Architecture des systèmes à microprocesseur
TP n°2 : GPIO, Timer

Objectifs de la séance

-Compréhension du fonctionnement du Timer
-Programmation des registres du Timer
-Savoir générer un signal PWM à partir du Timer

Matériel requis :

-Une plateforme **nucleo-board STM32F446RE** par binôme

Organisation de la séance :

1) Exercices pour exploiter les GPIO, avec configuration sur STM32CubeMX
2) Description du Timer et de ses registres
3) Configuration du projet sous STM32CubeMX
4) Génération d'un signal d'horloge

Tout au long de ce TP, des questions vous sont posées. Prenez le temps d'y répondre et de prendre des notes.

I. Exploitation des GPIO *via* STM32CubeMX et Keil uVision

L'objectif de cette partie est de continuer à vous familiariser avec les GPIO des cartes STM32F4 (ici, STM32f446RE). Pour chaque exercice, s'assurer que les différentes entrées et sorties GPIO sont correctement configurées à l'aide de STM32CubeMX. Les programmes doivent être écrits sur Keil uVision. Referez-vous au TP 1 ainsi qu'au programme d'exploitation des timers à la fin de ce document pour correctement configurer votre carte avec STM32CubeMX et Keil uVision.

Exercice 1 :
On désire compter le nombre de fois que le bouton utilisateur a été pressé, et afficher le résultat sous forme binaire en passant par le tableau de LED fourni avec votre carte.

Exercice 2 :
Ecrire un programme qui parcourt un tableau de nombres entiers signés, dont la valeur est comprise entre -64 et +63. Pour chaque entier, afficher sa valeur binaire grâce au tableau de LED. Si le nombre est négatif, alors allumer la LED la plus à gauche de la carte. Attendre une seconde entre deux affichages.

Exercice 3 :
Ecrire un programme qui allume une LED différente a chaque pression du bouton utilisateur. Si jamais toutes les LED possibles ont été utilisées, la dernière LED (la plus « à gauche » ou la plus « à droite ») reste allumée, et on recommence au début. Si toutes les LED sont allumées, alors une nouvelle pression sur le bouton les éteint toutes. Exemple avec un tableau de 4 LED :

```
[ ] [ ] [ ] [ ]  ≡ Etat initial
[ ] [ ] [ ] [x]  ≡ première pression
[ ] [ ] [x] [ ]  ≡ deuxième pression
[ ] [x] [ ] [ ]  ≡ troisième pression
[x] [ ] [ ] [ ]  ≡ quatrième pression
[x] [ ] [ ] [x]  ≡ cinquième pression
...
[x] [x] [ ] [ ]  ≡ onzième pression
...
[x] [x] [x] [x]  ≡ quatorzième pression
[ ] [ ] [ ] [ ]  ≡ quinzième pression
```

Note : vous n'êtes pas obligés d'utiliser les 8 LED de la carte ; 3 ou 4 suffisent pour l'exemple.

II. Timers STM32 : Introduction

Lors de cette séance, vous allez mettre en œuvre un périphérique incontournable des microcontrôleurs, le **timer.**
Le timer est un compteur configurable de 16 ou 32 bits permettant de délivrer une base de temps, servant généralement de référence dans toutes sortes d'applications.
Le STM32F446RE possède 17 timers ; dont 12 de 16 bits et 2 de 32 bits, les 3 restants étant dédiés au SysTick et à 2 watchdog[1].

1 Documentation complète du STM32F446RE disponible ici (début page 446):
http://www.st.com/content/ccc/resource/technical/document/reference_manual/4d/ed/bc/89/b5/70/40/dc/DM00135183.pdf/files/DM00135183.pdf/jcr:content/translations/en.DM00135183.pdf

Documentation des timers des STM32 :
http://www.st.com/content/ccc/resource/technical/document/application_note/group0/91/01/84/3f/7c/67/41/3f/DM00236305/files/DM00236305.pdf/jcr:content/translations/en.DM00236305.pdf

- **Les timers**
  Les timers peuvent être utilisés dans plusieurs **modes** :
  -input capture : mesurer la durée d'un pulse d'un signal en entrée
  -output compare : génération de signaux périodiques
  -PWM (Pulse Width Modulation) : modulation de largeur d'impulsion de signaux
  -One pulse : génération d'un pulse en sortie

La durée des signaux générés peut s'étendre typiquement de quelques microsecondes à quelques millisecondes. En effet, l'ajustement de la durée s'effectue par la sélection de la source d'horloge d'entrée du timer et des valeurs de **prescaling**. Au niveau du timer et du contrôleur de reset et des d'horloges (RCC), ces blocs de **prescaling** permettent de diviser une fréquence d'horloge par une valeur spécifique configurable. Par exemple, le Timer 1 possède un **prescaler** sur 16-bits.

