Bài 11:Lập trình Timer cơ bản với STM32

Timer(Bộ định thời) là ngoại vi không thể thiếu đối với các dòng vi điều khiển. Đây là khối thực hiện nhiều chức năng quan trọng như làm bộ đếm, phát hiện, đo tín hiệu đầu vào, tạo xung PWM, điều khiển và cấp xung cho các thiết bị bên ngoài, định thời các sự kiện đặc biệt

I.Lý thuyết

Dòng vi điều khiển STM32 có ba loại Timer:

- **Basic Timer**: là loại Timer đơn giản và dễ sử dụng nhất, chỉ có chức năng đếm và thường được dùng để tạo cơ sở thời gian.
- **General Purpose Timer**: là loại Timer nhiều tính năng hơn Basic Timer, có đầy đủ các tính năng của một bộ định thời như đếm thời gian, tạo xung PWM, xử lí tín hiệu vào, so sánh đầu ra, ...
- **Advanced Timer**: đây là loại Timer nâng cao, mang đầy đủ đặc điểm của General Purpose Timer, ngoài ra còn có nhiều tính năng khác và độ chính xác cao hơn. Thường được sử dụng để làm bộ đếm thời gian cho hệ thống.

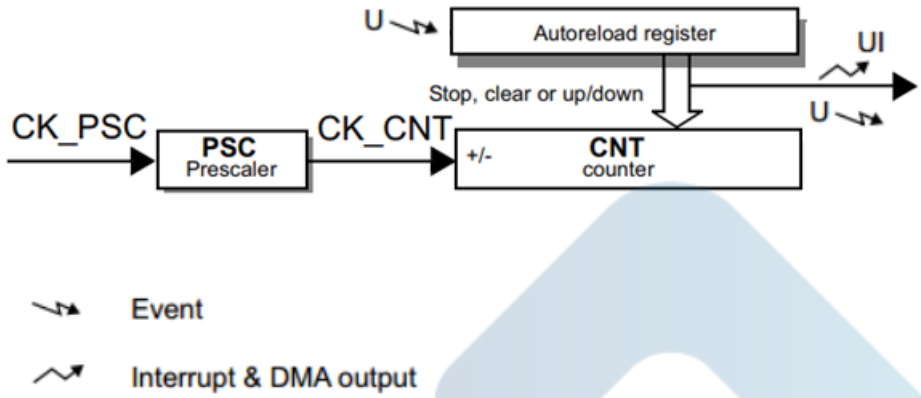
Ở bài này mình sử dụng vi điều khiển STM32L476VG, được tích hợp 11 bộ Timers:

Table 10. Timer feature comparison

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary outputs
Advanced control	TIM1, TIM8	16-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	3
General-purpose	TIM2, TIM5	32-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	No
General-purpose	TIM3, TIM4	16-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	No
General-purpose	TIM15	16-bit	Up	Any integer between 1 and 65536	Yes	2	1
General-purpose	TIM16, TIM17	16-bit	Up	Any integer between 1 and 65536	Yes	1	1
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No

Cấu trúc cơ bản của một bộ Timer

- Bộ đếm (Giá trị được lưu ở thanh ghi Counter Register)
- Giá trị Auto Reload (Giá trị được lưu ở thanh ghi Auto Reload)
- Bộ chia tần (Giá trị được lưu ở thanh ghi Prescaler)



