SET UP A FIREWALL

Link

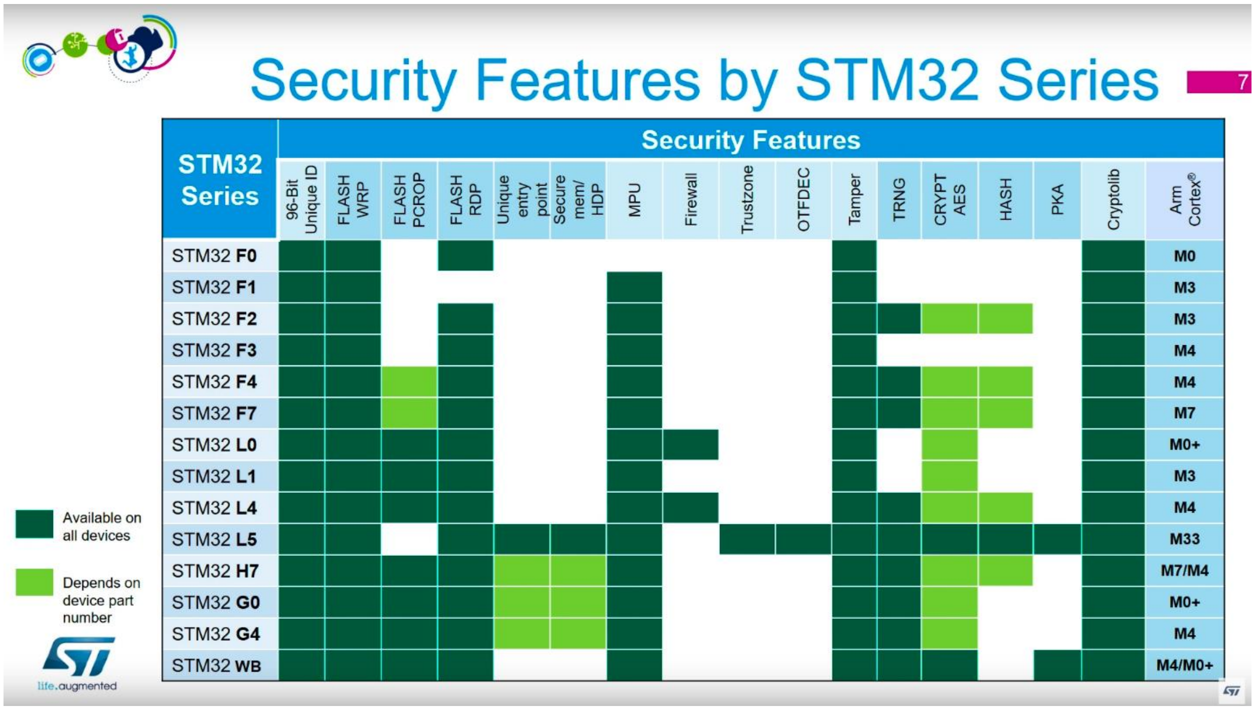
https://www.youtube.com/watch?v=qCuVBD2dmTA&list=PLnMKNibPkDnFzux3PHKUEi14ftDn9Cbm7&index=17

Description

In this paperwork, we will learn how to setup and configure a firewall on our STM32 board.

Contents

Prerequisites



Security Features by STM32 Series

7

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	■	■		■							■					■	**M0**
STM32 **F1**	■	■					■				■					■	**M3**
STM32 **F2**	■	■		■			■				■	■	▨	▨		■	**M3**
STM32 **F3**	■	■		■			■				■					■	**M4**
STM32 **F4**	■	■	▨	■			■				■	■	▨	▨		■	**M4**
STM32 **F7**	■	■	▨	■			■				■	■	▨	▨		■	**M7**
STM32 **L0**	■	■	■	■			■	■			■		▨			■	**M0+**
STM32 **L1**	■	■	■	■			■				■		▨			■	**M3**
STM32 **L4**	■	■	■	■			■	■			■	■	▨	▨		■	**M4**
STM32 **L5**	■	■		■	■	■	■		■	■	■	■	■	■	■	■	**M33**
STM32 **H7**	■	■	■	■	▨	▨	■				■	■	▨	▨		■	**M7/M4**
STM32 **G0**	■	■	■	■	▨	▨	■				■	■	▨			■	**M0+**
STM32 **G4**	■	■	■	■	▨	▨	■				■	■	▨			■	**M4**
STM32 **WB**	■	■	■	■			■				■	■	■		■	■	**M4/M0+**

