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

مقرر المتحكمات المصغرة

الجلسة الخامسة

السنة الرابعة ميكاترونيك

2023/2202

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
4- التعرف على وحدة توليد الزمن الحقيقي الـ RTC
5- التطبيق3: استخدام وحدة الـــ RTC لإظهار التاريخ و الوقت على شاشة lcd و ضبط المنبه على وقت محدد لتشغيل ليد

1- أنماط العمل المختلفة للمؤقتات في متحكمات STM32:

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

- **نمط المؤقت Timer Mode**
- **نمط تعديل عرض النبضة PWM Mode**

1. نمط المؤقت Timer Mode:

في هذا النمط من العمل فإن المؤقت يحصل على نبضات الساعة من نبضات الساعة الخاصة بالمتحكم 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.

ضبط إعدادات المؤقتات في متحكمات STM32 :

كما ذكرنا سابقاً فإن المؤقت عبارة عن عداد بإمكانه العد بشكل تصاعدي ، حيث يقوم بالعد من الصفر إلى القيمة المحددة في حقل الـــ $(Preload)$ 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 عمل المؤقت (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 الـــ Prescaler بطول 16بت)، وعندما يصل العداد إلى القيمة المحددة Preload تحدث مقاطعة أي يتم تصفير مسجل القيمة الحالية للمؤقت ويتم العد مرة أخرى من الصفر و يتم رفع العلم الخاص بالمقاطعة Update Event(UEV).

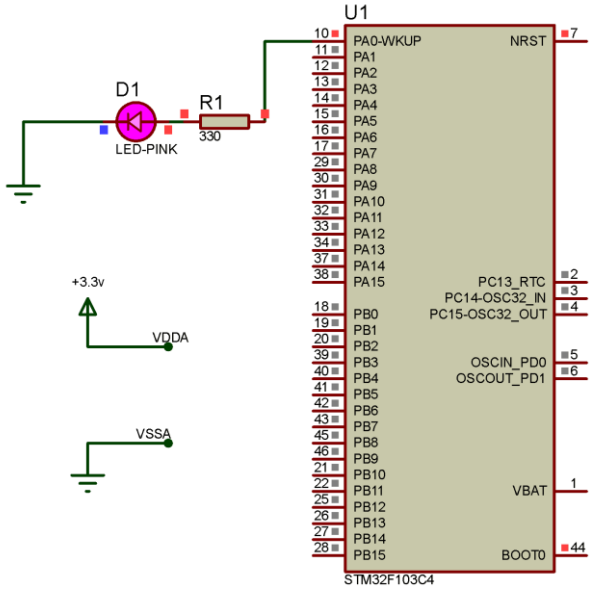
أي أن المسجلات الخاصة بالـــ $(Preload)$ وPrescaler هي التي تحدد تردد المؤقت أي الزمن الذي سيستغرقه المؤقت حتى يحدث تحدث المقاطعة وعندها يتم رفع العلم UEV، ويتم اختيار القيم المناسبة لهذه المسجلات بناءً على هذه المعادلة:

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

على سبيل المثال،إذا أردنا أن نحصل على زمن 0.5sec ، وكان تردد الساعة للمتحكم مضبوط على 48MHz عندها سنضبط الـ Prescaler على 48000 والـ period على 500 وفق العلاقة التالية:

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

2- 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

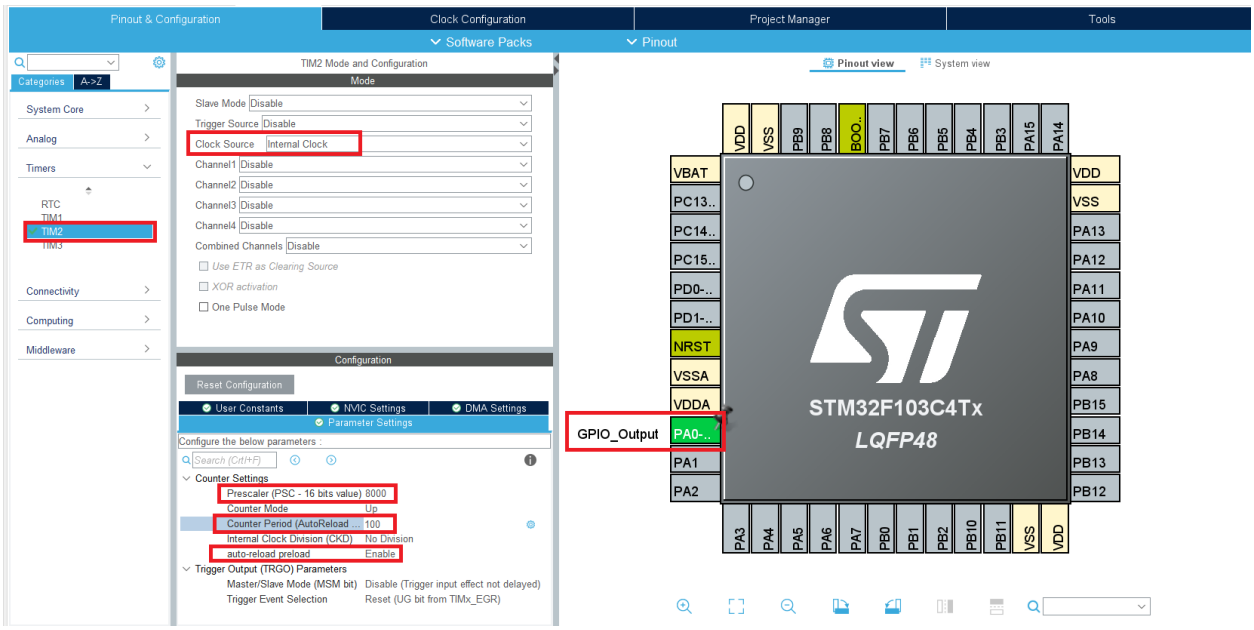
الخطوة الرابعة: ضبط إعدادات المؤقت

كي نحصـــل على زمن 100msec لعكس حالة الليد الموصـــول على القطب رقم 5 من المنفذ A، من المعادلة السـابقة سـنفترض أن تردد سـاعة المتحكم هي 8MHz والمقسـم الترددي 8000 بقي فقط حسـاب(Preload) ، بتعويض القيم في المعادلة:

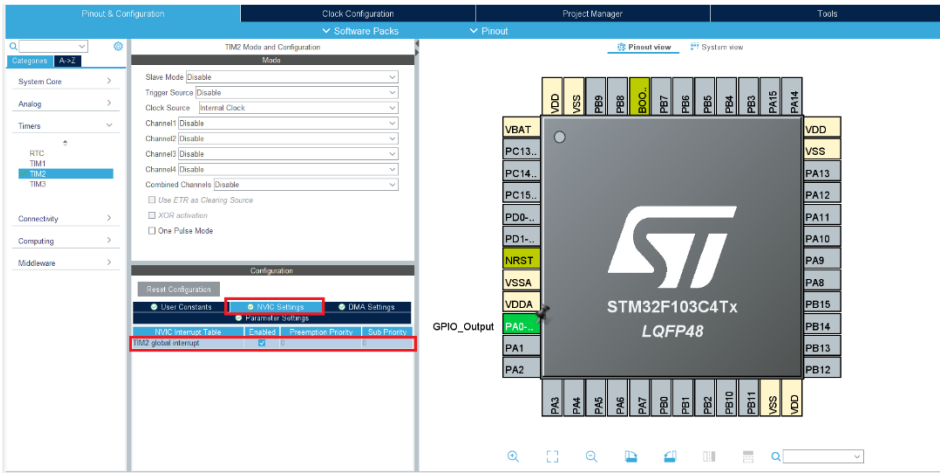
$$T_{out} = \frac{Prescaler \times Preload}{F_{CLK}} = \frac{8000 \times Preload}{8000000}$$

$$Preload = 100$$

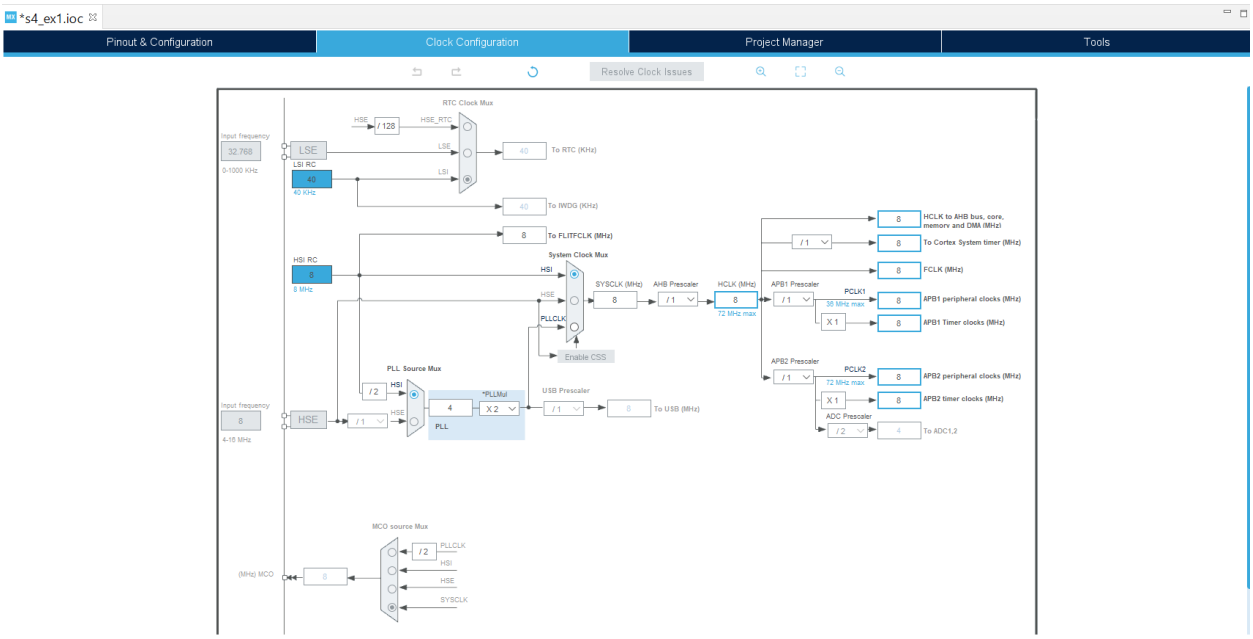
سنقوم باختيار مصدر الساعة للمؤقت داخلي، المقسم الترددي8000، الـ Preload=100، أيضاً سنقوم بتفعيل إعادة التحميل التلقائي، كما في الشكل التالي:



الخطوة الخامسة: تفعيل مقاطعة المؤقت من شريط الـ NVIC tab



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)
  {

  }

}
```

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 :

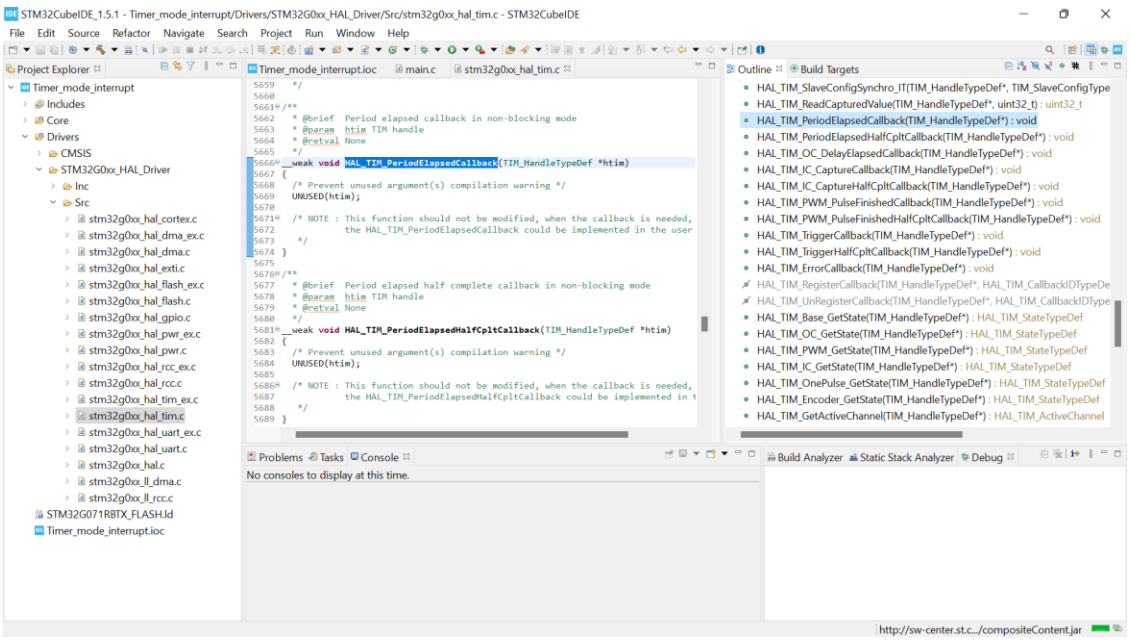
HAL_TIM_Base_Start_IT()

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 ويدعى

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 وتحميله إلى المتحكم ضمن بيئة Proteus

الخطوة الحادية عشر: إعادة الخطوات السابقة ولكن من أجل المتحكم stm32G0B1CE الموجود على البورد ، ثم ترجمة الكود وتحميله للمتحكم

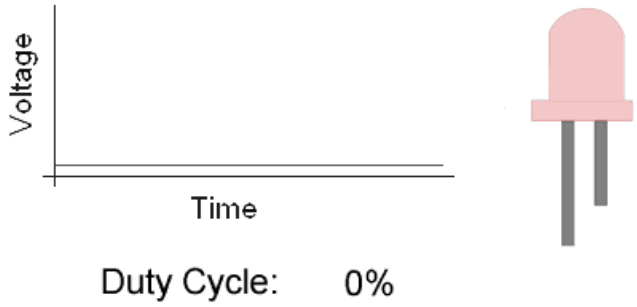
$T_{out} = \frac{Prescaler \times Preload}{F_{CLK}}$			
قيمة العد الأعظمية	preload register	Timer 16 bits	0xffff= 65535
	preload register	Timer 32 bits	0xffff ffff= 4,294,967,295
المقسم الترددي	Prescaler	Timer 16 bits	1~65535
التعليمات اللازم إضافتها بالكود			
تعريف التايمر2 المستخدم		TIM_HandleTypeDef htim2;	
الدالة الخاصة ببدأ عمل مقاطعة التايمر2		HAL_TIM_Base_Start_IT(&htim2)	
برنامج خدمة المقاطعة الخاص بالطفحان		HAL_TIM_PeriodElapsedCallback()	

2. نمط تعديل عرض النبضة PWM Mode:

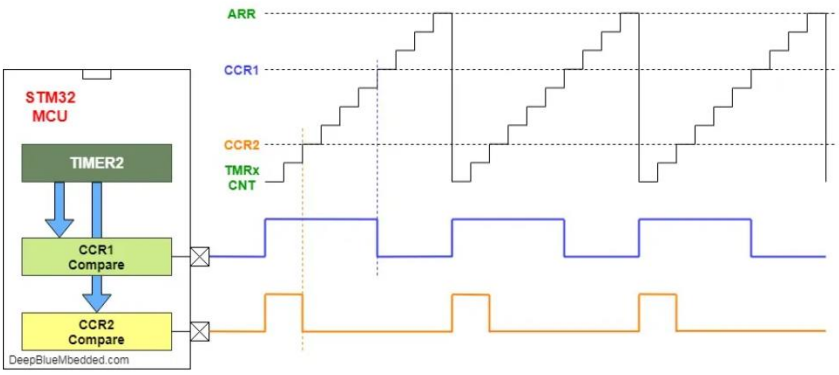
يمكن أن يعمل المؤقت في نمط PWM mode أي يستقبل نبضات الساعة الخاصة به من الساعة الداخلية للمتحكم حيث يبدأ بالعد من الصفر ويزداد مع كل نبضة ساعة للمتحكم (طبعاً مع مراعاة إعدادات المقسم الترددي للمؤقت) ويتم وضع قطب الخرج الخاص بالـ PWM في وضع HIGH ويبقى كذلك إلى أن يصل العداد إلى القيمة المخزنة في المسجل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

يوضح المخطط التالي كيفية تأثير قيمة المسجل ARR في 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:



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 بتردد إشارة الـ PWM من خلال البارامترات التالية:

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

وذلك من خلال العلاقة التالية:

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

مثال:

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

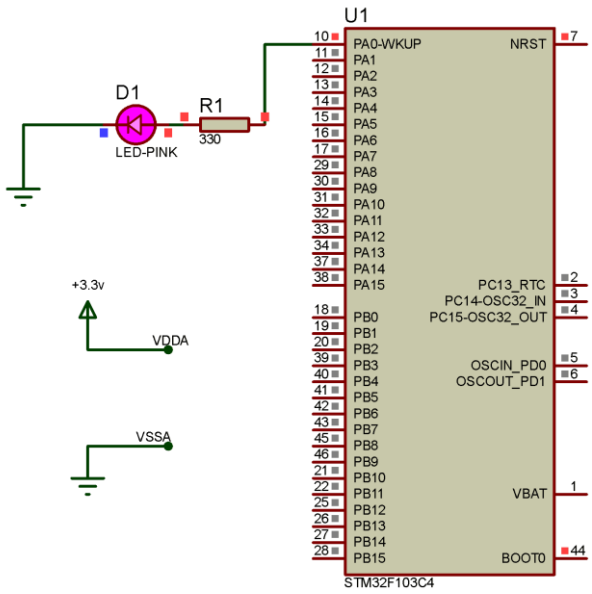
$$F_{PWM} = \frac{72 \times (10\text{^}6)}{(65535 + 1) \times (1 + 1) \times 1} = 549.3HZ$$

دورة التشغيل Duty Cycle:

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

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

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 الـــ PWM وباستخدام الساعة الداخلية للمتحكم internal clock ، ثم تفعيل القناة الأولى CH1 لاستخدامها كقناة الخرج لإشارة الـ PWM
- ضـــبط قيمة المســـجل ARR على القيمة العظمى وهي 65535 ، والمقســـم الترددي prescaler على 1، فيصبح التردد 61HZ من خلال العلاقة:

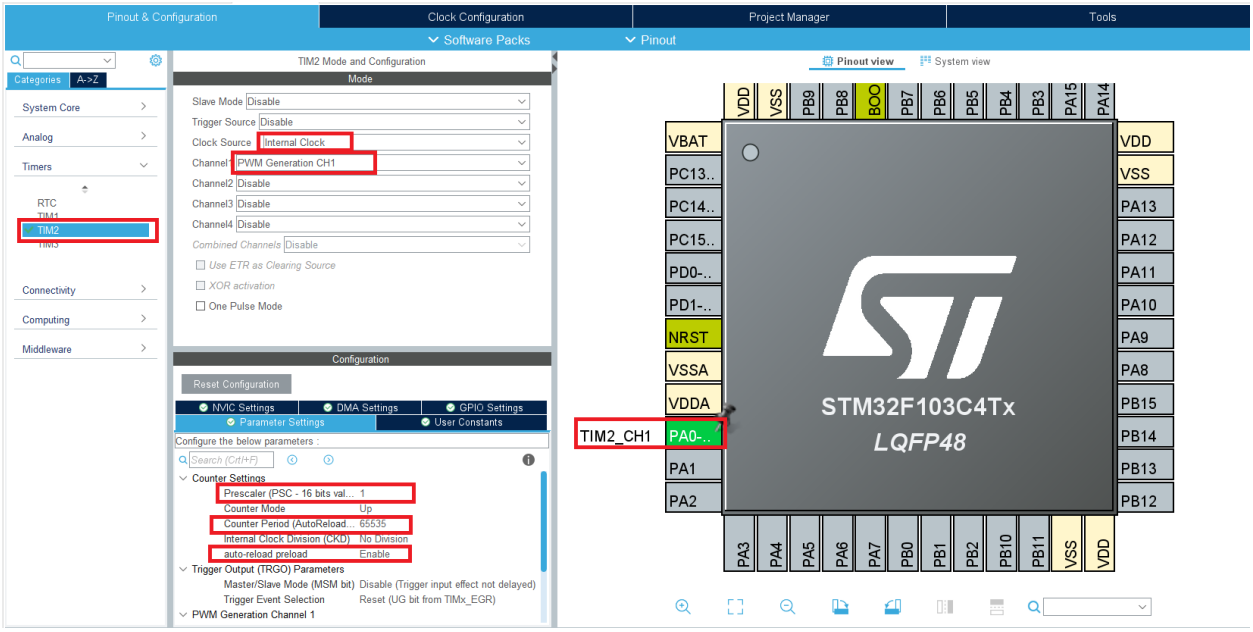
$$F_{PWM} = \frac{8 \times (10\text{^}6)}{(65535 + 1) \times (1 + 1) \times 1} = 61HZ$$

- التحكم بدورة التشغيل dutycycle من خلال كتابة القيمة المناسبة على المسجل CCR1
- جعل دورة التشغيل تتغير من 0% حتى 100% وتعيد الكرة باستمرار

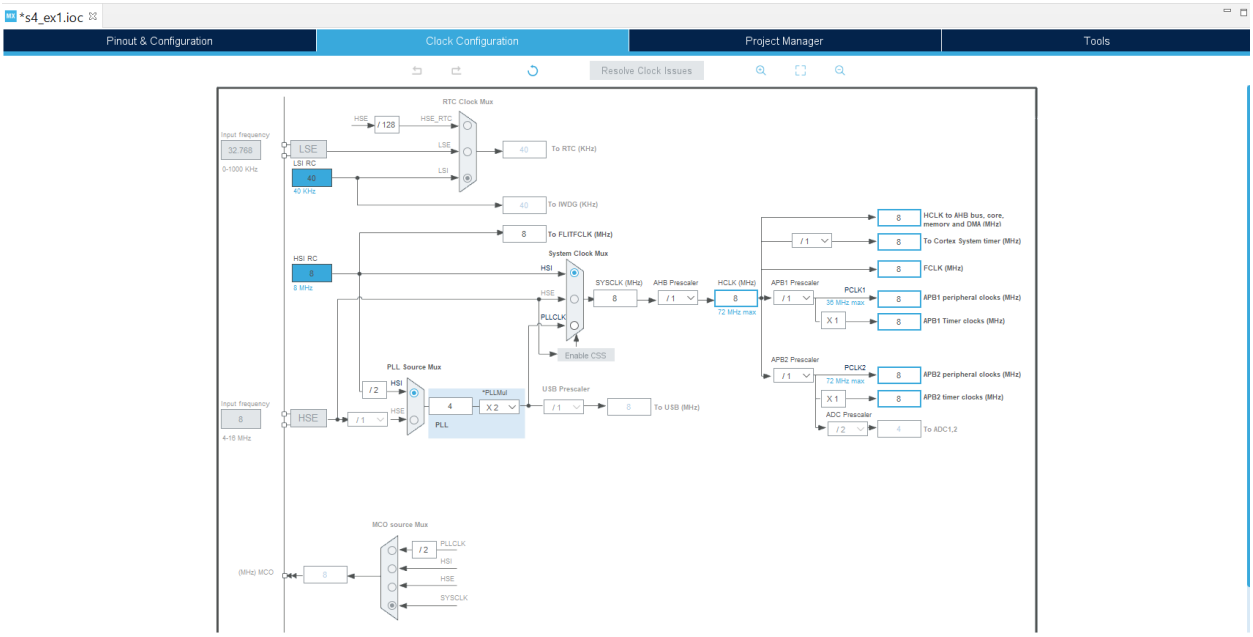
ســـنقوم بضـــبط الإعدادات من خلال أداة CubeMx المدمجة داخل بيئة STM32CubeIDE وفقاً للخطوات التالية:

الخطوة الأولى: فتح بيئة STM32CubeIDE وإنشـاء مشـروع جديد ثم اختيار المتحكم المناسـب واختيار اسـم 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 سـيتم إخراج إشـارة الـــــ PWM عليها، نضبط القيمة العظمى للمسجل ARR على القيمة 65535 والمقسم الترددي prescaler على 1، ليصبح تردد إشارة PWM هو61HZ، 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->CCR1 = CH1_DC;
            CH1_DC += 70;
            HAL_Delay(1);
        }
        while(CH1_DC > 0)
        {
            TIM2->CCR1 = CH1_DC;
            CH1_DC -= 70;
            HAL_Delay(1);
        }
    }
}
```

