Timers in STM32

(2)

محتويات الجلسة:

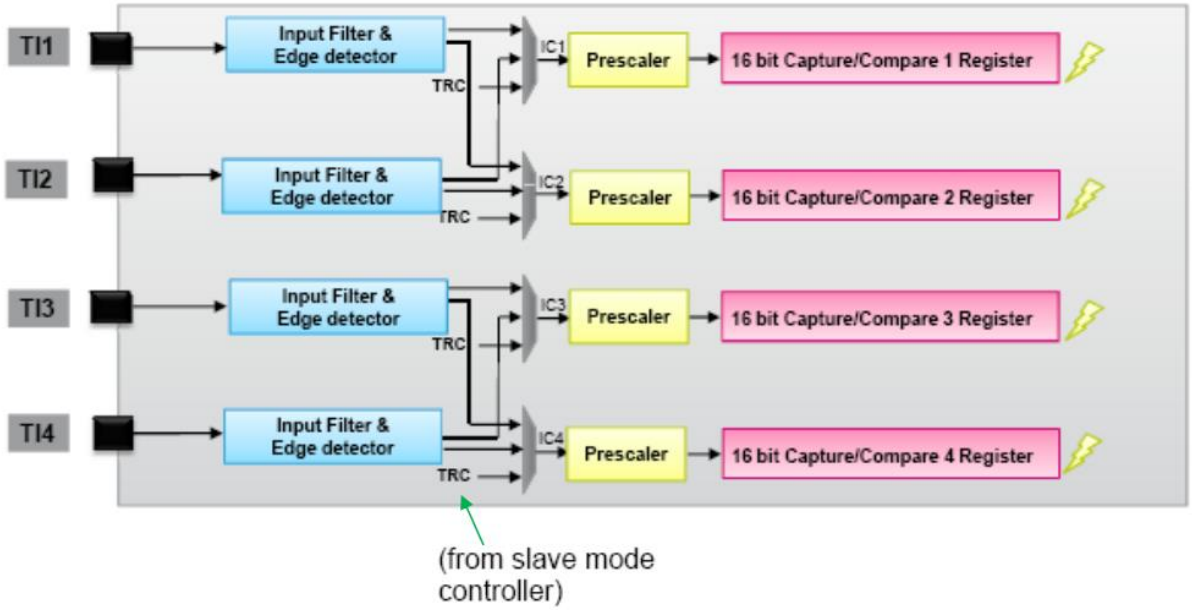
1- استخدام المؤقت في نمط Input Capture mode
2- التطبيق العملي الثالث
3- استخدام المؤقت في نمط PWM mode
4- التطبيق العملي الرابع
5- استخدام المؤقت في نمط Encoder interface mode
6- التطبيق العملي الخامس.

الأدوات اللازمة للجلسة:

- لوحة Nucleo-64bit
- كبل Type-A to Mini-B
- ليدات
- مفاتيح لحظية
- Rotary Encoder

1. استخدام المؤقت في نمط Input Capture mode:

كما ذكرنا سابقاً فإن المؤقت يمكن أن يعمل في نمط Input Capture أي يستقبل نبضات الساعة الخاصة به من مـ صدر داخلي (ساعة المتحكم بعد ا ستخدام المق سم الترددي) ويـ ستمر بالعد إلى أن يحدث حدث معين (جبهة صاعدة/ جبهة هابطة) على قطب المتحكم الخاص بقناة الـ - Input Capture Channel عندها يتم حفظ القيمة التي و صل إليها المؤقت إلى مـسجل input capture register ، حيث لكل مؤقت في متحكمات STM32 عدة قنوات (input capture/compare output) channels مرتبطة بأقطاب المتحكم يمكن معرفتها من خلال الـ datasheet الخاصة بالمتحكم.



الشكل(1): استخدام المؤقت في نمط Input Capture mode

في نمط Input Capture عند التقاط حدث معين (جبهة صاعدة/هابطة) على قناة Input Capture يتم تخزين قيمة العداد الحالية في مسجلات Capture/Compare Registers (TIMx_CCRx)، كما يتم رفع (تفعيل) العلم CCXIF flag الموجود في المسجل TIMx_SR register و طلب المقاطعة/ DMA (في حال كانت مفعلة) ، في حال حدث التقاط آخر (Capture) بينما كان العلم CCxIF أصلاً مفعل عندها سيتم رفع العلم CCxOF الموجود في المسجل TIMx_SR register .

يمكن تصفير العلم CCxIF عن طريق الكود من خلال كتابة صفر منطقي عليه أو عند قراءة البيانات المخزنة في المسجل TIMx_CCRx register ، كما يتم تصفير العلم CCxOF بكتابة صفر منطقي عليه.

2. التطبيق العملي الثالث: استخدام المؤقت في نمط Input Capture mode وبوضع المقاطعة لاستخدامه في قياس تردد إشارة معينة :

سنتبع في هذا المثال الخطوات التالية لقياس تردد إشارة معينة:

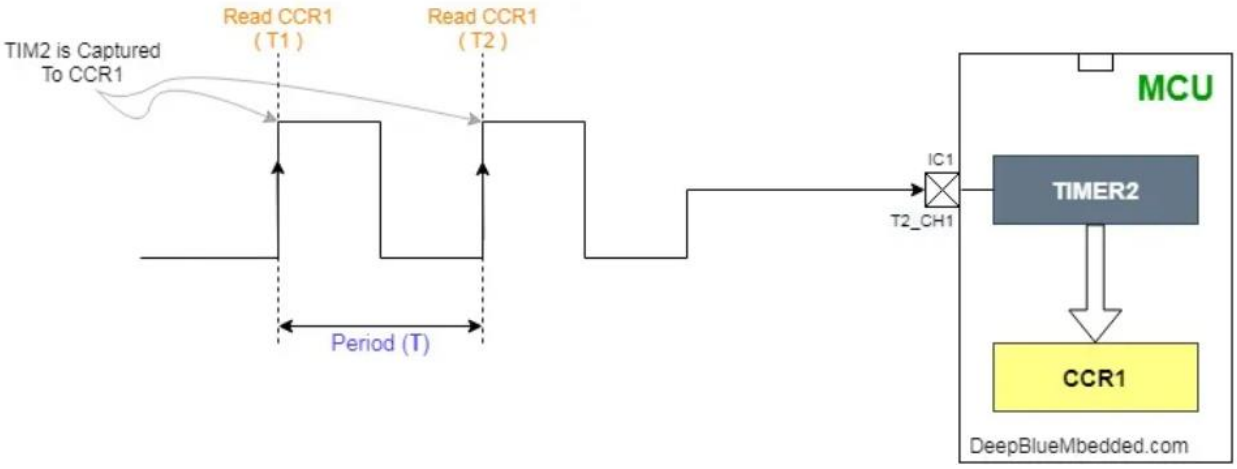
- ضبط تردد المؤقت TIM2 على الساعة الداخلية للمتحكم Internal clock ، و ضبط إعدادات القناة CH1 لاستخدامها في نمط Input Capture عند كل جبهة صاعدة.
- قراءة القيمة المخزنة في المسجل CCR1 وتخزينها في متحول وليكن T1 عند أول جبهة صاعدة يتم التقاطها على القناة ، ثم قراءتها مجدداً عند ثاني جبهة صاعدة يتم التقاطها، عندها يتم حساب دور الإشارة الملتقطة والذي يكون عبارة عن ناتج طرح القيمتين T2-T1 ثم بسهولة يمكن حساب تردد الإشارة الملتقطة والذي يمثل مقلوب الدور 1/period.
- ضبط إعدادات المنفذ التسلسلي USART2 في نمط Async mode وبمعدل نقل بيانات baud rate = 9600bps

- طباعة تردد الإشارة على المنفذ التسلسلي.

في هذا المثال سيمر المتحكم في حالتين سنقوم بتسميتهم اصطلاحاً (IDLE , DONE)، في حالة الـ IDLE - فإن المتحكم يكون جاهز لالتقاط الحدث الأول (الجبهة الـ صاعدة في حالتنا) أو كما ذ سميه the first timestamp باستخدام المؤقت TIM2 والذي يعمل في الخلفية باستمرار إلى حين وصول الجبهة الصاعدة على القطب IC1 .

عند الجبهة الصاعدة الأولى، يتم تخزين قيمة التي وصل إليها العداد في الـ TIM2 - إلى المسجل CCR1 ويتم طلب المقاطعة، ضمن برنامج خدمة المقاطعة ISR يتم تخزين قيمة الم سجل CCR1 في المتحول T1 و ستنتقل حالة المتحكم من IDLE إلى DONE .

في حالة الـ DONE state - ، فإن المتحكم يستمر بالعمل بشكل طبيعي إلى حين التقاط جبهة صاعدة علة القطب IC1 سيتم طلب المقاطعة مرة أخرى و سيتم تخزين قيمة الم سجل CCR1 فب المتحول T2 و سيتم ح ساب دور الإ شارة الملتقطة ويمثل (T2-T1) ويتم ح ساب التردد ((1/(T2-T1)) ثم يتم طباعته على المنفذ التـ سلـ سلي UART2 ، وستعود حالة المتحكم إلى IDLE state مجدداً، وتعاد هذه العملية باستمرار.

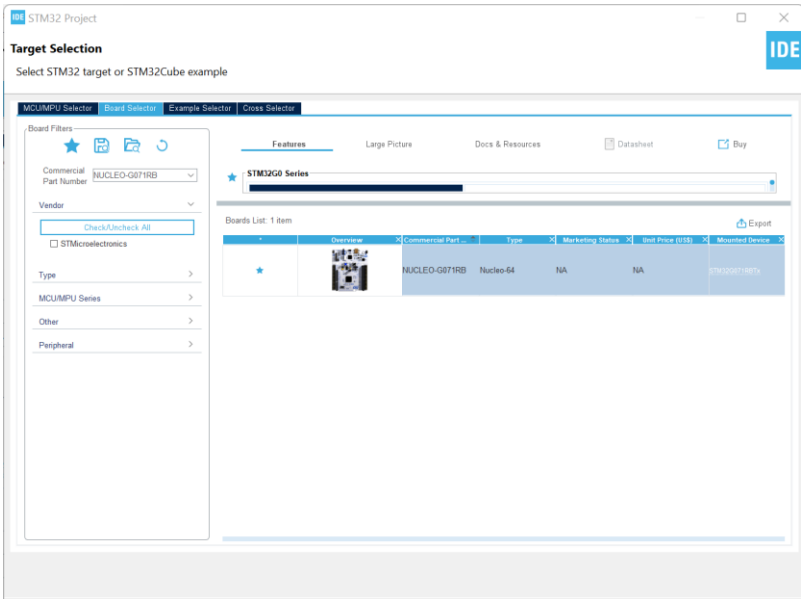


الشكل (2): المؤقت في نمط Input Capture

ملاحظة: عند قياس الإشارات ذات التردد المنخفض عندها يكون دور الإشارة طويل نسبياً ، في هذه الحالة قد يطفح المؤقت عدة مرات أثناء قياس دور الإشارة، لذا ومن أجل هذه الحالة يتم تفعيل مقاطعة الطفحان للمؤقت ونقوم بعد عدد مرات طفحان المؤقت بدءاً من اللحظة التي تم فيها تخزين القيمة في المتغير T1 ويتم إضافة عدد الطفحانات للمؤقت إلى القيمة التي سيتم تخزينها في المتغير T2 ، حيث كل طفحان يمثل 65636 عدة.

