Chức năng ADC trên vi điều khiển STM32F4

I, ADC là gì? Tác dụng của nó như thế nào?

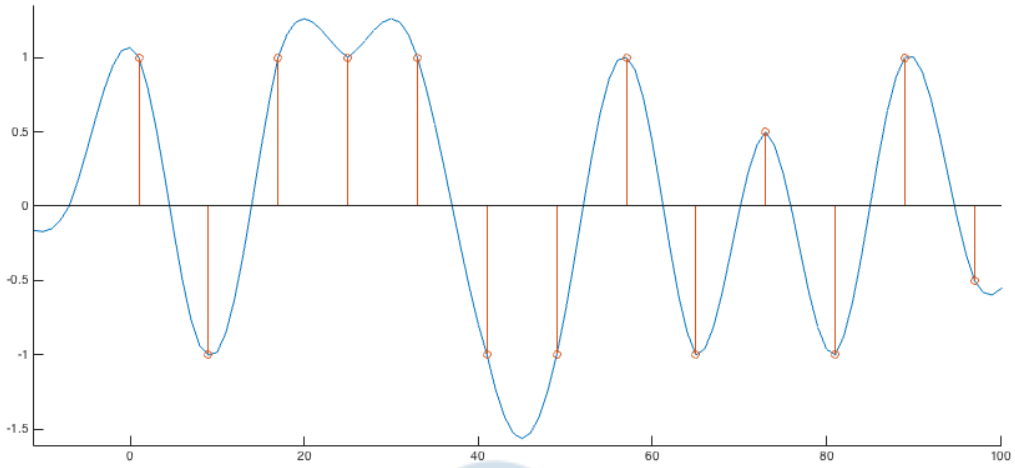
Các tín hiệu chúng ta thường gặp trong tự nhiên như điện áp, ánh sáng, âm thanh, nhiệt độ… đều tồn tại dưới dạng tương tự (Analog), có nghĩa là tín hiệu liên tục và mức độ chia nhỏ vô hạn. Ví dụ: trong khoảng điện áp từ 0 -> 5V có vô số khoảng giá trị điện áp, ánh sáng sẽ tồn tại từ mờ cho tới sáng tỏ, âm thanh từ nhỏ cho đến lớn dưới dạng liên tục.

Ngược lại trong vi điều khiển chỉ có khái niệm số (Digital), cấu trúc từ nhân cho đến bộ nhớ hoạt động dựa trên các Transistor chỉ gồm mức 0-1 nên nếu muốn giao tiếp với chip thì tín hiệu phải được số hóa trước khi đưa vào chip. Quá trình số hóa có thể thực hiện bằng nhiều cách và nhiều công đoạn nhưng mục đích cuối cùng là để vi điều khiển hiểu được tín hiệu tương tự đó.

ADC (Analog-to-Digital Converter) bộ chuyển đổi tín hiệu tương tự - số là thuật ngữ nói đến sự chuyển đổi một tín hiệu tương tự thành tín hiệu số để dùng trong các hệ thống số (Digital Systems) hay vi điều khiển.

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. Bộ chuyển đổi ADC của STM32F407VG 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 đồ thị dưới đây, 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.



Để hiểu quá trình số hóa trong STM32 diễn ra như thế nào ta theo dõi ví dụ sau. Giả sử ta cần đo điện áp tối thiểu là 0V và tối đa là 3.3V, trong STM32 sẽ chia 0 → 3.3V thành 4096 khoảng giá trị (từ 0 → 4095, do $2^{12} = 4096$), giá trị đo được từ chân IO tương ứng với 0V sẽ là 0, tương ứng với 1.65V là 2047 và tương ứng 3.3V sẽ là 4095.

Trong STM32F407VG có 3 bộ ADC chuyển đổi tín hiệu tương tự thành tín hiệu số với độ phân giải 12-bit. Mỗi ADC có khả năng tiếp nhận tín hiệu từ 16 kênh ngoài.

Ở bài viết này, chúng ta sẽ sử dụng ADC1 channel 0 để đọc điện áp từ một biến trở đưa vào.