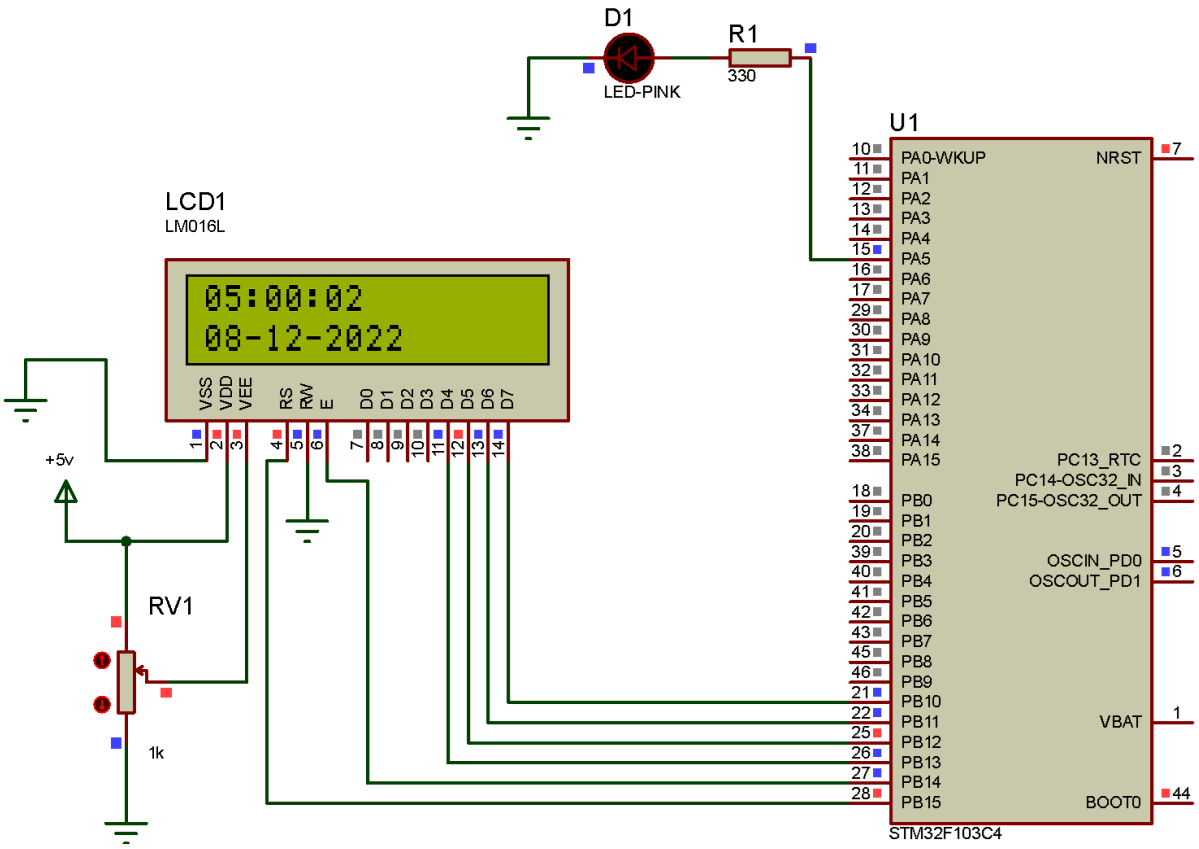
الخطوة السادسة: ترجمة الكود وتحميله إلى المتحكم ضمن بيئة Proteus