■ Available on all devices

▨ Depends on device part number

life.augmented

STM32 Board

ST-Link cable

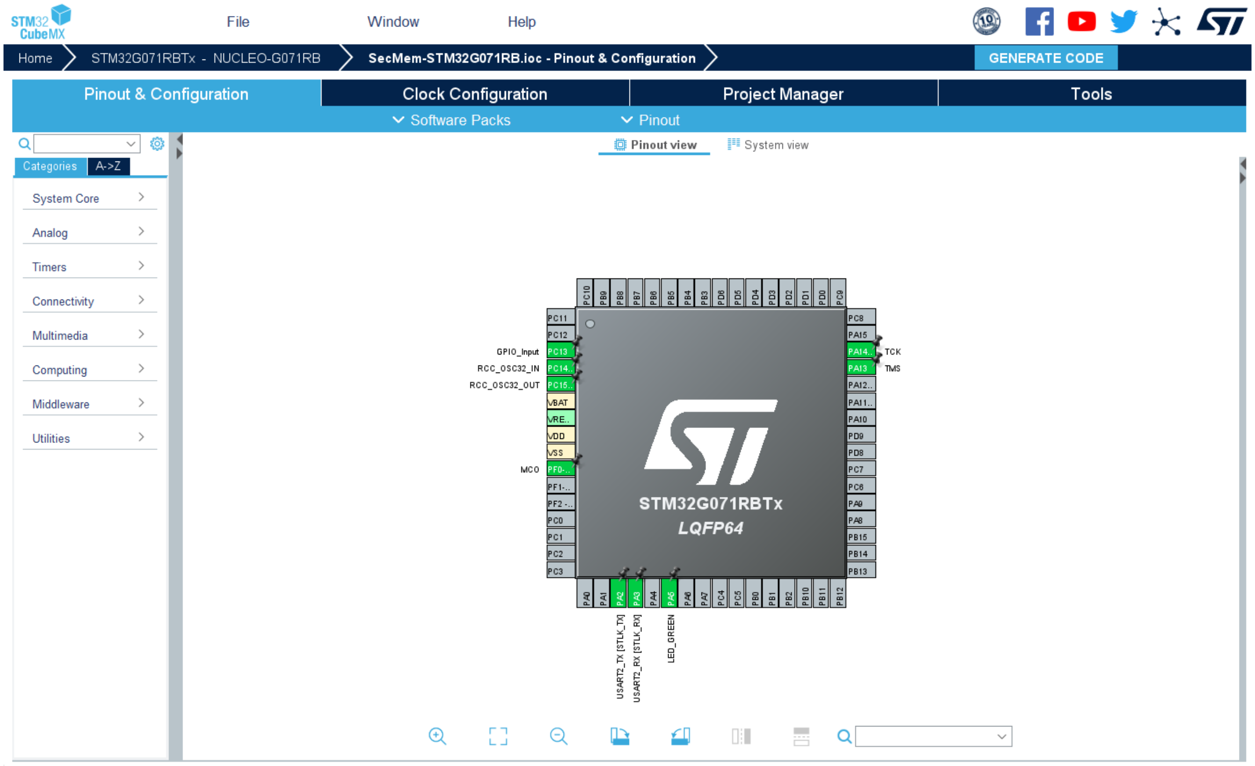
STM32CubeMX

STM32CubeIDE

Walkthrough

Step 1 : Launch STM32CubeMX and generate the code

Launch STM32CubeMX and select the right board depending on the one you are using. In my case I use the L476RG Nucleo board. Then you can generate the code of your project.
Don’t forget to select the correct IDE (in my case STM32CubeIDE).



