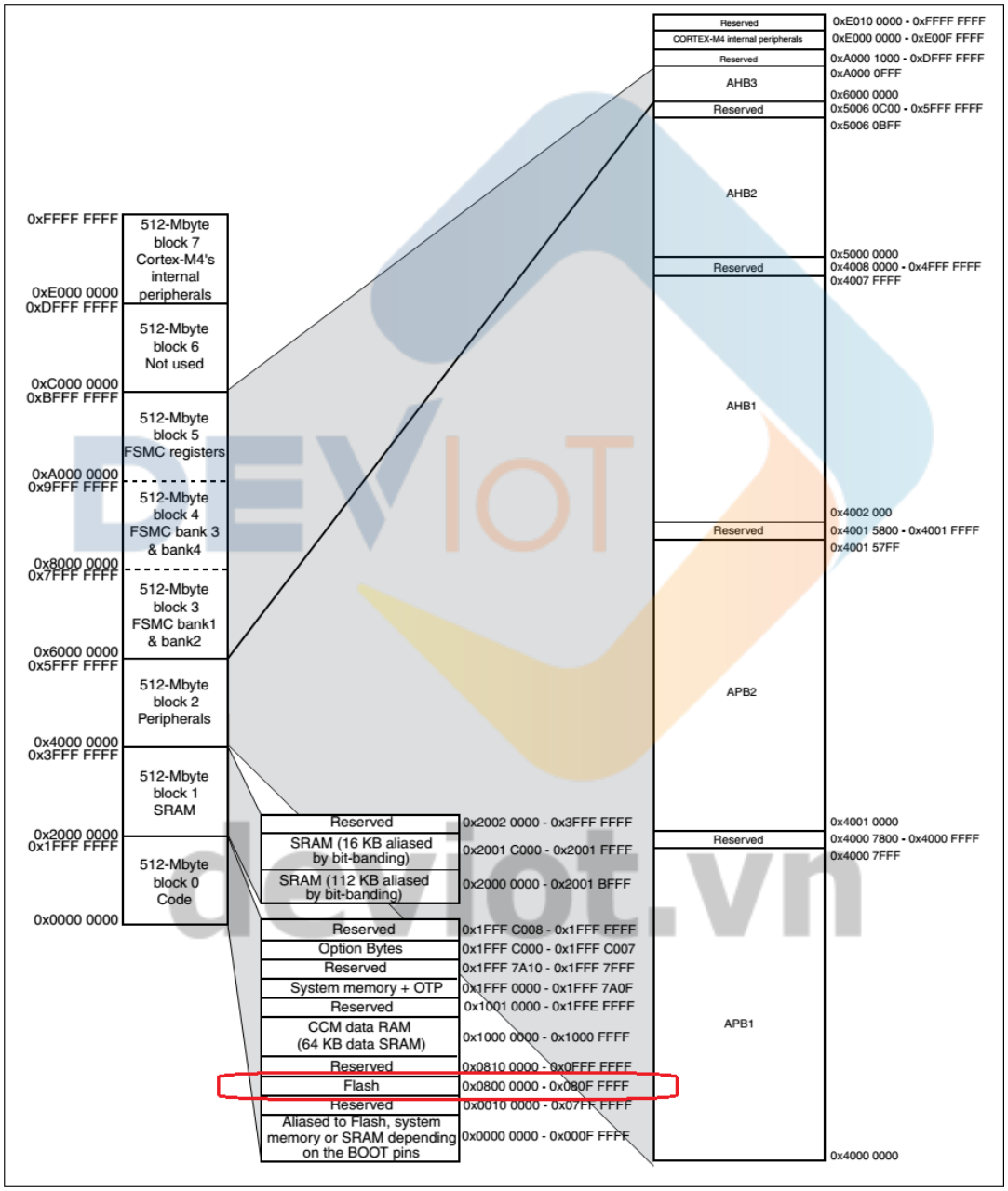
Bài 18 : Lập trình Flash (đọc, ghi, xóa) trên STM32

