TÀI LIỆU

THỰC HÀNH KỸ THUẬT VI XỬ LÝ ARM

(Sử dụng cho Custom Board STM32F103C8T6)

Hà Nội, 2019

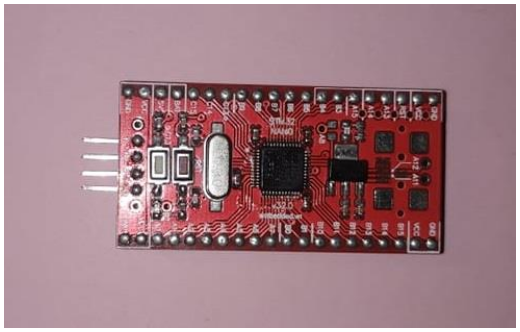
MỤC LỤC

(Tài liệu xây dựng theo datasheet của ST và tài liệu từ nguồn internet)

1. Tổng quan về KIT STM32F103C8T6

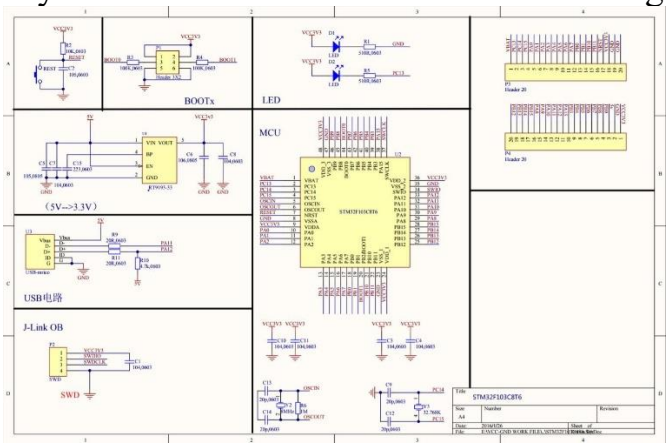
1.1. Sơ lược về KIT STM32F103C8T6

- STM32 là một trong những dòng chip phổ biến của ST với nhiều họ thông dụng như F0,F1,F2,F3,F4….. Stm32f103 thuộc họ F1 với lõi là ARM COTEX M3. STM32F103 là vi điều khiển 32 bit, tốc độ tối đa là 72Mhz. Giá thành cũng khá rẻ so với các loại vi điều khiển có chức năng tương tự. Mạch nạp cũng như công cụ lập trình khá đa dạng và dễ sử dụng.
- Một số ứng dụng chính: dùng cho driver để điều khiển ứng dụng, điều khiển ứng dụng thông thường, thiết bị cầm tay và thuốc, máy tính và thiết bị ngoại vi chơi game, GPS cơ bản, các ứng dụng trong công nghiệp, thiết bị lập trình PLC, biến tần, máy in, máy quét, hệ thống cảnh báo, thiết bị liên lạc nội bộ…
- Phiên bản KIT STM32F103C8T6 sử dụng trong bài viết



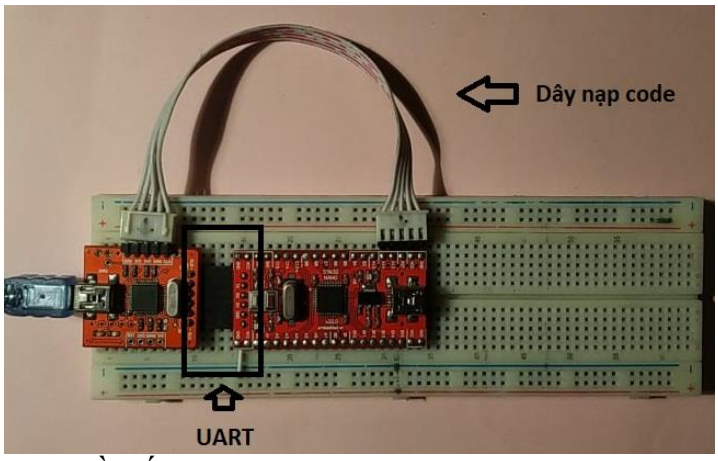
1.2. Sơ đồ nguyên lý:

Xây dựng và tùy biến trên Schiematic reference của hãng cung cấp



1.3. Mạch nạp:

- Sử dụng mạch nạp Jlink kèm theo bộ KIT.
- Cách kết nối



- Sơ đồ nối dây:

Mạch nạp Jlink	**KIT STM32F103C8T6**
GND	GND
3V3	VCC
RST	RST
SWD	A13
CLK	A14

- Sơ đồ cắm UART:

Mạch nạp Jlink	**KIT STM32F103C8T6**
GND / 5V (Cắm theo thứ tự)	GND / 5V
TX	RX
RX	TX
CTS, DTR	Cắm vào vị trí trống

2. Cài đặt phần mềm cần thiết

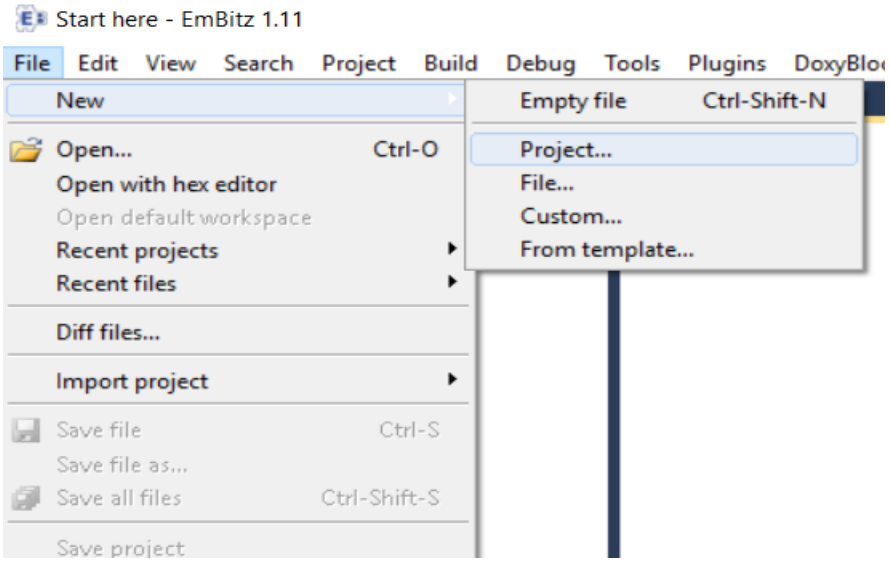
2.1.Cài đặt Compiler

- Trong hướng dẫn này, chúng ta lựa chọn trình biên dịch là ARM-GCC. Vì vậy tải và cài đặt chương trình Embitz tại : https://www.embitz.org/

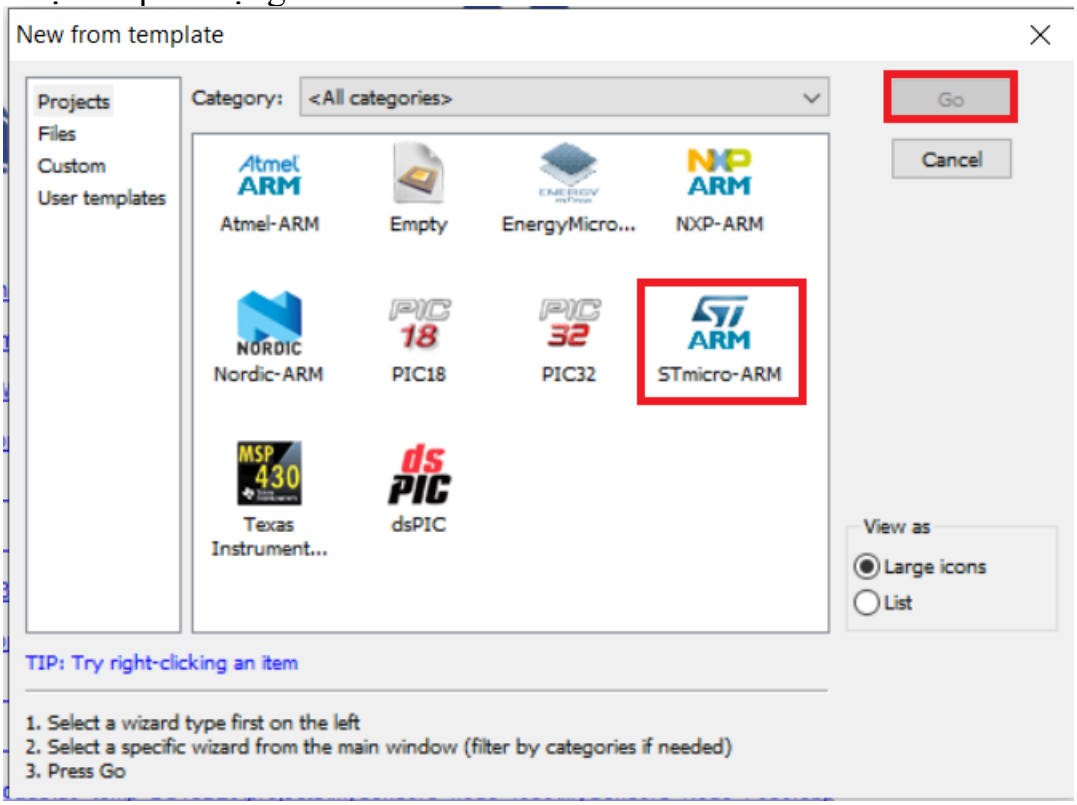
2.2.Tạo một project hoàn chỉnh

- Để tạo 1 project hoàn chỉnh , thực hiện các bước sau:

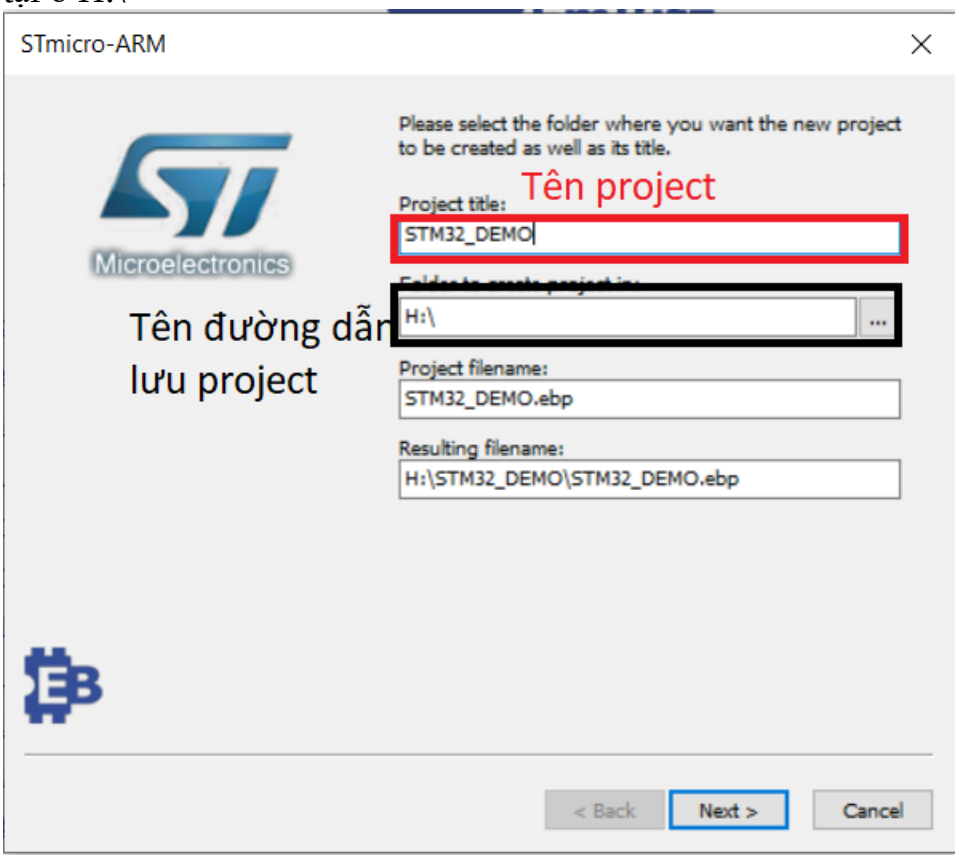
1. Mở chương trình Embitz , chọn File >> New >> Project