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

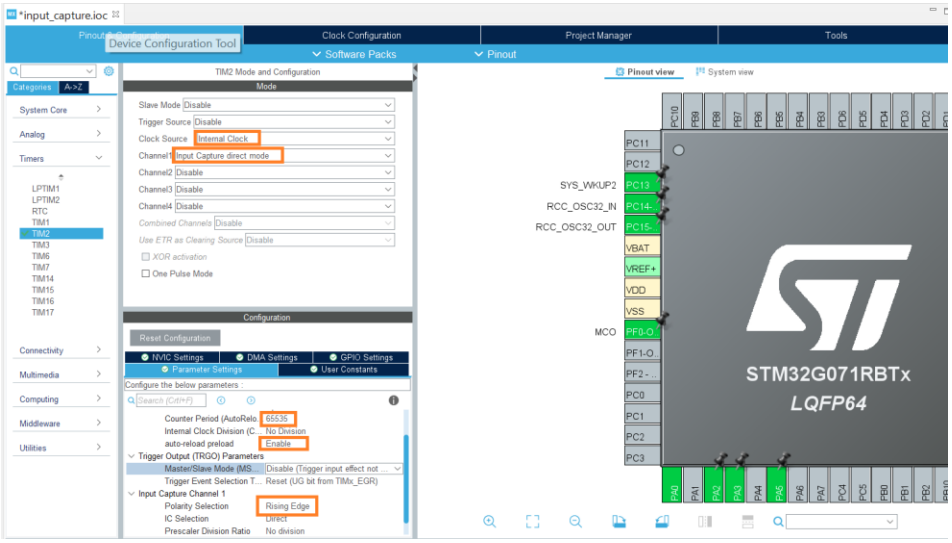
الخطوة الأولى: قم بفتح برنامج STM32CubeIDE ومن ثم قم بإن شاء م شروع جديد من نافذة File ثم New ثم STM32Project ثم قم باختيار المتحكم الم صغر أو من خلال اختيار ا سم اللوحة الم ستخدمة وهي في حالتنا Nucleo-G071RB كما في الشكل التالي:



الشكل(3) :بدء مشروع جديد في بيئة STM32CubeIDE

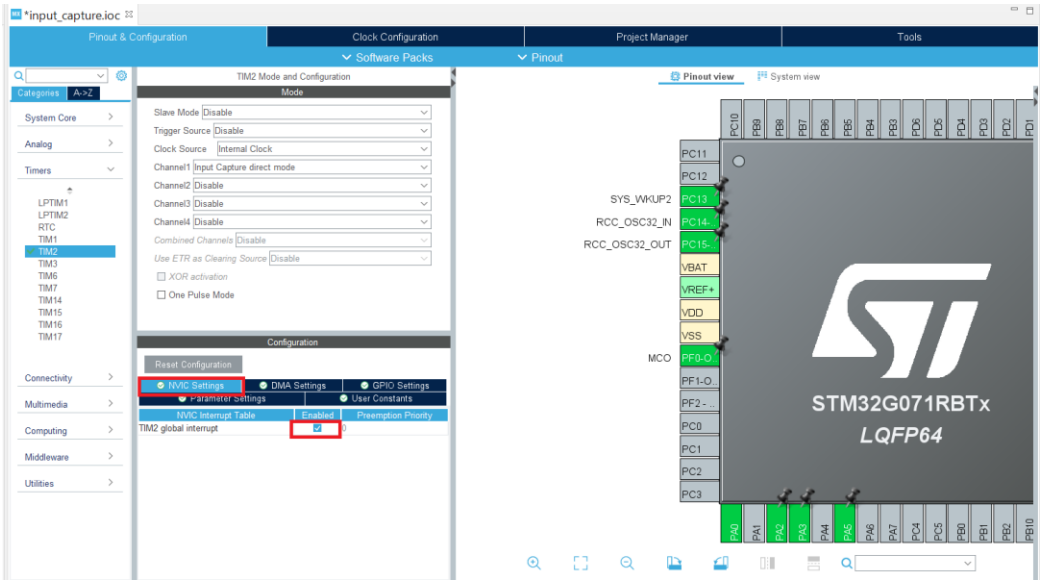
الخطوة الثانية: ضبط إعدادات المؤقت ليعمل كمقياس تردد

نقوم بضبط مصدر الساعة للمؤقت على الساعة الداخلية للنظام internal clock



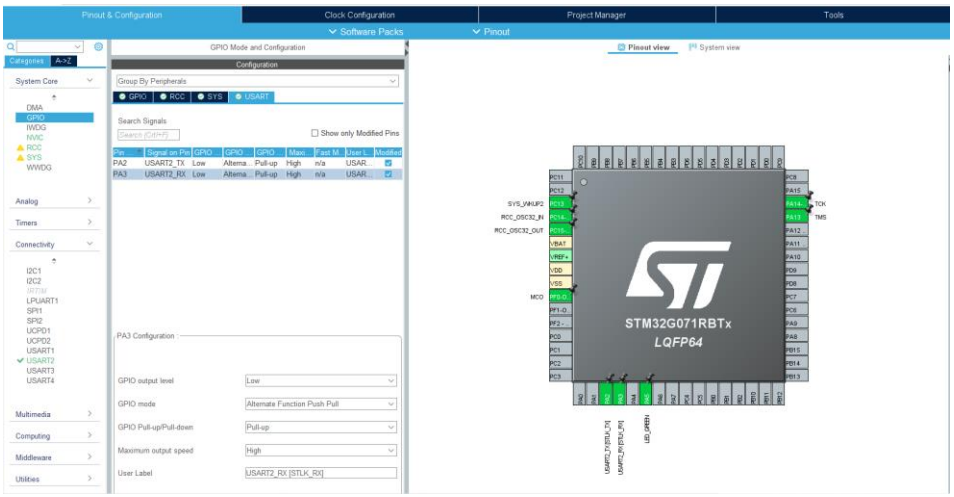
الشكل(4): ضبط إعدادات المؤقت كي يعمل في نمط Input Capture

الخطوة الثالثة: تفعيل مقاطعة المؤقت من خلال قائمة NVIC

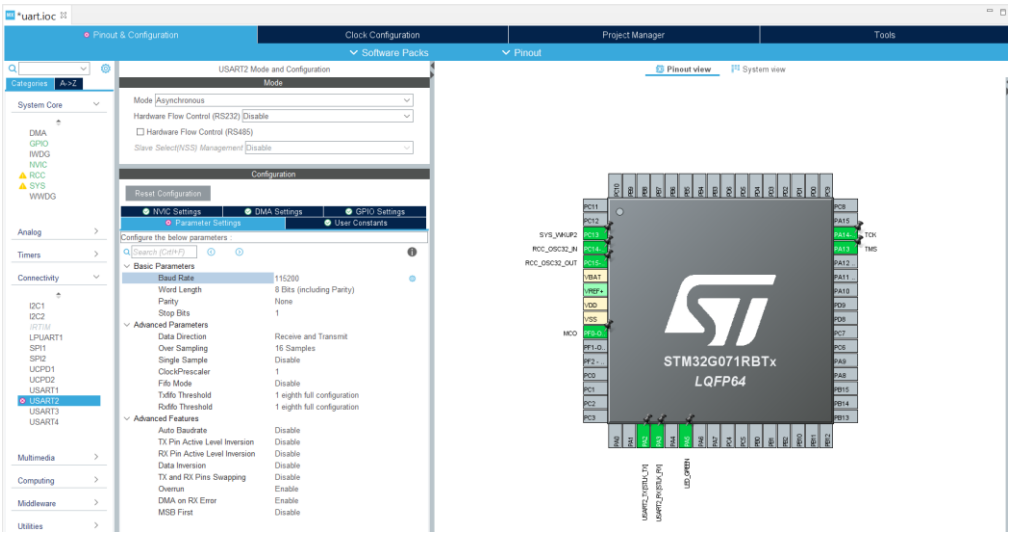


الشكل(5):تفعيل مقاطعة المؤقت

الخطوة الرابعة: قم بضبط إعدادات المنفذ UART2 المتصل داخلياً مع دارة الـ ST-LINK المدمجة مع اللوحة، كما في الشكل التالي:

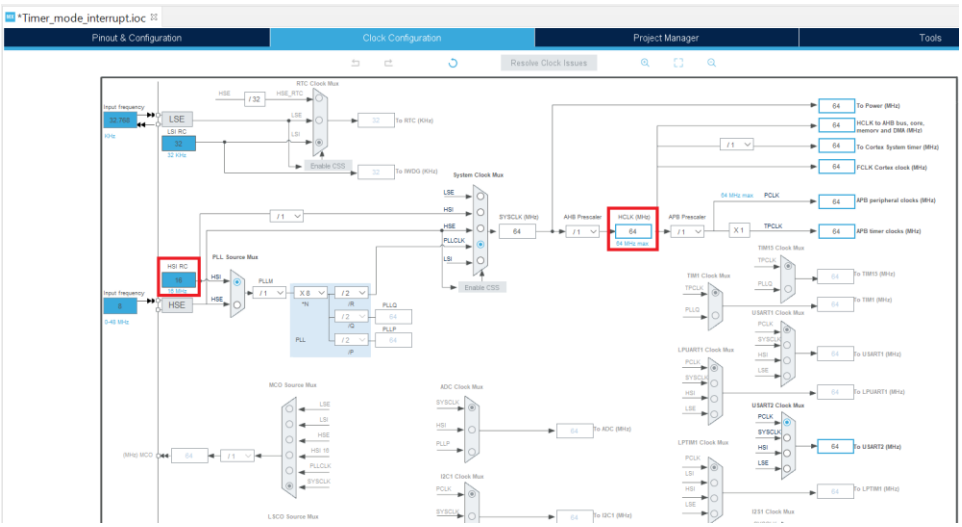


الشكل(6):ضبط إعدادات المنفذ التسلسلي UART2



الشكل(7):ضبط إعدادات المنفذ التسلسلي UART2

الخطوة الخامسة: ضبط تردد ساعة المتحكم



الشكل(8):ضبط تردد ساعة المتحكم

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

```
#include "main.h"

TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
```

```
int main(void)
{
   HAL_Init();
   SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();
   while (1)
  {

  }}
```