II, Thực hành

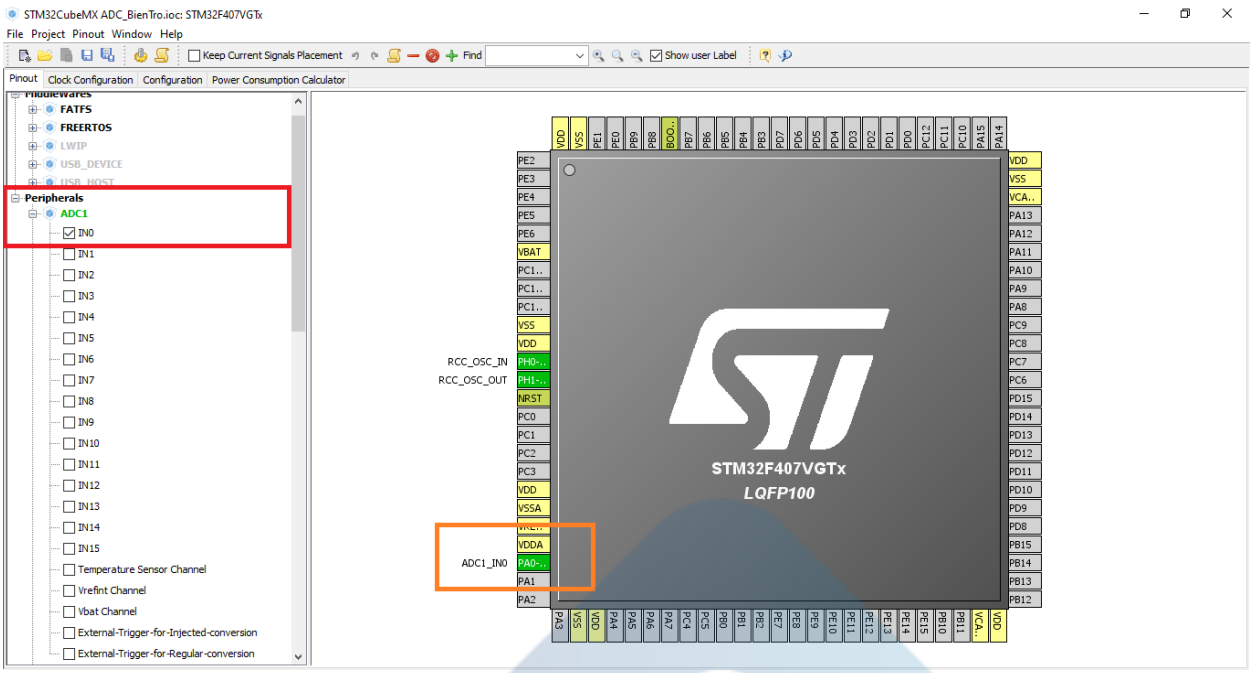
1, Cấu hình với CubeMX

Khởi động CubeMX, chọn chip mà chúng ta sử dụng. Ở đây mình sẽ chọn STM32F407VG.

Tại mục "**Peripherals**", các bạn chọn ADC1 và click chọn IN0, thao tác này đã kích hoạt ADC1 channel 0 trên chip.

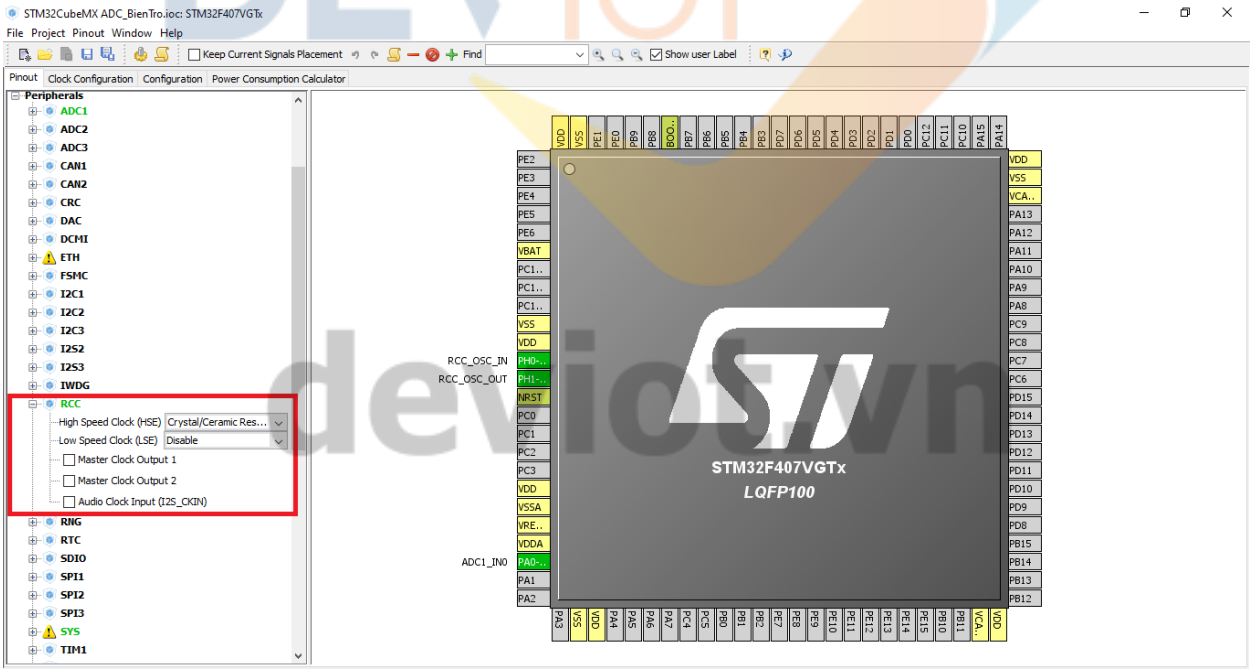
Hoặc chúng ta có thể tìm đến chân PA0 và chọn chế độ ADC1_IN0 .

