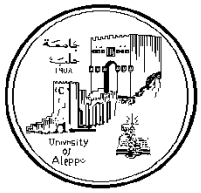
جامعة حلب
كلية الهندسة الكهربائية والالكترونية
قسم هندسة التحكم والأتمتة
مخبر التحكم



مقرر_____

الجلسة الخامسة

السنة _____

2023ـ2022

الغاية من الجلسة

1- التعرف على أنماط العمل المختلفة للمؤقتات في متحكمات STM32
2- التطبيق1: استخدام المؤقت في نمط Timer mode
3- التطبيق2: استخدام المؤقت في نمط PWM mode

1- أنماط العمل المختلفة للمؤقتات في متحكمات STM32:

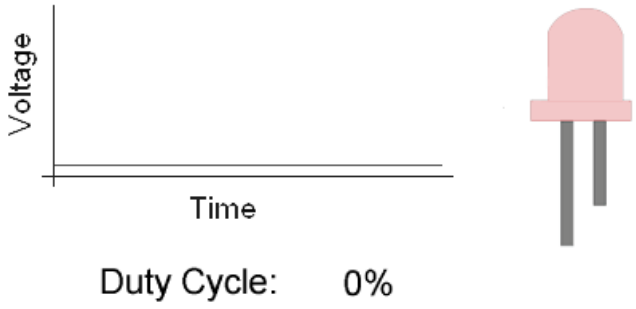
للمؤقتات في متحكمات STM32 أنماط عمل مختلفة سنذكر منها نمطين هما :

1. نمط المؤقت Timer Mode:

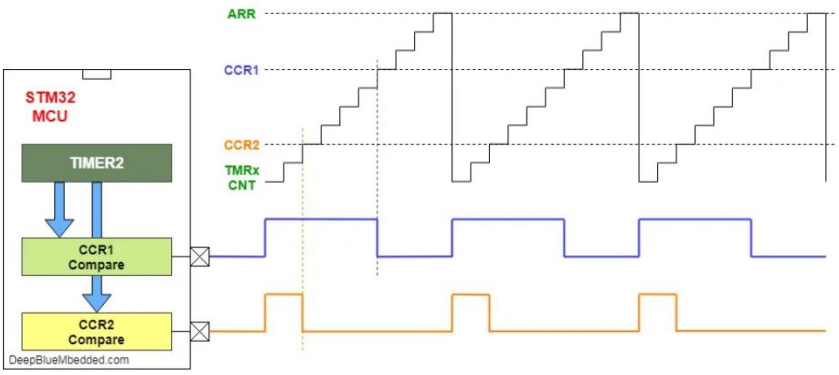
في هذا النمط من العمل فإن المؤقت يحصل على نبضات الساعة من ساعة المتحكم وباعتبار أن تردد ساعة المتحكم معروف بالتالي يمكن حساب زمن طفحان المؤقت كما يمكن التحكم بزمن الطفحان من خلال مسجل preload register من أجل الحصول على أي زمن مراد، وعند حدوث الطفحان تحدث مقاطعة الطفحان، هذا النمط من العمل يستخدم عادةً من أجل جدولة ومزامنة الأعمال والمهام المطلوبة من المتحكم خلال أزمنة معينة لكل مهمة من المهام ، كما يمكن استخدامه لاستبدال دالة التأخير الزمني الـ Delay() - مما يؤدي إلى رفع مستوى الأداء للمعالج.

2. نمط تعديل عرض النبضة PWM Mode:

في هذا النمط من العمل فإن نبضات الساعة للـ Timer - تأتي داخلياً من ساعة المتحكم ، حيث يقوم الـ Timer بتوليد إشارة رقمية PWM على قطب الخرج للمؤقت من خلال استخدام مسجلات المقارنة OCR، حيث تتم مقارنة القيمة الحالية للمؤقت مع القيمة الموجودة في مسجل المقارنة OCR وعندما تتساوى القيم يتم عكس الحالة المنطقية لقطب الخرج حتى نهاية الدور ثم تعاد العملية مرة أخرى، حيث يمكن التحكم بتردد نبضات الـ PWM وأيضاً دورة التشغيل duty cycle برمجياً من خلال المسجلات المناسبة، وتتعلق دقة الـ PWM بتردد الإشارة أي F_{PWM}