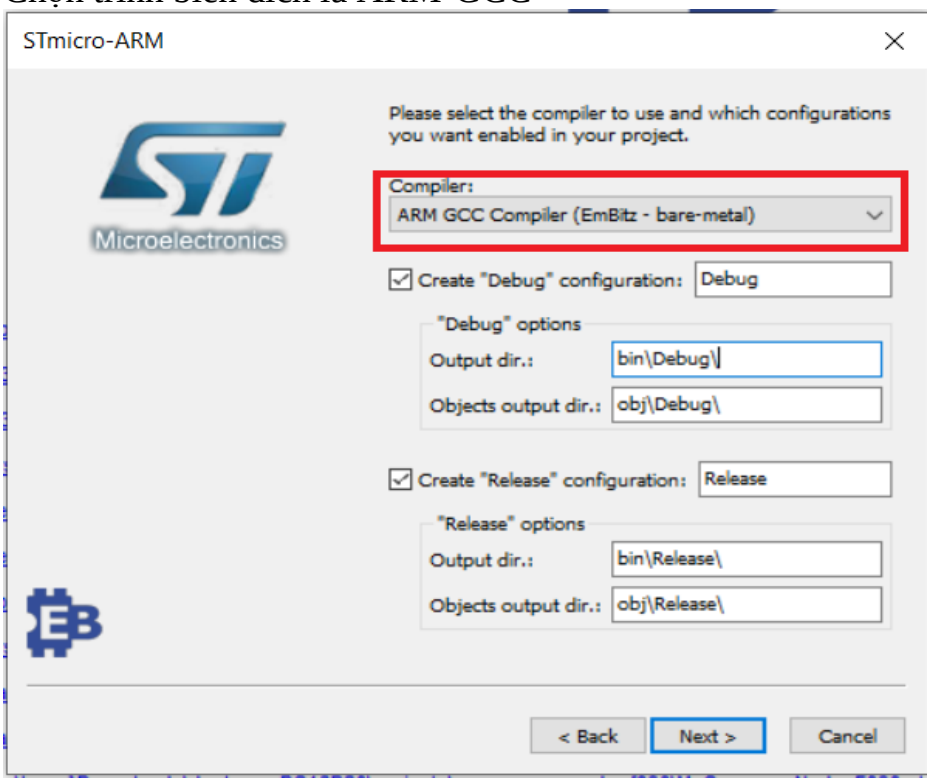
2. Chọn chip sử dụng như hình



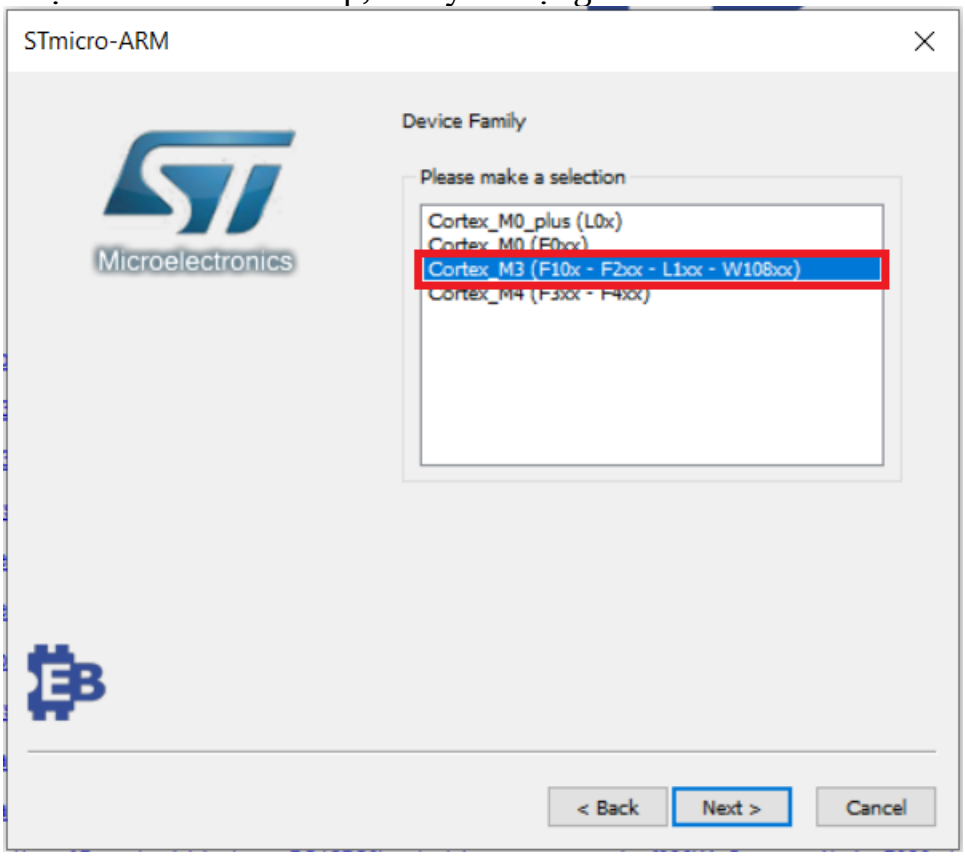
3. Chọn tên và đường dẫn project. ở đây tôi đặt tên project là STM32_DEMO tại ổ H:\



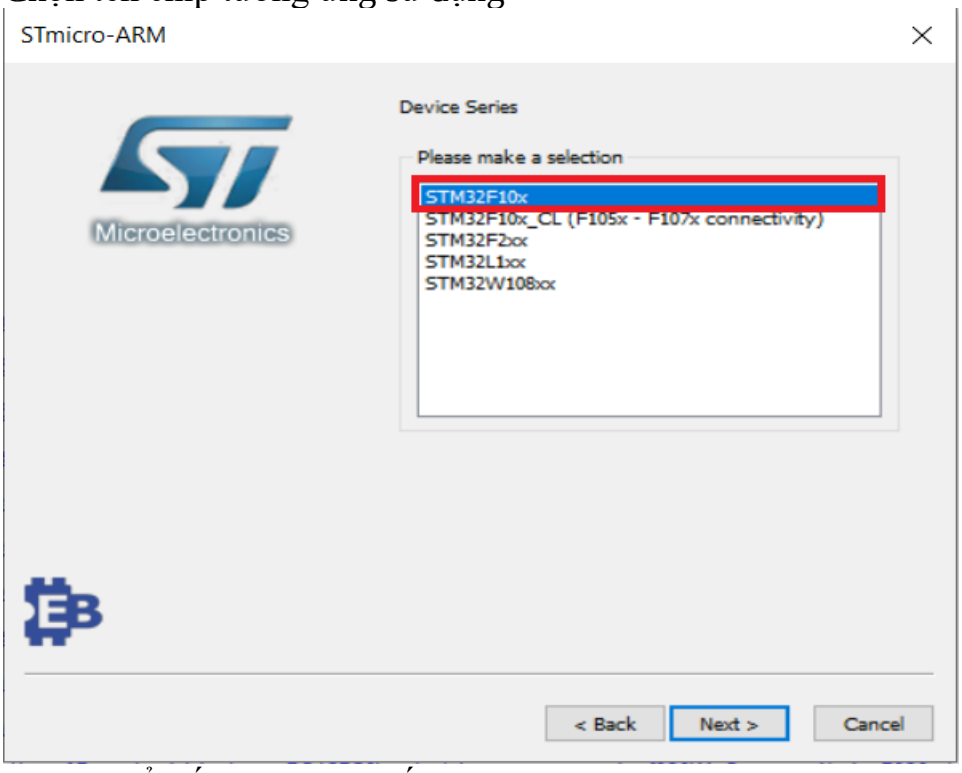
4. Chọn trình biên dich là ARM-GCC



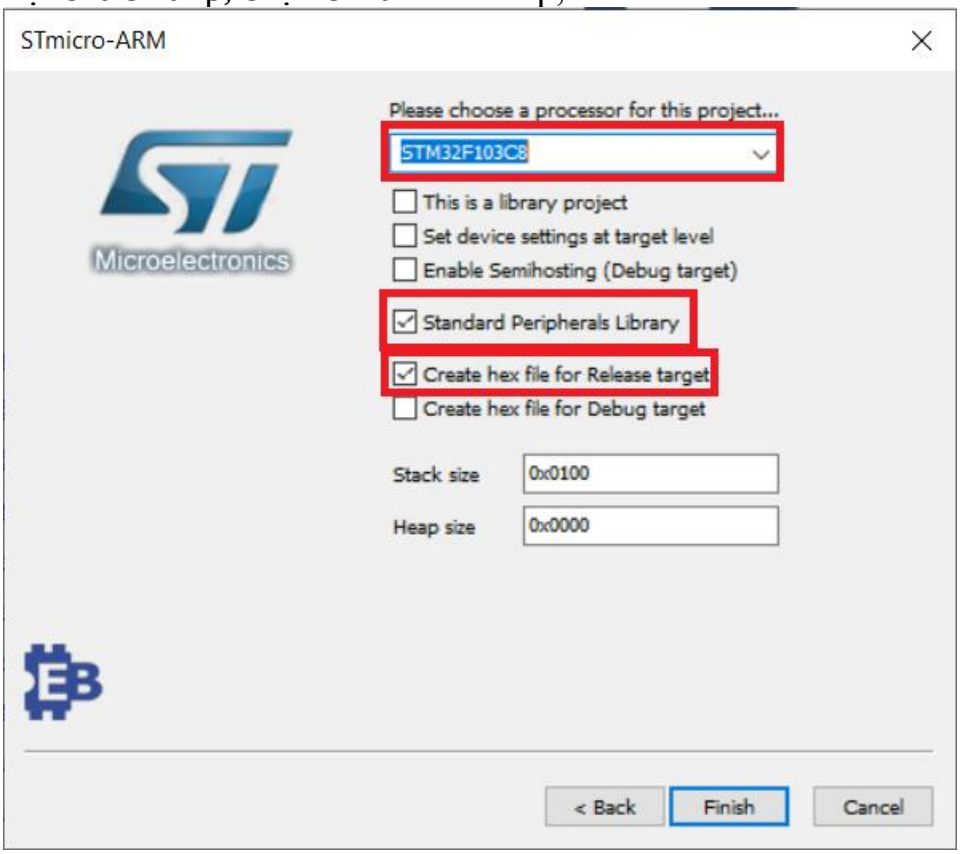
5. Chọn lõi ARM của chip, ở đây sử dụng STM32F1



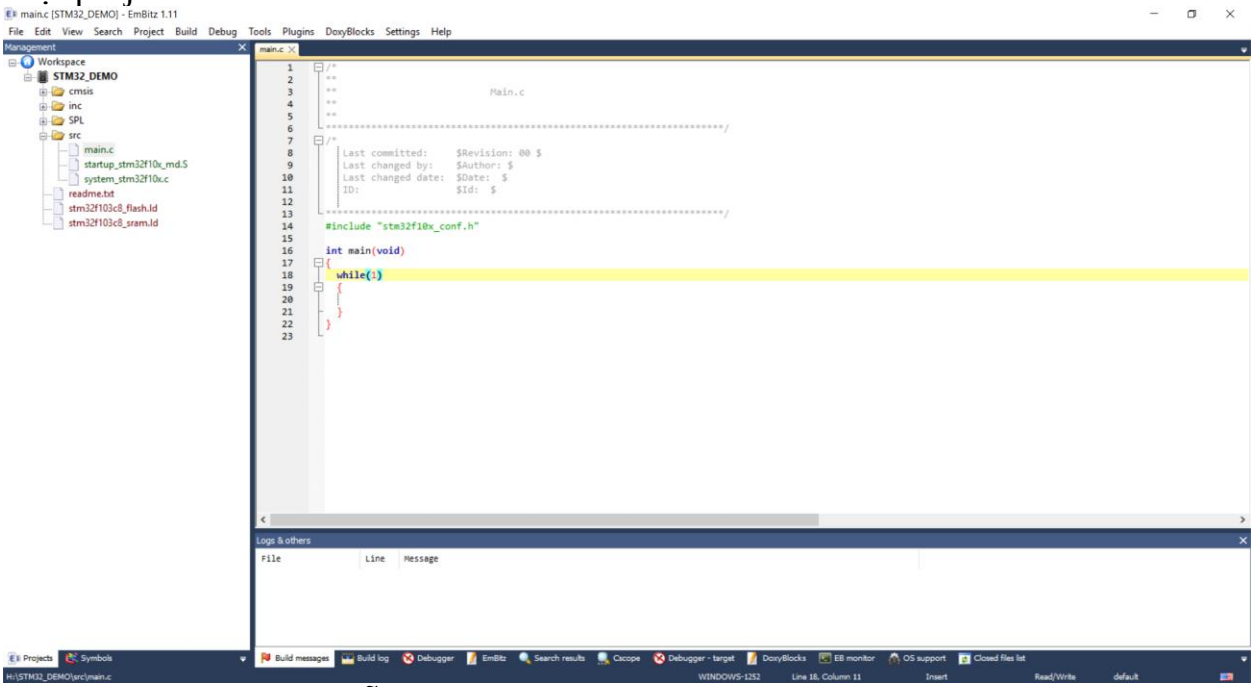
6. Chọn tên chip tương ứng sử dụng



7. Tại cửa sổ tiếp, chọn chi tiết tên chip, và tích như hình

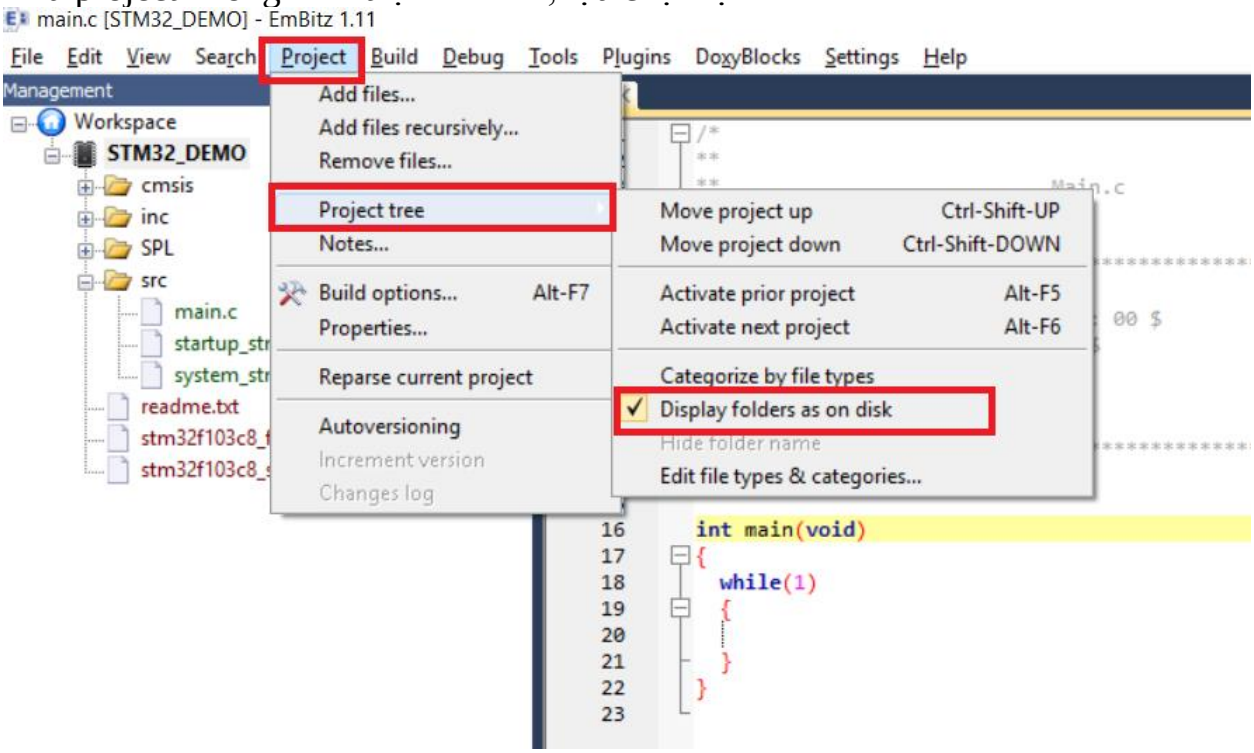


8. Một project hoàn chỉnh sẽ như sau

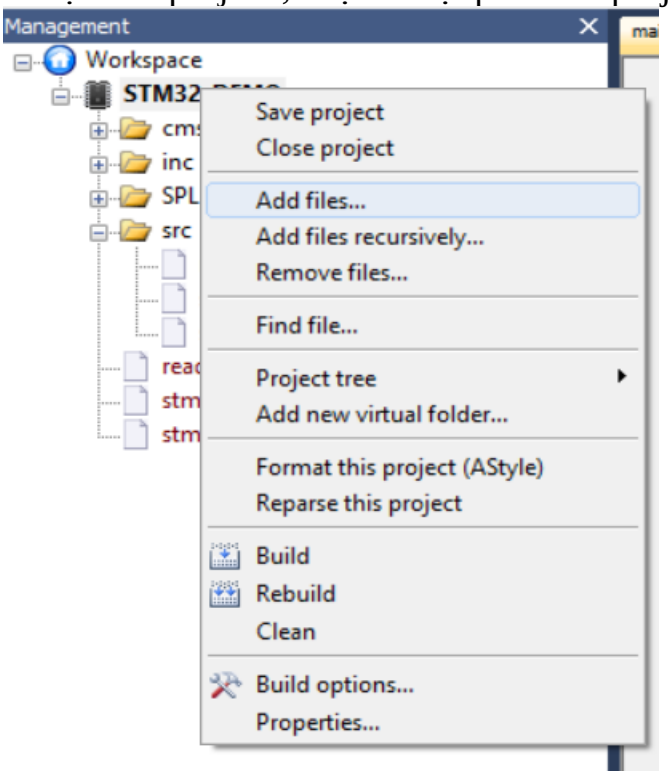


Lưu ý: mọi thư viện mẫu của hãng đã được mặc định add vào trong project.

