BOOTLOADER TRÊN STM32F411VET (PHẦN 1)

I. Kiến thức cần chuẩn bị

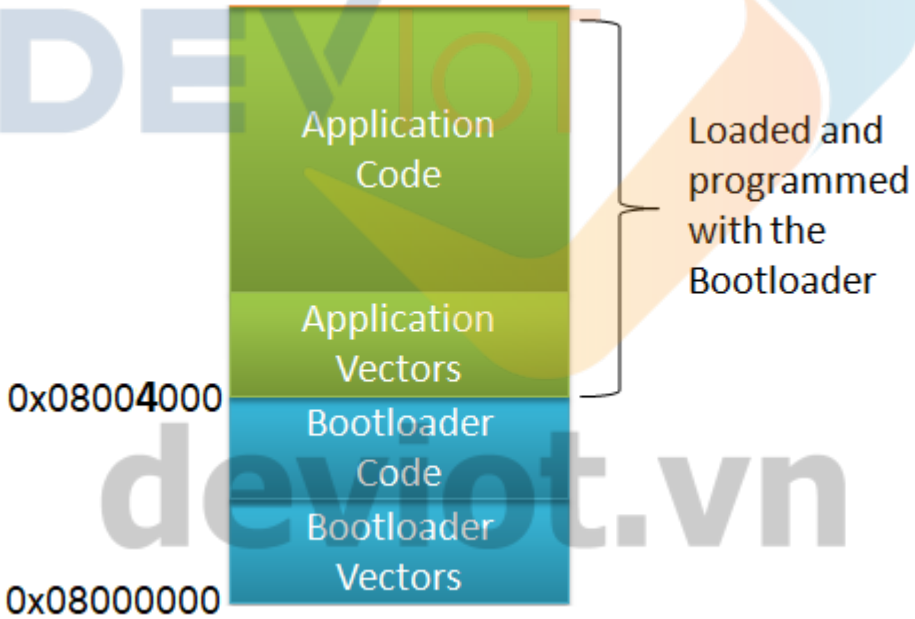
1. 1. Chương trình Bootloader

Bootloader là một chương trình và là chương trình đầu tiên chạy khi Chip hoạt động

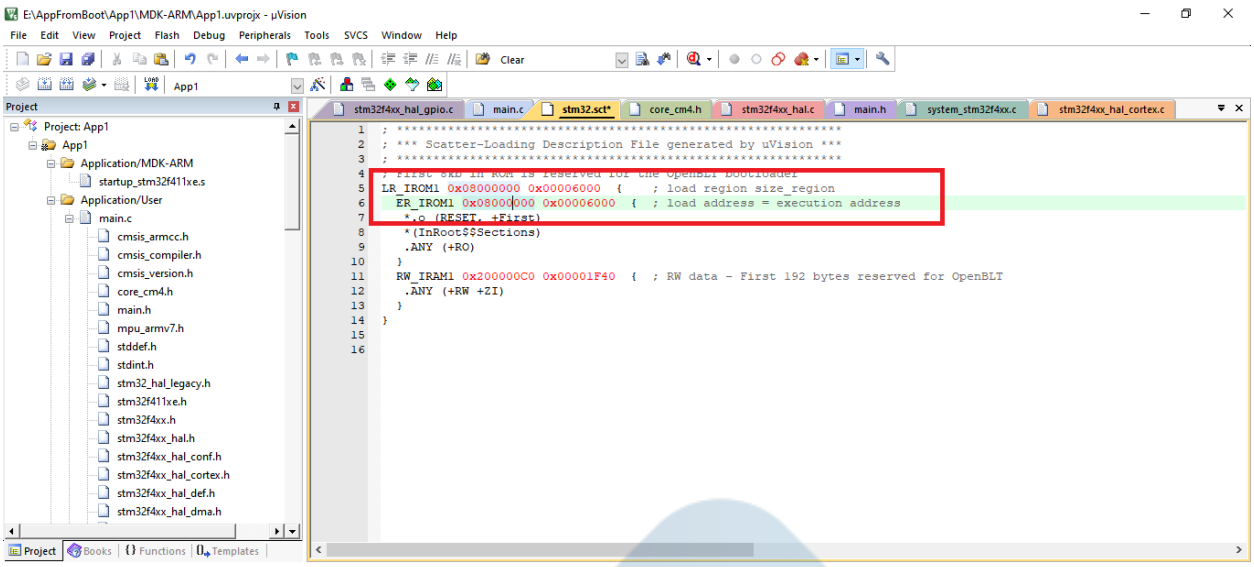
Nhiệm vụ của Bootloader là kiểm tra các điều kiện để lựa chọn thực thi một trong các chương trình: **FOTA**(bản Firmware update mới nhất), **Factory Firmware**(bản Firmware được nạp vào do hãng cung cấp trước khi xuất ra thị trường) hoặc **Current Firmware**(bản Firmware hiện tại đang chạy) mỗi khi CPU reset. Vị trí của chương trình Bootloader thường được bắt đầu tại địa chỉ đầu tiên của bộ nhớ Flash, đây là địa chỉ mặc định sẽ được CPU thực thi sau khi Reset. Với dòng vi điều khiển STM32 thì vị trí bắt đầu của bộ nhớ Flash là **0x08000000**.

Chương trình này sẽ được người dùng lập trình và nạp thủ công vào Chip.

Dưới đây là mô hình bộ nhớ đơn giản khi sử dụng chương trình Bootloader mà mình sử dụng với vi điều khiển STM32:



Bootloader và Application chạy riêng biệt lưu trên 2 vùng nhớ khác nhau . Địa chỉ mặc định khi chạy của Chip. Các bài trước khi ta tạo Project thì địa chỉ mặc định của App sẽ lưu tại File "***.sct***" trong thư mục Build. Tại đây địa chỉ mặc định là **0x08000000**.



Các thanh ghi liên quan đến quá trình khởi động và thực thi chương trình

Stack Pointer (SP): Thanh ghi lưu trữ giá trị trỏ tới vùng nhớ hiện tại trong bộ nhớ Stack. Mỗi chương trình sẽ có giá trị khởi tạo SP khác nhau, giá trị khởi tạo của SP sẽ được lưu tại địa chỉ bắt đầu của chương trình.