Questions :

-Par quel facteur peut-on diviser la fréquence d'horloge à l'aide d'un **prescaler** sur 16 bits ? [1 à].

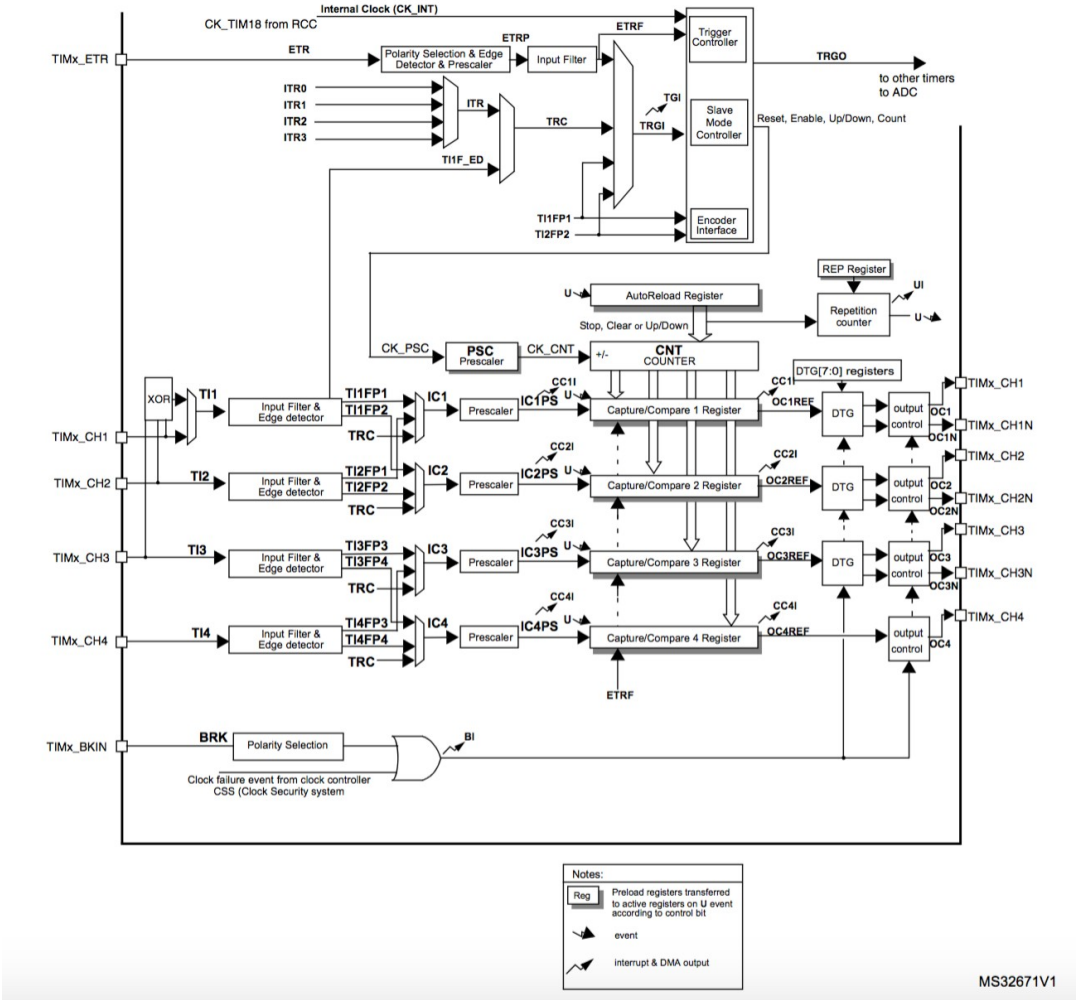


Figure 1 : bloc diagramme du timer 1

On peut noter sur la figure 1, que Timer 1 possède 4 canaux indépendants (TIM_CH1 à TIM_CH3) pouvant fonctionner dans les différents modes décrits précédemment.

