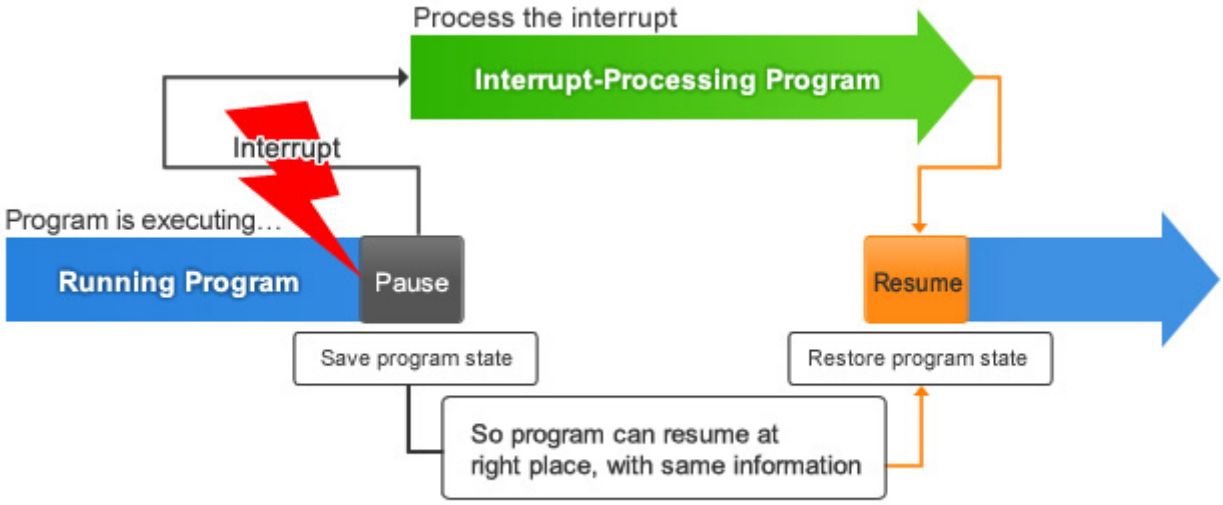
Quá trình thực hiện ngắt của vi điều khiển – MCU Interrupt processing

Bình thường, vi điều khiển sẽ thực thi các lệnh do người dùng viết một cách tuần tự từ trên xuống. Tuy nhiên, nó cũng được thiết kế để sẵn sàng xử lý các tình huống, sự kiện do tác động từ bên ngoài của con người, các cảm biến, hoặc từ các ngoại vi bên trong như Timer, UART, ADC…vv… mà chúng ta không biết, không dự đoán trước được khi nào tình huống, sự kiện đó sẽ xảy ra.



Một cách tổng quát, khi xảy ra Interrupt, vi điều khiển sẽ thực hiện qua các bước sau:

1. Thực hiện xong câu lệnh đang thực hiện (câu lệnh ở mã máy sau quá trình compiler, asembler từ ngôn ngữ bật cao do người dùng viết. Để thực hiện 1 câu lệnh ở mã máy, vi điều khiển thường thực hiện các bước sau: lấy lệnh từ bộ nhớ; giải mã lệnh; thực thi lệnh).

2. Lưu ngữ cảnh gồm lưu địa chỉ câu lệnh tiếp theo sẽ thực hiện (giá trị thanh ghi Program Counter), lưu trạng thái năng lượng đang hoạt động (trong thanh ghi Status) vào vùng nhớ Stack, gọi là quá trình Stacking.(Vùng nhớ Stack là vùng nhớ First In Last Out.)

3. Xóa bit cho phép ngắt toàn cục trong thanh ghi Status, đưa vi điều khiển về chế độ hoạt động bình thường (active mode) nếu nó đang ở chế độ tiết kiệm năng lượng. Bit cho phép ngắt cũng có thể được bật lên lại để cho phép ngắt chồng ngắt (Nested Interrupt)

4. Vi điều khiển thực thi chương trình phục vụ ngắt (ISR) bằng cách nạp địa chỉ câu lệnh đầu tiên của chương trình phục vụ ngắt vào thanh ghi PC. (Địa chỉ này cũng là địa chỉ của vecter ngắt trong interrupt vector table)

5. Khi thực hiện xong chương trình phục vụ ngắt, vi điều khiển sẽ thực hiện quá trình unstacking: nạp lại giá trị thanh ghi PC đã lưu, bật lại bit cho phép ngắt toàn cục, quay về trạng thái năng lượng ban đầu.

Một số ngắt phổ biến trên vi điều khiển phổ biến mà chúng ta thường sử dụng:
– *Ngắt ngoài*: Sự kiện là khi sự thay đổi sườn tín hiệu (edge) sườn lên, sườn xuống, hoặc cả 2.

– *Ngắt UART*: Thường sử dụng ngắt nhận, sự kiện là khi buffer nhận đủ 1 byte dữ liệu

– *Ngắt ADC*: Thường sử dụng khi hoàn thành việc chuyển đổi ADC

– *Ngắt Timer*: Thường sử dụng khi tràn thanh ghi đếm, hoặc khi giá trị đếm bằng với thanh ghi so sánh

Ngắt ngoài trên STM32F103C8T6

NVIC – Nested vectored interrupt controller là bộ điều khiển xử lý ngắt có trong MCU STM32F103C8T6. Việc lập trình sử dụng ngắt là một kĩ năng quan trọng khi các bạn lập trìnhh vi điều khiển, nếu không có ngắt thì chương trình sẽ thực hiện theo một trình tự từ trên xuống dưới, ngắt giúp chương trình xử lý theo sự việc, đáp ứng được các sự kiện như sự thay đổi mức logic từ 1 chân vi điều khiển (ngắt ngoài), nhận một kí tự (ngắt nhận UART)…. Trong bài viết này, mình sẽ trình bày về ngắt ngoài (external interrupt) của vi điều khiển STM3232F103C8T6.

Một số thông số tính năng chính của NVIC:

- 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

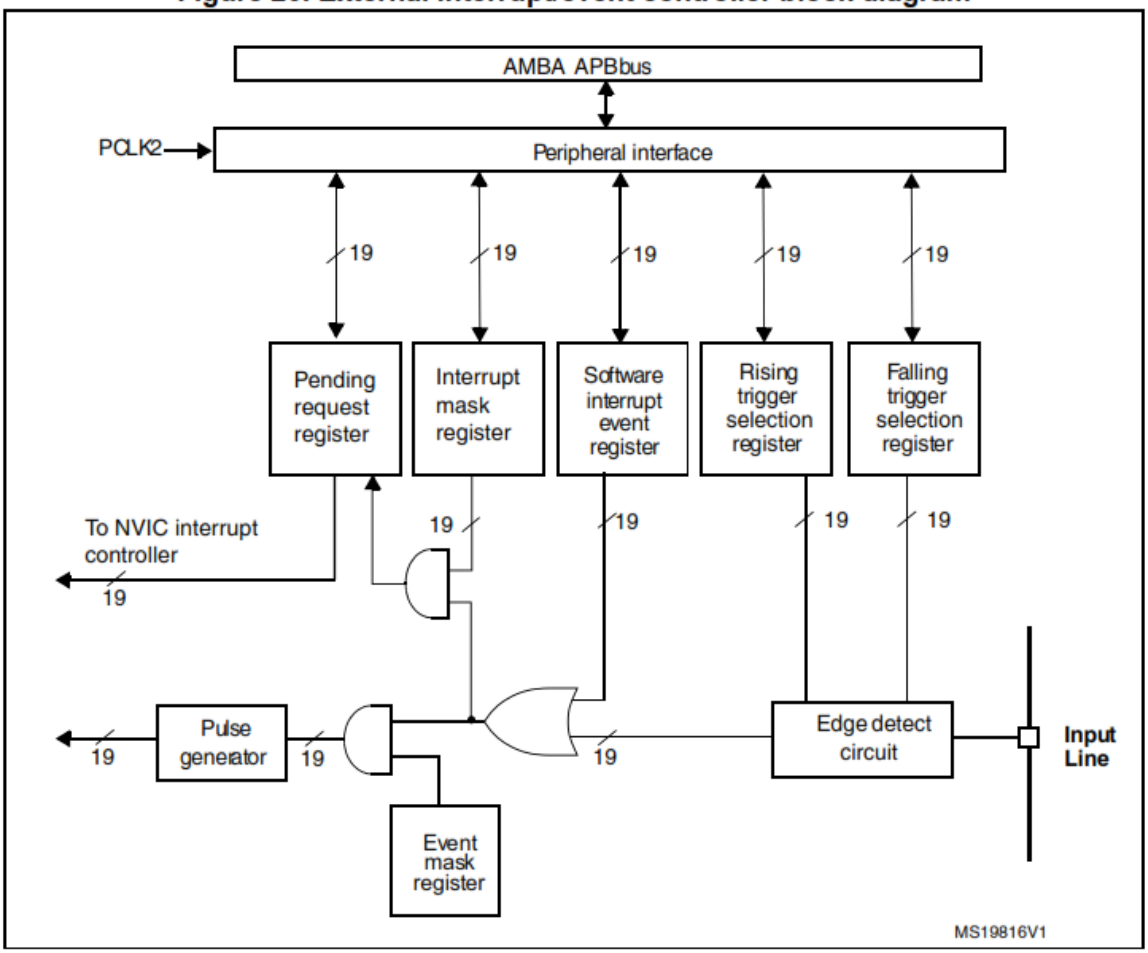
Ngắt ngoài nằm trong 1 phần của NVIC. Mỗi EXTI – interrupt/event controller có thể được lập trình chọn loại sự kiện/ ngắt, chọn cạnh lên, cạnh xuống hoặc cả 2, sắp xếp mức ưu tiên ngắt.