Program Counter (PC): Thanh ghi chứa địa chỉ câu lệnh tiếp theo sẽ được thực thi. Thanh ghi này sẽ tự động tăng lên mỗi khi thực hiện xong một lệnh .

Khi muốn chuyển sang chương trình mới, thanh ghi PC phải lấy giá trị từ địa chỉ bắt đầu chương trình + 4. Vì đây là nơi lưu trữ địa chỉ của Reset Handler, và hàm main() chương trình sẽ được thực thi từ đây.

Để một chương trình được thực thi thì cần có bảng **Vector Table** và mặc định khi không cấu hình thay đổi thì bảng **Vector Table** này nằm ở phần đầu tiên của bộ nhớ của bộ nhớ Flash. Bảng **Vector Table** chứa giá trị Reset của con trỏ Stack Pointer và địa chỉ cho tất cả exception handlers. Với chương trình Bootloader thì chúng ta không cần thay đổi vị trí bảng **Vector Table** vì Bootloader nằm ngay ở vị trí đầu tiên của bộ nhớ Flash.(*Address = 0x08000000*). Còn các chương trình còn lại gồm FOTA, Current Firmware và Factory Firmware nằm ở các vị trí khác trên bộ nhớ Flash nên chúng ta cần thay đổi địa chỉ bảng **Vector Table** tương ứng với vị trí bắt đầu của từng chương trình.

Sau khi chương trình Bootloader thực hiện xong ghi giá trị cho thanh ghi SP và CP thì chương trình sẽ nhảy tới một trong các chương trình Firmware tùy điều kiện.

Sau đó bắt buộc phải dời bảng vector ngắt ngay khi bắt đầu thực thi các chương trình ứng dụng.

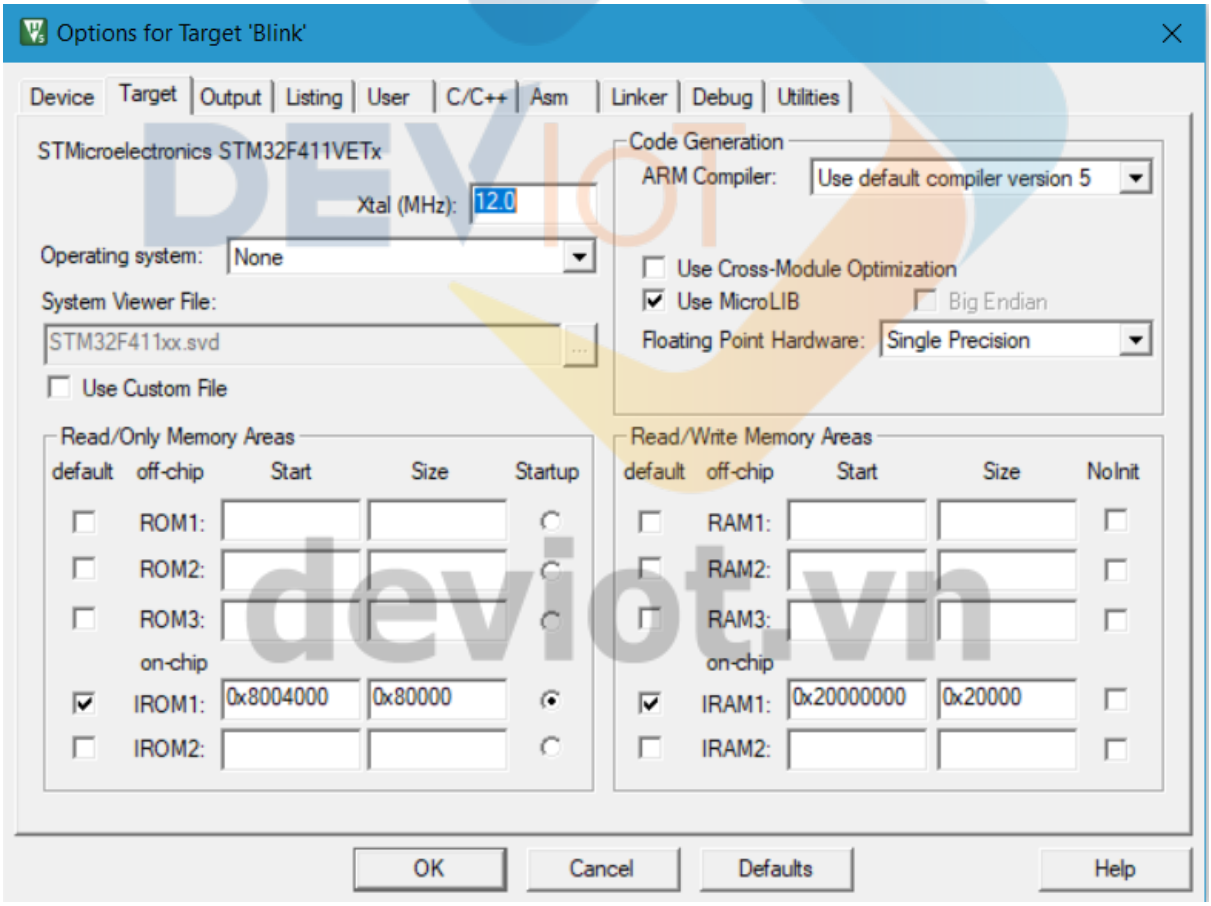
Ý tưởng Demo : Mình sẽ nạp 2 chương trình vào bộ nhớ Flash của chip STM32F411, chương trình 1 là chương trình Bootloader bắt đầu tại địa chỉ 0x8000000. Chương trình 2 là chương trình Blink LED đặt tại địa chỉ 0x8004000 trong Flash. Khi mình Reset Kit, chương trình Bootloader được thực thi, tại đây mình sẽ sử dụng lệnh để nhảy tới chương trình Blink LED.

2. II. Lập trình

3. 1. Tạo chương trình Blink LED.

Vẫn như mọi bài Application sẽ tạo bằng CubeMX. Các bạn có thể xem lại bài thực hành STM32F4 với GPIO nhé, ở đây mình sẽ không nhắc lại các bước nữa vì nó khá đơn giản.