- Les principaux registres des timers

L'unité de comptage du temps se base essentiellement sur 4 registres :

- Counter register (TIMx_CNT)

- Prescaler register (TIMx_PSC)

- Auto-reload register (TIMx_ARR)

- Repetition counter register (TIMx_RCR)

Le TIMx_CNT est le registre de comptage, le TIMx_PSC est le registre permettant de diviser l'horloge de référence, TIMx_ARR permet de définir une valeur seuil déclenchant la remise à 0 du compteur ou le décomptage par exemple. Pour finir, le registre TIMx_RCR est utilisé pour la répétition. Le contenu du registre indique combien de fois (TIMx_RCR+1) doit être répété le comptage avant de générer un Update Event (UEV). Sinon un UEV est généré à chaque fois que le compteur génère un overflow.

- Définition de l'horloge de référence

L'horloge de référence qui est fournie au prescaler est notée CK_PSC. L'horloge issue du **prescaler** est l'horloge du timer et elle est notée CK_CNT (cf. la figure 1).

L'horloge de référence CK_PSC peut provenir de plusieurs sources :

- ➔ Horloge interne (CK_INT)
- ➔ D'une broche d'entrée externe (*External clock mode 1*)
- ➔ D'un trigger externe d'entrée ETR (*External clock mode 2*)
- ➔ De plusieurs triggers internes (ITRx). Par exemple, dans le cas où un timer servirait de diviseur d'horloge d'un autre timer.

- Les types de comptage

Les registres de comptage peuvent s'incrémenter, se décrémenter, ou bien les deux. Ils se rechargent automatiquement à l'aide du registre TIMx_ARR.

- Upcounting

Dans ce mode, le compteur compte de 0 jusqu'à la valeur définie dans le registre TIMx_ARR. Puis il recommence à 0 et génère un évènement d'overflow (Update EVent - UEV). Dans le cas d'un compteur à répétition, cet UEV après que le compteur est effectué TIMx_RCR+1 boucle de comptage. Sinon, l'UEV est généré à chaque overflow.
Un flag d'interruption peut aussi être généré.

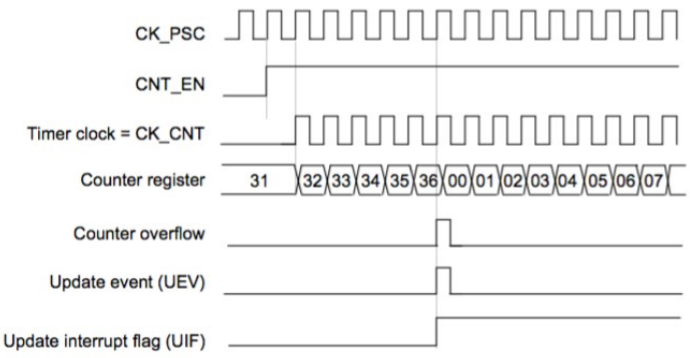


Figure 2 : Upcounting mode avec seuil à 0x36 (TIMx_ARR)

et prescaling d'horloge à 1

- Downcounting

Même principe que dans le cas précédent où cette fois-ci, le compteur décrémente de la valeur seuil (TIMx_ARR-1) jusqu'à 0, puis recommence si la répétition est activée. L'événement est aussi généré à la fin de la décrémentation.

- Center-aligned mode (up/down counting)

Dans ce mode, le compteur va de 0 à la valeur du registre TIMx_ARR – 1, génère un événement d'overflow, puis décompte jusqu'à 1 et génère un événement d'underflow. Puis le compteur recommence le comptage à partir de 0.