Một số tính năng chính của ngắt ngoài:

- Kích hoạt độc lập và mask cho mỗi line sự kiện/ngắt.
- Có bit trạng thái (status) 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:

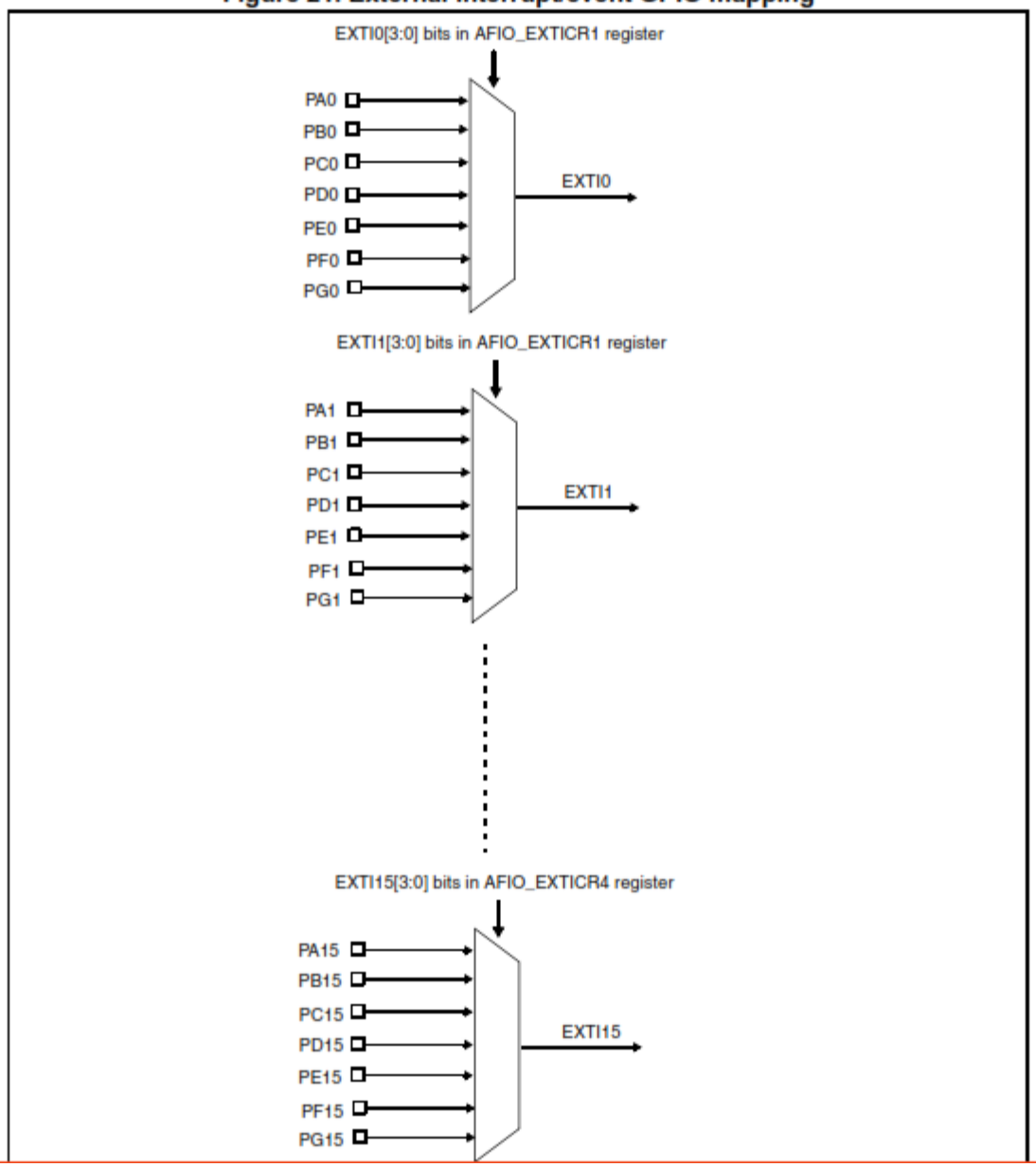
Figure 20. External interrupt/event controller block diagram



Cấu hình với thư viện chuẩn của ST. Có 2 loại ngắt ngoài chính đó là ngắt ngoài trên các chân điều khiển ở dạng thông thường và ngắt ngoài trên các ứng dụng như : PVD, RTC, USB, Ethernet.

Ngắt ngoài của chip STM32F103 bao gồm có 16 line:

Figure 21. External interrupt/event GPIO mapping



Ở đây chúng ta có thể thấy chip STM32F103C8 gồm có 16 Line ngắt riêng biệt.

Ví dụ:

- Line0 sẽ chung cho tất cả chân Px0 ở tất cả các Port, với x là tên của Port A, B…
- Line0 nếu chúng ta đã chọn chân PA0 (chân 0 ở port A) làm chân ngắt thì tất cả các chân 0 ở các Port khác không được khai báo làm chân ngắt ngoài nữa
- Line1 nếu chúng ta chọn chân PB1 là chân ngắt thì tất cả chân 1 ở các Port khác không được khai báo làm chân ngắt nữa.

Tiếp theo các Line ngắt sẽ được phân vào các Vector ngắt tương ứng. Các Line ngắt của chip STM32F103 được phân bố vào các vector ngắt như sau:

6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0

Các Line0, Line1, Line2, Line3, Line4 sẽ được phân vào các vector ngắt riêng biệt EXTI0, EXTI1, EXTI2, EXTI3, EXTI4, còn từ Line5->Line9 sẽ được phân vào vector ngắt EXTI9_5, Line10->Line15 được phân vào vecotr EXTI15_10.

Một số thanh ghi quan trọng:

1.EXTI_IMR – Interrupt mask register: Thanh ghi này cài đặt cho phép có yêu cầu ngắt trên Line tương ứng. (cho phép ngắt)

Address offset: 0x00
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx:** Interrupt Mask on line x
0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked
Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

2. EXTI_RTSR – Rising trigger selection register: Thanh ghi này được sử dụng để cấu hình chọn sườn lên làm tín hiệu kích hoạt ngắt.

10.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Rising trigger event configuration bit of line x
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line.
Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

3. EXTI_FTSR – Falling trigger selection register: Thanh ghi này được sử dụng để cấu hình chọn sườn xuống làm tín hiệu kích hoạt ngắt

10.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Falling trigger event configuration bit of line x
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.
Note: Bit 19 used in connectivity line devices and is reserved otherwise.

4. EXTI_SWIER – Software interrupt even register: Thanh ghi này cho phép kích hoạt Line ngát tương tứng bằng phần mềm.

10.3.5 Software interrupt event register (EXTI_SWIER)

Address offset: 0x10
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												SWIER 19	SWIER 18	SWIER 17	SWIER 16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **SWIERx:** Software interrupt on line x

If the interrupt is enabled on this line in the EXTI_IMR, writing a '1' to this bit when it is set to '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation.

This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a 1 into the bit).

Note: Bit 19 used in connectivity line devices and is reserved otherwise.

5. EXTI_PR – Pending register: Đây là thanh ghi chờ xử lý ngắt, khi có yêu cầu ngắt được tạo ra trên một Line ngắt thì bit tương tứng của thanh ghi này được bật lên cho đến khi ngắt này được xử lý. Nhiều trước hợp có sự thay đổi sườn tín hiệu tạo ra yêu cầu ngắt nhưng ngắt không được thực thi như: độ ưu tiên thấp, chưa cho phép ngắt toàn cục.

