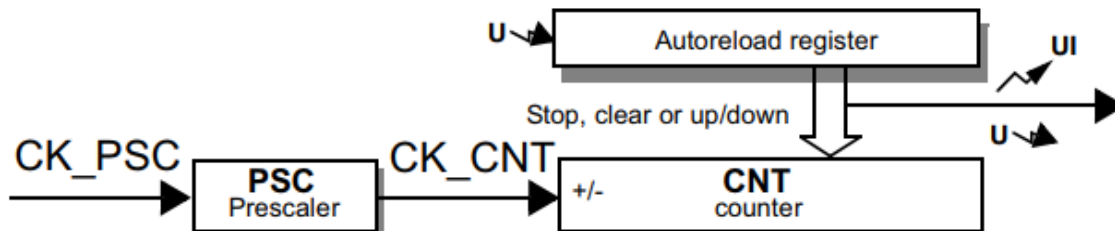


## Tìm hiểu và sử dụng timer trên STM32

Timer là một loại ngoại vi được tích hợp ở hầu hết các vi điều khiển, cung cấp cho người dùng nhiều ứng dụng như xác định chính xác một khoảng thời gian, đo – đếm xung đầu vào, điều khiển dạng sóng đầu ra, bấm xung....



**I. Time-base unit (Khối cơ sở của bộ Timer):** Thành phần chính của timer chính là bộ đếm – counter (CNT), với các ngưỡng trên được thiết lập bởi thanh ghi **Auto Reload (ARR)**. Counter có thể đếm lên lên hoặc đếm xuống. Clock đưa vào bộ đếm có thể được chia bởi một bộ chia tần – **Prescaler**.

Người dùng có thể thực hiện các lệnh đọc, ghi vào các thanh ghi CNT, ARR và PSC để cấu hình cho khối cơ sở của mỗi bộ Timer.

– *Counter Register (TIMx\_CNT)*: Khi hoạt động, thanh ghi này tăng hoặc giảm giá trị theo mỗi xung clock đầu vào. Tùy vào bộ timer mà counter này có thể là 16bit hoặc 32bit.

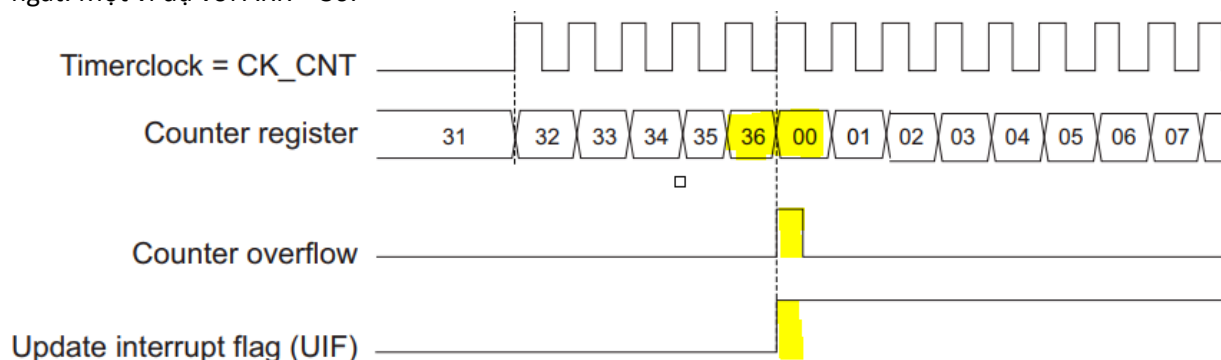
– *Prescaler Register (TIMx\_PSC)*: Giá trị của thanh ghi bộ chia tần (16bit) cho phép người dùng cấu hình chia tần số đầu vào (CK\_PSC) cho bất kì giá trị nào từ [1- 65536]. Sử dụng kết hợp bộ chia tần của timer và của RCC giúp chúng ta có thể thay đổi được thời gian của mỗi lần CNT thực hiện đếm, giúp tạo ra được những khoảng thời gian, điều chế được độ rộng xung phù hợp với nhu cầu.

– *Auto-Reload Register (TIMx\_ARR)*: Giá trị của ARR được người dùng xác định sẵn khi cài đặt bộ timer, làm cơ sở cho CNT thực hiện nạp lại giá trị đếm mỗi khi tràn (overflow khi đếm lên – CNT vượt giá trị ARR, underflow khi đếm xuống – CNT bé hơn 0). Tùy vào bộ timer mà counter này có thể là 16bit hoặc 32bit.

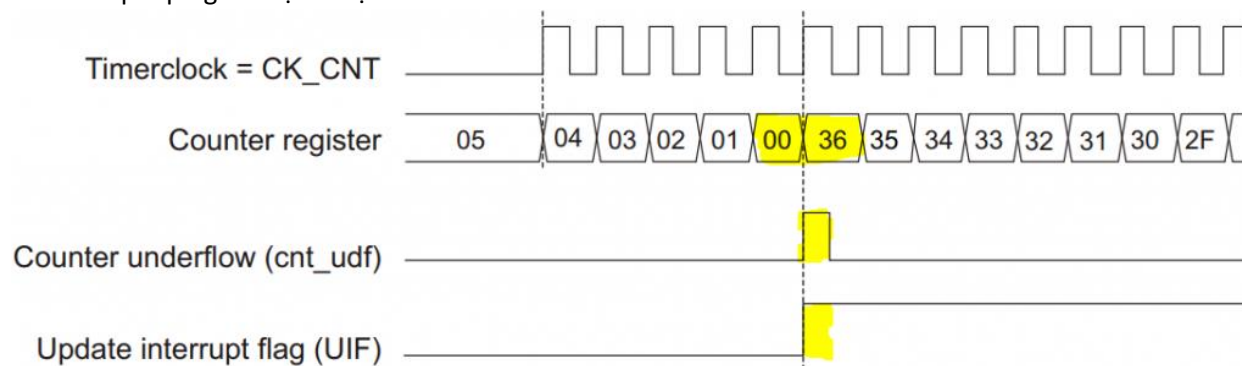
### II. Các chế độ hoạt động:

– **Các chế độ đếm:** Mỗi bộ timer đều hỗ trợ 3 chế độ đếm sau:

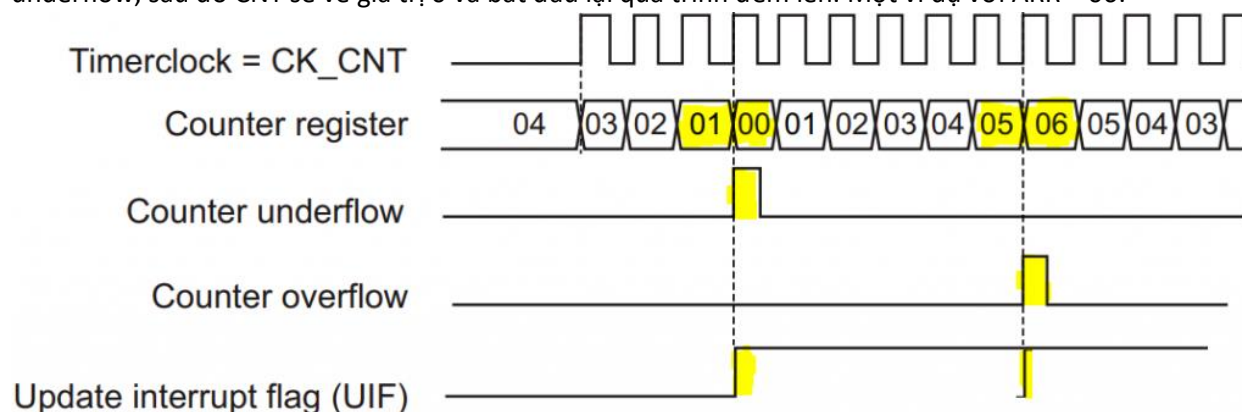
+ *Upcounting mode (chế độ đếm lên)*: Ở chế độ này, CNT đếm lên từ 0 (hoặc một giá trị nào đó được người dùng ghi vào CNT trước) đến giá trị của thanh ghi ARR, sau đó CNT bắt đầu lại từ 0. Lúc này có sự kiện tràn counter – overflow, sự kiện này có thể tạo yêu cầu ngắt nếu người dùng cấu hình cho phép ngắt. Một ví dụ với ARR = 36:



+ *Downcounting mode (chế độ đếm xuống)*: Ở chế độ này, CNT đếm xuống từ giá trị thanh ghi ARR (hoặc 1 giá trị nào đó do người dùng ghi trực tiếp vào CNT trước) đến 0, sau đó CNT bắt đầu lại từ giá trị ARR, lúc này có sự kiện tràn counter – underflow, sự kiện này có thể tạo yêu cầu ngắt nếu người dùng cấu hình cho phép ngắt. Một ví dụ với ARR = 36:



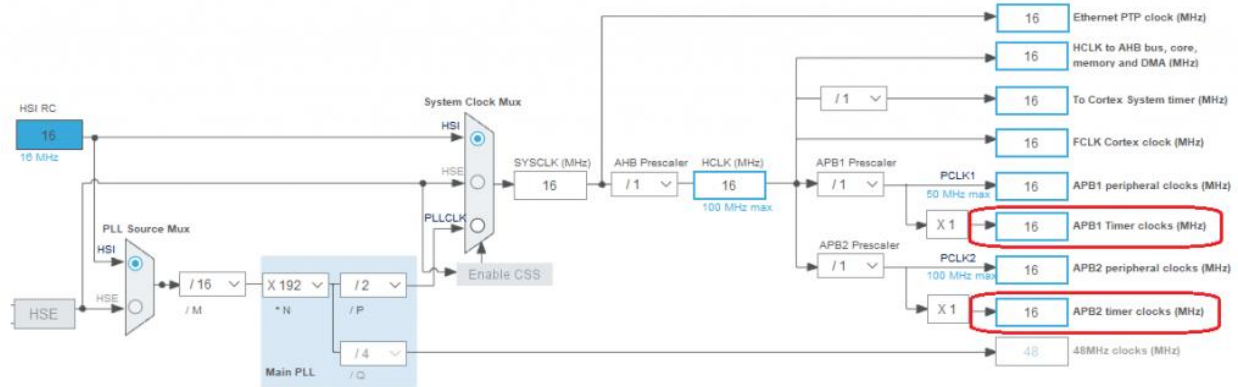
+ *Center-Aligned mode (chế độ đếm lên và xuống)*: Ở chế độ này, counter sẽ đếm lên từ 0 (hoặc một giá trị nào đó được người dùng ghi vào CNT trước) đến giá trị thanh ghi ARR – 1, lúc này xuất hiện sự kiện tràn counter – overflow, tiếp theo CNT sẽ đếm xuống từ ARR tới 1, lúc này có sự kiện tràn counter – underflow, sau đó CNT sẽ về giá trị 0 và bắt đầu lại quá trình đếm lên. Một ví dụ với ARR = 06:



### III. Lựa chọn Clock cho bộ Timer:

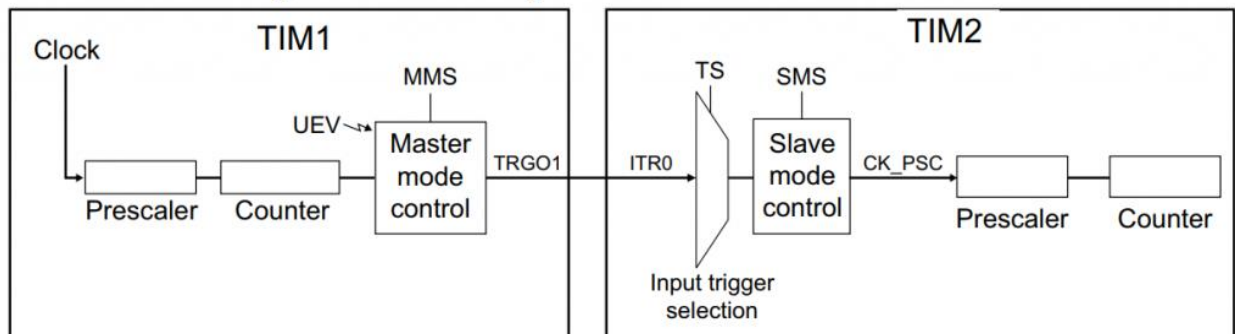
Counter clock là nguồn gốc của hoạt động tăng/ giảm giá trị CNT, clock này có thể được cấu hình lựa chọn từ các nguồn sau:

– Internal clock (CK\_INT): Chọn nguồn clock từ clock của hệ thống, có thể là từ thạch anh dao động tần số cao bên ngoài (HSE) hay bộ giao động RC tần số cao được tích hợp sẵn bên trong STM(HSI), từ đây qua các bộ chia tần của hệ thống clock để cấp cho ngoại vi timer (TIM2 đến TIM5 có clock CK\_PSC đầu vào bằng clock của bus APB1, TIM9 đến TIM11 có clock Ck\_PSC bằng clock bus APB2). CK\_PSC là clock chưa qua bộ Prescaler của khối timer.