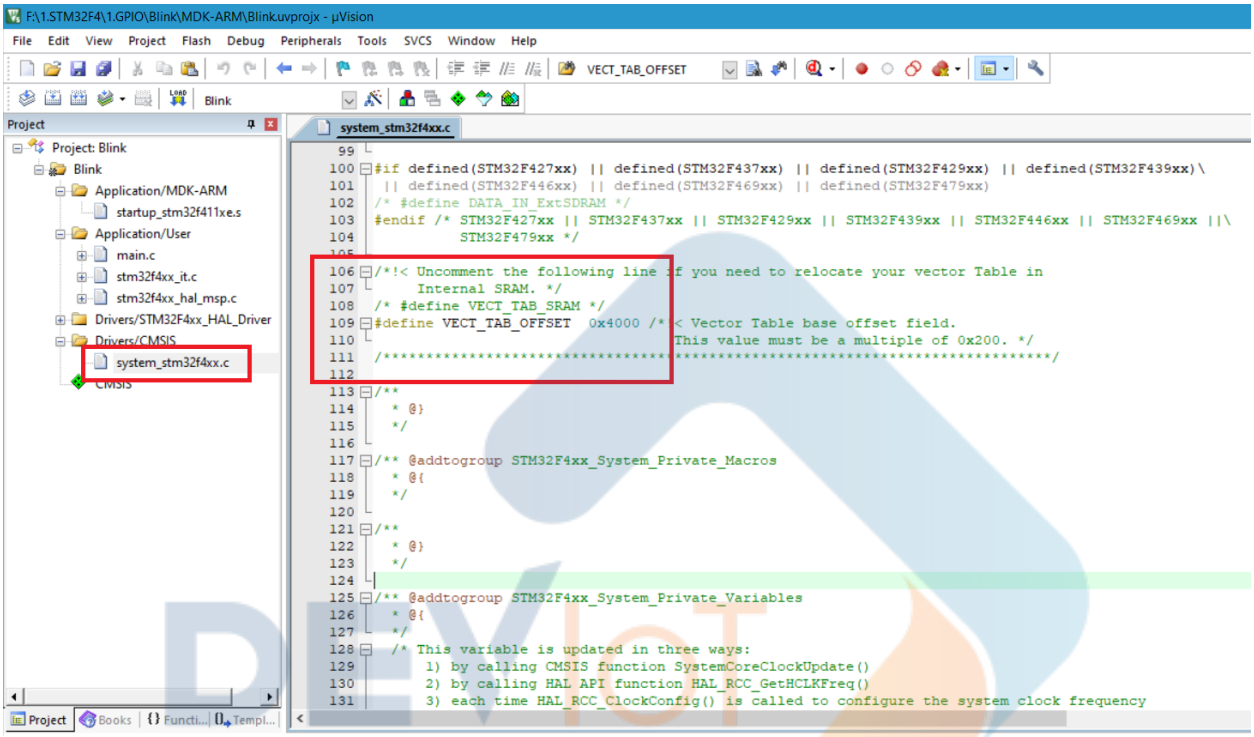
Mình sẽ đặt địa chỉ ghi chương trình **Blink LED** tại **0x8004000** nhé.



Tiếp theo mình cần cấu hình bảng Vector Table tới địa chỉ **0x8004000**. Ở đây mình sẽ có 2 cách để thực hiện việc này.

Cách 1: Các bạn mở file ***system_stm32f4xx.c*** và sửa thành

#define VECT_TAB_OFFSET 0x4000



Cách 2: Ngay sau khi vào hàm main các bạn thêm dòng code sau

```
66  int main(void)
67  {
68    /* USER CODE BEGIN 1 */
69    SCB->VTOR = (uint32_t)0x08004000;
70    /* USER CODE END 1 */
71
```

Tiếp đó, các bạn Build Project và nhớ đường dẫn tới output file **Blink.hex** để tiến hành nạp xuống KIT nhé.

Để nạp, các bạn tải và cài đặt phần mềm STM32 ST-LINK Utility theo link sau:

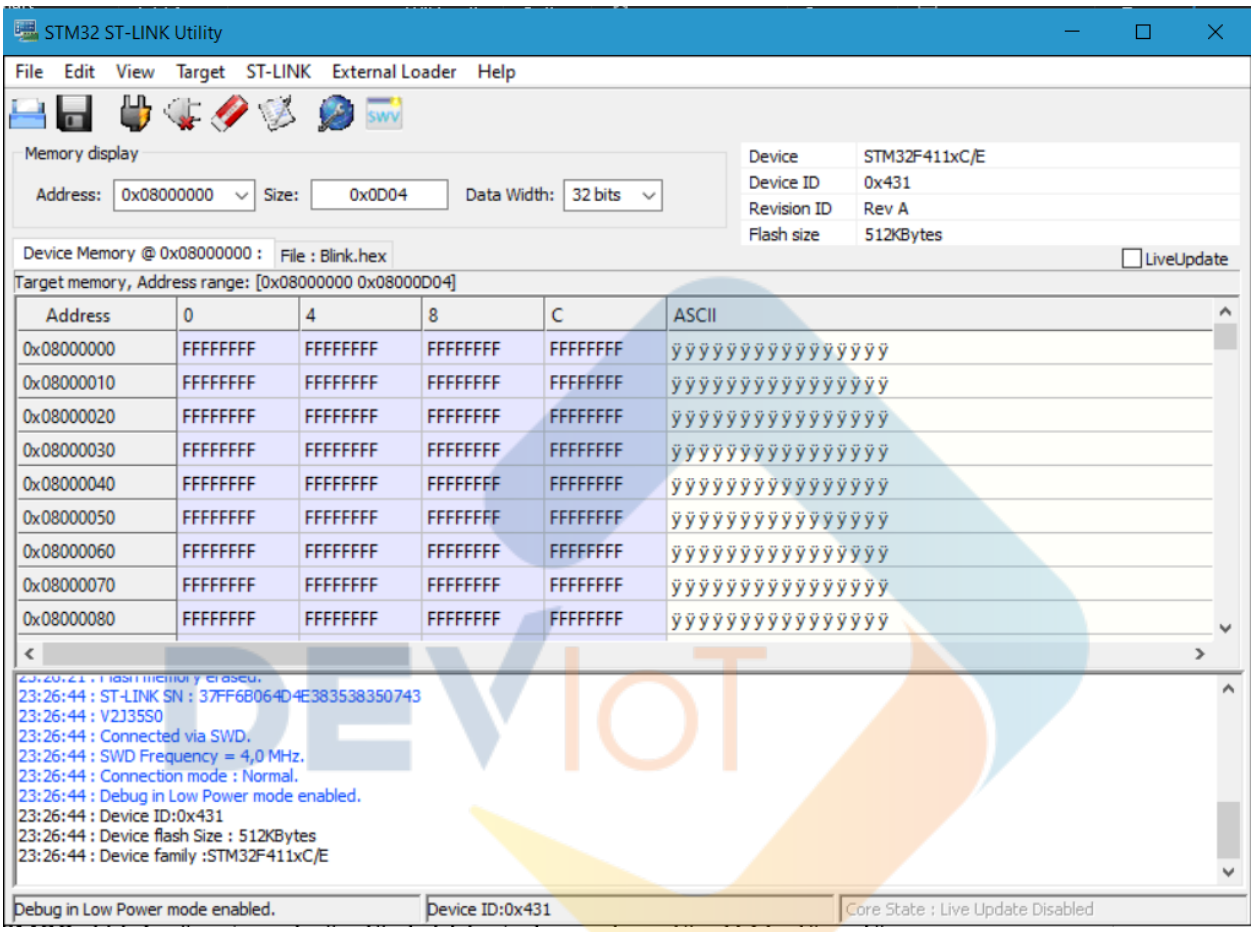
https://www.st.com/en/development-tools/stsw-link004.html