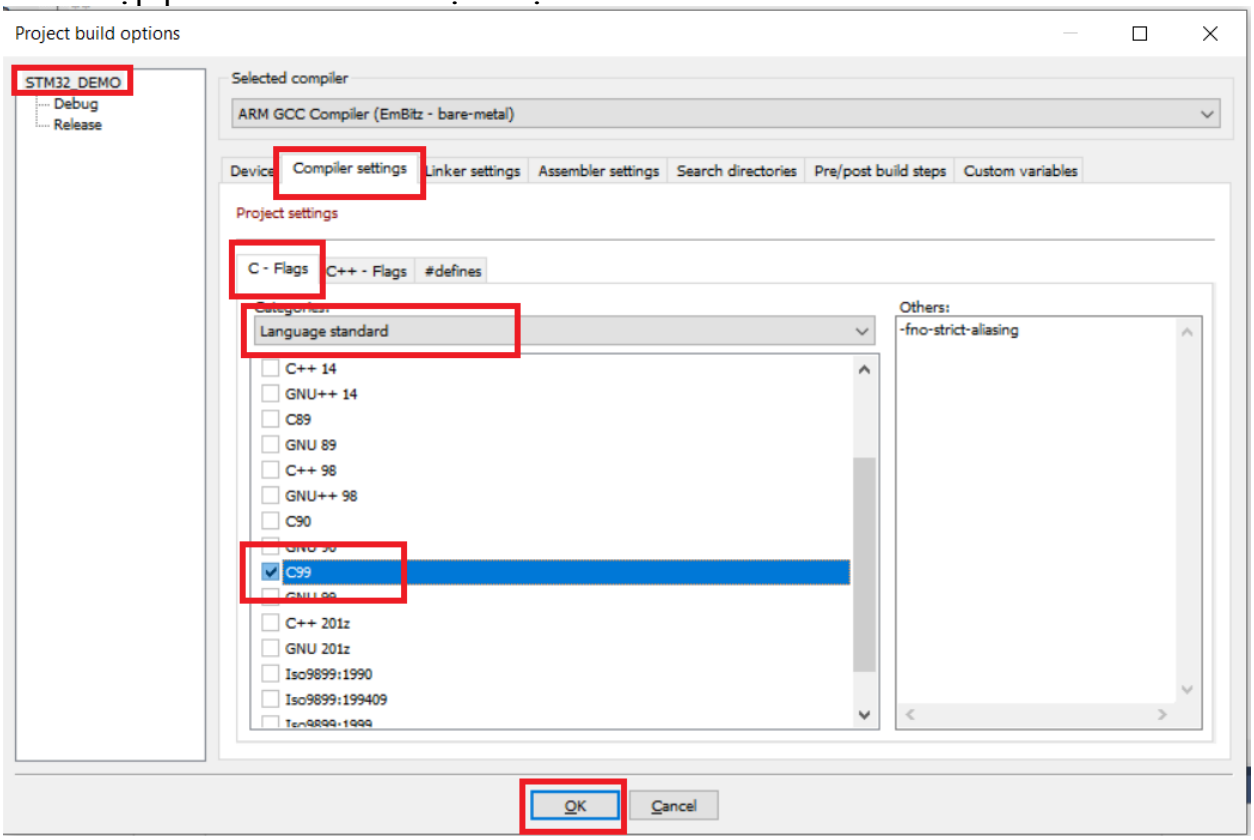
Nếu project không hiển thị như trên , lựa chọn lại như sau



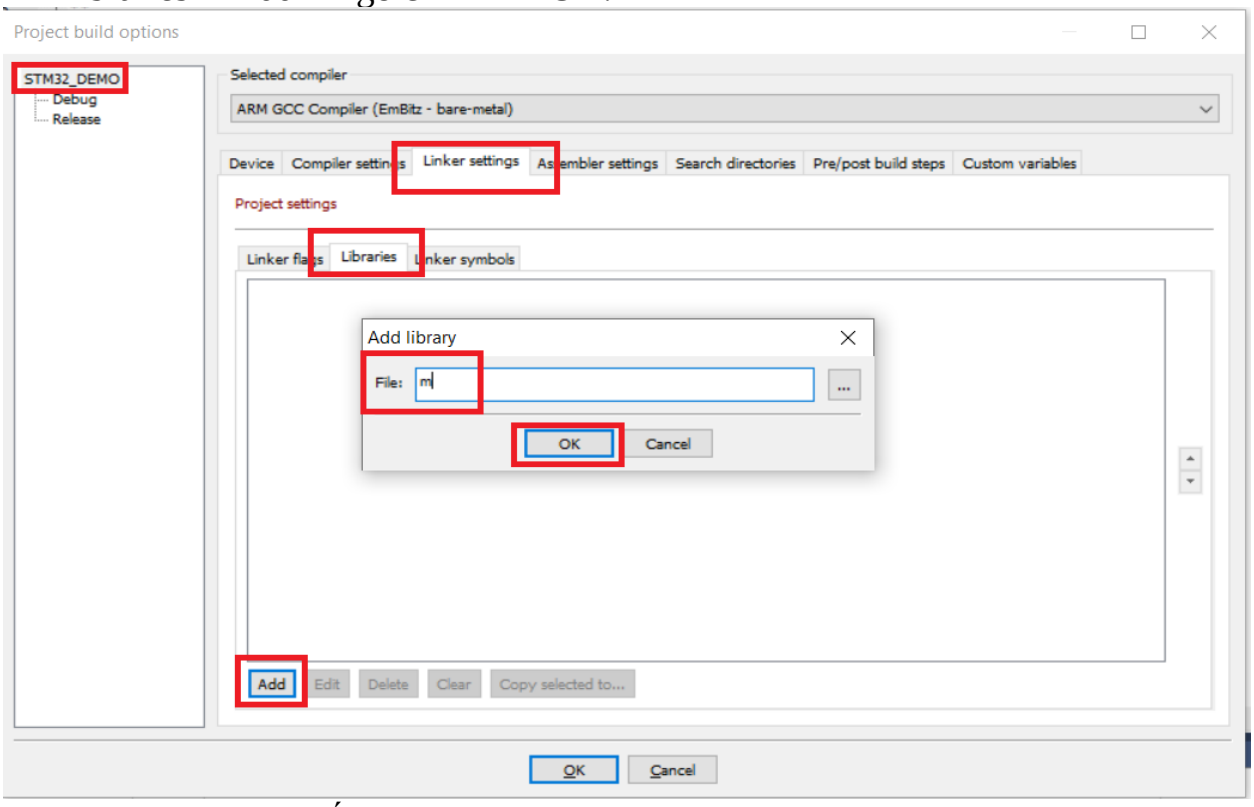
9. Để add thêm thư viện vào project , chọn chuột phải vào project >> add file



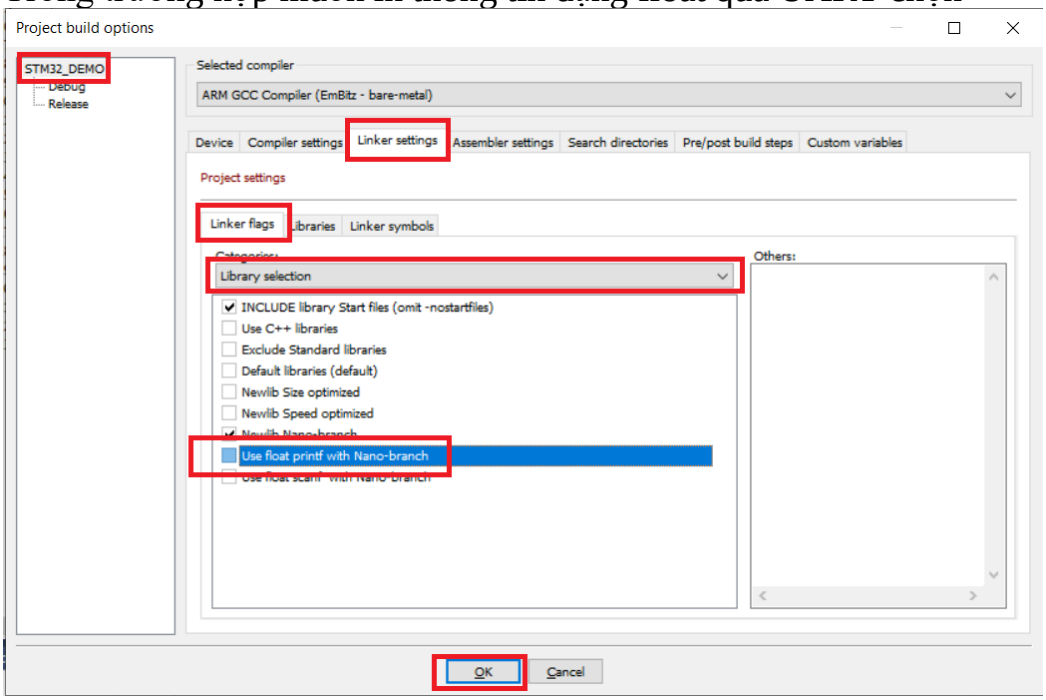
10. Ấn tổ hợp phím ALT+F7 và lựa chọn



11. Trong trường hợp muốn gọi thư viện toán học math.h , chọn Linker setting >> Libraries>>Add >> gõ chữ m >>OK:



12. Trong trường hợp muốn in thông tin dạng float qua UART chọn



2.3.Một số phần mềm nạp chương trình

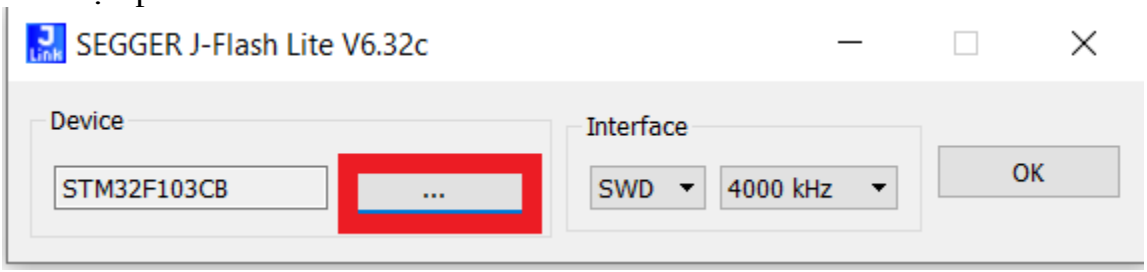
Sau khi build 1 project hoàn chỉnh , truy cập vào folder nơi lưu poject …./bin/Release/xxxx.hex. Đây là file để nạp vào KIT. Có 3 phương pháp nạp file hex (có thể file .bin,….) cho kit như sau:

2.3.1.Sử dụng mạch nạp Jlink & phần mềm JFlashLite.