I. Kiến thức cần chuẩn bị

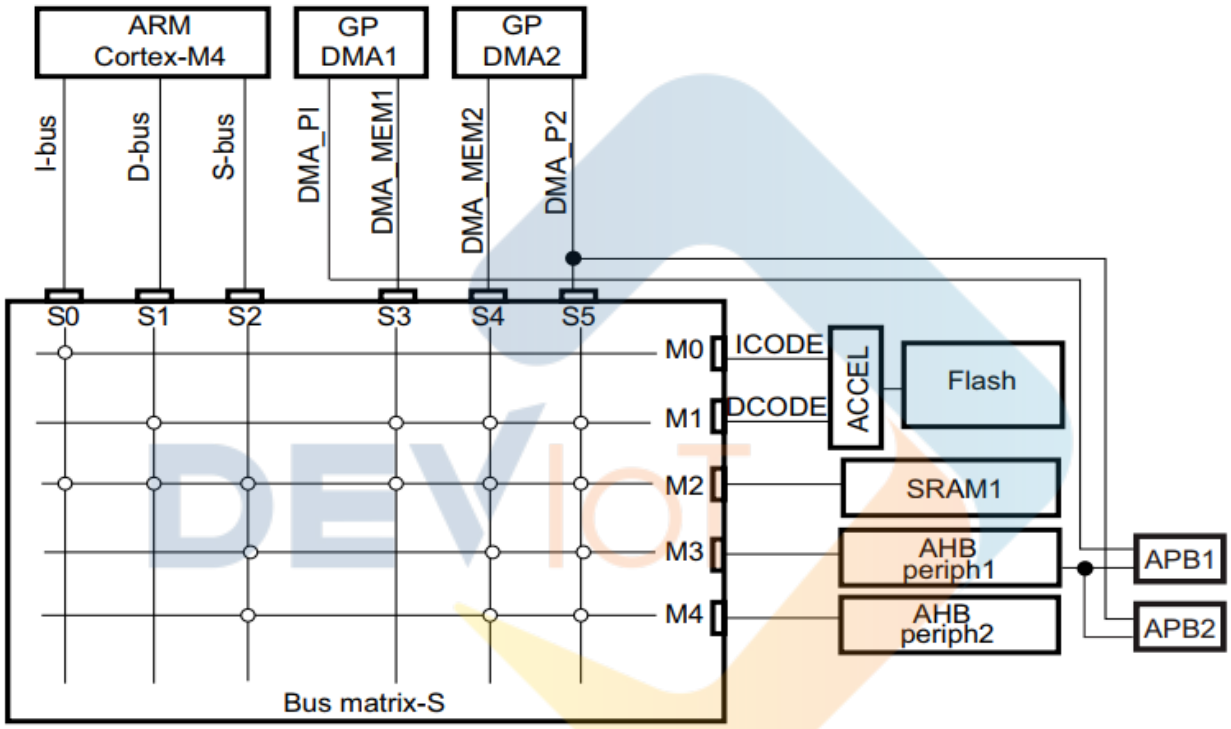
1. Lý thuyết

ARM Cortex-M4 dành 512MB địa chỉ đầu tiên cho vùng nhớ Code (Block 0). Trong đó vùng địa chỉ từ 0x0800 0000 đến 0x080F FFFF dành cho bộ nhớ Flash (xem hình 1).