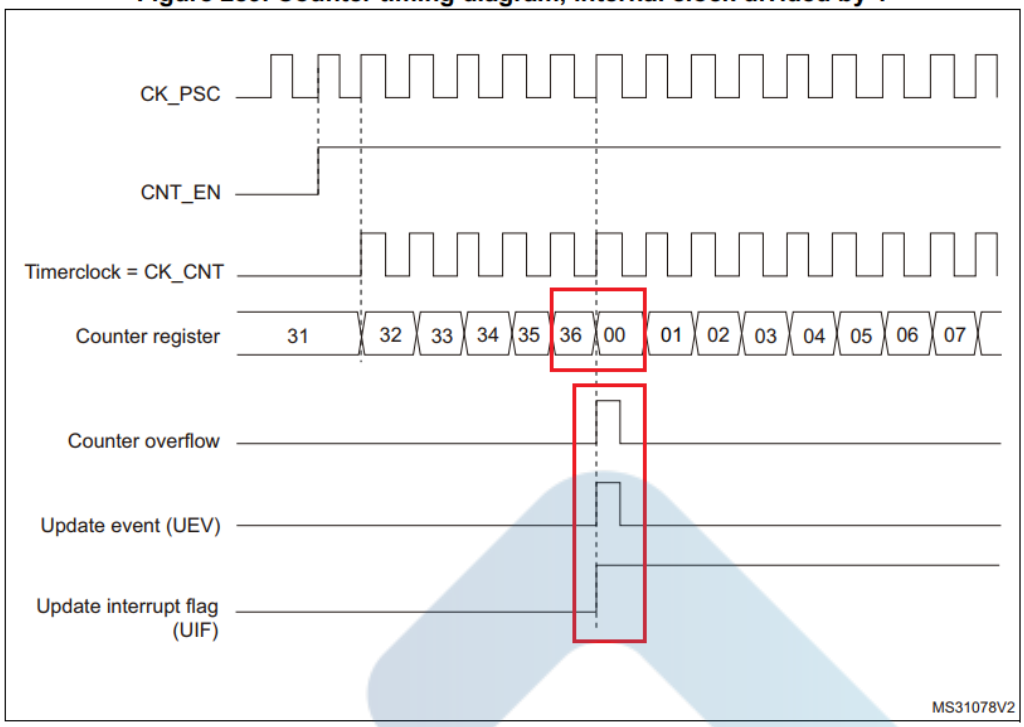
Các thanh ghi quan trọng:

- **Auto Reload**(TIMx_ARR): Lưu giá trị Auto Reload (ARR), là ngưỡng trên của giá trị đếm Counter.
- **Counter Register**(TIMx_CNT): Lưu giá trị đếm Counter (CNT), tăng hoặc giảm mỗi nhịp xung clock của Timer. Giá trị của Counter luôn nằm trong khoảng [0; ARR]. Nếu ngoài khoảng đó, Timer sẽ thực hiện nạp lại giá trị CNT như ban đầu và tiếp tục hoạt động. Tùy vào mỗi Timer mà CNT và ARR có cỡ 16 hoặc 32 bit.
- **Prescaler** (TIMx_PSC): Lưu giá trị chia tần PSC (16 bit), thuộc khoảng [1;65536]. Kết hợp việc sử dụng hai giá trị PSC và ARR, chúng ta có thể tính toán được tần số, chu kì đếm của Timer.

Các chế độ đếm của Timer:

Upcounting Mode: CNT bắt đầu tăng từ giá trị 0 đến giá trị ARR. Sau đó, CNT quay trở lại giá trị 0, đồng thời sinh ra sự kiện tràn trên(counter overflow event).

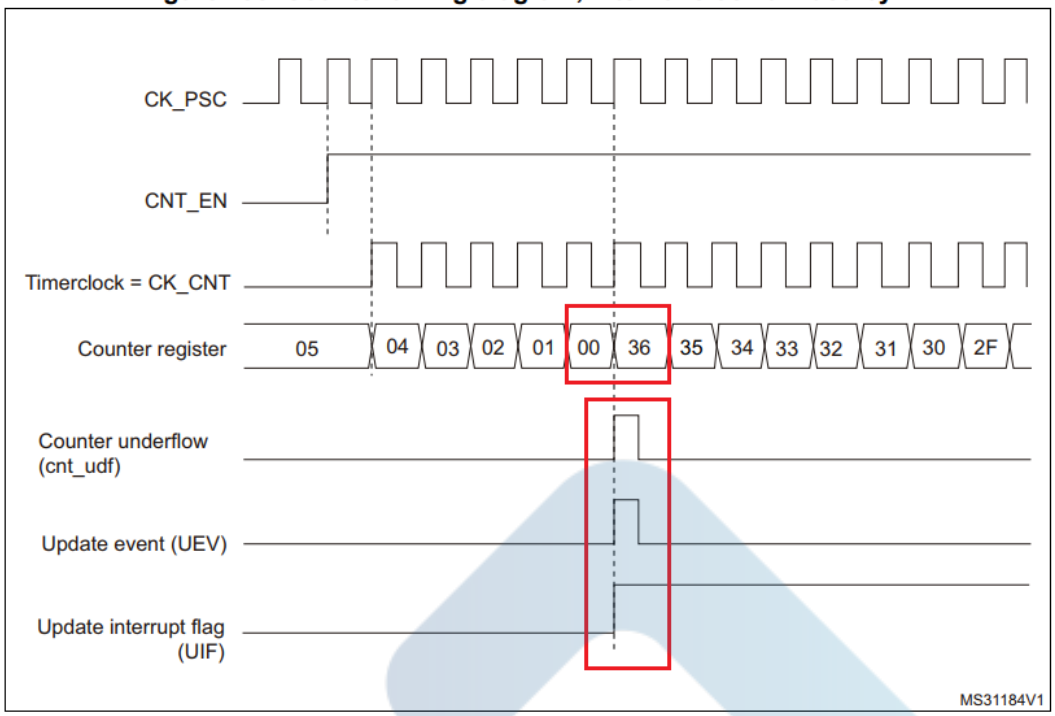
Figure 289. Counter timing diagram, internal clock divided by 1