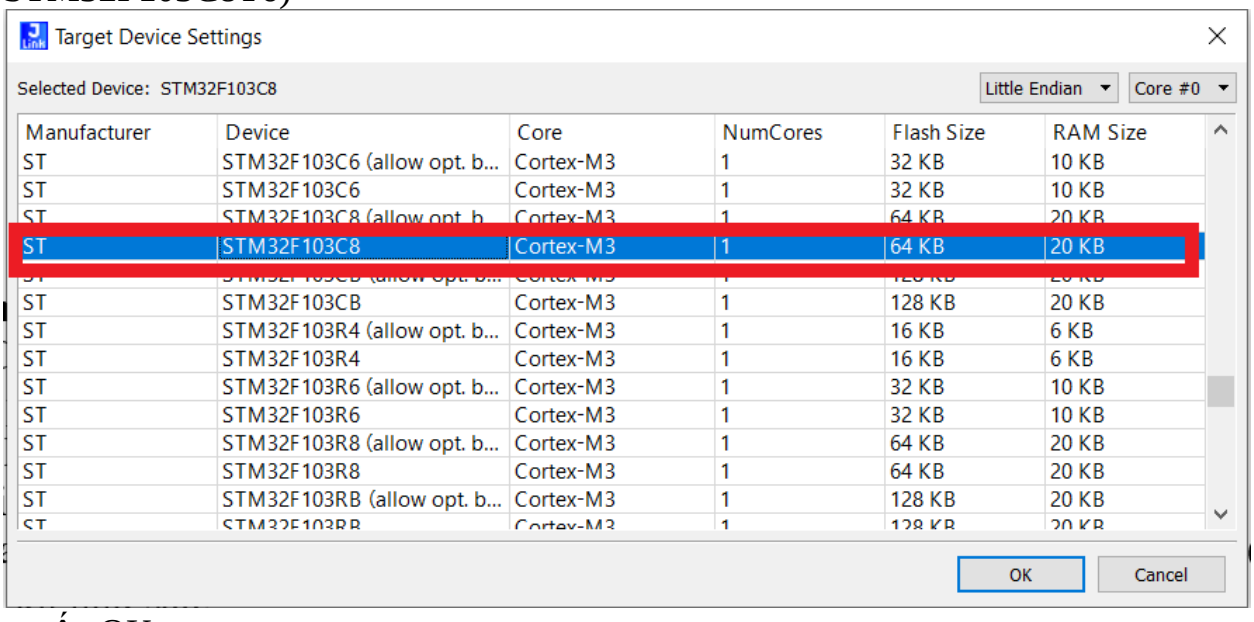
Tải phần mềm Jlink từ đường dẫn : https://www.segger.com/downloads/jlink/

Sau khi tải và cài đặt chương trình. Truy cập vào folder nơi cài chương trình . Ở đây tôi sử dụng có địa chỉ như :" C:\Program Files (x86)\SEGGER\JLink_V632c"

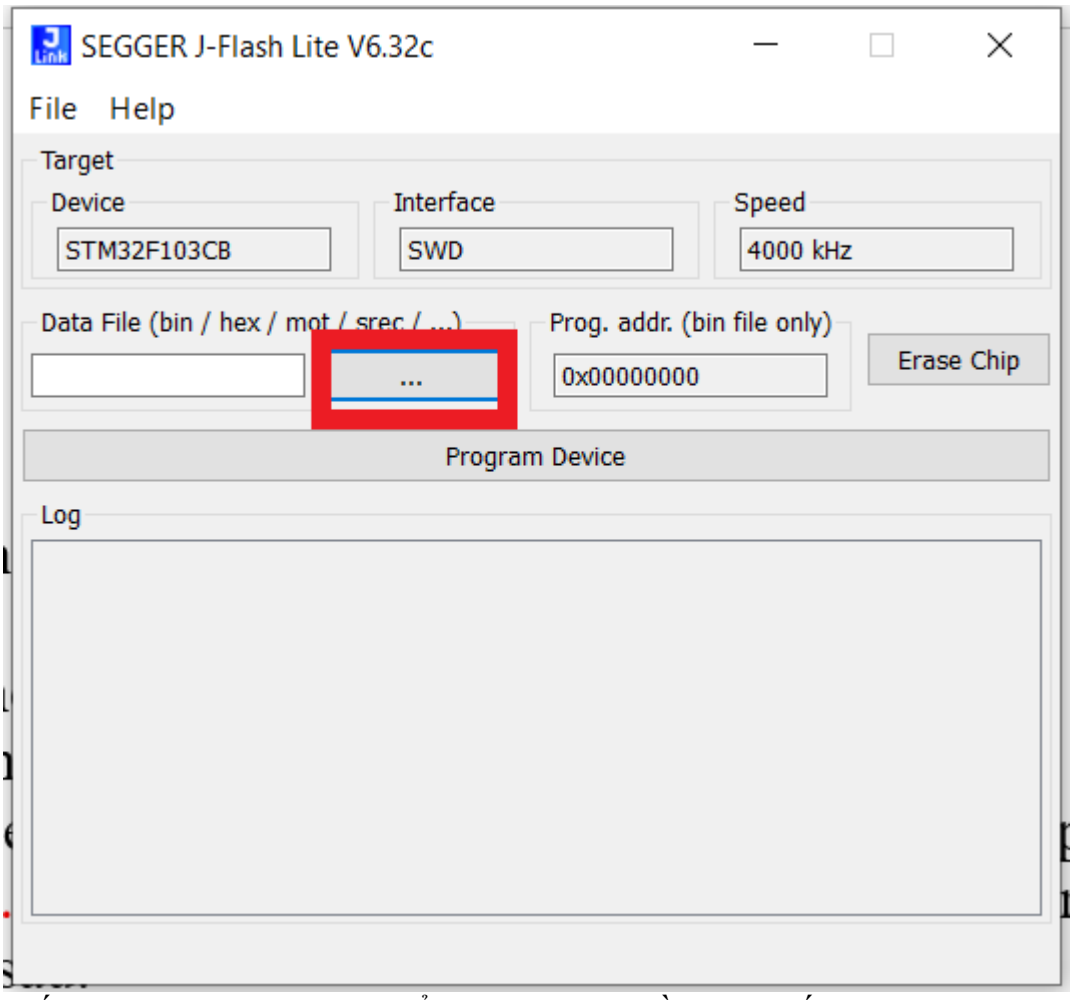
>> chọn phần mềm JFlashLite.exe.



Click vào phần khoanh đỏ và chọn đúng tên chip sử dụng (ở đây là STM32F103C8T6)



>> ấn OK



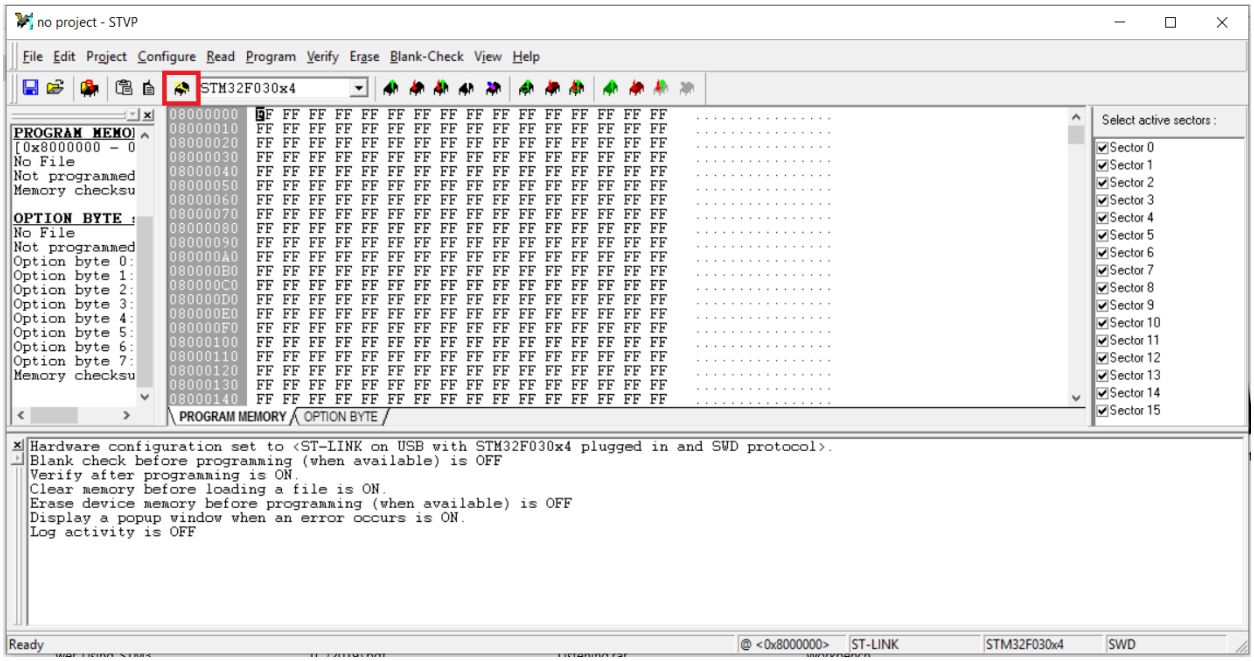
Tiếp tục lựa chọn file hex để nạp vào KIT bằng cách ấn vào như hình. Sau khi lựa chọn phù hợp với file hex,kết nối KIT với mạch nạp rồi ấn Program Device . Sơ đồ kết nối:

Jlink	KIT STM32F103C8T6
GND	GND
VCC	VCC
RST	RST
SWDIO	PA13

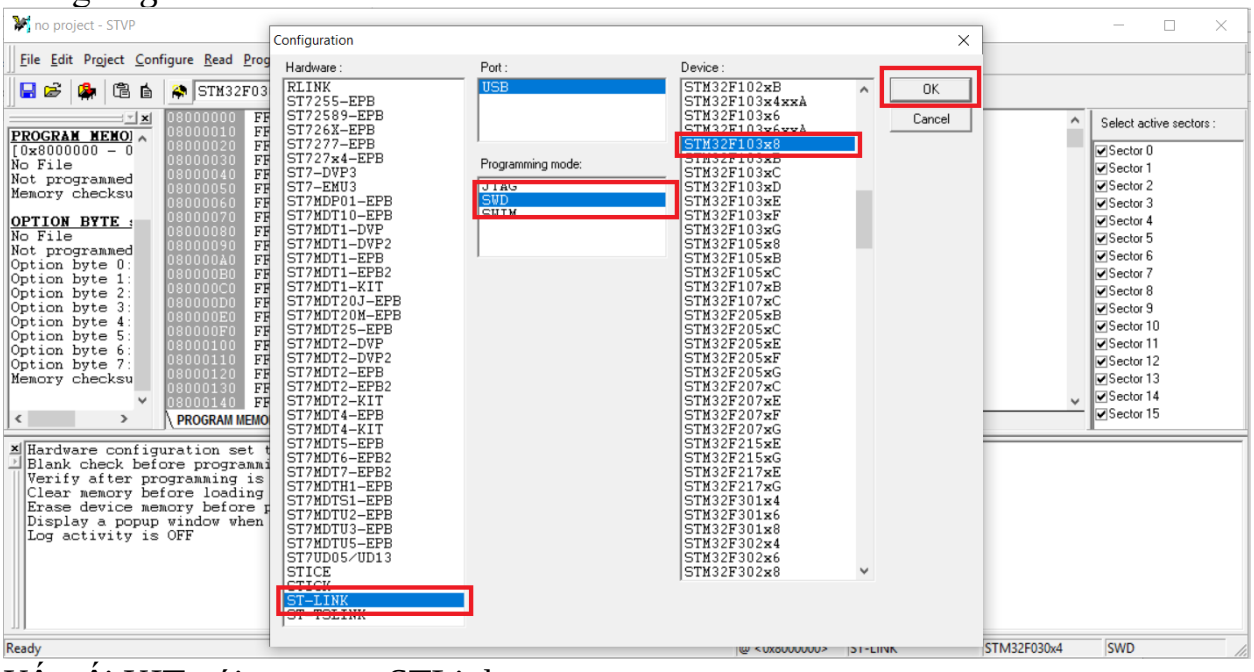
2.3.2. Sử dụng mạch nạp STlink & STVP Tool.

Truy cập : https://www.st.com/en/development-tools/stvp-stm32.html , tải và cài đặt phần mềm STVP. Sau khi cài đặt, phần mềm sẽ có Shortcut ngoài Desktop

Mở chương trình và chọn mạch nạp sử dụng. Click phần khoan đỏ



Ở đây chọn mạch nạp Stlink vào kiểu nạp là SWD . Cuối cùng là chọn tên chip tương ứng



Kết nối KIT với mạc nạp STLink

STlink	KIT STM32F103C8T6
GND	GND
VCC	VCC
RST	RST

SWDIO	PA13
SWCLK	PA14

Ấn tổ hợp Ctrl + O >> chọn đến file hex cần nạp >> Ấn Ctrl + P để nạp.

Lưu ý : Đối với mạch nạp Stlink mini trên thị trường sẽ không có chân reset, giữ nút reset >> ấn Ctrl+P >> nhả nút reset để nạp mạch.

2.3.3. Sử dụng bằng USB TTL và phần mềm Flash Loader

Truy cập đường dẫn sau để tải và cài đặt chương trình :

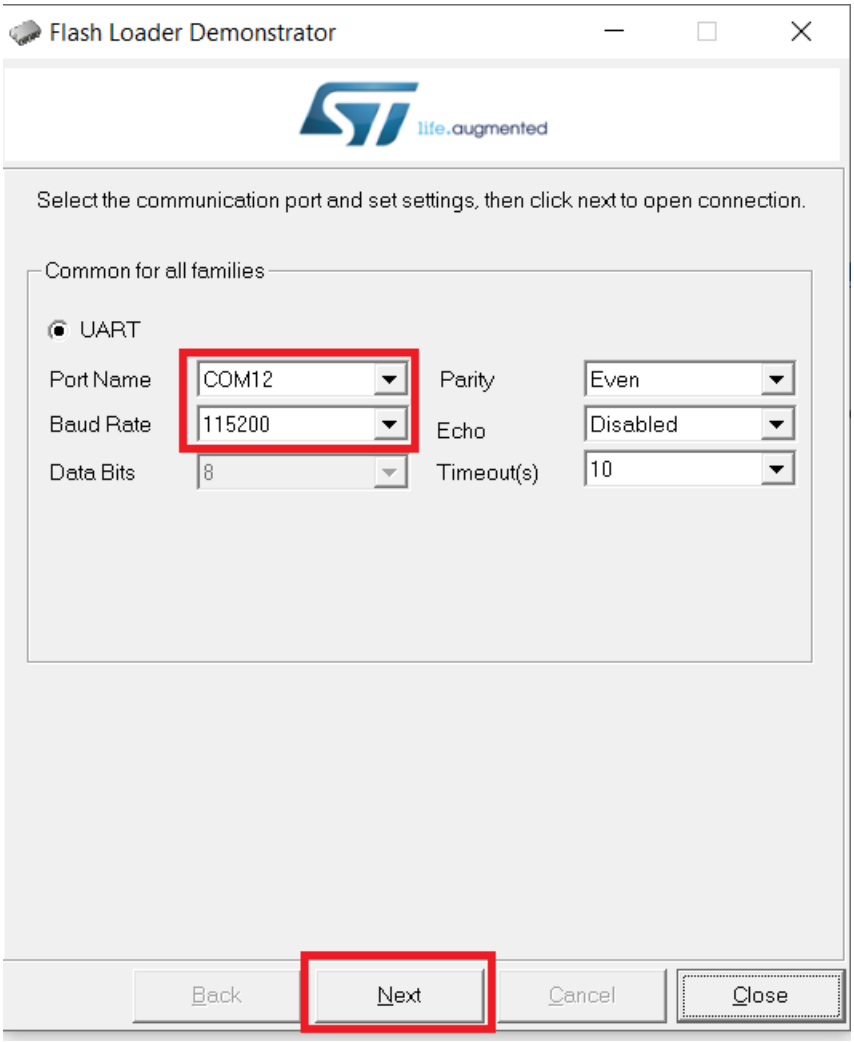
https://www.st.com/en/development-tools/flasher-stm32.html

Mở ứng dụng STMFlashLoader Demo.exe trong folder cài chương trình (ở đây tôi sử dụng là:" C:\Program Files (x86)\STMicroelectronics\Software\Flash Loader Demo")