لكل مؤقت من مؤقتات المتحكم STM32 عدة قنوات ، لذا فإن كل مؤقت بإمكانه توليد عدة إشارات PWM لكل منها دورة تشغيل مختلفة ولكن لها نفس التردد وتعمل بالتزامن مع بعضها ، يوضح الشكل التالي مخطط للمؤقت TIM2 ويبين وجود عدة قنوات للمؤقت output compare channels:



تردد إشارة الـ PWM:

تحتاج في كثير من التطبيقات لتوليد نبضات PWM بتردد معين، كالتحكم بمحرك السيرفو، التحكم بإضاءة الليدات ، قيادة المحركات والعديد من التطبيقات الأخرى، حيث يتم التحكم بدور إ شارة الـ PWM (1/FPWM) من خلال البارامترات التالية:

- قيمة المسجل ARR(Auto Reload register)
- قيمة المقسم الترددي(PSC) Prescaler
- تردد الساعة الداخلية internal clock
- عدد مرات التكرار

وذلك من خلال العلاقة التالية:

$$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) \times (PSC + 1) \times (\mathrm{RCR} + 1)}$$

مثال:

بفرض أن F_{CLK} =72MHZ ، Prescaler=1، ARR=65535، RCR =0، احسب تردد نبضات الـ PWM:

$$F_{PWM} = \frac{72 \times (10\text{^}6)}{(65535 + 1) \times (1 + 1) \times 1} = 549.3HZ$$

دورة التشغيل Duty Cycle:

عند عمل المؤقت بنمط PWM وتوليد النبضات في وضع الـ edge-aligned mode up-counting ، فإن دورة التشغيل يتم حسابها من خلال العلاقة التالية:

$$DutyCycle_{PWM}[\%] = \frac{CCRx}{ARRx}[\%]$$

2- ضبط إعدادات المؤقتات في متحكمات STM32 :

كما ذكرنا سابقاً فإن المؤقت عبارة عن عداد بإمكانه العد بشكل تصاعدي فقط، حيث يقوم بالعد من الصفر إلى القيمة المحددة في حقل الـ Period($Preload$) - أثناء تهيئة المؤقت ، وأكبر قيمة يمكن أن يصل إليها تحدد حسب طول المؤقت، حيث المؤقت 16 بت يمكنه العد إلى 0xffff والمؤقت ذو 32 بت يمكنه العد إلى 0xffff ffff ، حيث يعتمد تردد (سرعة العد) على سرعة الناقل المتصل به المؤقت بالإضافة إلى المقسم الترددي Prescaler حيث يتم تقسيم تردد ساعة المؤقت على واحدة من القيم المتاحة وهي من 1 حتى 65535 (حيث أن مسجل الـ Prescaler - بطول 16بت)، وعندما يصل العداد إلى القيمة المحددة Period($Preload$) يحدث ما يسمى بالطفحان overflow أي يقوم بالتصفير والعد مرة أخرى من الصفر و يتم رفع العلم الخاص بالطفحان Update Event(UEV).

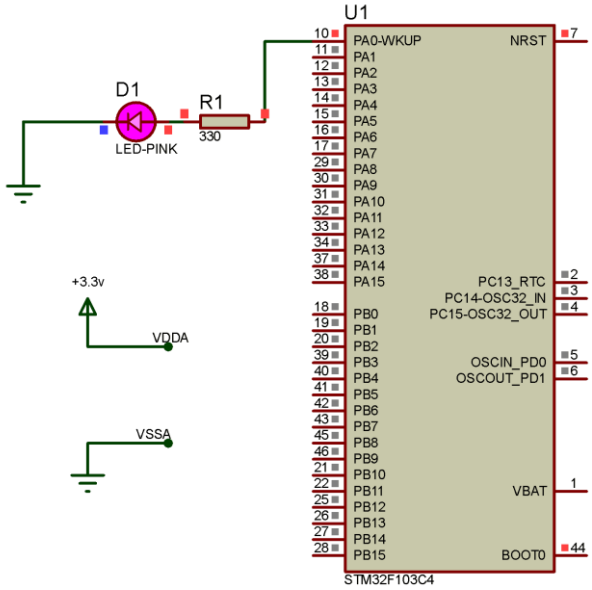
أي أن المسجلات الخاصة بالـ Period - ($Preload$) و Prescaler هي التي تحدد تردد المؤقت أي الزمن الذي سيستغرقه المؤقت حتى يحدث حدث الطفحان Overflow وعندها يتم رفع العلم UEV ، ويتم اختيار القيم المناسبة لهذه المسجلات بناءً على هذه المعادلة:

$$T_{out} = \frac{Prescaler \times Preload}{F_{CLK}}$$

على سبيل المثال ، لنفترض أن تردد الساعة للمتحكم مضبوط على 48MHz وقيمة الـ Prescaler - تساوي 48000 والـ period تساوي 500 سيحدث overflow للمؤقت كل:

$$T_{out} = \frac{48000 \times 500}{48000000} = 0.5sec$$

3- التطبيق العملي الأول: استخدام المؤقت في نمط Timer mode وبوضع المقاطعة لتوليد زمن بدلاً من استخدام دالة ()delay واستخدامه في عمل Toggle لليد الموصول على القطب PA5 :



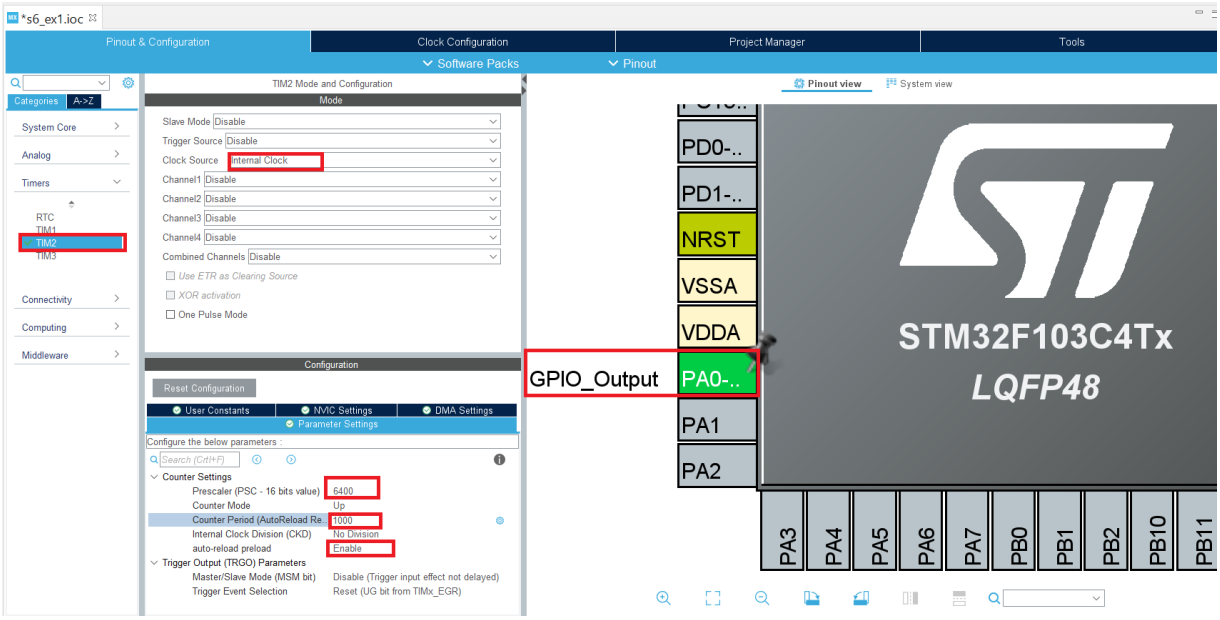
ضبط إعدادات المشروع:

الخطوة الأولى: فتح بيئة STM32CubeIDE وإنشاء مشروع جديد ثم اختيار المتحكم المناسب
الخطوة الثانية: اختيار اسم للمشروع
الخطوة الثالثة: اختر القطبPA5 لضبطه كقطب خرج
الخطوة الرابعة: ضبط إعدادات المؤقت
كي نحـ صل على زمن 100msec لعكس حالة الليد المو صول على القطب رقم 5 من المنفذ A ، من المعادلة الـ سابقة سنفترض أن تردد ساعة المتحكم هي 8MHz والمقـ سم الترددي 8000 بقي فقط حـ ساب(Preload) Period، بتعويض القيم في المعادلة :

$$T_{out} = \frac{Prescaler \times Preload}{F_{CLK}} = \frac{6400 \times Preload}{64000000}$$