Sau khi cài xong, các bạn mở lên và làm theo các bước sau.

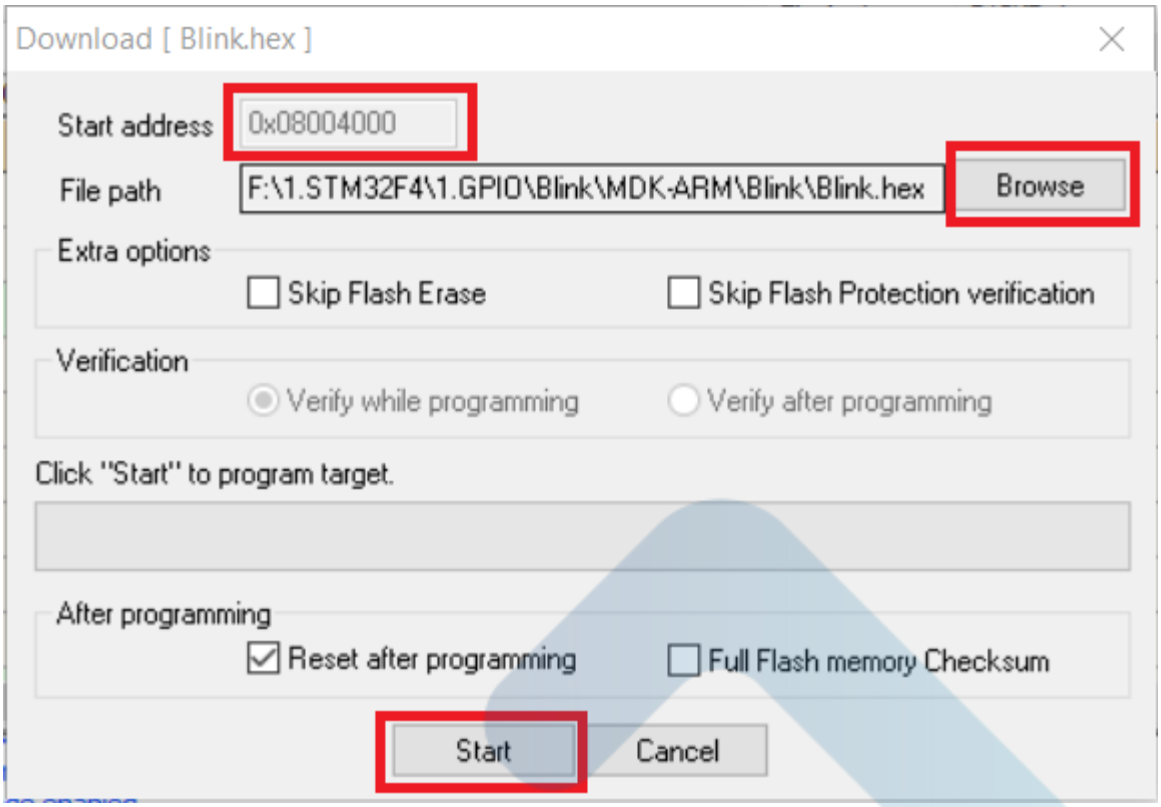
Chọn **Target** → **Connect**.

Chọn **Target** → **Erase Chip** để xóa toàn bộ Flash Chip.

Kết quả được như hình.