(Chân PA0 là chân được liên kết sẵn với channel 0 của bộ ADC1)



Cấu hình thạch anh và xung clock cho chip:

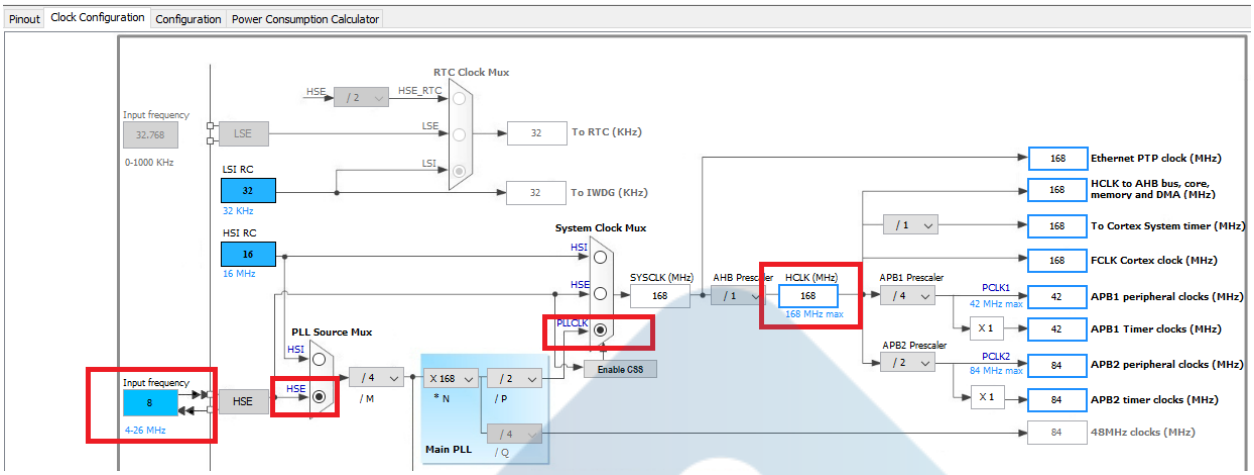
Tại mục RCC, trong phần "**High Speed Clock(HSE)**", chúng ta chọn "**Crystal/Ceramic Resonator**" để cấu hình cho chip hoạt động với thạch anh ngoại.



Tại Tab "**Clock Configuration**", chúng ta thiết lập các thông số Clock cho Chip.

Chọn HSE, xung Clock sẽ đi qua bộ nhân tần PLLCLK, giúp Chip hoạt động với tần số tốt nhất.