Step 2 : Add firewall files

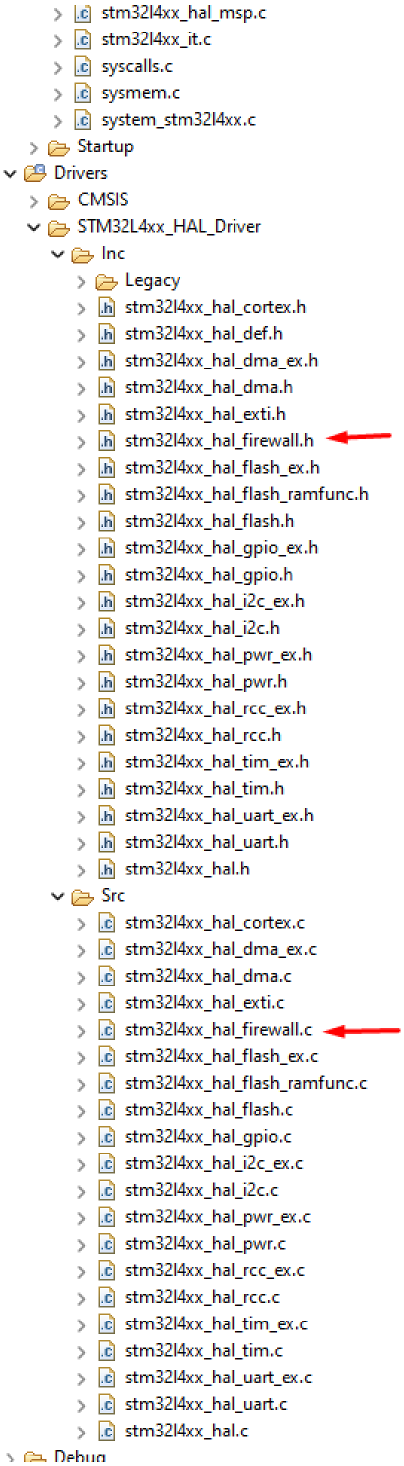
Go to your project, and go in Drivers > STM32L4xx_HAL_Driver > Inc.

In this folder, you will add the 2 files :