الخطوة السابعة: إعادة الخطوات السابقة ولكن من أجل المتحكم stm32G0B1CE الموجود على البورد ، ثم ترجمة الكود وتحميله للمتحكم

3. وحدة توليد الزمن الحقيقي Real Time Clock (RTC)

ســاعة الزمن الحقيقي عبارة عن أداة لحفظ الوقت، تســتخدم مع التطبيقات التي يتم تنفيذها عند أزمنة محددة، كساعة التوقيت الموجودة ضمن الغسالات الآلية ، تطبيق إعطاء الأدوية للمرضى بأزمنة محددة وغيرها...
فهي عبارة عن عداد زمن لكنها تعطي دقة أكبر من المؤقتات الموجودة في المتحكم، فالمؤقتات مناسبة لتوليد الأزمنة المختلفة و إشارات الـ PWM على سبيل المثال..
معظم متحكمات bit8 لاتحتوي على RTC داخلية وإنما يتم استخدام إحدى شرائح الـــ RTC الخارجية كـ DS1302 أو PCF8563 بالإضـــافة إلى بعض العناصـــر الالكترونية اللازمة كي تعمل بشـــكل أمثلي كما تحتاج إلى مســـاحة إضـــافية على الدارة المطبوعة، تحتوي متحكمات stm32 على موديول RTC مدمج بداخل المتحكم وهي لا تحتاج لأية عناصر إضافية أو دارات ملائمة كي تعمل
نبضات الساعة لوحدة الـ RTC (RTCCLK) يمكن أن تكون قادمة من:
High Speed External clock (HSE) :وفي هذه الحالة تكون مقسمة على 32
Low Speed External clock (LSE)
Low Speed Internal Clock (LSI)
لكن عندما يكون المتحكم يعمل في نمط VBAT أو يكون في حالة إيقاف تشـــغيل shutdown ، في هذه الحالة يجب أن يكون RTC clock هي LSE أو LSI
يمكن لوحدة الـ RTC أن تعمل في جميع أنماط الطاقة المنخفضة للمتحكم فعندما يتم تزويدها بنبضات الساعة من خلال Low speed external oscillator (LSE) بتردد 32.768 KHz، عندها ستستمر وحدة الـــ RTCبالعمل حتى عند فصل التغذية الأساسية عن المتحكم ، عندما يكون قطب VBAT موصول ببطارية احتياطية.
لوحدة الـــ RTC مخرجان لهما القدرة على توليد تنبيهان قابلان للبرمجة ولهما القدرة على إيقاظ المعالج من كافة أنماط توفير الطاقة ،كما تحتوي وحدة الـــ RTC على مؤقت مدمج قابل للضبط وإعادة تحميل القيمة آلياً والذي يستخدم لتوليد مقاطعات دورية لها القدرة على إيقاظ المعالج ، كما يمكن ضبط دقة هذا المؤقت