Hình 1. Sơ đồ bộ nhớ của ARM Cortex-M4

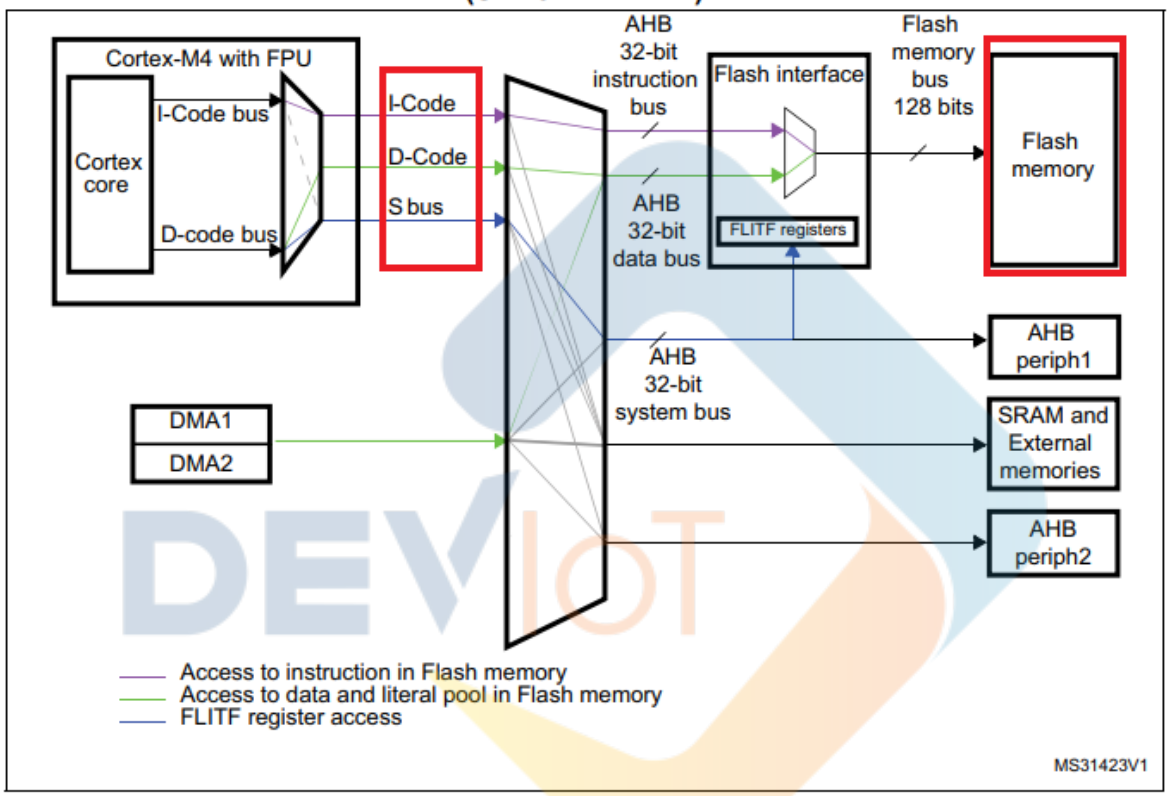
Chip STM32F411VE gồm 6 masters và 5 slaves được kết nối với nhau thông qua một ma trận bus đa lớp AHB 32-bit . 6 masters bao gồm: Cortex-M4 với I-bus, D-bus và S-bus của lõi FPU; DMA1 memory bus, DMA2 memory bus và DMA2 peripheral bus. 5 slaves bao gồm: ICode bus và DCode bus của bộ nhớ Flash nội; SRAM; APB1 peripharals và APB2 peripherals.



- I-bus: thực hiện nhiệm vụ kết nối Instruction bus (của Cortex-M4 với FPU) đến ma trận Bus để lấy các câu lệnh từ bộ nhớ Flash hoặc SRAM.
- D-bus: thực hiện nhiệm vụ kết nối Data bus (của Cortex®-M4 với FPU) đến ma trận Bus để tải các câu lệnh xuống hoặc truy cập debug từ bộ nhớ Flash hoặc SRAM.
- S-bus: kết nối bus hệ thống của Cortex®-M4 với FPU đến ma trận Bus để truy cập dữ liệu trên một ngoại vi hoặc trên SRAM. Ngoài ra bus này còn được dùng để lấy các câu lệnh giống I-bus nhưng kém hiệu quả hơn.

Giao diện bộ nhớ Flash (**Flash memory interface**) quản lý việc truy cập của các bus I-Code và D-Code AHB CPU vào bộ nhớ Flash. Nó thực hiện các hoạt động xóa và lập trình bộ nhớ Flash và các cơ chế bảo vệ đọc/ghi bộ nhớ Flash.

Figure 2. Flash memory interface connection inside system architecture (STM32F411xC/E)



Bộ nhớ Flash của STM32F411 có các đặc điểm chính sau:

- Kích thước lên tới 1 Mbyte
- Ghi theo Byte(8bit), Half-word(16bit), Word(32bit) hoặc Double Word(64bit).
- Xóa theo Sector(xóa 1 phần) hoặc Mass(xóa toàn bộ).
- Tổ chức bộ nhớ Flash (xem hình 2):
    - Vùng nhớ chính được chia thành 4 Sectors kích thước 16 Kbytes, 1 Sector kích thước 64 Kbytes, và 3 Sectors kích thước 128 Kbytes.