10.3.6 Pending register (EXTI_PR)

Address offset: 0x14
Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												PR19	PR18	PR17	PR16
												rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **PRx:** Pending bit

0: No trigger request occurred
1: selected trigger request occurred
This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a '1' into the bit.

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

Bảng mức độ ưu tiên ngắt NVIC:

NVIC_PriorityGroup	NVIC_IRQChannelPreemptionPriority	NVIC_IRQChannelSubPriority	Description
NVIC_PriorityGroup_0	0	0-15	0 bits for pre-emption priority, 4 bits for subpriority
NVIC_PriorityGroup_1	0-1	0-7	1 bits for pre-emption priority, 3 bits for subpriority
NVIC_PriorityGroup_2	0-3	0-3	2 bits for pre-emption priority, 2 bits for subpriority
NVIC_PriorityGroup_3	0-7	0-1	3 bits for pre-emption priority, 1 bits for subpriority
NVIC_PriorityGroup_4	0-15	0	4 bits for pre-emption priority, 0 bits for subpriority

Có hai loại ưu tiên ngắt khác nhau trên MCU STM32F103C8T6 đó là Preemption Priorities và Sub Priorities:

– Mặc định thì ngắt nào có Preemtion Priority cao hơn thì sẽ được thực hiện trước.

– Khi nào 2 ngắt có cùng một mức Preemption Priority thì ngắt nào có Sub Priority cao hơn thì ngắt đó được thực hiện trước.

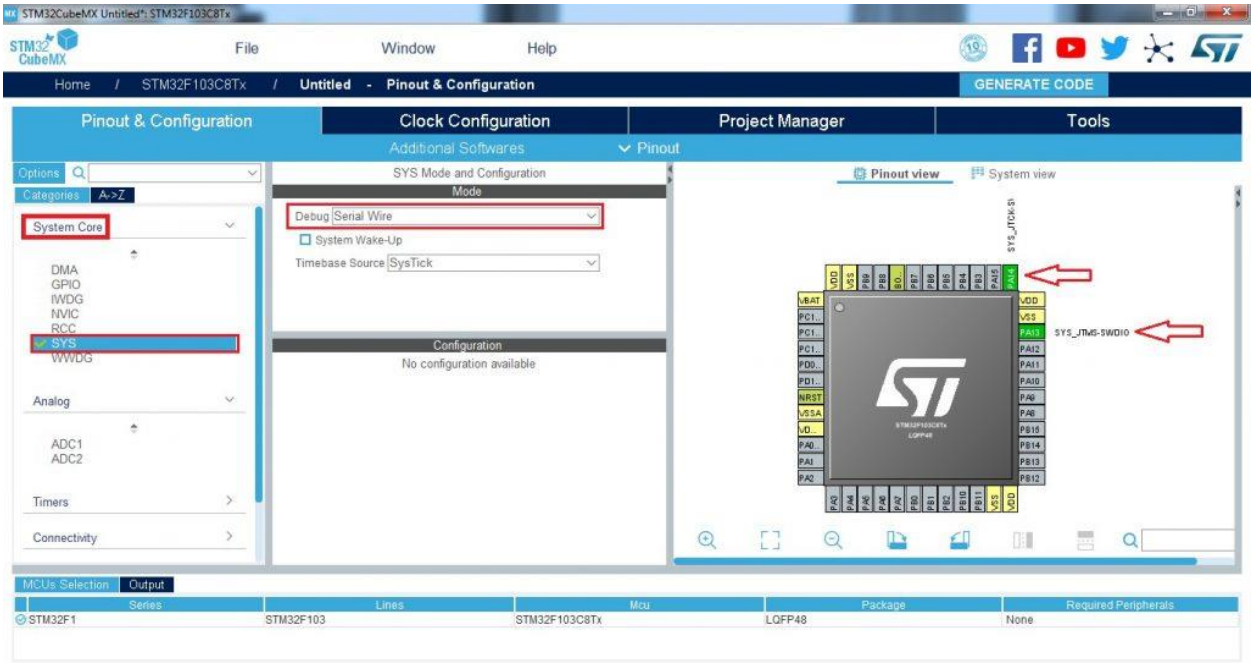
– Còn trường hợp 2 ngắt có cùng mức Preemption và Sub Priority luôn thì ngắt nào đến trước được thực hiện trước.

THỰC HÀNH:

Tiếp theo chúng ta sẽ thực hành project ngắt ngoài trên chip STM32F103C8T6 với CubeMX. Ở ví dụ này, chúng ta sẽ sử dụng chân PA0 tương ứng với Line0 – vector EXTI0 và PA6 tương ứng Line6 – vector EXTI5_9 để thực hành. Khai báo thêm chân Led PC13, khi ngắt ngoài ở chân PA0 thì chúng ta sẽ tắt Led, khi ngắt ngoài ở chân PA6 thì chúng ta sẽ bật Led

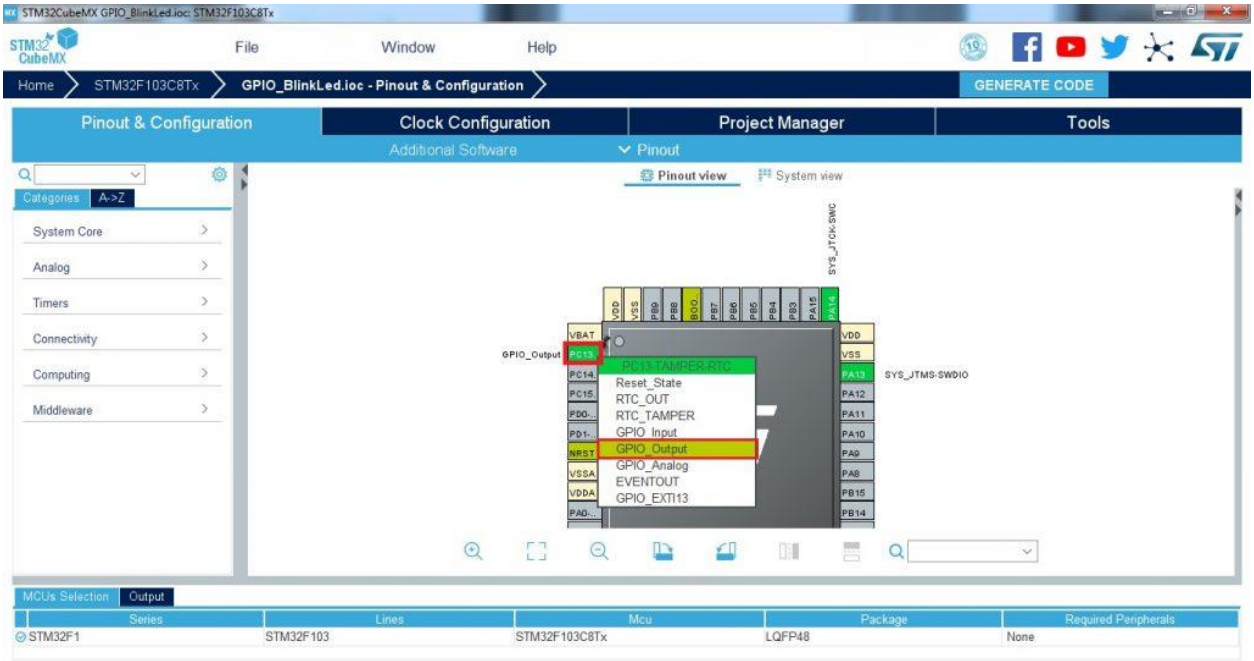
Bước 1:

Khởi tạo project CubeMX, sau đó chọn cổng nạp dữ liệu Serial Wire



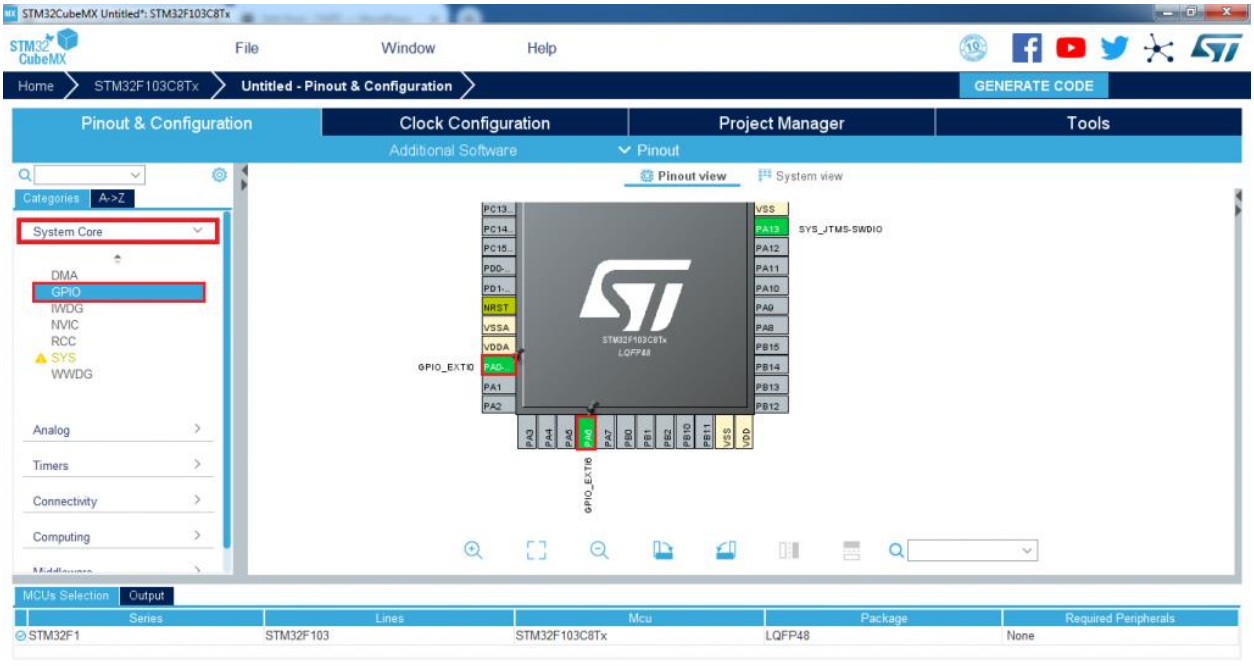
Bước 2:

Chọn chân PC13 là chân GPIO_Output



Bước 3:

Chọn chân PA0 là chân GPIO_EXTI0, chúng ta cũng làm tương tự đối với chân PA6 là chân GPIO_EXTI6

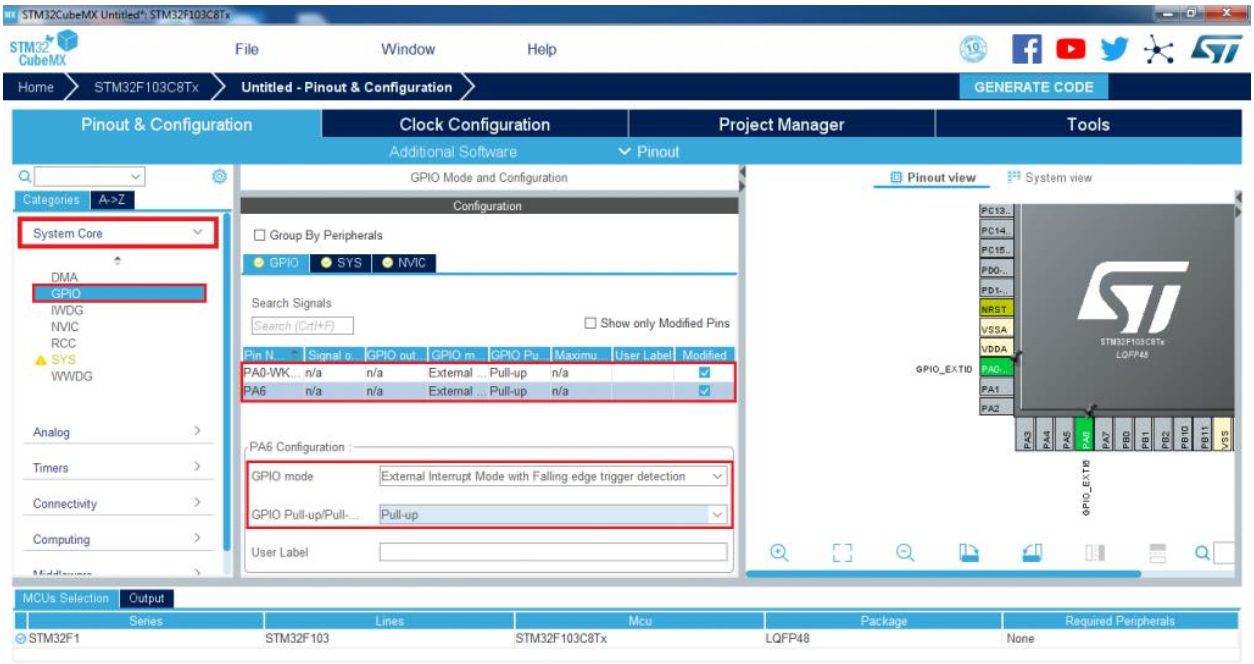


Bước 4:

Tiếp theo, click vào System Core sau đó click vào GPIO. Lúc này sẽ hiện ra một bảng Pin Configuration. Click vào lần lượt các chân PA0 và PA6 để cấu hình:

GPIO_Mode: External Interrupt Mode with Falling edge trigger detection

GPIO_Pullup/Pull-down: Pull-up

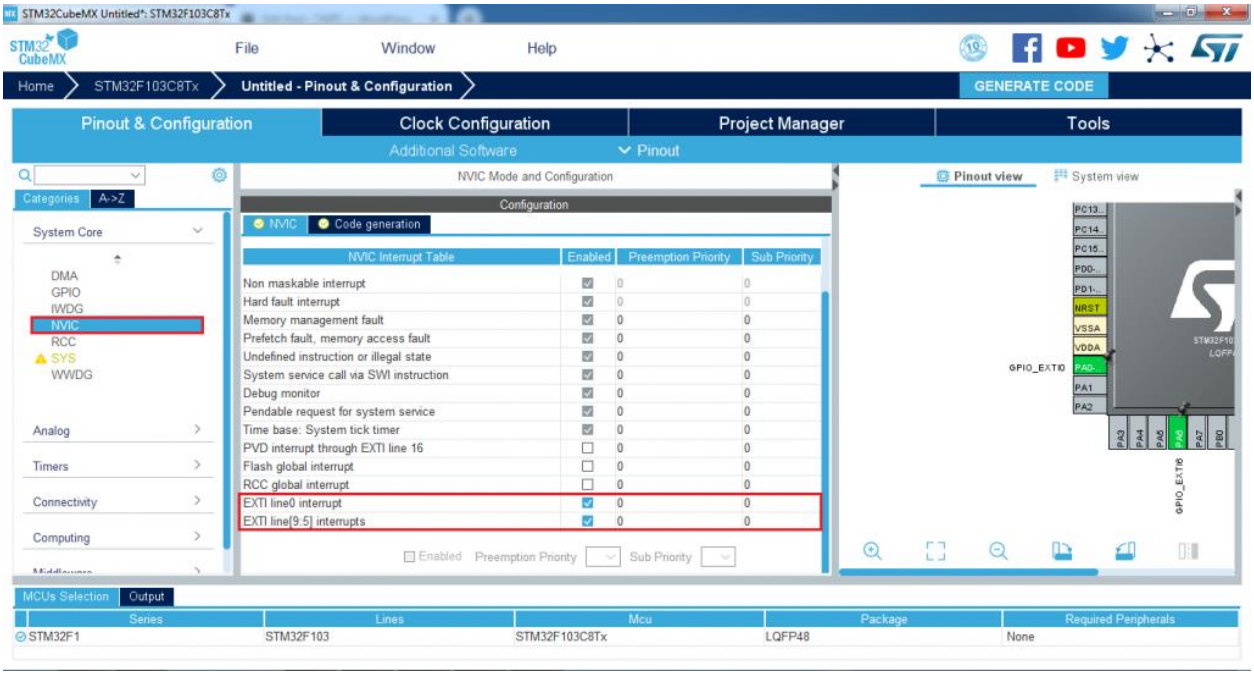


Bước 5:

Tiếp theo các bạn click vào NVIC, ở đây chúng ta có thể thấy:

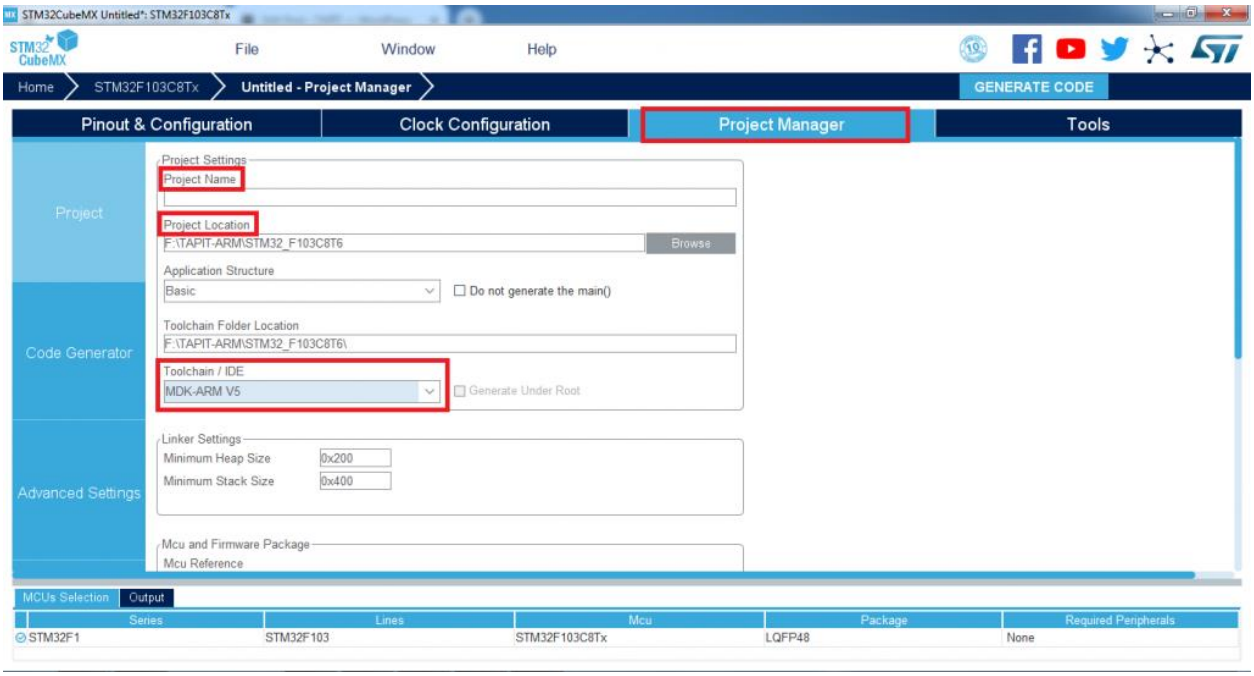
Piority Group: 4bits for pre-emtion piority 0 bit for subpiority, ở đây mặc định ban đầu CubeMX đã set nhóm ưu tiên ngắt là ở nhóm 4 gồm 2 trường preempiority, subpiority, với 16 giá trị preempiority và 0 giá trị preempiority.

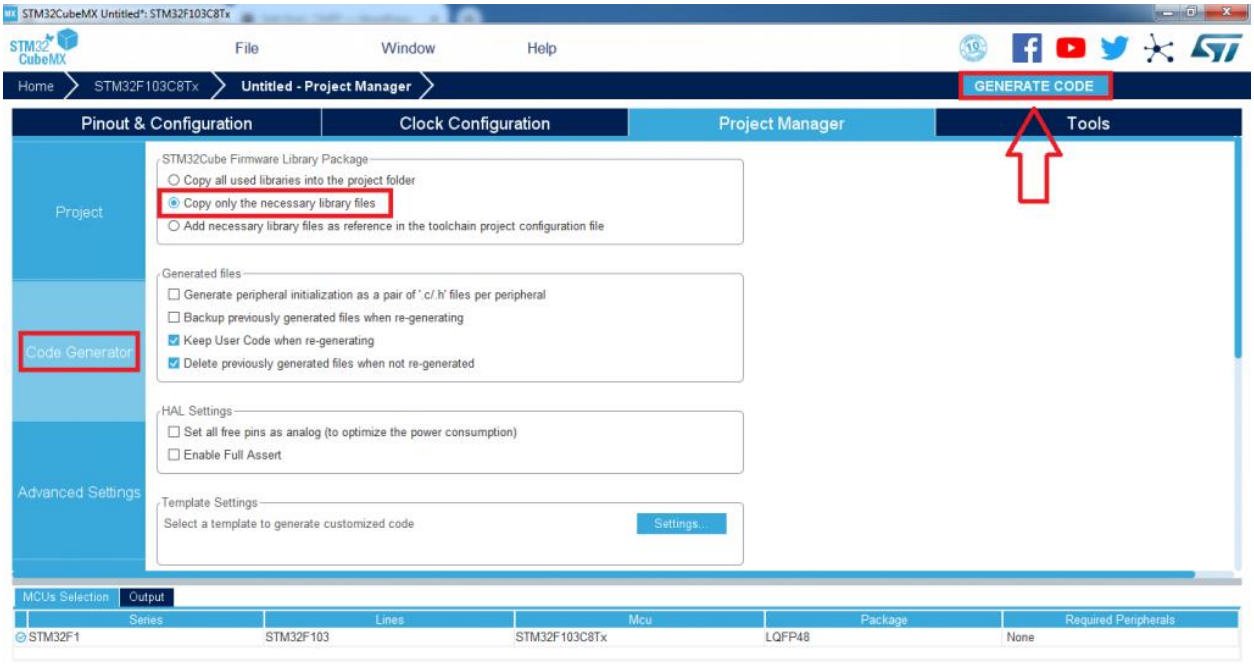
Chúng ta tít chọn để bật ngắt cho EXTI0 và EXTI5_9, mặc định ban đầu preempiority và subpiority của ngắt vector ngắt này đều bằng 0, nghĩa là có mức độ ưu tiên như nhau. Chúng ta có thể cài đặt lại mức độ ưu tiên của từng vector ngắt tùy theo mục đích sử dụng. Các vector ngắt nào có preempiority cao hơn thì mức độ ưu tiên thấp hơn và ngược lại các vector ngắt nào có subpiority cao hơn thì mức độ ưu tiên cao hơn.



Bước 6:

Cấu hình project và sinh code.



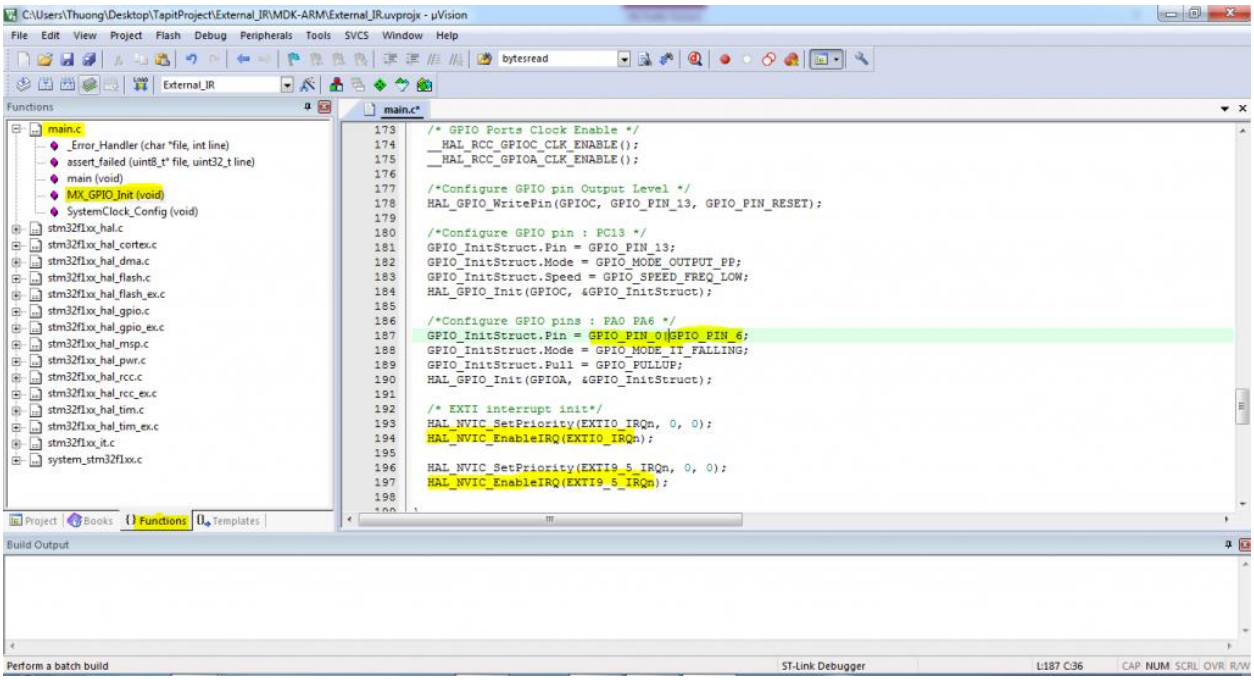


Bước 7:

Các bạn click vào Function, mở rộng main.c ở đây chúng ta có thể xem tất cả các hàm hiện có trong file main.

