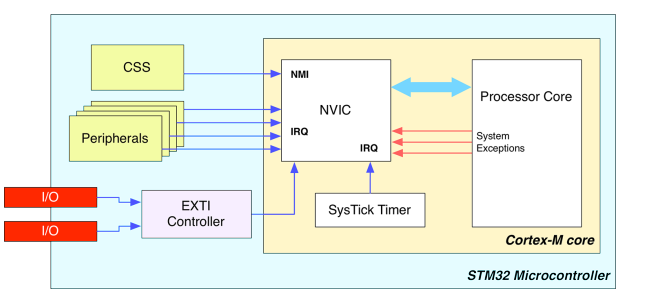
Ngắt

1. Ngắt là gì?
2. Ngắt trong STM32?
3. Bộ điều khiển NVIC:

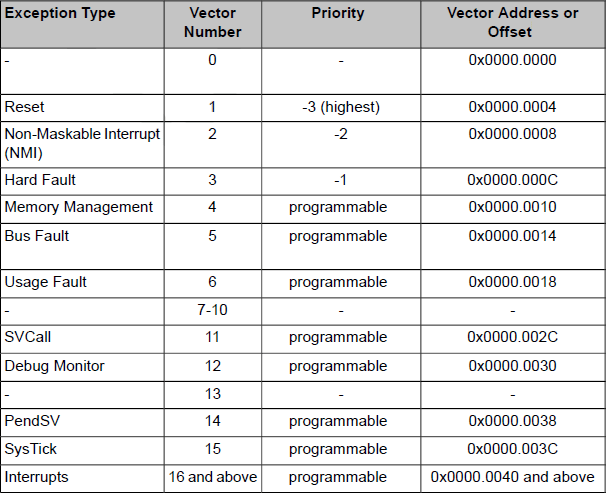
* NVIC là 1 đơn vị phần cứng chuyên dụng nằm trong vi điều khiển lõi ARM Cortex-M. **Hình 1** cho ta thấy mối quan hệ giữa NVIC, lõi vi xử lý và các thiết bị ngoại vi. Bộ điều khiển ngắt ngoài (EXTI controller), sẽ chịu trách nhiệm kết nối giữa tín hiệu I/O và bộ điều khiển NVIC

*Hình 1: Mối quan hệ giữa NVIC, lõi vi xử lý & ngoại vi*

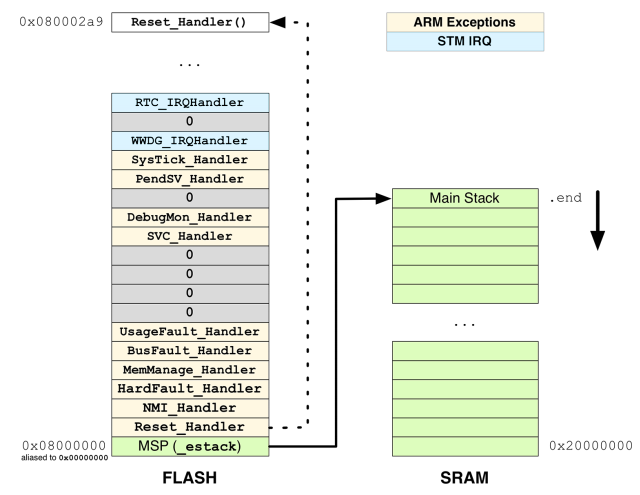
* Ta có thể hiểu đơn giản, *IRQ (Interrupts Requests)* hay còn gọi là yêu cầu ngắt là những “ngoại lệ” xảy ra trong lõi CPU hay từ các ngoại vi trong quá trình chạy code. Và khi có 1 yêu cầu ngắt, chương trình sẽ chạy đến thực hiện các câu lệnh được chỉ định từ trước được gọi là ***trình phục vụ ngắt*** (*ISRs* - *Interrupt Service Routine).* Bộ vi xử lý sẽ biết địa chỉ, nơi mà chứa trình phục vụ ngắt. Khi có ngắt bất kì, vi xử lý sẽ trỏ tới địa chỉ chứa trình phục vụ ngắt tương ứng để thực thi. Và các địa chỉ đó được quy định trong 1 bảng được gọi là *vector table.* Hãy cùng phân tích *vector table* của STM32f103C8.