- Vùng nhớ hệ thống là nơi thiết bị khởi động ở chế độ System memory boot (30Kbytes)
- 512 byte OTP (one-time programmable) dành cho dữ liệu người dùng. Vùng nhớ OTP chứa 16 byte mở rộng dùng để khóa khối dữ liệu OTP tương ứng.
- Các byte tùy chọn để cấu hình cho việc bảo vệ đọc/ghi (option bytes) , BOR level, watchdog software/hardware và reset khi thiết bị ở chế độ Standby hoặc Stop.

- Các chế độ năng lượng thấp (chi tiết xem thêm Power control (PWR) trong tài liệu của ST Reference Manual.
  (https://www.st.com/resource/en/reference_manual/dm00119316-stm32f411xc-e-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf)

Table 4. Flash module organization (STM32F411xC/E)

Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	Sector 7	0x0806 0000 - 0x0807 FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

Hình 2. Tổ chức bộ nhớ Flash của STM32F411

2. Đọc dữ liệu từ Flash

Để đọc chính xác dữ liệu từ bộ nhớ Flash, số lượng các trạng thái chờ/độ trễ (**LATENCY**) phải được lập trình trong thanh ghi **FLASH_ACR** (Flash access control register) đúng với khoảng điện áp cung cấp của thiết bị và tần số clock của CPU (HCLK).

Table 5. Number of wait states according to CPU clock (HCLK) frequency

Wait states (WS) (LATENCY)	HCLK (MHz)			
	Voltage range 2.7 V - 3.6 V	Voltage range 2.4 V - 2.7 V	Voltage range 2.1 V - 2.4 V	Voltage range 1.71 V - 2.1 V
0 WS (1 CPU cycle)	0 < HCLK≤30	0 < HCLK ≤24	0 < HCLK ≤18	0 < HCLK ≤16
1 WS (2 CPU cycles)	30 < HCLK ≤64	24 < HCLK ≤48	18 < HCLK ≤36	16 <HCLK ≤32
2 WS (3 CPU cycles)	64 < HCLK ≤90	48 < HCLK ≤72	36 < HCLK ≤54	32 < HCLK ≤48
3 WS (4 CPU cycles)	90 < HCLK ≤100	72 < HCLK ≤96	54 < HCLK ≤72	48 < HCLK ≤64
4 WS (5 CPU cycles)	-	96 < HCLK ≤100	72 < HCLK ≤90	64 < HCLK ≤80
5 WS (6 CPU cycles)	-	-	90 < HCLK ≤100	80 < HCLK ≤96
6 WS (7 CPU cycles)	-	-	-	96 < HCLK ≤100

Tuy vậy việc đọc dữ liệu từ Flash vẫn là khá dễ dàng, chỉ cần biết địa chỉ ô nhớ là chúng ta có thể đọc dữ liệu như trong hàm dưới đây:

```
uint32_t flash_read(uint32_t address)
{
	return (*(__IO uint32_t*) address);
}
```

3. Xóa dữ liệu Flash:

Xóa bộ nhớ Flash:

Xóa bộ nhớ Flash có 2 kiểu:

- Xóa theo từng Sector
- Xóa Mass Erase (toàn bộ Chip).

```
HAL_StatusTypeDef	HAL_FLASHEx_Erase(FLASH_EraseInitTypeDef	*pEraseInit,
uint32_t *SectorError)
```

- **FLASH_EraseInitTypeDef** là cấu trúc FLASH Erase. Bao gồm các thông số sau
  - TypeErase: Kiểu xóa (Sector hoặc Mass Erase)
  - Banks: ở đây STM32F411 chỉ có Bank 1
  - Sector: Sector đầu tiên trong chuỗi Sector sẽ bị xóa
  - NbSectors: số Sector cần xóa
  - VoltageRange: Dải điện áp hoạt động của thiết bị
- **uint32_t *SectorError**: là con trỏ trỏ tới Sector bị lỗi nếu quá trình xóa bộ nhớ xảy ra lỗi. Giá trị mặc định của con trỏ này bằng 0xFFFFFFFF, nếu kết thúc quá trình xóa mà không xảy ra bất kì lỗi nào thì giá trị của con trỏ này sẽ không bị thay đổi.