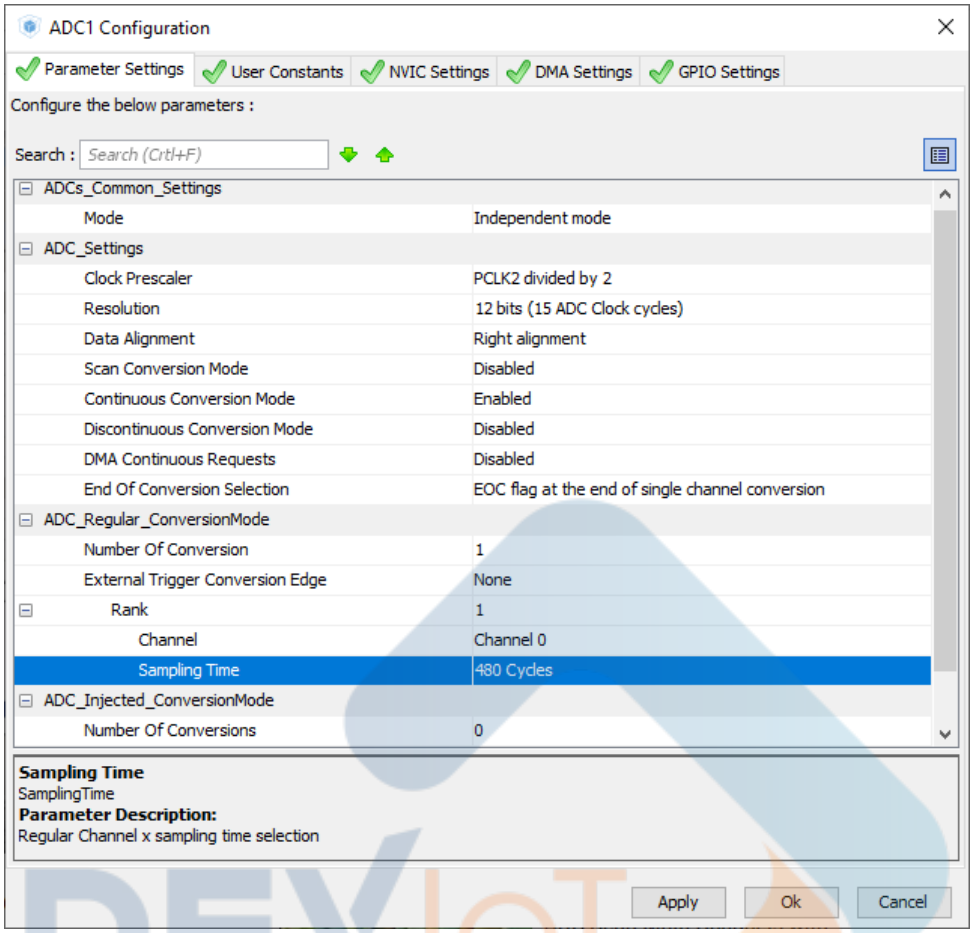
Điền “8” ở mục “Input frequency” (thông số của thạch anh ngoại gắn trên kit) và điền “168” ở mục “HCLK” (168MHz là tần số tối đa của chip).



Cấu hình cho bộ ADC

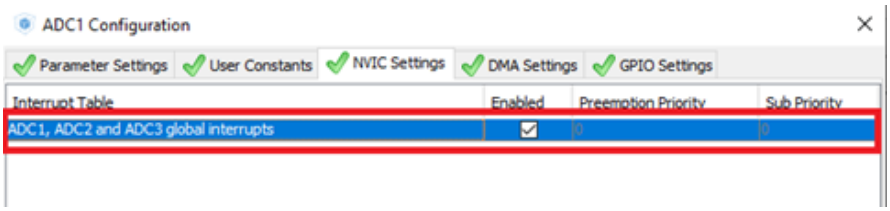
Trong Tab “Configuration”, chọn mục ADC1.

Khi cửa sổ “ADC1 Configuration” hiện lên, chúng ta thiết lập các thông số như sau:

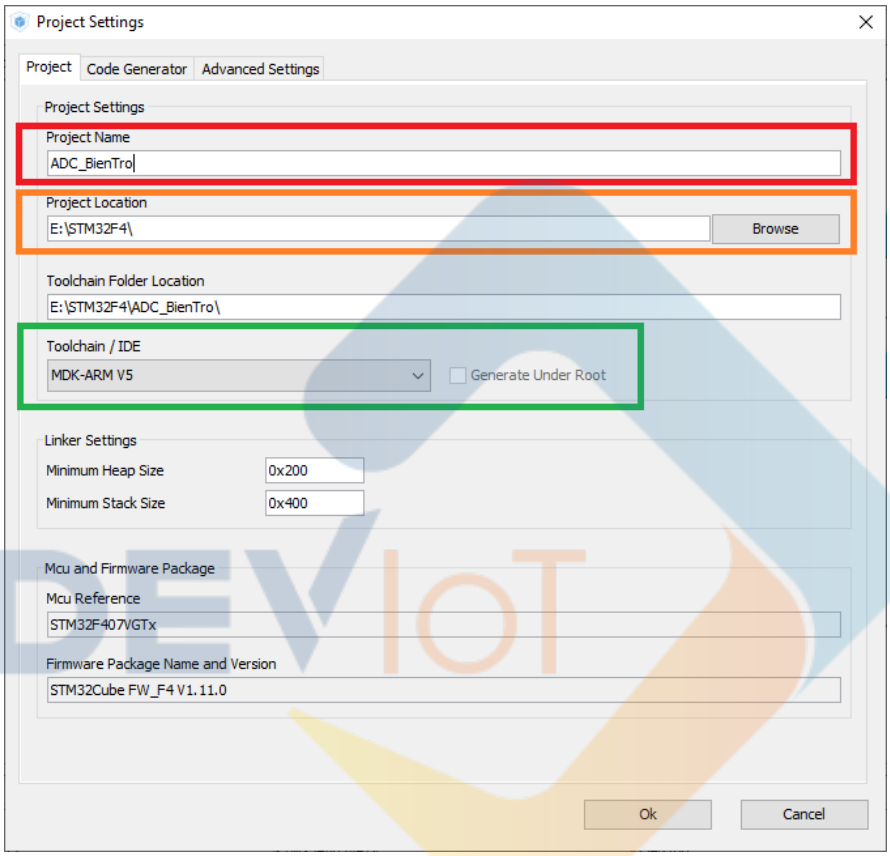


- **Resolution**: như đã giới thiệu ở trên, độ phân giải càng cao, quá trình chuyển đổi sẽ càng chính xác, vì vậy chúng ta chọn "12 bits (15 ADC Clock cycles".
- **Data Alignment**: vì độ phân giải là 12 bit, nên chúng ta cần 1 thanh ghi 16 bit để lưu dữ liệu, như vậy sẽ còn dư 4bit. Chúng ta chọn "Right alignment" tức là lưu 12bit dữ liệu dịch về phía bên phải của thanh ghi 16bit này.
- **Continuous Conversion Mode**: cho phép các quá trình chuyển đổi diễn ra liên tục, chúng ta sẽ "enable" chức năng này.
- **Sampling Time**: thời gian lấy mẫu, nếu thông số này càng lớn, độ chính xác càng lớn nhưng bù lại quá trình chuyển đổi sẽ diễn ra lâu hơn.

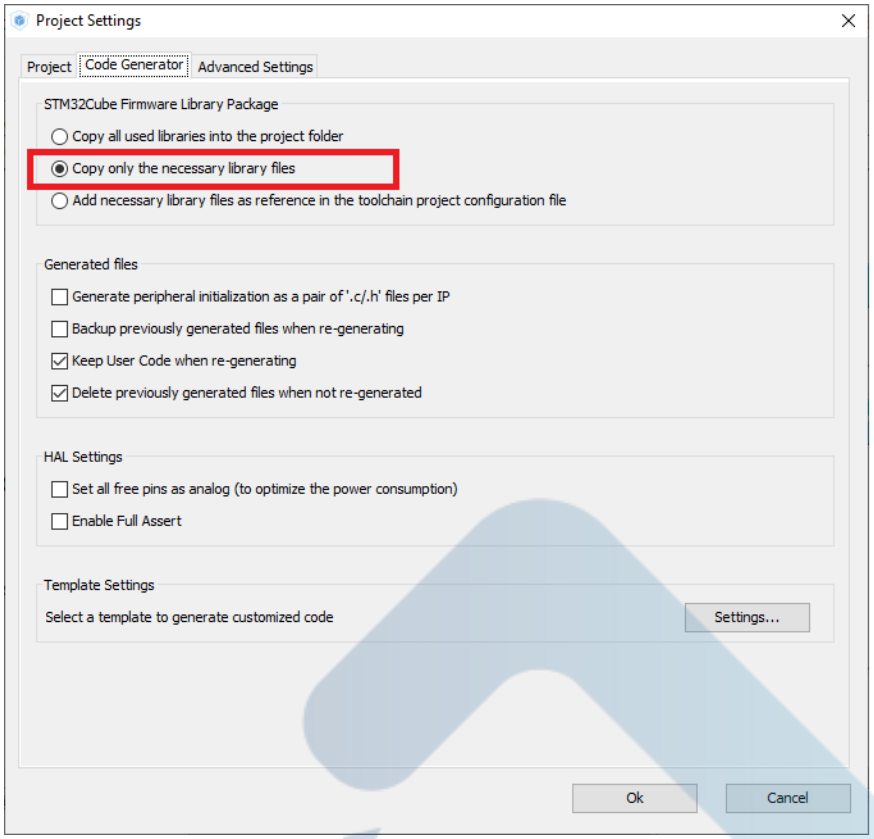
Trong Tab "**NVIC Settings**", chọn "**Enable**" để cho phép ngắt xảy ra trên bộ ADC.