Kết nối mạch nạp với KIT như sau :

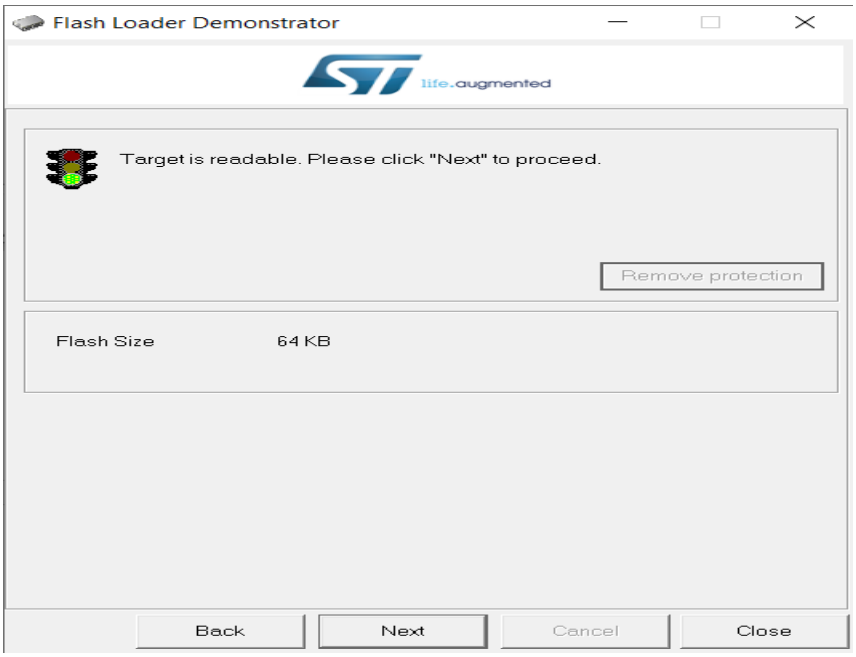
USB TTL	**KIT STM32F103C8T6**
GND	GND
VCC	VCC
RX	TX (PA9)
TX	RX (PA10)

Chọn tên cổng COM và baudrate tương ứng (baud càng cao thì tốc độ nạp càng nhanh tuy nhiên sẽ dễ gây lỗi cũng như 1 số USB TTL không hỗ trợ baud cao)

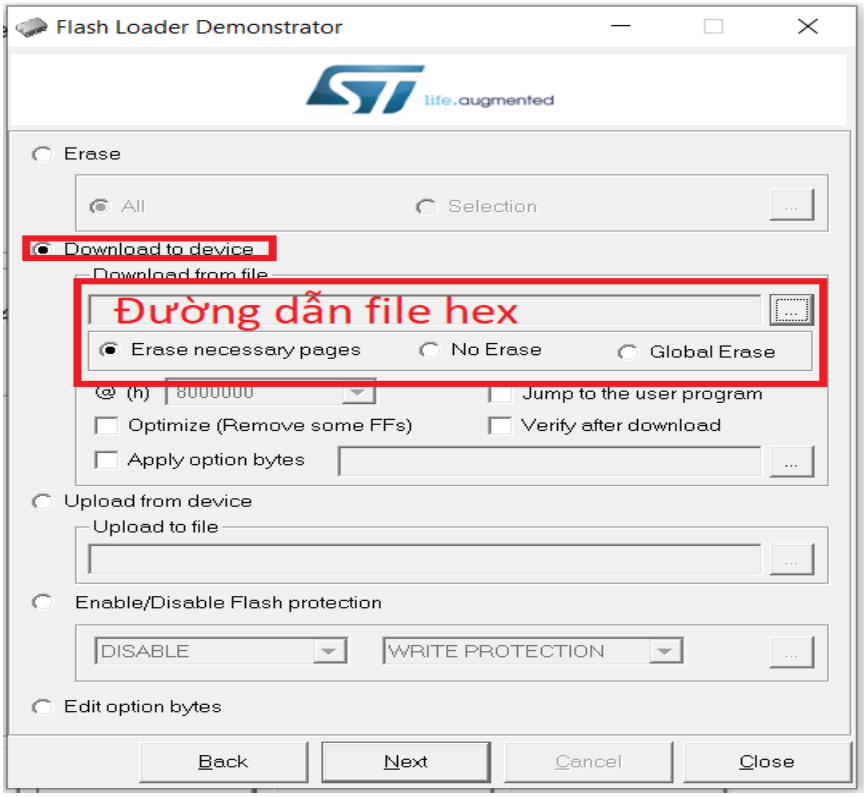


Chuyển BOOT loader cho KIT bằng cách : Ấn giữ lần lượt nút đỏ >> trắng . Sau đó nhả lần lượt nút đỏ >> trắng.

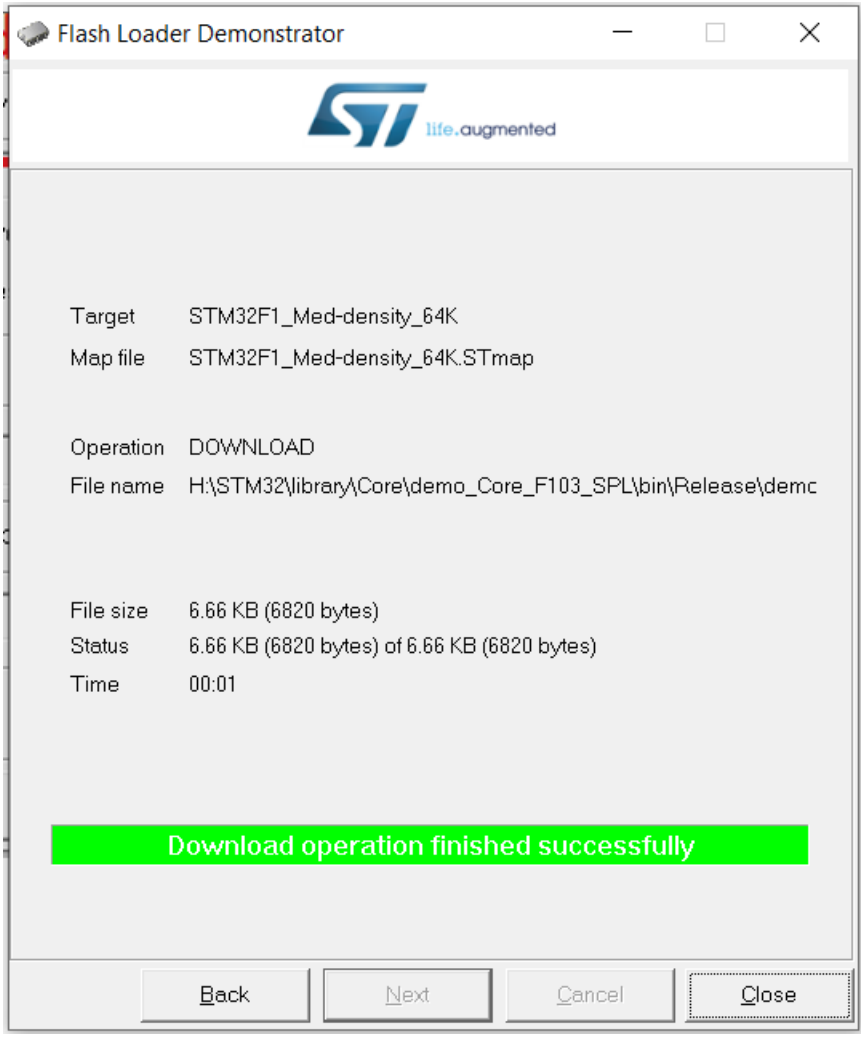
Chọn Next, nếu chuyển BOOT thành công sẽ hiện như sau



>> Next cho đến khi bảng sau thì chọn



>>Next. Nếu nạp thành công, cửa số sẽ báo



2.4.Hướng dẫn DEBUG

3.THỰC HÀNH

3.1. Lập trình GPIO

a.Lý thuyết:

- GPIO là từ viết tắt của General purpose I/O ports tạm hiểu là nơi giao tiếp chung giữa tín hiệu ra và tín hiệu vào.

- Ở STM32 thì các chân GPIO chia ra làm nhiều Port vd: PortA, PortB….. Số lượng Port phụ thuộc vào số lượng chân(pin) và cách gọi phụ thuộc vào nhà sản xuất(ví dụ VĐK X có PortA mà lại không có PortD). Mỗi Port thường có 16 chân đánh số từ 0 -> 15 tương ứng với mỗi chân là 1bit. Mỗi chân có 1 chức năng khác nhau như analog input, external interrupt.. hay đơn thuần chỉ là xuất tín hiệu on/off ở mức 0,1.

Các mode hoạt động:

- **Input floating**: cấu hình chân I/O là ngõ vào và để nổi.
- **Input pull-up:** cấu hình chân I/O là ngõ vào, có trở kéo lên nguồn.
- **Input-pull-down**: cấu hình chân I/O là ngõ vào, có trở kéo xuống GND.
- **Analog**: cấu hình chân I/O là Analog, dùng cho các mode có sử dụng ADC hoặc DAC.
- **Output open-drain**: cấu hình chân I/O là ngõ ra, khi output control = 0 thì chân I/O sẽ nối GND, còn khi output control = 1 chân I/O được để nổi (không có điện thế).
- **Output push-pull:** cấu hình chân I/O là ngõ ra, khi output control = 0 thì chân I/O sẽ nối GND, còn khi output control = 1 thì chân I/O được nối VCC.
- **Alternate function push-pull** : sử dụng chân I/O vừa là ngõ ra và vừa là ngõ vào, tuy nhiên sẽ không có trở kéo lên và kéo xuống ở input, chức năng output giống Output open-drain. Ngoài ra nó còn để sử dụng cho chức năng remap.

b. Thực hành

Bài 1: Lập trình led nhấp nháy tại chân PB2

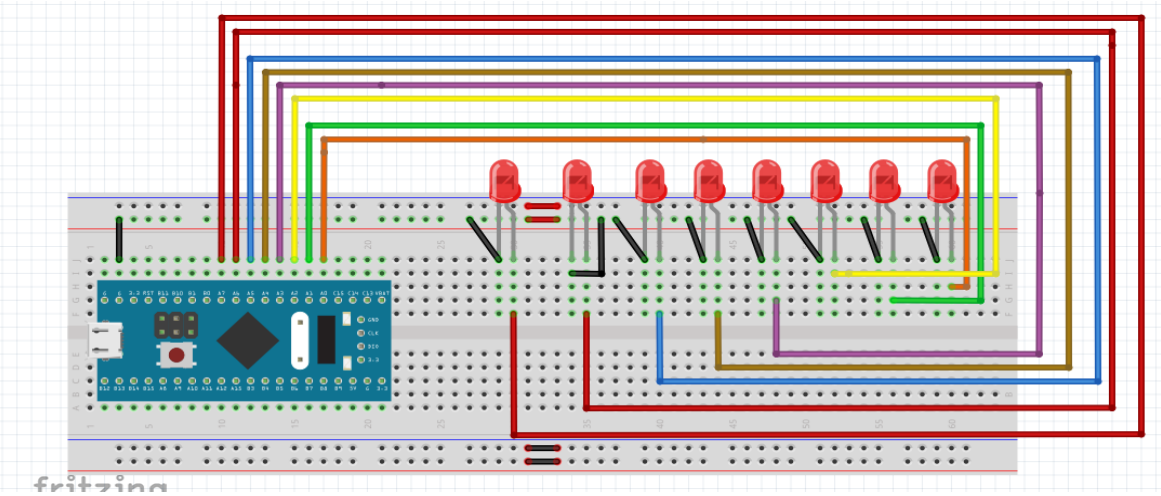
- Phân tích:
  - Để điều khiển led tại chân PB2 cần cấu hình các thông số của I/O
  - Lắp mạch: trên KIT STM32F103C8 đã tích hợp sẵn 1 led tại chân PB2, do vậy có thể sử dụng ngay mà không cần phải cắm mạch.
- Code tham khảo:

```
// Blink led
// include libarary
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

// khai bao ham
void Delay(unsigned int time);
GPIO_InitTypeDef GPIO_Struct;

int main(void){
 // cap clock cho GPIOB
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,
ENABLE);
 //cai dat cho chan PB2
 GPIO_Struct.GPIO_Mode = GPIO_Mode_Out_PP;
 GPIO_Struct.GPIO_Pin = GPIO_Pin_2 ;
 GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;
 GPIO_Init(GPIOB, &GPIO_Struct);
 while(1){
        GPIO_SetBits(GPIOB, GPIO_Pin_2);
        Delay_ms(1000);
        GPIO_ResetBits(GPIOB, GPIO_Pin_2);
        Delay_ms(1000);
 }
}
// ham delay chinh xac ms
void Delay_ms(uint16_t time)
{
```

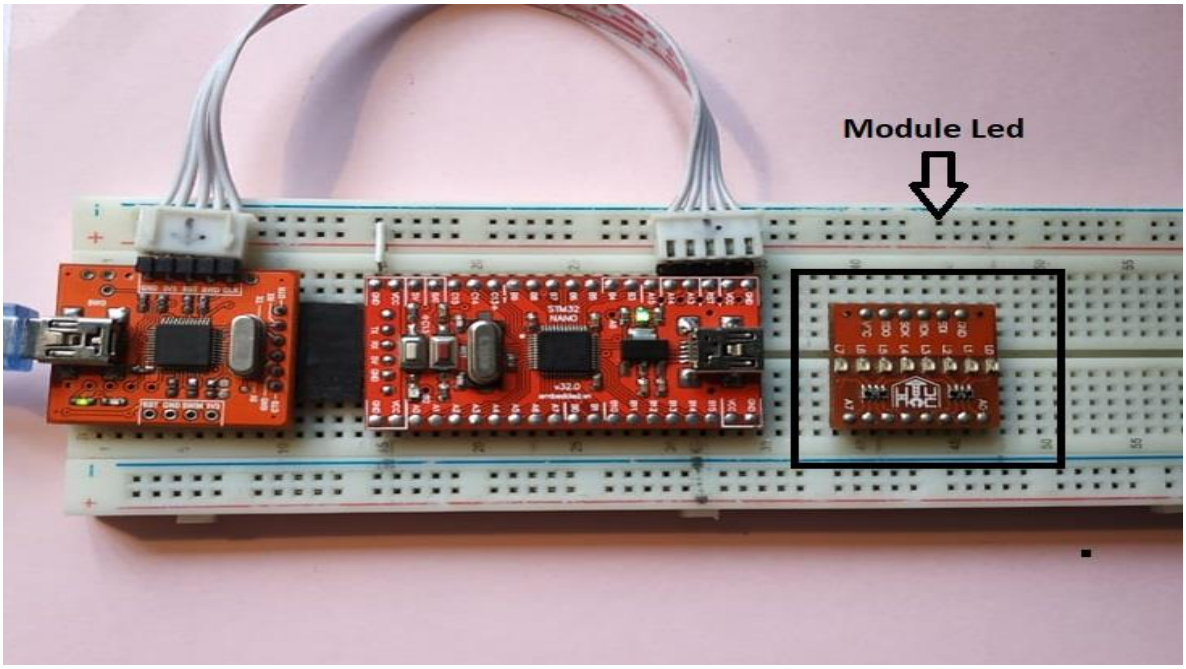
```
 // tang bien dem len 12000 lan
 uint32_t time_n=time*12000;
 // cho den khi biem time_n giam =0 thi thoat
 while(time_n!=0){time_n--;}
}
```