4. Ghi dữ liệu vào Flash

Việc ghi dữ liệu vào Flash có phần khó khăn hơn so với việc đọc. Để ghi được dữ liệu lên vùng nhớ trước hết cần xóa vùng nhớ đó đi. Muốn xóa vùng nhớ thì cần mở khóa Flash trước.

Khóa/mở khóa bộ nhớ Flash :

Để khóa Flash ta dùng hàm:

```
HAL_StatusTypeDef HAL_FLASH_Lock(void)
```

Để mở khóa Flash ta dùng hàm:

```
HAL_StatusTypeDef HAL_FLASH_Unlock(void)
```

Việc ghi vào Flash có thể thực hiện theo Byte (8 bit), Half-word (16 bit), Word (32 bit) hoặc Double word (64 bit) bằng hàm dưới đây:

```
HAL_StatusTypeDef  HAL_FLASH_Program(uint32_t  TypeProgram,  uint32_t  Address,
uint64_t Data)
```

Trong đó:

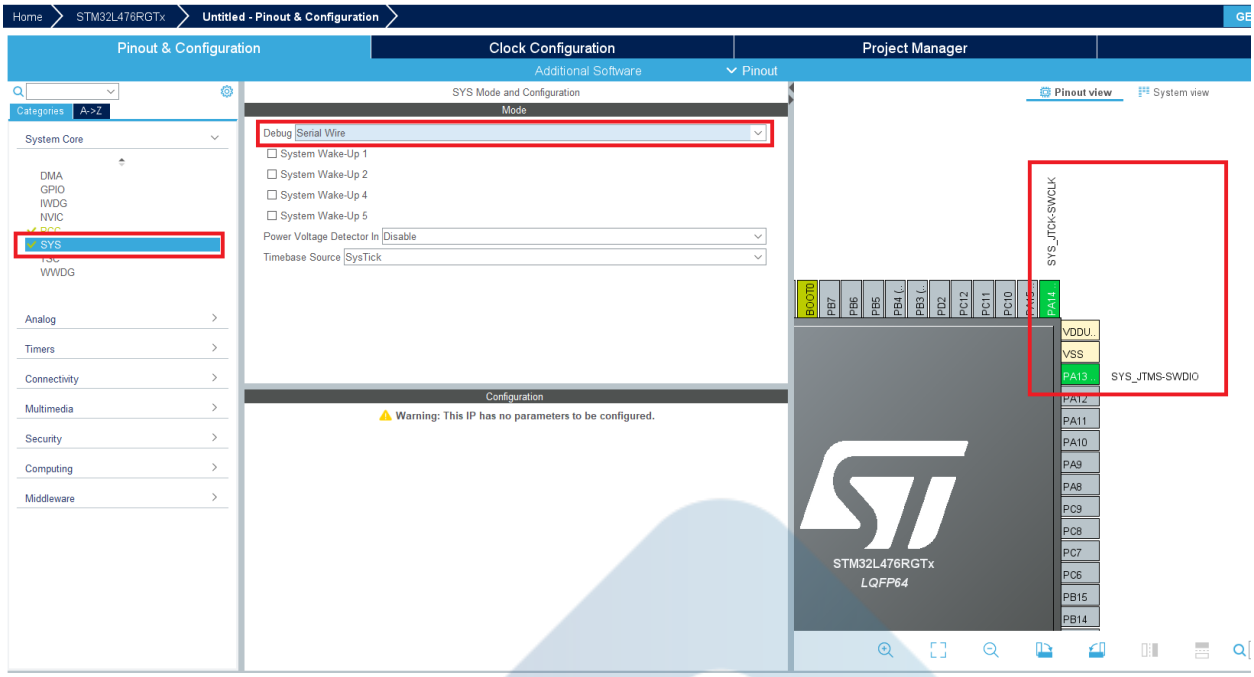
- TypeProgram: gồm có 4 loại, ghi 8bit, 16bit, 32bit, 64bit
- Address: địa chỉ cần ghi dữ liệu
- Data: dữ liệu cần ghi