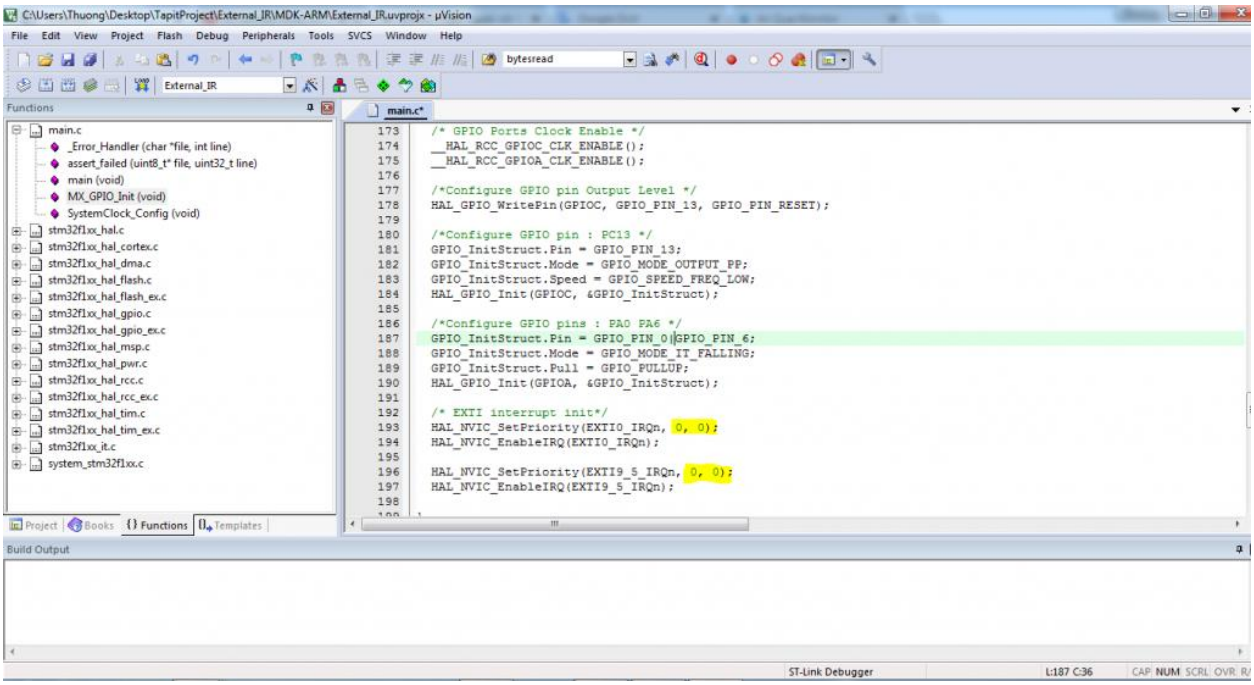
Click vào MX_GPIO_Init(void) ở đây chúng ta thấy rằng đã khởi tạo chân PA0,PA6, ngắt khi có cạnh xuống, chân này ban đầu được kéo lên nguồn.

Bật ngắt trên 2 vector EXTI0, EXTI5_9



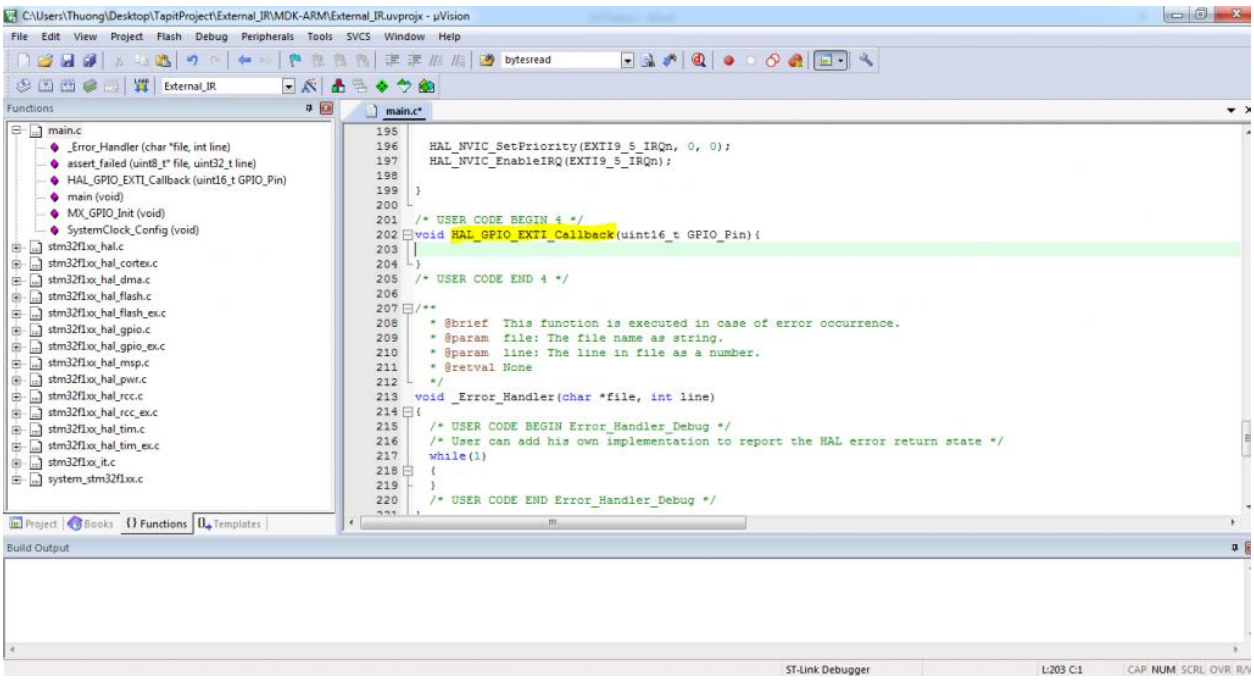
Bước 8:

Ở hàm HAL_NVIC_Setpriority(), chúng có thể thấy thông số đầu tiên là vector, thông số thứ 2 là mức độ ưu tiên của preemprioity, thứ 3 là mức độ ưu tiên của subpriority. Mặc định ban đầu 2 thông số này của 2 vector ngắt set bằng 0 tương ứng với mức độ ưu tiên của 2 vector ngắt EXTI0, EXTI5_9 bằng nhau.



Bước 9:

Tiếp theo chúng ta khởi tạo hàm void HAL_GPIO_EXTI_Callback(){}

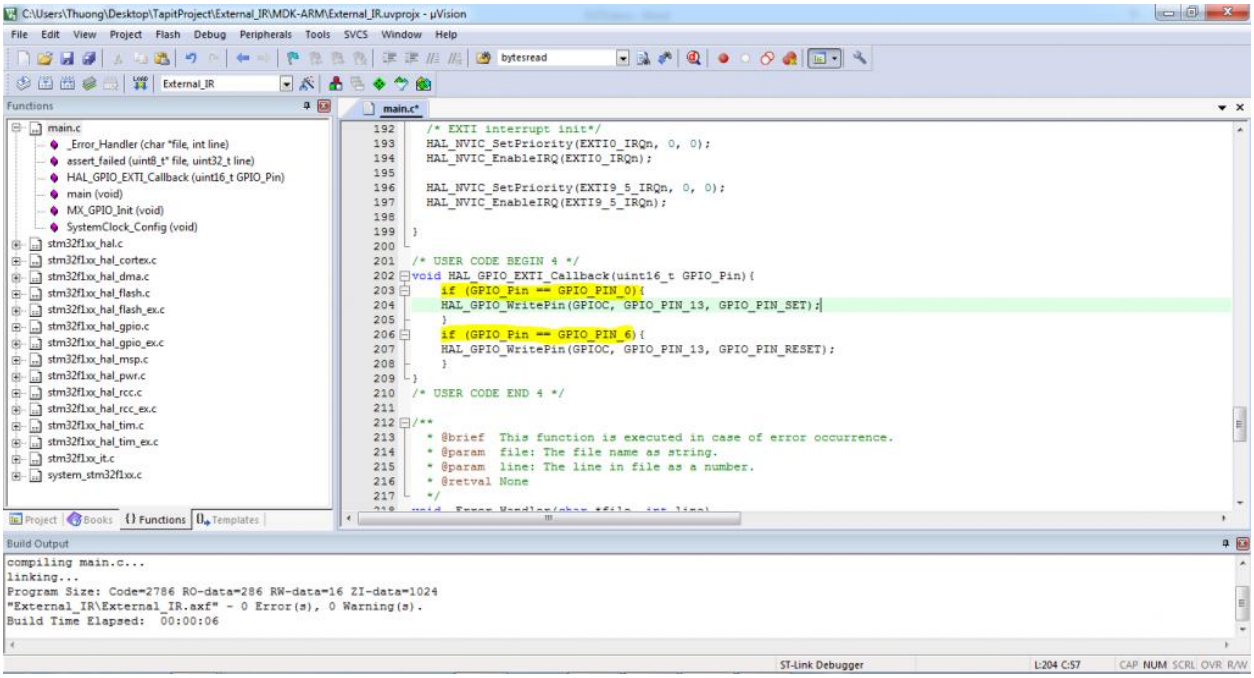


Bước 10:

Ở trong hàm này, mình sẽ có 2 lệnh điều kiện if() để phân luồng ngắt kiểm tra rằng ngắt hiện tại đang sinh ra ở chân nào.