1. Vector Table:

* Tất cả bộ vi xử lý Cortex-M đều định nghĩa 1 tập hợp các ngoại lệ cố định (15 đối với Cortex-M3/4/7 13 đối với M0). Dưới đây là các Ngoại lệ trong STM32F103:
  + **Reset:** Ngoại lệ này được đưa lên ngay khi CPU resets. Lệnh xử lý của nó trỏ thẳng vào firmware. Tất cả các dòng STM32 đều bắt đầu từ ngoại lệ này.
  + **NMI:** Đây là 1 ngoại lệ đặc biệt, có mức ưu tiên cao thứ 2. Cũng như Reset, đây là ngắt không thể che. Trong STM32, nó được kết nối tới Clock Security System (CSS). CSS là một ngoại vi có chức năng chẩn đoán lỗi của HSE (thạch anh ngoại tốc độ cao). Nếu có lỗi xảy ra, HSE sẽ tự động tắt (đồng nghĩa với việc HSI (thạch anh nội tốc độ cao) sẽ tự động kích hoạt). Và ngắt **NMI** được đưa lên để thông báo có vấn đề gì đó xảy ra với HSE.
  + **Hard Fault:** Đây là 1 lỗi ngoại lệ chung. Vì thế nó có dính dáng tới ngắt trong phần mềm. Khi các lỗi ngoại lệ khác không kích hoạt. Nó hoạt động như bộ thu thập các loại ngoại lệ
  + **Memory management fault:** Xảy ra khi mã thực thi cố gắng truy cập vào một vùng nhớ mà vi phạm quy tắc của Memory Protection Unit (MPU).
  + **Bus Fault:** Xảy ra khi AHB nhận được phản hồi từ bus slave, hoặc khi truy cập bất hợp pháp (ví dụ truy cập vào vị trí bộ nhớ SRAM không tôn tại).
  + **Usage Fault:** Xảy ra khi có lỗi chương trình. Chẳng hạn như khi truy cập vào bộ xử lý không tồn tại.
  + **SVCCall:** Đây không phải là một lỗi, nó được kích hoạt khi SVC (Supervisor Call) được gọi. Điều này được sử dụng bởi *Hệ điều hành thời gian thực (Real Time Operating Systems).*
  + **Debug Monior:** Ngoại lệ này xảy ra khi bật chế độ gỡ lỗi phần mềm. Nó cũng được sử dụng làm ngoại lệ cho các sự kiện gỡ lỗi như *Breakpoints, watchpoints,…*
  + **PendSV:** Đây là 1 ngoại lệ khác trong *Hệ điều hành thời gian thực.* Tuy nhiên nó có thể bị hoãn lại. điều này cho phép RTOS hoàn thành các nhiệm vụ với mức độ ưu tiên cao hơn.
  + **SysTick:**Ngoại lệ này cũng thường liên quan đến RTOS. Mỗi RTOS cần một bộ đếm thời gian để ngắt định kì việc thực thi mã hiện tại và chuyển sang tác vụ khác.
* Các ngoại lệ còn lại được xác định cho một MCU nhất định. Cortex-M0/0+ cho phép 32 ngắt ngoài. Trong khi Cortex-M3/4/7 cho phép định nghĩa lên tới 240 ngắt.
* Ngay cả khi *vector table* chứa địa chỉ của trình xử lý ngắt. Lõi Cortex-M vẫn cần cách để tìm *vector table* bên trông bộ nhớ. Theo quy ước, *vector table* bắt đầu tại địa chỉ hardware là 0x0000 0000. Nếu *vector table* nằm trong bộ nhớ flash, và vì bộ nhớ flash của STM32 ánh xạ từ địa chỉ 0x0800 0000, nên nó sẽ được đặt từ địa chỉ 0x0800 0000 ( có thể hiểu nôm na, 0x0000 0000 là địa chỉ đầu tiên của bộ nhớ chứa *vector table,* tùy mỗi dòng vi điều khiển mà sẽ có địa chỉ cụ thể khác nhau)