Cuối cùng là Setting Project và sinh code.



Ở Tab "**Code Generator**", các bạn chọn "**Copy only the necessary library files**" để project tạo ra chỉ với những thư viện cần thiết, tiết kiệm dung lượng và thời gian build code nhé!



2, Lập trình với KeilC

Sau khi CubeMX sinh code xong, các bạn "Open Project" để mở chương trình trên KeilC

Trong file "main.c", mình sẽ khai báo 1 biến toàn cục `volatile uint16_t adc_value = 0;`

Biến "adc_value" này mình sẽ dùng để nhận giá trị chuyển đổi của bộ ADC.

Ở đây, "`volatile`" có tác dụng tránh những lỗi sai khó phát hiện do tính năng Optimization của Compiler gây ra vì giá trị của biến "adc_value" sẽ bị thay đổi đột ngột do tác động của phần cứng. Khi Compiler nhận định biến này không hề có sự thay đổi giá trị do phần mềm nó sẽ coi như đây là 1 biến không được sử dụng và loại bỏ nó trong quá trình Optimize.

Trong hàm main, chúng ta thêm câu lệnh sau HAL_ADC_Start_IT(&hadc1);

Câu lệnh này đã cho phép ngắt và bắt đầu chuyển đổi với các kênh của bộ ADC1.

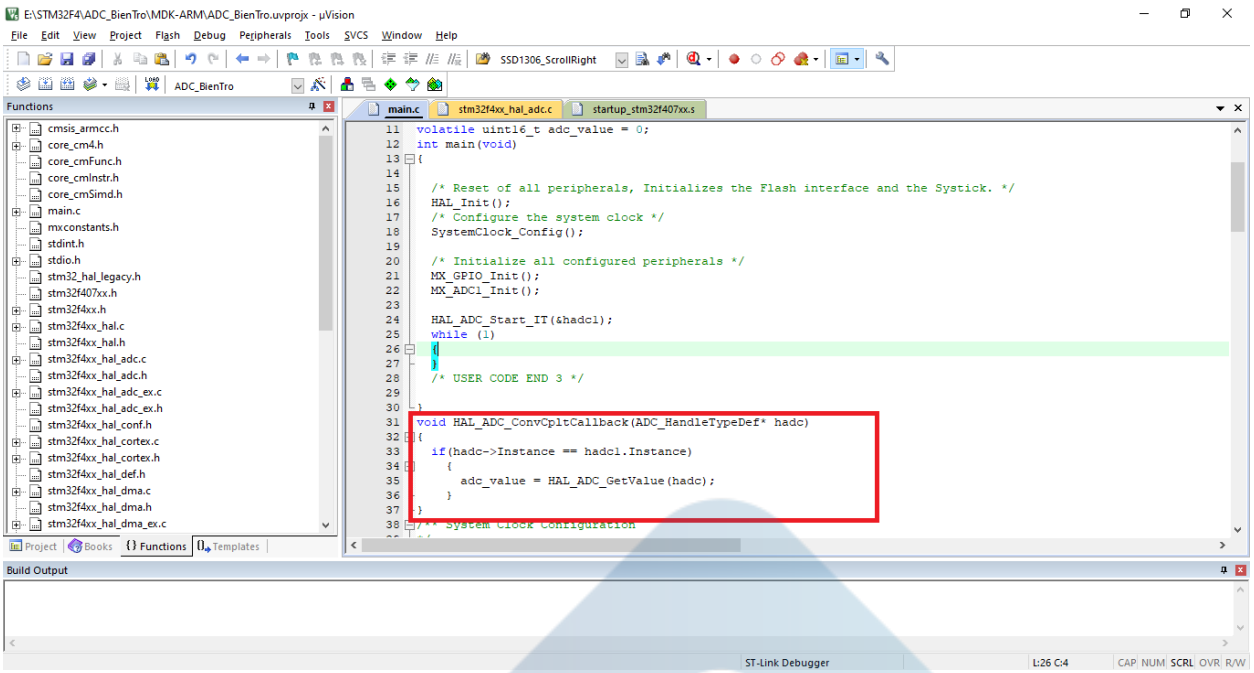
```
volatile uint16_t adc_value = 0;
int main(void)
{

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  HAL_ADC_Start_IT(&hadc1);

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

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();

  while (1)
  {

  }
```

Tại mục "**Functions**", các bạn tìm đến file "**stm32f4xx_hal_adc.c**" chọn hàm "**HAL_ADC_ConvCpltCallback**". Hàm này là hàm phục vụ ngắt ADC, được gọi đến khi 1 quá trình chuyển đổi ADC hoàn thành.