If(GPIO_Pin == GPIO_PIN_0)

If(GPIO_Pin == GPIO_PIN_6)

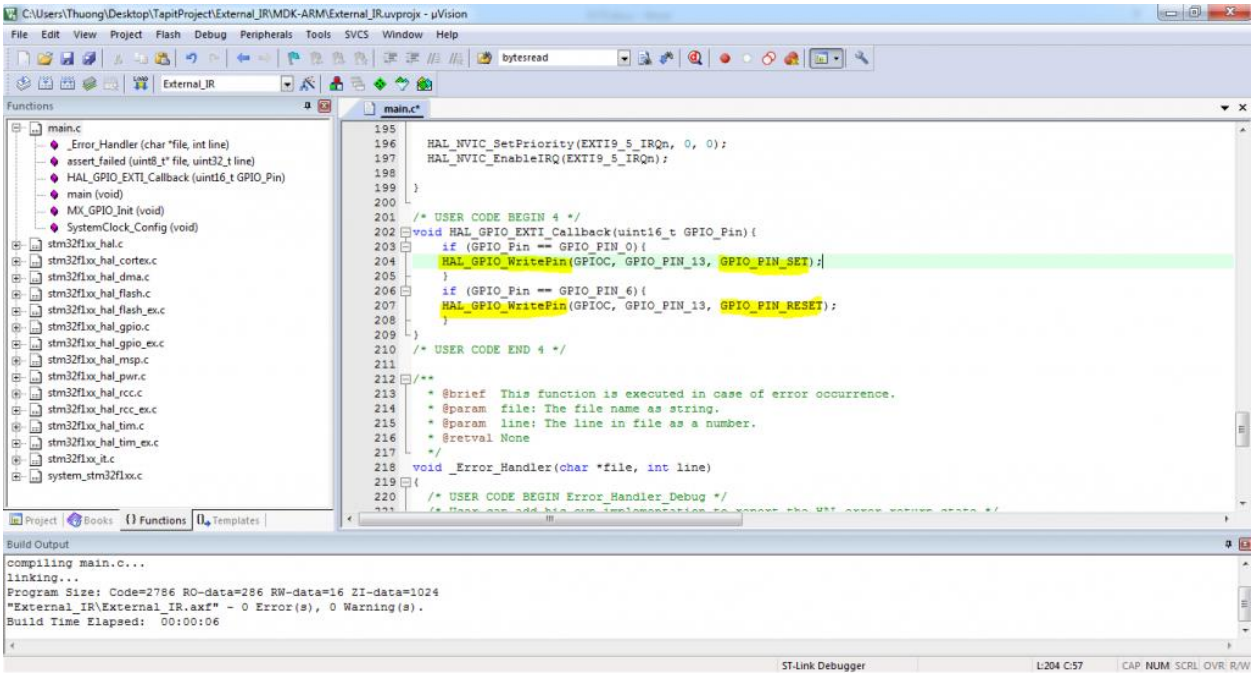


Bước 11:

Sau khi đã phân luồng ngắt được rồi, tiếp theo ở trong hàm kiểm tra chân nào đang ngắt:

+ PA0 mình sẽ cho tắt Led PC13

+ PA6 mình sẽ cho bật Led PC13

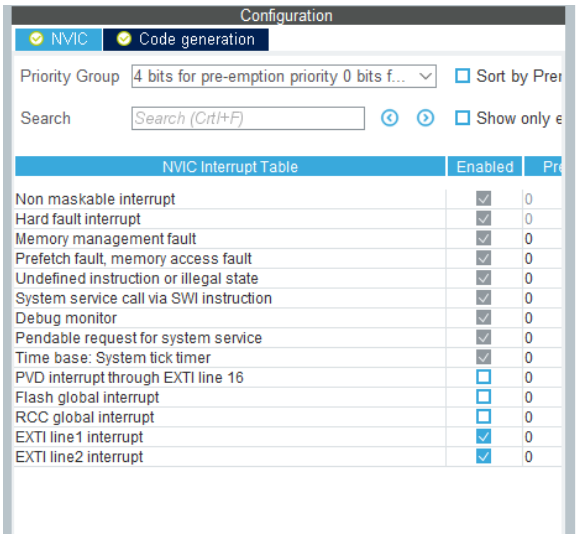


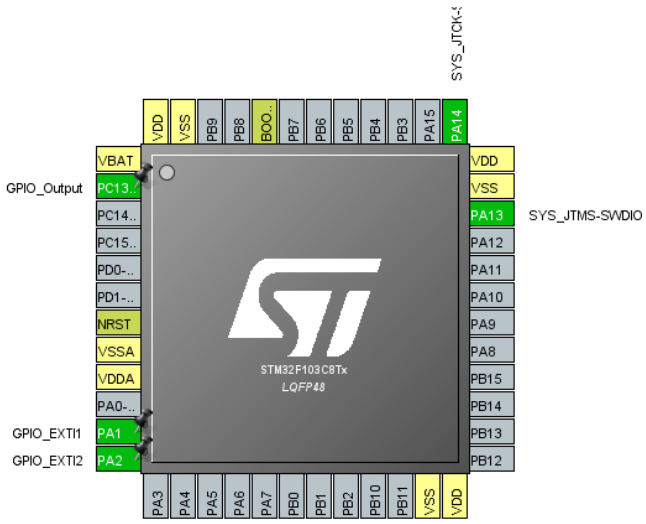
Tiếp theo các bạn biên dịch và nạp code vào chip STM32F103C8T6.

Cách phân biệt các chân sinh ra ngắt trên vi điều khiển STM32

Làm sao để biết yêu cầu ngắt được tạo ra từ chân GPIO nào khi chương trình của bạn sử dụng nhiều chân External Interrupt? Trong bài viết này, mình sẽ giúp các bạn giải quyết được vấn đề trên với các ví dụ cụ thể. Mình sử dụng vi điều khiển STM32F103C8T6, cấu hình bằng STM32CubeMX sau đó code bằng Keil C IDE, sử dụng thư viện HAL.

Ví dụ mình có 02 nút bấm ở chân PA1 và PA2 khai báo External Interrupt, sử dụng trở kéo lên để định mức logic khi không nhấn nút, sử dụng sườn xuống để kích hoạt ngắt. Mình cấu hình thêm chân PC13 là GPIO OUTPUT để điều khiển LED.





Khi sinh code ra thì sẽ có các hàm liên quan đến ngắt ngoài trong thư viện HAL như sau:

Trong file stm32f1xx_hal_gpio.c

```
1  /**
2  * @brief EXTI line detection callbacks.
3  * @param GPIO_Pin: Specifies the pins connected EXTI line
4  * @retval None
5  */
6  __weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
7  {
```

```
8   /* Prevent unused argument(s) compilation warning */
9   UNUSED(GPIO_Pin);
10  /* NOTE: This function Should not be modified, when the callback is needed,
11   the HAL_GPIO_EXTI_Callback could be implemented in the user file
12  */
13}
14
15
16/**
17* @brief This function handles EXTI interrupt request.
18* @param GPIO_Pin: Specifies the pins connected EXTI line
19* @retval None
20*/
21void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
22{
23   /* EXTI line interrupt detected */
24   if (__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
25   {
26     __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
27     HAL_GPIO_EXTI_Callback(GPIO_Pin);
28    }
29}
```

Trong file stm32f1xx_it.c:

```
1  /**
2  * @brief This function handles EXTI line1 interrupt.
```

```
3  */
4  void EXTI1_IRQHandler(void)
5  {
6    /* USER CODE BEGIN EXTI1_IRQn 0 */
7
8    /* USER CODE END EXTI1_IRQn 0 */
9    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
10  /* USER CODE BEGIN EXTI1_IRQn 1 */
11
12  /* USER CODE END EXTI1_IRQn 1 */
13}
14
15
16
17/**
18* @brief This function handles EXTI line2 interrupt.
19*/
20void EXTI2_IRQHandler(void)
21{
22    /* USER CODE BEGIN EXTI2_IRQn 0 */
23
24    /* USER CODE END EXTI2_IRQn 0 */
25   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);
26   /* USER CODE BEGIN EXTI2_IRQn 1 */
27
28    /* USER CODE END EXTI2_IRQn 1 */
29}
```

Bình thường, với 1 chương trình đơn giản, các bạn vào file **stm32f1xx_hal_gpio.c** copy cả phần định nghĩa hàm Callback ra file **main.c**, bỏ đi từ khóa **weak** và viết chương trình phục vụ ngắt vào đấy là xong.