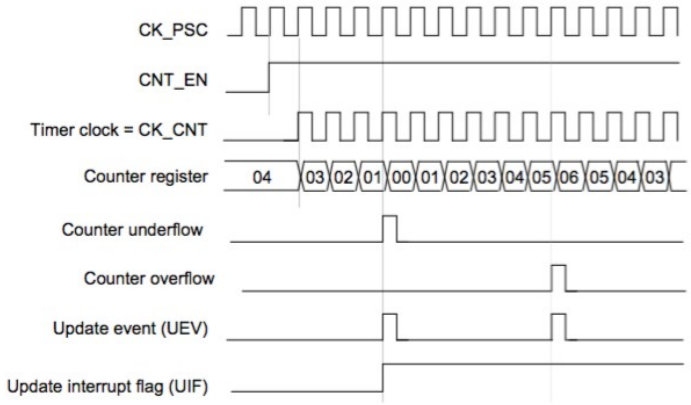
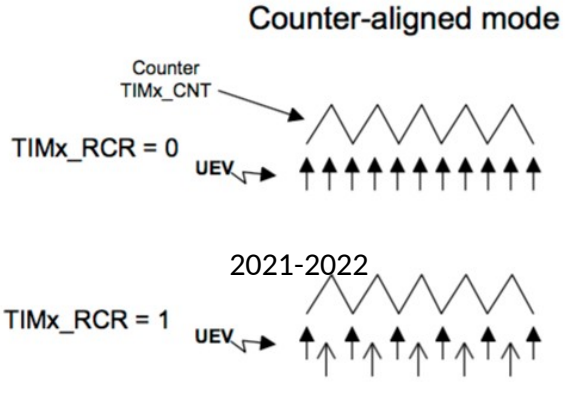


Figure : Illustration du timer en mode center-aligned (Prescaling=1 et TIMx_ARR = ?)

Question : Quelle est la valeur inscrite dans le registre TIMx_ARR dans l'exemple présenté en figure 3 ?

- Repetition counter

Nous avons vu qu'il était possible de compter/décompter de manière répéter un certain nombre de fois indiqué par le registre TIMx_RCR, avant de générer un UEV. Ceci est très utile pour la génération de signaux de type PWM par exemple.
Sur la figure suivante, on peut observer l'impact du choix de la valeur du nombre de répétition indiqué dans TIMx_RCR sur la génération des UEV.



Impact

- **Mode Capture/Compare**

Dans ce mode, l'objectif est l'acquisition d'un signal (Capture), son filtrage puis la détection de front avec sélection de polarité via le registre TIxFPx. La détection de front peut servir de trigger d'entrée ou ordre de capture.

L'étage de sortie (capture/compare block) est constitué de deux registres, un comparateur et un registre de sortie de contrôle.
Dans ce mode, il est possible de mesurer des largeurs ou des fréquences d'impulsion.

- **Mode Input Capture**

Dans ce mode, les registres TIMx_CCRx permettent de récupérer la valeur du compteur après qu'une transition ait été détectée sur le signal d'entrée ICx. Dans ce cas, le flag CCXIF du registre TIMx_SR est mis à 1 et une interruption (ou requête DMA) est générée (si elles sont autorisées). Ce flag est remis à 0 lors de la lecture du registre de comptage TIMx_CCRx.

A titre d'exemple, on pourrait utiliser ce mode pour mesurer l'intervalle de temps entre deux fronts montants détectés consécutivement sur une entrée.

- **Output Compare Mode (cf. p262 [1])**

Cette fonction permet le contrôle et la génération de signaux. Elle peut aussi être utilisée pour indiquer lorsqu'une période de temps précise vient de s'écouler, en positionnant une sortie à 0 ou 1 par exemple.

Comme son nom l'indique, l'objectif est de comparer la valeur du compteur avec celle du registre de capture/compare (TIMx_CCRx).

A chaque détection de valeur égale, l'étage de comparaison peut alors :

-assigner en sortie une valeur définie via les bits OCxM du registre TIMx_CCMRx,

-changer la polarité de la broche sortie en modifiant le bit CCxP du registre TIMx_CCER

-ne pas changer la valeur de la broche de sortie (OCxM = 000), l'activer (001), la désactiver (010) ou la toggler (011).

Un flag d'interruption est mis à 1 (bit CCxIF du registre TIMx_SR).

Si le masque d'interruption est activé (bit CCXIE du registre TIMx_DIER), alors une interruption est générée. Une requête DMA peut aussi être activée si le bit CCxDE du registre TIMx_DIER et le bit CCDS du registre TIMx_CR2 (sélection) sont activés.