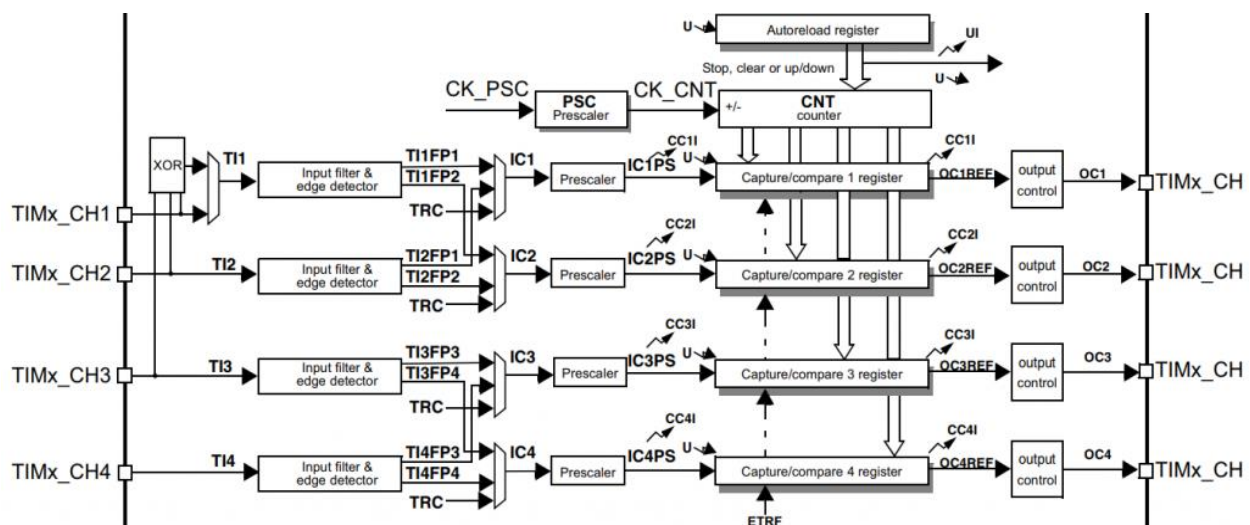


- External clock mode1: Nếu chọn mode này thì counter có thể đếm mỗi khi chân external input pin (TIx) xuất hiện sườn lên hoặc sườn xuống, người dùng cấu hình chọn sườn.
- External clock mode2: Nếu chọn mode này thì counter có thể đếm mỗi khi chân external trigger input (ETR) xuất hiện sườn lên hoặc sườn xuống. (Mode này chỉ có ở các bộ TIM2, TIM3, TIM4).
- Internal trigger inputs (ITRx): Mode này cho phép sử dụng một timer làm bộ prescaler cho một bộ timer khác.

### Using one timer as prescaler for another timer



### IV. Các kênh Caputer/Campare:

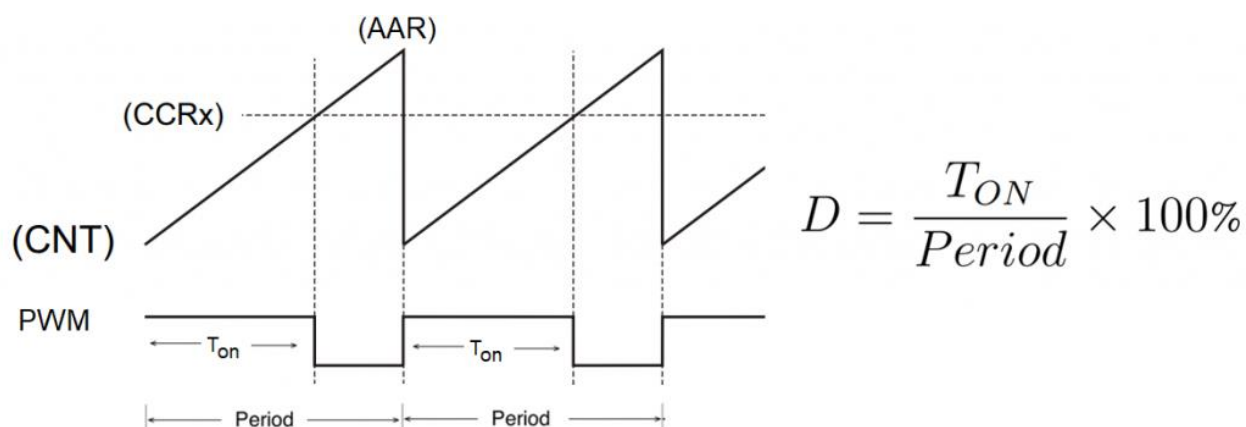


Mỗi bộ timer có 4 kênh Capture/Compare độc lập, mỗi kênh này phối hợp Time-base unit có thể tạo ra các tính năng sau:

– *Input capture*: Ở chế độ Input capture, thanh ghi CCR của kênh đầu vào tương ứng sẽ được sử dụng để lưu giá trị của CNT khi phát hiện sự thay đổi mức logic (sườn lên/ sườn xuống) như được cấu hình trước đó. Từ đó có thể biết được khoảng thời gian giữa 2 lần có sườn lên hoặc sườn xuống.

– *Output compare*: Chế độ này thường được sử dụng để điều khiển đầu ra của 1 I/O PIN khi timer đạt được một chu kỳ thời gian, cũng chính là khi giá trị CNT đếm tới giá trị bằng với giá trị thanh ghi capture/campare (đã được nạp sẵn). Người dùng có thể cài đặt I/O Pin tương ứng với các giá trị logic: mức 1, mức 0 hoặc đảo giá trị logic hiện tại. Đồng thời, cờ ngắt được bật lên và yêu cầu ngắt cũng được tạo ra nếu người dùng cấu hình cho phép ngắt.

– *PWM generation*: Tính năng điều chế độ rộng xung cho phép tạo ra xung với tần số được xác định bởi giá trị của thanh ghi ARR, và chu kỳ nhiệm vụ (Duty cycle) được xác định bởi giá trị thanh ghi CCR.



STM32F411 hỗ trợ 2 chế độ PWM như sau:

+ Mode1: Nếu sử dụng chế độ đếm lên thì ngõ ra sẽ ở mức logic 1 khi CNT < CCR và ngược lại, ở mức 0 nếu CNT > CCR. Nếu sử dụng chế độ đếm xuống, đầu ra sẽ ở mức 0 khi CNT > CCR và ngược lại, ở mức 1 khi CNT < CCR.

+ Mode2: Nếu sử dụng chế độ đếm lên thì ngõ ra sẽ ở mức logic 0 khi CNT < CCR và ngược lại, ở mức 1 nếu CNT > CCR. Nếu sử dụng chế độ đếm xuống, đầu ra sẽ ở mức 1 khi CNT > CCR và ngược lại, ở mức 0 khi CNT < CCR.

PWM được sử dụng trong rất nhiều ứng dụng như điều chỉnh điện áp đầu ra để thay đổi độ sáng đèn LED, điều khiển động cơ; điều chế bản tin bởi sóng mang, tạo âm thanh...

– *One-pulse mode output*: Chế độ này là một trường hợp cụ thể PWM, giúp tạo ra 1 xung với độ rộng xung có thể cấu hình được. Counter sẽ được dừng lại tự động khi tràn overflow/underflow.

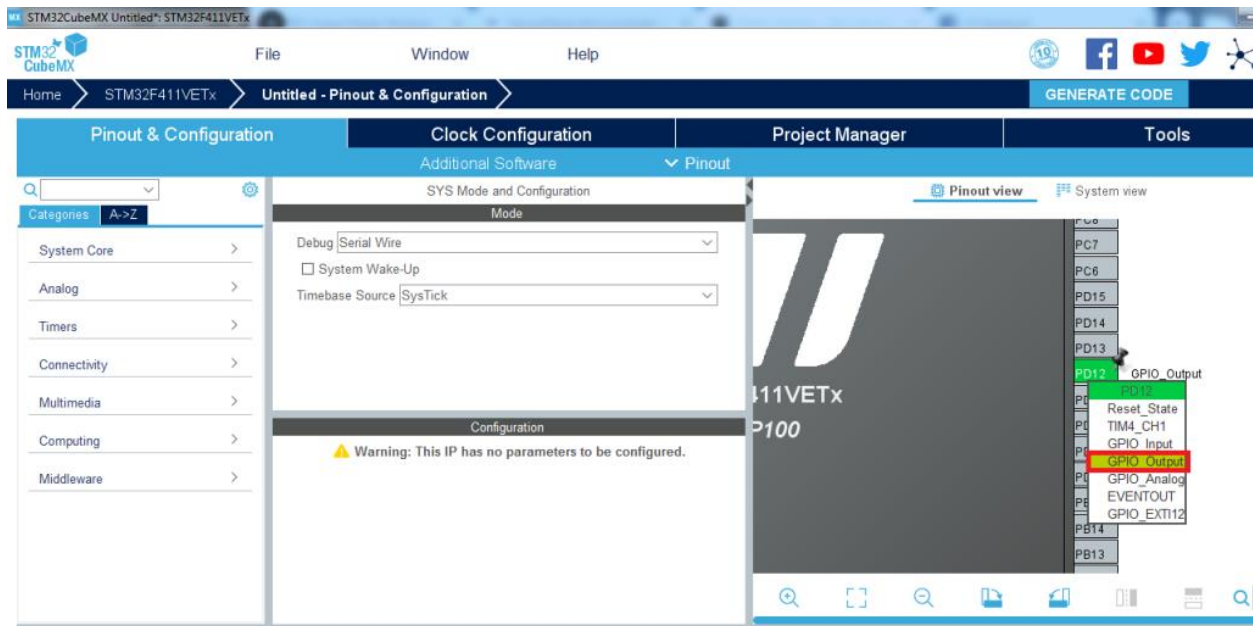
## V. Thực hành với timer

**Thực hành 1: Thay đổi trạng thái đèn LED mỗi 1 giây, sử dụng time-base unit.**

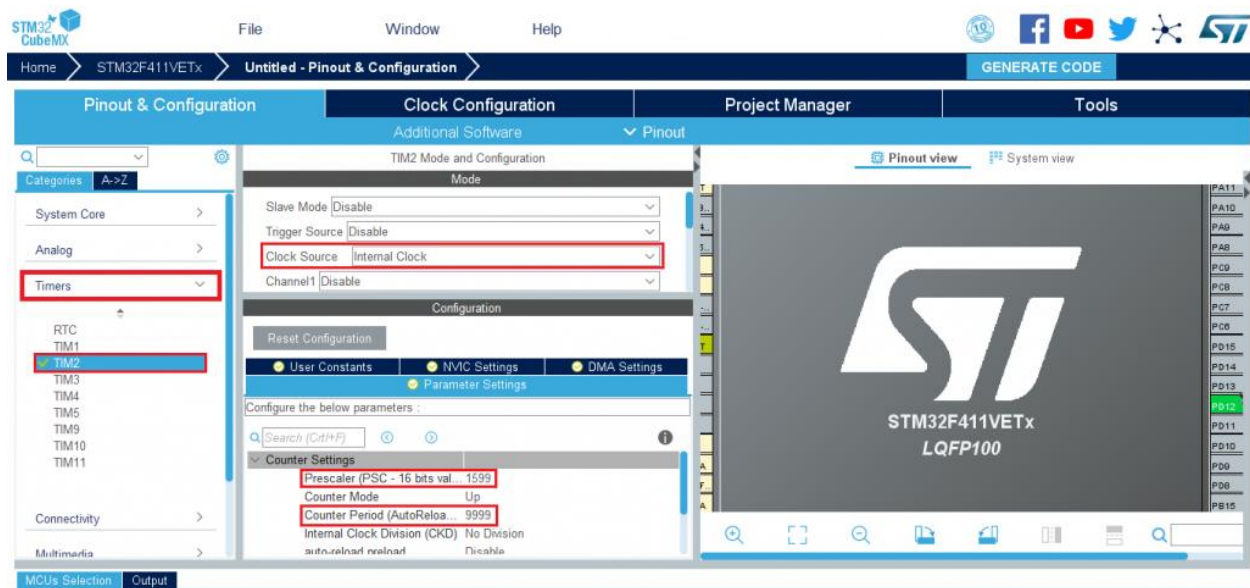
Bước 1: Kích chọn dòng STM32F411VETx và sau đó chọn “Start Project”.