3- التطبيق العملي الثالث: استخدام وحدة الـ RTC لإظهار التاريخ و الوقت على شاشة lcd و ضبط المنبه على وقت محدد لتشغيل ليد

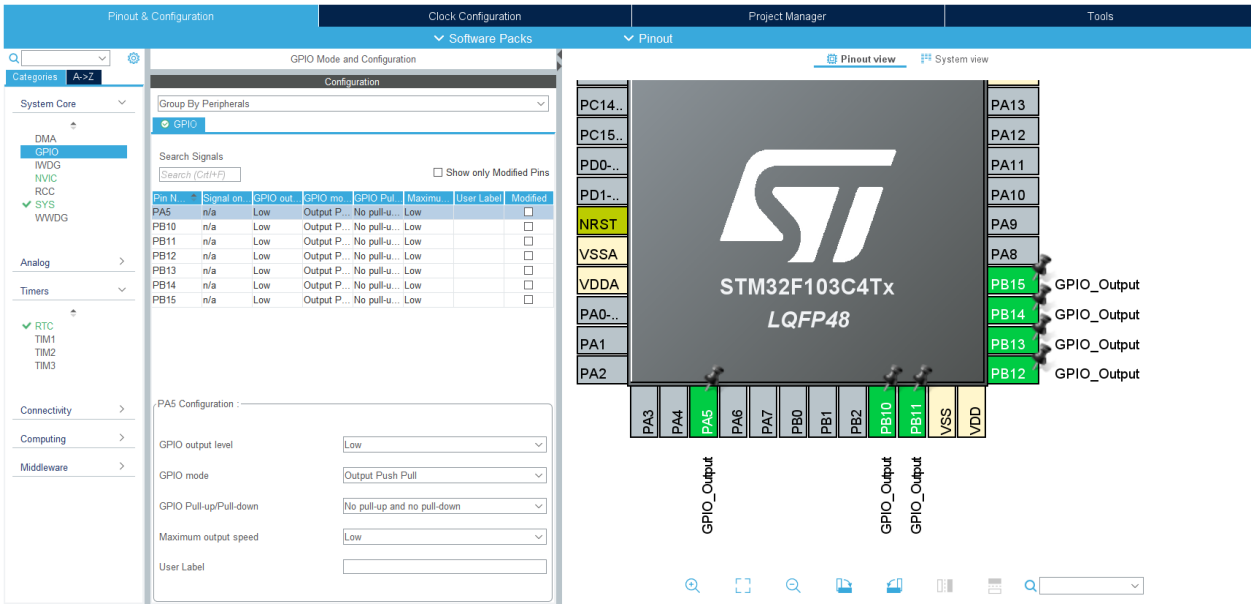


ضبط إعدادات المشروع:

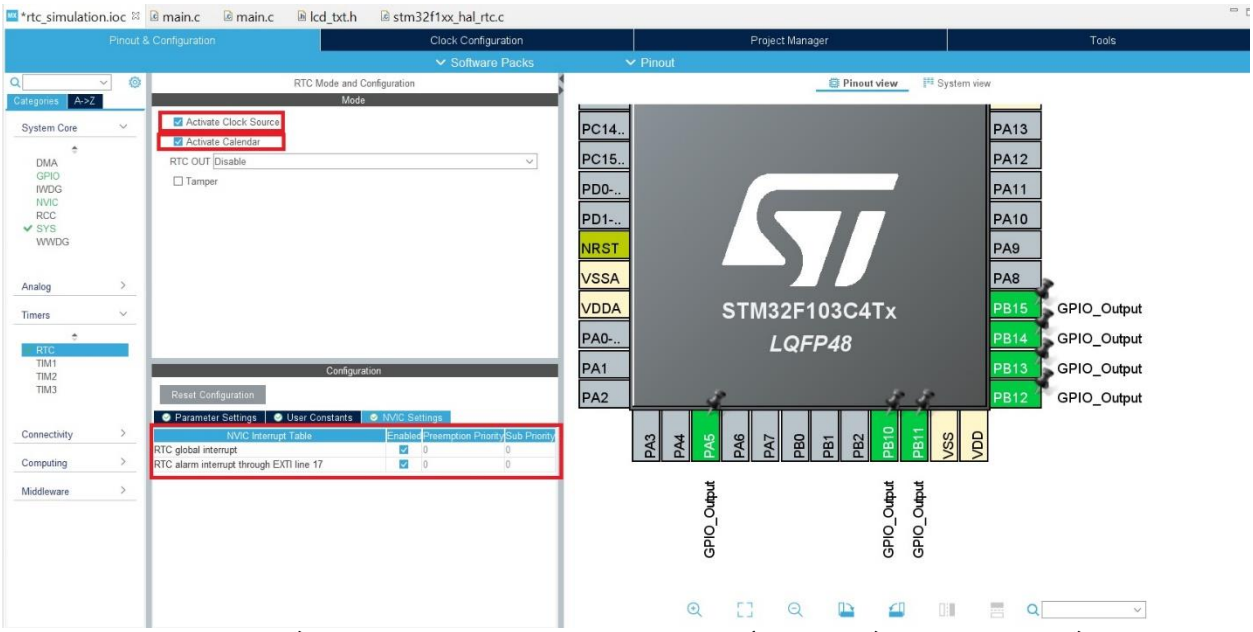
الخطوة الأولى: فتح بيئة STM32CubeIDE وإنشاء مشروع جديد ثم اختيار المتحكم المناسب