الخطوة السابعة: 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 في نمط المقاطعة وهي :

```
HAL_TIM_Base_Start_IT(&htim2);
```

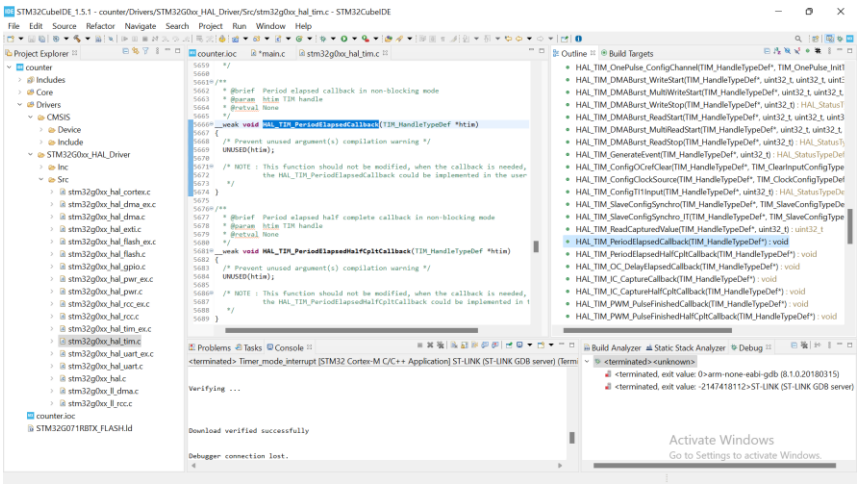
سنستدعي هذه الدالة في بداية الكود.

كما يجب استدعاء الدالة الخاصة ببدأ العمل في نمط الـ Input Capture:

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

الخطوة الثامنة: إضافة دالة خدمة مقاطعة الطفحان

عند حدوث طفحان للمؤقت بوصوله للقيمة في counter Period وهي 65535 ، 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 ويدعى HAL_TIM_PeriodElapsedCallback()، حيث نقوم بنسخ اسمه وإضافته للبرنامج الرئيسي:



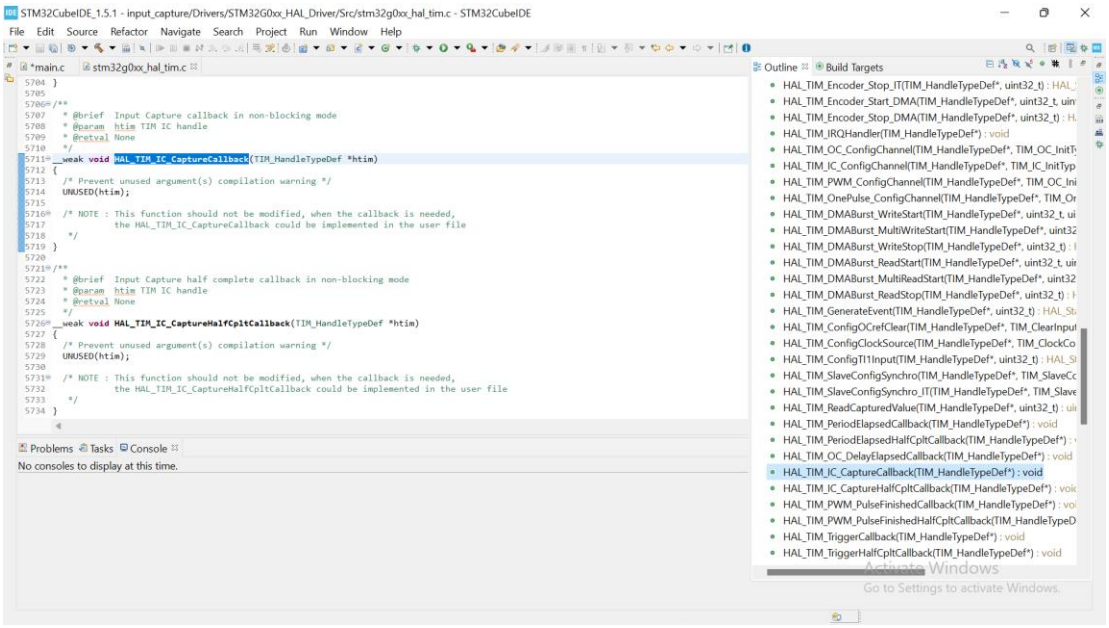
الشكل(9): نسخ دالة خدمة مقاطعة الطفحان

ثم نقوم ضمنه بزيادة قيمة المتغير gu16_TIM2_OVC والذي يعبر عن عدد مرات الطفحان:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
    gu16_TIM2_OVC++;
                                                    }
```

الخطوة التاسعة: إضافة دالة خدمة مقاطعة Input Capture

عند التقاط جبهة صاعدة على قطب IC1 ، يذهب المعالج تلقائياً لبرنامج خدمة المقاطعة الخاص بـ Input Capture والذي يكون معرف بشكل افتراضي كـ weak_ ضمن ملف الـ stm32g0xx_hal_tim.c الموجود ضمن مجلد الـ Drivers ثم مجلد الـ Src - ويدعى HAL_TIM_IC_CaptureCallback()، حيث نقوم بنسخ اسمه وإضافته للبرنامج الرئيسي ثم نقوم ضمنه بزيادة قيمة المتغير gu16_TIM2_OVC والذي يعبر عن عدد مرات الطفحان.



الشكل(10): نسخ دالة خدمة مقاطعة Input Capture

حيث نقوم ضمنه بفحص حالة المتحكم الحالية التي تحدثنا عنها وهو إما IDLE أو DONE ، في حال كان المتحكم في IDLE state أي عند التقاط أول جبهة صاعدة على القطب IC1 ، في هذه الحالة يتم تخزين قيمة المسجل CCR1 في المتغير gu32_T1 وتصفير عدد مرات الطفحان gu16_TIM2_OVC، وتفعيل حالة الـ DONE state.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim)
{
  if(gu8_State == IDLE)
  {
    gu32_T1 = TIM2->CCR1;
    gu16_TIM2_OVC = 0;
    gu8_State = DONE;
  }
```

```
  else if(gu8_State == DONE)
  {
     gu32_T2 = TIM2->CCR1;
     gu32_Ticks = (gu32_T2 + (gu16_TIM2_OVC * 65536)) - gu32_T1;
     gu32_Freq = (uint32_t)(F_CLK/gu32_Ticks);
     if(gu32_Freq != 0)
     {
      sprintf(gu8_MSG, "Frequency = %lu Hz\n\r", gu32_Freq);
      HAL_UART_Transmit(&huart2, gu8_MSG, sizeof(gu8_MSG), 100);
     }
     gu8_State = IDLE;
      }
}
```

يصبح الكود بالشكل التالي:

```
#include "main.h"
#define IDLE   0
#define DONE   1
#define F_CLK  72000000UL
 //*************************
volatile uint8_t gu8_State = IDLE;
volatile uint8_t gu8_MSG[35] = {'\0'};
volatile uint32_t gu32_T1 = 0;
volatile uint32_t gu32_T2 = 0;
volatile uint32_t gu32_Ticks = 0;
volatile uint16_t gu16_TIM2_OVC = 0;
volatile uint32_t gu32_Freq = 0;
//*************************
TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
//*************************
int main(void)
{
   HAL_Init();
   SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1);
   while (1)
```

```
  {

  }
 }
//*********************************************
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim)
{
    if(gu8_State == IDLE)
    {
        gu32_T1 = TIM2->CCR1;
        gu16_TIM2_OVC = 0;
        gu8_State = DONE;
    }
    else if(gu8_State == DONE)
    {
        gu32_T2 = TIM2->CCR1;
        gu32_Ticks = (gu32_T2 + (gu16_TIM2_OVC * 65536)) - gu32_T1;
        gu32_Freq = (uint32_t)(F_CLK/gu32_Ticks);
        if(gu32_Freq != 0)
        {
          sprintf(gu8_MSG, "Frequency = %lu Hz\n\r", gu32_Freq);
          HAL_UART_Transmit(&huart2, gu8_MSG, sizeof(gu8_MSG), 100);
        }
        gu8_State = IDLE;
    }
}
//*********************************************
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
    gu16_TIM2_OVC++;
}
```

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

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

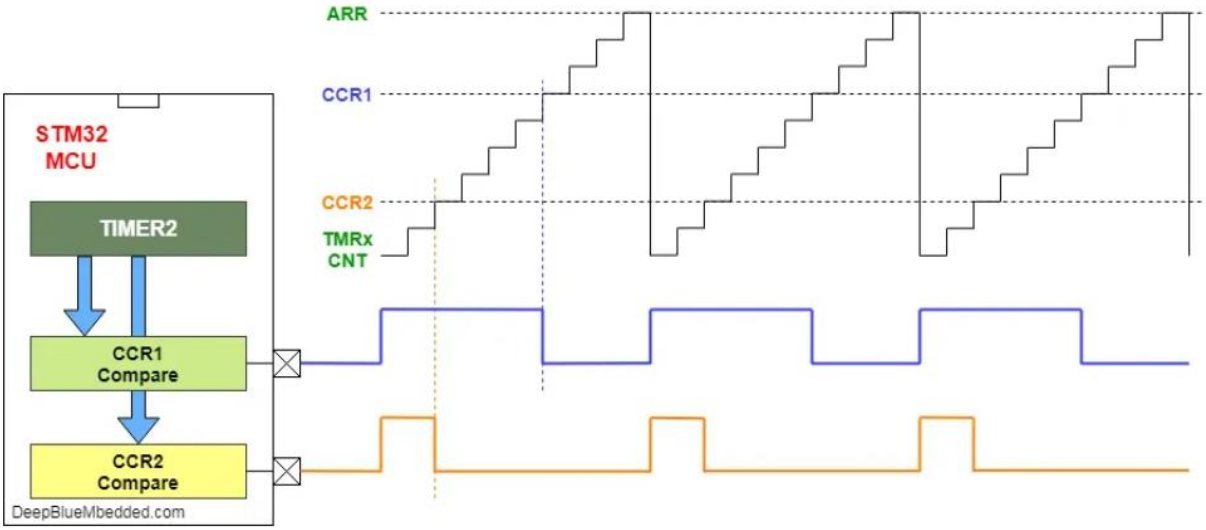
3. استخدام المؤقت في نمط PWM mode

كما ذكرنا سابقاً فإن المؤقت يمكن أن يعمل في نمط PWM mode أي يستقبل نبضات الساعة الخاصة به من الساعة الداخلية للمتحكم حيث يبدأ بالعد من الصفر ويزداد مع كل نبضة ساعة للمتحكم (طبعاً مع مراعاة إعدادات المقسم الترددي للمؤقت) ويتم وضع قطب الخرج الخاص بالـ PWM في وضع HIGH ويبقى كذلك إلى أن يصل

العداد إلى القيمة المخزنة في المسجل CCRx عندها يصبح قطب الخرج في وضع LOW إلى أن يصل العداد إلى القيمة المخزنة في المسجل ARRx ، وهكذا ....

