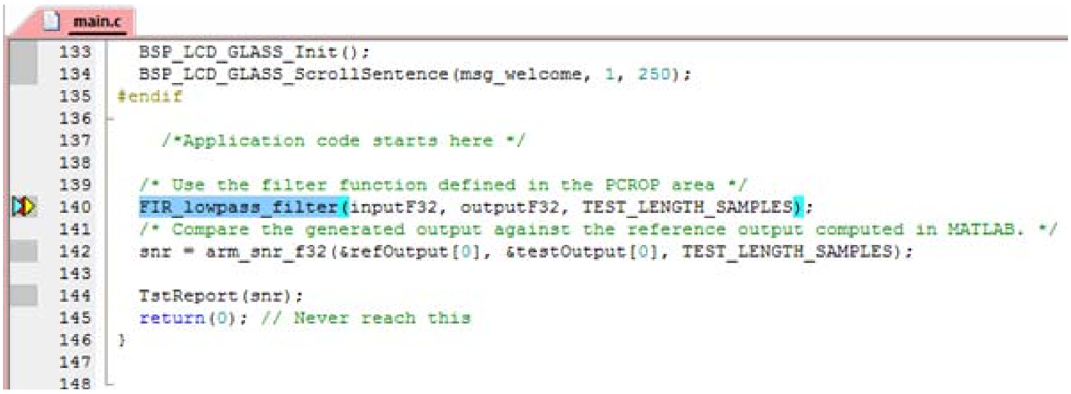
In this section we will show how the PCROP protects the preprogrammed IP code against reading when debugging user code. The debugging example described below is carried out on Keil® IDE.

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 in order 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 25*.
7. Fill in the address field the PCROP-ed area starting address and click on Go To button as shown in *Figure 26*: as shown in *Figure 27*, 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 28*).