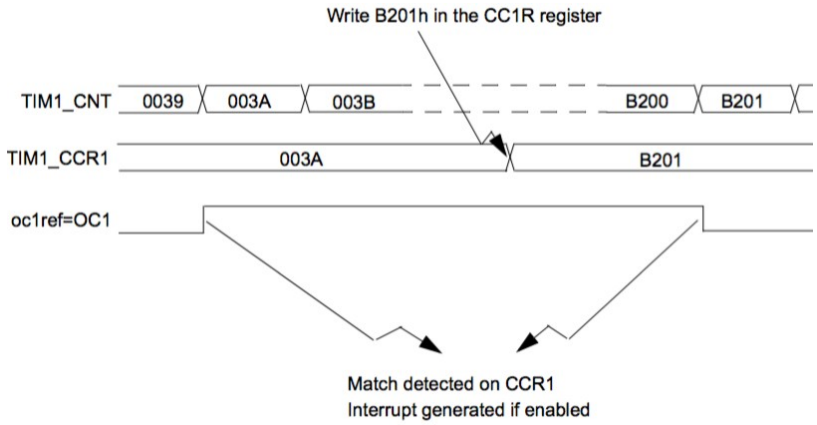


Figure : illustration du monde Output compare

Figure : Mode esclave, le mode 'trigger + external clock mode 2'

III. Programmes à réaliser avec le Timer

ALLUMAGE PERIODIQUE D'UNE LED

On souhaite utiliser le timer 1 en mode interruption afin de *toggler* la led LD2 toutes les secondes (f=1Hz). Une interruption sera donc générée toutes les secondes.

Pour cela, on va configurer le timer pour :

-définir la fréquence d'horloge du compteur (Prescaler)

-définir la période de comptage pour générer une interruption à un instant donné.

Questions :

-Quelle est la fréquence d'horloge en entrée du timer (avant prescaler)

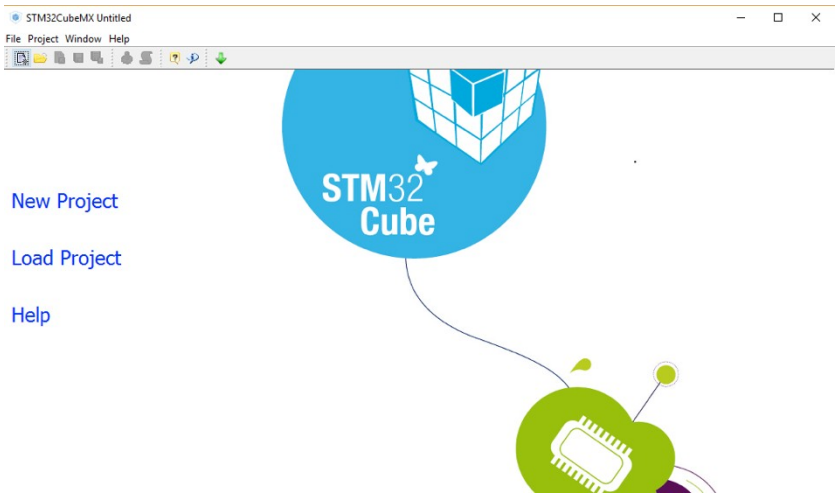
-Par combien doit-on diviser la fréquence de cette horloge au niveau du prescaler ?

-Jusqu'à combien doit s'incrémenter le compteur avant de générer l'interruption ?

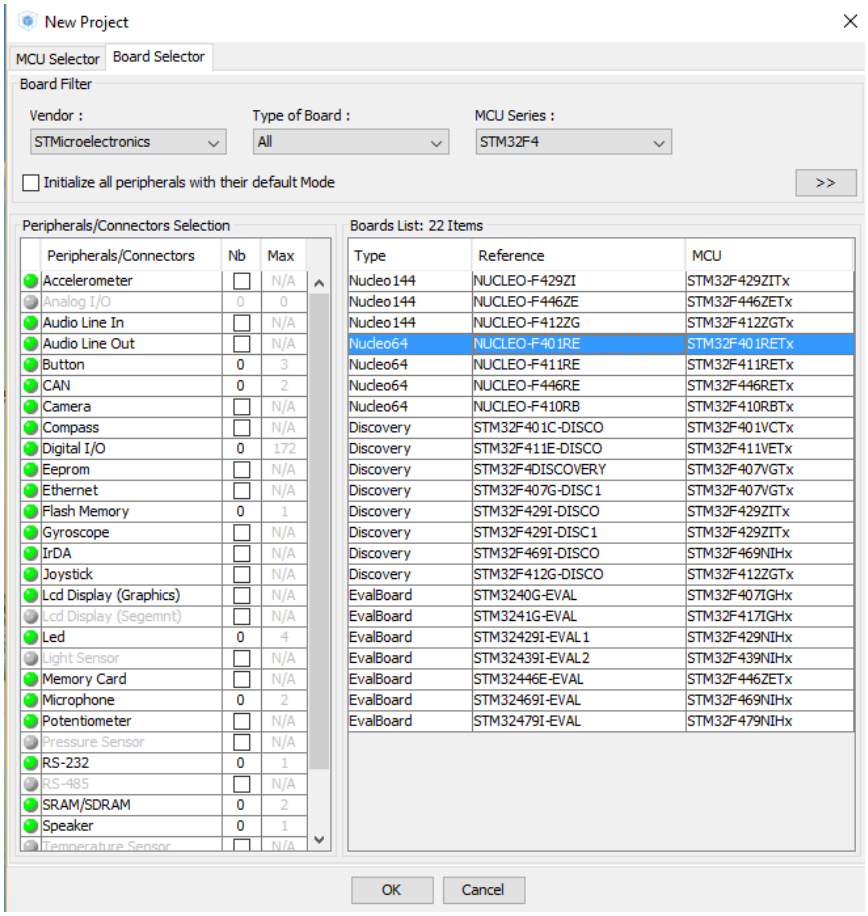
- **Configuration du projet sous STM32CubeMX**