In Src : add STM32L4xx_hal_firewall.c

In Inc : add STM32L4xx_hal_firewall.h

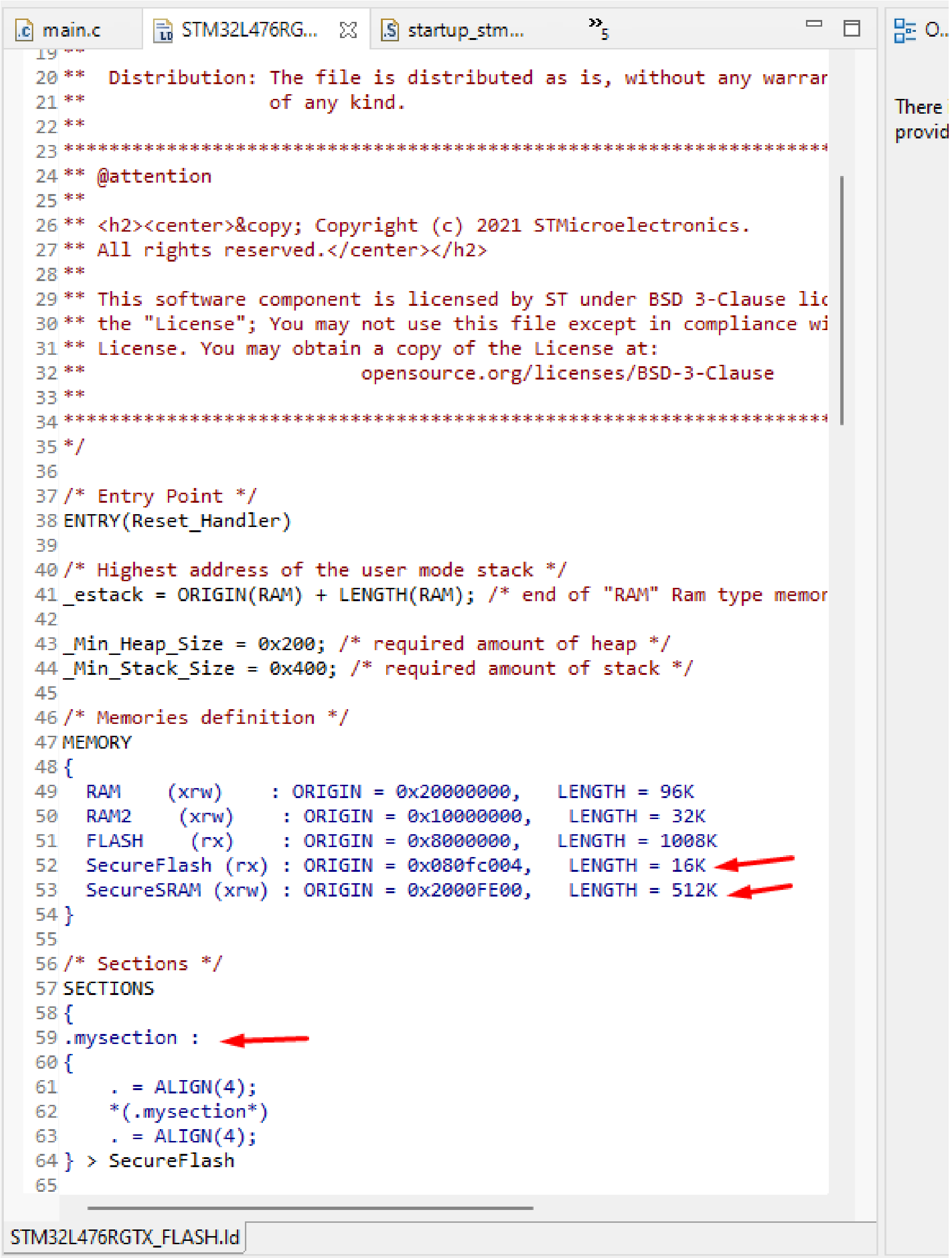
You could find your drivers in your STM32 File.



```
        > stm32l4xx_hal_msp.c
        > stm32l4xx_it.c
        > syscalls.c
        > sysmem.c
        > system_stm32l4xx.c
    > Startup
  v Drivers
    > CMSIS
    v STM32L4xx_HAL_Driver
      v Inc
        > Legacy
        > stm32l4xx_hal_cortex.h
        > stm32l4xx_hal_def.h
        > stm32l4xx_hal_dma_ex.h
        > stm32l4xx_hal_dma.h
        > stm32l4xx_hal_exti.h
        > stm32l4xx_hal_firewall.h
        > stm32l4xx_hal_flash_ex.h
        > stm32l4xx_hal_flash_ramfunc.h
        > stm32l4xx_hal_flash.h
        > stm32l4xx_hal_gpio_ex.h
        > stm32l4xx_hal_gpio.h
        > stm32l4xx_hal_i2c_ex.h
        > stm32l4xx_hal_i2c.h
        > stm32l4xx_hal_pwr_ex.h
        > stm32l4xx_hal_pwr.h
        > stm32l4xx_hal_rcc_ex.h
        > stm32l4xx_hal_rcc.h
        > stm32l4xx_hal_tim_ex.h
        > stm32l4xx_hal_tim.h
        > stm32l4xx_hal_uart_ex.h
        > stm32l4xx_hal_uart.h
        > stm32l4xx_hal.h
      v Src
        > stm32l4xx_hal_cortex.c
        > stm32l4xx_hal_dma_ex.c
        > stm32l4xx_hal_dma.c
        > stm32l4xx_hal_exti.c
        > stm32l4xx_hal_firewall.c
        > stm32l4xx_hal_flash_ex.c
        > stm32l4xx_hal_flash_ramfunc.c
        > stm32l4xx_hal_flash.c
        > stm32l4xx_hal_gpio.c
        > stm32l4xx_hal_i2c_ex.c
        > stm32l4xx_hal_i2c.c
        > stm32l4xx_hal_pwr_ex.c
        > stm32l4xx_hal_pwr.c
        > stm32l4xx_hal_rcc_ex.c
        > stm32l4xx_hal_rcc.c
        > stm32l4xx_hal_tim_ex.c
        > stm32l4xx_hal_tim.c
        > stm32l4xx_hal_uart_ex.c
        > stm32l4xx_hal_uart.c
        > stm32l4xx_hal.c
  > Debug
```