Upcounting mode với ARR=36

Downcounting mode: CNT bắt đầu giảm từ giá trị ARR về 0. Sau khi về 0, CNT quay trở lại giá trị ARR, đồng thời sinh ra sự kiện tràn dưới(counter underflow event).

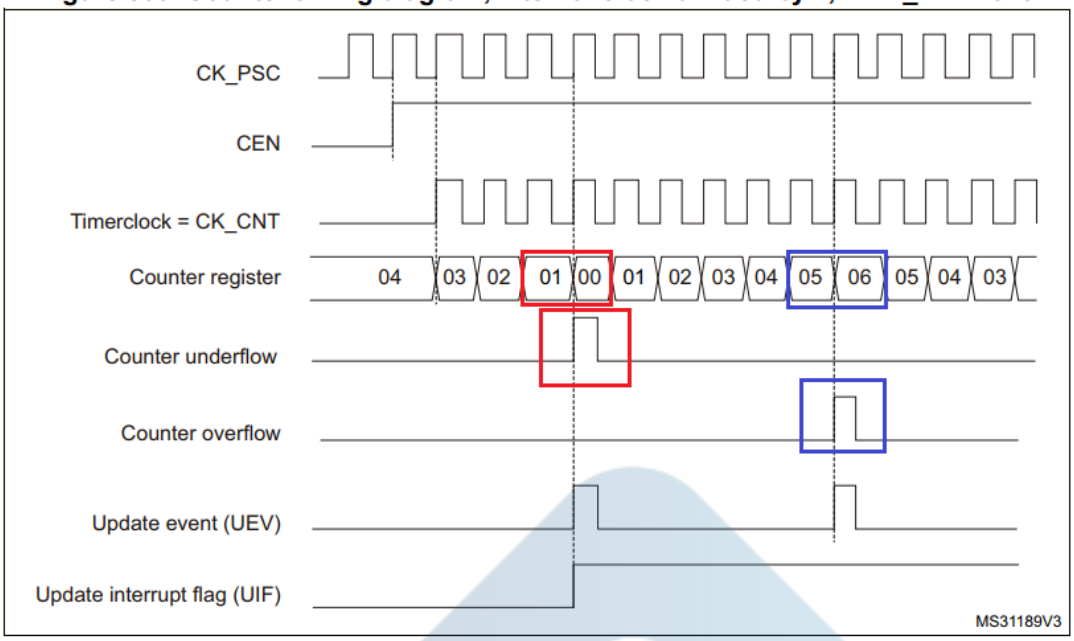
Figure 295. Counter timing diagram, internal clock divided by 1



Downcounting mode với ARR=36

Center-aligned mode (up/down counting): ở chế độ đếm này, CNT bắt đầu tăng từ 0, đến giá trị ARR-1, khi đó sinh ra sự kiện tràn trên(counter overflow event). Sau đó CNT đạt giá trị ARR và bắt đầu giảm về giá trị 1, khi đó sinh ra sự kiện tràn dưới(counter underflow event). Cuối cùng, CNT về giá trị 0 và quá trình trên lại tiếp tục.