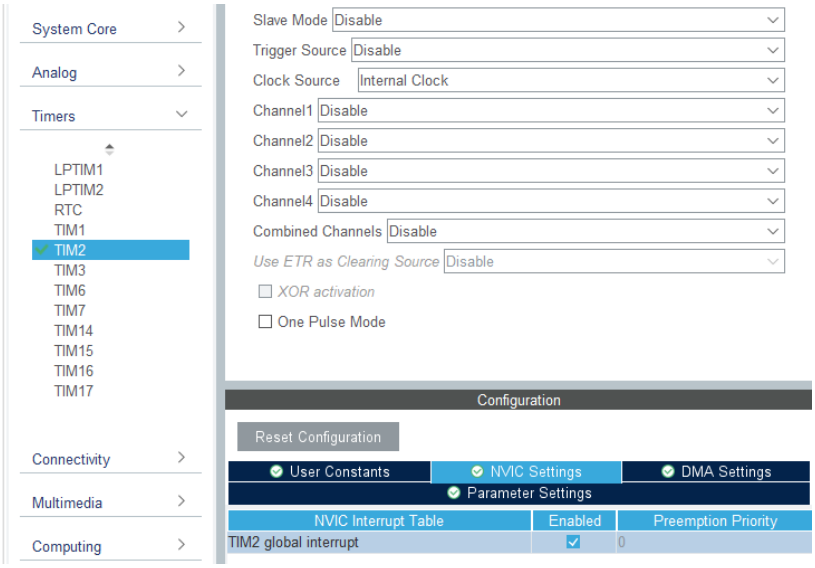
$$Preload = 100$$

سنقوم باختيار مصدر الساعة للمؤقت داخلي، المقسم الترددي8000 ، الـ Preload=100، أيضاً سنقوم بتفعيل إعادة التحميل التلقائي، كما في الشكل التالي:

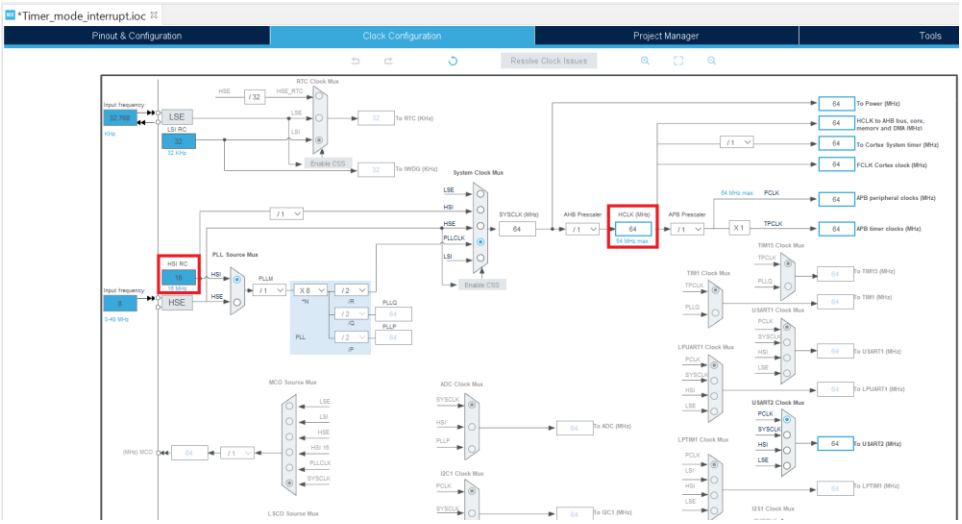


الخطوة الخامسة: تفعيل مقاطعة المؤقت من شريط الـ NVIC tab

مدرس المقرر: -----=



الخطوة السادسة: ضبط تردد ساعة المتحكم: أثناء الـ Simulation نختار تردد الساعة 8MHZ