Step 3 : Separate the memory.

First thing, you have to separate the memory in the file named : <board>_FLASH.id.



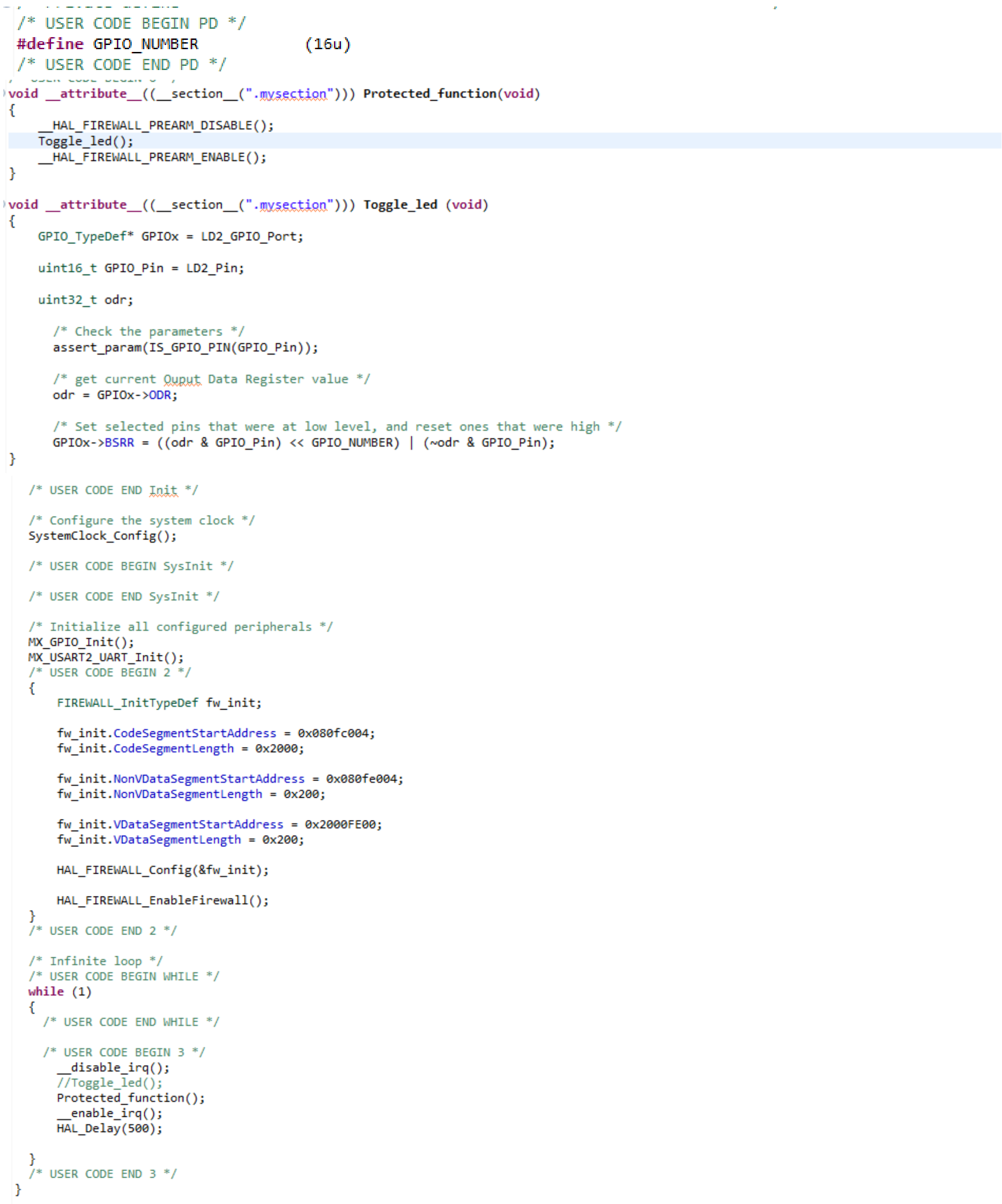
```
**
**  Distribution: The file is distributed as is, without any warrar
**                of any kind.
**
*******************************************************************
** @attention
**
** <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
** All rights reserved.</center></h2>
**
** This software component is licensed by ST under BSD 3-Clause lic
** the "License"; You may not use this file except in compliance wi
** License. You may obtain a copy of the License at:
**                        opensource.org/licenses/BSD-3-Clause
**
*******************************************************************
*/

/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = ORIGIN(RAM) + LENGTH(RAM); /* end of "RAM" Ram type memor

_Min_Heap_Size = 0x200; /* required amount of heap */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 96K
  RAM2    (xrw)    : ORIGIN = 0x10000000,   LENGTH = 32K
  FLASH    (rx)    : ORIGIN = 0x8000000,   LENGTH = 1008K
  SecureFlash (rx) : ORIGIN = 0x080fc004,   LENGTH = 16K
  SecureSRAM (xrw) : ORIGIN = 0x2000FE00,   LENGTH = 512K
}

/* Sections */
SECTIONS
{
.mysection :
{
    . = ALIGN(4);
    *(.mysection*)
    . = ALIGN(4);
} > SecureFlash
```