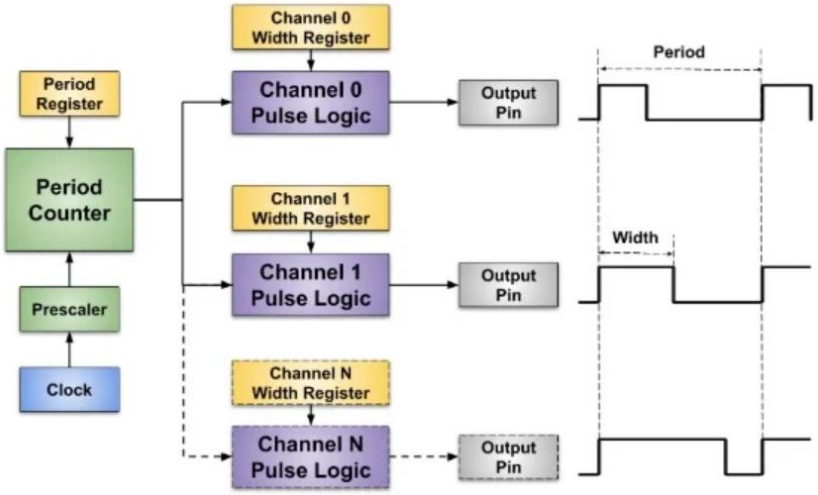
يدعى شكل الإشارة الناتجة بالـ PWM(Pulse Width Modulation) - ، حيث يتم التحكم بالتردد من خلال تردد الساعة الداخلية للنظام، والمقسم الترددي Prescaler بالإضافة إلى قيمة المسجل ARRx(Auto Reload register) ، كما يتم تحديد قيمة دورة التشغيل الـ duty cycle - من خلال قيمة المسجل الـ CCR1 - ، تعتبر هذه الطريقة الأسهل في توليد نبضات الـ PWM وهناك طرق أخرى سنتطرق إليها.

يوضح المخطط التالي كيفية تأثير قيمة المسجل ARR في دور(تردد) إشارة الـ PWM - ، وكيف تؤثر قيمة المسجل CCR1 في قيمة دورة التشغيل duty cycle ، كما يوضح كامل عملية توليد نبضات الـ PWM - في نمط UP-counting normal mode:



الشكل(11): UP-counting normal PWM mode

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:



الشكل(12): A PWM timer block diagram

يتيح نمط PWM mode توليد إ شارة بتردد يمكن التحكم به من خلال الم سجل TIMxARR وبدورة ت شغيل dutycycle يمكن التحكم بها من خلال قيمة الم سجل TIMxCCRx ، حيث يتم توليد إ شارة PWM عند كل قناة OCx للمؤ قت، من خلال كتابة 110 من أجل PWM mode1 ، أو 111 من أجل PWM mode2 في البتات OCxM الموجودة في المسجل TIMx-CCMRx ،

كما على المستخدم أن يقوم بتفعيل البت OCxPE في المسجل TIMx-CCMRX من أجل تفعيل مسجل Preload register ، كما يجب عليه أن يقوم بتفعيل البت ARPE في الم سجل TIMx-CR1 من أجل تفعيل الم سجل auto-reload preload register.

كما يتم تحديد قطبية (polarity)إشارة الـ PWM على القناة OCx من خلال الكود باستخدام البت CCxP الموجود في الم سجل TIMx-CCER حيث يمكن برمجته كي يكون إما active high أو active low، حيث تحتاج بعض التطبيقات إلى وجود الإشارة المكملة لإشارة الـ PWM.

في نمط الـ PWM يتم بـ شكل دائم مقارنة القيمة الحالية للعداد (TIMx_CNT) مع القيمة المخزنة في الم سجل CCRxوذلك لتحديد إذا كانت TIMx_CCRx≤TIMx_CNT أو TIMx_CNT≤TIMx_CCRx

يمكن للمؤقت توليد نبضات الـ PWM في نمط الـ edge-aligned mode أو في نمط الـ center-aligned mode وذلك تبعاً لـ CMS bits الموجودة في المسجل TIMx_CR1 register.

- **الفرق بين PWM mode1 و PWM mode2:**

في حالة PWM mode1: أثناء العد التـ صاعدي تكون القناة CH1 في حالة 1 منطقي (active) طالما TIMx_CNT<TIMx_CCR1 أي طالما القيمة الحالية للعداد أ صغر من القيمة المخزنة في مـ سجل المقارنة وإلا تكون في حالة صفر منطقي، وأثناء العد التنازلي للعداد تكون القناة CH1 في حالة صفر منطقي (مـ ستوى منخفض inactive) طالما TIMx_CNT>TIMx_CCR1 أي طالما القيمة الحالية للعداد أكبر من القيمة المخزنة في مسجل المقارنة ، وإلا تكون في حالة مستوى مرتفع أو high level

في حالة PWM mode2: أثناء العد التـ صاعدي تكون القناة CH1 في حالة 0 منطقي (inactive) طالما TIMx_CNT<TIMx_CCR1 أي طالما القيمة الحالية للعداد أ صغر من القيمة المخزنة في مـ سجل

المقارنة وإلا تكون في حالة صفر منطقي، وأثناء العد التنازلي للعداد تكون القناة CH1 في حالة واحد منطقي (مـ ستوى مرتفع active) طالما TIMx_CNT>TIMx_CCR1 أي طالما القيمة الحالية للعداد أكبر من القيمة المخزنة في مسجل المقارنة ، وإلا تكون في حالة مستوى منخفض أو low level

باختصار:

فإن شكل إشارة الـ PWM يكون معكوس في الـ mode2

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
- قيمة المقسم الترددي Prescaler
- تردد الساعة الداخلية internal clock
- عدد مرات التكرار

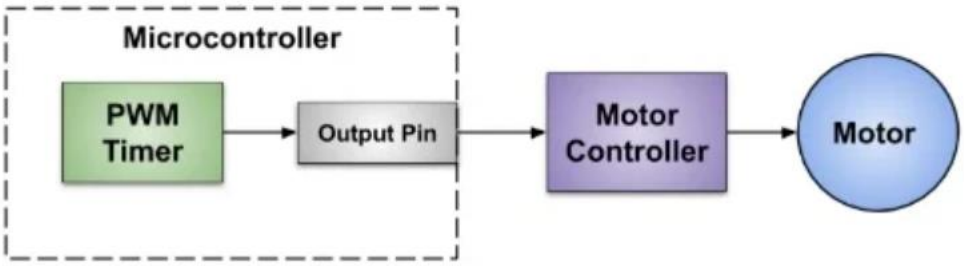
وذلك من خلال العلاقة التالية:

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

مثال:

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

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



الشكل(13):A simplified block diagram of a PWM setup

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:

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

دقة الإشارة الناتجة PWM Resolution:

تعد دقة الدقة إ شارة الـ PWM - من أهم الخ صائص التي تميزها وتمثل عدد الخطوات (الم ستويات الرقمية) التي تحتاجها دورة التشغيل dutyCycle كي تصل إلى القيمة العظمى لها وبالتالي تحدد مقدار النعومة في الانتقال بين قيم دورة التشغيل للو صول إلى القيمة المرغوبة ، وهو أمر مهم جداً خ صو صاً في بعض التطبيقات كالتحكم بالمحركات وأيضاً أنظمة التحكم بالإضاءة.

يمكن حساب دقة إشارة الـ PWM من أجل تردد معين من خلال العلاقة التالية:

$$Resolution_{PWM} = \frac{\log\left(\frac{F_{clk}}{F_{PWM}}\right)}{\log(2)}[Bits]$$

أو من خلال العلاقة التالية:

$$Resolution_{PWM}[Bits] = \frac{\log(ARR+1)}{\log(2)}$$

يوضح الجدول التالي بعض الأمثلة لترددات F_{PWM} مع القيم المقابلة لكل تردد من الدقة :

PWM frequency versus PWM Resolution

STM32 16-bit timer	PWM resolution	PWM frequency
72 MHz	16 bit	~1.1 kHz
72 MHz	14 bit	~4.4 kHz
72 MHz	12 bit	~17.5 kHz
72 MHz	10 bit	~70 kHz
72 MHz	8 bit	~281 kHz
72 MHz	6 bit	~1.125 MHz
72 MHz	4 bit	~4.5 MHz

الجدول(1): ترددات إشارة PWM مقابل الدقة

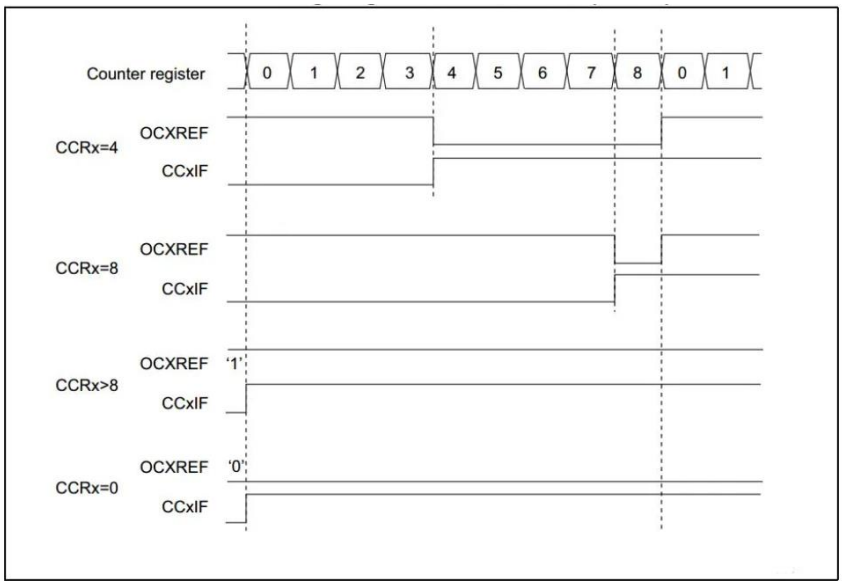
أنماط الـ PWM المختلفة في متحكمات STM32:

يمكن توليد نبضات الـ PWM بعدة أنماط تقسم إلى ثلاث مجموعات أساسية هي:

1. Basic PWM:

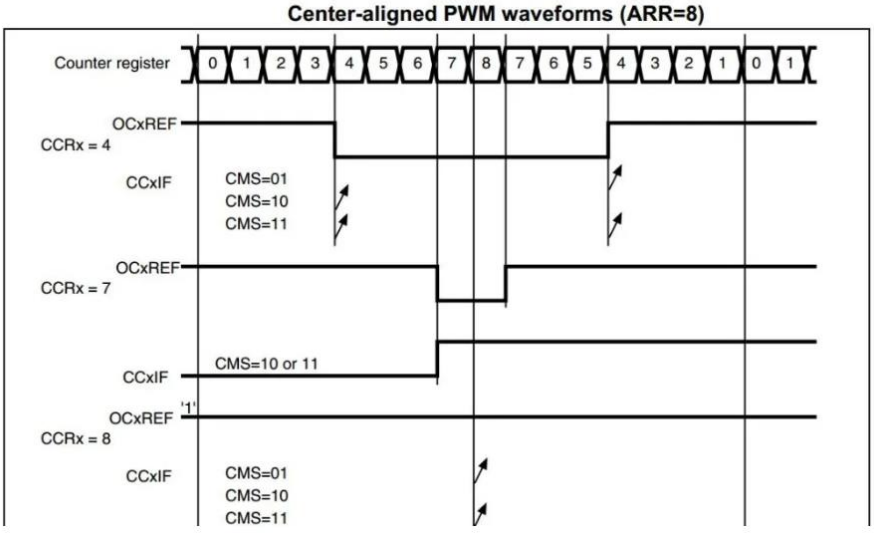
- **Edge-aligned mode**: في هذا النمط يتم التحكم بدورة التشغيل dutycycle لإشارة الـ PWM - من خلال م سجل المقارنة TIMx_CCRx ، والتحكم بالتردد من خلال م سجل auto-reload

register(TIMx_ARR)، وفي هذه الحالة فإن العداد يقوم بالعد بـ شكل تـ صاعدي فقط أو تنازلي فقط، وبإمكان المؤقت الواحد أن يولد حتى الـ 6 إشارات PWM (حسب خصائص المتحكم) لها نفس التردد ولكن بدورات تـ شغيل مختلفة، وهذه الإ شارات جميعها متزامنة باعتبار أن الجبهة الهابطة هي نف سها لجميع الإشارات.



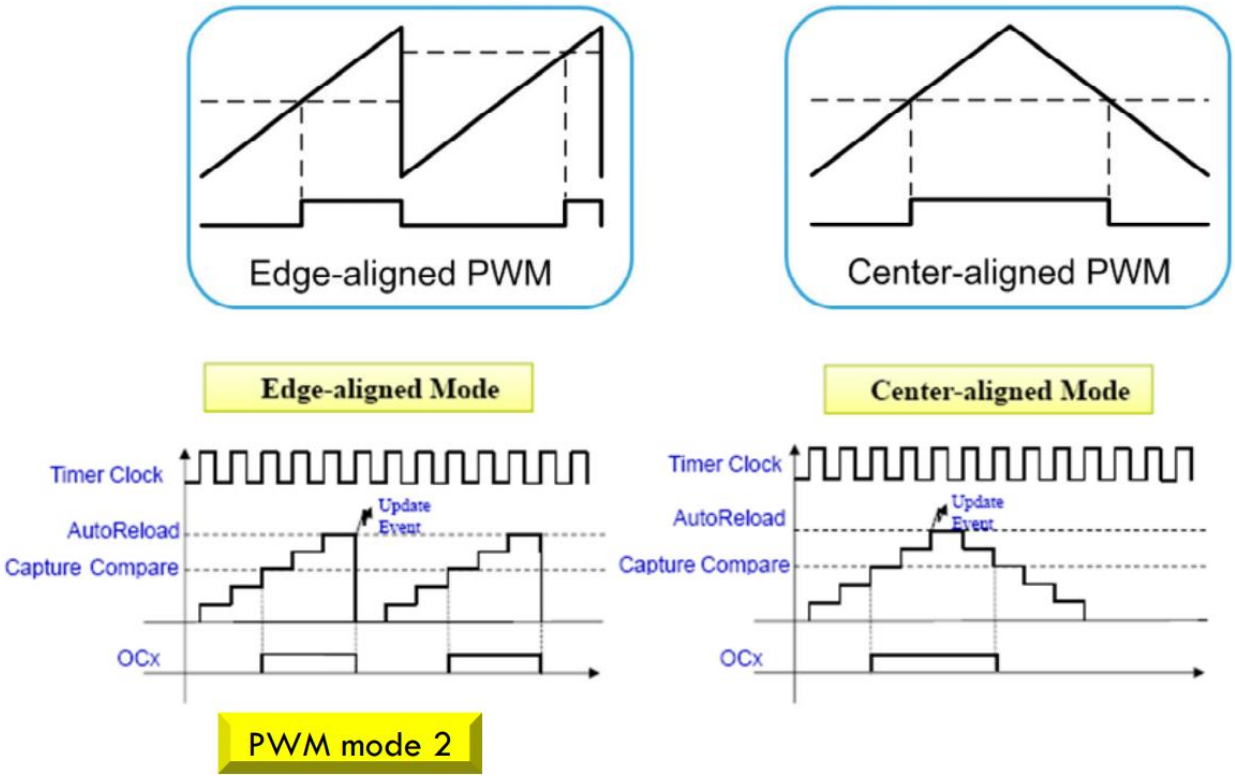
الشكل(14): Edge-Aligned PWM waveform(ARR=8)

- **Center-aligned mode** : في هذا النمط يتم رفع علم المقارنة خلال العد التـ صاعدي أو التنازلي أو عند كليهما تبعاً لإعدادات البتات CMS bits ، ويتم عمل update لبت الاتجاه DIR في مـ سجل TIMx_CR1 من خلال الـ Hardware - ولا يجوز تغيير قيمته من خلال الكود Software، في المخطط الزمني التالي سنعرض عدة إ شارات PWM في نمط Center-aligned mode و سنفترض أن القيمة المخزنة في المسجل TIMx_ARR هي 8 ، وأن PWM mode1 :



الشكل(15):Center-aligned PWM waveform

تكون إشارات الـ PWM الناتجة من مؤقت واحد غير متزامنة لأن الجبهة الهابطة لكل منها مختلفة، لذا فإن أزمنة التبديل لكل إ شارة PWM تكون مختلفة عن الإ شارة الأخرى، ويمكن العمل بهذا النمط من خلال ضبط العداد كعداد ت صاعدي/تنازلي، ويفيد هذا النمط من الـ PWM في تخفيف الـ ضجيج الناتج عن الـ Switching عندما يتم توليد إشارات الـ PWM من نفس المؤقت.

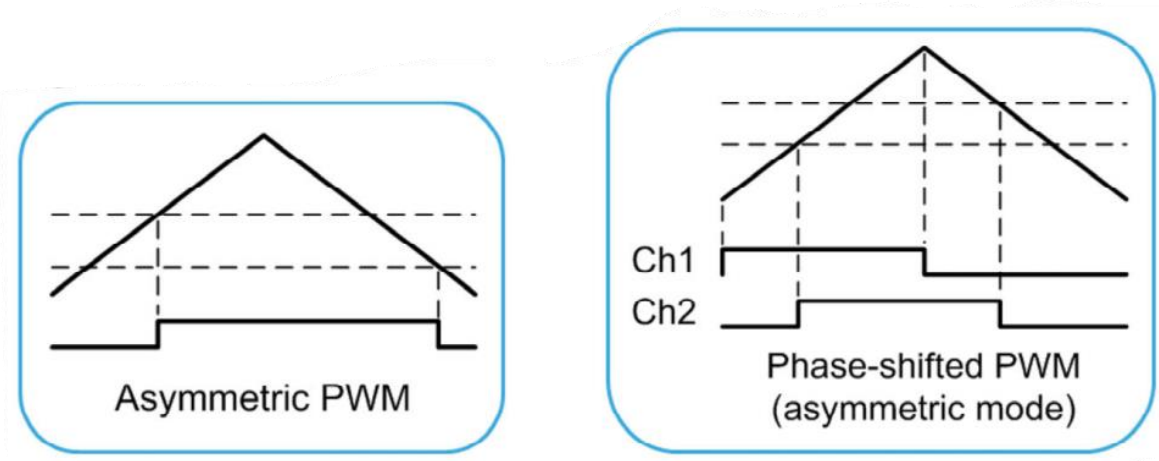


الشكل(16):Basic PWM

2. Asymmetric center-aligned PWM:

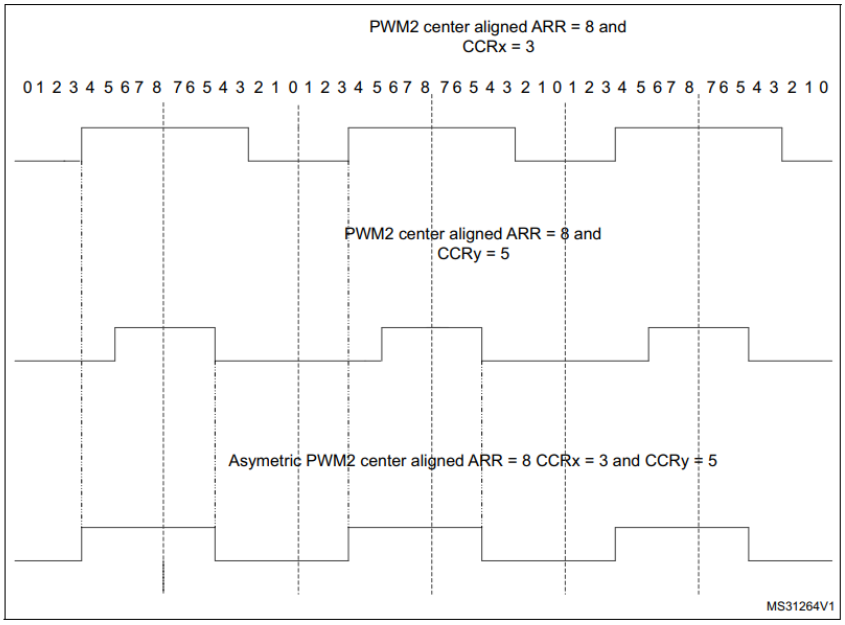
- Asymmetric mode
- Phase-shifted mode

في هذين النمطين يتم استخدام مسجلي مقارنة TIMx_CCR1 وTIMx_CCR2، أو TIMx_CCR3 و TIMx_CCR4 ويتم ت حد يد التردد من خلال الم سـ جل TIMx_ARR ، لذا يمكن اخت يار نمط الـ Asymmetric mode على قناتين من قنوات المؤقت من خلال اختيار القيم المنا سبة للبتات OCxM الموجودة في المسجل TIMx_CCMRx.