II. Lập trình

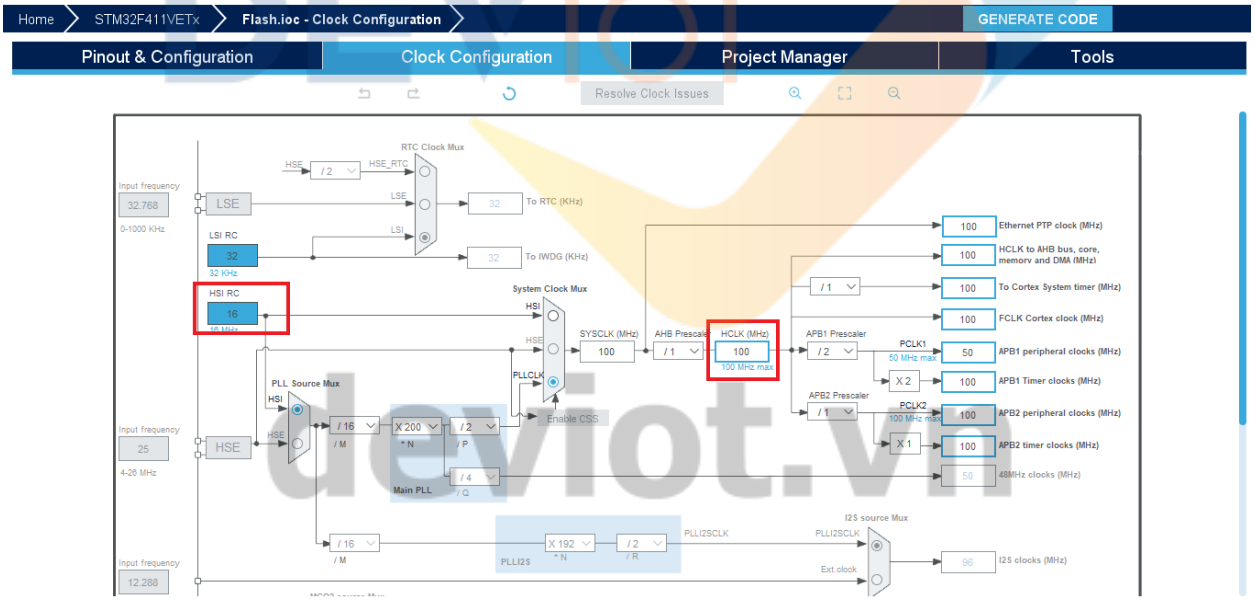
Mở phần mềm STM CubeMX, chọn dòng chip bạn sử dụng. Ở đây mình chọn chip STM32F411VE.

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.