Bước 3: Click vào mục Timers -> chọn TIM2



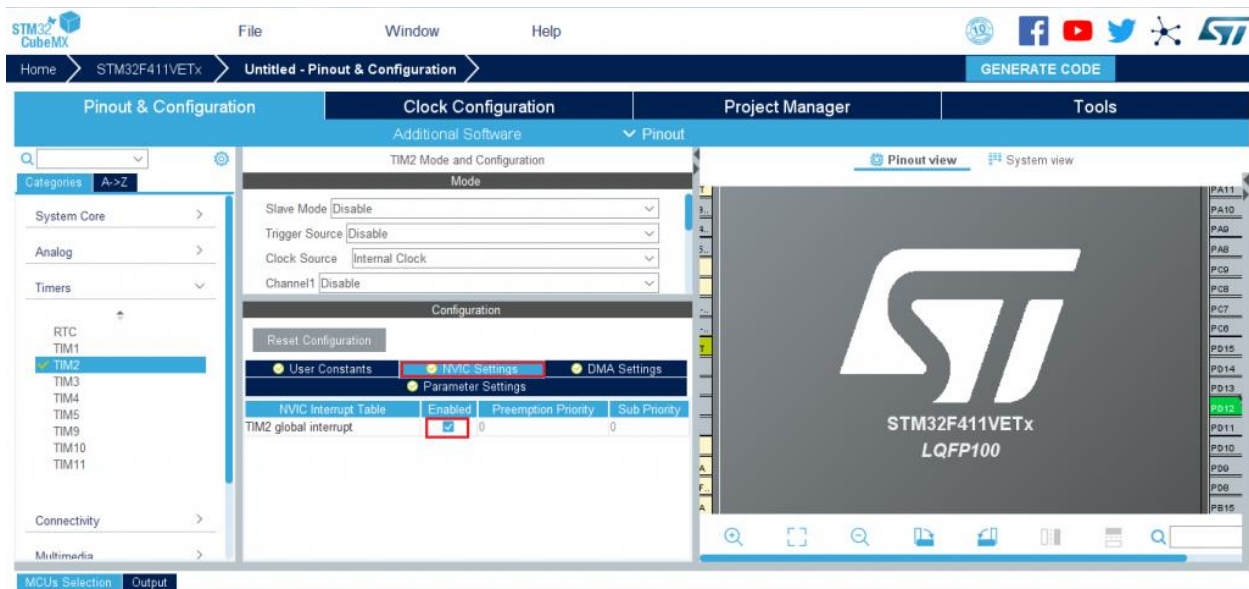
Áp dụng công  
thức

$$UpdateEvent = \frac{Timer_{clock}}{(Prescaler + 1)(Period + 1)}$$

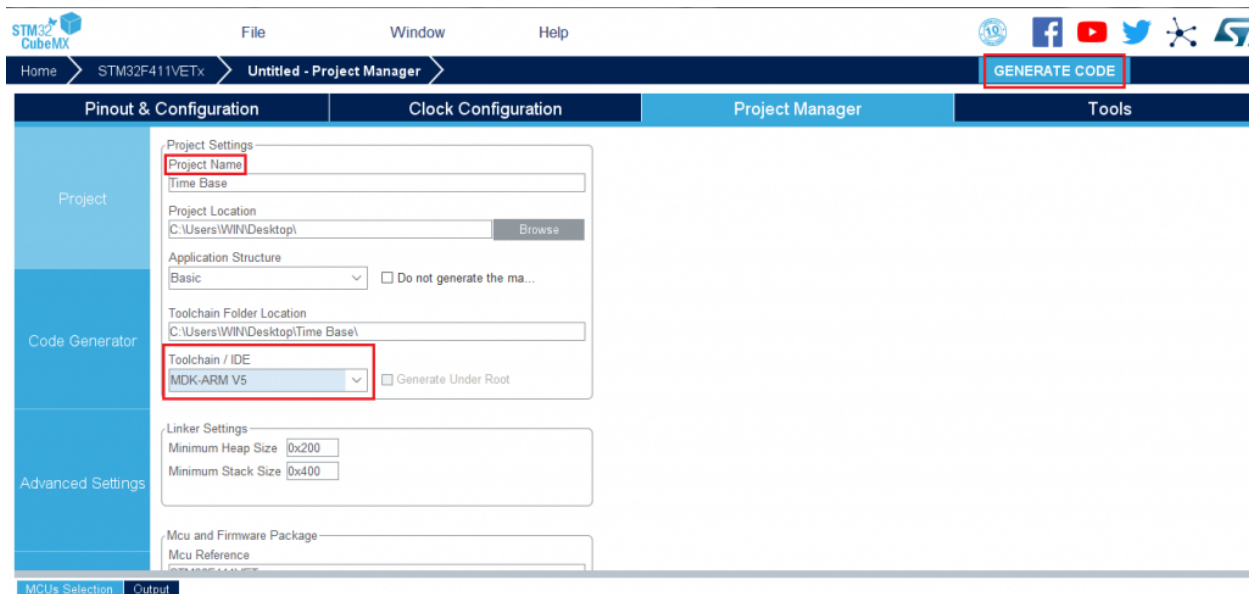
(1)

với Prescaler = 1599, Period = 9999 ta có thời gian xảy ra ngắt Timer = 1s

Bước 4: Click vào tab NVIC Setting, click chọn Enabled cho TIM2 Interrupt

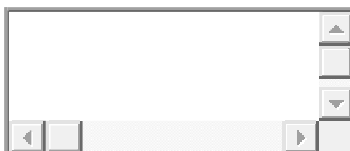


Bước 5: Setting project và sinh code



Bước 6: Lập trình bên phần mềm Keil C

Ở file **main.c** tab **Functions**, các bạn tìm đến file stm32f4xx\_hal\_tim.c. Gọi hàm HAL\_TIM\_PeriodElapsedCallback như sau:

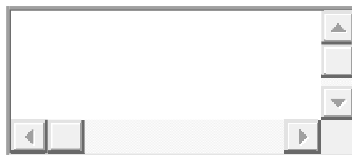


```

1  /* Private user code ----- */
2  /* USER CODE BEGIN 0 */
3  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
4  {
5      if(htim->Instance == TIM2)
6      {
7          HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
8      }
9  }
10 /* USER CODE END 0 */

```

Tiếp theo ở hàm main(), các bạn gọi lệnh **HAL\_TIM\_Base\_Start\_IT** để bắt đầu ngắt TIM2:



```

1/* USER CODE BEGIN 2 */
2    HAL_TIM_Base_Start_IT(&htim2);
3
4 /* USER CODE END 2 */

```

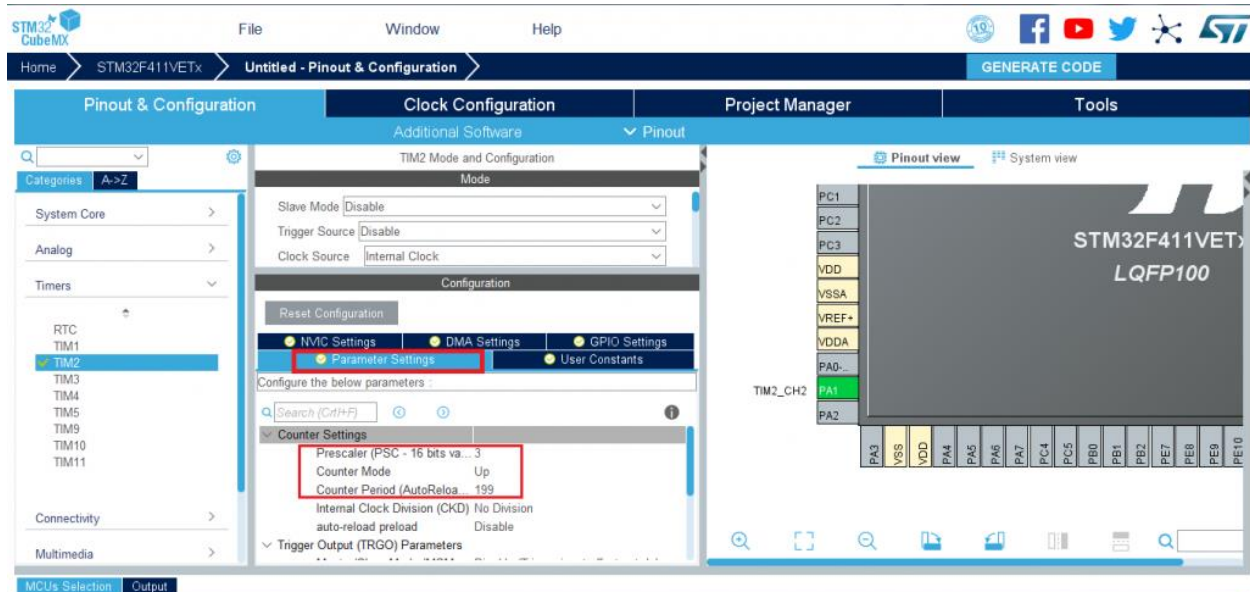
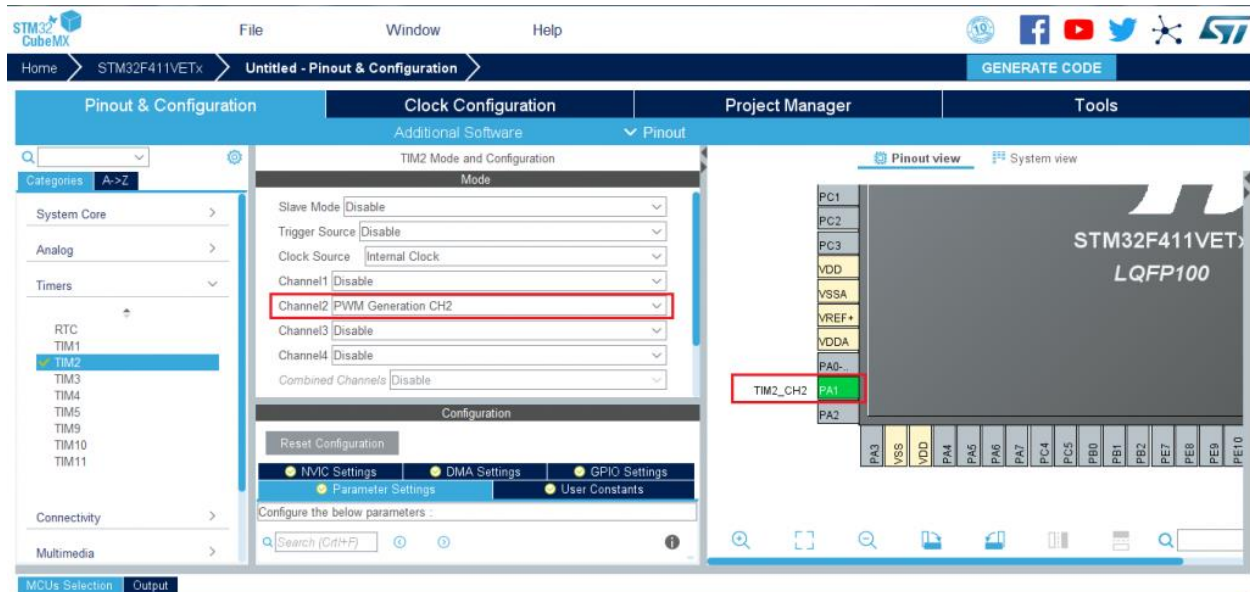
Bước 7: Build and Run. Quan sát led PD12 sẽ thay đổi trạng thái 1s 1 lần

## Thực hành 2: PWM điều khiển độ sáng.

Cấu hình System Core và GPIO vẫn thực hiện tương tự như phần hướng dẫn ở trên.