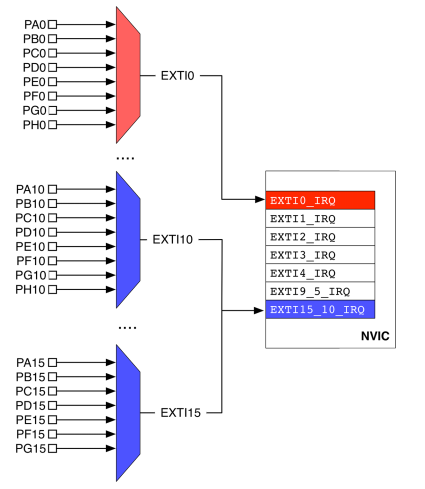
* **Hình 2** cho ta thấy cách mà *vector table* được tổ chức trong bộ nhớ. Mục ban đầu của mảng này là địa chỉ của con trỏ ngăn xếp chính (MSP) bên trong SRAM. Từ mục thứ 2 ta sẽ thấy các ngoại lệ và trình xử lý ngắt.
* Một số điều cần phải làm rõ về *vector table:*
  + Tên của các trình xử lý ngoại lệ (biểu thị trong **hình 2)** chỉ là một quy ước và bạn hoàn toàn có thể tự do đổi tên chúng nếu thích.



*Hình 2: Bố cục của vector table*

1. EXTI và NVIC:

* Như đã nói ở trên. STM32 cung cấp 1 lượng nguồn ngắt ngoài được kết nối với NVIC thông qua EXTI controller. Số lượng nguồn ngắt sẽ tùy thuộc vào họ STM32 cụ thể
* GPIO kết nối với các EXTI line. Và có thể cho phép ngắt mọi GPIO MCU ngay cả khi hầu hết chúng chỉa sẻ cùng 1 dòng ngắt. Ở đây ta sẽ lấy ví dụ với STM32F4. Có tận 114 GPIO kết nối với 16 EXTI line. Nhưng chỉ 7 trong số đó là có ngắt độc lập liên kết với chúng.
* **Hình 3** cho ta thấy EXTI lines 0, 10 và 15 trong STM32F4. Tất cả các Px0 liên kết với EXTI0, tât cả các Px10 thì nối với EXTI10, tương tự với Px15. Tuy nhiên EXTI10 và 15 lại dùng chung cùng IRQ trong NVIC (và do đó nó sẽ được phục vụ bởi cùng 1 ISR).
* Điều này có nghĩa:
  + Chỉ 1 PxY có thể là 1 nguồn ngắt. Ví dụ, chúng ta không thể cấu hình PA0 và PB0 đều là chân ngắt.
  + Đối với các EXTI line chia sẻ cùng IRQ trong NVIC. Chúng ta phải giải mã ISR tương ứng để chúng có thể phân biệt được line nào tạo ra ngắt

*Hình 3: mối quan hệ giữa GPIO, EXTI và NVIC*

1. Lập trình ngắt trong STM32:

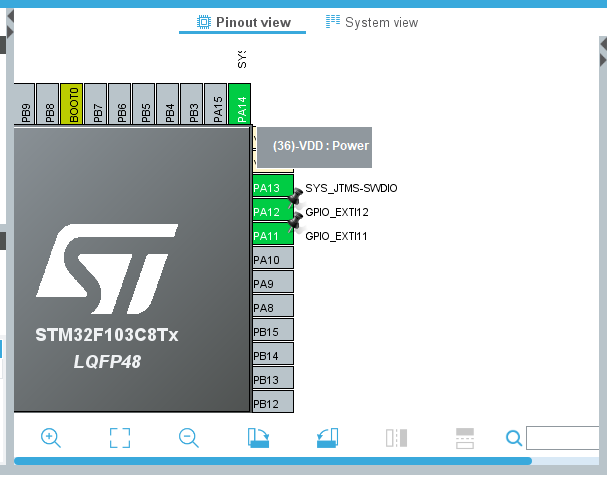
Ở trong ví dụ này, chúng ta sẽ kết hợp với bài trước, điều khiển led bằng nút nhấn có sử dụng ngắt ngoài trên STM32 start kit (link mua sản phẩm: <https://banlinhkien.com/kit-stm32f103c8t6-ban-start-p6649308.html>).