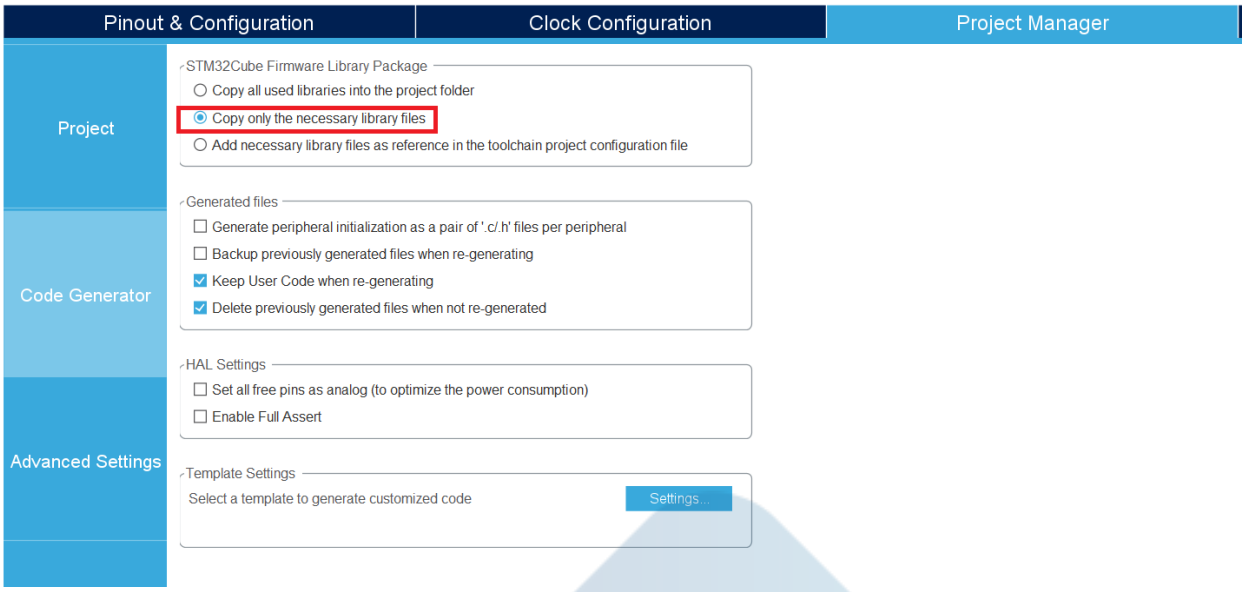
Cấu hình Chip Debug bằng mode SWD



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.

Sinh code và chuyển tới KeilC.