Figure 300. Counter timing diagram, internal clock divided by 1, TIMx_ARR=0x6



Center-aligned mode với ARR=6

<u>Một vài chức năng thường xuyên được sử dụng của Timer:</u>

Ngoại trừ các Basic Timer chỉ có hoạt động cơ bản là đếm, các Timers còn lại của vi điều khiển còn có nhiều chức năng khác, điển hình như:

- **PWM Generation**: Tính năng điều chế độ rộng xung (băm xung).
- **One Pulse Mode**: Tạo ra một xung duy nhất với độ rộng có thể cấu hình được, CNT sẽ tự động dừng khi có sự kiện tràn.
- **Input Capture**: Chế độ này phát hiện và lưu lại sự xuất hiện sự thay đổi mức logic (sườn lên/ sườn xuống) của tín hiệu. Từ đó, ta có thể biết được khoảng thời gian giữa hai lần có sườn lên/ sườn xuống.
- **Output Compare**: Đây là chế độ giúp tạo ra các sự kiện(ví dụ như ngắt) khi CNT đạt đến giá trị được lưu trong các thanh ghi TIMx_CCMRx (capture/compare mode register). Ứng dụng phổ biến nhất của Output Compare là tạo ra nhiều xung PWM với các tần số khác nhau trên cùng một Timer.

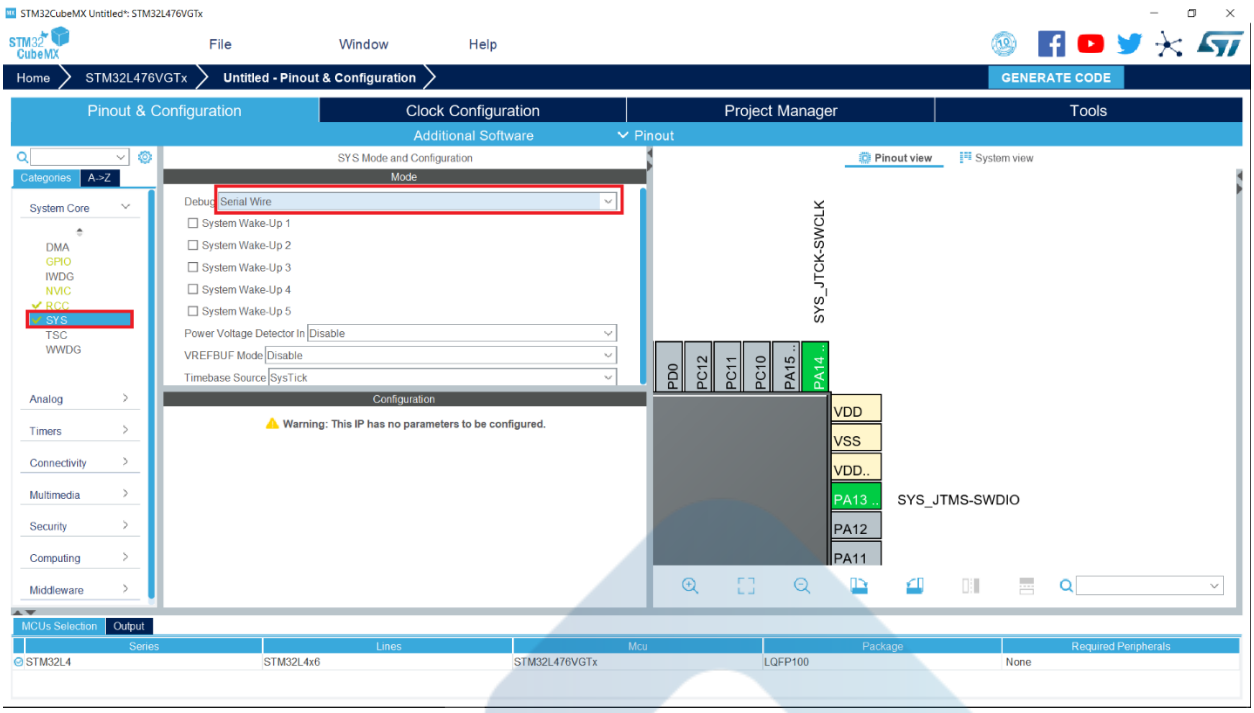
II. Lập trình cơ bản với Timer

Ở bài này, chúng ta sẽ làm ví dụ đơn giản nhất với Timer, đó là dùng Timer để điều khiển một LED sáng 1s, sau đó tắt 1s, rồi lại sáng, tắt, … Lập trình các tính năng nâng cao hơn như PWM, Input Capture, Output Compare, … sẽ được nói đến ở những bài sau.