الخطوة السابعة: توليد الكود بناءً على الإعدادات التي تم ضبطها

```c
#include "main.h"

TIM_HandleTypeDef htim2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{

  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();

  while (1)
  {

  }

}
```


مدرس المقرر: -----=

الخطوة الثامنة: بدء المؤقت

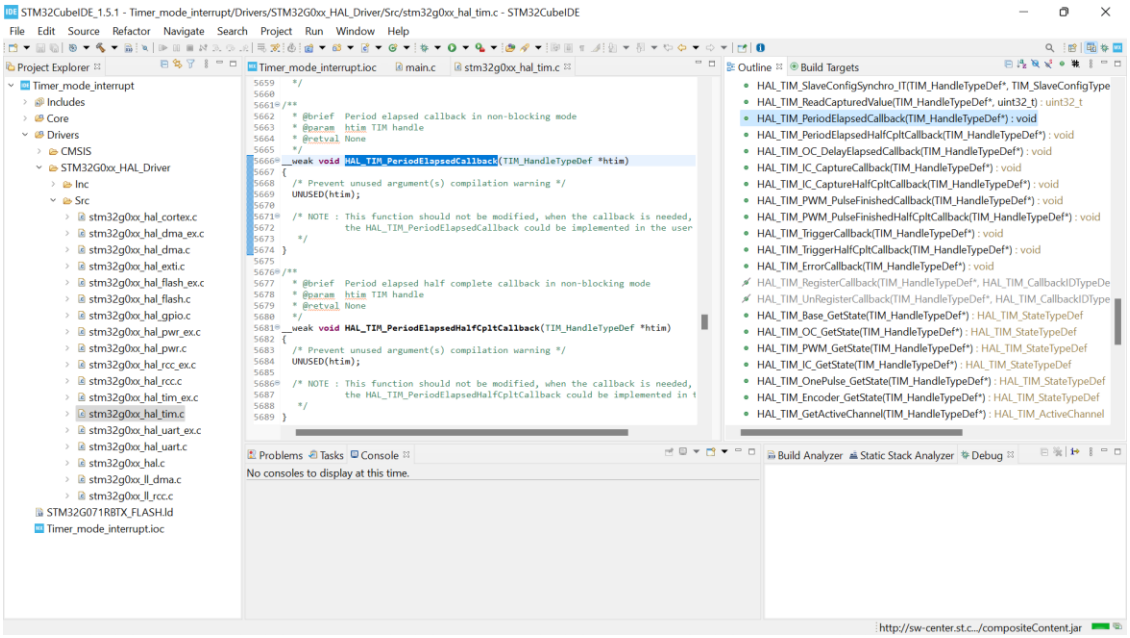
على الرغم من ضبط إعدادات المؤقت فإنه سيبقى في حالة IDLE أي خمول ولن يبدأ بالعد حتى تقوم باستدعاء الدالة الخاصة ببدأ عمل المؤقت وهي :

HAL_TIM_Base_Start_IT()

سنستدعي هذه الدالة في بداية الكود.

الخطوة التاسعة: إضافة دالة خدمة مقاطعة الطفحان

عند حدوث طفحان للمؤقت بوصوله للقيمة التي تم ضبطها في counter Period ، يذهب المعالج تلقائياً لبرنامج خدمة المقاطعة الخاص بالطفحان والذي يكون معرف بشكل افتراضي كـ weak_ ضمن ملف الـ stm32g0xx_hal_tim.c الموجود ضمن مجلد الـ Drivers ثم مجلد الـ Src و يدعى HAL_TIM_PeriodElapsedCallback()، حيث نقوم بنسخ اسمه وإضافته للبرنامج الرئيسي ثم نقوم ضمنه بعكس حالة الليد.