Bước 1: Cấu hình cài đặt các thông số cho tính năng PWM





Với prescaler = 3 và Counter Period = 199, áp dụng công thức (1) ở trên ta tạo ra xung với tần số PWM\_Frequency = 20000 KHz

Bước 2: Nối led ngoài vào chân PA1 để điều khiển độ sáng led

Bước 3: Cấu hình project và sinh code

Bước 4: Chuyển qua phần mềm Keil C để bắt đầu code

Ở USER CODE BEGIN PV ta khai báo biến uint8\_t **pwm** để set giá trị cho thanh ghi CCR (Capture Compare Register)



```
1/* USER CODE BEGIN PV */
```

```
2uint8_t pwm=0;
```

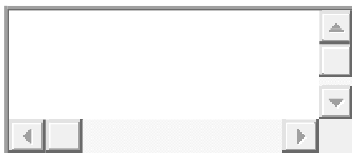
Gọi hàm **HAL\_TIM\_PWM\_Start** để bắt đầu băm xung tại chân PA1



```
1/* USER CODE BEGIN 2 */
```

```
2HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
```

Trong hàm while() ta thực hiện các bước cho Led sáng dần, đến khi Led đạt độ sáng lớn nhất thì 2 giây sau cho độ sáng của Led giảm dần. Đến khi Led tắt thì 2 giây sau lặp lại các bước như trên.



```
1 /* USER CODE BEGIN WHILE */
```

```
2 while (1)
```

```
3 {
```

```
4 /* USER CODE END WHILE */
```

```
5
```

```
6 /* USER CODE BEGIN 3 */
```

```
7     for(pwm =0;pwm<200;pwm=pwm+10)
```

```
8     {
```

```
9         __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_2,pwm);
```

```
10        HAL_Delay(100);
```

```
11    }
```

```
12    HAL_Delay(2000);
```

```
13    for(pwm =200;pwm>0;pwm=pwm-10)
```

```
14         {
15             __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_2,pwm);
16             HAL_Delay(100);
17         }
18         HAL_Delay(2000);
19 }
```

Bước 5: Build project và nhấn Load để chạy chương trình.

## Timer là gì?

**Timer** dịch đơn giản là người ghi giờ, bộ hẹn giờ.

Ví dụ khi bạn cần thức dậy sau 8 tiếng ngủ bạn sử dụng một bộ hẹn giờ, đếm ngược hoặc đếm xuôi đủ 8 tiếng, chuông sẽ kêu và bạn thức dậy.

Hoặc khi chạy 100m người trọng tài chính là Timer để đo thời gian bạn chạy hết 100m đó.

Trong lập trình Timer là một khối độc lập, có tác dụng tạo ra các sự kiện hoặc ngắt để kích hoạt các ngoại vi khác hoạt động, hoặc đo thời gian hoạt động của 1 giá trị đầu vào nào đó.

Trong STM32F103C8 có 4 bộ Timer trong đó:

- Timer 1: Là bộ Advanced – control Timer hay là bộ Timer điều khiển nâng cao, có nhiều chức năng nhất
- 3 bộ Timer chung là Timer 2,3,4 Có chức năng tương tự nhau và độc lập với nhau

Trong bài này chúng ta sẽ làm việc với Timer 2, các chức năng đều tương tự với các Timer khác

## Các chức năng chính của Timer STM32

Các chức năng chính của Timer STM32 bao gồm:

- Thanh ghi 16bit đếm lên, xuống, lên/xuống tự nạp lại
- 16 bit bộ chia tần số để chia tần số từ APB(giá trị dao động từ 1 – 65536)
- 4 Kênh độc lập mỗi Timer cho các chức năng:
  - + Input Capture
  - + Output Compare
  - + One Pulse

– Đồng bộ hóa với các mạch tạo tín hiệu bên ngoài để kết hợp nhiều bộ Timer với nhau

– Ngắt/DMA được sinh ra khi có các sự kiện:

+ Cập nhật: tràn Counter, khởi tạo Counter (bởi phần mềm hoặc kích hoạt internal/external trigger)

+ Sự kiện kích hoạt (Bắt đầu đếm, dừng đếm, khởi tạo bộ đếm hoặc đếm bởi internal/external trigger)

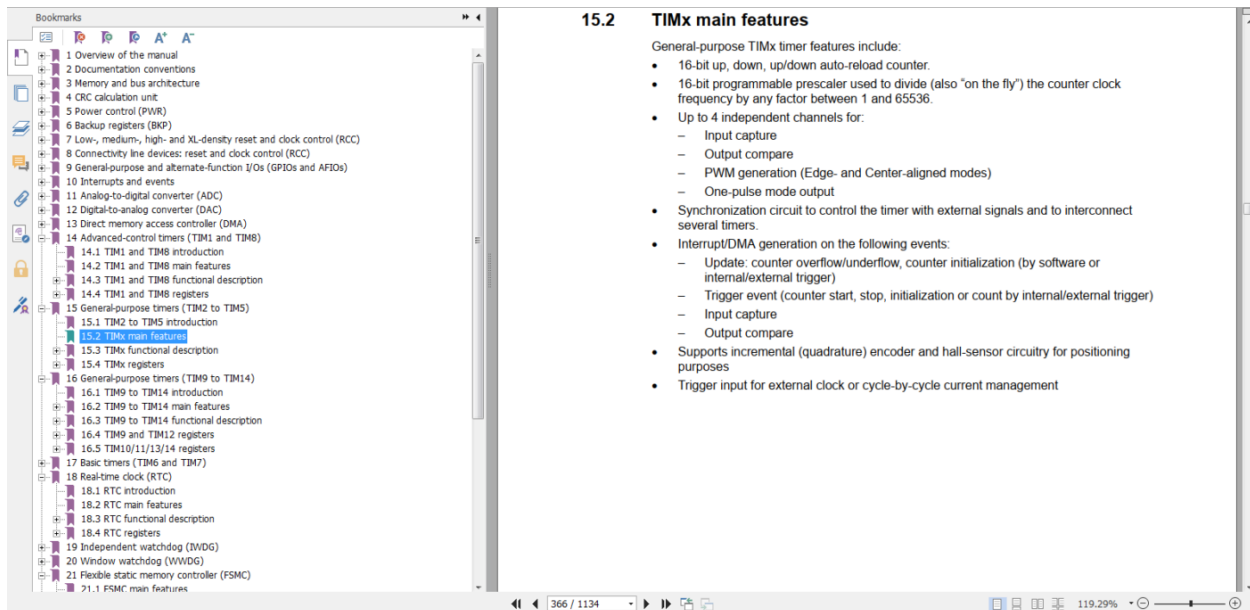
+ Input Capture: Bắt xung đầu vào

+ Output Compare: So sánh xung đầu ra

– Hỗ trợ điều khiển Encoder và Hall-sensor

– Đầu vào kích hoạt cho đồng hồ bên ngoài hoặc quản lý theo chu kỳ

Chi tiết các bạn tham khảo mục 15.2 trong Reference Manual

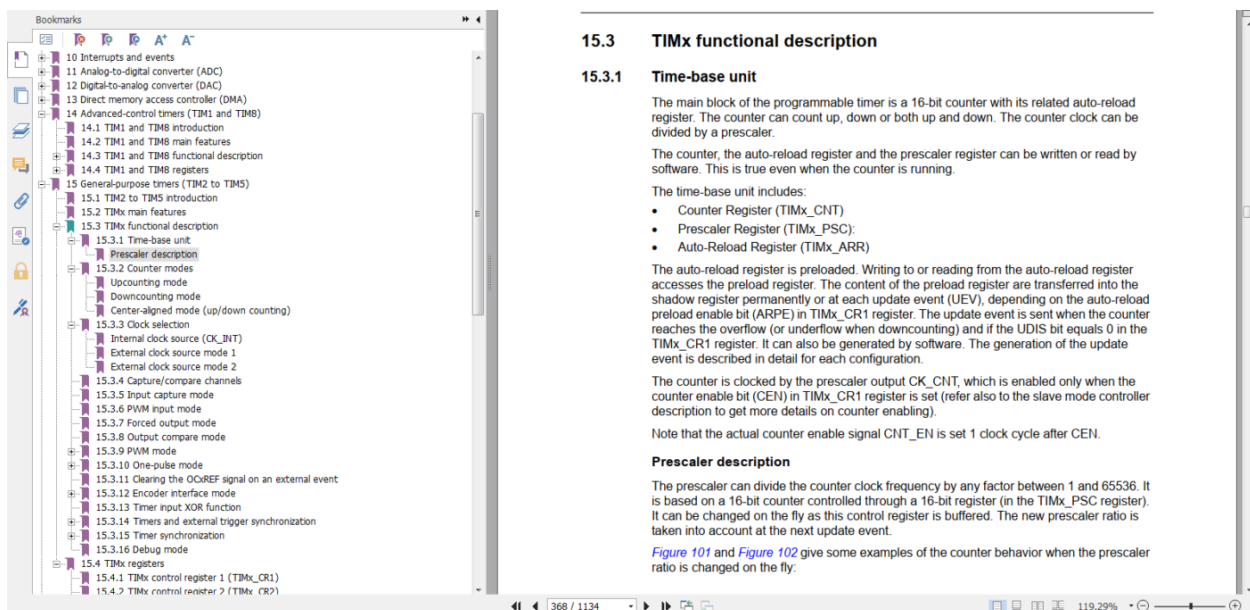


Trong bài này chúng ta sẽ cấu hình Timer 2 chế độ Time-base unit.

Mô tả chức năng này như sau:

- Chế độ Timer-Base unit là chế độ chính của Timer bao gồm các chế độ đếm: lên, xuống hoặc cả lên lên và xuống
- Xung Clock được chia bởi bộ chia tần Prescaler để lấy thời gian thích hợp đếm 1 lần
- Các thanh ghi quản lý bao gồm:
  - + Counter Register (TIMx\_CNT): lưu giá trị đếm
  - + Prescaler Register (TIMx\_PSC): lưu giá trị chia từ tần số cơ sở cấp cho Timer để tạo ra tần số thích hợp
  - + Auto-Reload Register (TIMx\_ARR): lưu giá trị đích đếm lên hoặc đếm xuống
- Thanh ghi Auto-Reload sẽ được nạp trước khi Timer hoạt động, và có thể nạp trong khi Timer hoạt động, trước khi một sự kiện cập nhật xảy ra (UEV).
- Chế độ đếm sẽ được hoạt động khi Bit CEN của thanh ghi TIMx\_CR1 được bật

### Chi tiết các bạn tham khảo mục 15.3.1



The screenshot displays a technical document with a table of contents on the left and a detailed description of the TIMx functional description on the right. The table of contents lists various topics from 10 to 15.4.7, with 15.3.1 Time-base unit being the current focus. The main text on the right describes the Time-base unit as a 16-bit counter with an auto-reload register, prescaler, and auto-reload register. It details the counter's operation, including upcounting, downcounting, and center-aligned modes. It also describes the prescaler's function and the auto-reload register's role in generating update events. The document includes a note about the CNT\_EN signal and a section on the prescaler description.

**15.3 TIMx functional description**

**15.3.1 Time-base unit**

The main block of the programmable timer is a 16-bit counter with its related auto-reload register. The counter can count up, down or both up and down. The counter clock can be divided by a prescaler.

The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.