Step 4 : Write the main

You have define the GPIO_NUMBER. Then create the Protected_function that disable the firewall to call the Toggle_led function that is using the low level commands directly (so they are restricted with the firewall).

In the code section 2, we have the instantiation of the Firewall, and the function that enables it.

In section 3 of the code, you will have the interruptions that are disabled, and then calls the protected function in a loop with delay.



```c
 /* USER CODE BEGIN PD */
 #define GPIO_NUMBER           (16u)
 /* USER CODE END PD */
void __attribute__((__section__(".mysection"))) Protected_function(void)
{
    __HAL_FIREWALL_PREARM_DISABLE();
    Toggle_led();
    __HAL_FIREWALL_PREARM_ENABLE();
}

void __attribute__((__section__(".mysection"))) Toggle_led (void)
{
    GPIO_TypeDef* GPIOx = LD2_GPIO_Port;

    uint16_t GPIO_Pin = LD2_Pin;

    uint32_t odr;

      /* Check the parameters */
      assert_param(IS_GPIO_PIN(GPIO_Pin));

      /* get current Ouput Data Register value */
      odr = GPIOx->ODR;

      /* Set selected pins that were at low level, and reset ones that were high */
      GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
}

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  {
      FIREWALL_InitTypeDef fw_init;

      fw_init.CodeSegmentStartAddress = 0x080fc004;
      fw_init.CodeSegmentLength = 0x2000;

      fw_init.NonVDataSegmentStartAddress = 0x080fe004;
      fw_init.NonVDataSegmentLength = 0x200;

      fw_init.VDataSegmentStartAddress = 0x2000FE00;
      fw_init.VDataSegmentLength = 0x200;

      HAL_FIREWALL_Config(&fw_init);

      HAL_FIREWALL_EnableFirewall();
  }
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      __disable_irq();
      //Toggle_led();
      Protected_function();
      __enable_irq();
      HAL_Delay(500);

  }
  /* USER CODE END 3 */
 }
```

Step 5 : Conclusion

When we compile and run this code with the debugger, we can see that the led is blinking. Which is normal. To test if the firewall really works, you just have to comment the Protected_function in the loop and use the Toggle_led function.

By doing this you will see that the led is not blinking anymore because the firewall is not disabled as in the Protected_function.