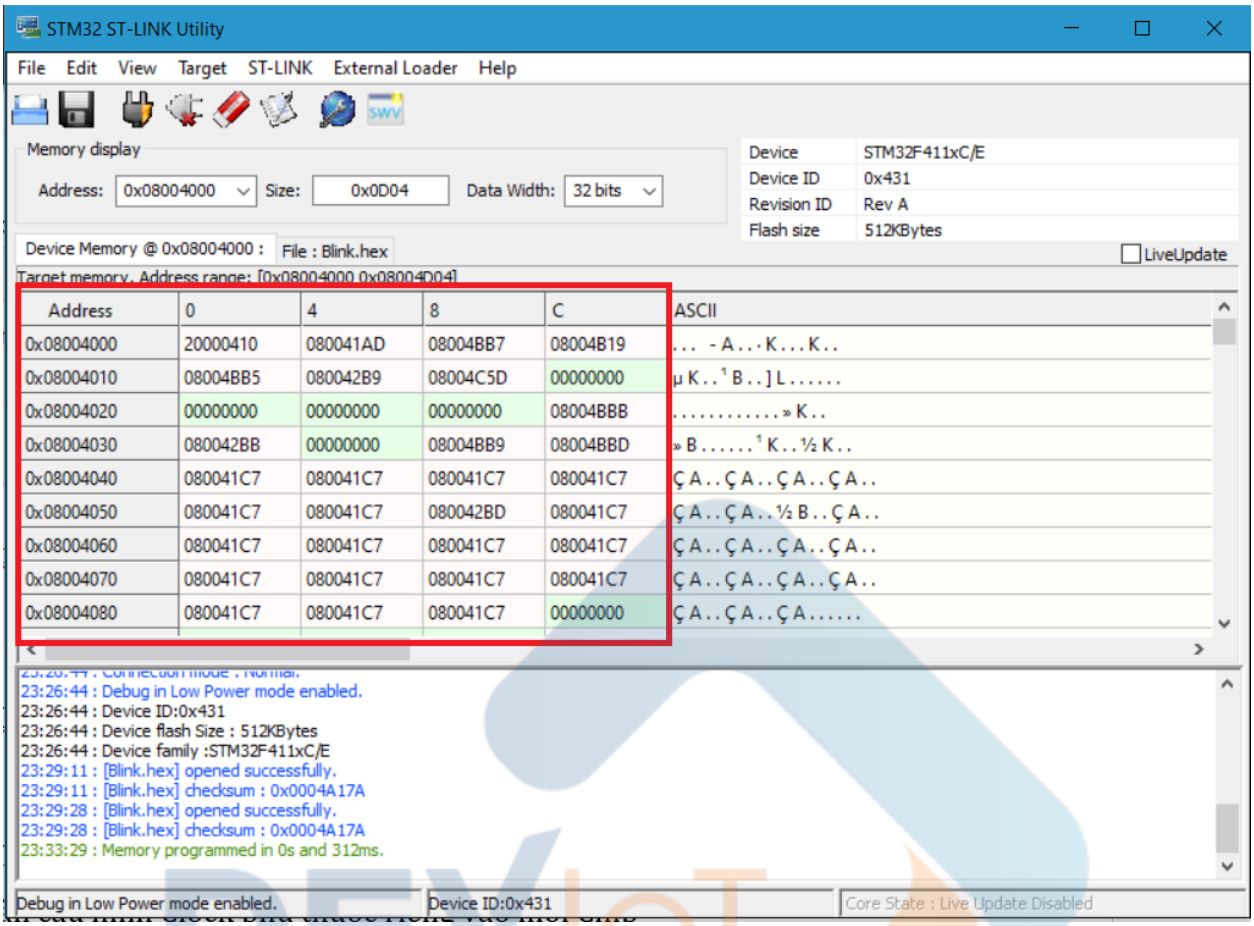


Để nạp chương trình **Blink.hex** vào địa chỉ **0x8004000**, các bạn chọn **Target** → **Program...**



Chọn **Browse** và tìm đến file **Blink.hex,** sau đó ấn **Start** và đợi phần mềm nạp xong nhé !