Chương trình xóa và ghi dữ liệu trong main.c

```
/* Base address of the Flash sectors */
#define ADDR_FLASH_SECTOR_0     ((uint32_t)0x08000000) /* Base @ of Sector 0, 16 Kbytes
*/
#define ADDR_FLASH_SECTOR_1     ((uint32_t)0x08004000) /* Base @ of Sector 1, 16 Kbytes
*/
#define ADDR_FLASH_SECTOR_2     ((uint32_t)0x08008000) /* Base @ of Sector 2, 16 Kbytes
*/
#define ADDR_FLASH_SECTOR_3     ((uint32_t)0x0800C000) /* Base @ of Sector 3, 16 Kbytes
*/
#define ADDR_FLASH_SECTOR_4     ((uint32_t)0x08010000) /* Base @ of Sector 4, 64 Kbytes
*/
#define ADDR_FLASH_SECTOR_5     ((uint32_t)0x08020000) /* Base @ of Sector 5, 128 Kbytes
*/
#define ADDR_FLASH_SECTOR_6     ((uint32_t)0x08040000) /* Base @ of Sector 6, 128 Kbytes
*/
#define ADDR_FLASH_SECTOR_7     ((uint32_t)0x08060000) /* Base @ of Sector 7, 128 Kbytes
*/

/**
  * @brief  Gets the sector of a given address
  * @param  None
  * @retval The sector of a given address
  */
uint32_t GetSector(uint32_t Address)
{
  uint32_t sector = 0;

  if((Address < ADDR_FLASH_SECTOR_1) && (Address >= ADDR_FLASH_SECTOR_0))
```

```
  {
    sector = FLASH_Sector_0;
  }
  else if((Address < ADDR_FLASH_SECTOR_2) && (Address >= ADDR_FLASH_SECTOR_1))
  {
    sector = FLASH_Sector_1;
  }
  else if((Address < ADDR_FLASH_SECTOR_3) && (Address >= ADDR_FLASH_SECTOR_2))
  {
    sector = FLASH_Sector_2;
  }
  else if((Address < ADDR_FLASH_SECTOR_4) && (Address >= ADDR_FLASH_SECTOR_3))
  {
    sector = FLASH_Sector_3;
  }
  else if((Address < ADDR_FLASH_SECTOR_5) && (Address >= ADDR_FLASH_SECTOR_4))
  {
    sector = FLASH_Sector_4;
  }
  else if((Address < ADDR_FLASH_SECTOR_6) && (Address >= ADDR_FLASH_SECTOR_5))
  {
    sector = FLASH_Sector_5;
  }
  else if((Address < ADDR_FLASH_SECTOR_7) && (Address >= ADDR_FLASH_SECTOR_6))
  {
    sector = FLASH_Sector_6;
  }
  else if((Address < ADDR_FLASH_SECTOR_8) && (Address >= ADDR_FLASH_SECTOR_7))
  {
    sector = FLASH_Sector_7;
  }
  return sector;
}

int main(void)
{
  HAL_Init();
  /* Cấu hình xung Clock hệ thống 100 MHz */
  SystemClock_Config();
  /* Unlock the Flash Để cho phép điều khiển thanh ghi *************/
  HAL_FLASH_Unlock();
  /* Xóa vùng Flash người sử dụng
    (xác định bởi FLASH_USER_START_ADDR và FLASH_USER_END_ADDR) ***********/
  /* Nhận vùng đầu tiên để xóa */
  FirstSector = GetSector(FLASH_USER_START_ADDR);
  /* Lấy số lượng sector để xóa từ sector 1 vừa nhận*/
  NbOfSectors = GetSector(FLASH_USER_END_ADDR) - FirstSector + 1;
  /* Cấu trúc khởi tạo Xóa*/
  EraseInitStruct.TypeErase = FLASH_TYPEERASE_SECTORS;
  EraseInitStruct.VoltageRange = FLASH_VOLTAGE_RANGE_3;
  /* với hiệu điện thế nằm giữa 2,7V và 3,6V ta truyền vào VoltageRange_3*/
  EraseInitStruct.Sector = FirstSector;
  EraseInitStruct.NbSectors = NbOfSectors;
  if(HAL_FLASHEx_Erase(&EraseInitStruct, &SectorError) != HAL_OK)
  {
    /* Xảy ra lỗi tròn khi xóa Sector sẽ cần thêm một số code để xử lý lỗi này
SectorError sẽ chứa sector bị lỗi,và sau đó để biết mã lỗi trên sector này bạn
cần gọi hàm 'HAL_FLASH_GetError()'*/
    /* FLASH_ErrorTypeDef errorcode = HAL_FLASH_GetError(); */
    Error_Handler();
```

```
  }

  /* Note: Nếu thao tác xóa trong bộ nhớ Flash cũng liên quan đến dữ liệu trong
the data hoặc bộ nhớ đệm,bạn phải chắc chắn các dữ liệu này được truy cập lại
trong khi chạy code
     Nếu điều này không thực hiện một cách an toàn có thể xóa bộ nhớ đệm bằng
cách cài đặt các bit
     DCRST và ICRST trong thanh ghi FLASH_CR . */
  __HAL_FLASH_DATA_CACHE_DISABLE();
  __HAL_FLASH_INSTRUCTION_CACHE_DISABLE();

  __HAL_FLASH_DATA_CACHE_RESET();
  __HAL_FLASH_INSTRUCTION_CACHE_RESET();

  __HAL_FLASH_INSTRUCTION_CACHE_ENABLE();
  __HAL_FLASH_DATA_CACHE_ENABLE();

  /* Truyền từng word vào vùng nhớ Flash
    (nằm giữa FLASH_USER_START_ADDR và FLASH_USER_END_ADDR) ***********/
  Address = FLASH_USER_START_ADDR;

  while (Address < FLASH_USER_END_ADDR)
  {
    if (HAL_FLASH_Program(FLASH_TYPEPROGRAM_WORD, Address, DATA_32) == HAL_OK)
    {
      Address = Address + 4;
    }
    else
    {
      /* Xảy ra lỗi khi ghi dữ liệu trong bộ nhớ Flash.
         Bạn có thể thêm vào đây code để xử lý lỗi này  */
      /* FLASH_ErrorTypeDef errorcode = HAL_FLASH_GetError(); */
      Error_Handler();
    }
  }
  /* Khóa Flash để tắt quyền truy cập điều khiển flash (được khuyến nghị
     để bảo vệ bộ nhớ FLASH ) *********/
  HAL_FLASH_Lock();

  /* Check if the programmed data is OK
      MemoryProgramStatus = 0: dữ liệu được truyền chính xác
      MemoryProgramStatus != 0: Sai rồi ******/
  Address = FLASH_USER_START_ADDR;
  MemoryProgramStatus = 0x0;

  while (Address < FLASH_USER_END_ADDR)
  {
    data32 = *(__IO uint32_t*)Address;

    if (data32 != DATA_32)
    {
      MemoryProgramStatus++;
    }

    Address = Address + 4;
  }
```

```
  if(MemoryProgramStatus)
{
      /*Có lỗi trong quá trình ghi*/
}

  /* Infinite loop */
  while (1)
  {
  }
}
```

DEVIOT - CÙNG NHAU HỌC LẬP TRÌNH IOT

📌 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