الخطوة الثانية: اختيار اسم للمشروع

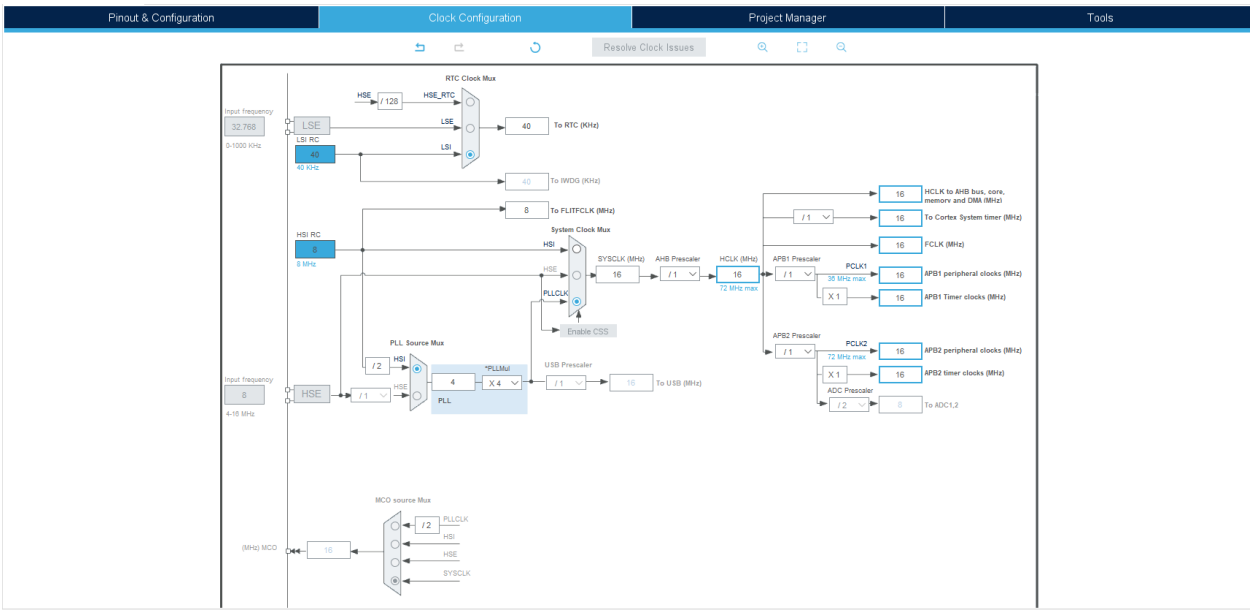
الخطوة الثالثة: اختر القطبPA5 لضـــبطه كقطب خرج، وضـــبط الأقطاب PB10..PB15 كأقطاب خرج ليتم وصلهم مع شاشة الـ lcd



الخطوة الرابعة: ضبط إعدادات وحدة الـ RTC وتفعيل المقاطعة الخاصة بها



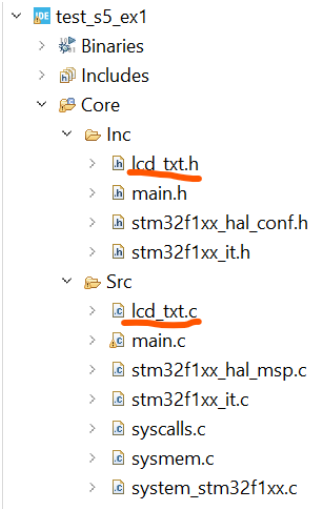
الخطوة الخامسة: ضبط تردد ساعة المتحكم: أثناء الـ Simulation نختار تردد الساعة 16MHZ



الخطوة السادسة: توليد الكود بناءً على الإعدادات التي تم ضبطها

الخطوة السابعة : إضافة مكتبة للـ LCD لتسهيل التعامل معها من خلال الخطوات التالية:

1- نقوم بنسخ الملف lcd_txt.h ثم لصقه ضمن المجلد inc للمشروع
2- نقوم بنسخ الملف lcd_txt.c ثم لصقه ضمن المجلد src للمشروع



3- نقوم بتعديل أسماء الأقطاب الموصولة مع الشاشة حسب التوصيل الخاص بمشروعنا

main.c lcd_txt.h
test_s5_ex1/Core/Src/main.c

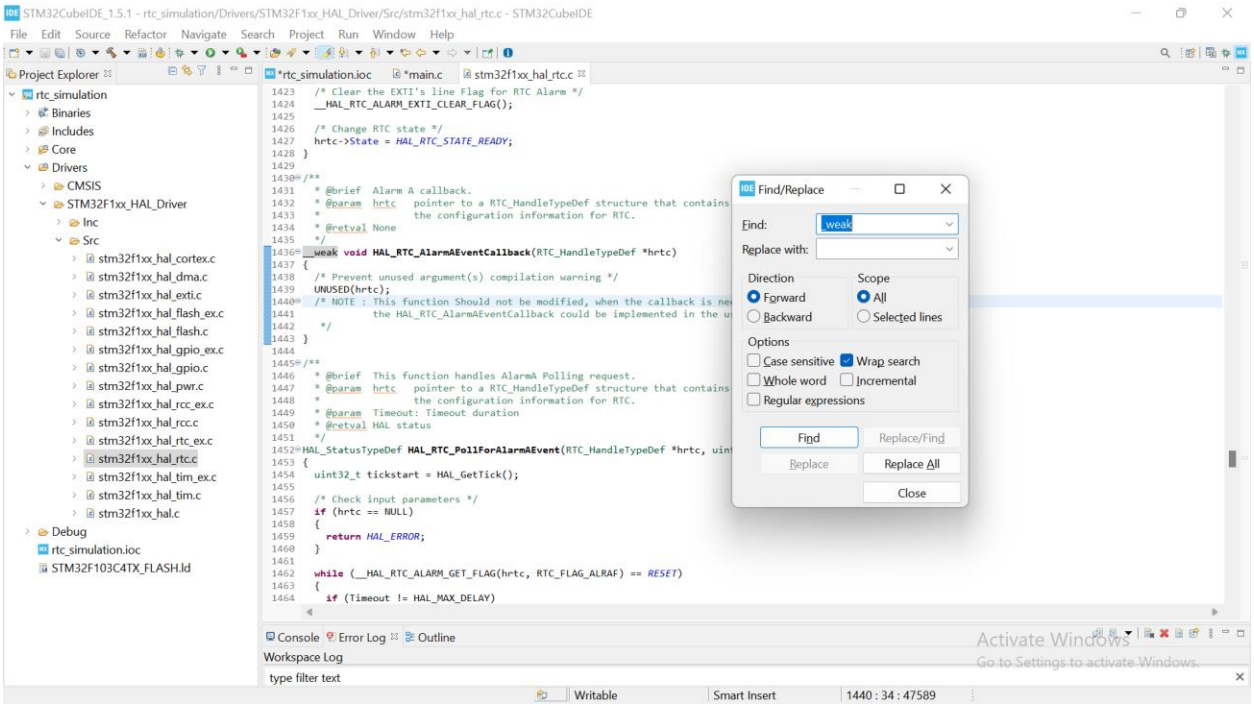
```
/*------------ Define LCD Use ----------------*/

/*Note: Comment which not use */

#define LCD16xN //For lcd16x2 or lcd16x4
//#define LCD20xN //For lcd20x4

/*------------- Define For Connection -----------------*/

#define RS_PORT     GPIOB
#define RS_PIN      GPIO_PIN_15

#define EN_PORT     GPIOB
#define EN_PIN      GPIO_PIN_14

#define D7_PORT     GPIOB
#define D7_PIN      GPIO_PIN_10

#define D6_PORT     GPIOB
#define D6_PIN      GPIO_PIN_11

#define D5_PORT     GPIOB
#define D5_PIN      GPIO_PIN_12

#define D4_PORT     GPIOB
#define D4_PIN      GPIO_PIN_13


/*------------ Declaring Private Macro -----------------*/
```