الشكل(17):Asymmetric center-aligned PWM

يبين المخطط التالي الفرق بين نمط Asymmetric PWM و نمط Center Aligned PWM

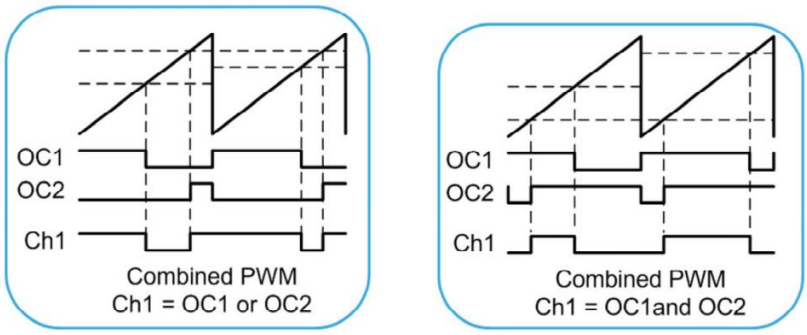


الشكل(18):الفرق بين نمط Asymmetric PWM و نمط Center Aligned PWM

3. Combined PWM:

- Combines two channels with OR/AND function:

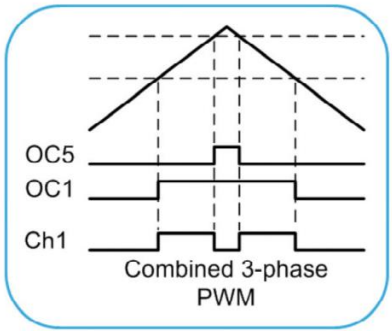
يسمح هذا النمط بالدمج المنطقي (OR/AND) لإشارتي PWM (على سبيل المثال القناة الأولى والثانية أو القناة الثالثة والرابعة) بهدف الحصول على إشارة PWM لها complex waveform ، مما يتيح لك الحصول على إشارتي PWM بأي عرض نبضة وبأي phase relationship value



الشكل(19):Combined PWM

4. Combined 3-phase mode PWM:

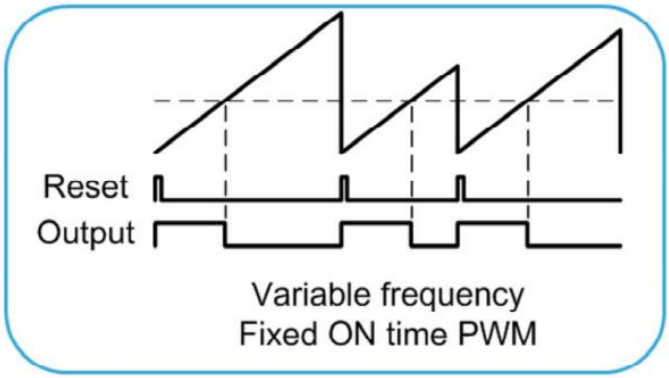
- **Combines two channels:** يستهدف هذا النمط من PWM تطبيقات التحكم بالمحرك ثلاثي الطور، حيث في هذه الحالة يتم دمج القناة الخامسة channel 5 للمؤقت مع أي قناة من القنوات الثلاث الأخرى (1 أو 2 أو 3) لإدخال low state في منتصف إشارة الـ centered-pattern PWM



الشكل(20):Combined 3-phase mode PWM

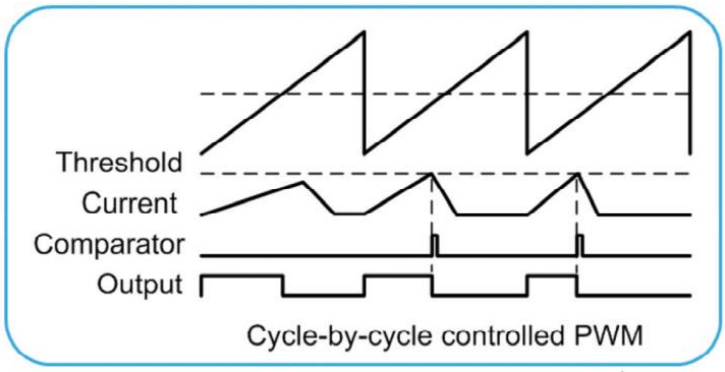
5. Advanced PWM mode:

- **Variable-frequency PWM:** في هذا النمط من الـ PWM يمكن التحكم بدورة التشغيل أو بتردد إشارة الـ PWM من خلال تطبيق إشارة تصفير خارجية external reset على القطب ETR أو على القناة الأولى CH1 أو على مدخلين للمؤقت، الهدف من هذا النمط هو الحصول على إشارة PWM بزمن تشغيل ton محدد أو بزمن toff محدد وثابت وبتردد يتم التحكم به من خلال الـ Hardware ، يستخدم هذا النمط في العديد من التطبيقات منها تطبيقات التحكم بمعامل الاستطاعة (Power Factor Controller) PFC ، أيضاً في تطبيقات mains-supplied applications وتطبيقات currentcontrolled digital LED lighting



الشكل(21):Variable-frequency PWM

- **Cycle-by cycle controlled duty cycle** : في هذا النمط يتم التحكم بدورة التشغيل dutycycle من خلال الـ Hardware ـ، إما عن طريق مقارن مدمج على الشريحة on-chip comparator أو عن طريق off-chip signal، حيث يتم ضبط إشارة الـ PWM ـ بتردد ثابت fixed frequency وبقيمة عظمى لدورة التشغيل maximum duty cycle يتم ضبطها من خلال مسجل المقارنة ، بينما يتم التحكم بالقيمة الحالية لدورة التشغيل cycle-by-cycle أي مع كل دور من إشارة الـ PWM ـ ، يستخدم هذا النمط من الـ PWM مع التطبيقات التي تحتاج إلى التحكم بالتيار خصوصاً محركات التيار المستمر و solenoids ، ففي هذه الحالة يقوم المقارن بمراقبة القيمة الحالية للتيار peak current value المار في الحمل ، وبمجرد تجاوز التيار للعتبة المسموح بها والتي تمت برمجتها وتحديدها مسبقاً ، فإن المقارن يقوم بتصفير إشارة الـ PWM.



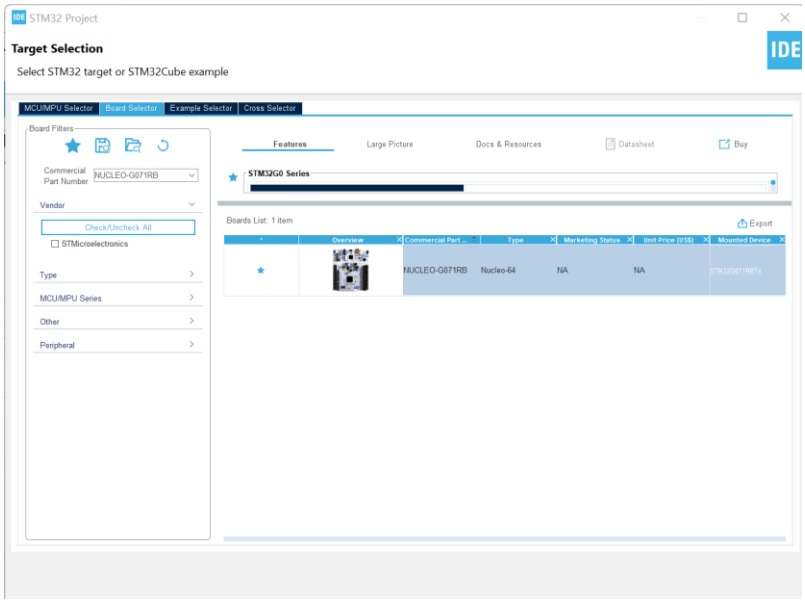
الشكل(22):Cycle-by cycle controlled duty cycle

4. التطبيق العملي الرابع: استخدام المؤقت في نمط PWM mode واستخدامه للتحكم في شدة إضاءة الليد:

سنتبع في هذا التطبيق الخطوات التالية للتحكم بشدة إضاءة الليد:

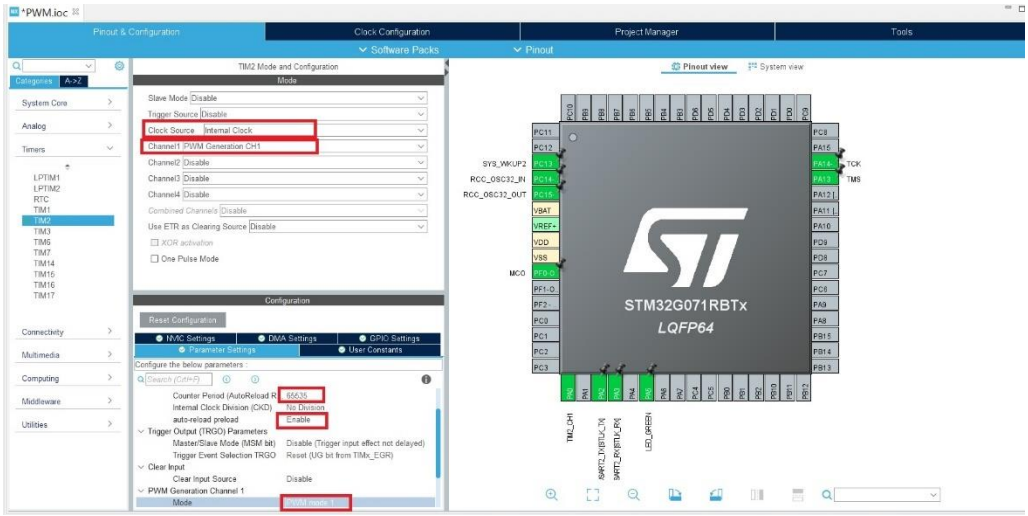
- ضبط بارامترات المؤقت TIM2 ليعمل في نمط الـ PWM ـ وباستخدام الساعة الداخلية للمتحكم internal clock ، ثم تفعيل القناة الأولى CH1 لاستخدامها كقناة الخرج لإشارة الـ PWM
- ضبط قيمة المسجل ARR على القيمة العظمى وهي 65535 ، فيصبح التردد 488.25HZ
- التحكم بدورة التشغيل dutycycle من خلال كتابة القيمة المناسبة على المسجل CCR1
- 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: قم بفتح برنامج STM32CubeIDE ومن ثم قم بإنـ شاء مـ شروع جديد من نافذة File ثم New ثم STM32Project ثم قم باختيار المتحكم المـ صغر أو من خلال اختيار ا سم اللوحة المـ ستخدمة وهي في حالتنا Nucleo-G071RB كما في الشكل التالي:



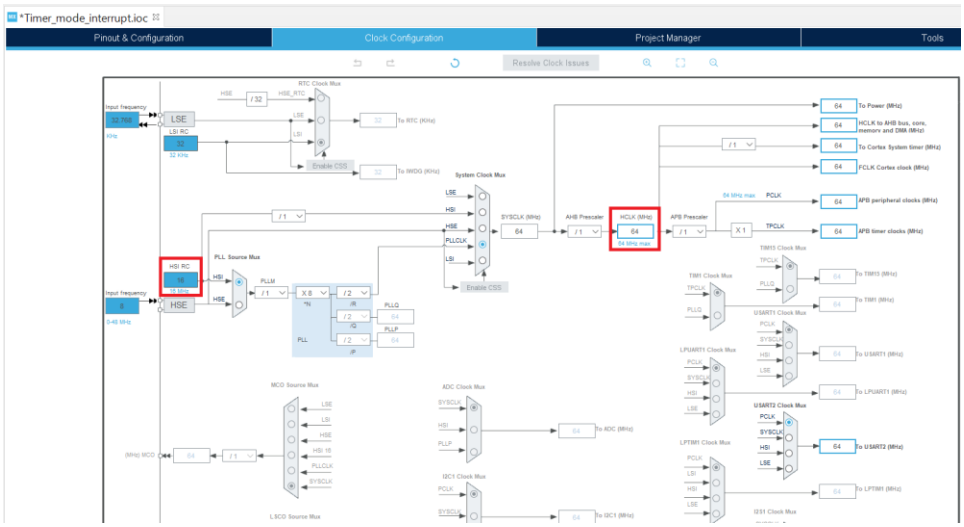
الشكل(23) :بدء مشروع جديد في بيئة STM32CubeIDE

الخطوة الثانية: 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 إشارة الـ PWM عليها، نضبط القيمة العظمى للمسجل ARR على القيمة 65535 ليـ صبح تردد إ شارة PWM هوZ488.25H ، نفعل خا صية Auto Reload preload ونختار نمط إشارة الـ PWM



الشكل(24):

الخطوة الثالثة: ضبط تردد ساعة المتحكم



الشكل(25):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);
```

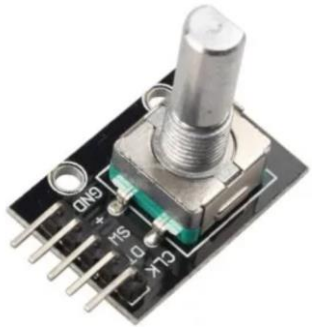
يصبح الكود النهائي:

```
#include "main.h"

TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_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_1);
 //**************************
```