The time-base unit includes:

- Counter Register (TIMx\_CNT)
- Prescaler Register (TIMx\_PSC)
- Auto-Reload Register (TIMx\_ARR)

The auto-reload register is preloaded. Writing to or reading from the auto-reload register accesses the preload register. The content of the preload register are transferred into the shadow register permanently or at each update event (UEV), depending on the auto-reload preload enable bit (ARPE) in TIMx\_CR1 register. The update event is sent when the counter reaches the overflow (or underflow when downcounting) and if the UDIS bit equals 0 in the TIMx\_CR1 register. It can also be generated by software. The generation of the update event is described in detail for each configuration.

The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

Note that the actual counter enable signal CNT\_EN is set 1 clock cycle after CEN.

**Prescaler description**

The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It is based on a 16-bit counter controlled through a 16-bit register (in the TIMx\_PSC register). It can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.

Figure 101 and Figure 102 give some examples of the counter behavior when the prescaler ratio is changed on the fly.

Nhìn vào mô tả này chúng ta có thể viết ra các bước để Timer hoạt động như sau:



1. Khởi tạo bộ Timer 1 với xung Clock thích hợp(Tất cả các ngoại vi đều phải có bước này)
2. Ghi giá trị cho bộ chia tần Prescaler
3. Ghi giá trị cho thanh ghi Auto-reload
4. Bật Timer cho hoạt động
5. Xử lý các sự kiện xảy ra như tràn ....

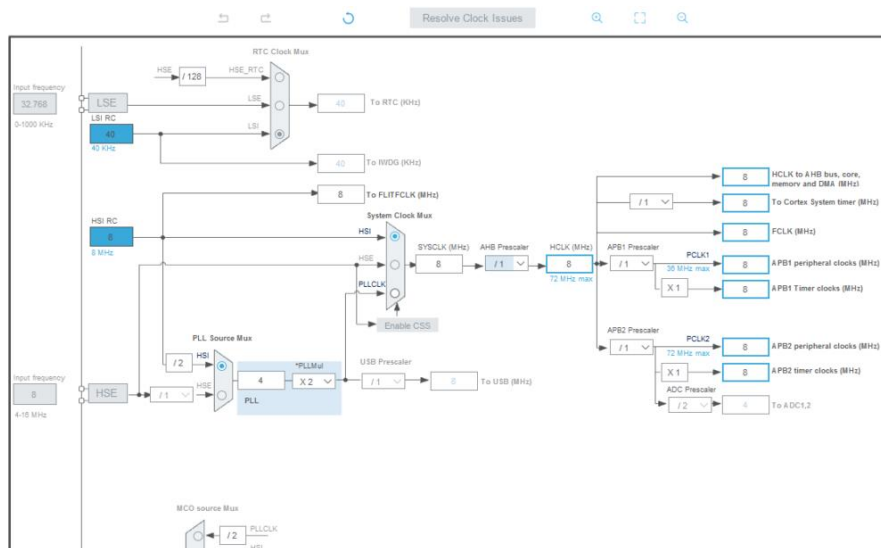
Các bước cơ bản là vậy, sau đây mình sẽ hướng dẫn các bạn cấu hình trên CubeMX

## Cấu hình Timer STM32 chế độ Time Base

Đề bài của chúng ta như sau: Sử dụng TimerSTM32 cụ thể là Timer 2 nhấp nháy led mỗi 500ms một lần.

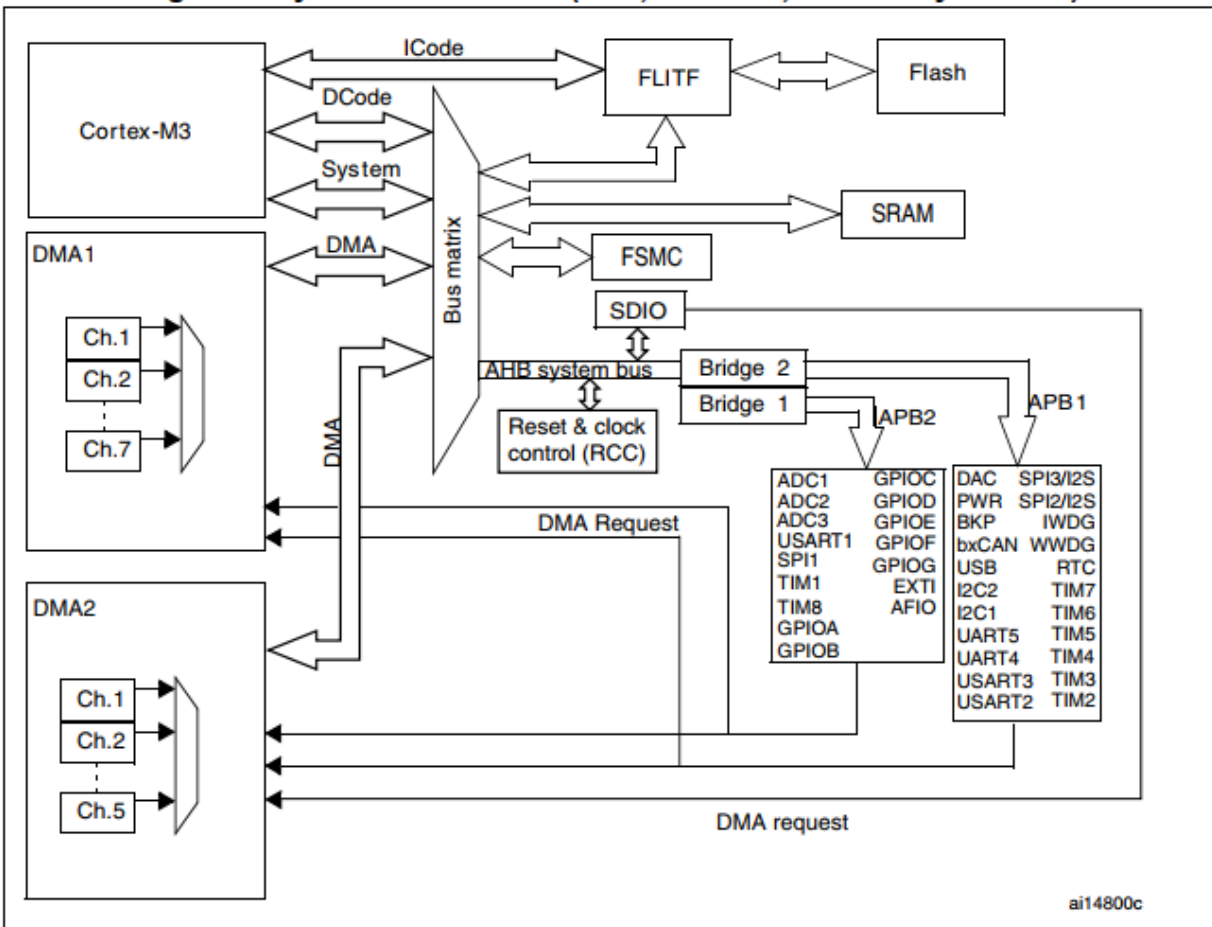
Các bạn mở phần mềm, chọn chip STM32F103C8, cấu hình SYS debug: Serial Wire.

Chuyển qua Tab Clock Configuration , chúng ta chọn Thạch Anh nội 8Mhz. Vì xung Clock của Timer 2 sẽ được cấp bởi bộ APB1 thế nên xung Internal Clock có tần số là 8Mhz



Chi tiết về cách cấu hình Clock các bạn xem Bảng 1 Mục 3.1

**Figure 1. System architecture (low-, medium-, XL-density devices)**



Trong Tab Timer các bạn chọn Timer 2. Clock Source chọn Internal Clock.

Trong bảng Parameter Settings:

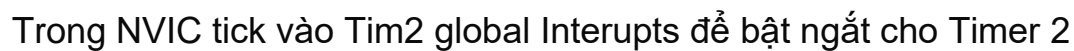
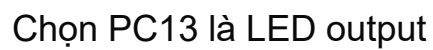
Chọn Prescale là 8000 sẽ đếm mỗi 1ms vì  $8\text{MHz}/8000 = 1\text{kHz} \Rightarrow T = 1\text{ms}$

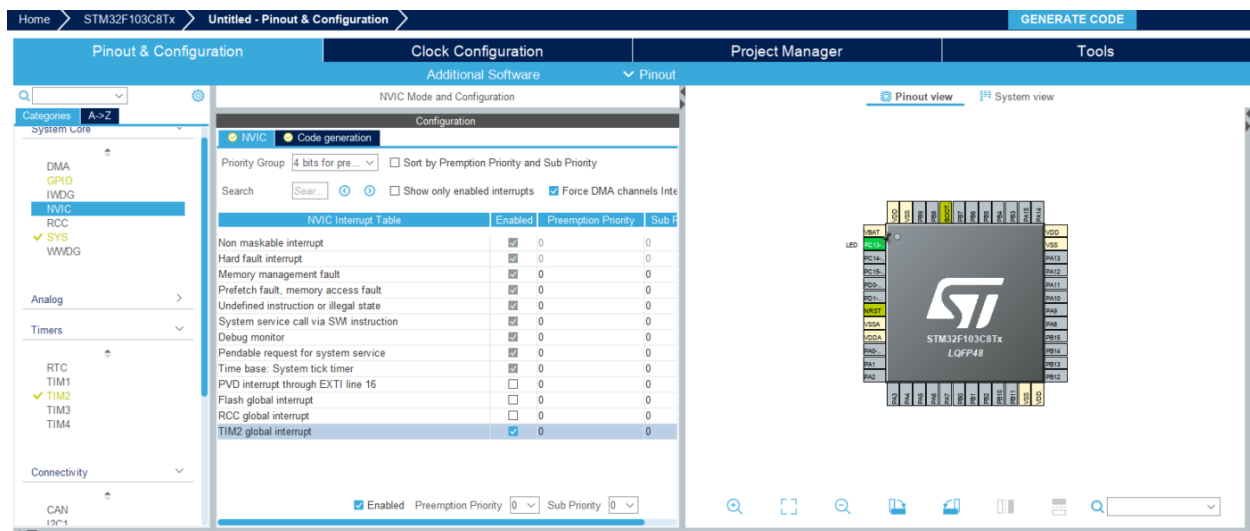
Counter Mode : Up down đều được

Counter Period: 499 , đếm từ 0 đến 499 là 500 lần 1ms ta sẽ được 500ms

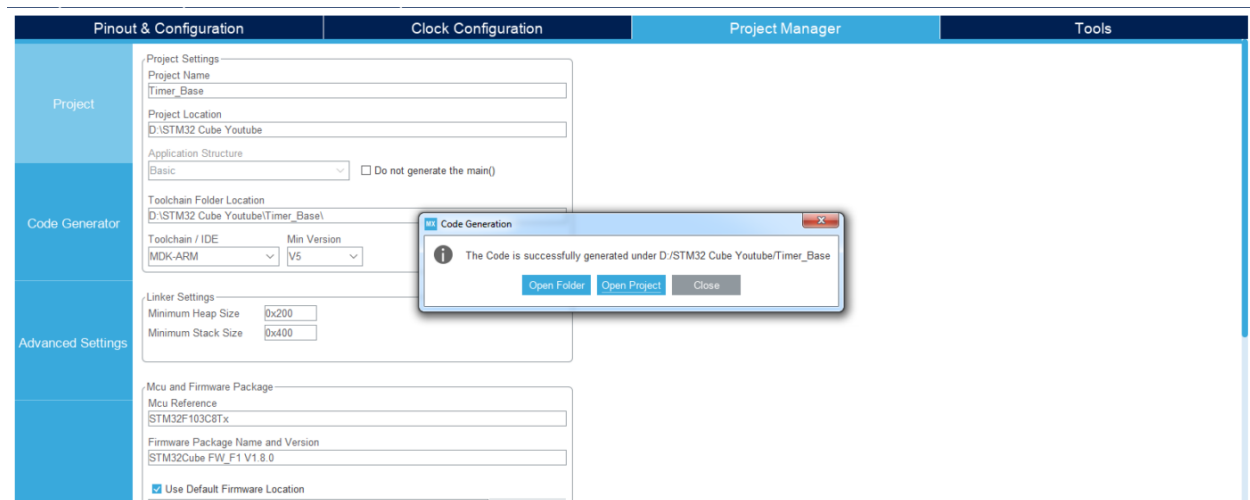
Auto-preload: Enable

Các thông số còn lại giữ nguyên





Sau đó đặt tên Project – chọn tool MDK-ARM V5 và ấn Gen Code



## Lập trình Timer STM32 chớp tắt led

Mở Project – Nhấn Build (F7)