Kết quả thành công như hình

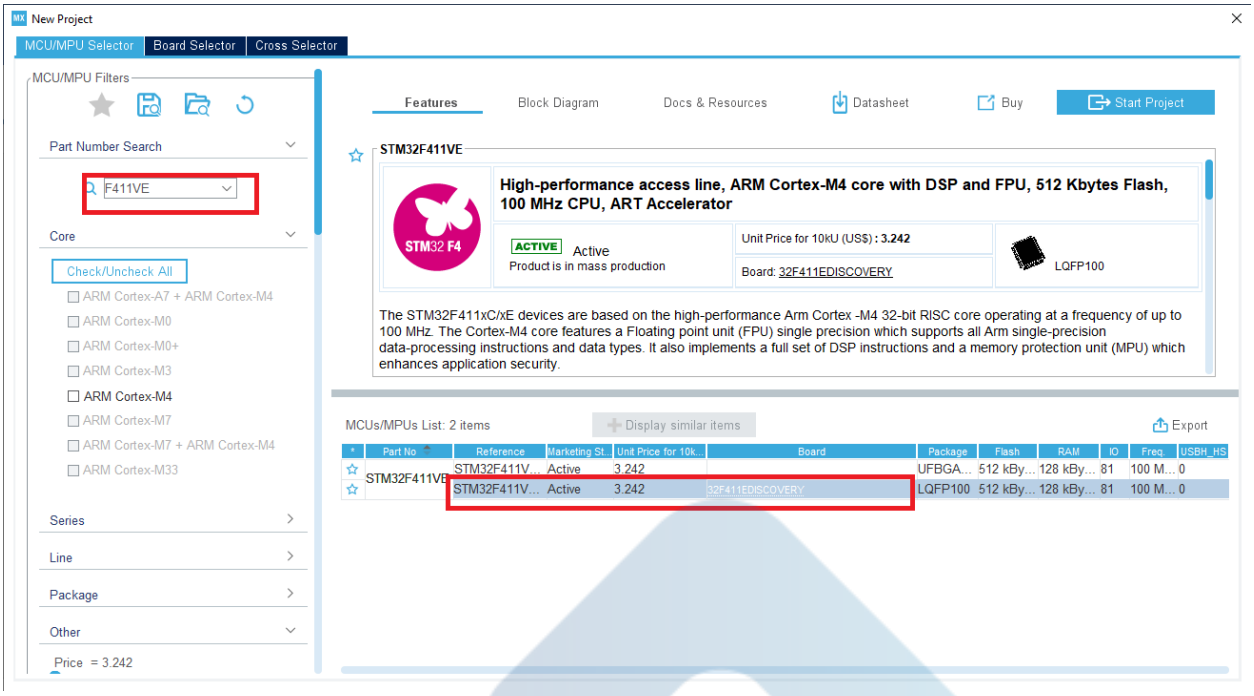


4. 2. Tạo chương trình Bootloader

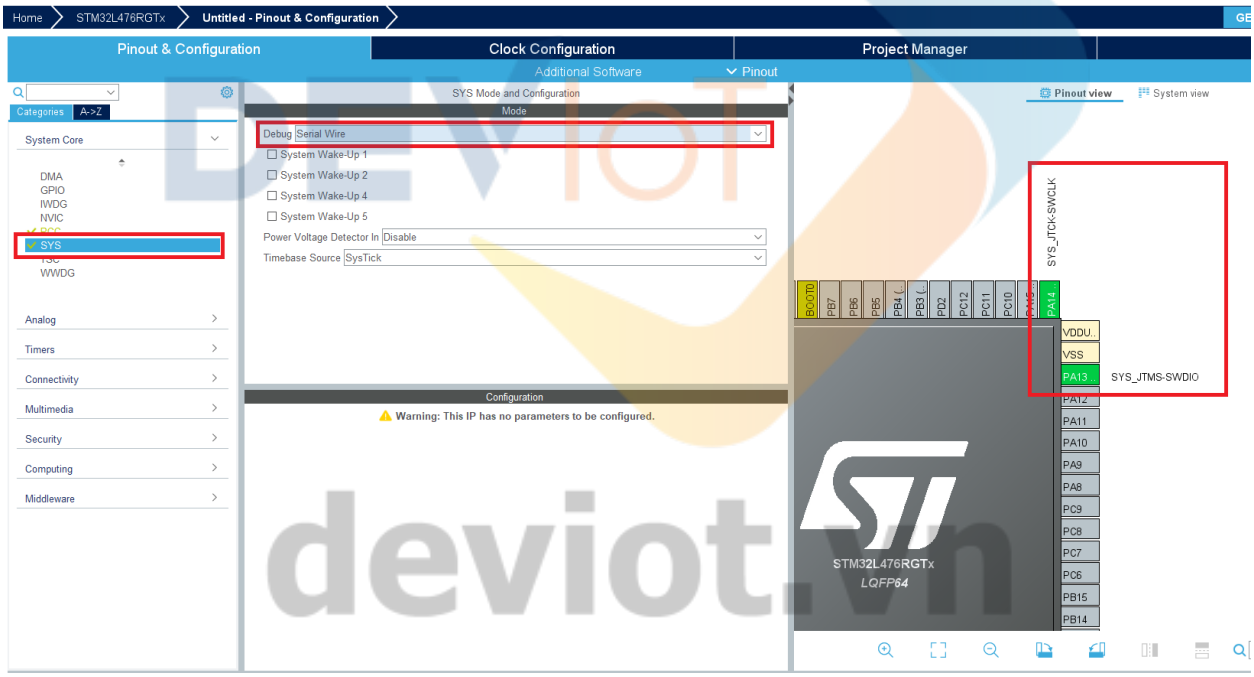
Tiếp theo chúng ta cùng nhau tạo chương trình Bootloader. Nó cũng chỉ là một chương trình bình thường nên các bạn sẽ tạo y các bước như chương trình Blink nhé !

Mở phần mềm STM CubeMX, chọn dòng chip bạn sử dụng. Ở đây mình chọn chip STM32F411VET.

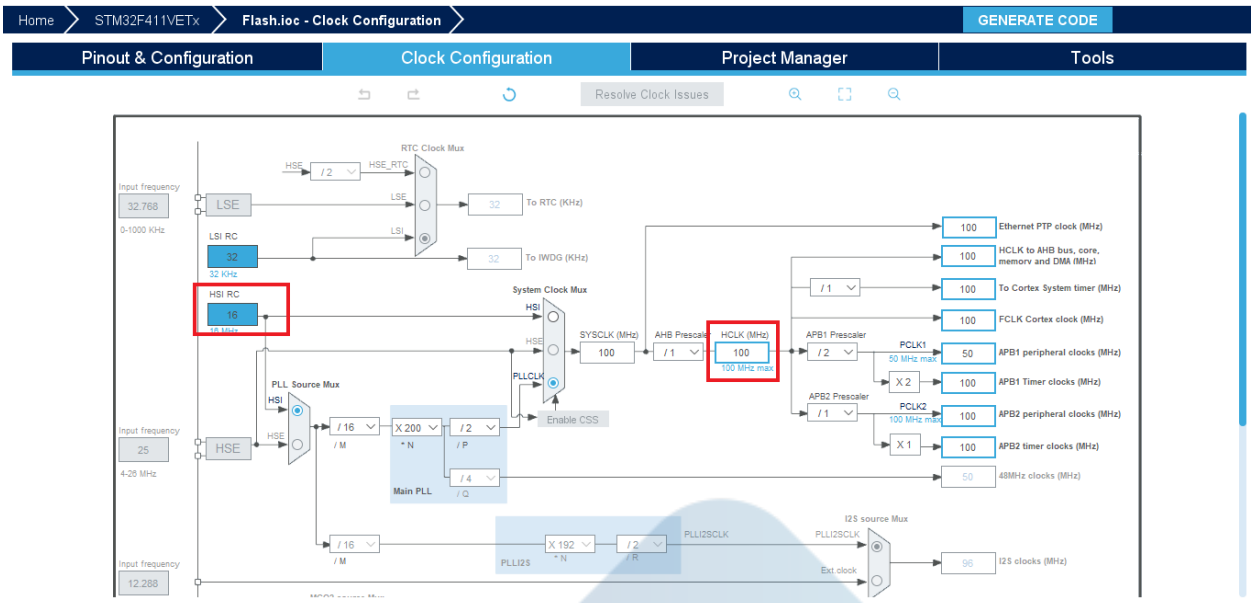
Đối với các dòng chip STM32 đời 4, tất cả mọi câu lệnh khi sử dụng thư viện HAL đều giống nhau. Chỉ khác nhau phần cấu hình Clock phụ thuộc riêng vào mỗi Chip.