```
    while (1)
    {
        while(CH1_DC < 65535)
        {
            TIM2->CCR1 = CH1_DC;
            CH1_DC += 70;
            HAL_Delay(1);
        }
        while(CH1_DC > 0)
        {
            TIM2->CCR1 = CH1_DC;
            CH1_DC -= 70;
            HAL_Delay(1);
        }
    }
}
```

5. استخدام المؤقت في نمط Encoder mode:

تستخدم المشفرات Encoders في العديد من التطبيقات على سبيل المثال التطبيقات التي تحتاج إلى تحسس مو ضع و سرعة المحرك، وتطبيقات قياس الطول والمسافة وغيرها من التطبيقات ، كما يمكن ا ستخدام المشفر encoder كجهاز إدخال يمكن تدويره بدون وجود limitation على عكس المقاومات المتغيرة ذات الذراع وغيرها، كما يمكنك الانكودر من معرفة مقدار تسارع المستخدم في تدويره.



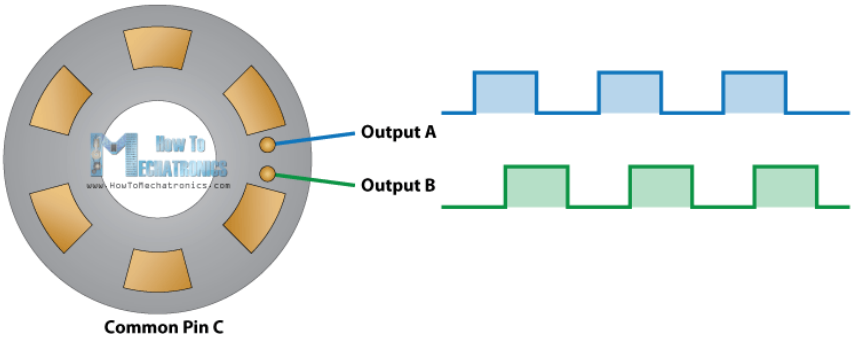
الشكل(26):

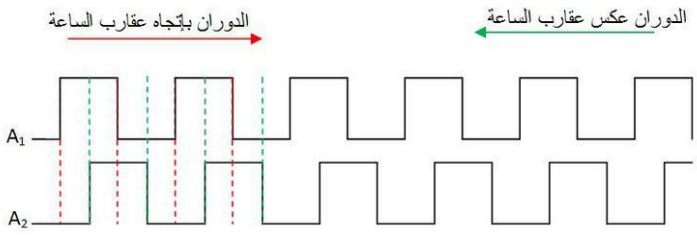
الانكودر عبارة عن قرص له محور مشترك مع المحرك، هذا القرص مثقب و له وجهان, عند الوجه الاول يوجد منبع ضوئي و عند الوجه الثاني يوجد حساس ضوئي، فعندما يدور المحرك يدور الانكودر بنفس سرعه المحرك ويمر الضوء عبر الثقوب ويستقبل الحساس الضوئي الضوء عند مرور الضوء عبر الثقوب و بالتالي نحصل على نبضة في كل ثقب يمر من خلاله الضوء, فاذا فرضنا ان القرص يحتوي على 365 ثقب عندها نحصل على 365 نبضة في كل دورة.

هناك نوعان رئيسيان للانكودر وهما:

1- الانكودر التزايدي Incermental Encoder

يحتوي القرص في هذا الانكودر على مجموعة من الثقوب أو ثقب واحد فقط





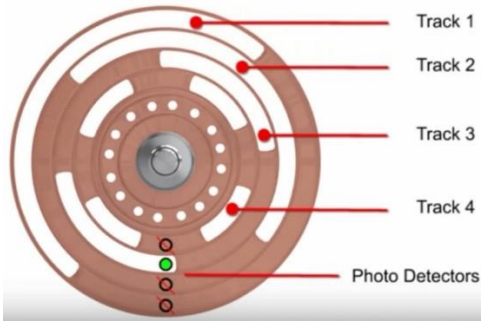
الشكل(27) الانكودر التزايدي incremental encoder

كما تلاحظ في المخطط أعلاه للموجات الخارجة من الانكودر فإن دوران الانكودر باتجاه عقارب الـ ساعة سيتسبب بتغير قيمة الفولتية على A1قبل تغيرها على A2 والعكس بالعكس. لذلك يجب معرفة أي من نفاط الارتباط تتغير قيمته أولاً لمعرفة اتجاه الدوران.

2- الانكودر المطلق Absolute Encoder

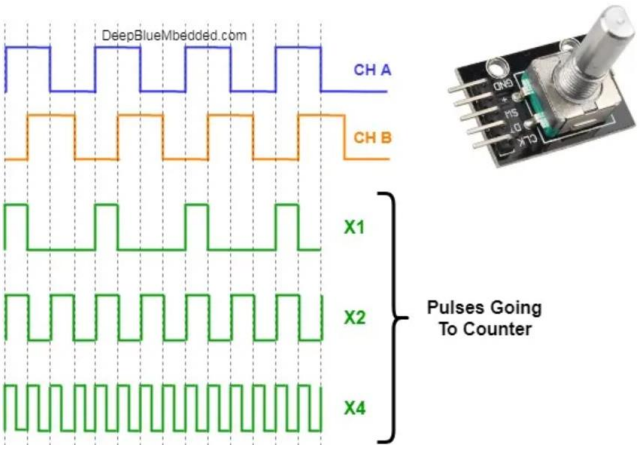
في هذا الانكودر يحتوي القرص الدوار على ثقوب في اربع دوائر متحدة المركز وأربع حـ سا سات لكـ شف النبـ ضات الـ ضوئية، وتكون الثقوب مرتبة بطريقة بحيث يكون الخرج المتتابع من الحـ ساس عبارة عن عدد بشفرة ثنائية، وكل عدد من هذه الأعداد يقابل وضع زاوي محدد.

بـا ستخدام أربع مـ سارات سيكون هناك أربع خانات في خرج الانكودر تمثل مو ضع أو سرعة المحرك المربوط معه ، وبالتالي سيكون لدينا عدد موا ضع هو 2 مرفوع للأس 4 أي 16 مو ضع، بالتالي تكون أقل زاوية يمكن قيا سها لدوران المحرك هي 360/16 أي 22.5 درجة (وهذا غير عملي لكون الدقة منخف ضة جداً)، حيث تحتوي المـ شفرات العملية على مابين 10 و12 مـ سار، حيث عدد الخانات للـ شفرة الناتجة عن الانكودر تساوي عدد المسارات. فـ ـ باستعمال 10 مسارات يكون هناك 10 بت وعدد المواضع التي يمكن كشفها هو 2 أس 10 أي 1024 وتكون الدقة 360/1024 أي 0.35 درجة.



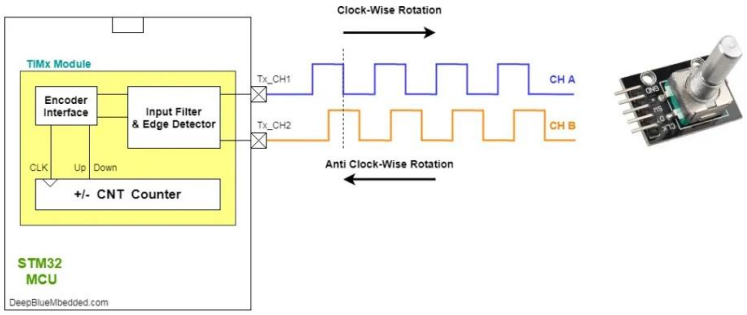
الشكل(28):

الانكودر الذي سنستخدمه في هذه الجلسة من نوع incremental encoder ويدعى أيضاً Quadrature Encoder ، وله 2 out-of-phase output channels ، كل قناة تعطي عدد محدد من النبضات مع كل دورة للانكودر ، كما يتم تحديد اتجاه الدوران من خلال العلاقة بين القناتين.



الشكل(29):

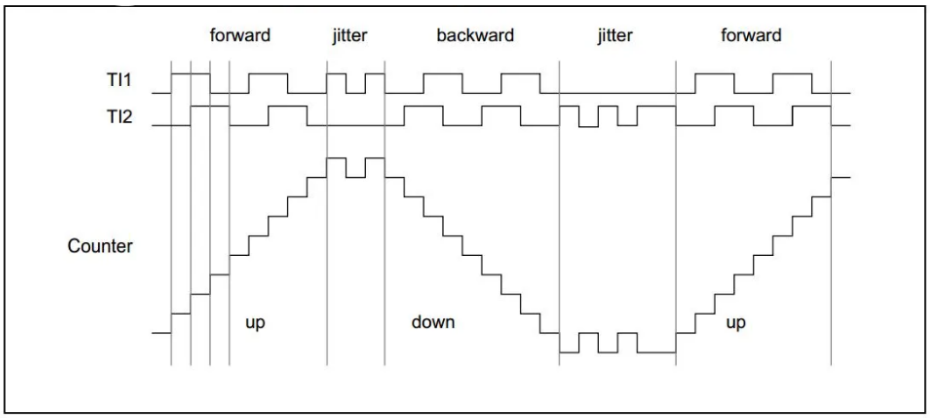
تستخدم القناتين GH1 و CH2 من المؤقت للاتصال مع الانكودر، حيث يؤدي تتابع النبضات القادمة على القناتين إلى العد بشكل تصاعدي أو تنازلي حسب اتجاه العد، ويتم تغيير قيمة بت الاتجاه DIR bit في المسجل TIMx_CR1 من خلال الـ Hardware - ، ويقوم العداد بعد النبضات القادمة على القناة TI1 أو القناة TI2 أو القناتين معاً.



الشكل(30):

يعمل المؤقت في نمط الـ Encoder interface كعداد لنبضات ساعة خارجية external clock مع إمكانية اختيار اتجاه العد ، أي أن العداد يبدأ بالعد من الصفر حتى القيمة المخزنة في المسجل TIMx_ARR.

أي أن العداد يقوم بملاحقة السرعة واتجاه الدوران للانكودر المتصل مع قناتي المؤقت، حيث لا يحتاج الانكودر لأي دارة ملاءمة بينه وبين المتحكم.



الشكل(31):مثال عن عمل المؤقت في نمط encoder interface

6. التطبيق العملي الخامس: استخدام المؤقت في نمط Encoder interface mode وعرض قيمة الانكودر على المنفذ التسلسلي:

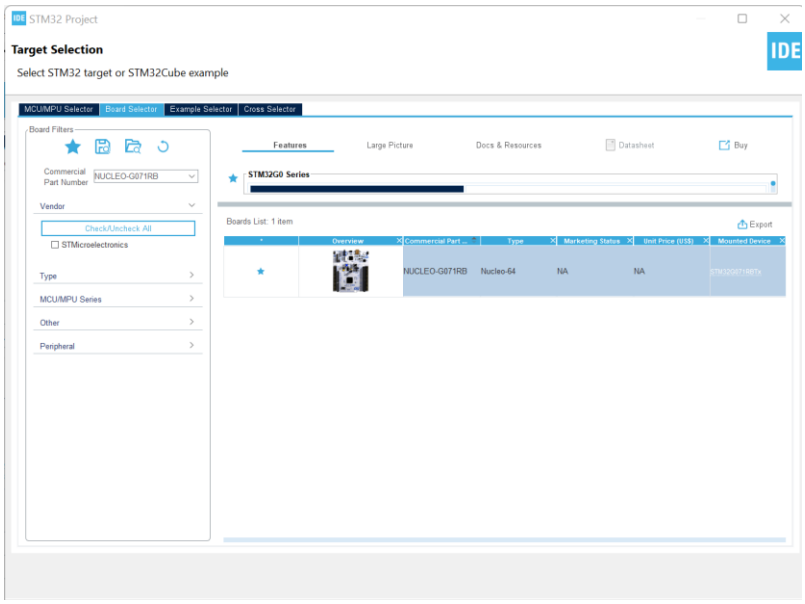
الغاية من التطبيق هي بناء نظام من خلال ضبط المؤقت TIM2 ليعمل بنمط الـ Encoder mode ثم قراءة قيمة العداد باستمرار وإرسالها إلى الحاسب من خلال المنفذ التسلسلي عند ضغط المفتاح الموجود على موديول الـ Encoder.

سنتبع في هذا التطبيق الخطوات التالية للتحكم بشدة إضاءة الليد:

- ضبط إعدادات المؤقت TIM2 للعمل في نمط encoder mode وباستخدام قناتي دخل (combined)
- ضبط إعدادات قطب من أقطاب الـ GPIO كقطب دخل لوصله مع المفتاح الموجود على موديول الانكودر.
- ضبط إعدادات المنفذ التسلسلي UART2 للعمل في نمط الـ async وبمعدل نقل بيانات 9600bps .
- قراءة قيم مسجل القيمة الحالية للمؤقت وعرضها بشكل دوري على المنفذ التسلسلي مع كل تغيير لحالة المفتاح

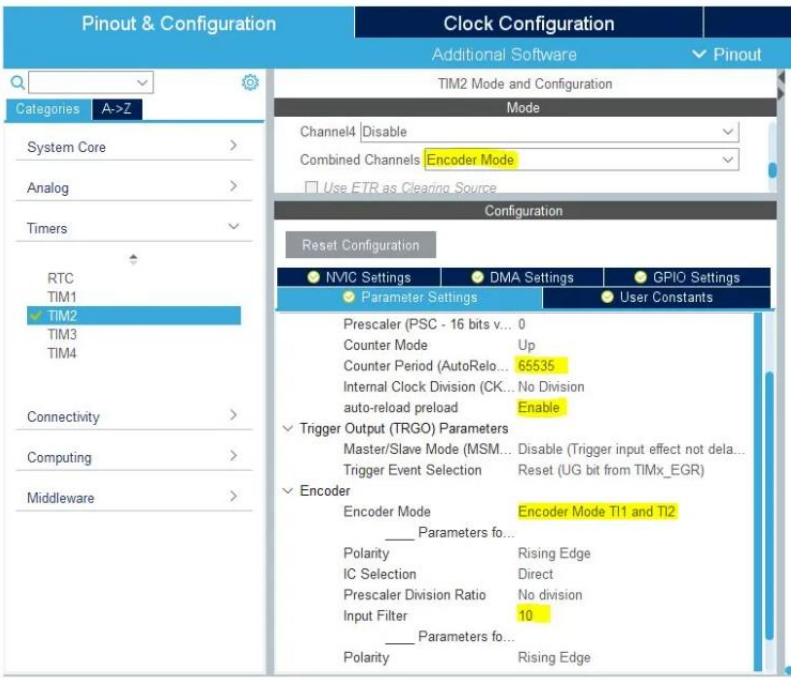
سنقوم بضبط الإعدادات من خلال أداة CubeMx المدمجة داخل بيئة STM32CubeIDE وفقاً للخطوات التالية:

الخطوة الأولى: قم بفتح برنامج STM32CubeIDE ومن ثم قم بإنشاء مشروع جديد من نافذة File ثم New ثم STM32Project ثم قم باختيار المتحكم المصغر أو من خلال اختيار اسم اللوحة المستخدمة وهي في حالتنا Nucleo-G071RB كما في الشكل التالي:



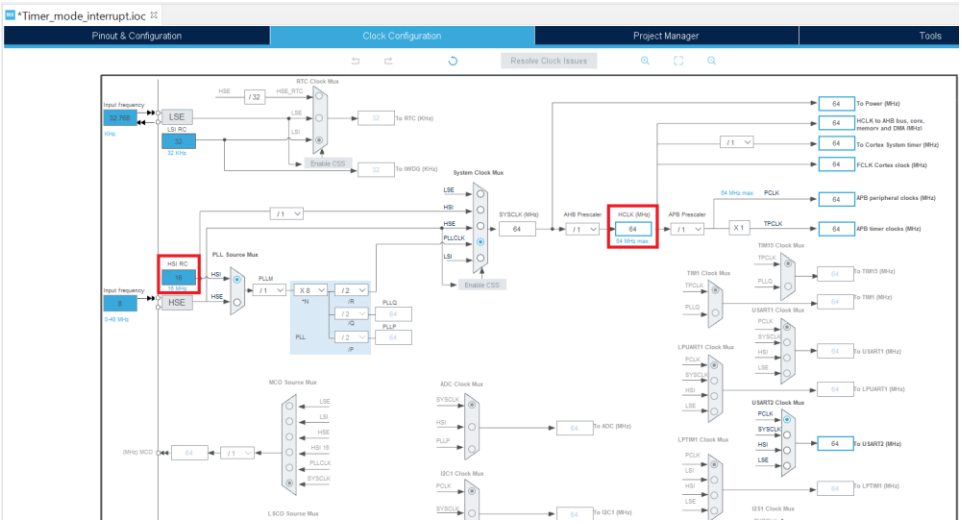
الشكل(32) :بدء مشروع جديد في بيئة STM32CubeIDE

الخطوة الثانية: ضبط إعدادات المؤقت ليعمل في نمط Encoder mode



الشكل(33):

الخطوة الثالثة: ضبط تردد ساعة المتحكم



الشكل(34):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 Encoder mode:

```
HAL_TIM_Encoder_Start(&htim2, TIM_CHANNEL_ALL);
```

يصبح الكود النهائي:

```
#include "main.h"

TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_USART2_UART_Init(void);

int main(void)
{
  uint8_t MSG[50] = {'\0'};

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

```
  MX_USART2_UART_Init();
  HAL_TIM_Encoder_Start(&htim2, TIM_CHANNEL_ALL);

  while (1)
  {
    if(HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_2))
    {
      sprintf(MSG, "Encoder Switch Released, Encoder Ticks = %d\n\r", ((TIM2->CNT)>>2));
      HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
    }
    else
    {
      sprintf(MSG, "Encoder Switch Pressed,  Encoder Ticks = %d\n\r", ((TIM2->CNT)>>2));
      HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
    }
    HAL_Delay(100);
  }
}
```

يمكن تطوير التطبيق الـ سابق لجعل الانكودر يتحكم بـ شدة إ ضاءة ليد مو صول على القناة الخا صة بالمؤقت TIM3 ويعمل بنمط PWM ، من خلال تمرير قيمة العداد في الـ TIM2 والذي يعبر عن موضع الانكودر إلى دورة التشغيل للمؤقت TIM3.