Trong phần stm32f1xx\_it.c đã có hàm thực thi ngắt Timer2, các bạn nhấn chuột phải chọn Go to Define hàm HAL\_TIM\_IRQHandler(&htim2);



```

58  /* Private user code -----*/
59  /* USER CODE BEGIN 0 */
60  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
61  {
62      /* Prevent unused argument(s) compilation warning */
63      UNUSED(htim);
64      HAL_GPIO_TogglePin(LED_GPIO_Port,LED_Pin);
65      /* NOTE : This function should not be modified, when the callback is needed,
66               the HAL_TIM_PeriodElapsedCallback could be implemented in the user file
67      */
68  }
69  /* USER CODE END 0 */
70
71  /**
72   * @brief The application entry point.
73   * @retval int
74   */
75  int main(void)
76  {
77      /* USER CODE BEGIN 1 */

```

Trước while (1) ta khởi chạy Timer với ngắt bằng câu lệnh sau

```

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim2);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

```

Sau đó Build và Nạp chương trình và xem kết quả



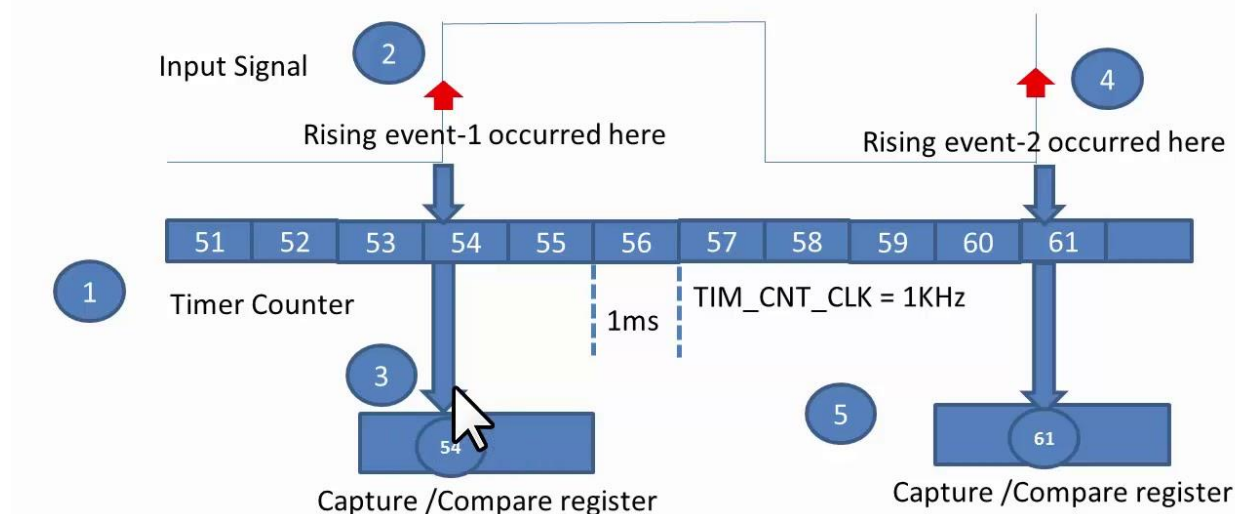
## STM32 Timer chế độ Input Capture

Input Capture là chế độ bắt sườn đầu vào, thường ứng dụng trong các hoạt động đo tần số hoặc độ rộng xung.

Input Capture có 2 chế độ là bắt xung sườn lên(Rising) hoặc sườn xuống(Falling)

Nguyên lý hoạt động như sau:

- + Sau khi khởi động Timer, mỗi khi có xung sườn lên (hoặc sườn xuống) tại đầu vào Channelx của Timer. Thanh ghi TIMx\_CCRx (x là Timer số 1,2,3,4...) sẽ được nạp giá trị của thanh ghi Counter TIMx\_CNT
- + Bit CC1IF được đặt lên 1 (Cờ ngắt), Bit CC1OF sẽ được set lên 1 nếu 2 lần sườn liên tiếp sườn CC1IF được đặt lên 1 mà không xóa. Nghĩa là mỗi khi có sự kiện xung sườn lên sẽ có ngắt xảy ra, lập trình viên phải đọc giá trị lưu vào CCR1 sau đó xóa cờ CC1IF, nếu không cờ CC1OF sẽ được bật lên báo tràn và dữ liệu sẽ không được ghi vào nữa.
- + Một sự kiện ngắt được sinh ra nếu Bit CC1IE được đặt lên 1
- + Một sự kiện DMA sinh ra nếu Bit CC1DE được đặt lên 1



Các bước khởi tạo IC như sau:

B1: Chọn nguồn xung đếm (internal, external, timer)

B2: Ghi dữ liệu vào Thanh ghi ARR và thanh ghi PSC

B4: Chọn bật Ngắt hoặc DMA bằng bit CCxIE hoặc CCxDE trong thanh ghi CR1

B5: Khởi chạy bộ đếm bằng Bit CEN của thanh ghi CR1

B6: Trong hàm ngắt đọc dữ liệu từ thanh ghi CCRx, xóa thanh ghi CNT về 0 khi có ngắt thứ 2 sinh ra, giá trị của thanh ghi CCRx là khoảng thời gian giữa 2 lần ngắt.

Chi tiết các bạn tham khảo mục 15.3.4 trong reference Manual

## STM32 Timer chế độ output compare

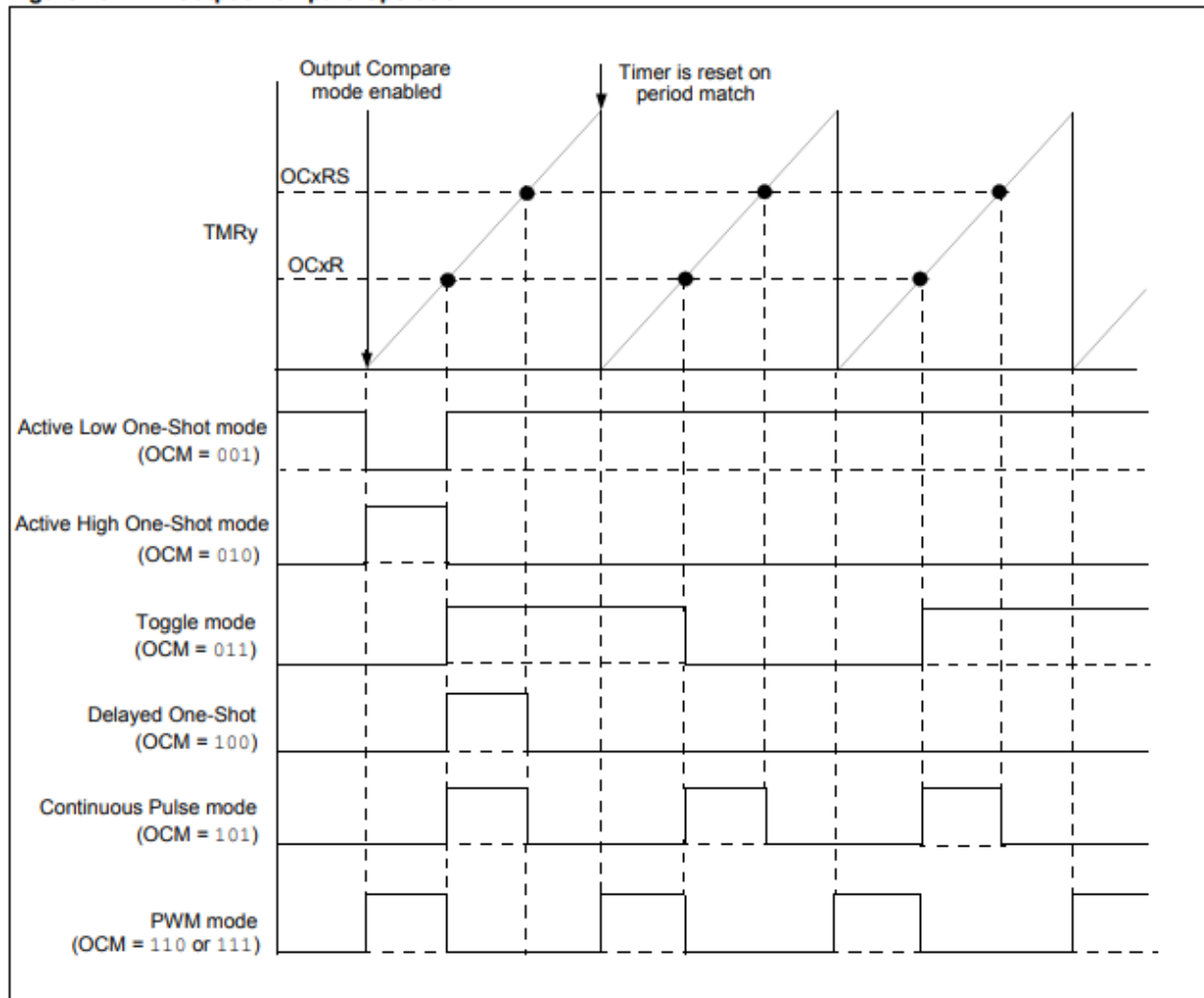
Chế độ output Compare là chế độ phát xung ra khi thời điểm được cài đặt, bằng thời điểm mà bộ đếm đang đếm lên/xuống.

Nguyên lý hoạt động: Khi bộ đếm, đếm đến giá trị được cài đặt trong thanh ghi CCR Timer sẽ:

- + Tạo ra một hoạt động tùy vào chế độ được chọn của bộ OC
- + Set cờ ngắt CCxIF lên 1
- + Tạo ra một ngắt hoặc DMA nếu bit CCxIE hoặc CCxDE được bật

Các chế độ làm việc của OC

**Figure 13-2: Output Compare Operation**



Các bước khởi tạo OC như sau:

B1: Chọn xung Clock cho Timer

B2: Ghi dữ liệu vào thanh ghi ARR và thanh ghi CCR

B3: Set bit CCxIE hoặc CCxDE nếu bật ngắt hoặc DMA

B4: Chọn chế độ OC bằng 4 bit OCxM trong thanh ghi CCMR1

B5: Chạy bộ đếm bằng bit CEN của thanh ghi CR1

Chi tiết các bạn tham khảo mục 15.3.8 trong reference Manual

## Cấu hình STM32 Timer trong CubeMX

Trong bài này, chúng ta sẽ cấu hình Channel 1 của Timer 2 là Input Capture, Channel 1 của Timer 3 là Output Compare.