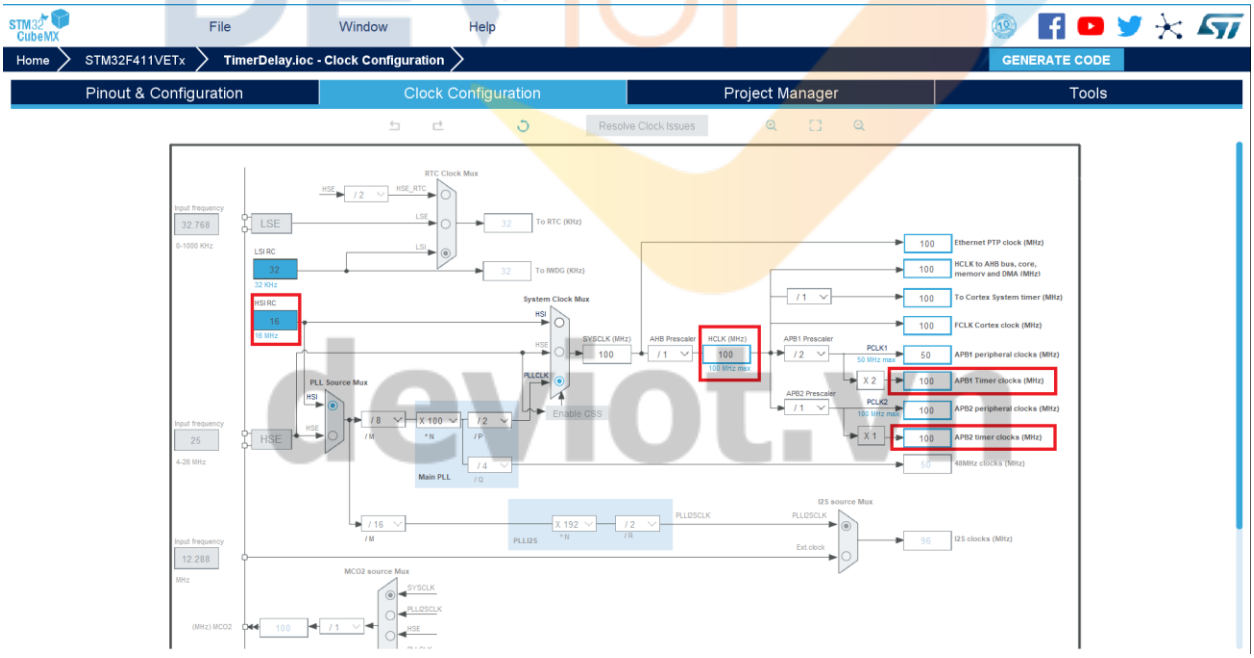
<u>Cấu hình Chip với Cube Mx:</u>

Mở phần mềm Cube Mx, chọn dòng chip các bạn sử dụng. Ở đây, mình chọn chip STM32L476VG.