Bài 2: Lập trình hiệu ứng 8 led

- Phân tích:
  - Sau khi đã điều khiển được 1 led theo ý muốn có thể điều khiển 8 led với các hiệu ứng khác nhau.
  - Các thức điều khiển tương tự khi điều khiển 1 led.
  - Do số lượng led cần điều khiển lớn, ngoài cách sử dụng câu lệnh: ‘GPIO_SetBits()’ và ‘GPIO_ResetBits’ còn có thể xác lập mức tín hiệu đầu ra trực tiếp bằng cách tác động đến thanh ghi ‘GPIOA->ODR = “bit_gia_tri” ’
- Lắp mạch:



Hoặc có thể sử dụng module led tích hợp sẵn 8 led:



- Code tham khảo

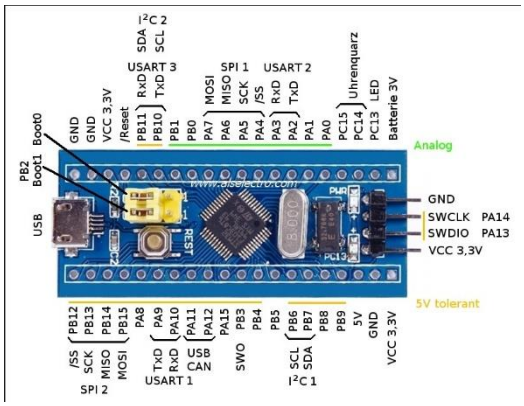
```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

void Delay(unsigned int time);
GPIO_InitTypeDef GPIO_Struct;

int main(void){
// cap clock cho GPIOA
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,
ENABLE);
//cai dat cho chan Port A
GPIO_Struct.GPIO_Mode = GPIO_Mode_Out_PP;
```

```
GPIO_Struct.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6 |
GPIO_Pin_7;
GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_Struct);
while(1){
        GPIOA->ODR = 0x00AA;
// 0x00AA = 1010 1010 tuong ung cac led vi tri le sang
        Delay(1000);
        GPIOA->ODR = 0x0055;
// 0x0055 = 0101 0101 tuong ung cac led vi tri chan sang
        Delay(1000);
}
}
void Delay(unsigned int time){

unsigned int i,j;

for(i=0;i<time;i++){
        for(j=0;j< 0x2AFF; j++);
}
}
```

3.2. Lập trình truyền thông nối tiếp UART

a. Lý thuyết:

- UART - Universal synchronous asynchronous receiver transmitter là một ngoại vi cơ bản và thường dùng trong các quá trình giao tiếp với các module như : Xbee, Wifi, Blutooth…. Khi giao tiếp UART kết hợp với các IC giao tiếp như MAX232CP, SP485EEN…. thì sẽ tạo thành các chuẩn giao tiếp RS232, RS485. Đây là các chuẩn giao tiếp thông dụng và phổ biến trong công nghiệp từ trước đến nay.
- Khi ta sử dụng chân UART_CLK thì giao tiếp UART sẽ trở thành giao tiếp đồng bộ và không dùng sẽ là chuẩn giao tiếp không đồng bộ. Các bạn để ý là với bất cứ 1 chuẩn truyền thông nào, khi có sử dụng 1 chân tín hiệu làm chân CLK thì chuẩn giao tiếp đó sẽ là chuẩn giao tiếp đồng bộ và ngược lại. Ở đây mình chỉ đề cập đến giao tiếp UART không đồng bộ.
- Ưu điểm của giao tiếp UART không đồng bộ: tiết kiệm chân vi điều khiển(2 chân), là ngoại vi mà bất kì 1 VĐK nào cũng có, có khá nhiều module, cảm biến dùng UART để truyền nhận data với VĐK. Nhược điểm của loại ngoại vi này là tốc độ khá chậm, tốc độ tối đa tùy thuộc vào từng dòng; quá trình truyền nhận dễ xảy ra lỗi nên trong quá trình truyền nhận cần có các phương pháp để kiểm tra(thông thường là truyền thêm bit hoặc byte kiểm tra lỗi). UART không phải là 1 chuẩn truyền thông, Khi muốn nó là 1 chuẩn truyền thông hoặc truyền data đi xa, chúng ta cần phải sử dụng các IC thông dụng để tạo thành các chuẩn giao tiếp đáng tin cậy như RS485 hay RS232....
- Thông thường chúng ta sẽ dùng ngắt nhận UART để nhận dữ liệu vì sử dụng ngắt sẽ tiện lợi, không tốn thời gian chờ cũng như mất dữ liệu.Các tốc độ thường dùng để giao tiếp với máy tính: 600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200.
- Một số phần mềm giao tiếp với máy tính: hercules_3-2-5,teraterm, Serial-Oscilloscope-v1.5...Một số modulde dùng để giao tiếp với máy tính: CP2102 USB 2.0, USB ra UART dùng PL2303, USB to UART dùng TTL FT232RL, USB ra UART dùng CH340G…



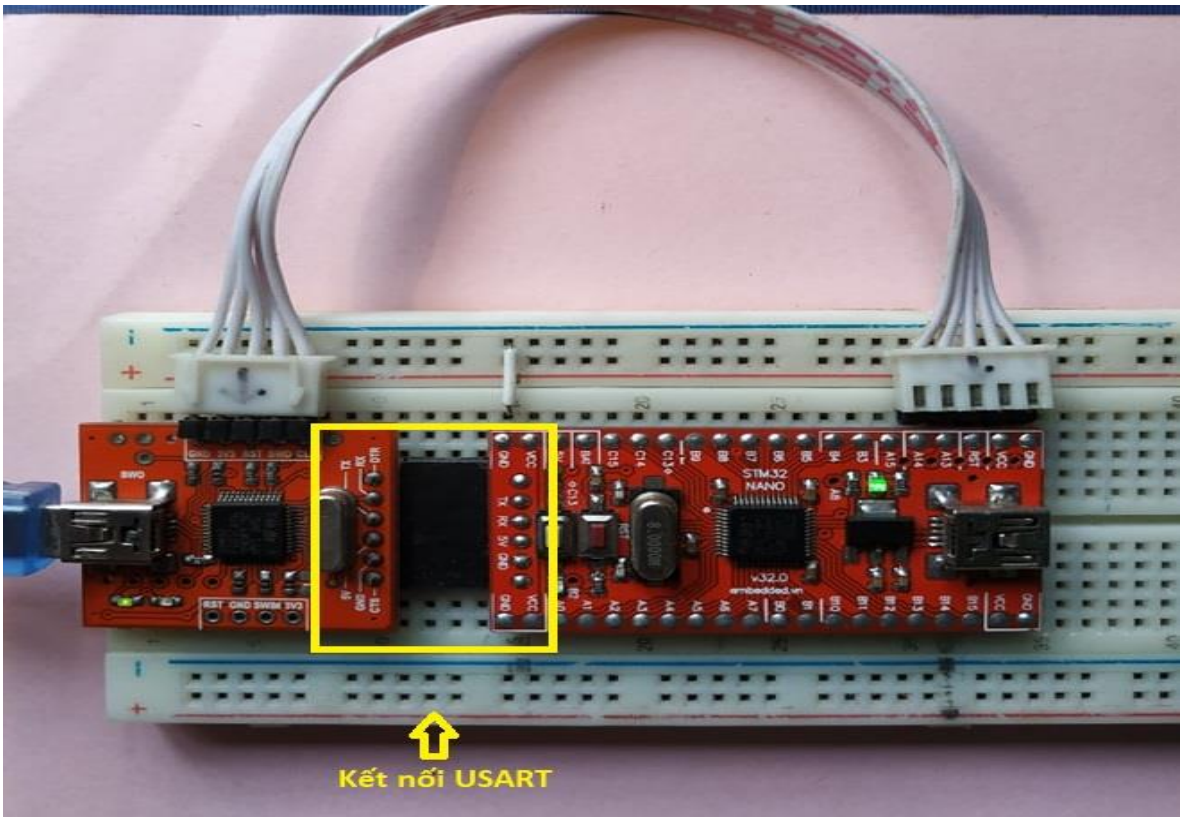
b. Kết nối KIT:

- KIT STM32F103C8 sử dụng bộ USART 1 tương ứng với 2 chân

  PA9 ----- TX

  PA10 ----- RX

- Cách kết nối:



c. Thực hành:

Bài 1: Truyền nhận ký tự lên màn hình Serial

Code tham khảo:

```
#include "stm32f10x.h"

/*Khoi tao bien cau hinh*/
GPIO_InitTypeDef                GPIO_InitStructure;
USART_InitTypeDef               UART_InitStructure;
NVIC_InitTypeDef                NVIC_InitStructure;


void GPIO_Configuration(void);
void Delay_ms(uint16_t time);
void UART_Configuration (void);

int main(void)
{
        GPIO_Configuration();
        UART_Configuration();
  while (
  }
}

void GPIO_Configuration(void)
{
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 ;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_Init(GPIOB, &GPIO_InitStructure);
}
void Delay_ms(uint16_t time)
{
uint32_t time_n=time*12000;
        while(time_n!=0){time_n--;}

}
void UART_Configuration (void)
```

```
	{
		/*Cap clock cho USART và port su dung*/
			RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,
ENABLE);
/* Cau Tx mode AF_PP, Rx mode FLOATING  */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/*Cau hinh USART*/
UART_InitStructure.USART_BaudRate = 19200;
UART_InitStructure.USART_WordLength = USART_WordLength_8b;
UART_InitStructure.USART_StopBits = USART_StopBits_1;
UART_InitStructure.USART_Parity = USART_Parity_No;
UART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
UART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USART2, &UART_InitStructure);

/* Cau hinh vector ngat va muc uu tien */
NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

/* xoa co ngat nhan cho lan dau su dung*/
USART_ClearFlag(USART2, USART_IT_RXNE);

/* Cau hinh cho phep ngat nhan*/
USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);

/* Cho phep UART hoat dong */
USART_Cmd(USART2, ENABLE);
```

}

Bài 2: In dòng chữ Hello World lên màn hình serial:

Code tham khảo:

```
#include "stm32f10x.h"
#include "stdio.h"

#ifdef __GNUC__
  /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small
printf
     set to 'Yes') calls __io_putchar() */
  #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
        /* Place your implementation of fputc here */
        /* e.g. write a character to the LCD */
        //lcd_Data_Write((u8)ch);
        USART_SendData(USART2,(u8)ch);

        /*Loop until the end of transmission */
        while (USART_GetFlagStatus(USART2,USART_FLAG_TC)==RESET)
        {}

        return ch;
}

/*Khoi tao bien cau hinh*/
GPIO_InitTypeDef                GPIO_InitStructure;
USART_InitTypeDef               UART_InitStructure;

uint8_t data =10;

void GPIO_Configuration(void);
void Delay_ms(uint16_t time);
void UART_Configuration (void);
```

```
int main(void)
{
	GPIO_Configuration();
	UART_Configuration();
  while (1)
  {
		printf("Hello World");
		Delay_ms(1000);

  }
}

void GPIO_Configuration(void)
{
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 ;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_Init(GPIOB, &GPIO_InitStructure);
}
void Delay_ms(uint16_t time)
{
uint32_t time_n=time*12000;
	while(time_n!=0){time_n--;}

}
void UART_Configuration (void)
	{
		/*Cap clock cho USART và port su dung*/
			RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,
ENABLE);
			RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART1,
ENABLE);

			/* Cau Tx mode AF_PP, Rx mode FLOATING  */
			GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
			GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
			GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
                GPIO_Init(GPIOA, &GPIO_InitStructure);

                GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
                GPIO_InitStructure.GPIO_Mode =
GPIO_Mode_IN_FLOATING;
                GPIO_Init(GPIOA, &GPIO_InitStructure);

            /*Cau hinh USART*/
                UART_InitStructure.USART_BaudRate = 115200;
                UART_InitStructure.USART_WordLength =
USART_WordLength_8b;
                UART_InitStructure.USART_StopBits = USART_StopBits_1;
                UART_InitStructure.USART_Parity = USART_Parity_No;
                UART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
                UART_InitStructure.USART_Mode = USART_Mode_Rx |
USART_Mode_Tx;
                USART_Init(USART2, &UART_InitStructure);

                /* Cho phep UART hoat dong */
                USART_Cmd(USART2, ENABLE);

    }
```

3.3. Ngắt ngoài(NVIC)

a. Lý thuyết:

- NVIC - Nested vectored interrupt controller là bộ vector ngắt lồng nhau. Nghĩa là chúng ta có thể sử dụng kết hợp nhiều ngắt trong một chương trình. Ngắt là một phần quan trọng và thiết yếu của chương trình. Nếu không có ngắt thì chương trình sẽ thực hiện theo 1 trình tự từ trên xuống dưới mà không có bất kì sự can thiệp nào. Điều đó là bất lợi khi có 1 tác động ngoài xảy ra, chương trình sẽ không xử lí kịp thời dẫn đến việc bỏ qua tác động đó. Ngắt ra đời để phục vụ cho các sự cố đó.