Ví dụ khi phát hiện nhấn nút (có sườn xuống tại bất 1 trong 2 chân đã khai báo) thì đảo trạng thái của LED.

```
1void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
2{
3  HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
4   /*
5    When the callback is needed,
6    the HAL_GPIO_EXTI_Callback could be implemented in the user file
7  */
8}
```

Nếu bài toán không phải là đảo LED khi nhấn bất kì 1 trong 2 nút bấm mà là nhấn nút PA1 thì bật LED còn nhấn nút PA2 thì tắt LED thì các bạn phải trả lời được câu hỏi:Làm sao để có thể phân biệt được chân nào đang sinh ra ngắt?
Có 2 cách:
Cách 1: Sử dụng hàm Callback như trên, tuy nhiên trong hàm Callback các bạn kiểm tra xem chân nào đang được truyền vào hàm thì chính là ngắt tại chân đó. Ví dụ:

```
1  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
2  {
3    if(GPIO_Pin == GPIO_PIN_1) //PA1 - Turn on LED
4    {
5      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, 1);
6    }
7    if(GPIO_Pin == GPIO_PIN_2) //PA1 - Turn on LED
8    {
9      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, 0);
```

```
10   }
11}
```

Cách 2: Viết nội dung ngắt riêng cho từng line ngay tại các hàm phục vụ ngắt theo line trong file thư viện **file stm32f1xx_it.c**

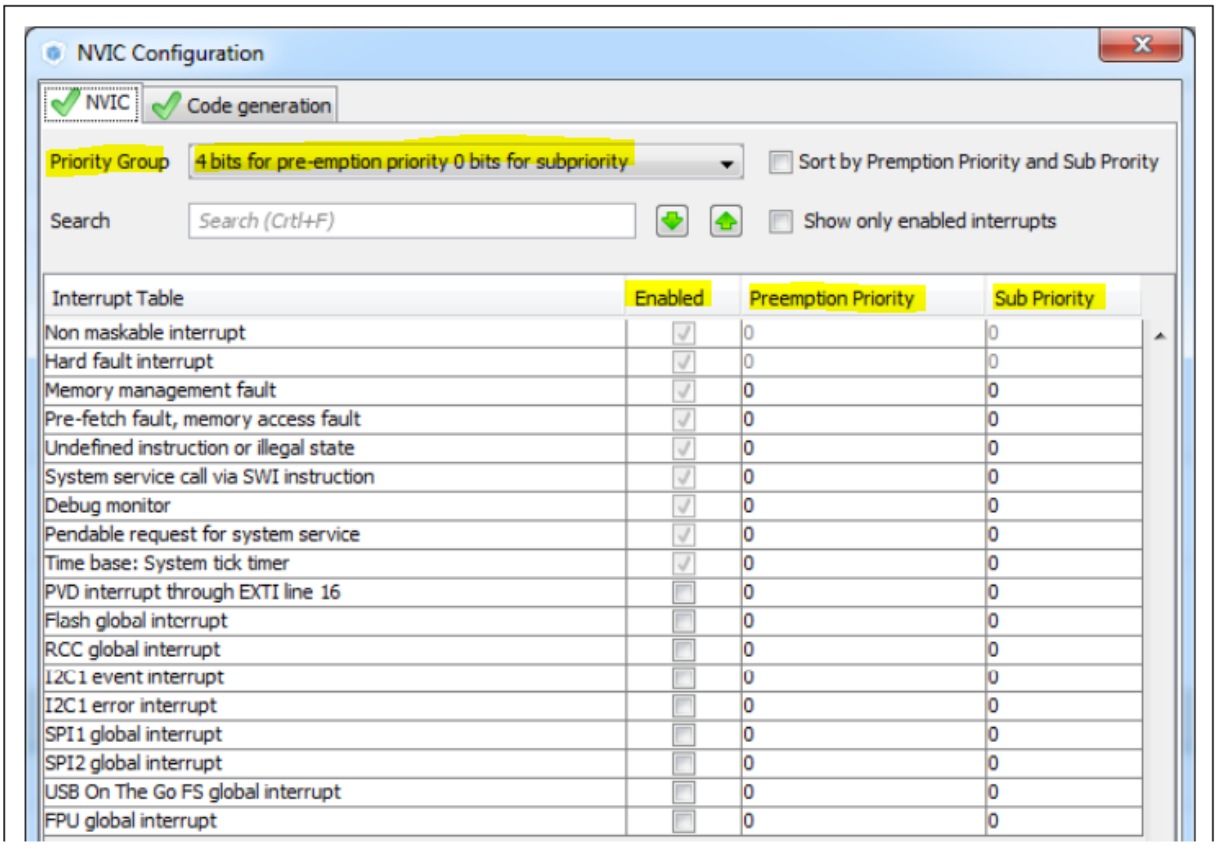
```
1  //PA1 - Turn on LED
2  void EXTI1_IRQHandler(void)
3  {
4    /* USER CODE BEGIN EXTI1_IRQn 0 */
5   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, 1);
6    /* USER CODE END EXTI1_IRQn 0 */
7   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
8    /* USER CODE BEGIN EXTI1_IRQn 1 */
9
10  /* USER CODE END EXTI1_IRQn 1 */
11}
12
13//PA2 - Turn off LED
14void EXTI2_IRQHandler(void)
15{
16  /* USER CODE BEGIN EXTI2_IRQn 0 */
17 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, 1);
18  /* USER CODE END EXTI2_IRQn 0 */
19 HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);
20  /* USER CODE BEGIN EXTI2_IRQn 1 */
21
22  /* USER CODE END EXTI2_IRQn 1 */
```

23}

Cấu hình ưu tiên ngắt vi điều khiển STM32 trên CubeMX

Vi điều khiển STM32 hỗ trợ rất nhiều ngắt khác nhau(interrupts) và chúng được quản lý bởi bộ NVIC (Nested Vector Interrupt Controller). Vậy chuyện gì xảy ra nếu có 2 yêu cầu ngắt đang chờ để được phục vụ hoặc nếu một trình phục vụ ngắt (ISR) đang được thực hiện và có 1 yêu cầu ngắt khác xuất hiện? Người lập trình hoàn toàn có thể cấu hình được các ưu tiên ngắt để xử lý các tình huống trên theo mong muốn bằng cách cấu hình NVIC trên phần mềm STM32CubeMX.

NVIC Configuration tab - FreeRTOS disabled



Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
I2C1 event interrupt	☐	0	0
I2C1 error interrupt	☐	0	0
SPI1 global interrupt	☐	0	0
SPI2 global interrupt	☐	0	0
USB On The Go FS global interrupt	☐	0	0
FPU global interrupt	☐	0	0

Ở cửa sổ trên, chúng ta có thể thấy có 2 loại ưu tiên ngắt là Preemption Priority và Sub Priority với ý nghĩa như sau:

– Ngắt nào có Preemption Priority cao hơn thì sẽ được ưu tiên thực hiện trước. Nếu một ngắt Preemption Priority thấp hơn đang trong quá trình được thực thi mà có một ngắt có Preemption Priority cao hơn yêu cầu thì ngắt có Preemption Priority cao hơn sẽ chiếm dụng vi xử lý, ngắt có Preemtion Priority thấp hơn sẽ tạm ngưng thực thi.

– Nếu một ngắt đang thực thi, một ngắt khác có cùng Preemption Priority và có Sub Priority cao hơn yêu cầu thì ngắt đến sau dù có Sub Priority cao hơn những vẫn sẽ không chiếm dụng Vi xử lý. Vi xử lý vẫn tiếp tục thực hiện ngắt có Subpriority thấp hơn. Sub Priority có ý nghĩa khi có nhiều ngắt đang ở trạng thái chờ (pending).

– Nếu các ngắt có cùng Preemtion Priority đang ở trạng thái chờ (pending) thì ngắt nào có Sub Priority cao hơn thì sẽ được thực hiện trước.
– Nếu các ngắt có cùng Preemtion Priority và Sub Priority thì ngắt nào đến trước sẽ được phục vụ trước.
Lưu ý về các giá trị cấu hình: số có giá trị càng nhỏ thì ưu tiên ngắt càng cao.