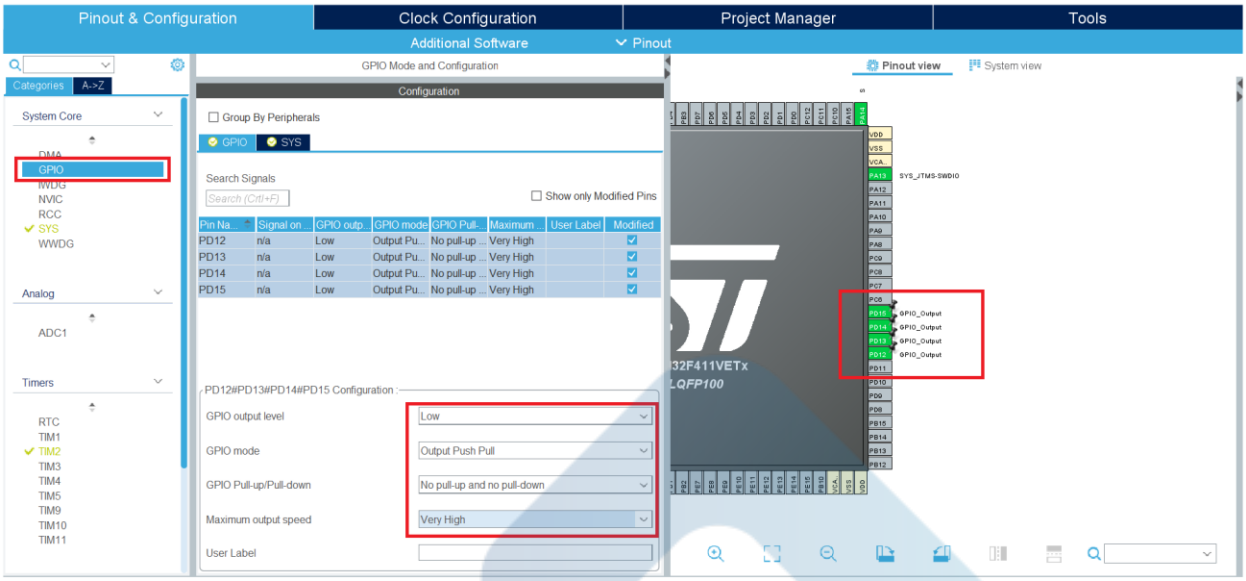
Cấu hình chip Debug bằng mode SWD



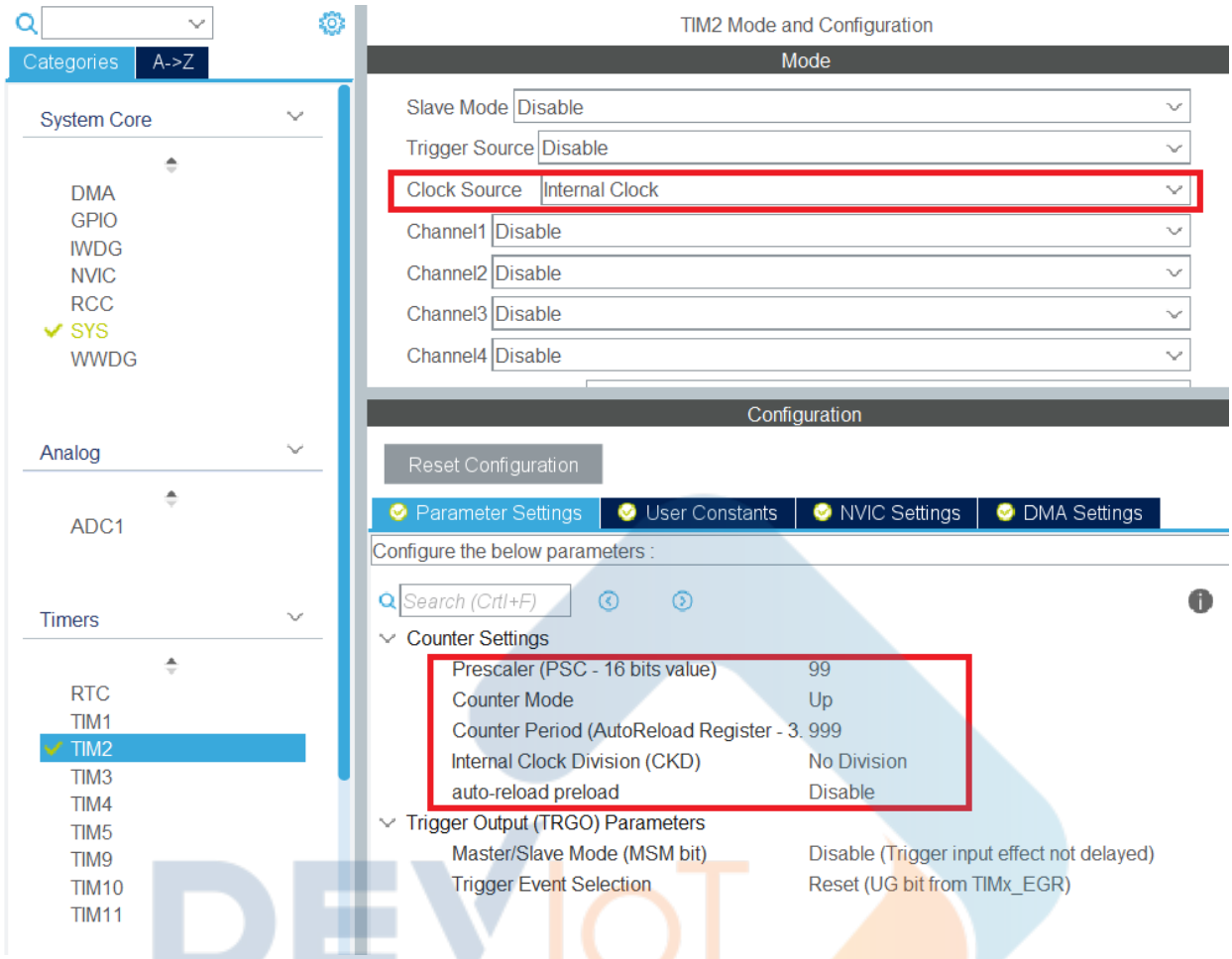
Sau đó mình cấu hình Clock cho hệ thống, ở đây mình chọn sử dụng nguồn Clock là bộ dao động RC nội được cung cấp sẵn của con Chip (trong trường hợp KIT của bạn không có hàn thạch anh ngoại nhé). Tần số hoạt động của Chip lên đến 100Mhz.