Mở phần mềm lên, chọn Khởi tạo trên MCU STM32F103C8, nhấn Start project

Trong Sys Debug chọn Serial Wire, nếu các bạn chưa biết các bước cơ bản này vui lòng đọc lại **Bài 3** nhé

Trong phần Timer chọn Timer 2 :

Clock Source – Internal Clock

Channel 1: Input Capture direct mode

Trong Parameters Settings chọn Prescaler là 8000 => mỗi lần đếm là 1ms (chi tiết trong bài 5)

Counter Period: 999 Chu kỳ tràn là 1000 ms. Vì mình sẽ đọc các xung có  $F > 1\text{Hz}$

Auto Reload: Enable

Polarity Selection: Rising (Chọn bắt xung lên)

TIM2 Mode and Configuration

Mode

Slave Mode Disable ▼  
 Trigger Source Disable ▼  
 Clock Source Internal Clock ▼  
 Channel1 Input Capture direct mode ▼  
 Channel2 Disable ▼  
 Channel3 Disable ▼  
 Channel4 Disable ▼

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

🔍

⏪ ⏩
i

▼ Counter Settings

Prescaler (PSC - 16 bits value) 8000  
 Counter Mode Up  

Counter Period (AutoReload Registe... 999

 Internal Clock Division (CKD) No Division  
 auto-reload preload Enable

▼ Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)  
 Trigger Event Selection Reset (UG bit from TIMx\_EGR)

▼ Input Capture Channel 1

Polarity Selection Rising Edge  
 IC Selection Direct  
 Prescaler Division Ratio No division  
 Input Filter (4 bits value) 0

Bài

Timer 3 Các bạn cấu hình như sau:

Internal Clock

Channel 1: Output Compare CH1

Prescaler 8000 => mỗi lần đếm là 1ms

Counter Period: 49 chu kì tràn là 50ms

Auto Reload: Enable

Mode: Toggle on Match: 24 (đổi trạng thái CH1 khi Counter đếm lên 24 => 25ms)

TIM3 Mode and Configuration

Mode

Slave Mode

Disable

Trigger Source

Disable

☒ Internal Clock

Channel1

Output Compare CH1

Channel2

Disable

Channel3

Disable

Channel4

Disable

Configuration

Reset Configuration

☒ NVIC Settings

☒ DMA Settings

☒ GPIO Settings

☒ Parameter Settings

☒ User Constants

Configure the below parameters :

Counter Settings

Prescaler (PSC - 16 bits value)

8000

Counter Mode

Up

Counter Period (AutoReload Register value)

49

Internal Clock Division (CKD)

No Division

auto-reload preload

Enable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)

Disable (Trigger input effect not delayed)

Trigger Event Selection

Reset (UG bit from TIMx\_EGR)

Output Compare Channel 1

Mode

Toggle on match

Pulse (16 bits value)

24

Output compare preload

Enable

Bài



## NVIC Setting tick vào bật Ngắt cho Timer 2

The screenshot shows the STM32CubeIDE interface for configuring the NVIC (Nested Vectored Interrupt Controller). The left sidebar lists various system components, with 'NVIC' selected under the 'System Core' category. The main panel displays the 'NVIC Mode and Configuration' window, which is currently in the 'Configuration' tab. The 'NVIC' and 'Code generation' checkboxes are both checked. The 'Priority Group' is set to '4 bits for pre...'. The 'Search' field is empty. The 'Show only enabled interrupts' checkbox is unchecked, and the 'Force DMA channels Interrupts' checkbox is checked. The 'NVIC Interrupt Table' is displayed, showing a list of interrupts with their corresponding 'Enabled', 'Preemption Priority', and 'Sub Priority' values. The 'TIM2 global interrupt' is checked in the 'Enabled' column. The 'Sub Priority' is set to 0.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Tin			
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
TIM2 global interrupt	<input checked="" type="checkbox"/>	0	0
TIM3 global interrupt	<input type="checkbox"/>	0	0

Đặt tên, Chọn Toolchain và Gen code

**Project Settings**

Project Name  
Bai6\_IC\_OC\_Tim2

Project Location  
D:\STM32 Cube Youtube

Application Structure  
Basic ☐ Do not generate the main()

Toolchain Folder Location  
D:\STM32 Cube Youtube\Bai6\_IC\_OC\_Tim2\

Toolchain / IDE  
MDK-ARM

Min Version  
V5 ☐ Generate Under Root

**Linker Settings**

Minimum Heap Size  
0x200

Minimum Stack Size  
0x400

**Mcu and Firmware Package**

Mcu Reference  
STM32F103C8Tx

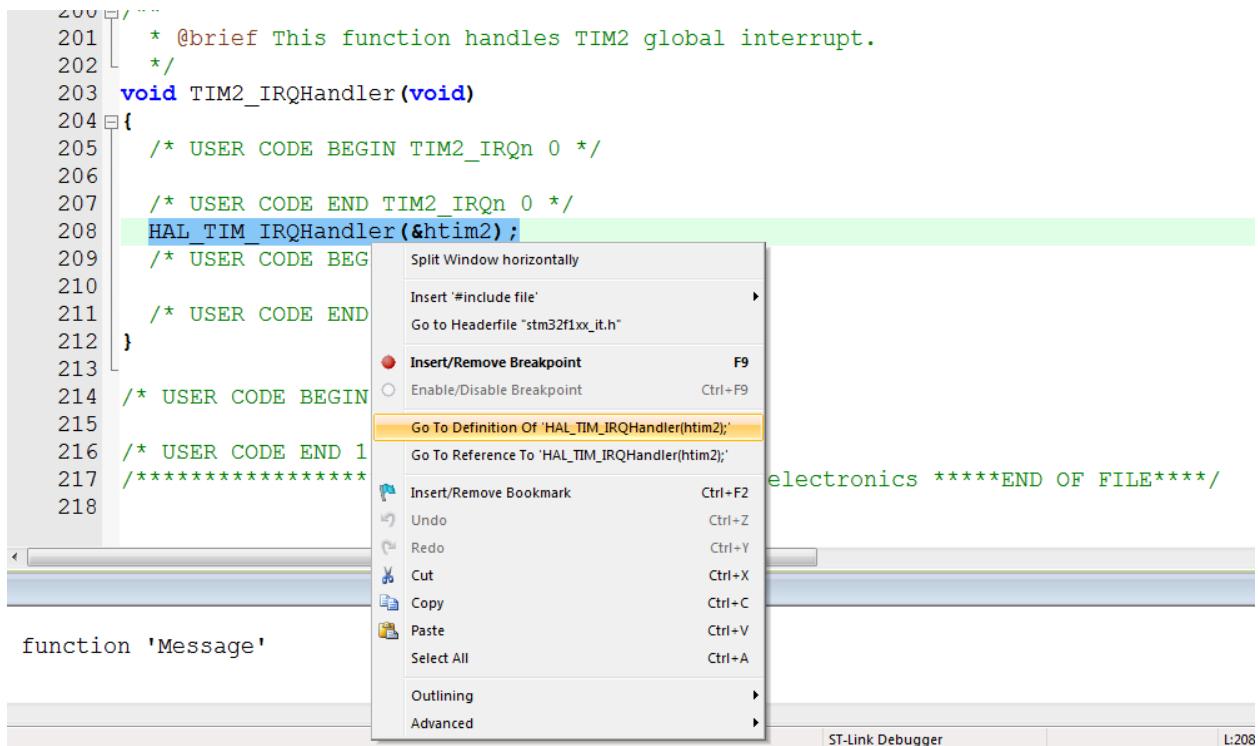
Firmware Package Name and Version  
STM32Cube FW\_F1 V1.8.0

☒ Use Default Firmware Location  
C:/Users/admin/STM32Cube/Repository/STM32Cube\_FW\_F1\_V1.8.0

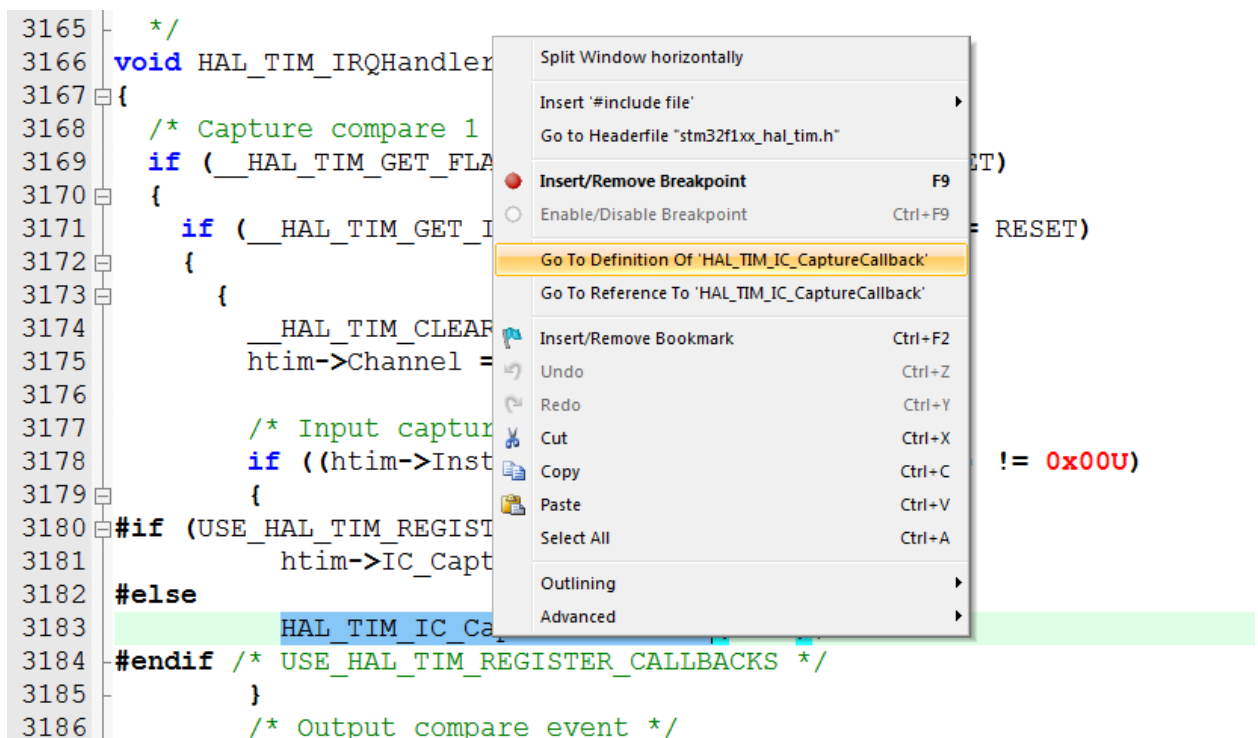
Bài 6:

## Lập trình STM32 Timer chế độ IC và OC

Open Project Nhấn F7 để Build Code, Trong stm32f1xx\_it.c các bạn tìm đến hàm HAL\_TIM\_IRQHandler(&htim2); và Goto define nó



Trong phần Define hàm đó các bạn sẽ tìm thấy hàm Call\_back cho ngắt Input Capture, Go to Define hàm đó copy và paste vào phần tiền xử lý trên hàm main()



Trong main.c chúng ta tạo một biến chứa Giá trị đọc được từ Counter, trong hàm xử lý ngắt ta sẽ phân luồng ngắt, đọc giá trị Capture, sau đó xóa giá trị Counter về 0.

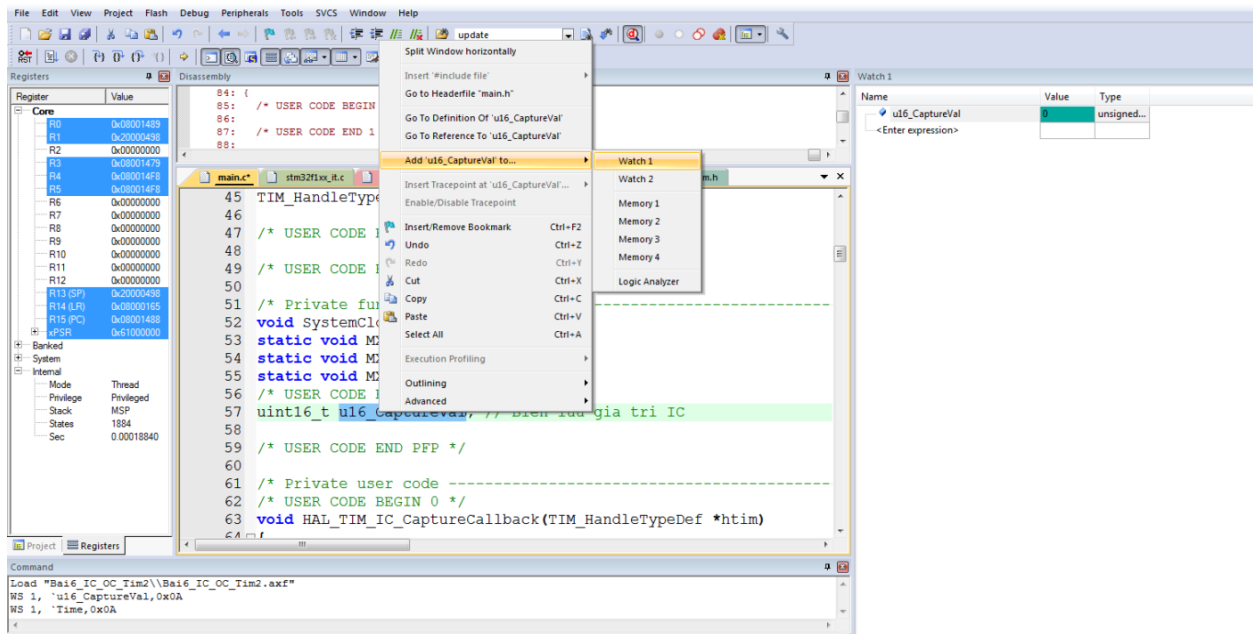
```
61 /* Private user code -----*/
62 /* USER CODE BEGIN 0 */
63 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
64 {
65     /* Prevent unused argument(s) compilation warning */
66     UNUSED(htim);
67     if(htim->Instance == TIM2) // Ngat IC thuoc TIM2
68     {
69         u16_CaptureVal = __HAL_TIM_GET_COMPARE(&htim2, TIM_CHANNEL_1); // Doc gia tri Capture
70         HAL_TIM_SET_COUNTER(&htim2, 0); // Xoa Counter
71     }
72     /* NOTE : This function should not be modified, when the callback is needed,
73              the HAL_TIM_IC_CaptureCallback could be implemented in the user file
74              */
75 }
76 /* USER CODE END 0 */
77
```