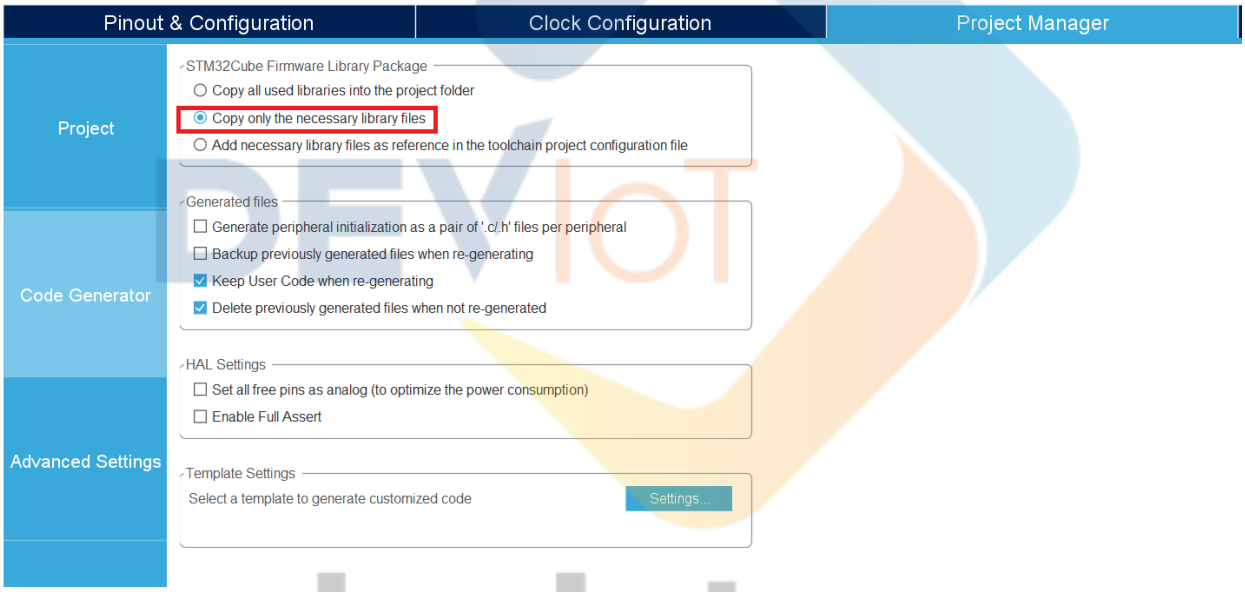
Cấu hình Chip Debug bằng mode SWD



Cấu hình Clock hoạt động. Ở đây mình chọn sử dụng nguồn Clock là xung Clock nội và chọn giá trị tối đa 16MHz. Clock đi qua bộ nhân tần PLLCLK để đạt được tần số hoạt động tối đa mà chip hỗ trợ HCLK = 100MHz. Việc còn lại Cube MX sẽ tự cấu hình cho các bạn.



Cuối cùng chọn file và sinh code cho Project



Chọn những thư viện cần thiết để sinh code nhanh hơn và giảm dung lượng Project nhé.

Trong hàm main, mình sử dụng code như sau

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
```

```
  * @brief          : Main program body
  ******************************************************************************

  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/

#include "main.h"

void SystemClock_Config(void);

static void MX_GPIO_Init(void);

uint32_t ADDRESS_START_BLINK_LED_APPLICATION = 0x08004000;

/**
  * @brief  The application entry point.
  * @retval int
  */

int main(void)
{
   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
   HAL_Init();
```

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

  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  /* Turn off Peripheral, Clear Interrupt Flag*/
  HAL_RCC_DeInit();

  /* Clear Pending Interrupt Request, turn  off System Tick*/
  HAL_DeInit();

  /* Turn off fault harder*/
  SCB->SHCSR &= ~( SCB_SHCSR_USGFAULTENA_Msk |\
  SCB_SHCSR_BUSFAULTENA_Msk | \
  SCB_SHCSR_MEMFAULTENA_Msk ) ;

  /* Set Main Stack Pointer*/
  __set_MSP(*((volatile uint32_t*) ADDRESS_START_BLINK_LED_APPLICATION));

 uint32_t JumpAddress = *((volatile uint32_t*) (ADDRESS_START_BLINK_LED_APPLICATION
+ 4));

 /* Set Program Counter to Blink LED Apptication Address*/
 void (*reset_handler)(void) = (void*)JumpAddress;
 reset_handler();

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */

 while (1)
 {
   /* USER CODE END WHILE */

   /* USER CODE BEGIN 3 */
 }
```

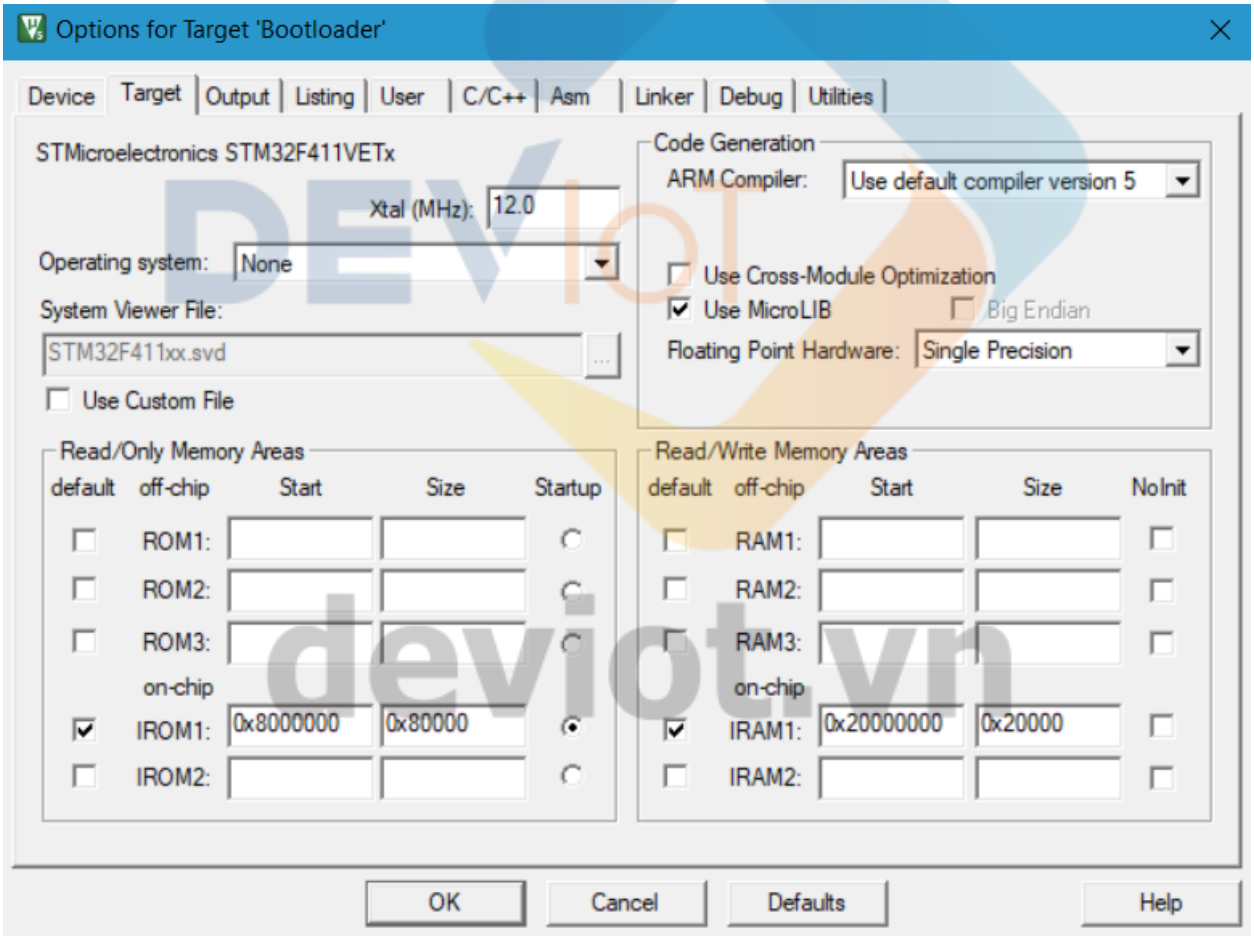
```
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the CPU, AHB and APB busses clocks*/

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 100;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB busses clocks*/
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
```

```
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,
GPIO_PIN_RESET);

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PD12 PD13 PD14 PD15 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
```