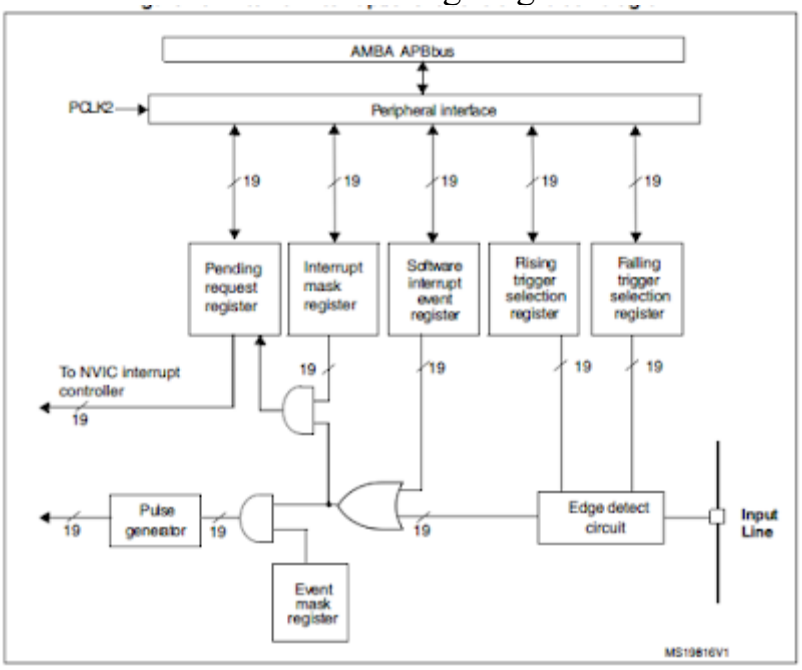
- Một số thông số ngắt chính của STM32F103:

- 16 mức ưu tiên có thể lập trình được.
- Độ trễ thấp (xảy ra ngắt cực kì nhanh).
- Có quản lí năng lượng cho vector ngắt.
- Có các thanh ghi điều khiển quá trình ngắt.
- 68 vector ngắt(xem thêm trong reference manual).

- Ngắt ngoài nằm trong 1 phần của ngắt NVIC. Mỗi EXTI – interrupt/event controller có thể được lập trình chọn loại sự kiện/ ngắt, chọn cạnh lên, cạnh xuống hoặc cả 2, mức ưu tiên ngắt.
- Một số tính năng chính của ngắt ngoài:

- Kích hoạt độc lập và mặt nạ cho mỗi line sự kiện/ngắt.
- Có bit trạng thái riêng cho mỗi line ngắt.
- Có thể có tối đa 20 sự kiện/ ngắt, tham khảo thêm trong reference manual.
- Kiểm tra tín hiệu ngoài có độ rộng xung nhỏ hơn clock trên APB2.

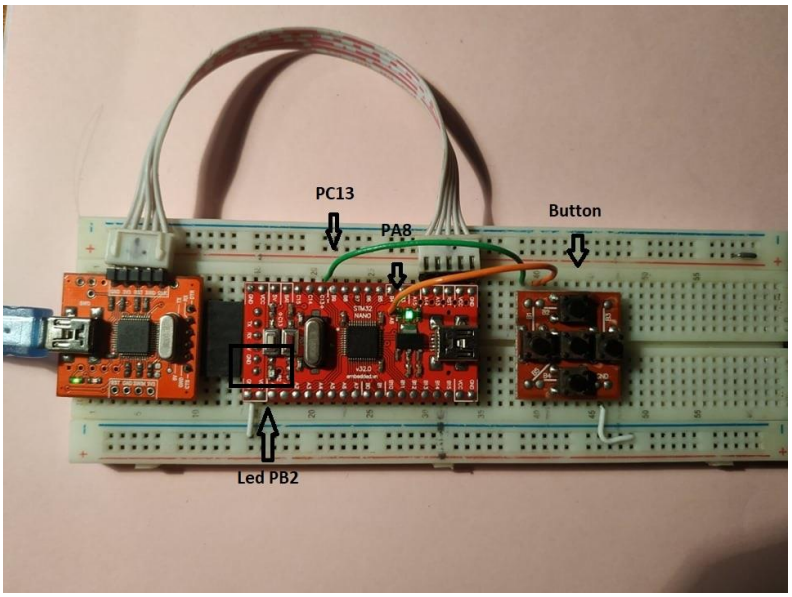
- Sơ đồ khối của các khối điều khiển ngắt ngoài:



b. Code tham khảo

Bài 1:

- Lắp mach:



- Code tham khảo:

```
#include "stm32f10x.h"

void LEDconfig(void);
void GPIO_EXTI(void);
void configEXTI(void);
void NVIC_EXTI(void);
void EXTI9_5_IRQHandler(void);
void EXTI15_10_IRQHandler(void);

// cau hinh cho led chan B2
void LEDconfig(void)
{
 GPIO_InitTypeDef GPIO_InitStructure;
 RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOB,
ENABLE);

 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
 GPIO_Init(GPIOB, &GPIO_InitStructure); //cai dat cac cau hinh tren
cho GPIOB
}

//cau hinh ngat ngoai cho 2 chan A8 va C13
```

```
// B1:cap xung va cau hinh chan A8 va C13 nhan tin hieu ngat ngoai la ngo vao keo len
//                              (cau hinh GPIO)

void GPIO_EXTI(void)
{
	GPIO_InitTypeDef GPIO_InitStructure;

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOC | RCC_APB2Periph_AFIO, ENABLE);
	//do ngat ngoai la chuc nang thay the nen phai bat AIFO

	//cau hinh chan A8
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	//cau hinh chan C13
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
	GPIO_Init(GPIOC, &GPIO_InitStructure);

	//chon chan A8 va C13 la chan nhan tin hieu ngat ngoai
	GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource8);
	GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource13);
}

// B2:Cau hinh va cho phep ngat ngoai o EXTI

void configEXTI(void)
{
 EXTI_InitTypeDef EXTI_InitStructure;

 EXTI_InitStructure.EXTI_Line = EXTI_Line8 | EXTI_Line13; // chon kenh 8 va kenh 13 ung voi A8 va C13
 EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt; //chon che do ngat ngoai
```

```
 EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling; //chon
canh tich cuc la canh xuong
 EXTI_InitStructure.EXTI_LineCmd = ENABLE; //cho phep kenh
ngat ngoai duoc cau hinh
 EXTI_Init(&EXTI_InitStructure); //lenh cau hinh cac thong so duoc
luu trong EXTI_InitStructure
}

// B3: cau hinh cap do uu tien va cho phep ngat ngoai o NVIC

void NVIC_EXTI()
{
 NVIC_InitTypeDef NVIC_InitStructure;

 NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);   //0 cap
PreemptionPriority va 16 cap SubPriority
 NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
          // chon cac kenh tu 5-9
 NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //chon thu tu
uu tien
 NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; // cho phep
ngoai vi dang dc cau hinh o NVIC
 NVIC_Init(&NVIC_InitStructure); // lenh cau hinh cac thong so duoc
luu trong NVIC_InitStructure cho NVIC

 NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
 //chon kenh tu 10 den 15
 NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; //chon muc
uu tien
 NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
 NVIC_Init(&NVIC_InitStructure);
}

// B4: Viet chuong trinh con phuc vu ngat ngoai

// chuong trinh con phuc vi ngat ngoai cho chan A8
void EXTI9_5_IRQHandler(void)
{
 if(EXTI_GetITStatus(EXTI_IMR_MR8) != RESET)
 //kiem tra co phai la kenh 8 ngat khong?
```

```
 {
                EXTI_ClearITPendingBit(EXTI_IMR_MR8); //xoa co
ngat kenh 8
                GPIO_SetBits(GPIOB, GPIO_Pin_2); //mo Led
 }
}

//chuong trinh con phuc vu ngat ngoai cho chan C13
void EXTI15_10_IRQHandler(void)
{
 if(EXTI_GetITStatus(EXTI_IMR_MR13) != RESET)
        //kiem tra co phai la kenh 13 ngat khong
 {
                EXTI_ClearITPendingBit(EXTI_IMR_MR13); //xoa co
ngat kenh 13
                GPIO_ResetBits(GPIOB, GPIO_Pin_2); //tat Led
 }
}

int main(void)
{
 SystemInit();
 LEDconfig();
 GPIO_EXTI();
 configEXTI();
 NVIC_EXTI();
 EXTI9_5_IRQHandler();
 EXTI15_10_IRQHandler();
 while(1);
}
```

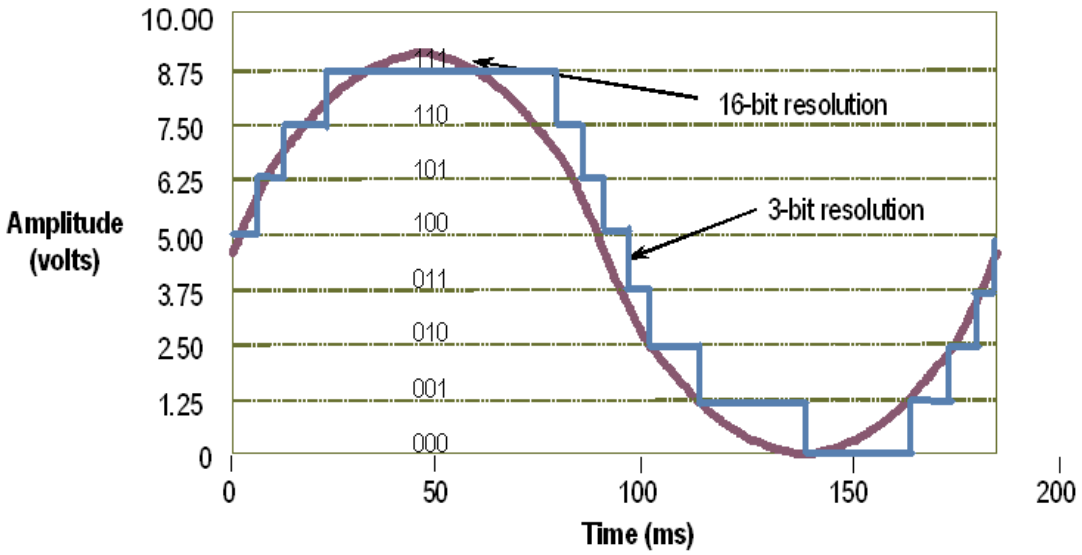
3.4. Bộ chuyển đổi Analog – Digital (ADC)

a. Lý thuyết:

- Trong các ứng dụng vi điều khiển – hệ thống nhúng, bộ chuyển đổi tương tự-số (ADC) là 1 thành phần rất quan trọng để có thể chuyển đổi các dữ liệu dạng analog từ môi trường (nhiệt độ, độ ẩm, độ sáng,…) sang dạng digital để vi điều khiển có thể xử lý được. STM32F103C8 có tích hợp sẵn các bộ chuyển đổi ADC với độ phân giải 12bit. Có 12 kênh cho phép đo tín hiệu từ 10 nguồn bên ngoài và 2 nguồn nội bên trong. Trong bài này, chúng ta sẽ

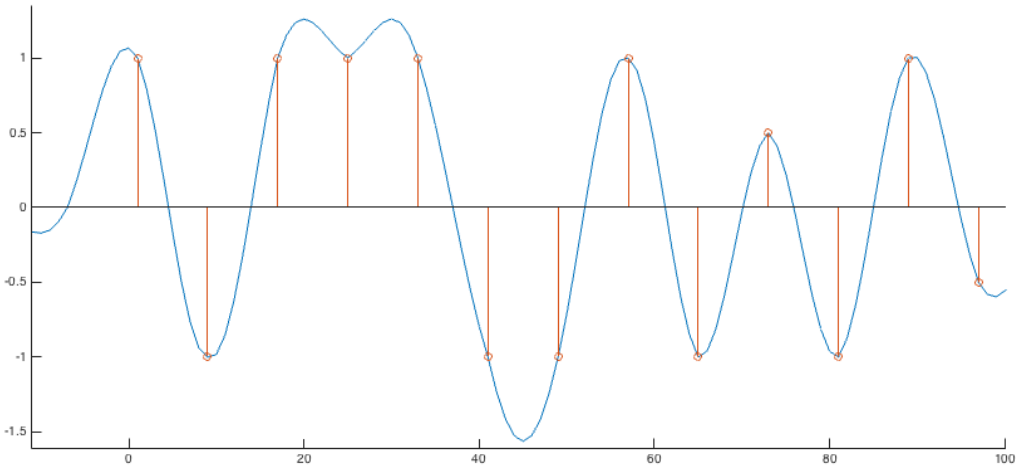
cùng tìm hiểu về chế độ đơn kênh với STM32, sử dụng Interrupt để báo quá trình chuyển đổi hoàn tất

- Trong bộ chuyển đổi ADC, có 2 thuật ngữ mà chúng ta cần chú ý đến, đó là độ phân giải (resolution) và thời gian lấy mẫu (sampling time)