Vì hàm này được khai báo với **__weak**, nếu muốn sử dụng nó, chúng ta phải khai báo sang 1 file khác. Ở đây mình sẽ khai báo và sử dụng nó trong file "main.c"

Chúng ta sẽ lập trình như sau với hàm này :

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  if(hadc->Instance == hadc1.Instance)
        {
            adc_value = HAL_ADC_GetValue(hadc);
        }
}
```



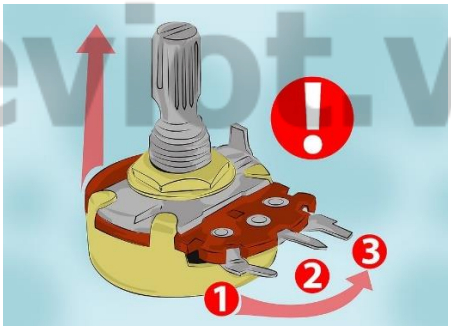
Câu lệnh `if(hadc->Instance == hadc1.Instance)` sẽ kiểm tra xem bộ ADC đang yêu cầu xử lý ngắt có phải là bộ ADC1 không.

Sau đó chúng ta thực hiện gán giá trị chuyển đổi từ bộ ADC1 về biến adc_value thông qua hàm `HAL_ADC_GetValue(hadc);`

Các bạn build chương trình (**F7**) và nạp code xuống kit (**F8**), nhấn nút reset KIT.

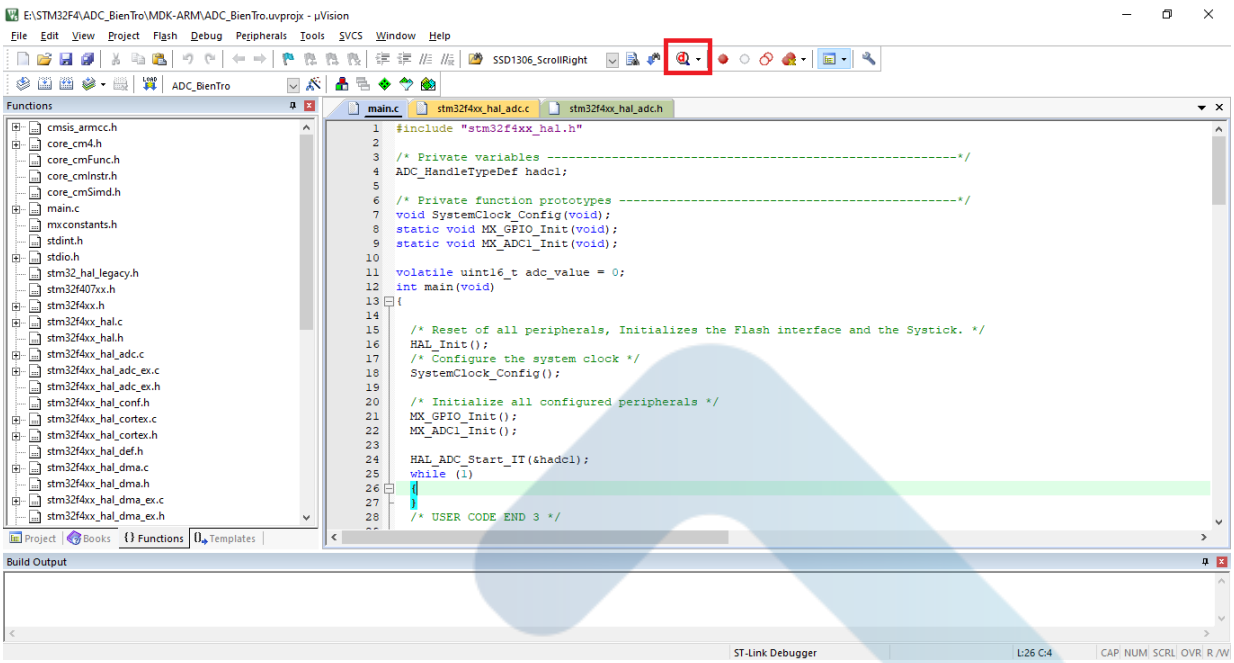
Kết nối biến trở với kit STM32F407 :

Chân 1 – GND
Chân 2 – PA0