4- نستخدم الدوال التي تؤمنها المكتبة للتعامل مع الشاشة وهي:

```
/*------------ Declaring Function Prototype -------------*/
void lcd_init(void);
void lcd_write(uint8_t type,uint8_t data);
void lcd_puts(uint8_t x, uint8_t y, int8_t *string);
void lcd_clear(void);
```

الخطوة الثامنة: إضافة دالة خدمة مقاطعة التنبيه الخاصة بالـ RTC

عند الوصـــول إلى الزمن الذي تم ضـــبط منبه الـــــ RTC عليه تتولد مقاطعة ويذهب المعالج تلقائياً لبرنامج خدمة المقاطعة الخاص بها والذي يكون معرف بشكل افتراضي كـــ weak_ ضمن ملف الـــ stm32f1xx_hal_rtc.c الموجود ضمن مجلد الـــ Driversثم مجلد الـــ Src ويدعى HAL_RTC_AlarmEventCallback()، حيث نقوم بنسخ اسمه وإضافته للبرنامج الرئيسي ثم نقوم ضمنه بتشغيل الليد وممكن إضافة تنبيه صوتي أيضاً.



يصبح الكود بالشكل التالي:

```
#include "main.h"
RTC_HandleTypeDef hrtc;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_RTC_Init(void);
#include "main.h"
#include "lcd_txt.h"
char time[10];
char date[10];
uint8_t alarm =0;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_RTC_Init(void);
void set_time (void)
{
         RTC_TimeTypeDef sTime = {0};
          RTC_DateTypeDef DateToUpdate = {0};

          sTime.Hours = 5;
          sTime.Minutes = 0;
          sTime.Seconds = 0;

          if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN) != HAL_OK)
          {
            Error_Handler();
          }
         // DateToUpdate.WeekDay = 8;
          DateToUpdate.Month = 12;
          DateToUpdate.Date =8 ;
          DateToUpdate.Year = 22;

          if (HAL_RTC_SetDate(&hrtc, &DateToUpdate, RTC_FORMAT_BIN) != HAL_OK)
          {
            Error_Handler();
          }
  HAL_RTCEx_BKUPWrite(&hrtc, RTC_BKP_DR1, 0x32F2);  // backup register
}
```

```
void get_time(void)
{
  RTC_DateTypeDef gDate;
  RTC_TimeTypeDef gTime;

  /* Get the RTC current Time */
  HAL_RTC_GetTime(&hrtc, &gTime, RTC_FORMAT_BIN);
  /* Get the RTC current Date */
  HAL_RTC_GetDate(&hrtc, &gDate, RTC_FORMAT_BIN);

  /* Display time Format: hh:mm:ss */
  sprintf((char*)time,"%02d:%02d:%02d",gTime.Hours, gTime.Minutes, gTime.Seconds);

  /* Display date Format: mm-dd-yy */
  sprintf((char*)date,"%02d-%02d-%2d",gDate.Date, gDate.Month, 2000 + gDate.Year);  // I like the date
first
}

//Let's display the time and date on lcd

void display_time (void)
{
        lcd_puts(0,0,time);
        lcd_puts(1,0,date);
}
void set_alarm (void)
{
          RTC_AlarmTypeDef sAlarm;

          sAlarm.AlarmTime.Hours = 5;
           sAlarm.AlarmTime.Minutes = 0;
           sAlarm.AlarmTime.Seconds = 5;


           sAlarm.Alarm = RTC_ALARM_A;
           if (HAL_RTC_SetAlarm_IT(&hrtc, &sAlarm, RTC_FORMAT_BIN) != HAL_OK)
           {
             Error_Handler();
           }
 }
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
{
        alarm = 1;
}

void to_do_on_alarm (void)
{
        HAL_GPIO_WritePin (GPIOA, GPIO_PIN_5, 1);  // set led ON
        lcd_clear();
        lcd_puts(0,0,"Alarm Time");
        HAL_Delay (100);
        lcd_clear();
}

int main(void)
{
   HAL_Init();
  SystemClock_Config();
   MX_GPIO_Init();
  MX_RTC_Init();
  lcd_init();
  lcd_clear();

  if(HAL_RTCEx_BKUPRead(&hrtc, RTC_BKP_DR1) != 0x32F2)
      {
      //   Set the time
        set_time();
      }
```

```
  set_alarm ();
   /* USER CODE BEGIN WHILE */
  while (1)
  {
          get_time();
          display_time();
          HAL_Delay(200);
          if (alarm)
                  {
                          to_do_on_alarm();
                          alarm =0;
                  }


  }
```

ملاحظة:

سنترك ضمن برنامج تهيئة الـ RTC فقط التعليمات التالية:

```
static void MX_RTC_Init(void)
{
  hrtc.Instance = RTC;
  hrtc.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
  hrtc.Init.OutPut = RTC_OUTPUTSOURCE_ALARM;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }

 }
```

كي لا يتم إعادة تهيئة الوقت والتنبيه في كل مرة يتم فيها تحميل الكود إلى المتحكم

الخطوة التاسعة: ترجمة الكود وتحميله إلى المتحكم ضمن بيئة Proteus

الخطوة العاشرة: إعادة الخطوات السابقة ولكن من أجل المتحكم stm32G0B1CE الموجود على البورد ، ثم ترجمة الكود وتحميله للمتحكم