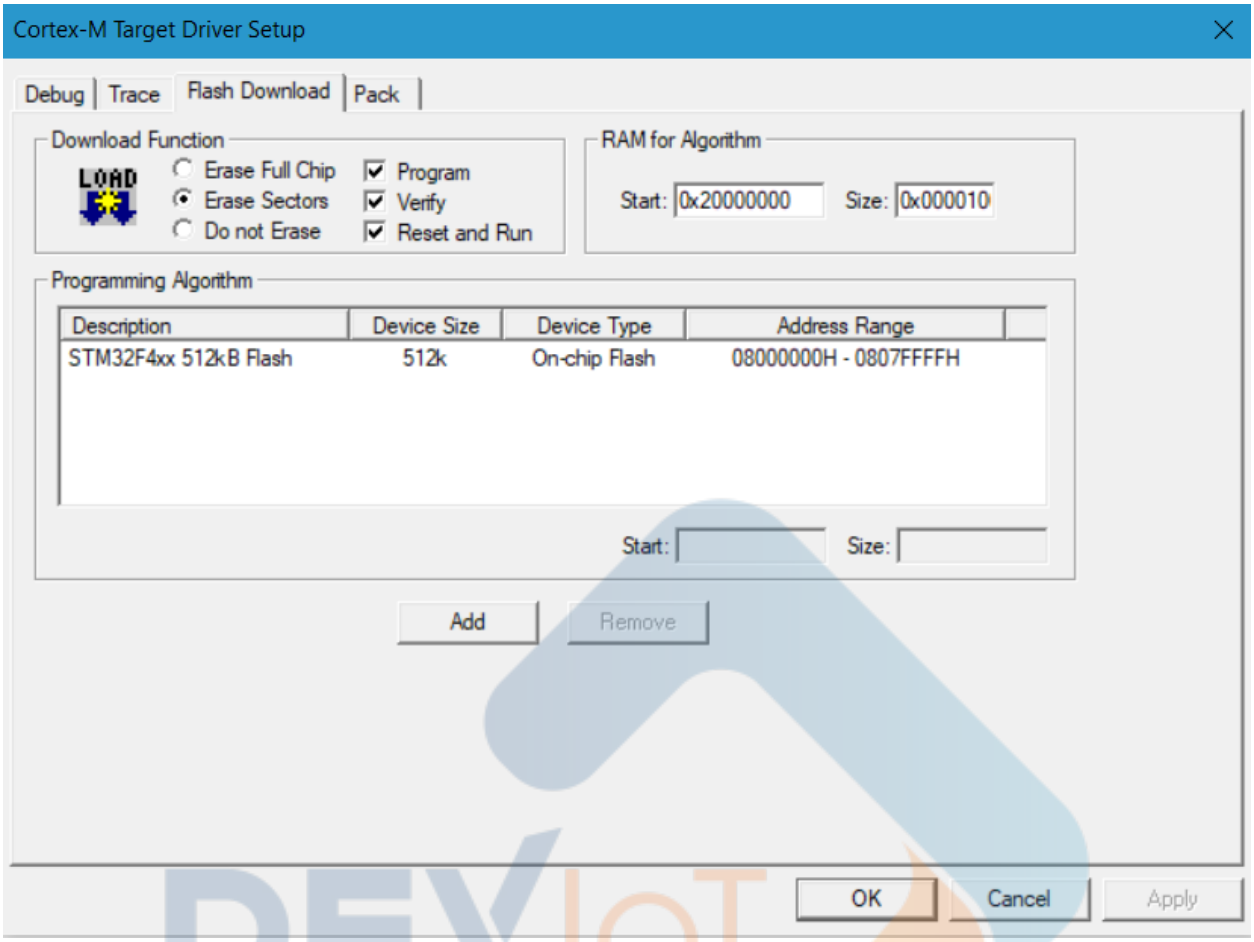
```
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}

/* USER CODE BEGIN 4 */
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */

void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */

  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**

  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
```

```
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */

  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  /* USER CODE END 6 */
}

#endif /* USE_FULL_ASSERT */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Tiếp theo, các bạn tiến hành Build và nạp code bằng STLink như bình thường.



Chú ý khi nạp có 1 số option như sau.



- **Erase Full Chip**: xóa toàn bộ Flash trước khi nạp, các bạn chọn Option này nó sẽ xóa luôn firmware Blink các bạn đã nạp ở phần 1.
- **Erase Sectors**: xóa Sector tính từ vị trí nạp Firmware, ở đây mình đặt Firmware Bootload ở 0x8000000. Vậy Option này sẽ xóa Sector 1 (từ 0x8000000 → 0x8003FFF). Nếu chọn Option này các bạn cần nạp Firmware Blink ngoài khoảng xóa, ở đây mình đặt chương trình Blink ở địa chỉ 0x8004000 nên vẫn an toàn.
- **Do not Erase**: không cần xóa, nạp luôn Firmware.

Cuối cùng mình Build chương trình và nạp xuống KIT.

Kết quả

Sau khi nạp, KIT của mình đã Blink được LED, chứng tỏ chương trình sau khi chạy vào Bootloader đã nhảy vào Blink Application thành công.

📌 Website: deviot.vn

📌 FanPage: Deviot - Thời sự kỹ thuật & IoT

📌 Group: Deviot - Cùng nhau học lập trình IOT

📌 Hotline: 0969.666.522

📌 Address: Số 101C, Xã Đàn 2

📌 Đào tạo thật, học thật, làm thật