Chân 3 – 3V



Debug và quan sát sự thay đổi giá trị của biến trở :

Trên thanh công cụ, chúng ta click vào icon Debug.

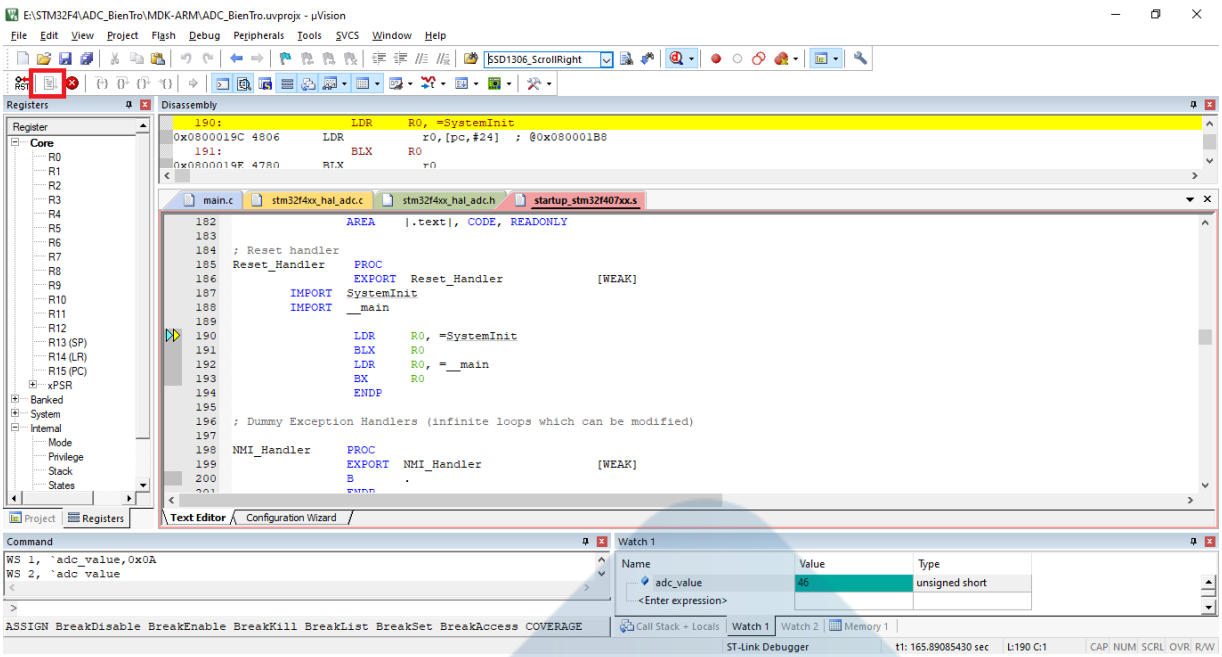


Khi cửa sổ Debug hiện lên, chúng ta tìm đến biến adc_value đã khai báo ở đầu chương trình

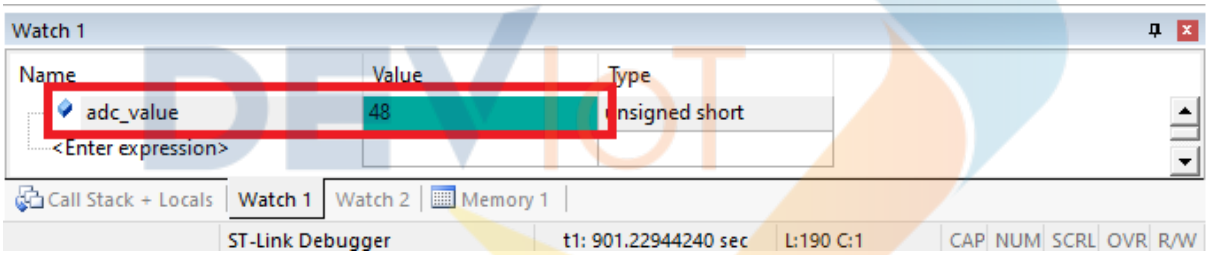
```
volatile uint16_t adc_value = 0;
```

Click chuột phải tại biến đó, chọn Add '**adc_value**' tới **Watch 1.**

Biến 'adc_value' đã được thêm vào cửa sổ **Watch 1**, các bạn bắt đầu chạy Debug bằng cách click vào icon **Run** trên thanh công cụ.



Tại cửa sổ **Watch 1**, để theo dõi giá trị của biến ở dạng số thập phân, chúng ta click chuột phải và "**adc_value**", bỏ tích tại "**Hexadecimal Display**".



Điều chỉnh biến trở và quan sát sự thay đổi giá trị của biến "**adc_value**"

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