1. Khởi tạo project với cubeMx:

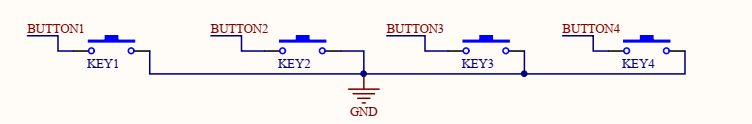
B1: Bước đầu tiên ta sẽ làm tương tự như bài 1 (cấu hình thạch anh, chế độ debug, …).

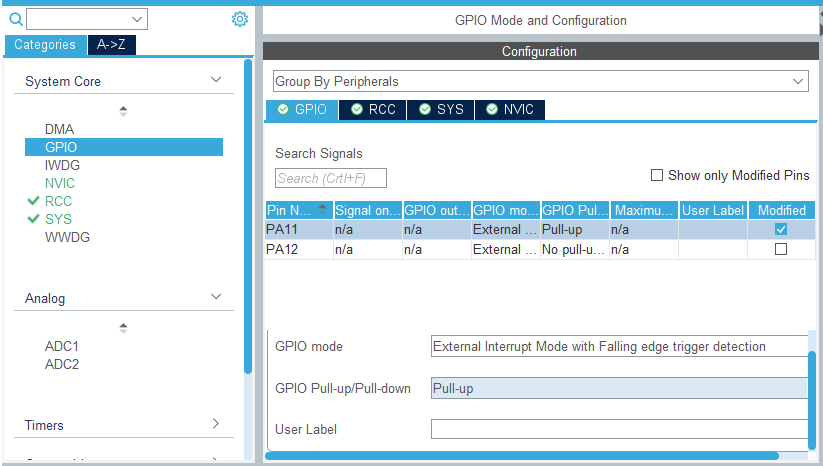
B2: Cấu hình chân ngắt:

* Ở đây ta sẽ cấu hình 2 chân PA12, và PA11. Click vào 2 chân và chọn GPIO\_EXTIx.

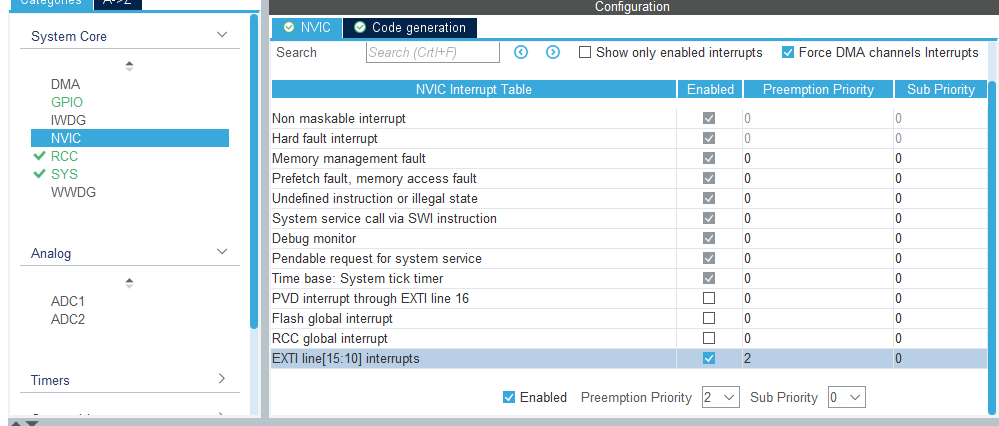


* Cấu hình ngắt cho từng chân. Ở đây ta thấy có 6 mode ngắt, theo thứ tự từ trên xuống dưới lần lượt là:
  + Ngắt ngoài sườn lên
  + Ngắt ngoài sườn xuống
  + Ngắt ngoài sườn lên hoặc xuống
* Vì trong schematic, khi nhấn nút thì GPIO sẽ = 0, nên ta sẽ chọn ngắt theo sườn xuống và chọn trở kéo lên (pull-up).