Figure 25. PCROP-ed IP code Assembly reading

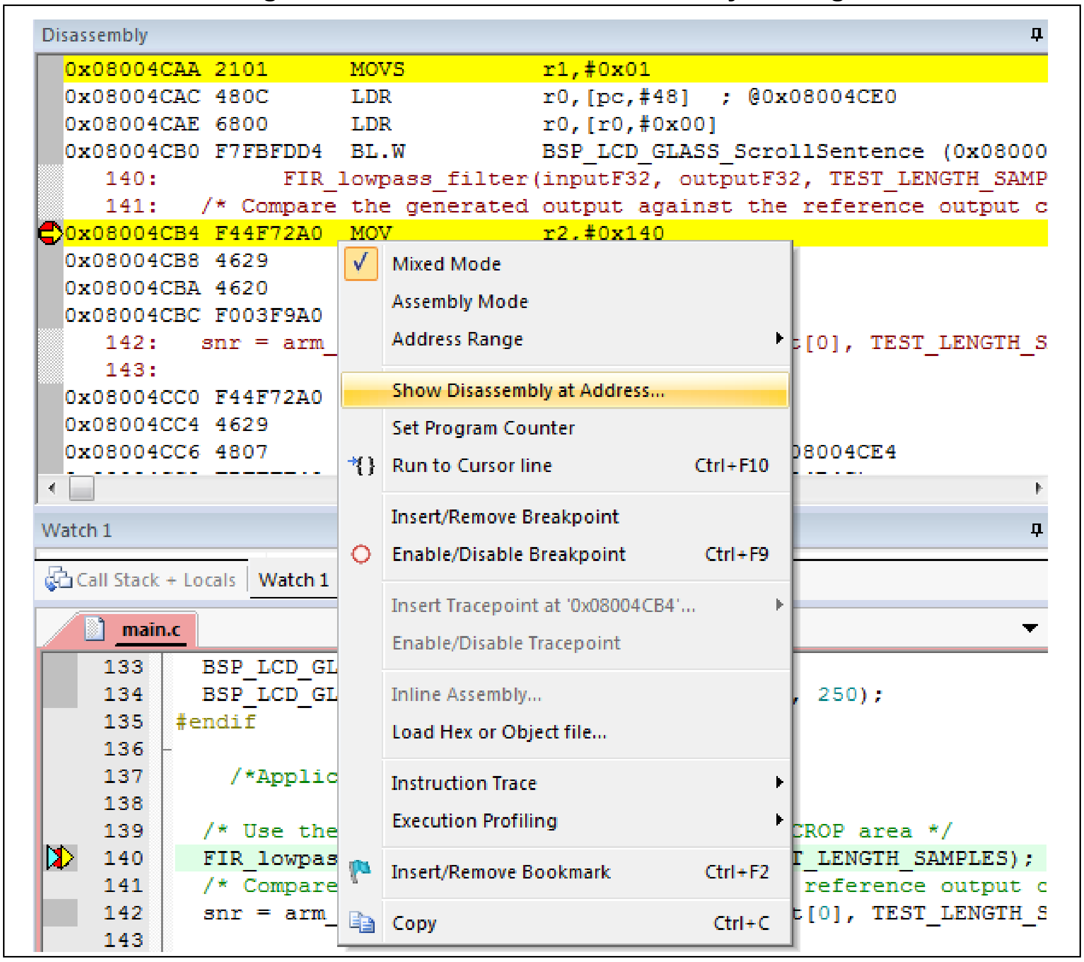


Figure 26. Filling PCROP-ed area starting address

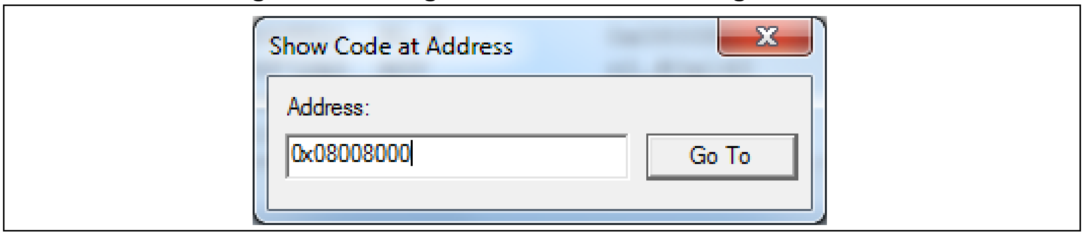


Figure 27. PCROP-ed IP code Assembly reading

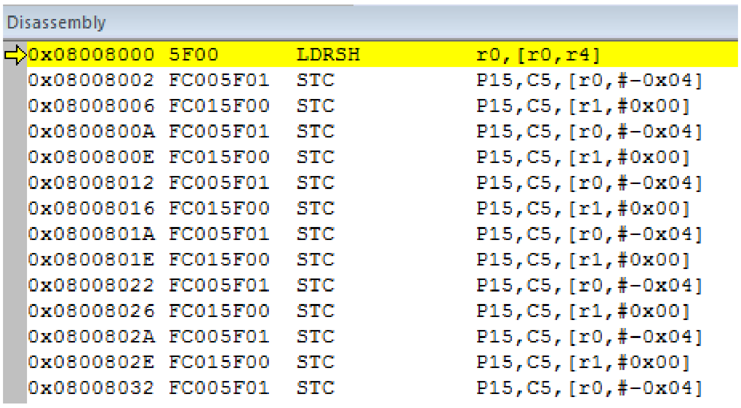
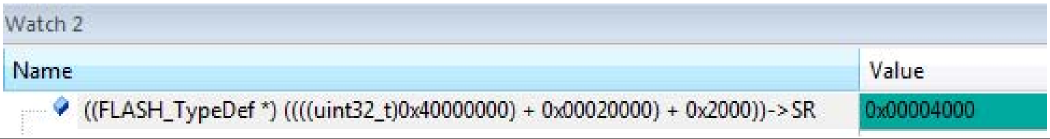


Figure 28. Reading PCROP-ed area sets RDERR flags in FLASH_SR register (bit[14]).



3 Conclusion

Microcontrollers of the STM32L4 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 in STM32L4xx MCUs can be used.

4 Revision history

Table 3. Document revision history

Date	Revision	Changes
26-Oct-2015	1	Initial release.
8-Sep-2017	2	Added references to RM0392 and RM0394 on the cover page.

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.

© 2017 STMicroelectronics – All rights reserved