Cấu hình GPIO cho Led: Mình chọn User LED nối với chân PD12,PD13,PD14,PD15 của chip. Các LED này đã được hàn sẵn trên KIT của mình



Tiếp theo ta cấu hình cho Timer: ở đây mình sử dụng **TIM2** nhé.



Clock Source: chọn Internal Clock (100Mhz như mình cấu hình ở trên)

Counter Mode: chọn chế độ đếm lên.

Giờ mình sẽ giải thích việc tính toán Clock cho Timer và thời gian Timer sẽ xảy ra ngắt tràn nhé.

Ta có: $F_{timer} = \frac{F_{hệ\ thống}}{(Prescaler+1)}$

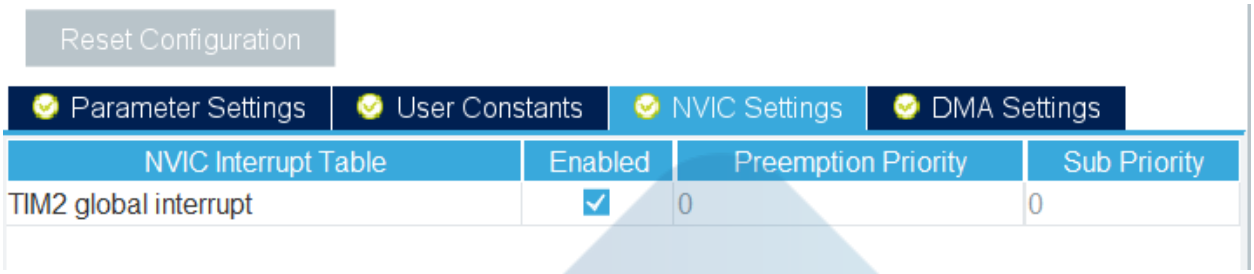
$$T_{event} = \frac{1}{F_{timer}} * (Counter\ Period + 1)$$