يصبح الكود بالشكل التالي:

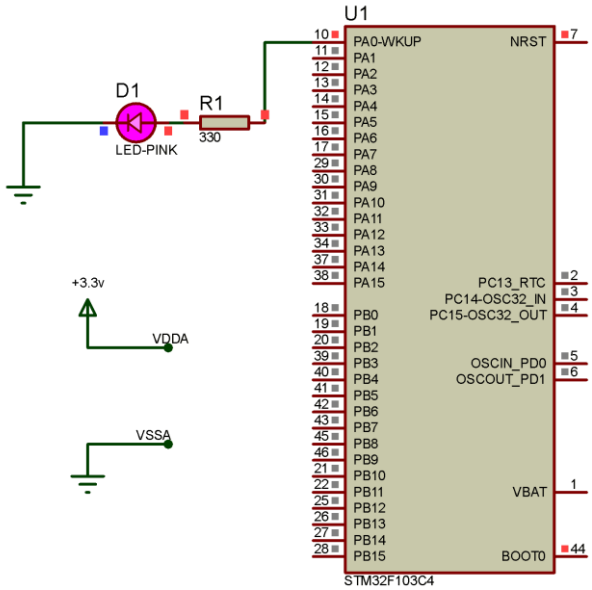
```
#include "main.h"

TIM_HandleTypeDef htim2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  HAL_TIM_Base_Start_IT(&htim2);
  while (1)
```

```
  {

  }

}
void HAL_TIM_PeriodElapsedCallback( TIM_HandleTypeDef* htim)
{

        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
}
```

الخطوة العاشرة: ترجمة الكود ورفعه للمتحكم ومراقبته من خلال فتح جلسة Debug

قم بالضغط على زر الـ Debug لترجمة الكود ورفعه للمتحكم من خلال الـDebug ، كما يمكنك بعد رفع الكود للمتحكم إغلاق جلسة الـ Debug وعمل Reset للمتحكم لبدء تنفيذ الكود الذي تم تحميله.

4- التطبيق العملي الثاني: استخدام المؤقت في نمط PWM mode واستخدامه للتحكم في شدة إضاءة الليد:



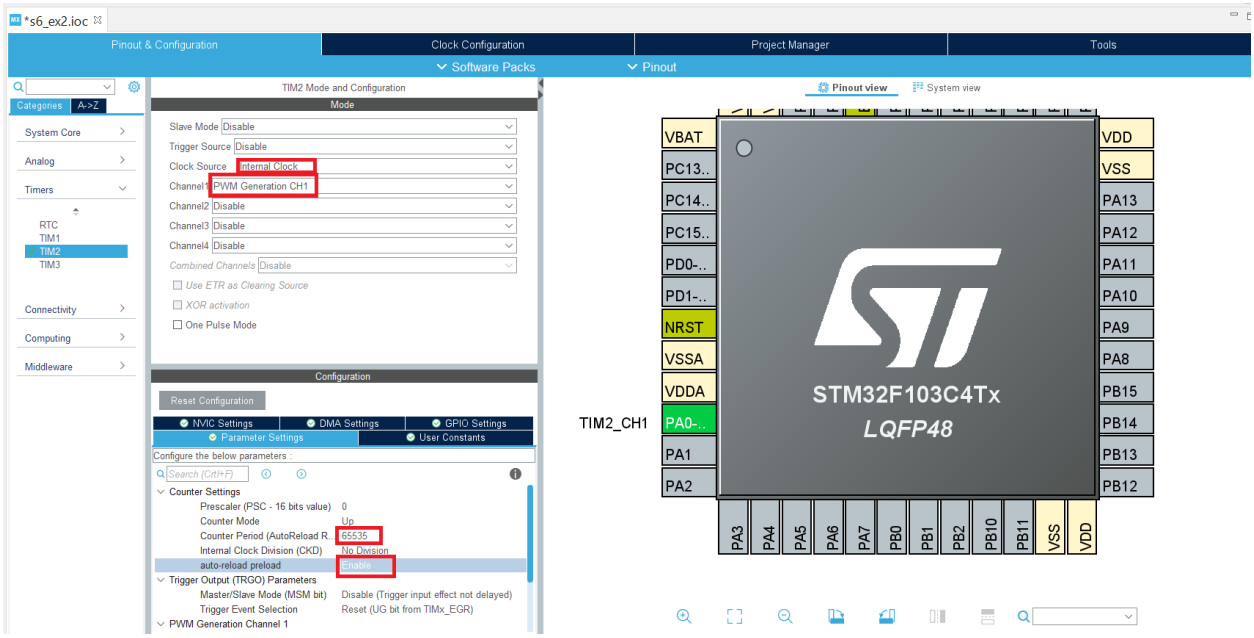
سنتبع في هذا التطبيق الخطوات التالية للتحكم بشدة إضاءة الليد:

- ضبط بارامترات المؤقت TIM2 ليعمل في نمط الـ PWM - وباستخدام الساعة الداخلية للمتحكم internal clock ، ثم تفعيل القناة الأولى CH1 لاستخدامها كقناة الخرج لإشارة الـ PWM
- ضبط قيمة المسجل ARR على القيمة العظمى وهي 65535 ، فيصبح التردد 488.25HZ
- التحكم بدورة التشغيل dutycycle من خلال كتابة القيمة المناسبة على المسجل CCMR1
- جعل دورة التشغيل تتغير من 0% حتى 100% وتعيد الكرة باستمرار

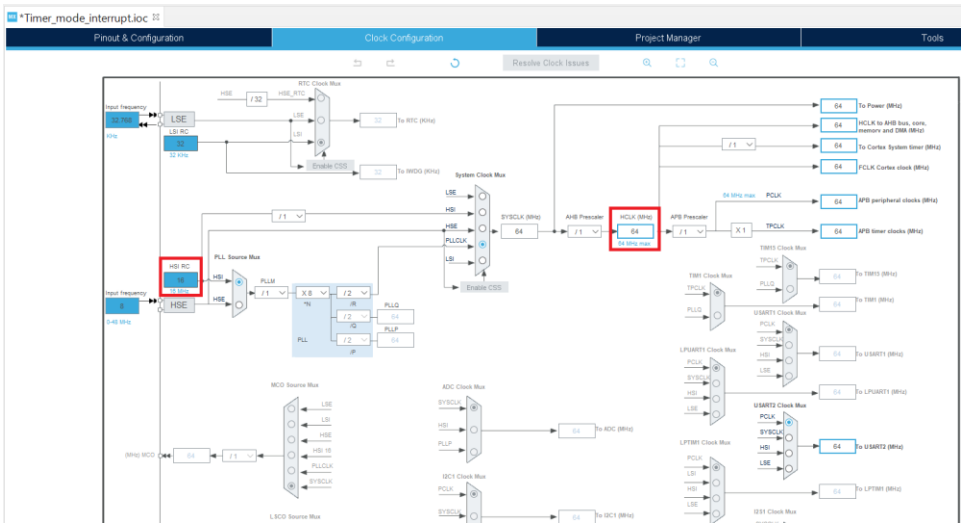
سنقوم بـ ضبط الإعدادات من خلال أداة CubeMx المدمجة داخل بيئة STM32CubeIDE وفقاً للخطوات التالية:

الخطوة الأولى: فتح بيئة STM32CubeIDE وإنشاء مشروع جديد ثم اختيار المتحكم المناسب واختيار اسم للمشروع

الخطوة الثالثة: ضبط إعدادات المؤقت ليعمل في نمط PWM نقوم بضبط مصدر الساعة للمؤقت على الساعة الداخلية للنظام internal clock ، نقوم بتفعيل القناة CH1 لتكون القناة التي سيتم إخراج إ شارة الـ PWM - عليها، نضبط القيمة العظمى للمسجل ARR على القيمة 65535 ليصبح تردد إشارة PWM هو488.25HZ ، نفعل خاصية Auto Reload preload ونختار نمط إشارة الـ PWM



الخطوة الثالثة: ضبط تردد ساعة المتحكم



الخطوة الرابعة: توليد الكود بناءً على الإعدادات التي تم ضبطها من خلال ctrl+s

الخطوة الخامسة : إضافة الدالة الخاصة ببدء المؤقت بالعمل وبنمط PWM:

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

يصبح الكود النهائي:

```
#include "main.h"
TIM_HandleTypeDef htim2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
//**************************
int main(void)
{
    int32_t CH1_DC = 0;
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_TIM2_Init();
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
 //**************************
    while (1)
    {
        while(CH1_DC < 65535)
        {
            TIM2->CCMR1 = CH1_DC;
            CH1_DC += 70;
            HAL_Delay(1);
        }
        while(CH1_DC > 0)
        {
            TIM2->CCMR1 = CH1_DC;
            CH1_DC -= 70;
            HAL_Delay(1);
        }
    }
}
```