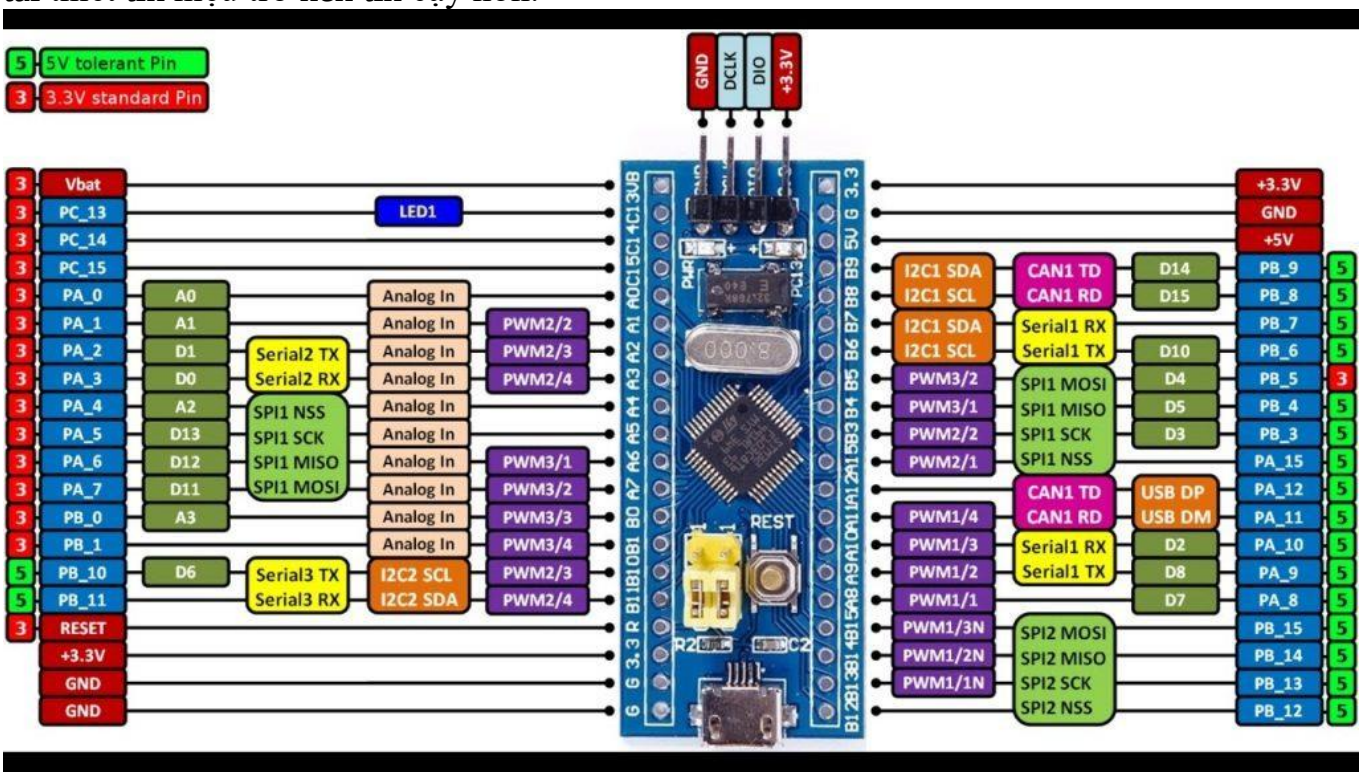


– Độ phân giải (resolution): dùng để chỉ số bit cần thiết để chứa hết các mức giá trị số (digital) sau quá trình chuyển đổi ở ngõ ra. Như trên biểu đồ:

- Màu xanh tương ứng thể hiện độ phân giải của bộ chuyển đổi này là 3 bit, tương ứng với 8 sự thay đổi ở đầu ra số (2^3=8). Khi đưa vào điện áp tương tự, bộ chuyển đổi sẽ thực hiện một công đoạn lượng tử hóa để đưa các kết quả tương ứng từ điện áp tương tự về số ở ngõ ra.
- Màu tím tương ứng với độ phân giải của bộ chuyển đổi 16 bit. Dễ dàng nhận thấy với một bộ chuyển đổi có độ phân giải càng thấp, quá trình chuyển đổi sẽ cho ra kết quả là một điện áp càng biến dạng ở ngõ ra so với ngõ vào và ngược lại. Bộ chuyển đổi ADC của STM32F103 có độ phân giải mặc định là 12 bit, tức là có thể chuyển đổi ra 2^{12} = 4096 giá trị ở ngõ ra số.



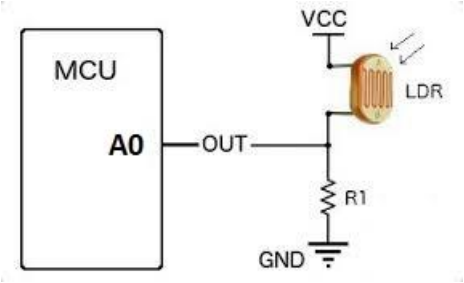
– Thời gian lấy mẫu (sampling time) là khái niệm được dùng để chỉ thời gian giữa 2 lần số hóa của bộ chuyển đổi. Như ở hình trên, sau khi thực hiện lấy mẫu, các điểm tròn chính là giá trị đưa ra tại ngõ ra số. Dễ nhận thấy nếu thời gian lấy mẫu quá lớn thì sẽ làm cho quá trình chuyển đổi càng bị mất tín hiệu ở những khoảng thời gian không nằm tại thời điểm lấy mẫu. Thời gian lấy mẫu càng nhỏ sẽ làm làm cho việc tái thiết tín hiệu trở nên tin cậy hơn.



b. Code tham khảo:

Bài 1: Đọc giá trị từ quang trở

- Lắp mạch:



- Code tham khảo:

```
#include "stm32f10x.h"

#include "stdio.h"

#ifdef __GNUC__
  /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
     set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
 /* Place your implementation of fputc here */
          /* e.g. write a character to the LCD */
 //lcd_Data_Write((u8)ch);
 USART_SendData(USART2,(u8)ch);

 /*Loop until the end of transmission */
 while (USART_GetFlagStatus(USART2,USART_FLAG_TC)==RESET)
 {}

          return ch;
}
```

```
/*Khoi tao bien cau hinh*/
GPIO_InitTypeDef                GPIO_InitStructure;
USART_InitTypeDef                       UART_InitStructure;
ADC_InitTypeDef                 ADC_InitStructure;

uint8_t data =10;
float value_adc1 = 0,sum_adc1=0, adc_tb=0;

void GPIO_Configuration(void);
void Delay_ms(uint16_t time);
void UART_Configuration (void);
void ADC_Configuration(void);
int main(void)
{
 GPIO_Configuration();
 UART_Configuration();
 ADC_Configuration();
  while (1)
  {
                // doc 10 lan gia tri ADC roi lay trung binh ket qua
                        for(int b=0;b<10;b++)
                                {
                                                value_adc1 =
ADC_GetConversionValue(ADC1);
                                                sum_adc1 = sum_adc1 +
value_adc1;
                                                Delay_ms(1);
                                }
                        adc_tb = sum_adc1/10;
                        sum_adc1 =0;
                        // xuat gia tri trung binh thong qua UART len may
tinh
                        printf("Gia tri ADC là: %4.3f\n",adc_tb);
                        Delay_ms(1000);

  }
}

void GPIO_Configuration(void)
{
```

```
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,
ENABLE);

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 ;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
 GPIO_Init(GPIOB, &GPIO_InitStructure);
}
void Delay_ms(uint16_t time)
{
uint32_t time_n=time*12000;
 while(time_n!=0){time_n--;}

}
void UART_Configuration (void)
 {
        /*Cap clock cho USART và port su dung*/

 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,
ENABLE);

 RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,
ENABLE);

                /* Cau Tx mode AF_PP, Rx mode FLOATING  */
                GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
                GPIO_InitStructure.GPIO_Mode =
GPIO_Mode_AF_PP;
                GPIO_InitStructure.GPIO_Speed =
GPIO_Speed_50MHz;
                GPIO_Init(GPIOA, &GPIO_InitStructure);

                GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
                GPIO_InitStructure.GPIO_Mode =
GPIO_Mode_IN_FLOATING;
                GPIO_Init(GPIOA, &GPIO_InitStructure);

        /*Cau hinh USART*/
                UART_InitStructure.USART_BaudRate = 115200;
```

```
		UART_InitStructure.USART_WordLength =
USART_WordLength_8b;
		UART_InitStructure.USART_StopBits =
USART_StopBits_1;
		UART_InitStructure.USART_Parity =
USART_Parity_No;
		UART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
		UART_InitStructure.USART_Mode =
USART_Mode_Rx | USART_Mode_Tx;
		USART_Init(USART2, &UART_InitStructure);

		/* Cho phep UART hoat dong */
		USART_Cmd(USART2, ENABLE);

 }
void ADC_Configuration(void)
 {
	/*cap clock cho chan GPIO va bo ADC1*/
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 ,
ENABLE);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,
ENABLE);

	/*cau hinh chan Input cua bo ADC1 la chan PA0*/
	GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_0;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	/*cau hinh ADC1*/
	ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
	ADC_InitStructure.ADC_ScanConvMode = DISABLE;
	ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
	ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_None;
	ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
	ADC_InitStructure.ADC_NbrOfChannel = 1;
	ADC_Init(ADC1, &ADC_InitStructure);
```

```
	/* Cau hinh chanel, rank, thoi gian lay mau */
	ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1,
ADC_SampleTime_55Cycles5);
		/* Cho phep bo ADC1 hoa dong */
	ADC_Cmd(ADC1, ENABLE);
	/* cho phep cam bien nhiet hoat dong */
	ADC_TempSensorVrefintCmd(ENABLE);
	/* Reset thanh ghi cablib  */
	ADC_ResetCalibration(ADC1);
	/* Cho thanh ghi cablib reset xong */
	while(ADC_GetResetCalibrationStatus(ADC1));
	/* Khoi dong bo ADC */
	ADC_StartCalibration(ADC1);
	/* Cho trang thai cablib duoc bat */
	while(ADC_GetCalibrationStatus(ADC1));
	/* Bat dau chuyen doi ADC */
	ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

3.5. Timer

a. Lý thuyết:

- STM32f103C8 có tất cả 7 timer nhưng trong đó đã bao gồm 1 systick timer, 2 watchdog timer. Vậy chỉ còn lại 4 timer dùng cho các chức năng như ngắt, timer base, PWM, Encoder, Input capture…. Trong đó TIM1 là Timer đặc biệt, chuyên dụng cho việc xuất xung với các mode xuất xung, các mode bảo vệ đầy đủ hơn so với các timer khác. TIM1 thuộc khối clock APB2, còn các TIM2,TIM3,TIM4 thuộc nhóm APB1.
- Có 3 vấn đề cần phải tìm hiểu trong bài này đó là :
  - Timer clock.
  - Prescaler
  - Auto Reload Value.
- **Timer clock:** Khi không có cấu hình gì liên quan đến clock và đã gắn đúng thạch anh ngoài trên chân PD0(5) và PD1(6) thì clock tương ứng của TIM1,TIM2,TIM3,TIM4 đã là 72Mhz. Cần ghi nhớ là sử dụng timer nào thì cấp clock cho timer đó theo đúng nhánh clock.
- **Prescaler** là bộ chia tần số của timer. Bộ chia này có giá trị tối đa là 16 bit tương ứng với giá trị là 65535. Các giá trị này có thể được thay đổi và điều chỉnh bằng lập trình. Tần số sau bộ chia này sẽ được tính là:

$$\mathbf{fCK\_CNT = fCK\_PSC/(PSC+1).}$$

  - FCK_CNT: tần số sau bộ chia.
  - fCK_PSC: tần số clock đầu vào cấp cho timer.
  - PSC: chính là giá trị truyền vào được lập trình bằng phần mềm
- **Auto Reload value** là giá trị bộ đếm tối đa có thể được điều chỉnh để nạp vào cho timer. Giá trị bộ đếm này được cài đặt tối đa là 16bit tương ứng với giá trị là 65535.Từ các thông số trên ta rút ra công thức cần tính cuối cùng đó là:

$$\mathbf{FTIMER= fSYSTEM/[(PSC+1)(Period+1)]}$$

  - FTIMER : là giá trị cuối cùng của bài toán, đơn vị là hz.
  - fSYSTEM : tần số clock hệ thống được chia cho timer sử dụng, đơn vị là hz.
  - PSC : giá trị nạp vào cho bộ chia tần số của timer. Tối đa là 65535.
  - Period : giá trị bộ đếm nạp vào cho timer. Tối đa là 65535.
- Ngắt timer: khi giá trị đếm của bộ đếm timer(thanh ghi CNT) vượt qua giá trị của Auto Reload Value thì cờ báo tràn sẽ được kích hoạt. Trình phục vụ ngắt tràn sẽ xảy ra nếu được cấu hình cho phép trước đó.

b. Ví dụ:

Ví dụ1: cấu hình cho timer 3 trong stm32f1 tạo interrupt với 100ms

Phân tích : (1 giây = 1ms x 1000)

- Tần số cao nhất mà timer 3 trong stm32f1 đạt được là 72Mhz.
- Yêu cầu cần timer 3 tạo interrupt mỗi 100ms.
- Giả sử cần counter của timer đếm 100 lần để được 100ms và phát interrupt thì ta có :
- Tần sô Fc_timer3 = (100ms x 100 clock) x 10 = 10.000 clock/ 1 giây (10Khz).

Kết quả :

- TIM_Prescaler = (72000000/10.000 – 1); //Fc_timer là 10khz
- TIM_Period = 100 lần – 1 = 99

- Giả sử cần counter timer đếm 1000 lần để được 100ms và phát interrupt thì tương tự :
- Tần số Fc_timer3 = (100ms x 1000 clock) x 10 = 100.000 clock/ 1 giây (100Khz)

Kết quả :

- TIM_Prescaler = (72000000/100.000 – 1); //Fc_timer là 100khz
- TIM_Period = 1000 lần – 1 = 999

Code tham khào:

```
//Cấu hình các thông số cho timer3 với trường hợp period = 99:
TIM_TimeBaseStructure.TIM_Period = 99; // delay 10ms
TIM_TimeBaseStructure.TIM_Prescaler = (72000000/10000 – 1); //10khz
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
```

Ví dụ 2: Blink led tại chân PB2 với chu kì 0.5s sủ dụng ngắt Timer 4.

Code tham khảo:

```
#include "stm32f10x.h"

// cau hinh led chan PB2
void GPIO_configuration(void)
```

```
{
	GPIO_InitTypeDef GPIO_InitStructure;

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOB, &GPIO_InitStructure);
}

// cau hinh timer 4
void TIM4_configuration(void)
{
	TIM_TimeBaseInitTypeDef TIM_InitStructure;

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

	TIM_InitStructure.TIM_ClockDivision = 0;
	TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_InitStructure.TIM_Period = 999;
	TIM_InitStructure.TIM_Prescaler = 35999;
	TIM_InitStructure.TIM_RepetitionCounter = 0;

	TIM_TimeBaseInit(TIM4, &TIM_InitStructure);
	TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE); //enable update interrup
	TIM_Cmd(TIM4, ENABLE);
}

// su kien ngat
void NVIC_configuration(void)
{
	NVIC_InitTypeDef NVIC_InitStructure;

	NVIC_PriorityGroupConfig( NVIC_PriorityGroup_0);

	NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);
```

```
}

void TIM4_IRQHandler(void)
{
	if(TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET) //if update flag
turn on
	{
		TIM_ClearITPendingBit(TIM4, TIM_IT_Update); //clear update flag
		GPIO_WriteBit(GPIOB, GPIO_Pin_2,
(BitAction)(1^GPIO_ReadInputDataBit(GPIOB,GPIO_Pin_2)));
	}
}

int main(void)
{
	SystemInit();
	GPIO_configuration();
	TIM4_configuration();
	NVIC_configuration();
	while(1)
	{

	}
}
```