$$= \frac{(Prescaler + 1) * (Counter\ Period + 1)}{F_{hệ\ thống}}$$

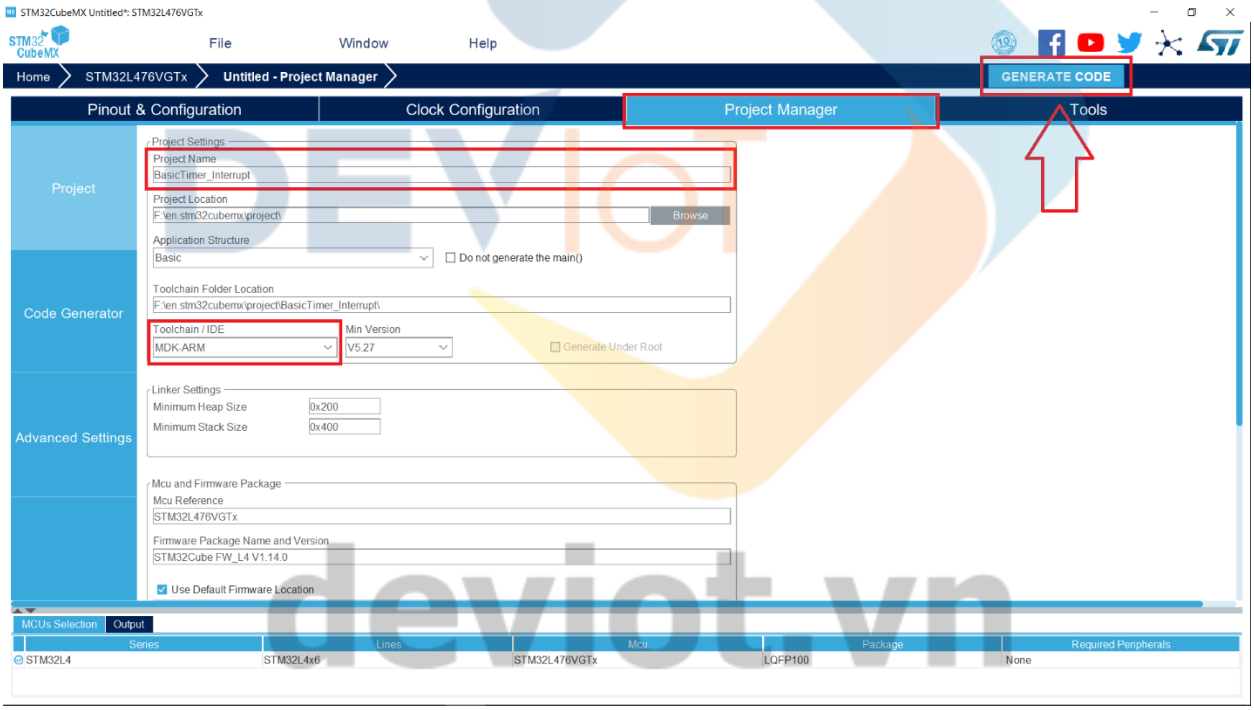
Áp dụng công thức trên, mình muốn tạo ra 1 ngắt Timer cứ 1ms xảy ra 1 lần. Nên mình chọn Prescaler = 99 và Period = 999.

$$T_{event} = \frac{(99+1)*(999+1)}{100{,}000{,}000} = 0{,}001\ (s)$$

Sau đó, các bạn chuyển qua phần NVIC Settings, bật ngắt cho TIM2.



Cuối cùng, ở **Project Manager**, đặt tên cho chương trình của bạn, chọn IDE và sinh code.



Các bạn nhớ chọn **Copy only the necessary library file** nhé.

Khi Cube Mx tạo code xong, mình xin nói về một số hàm quan trọng khi làm việc với Timer ở ví dụ này:

```c
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  HAL_TIM_Base_Start_IT(&htim2);

  while (1)
  {
  }
}
```

Hàm HAL_TIM_Base_Start_IT là hàm cho phép bắt đầu chạy TIM2 đồng thời Enable ngắt tràn cho TIM2, bất cứ khi nào CNT của TIM2 tăng quá giá trị 999 nó sẽ được reset về 0 đồng thời tạo ra 1 ngắt tràn.

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  static uint16_t cnt = 0;
  if(htim->Instance == TIM2)
  {
    if(cnt++ == 1000)
    {
      cnt = 0;
      HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15);
    }
  }
}
```

Hàm xử lý khi có ngắt TIM2, ở đây mình tạo 1 biến đếm **cnt** dạng **static** để sau khi ra khỏi hàm thì giá trị của biến **cnt** vẫn được giữ mà không bị giải phóng mất.

Do mình tạo ra ngắt TIM2 là 1ms nên mình đếm khi tạo ra 1000 ngắt TIM2 (tức **cnt** = 1000, nghĩa là đủ 1s) thì mình cho nháy các 4 LED trên KIT (Đảo trạng thái).

📌 Hotline: 0969.666.522

📌 Address: Số 101C, Xã Đàn 2

📌 Đào tạo thật, học thật, làm thật