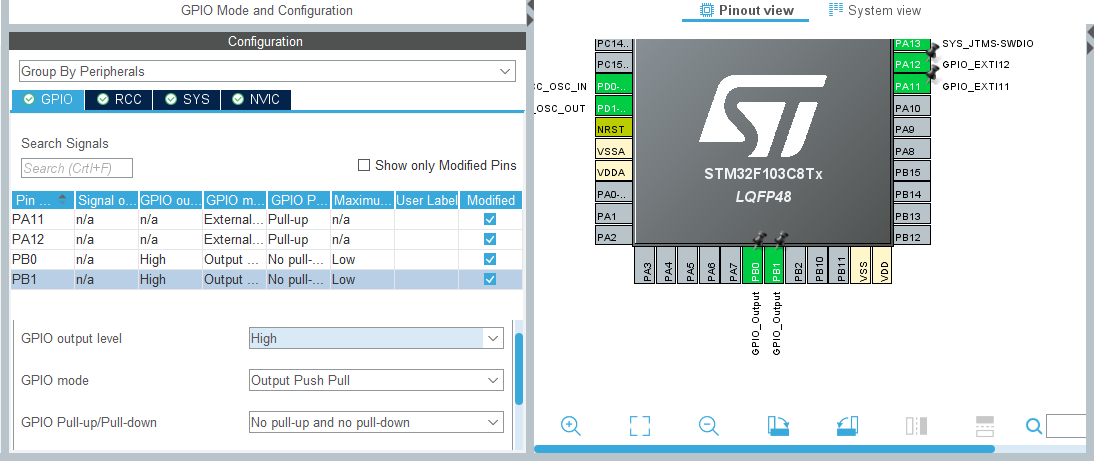




B3: Kích hoạt ngắt và mức độ ưu tiên.

* Ở đây có 3 cột, tương ứng là:
  + Cho phép ngắt
  + Mức ưu tiên chính
  + Mức ưu tiên phụ ( đối với 2 ngắt có mức ưu tiên chính bằng nhau, ngắt nào có mức ưu tiên phụ cao hơn thì được ưu tiên thực thi trước).
* Click chọn enable cho EXTI line[15:10], ở đây ta có thể thấy tuy dùng 2 chân ngắt nhưng chỉ có 1 EXTI line, điều đó chứng tỏ 2 chân ngắt này dùng chung 1 NVIC IRQ (đọc lý thuyết ở phần trên).
* Lưu ý khi chọn mức độ ưu tiên, ta không nên chọn mức 0 trừ khi nó quá quan trọng, số thứ tự càng thấp thì độ ưu tiên càng cao. Và mức 0 được fix cứng cho các ngắt hệ thống nên hạn chế làm ảnh hưởng tới chúng.

B4: Cấu hình 2 chân PB0, PB1 làm 2 chân điều khiển LED như trong ảnh



B5: Đặt tên project và GENERATE CODE.

1. Lập trình trên Keil C:

B1: Cho nháy LED0 trong hàm main với chu kỳ 1s.

int main(void)

{

  HAL\_Init();

  SystemClock\_Config();

  MX\_GPIO\_Init();

  while (1)

  {

    HAL\_GPIO\_TogglePin(GPIOB, GPIO\_PIN\_0);

    HAL\_Delay(1000);

  }

}

B2: Lôi hàm phục vụ ngắt ra và code. Khi có ngắt ngoài xảy ra thì chương trình sẽ nhảy tới hàm này và thực thi các câu lệnh trong đây