Trước While(1), chúng ta viết các hàm khởi chạy IC và OC

```
103
104 /* Initialize all configured peripherals */
105 MX_GPIO_Init();
106 MX_TIM2_Init();
107 MX_TIM3_Init();
108 /* USER CODE BEGIN 2 */
109 HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1); // bat IC che do Ngat
110 HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_1); // Bat OC ko ngat
111 /* USER CODE END 2 */
112
113 /* Infinite loop */
114 /* USER CODE BEGIN WHILE */
115 while (1)
116 {
117     /* USER CODE END WHILE */
118
119     /* USER CODE BEGIN 3 */
120 }
121 /* USER CODE END 3 */
122 }
```

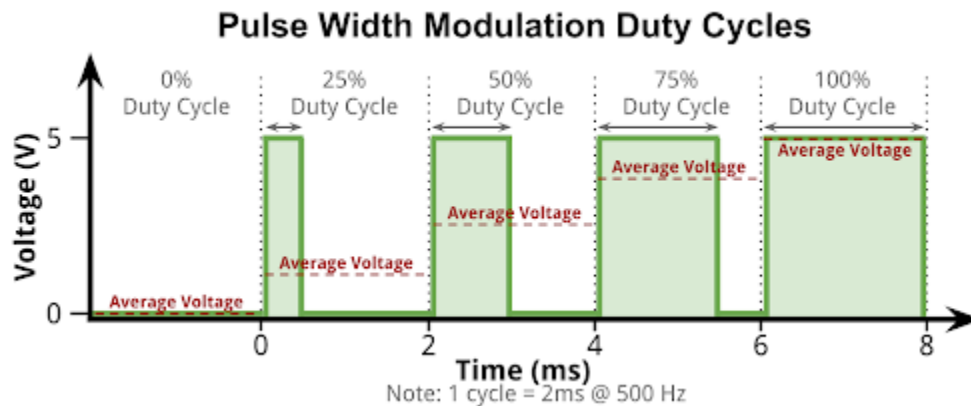
Build và nạp chương trình vào Kit.

Kết nối chân PA0 và chân PA6 vào nhau. (Timer 2 sẽ đọc xung từ Timer 3 truyền tới). Bật Debug, click vào tên của biến u16\_CaptureVal chọn Add to Watch 1



## PWM là gì?

**PWM** hay Pulse Width Modulation là phương pháp điều chỉnh độ rộng của xung có chu kỳ cố định, nhằm tạo ra sự thay đổi điện áp tại đầu ra.



PWM ứng dụng nhiều trong việc điều khiển động cơ, các bộ nguồn xung boost, buck, nghịch lưu 1 pha, 3 pha...

Trong mỗi Timer có 4 kênh độc lập phát PWM

Chu kỳ xung của PWM được quản lý bằng thanh ghi PSC và thanh ghi ARR.

Duty Cycles được quản lý bằng thanh ghi CCR

Chi tiết tham khảo mục 15.3.9 trong Reference Manual

## Cấu hình PWM trong STM32 Timer

Tạo project mới trên chip STM32F103C8, trong thẻ Sys chọn Debug là Serial Wire

Trong Tab Timer 2 cấu hình như sau:

Clock source: internal clock

Channel 1: PWM generation CH1



Additional Software

TIM2 Mode and Configuration

Mode

Slave Mode

Trigger Source

Clock Source

Channel1

Channel2

Channel3

Channel4

Combined Channels

☐ Use ETR as Clearing Source

☐ XOR activation

☐ One Pulse Mode

XOR activation Disabled:  
Requires Channel 1 and 2, 1 and 3 or 1 and 4

Trong mục configuration

Prescaler: 8000 => mỗi lần đếm là 1ms, nói chi tiết trong Bài 5

Counter Mode: Up

Counter Period: 99 => ở đây mình tạo chu kì xung là 100ms

Auto-preload: Enable

Mode: PWM mode 1

Pulse: 24 => duty cycles chọn là  $25/100 = 25\%$

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Search (Ctrl+F)

⏪ ⏩

i

Counter Settings

Prescaler (PSC - 16 bits value)

8000

Counter Mode

Up

Counter Period (AutoReload Register...)

99

Internal Clock Division (CKD)

No Division

auto-reload preload

Enable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)

Disable (Trigger input effect not delayed)

Trigger Event Selection

Reset (UG bit from TIMx\_EGR)

PWM Generation Channel 1

Mode

PWM mode 1

Pulse (16 bits value)

25

Output compare preload

Enable

Fast Mode

Disable

CH Polarity

High

Bài

Đặt tên Project chọn toolchain là MDK-ARM V5 sau đó Gen code

---

Project Settings

Project Name

Bai7\_PWM

Project Location

D:\STM32 Cube Youtube\

Browse

Application Structure

Basic

☐ Do not generate the main()

Toolchain Folder Location

D:\STM32 Cube Youtube\Bai7\_PWM\

Toolchain / IDE

MDK-ARM

Min Version

V5

☐ Generate Under Root

Linker Settings

Minimum Heap Size

0x200

Minimum Stack Size

0x400

Mcu and Firmware Package

Mcu Reference

STM32F103C8Tx

Firmware Package Name and Version

STM32Cube FW\_F1 V1.8.0

☒ Use Default Firmware Location

C:/Users/admin/STM32Cube/Repository/STM32Cube\_FW\_F1\_V1.8.0

Browse

Bài 7:

## Lập trình PWM

Trong KeilC bạn chỉ cần gọi hàm Start PWM là PWM sẽ chạy như sau:

```

/* USER CODE END SysInit */

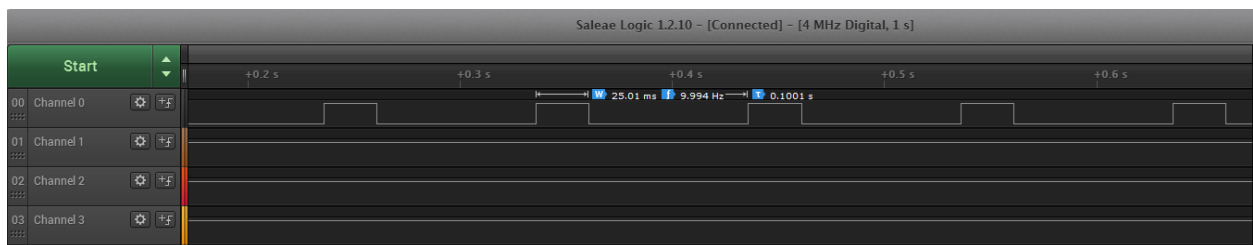
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{

```

Ta mắc kênh CH0 của bộ Analyzer vào CH1 của Timer 1 rồi đo xung nhé.  
Nếu bạn chưa biết sử dụng thì hãy đọc bài : Hướng dẫn sử dụng Logic Analyzer

Nhấn Start ta sẽ đo được sóng có chu kỳ 100ms và duty là 25%



Muốn thay đổi Duty Cycles chúng ta gọi hàm  
\_\_HAL\_TIM\_SET\_COMPARE(&htim2,TIM\_CHANNEL\_1, 50);

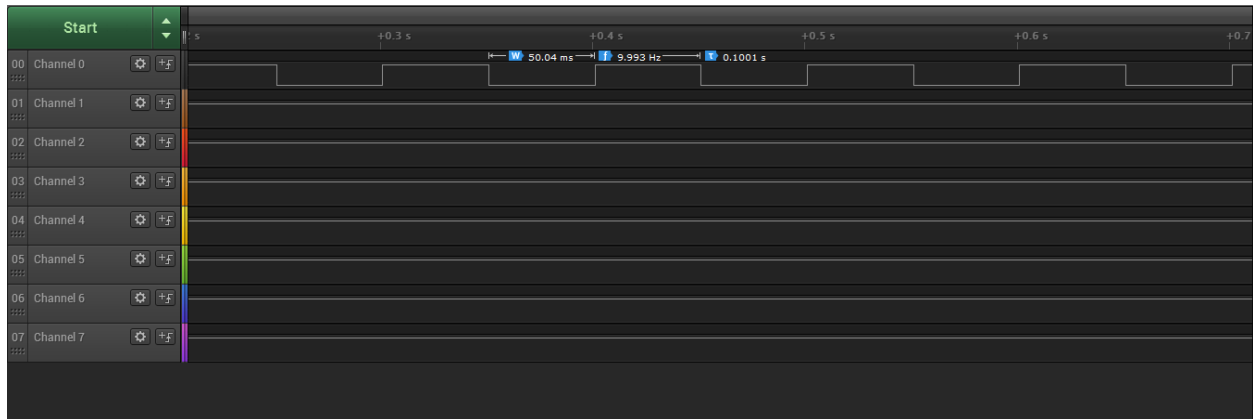
Với 50 là duty cần nạp vào. Sau đó nạp lại

```

88
89 /* Initialize all configured peripherals */
90 MX_GPIO_Init();
91 MX_TIM2_Init();
92 /* USER CODE BEGIN 2 */
93 HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
94 /* USER CODE END 2 */
95
96 /* Infinite loop */
97 /* USER CODE BEGIN WHILE */
98 while (1)
99 {
100     /* USER CODE END WHILE */
101
102     /* USER CODE BEGIN 3 */
103     __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1, 50);
104     }
105     /* USER CODE END 3 */
106 }
107

```

Đo lại kết quả sẽ được như sau:



Trong thực tế chúng ta sẽ quản lý duty bằng 1 biến, sau đó thay đổi giá trị duty nạp vào như sau.

Khởi tạo một biến `u8_Duty`

Trong `while(1)` tăng giảm `u8_Duty` đó mỗi 1s một lần

```
88
89  /* Initialize all configured peripherals */
90  MX_GPIO_Init();
91  MX_TIM2_Init();
92  /* USER CODE BEGIN 2 */
93  HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1);
94  uint8_t u8_Duty = 10; // 10/100 = 10%
95  /* USER CODE END 2 */
96
97  /* Infinite loop */
98  /* USER CODE BEGIN WHILE */
99  while (1)
100  {
101      /* USER CODE END WHILE */
102
103      /* USER CODE BEGIN 3 */
104      if (u8_Duty <= 90) u8_Duty += 10; // tăng giá trị duty lên 10
105      else u8_Duty = 10;
106      HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1, u8_Duty);
107      HAL_Delay(1000); // cứ 1s sẽ thay đổi 1 lần
108  }
109  /* USER CODE END 3 */
110
111  }
```

Lắp thêm Led tại chân PA0( CH1 của Timer 2) và thấy rằng Led thay đổi độ sáng theo Duty đó.

Lưu ý: Các kênh PWM của cùng 1 Timer dùng chung thanh ghi ARR và PRS vì vậy chu kỳ PWM sẽ giống nhau. Chúng ta chỉ setup được Duty khác nhau mà thôi.