Comme au TP précédent, on se propose d'utiliser l'outil **STM32CubeMX** pour configurer graphiquement le microcontrôleur. On y spécifie notamment les sources d'horloge, les périphériques que l'on souhaite utiliser, les sources interruptions, etc.

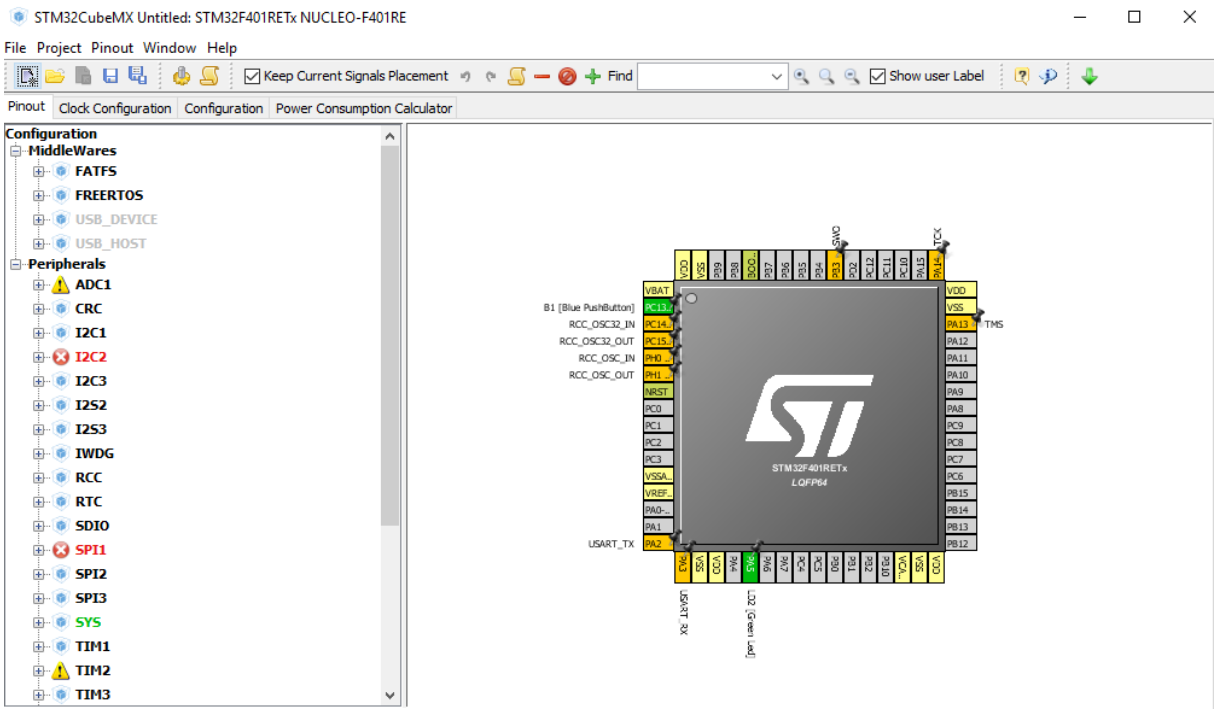
1. Lancer STM32CubeMX



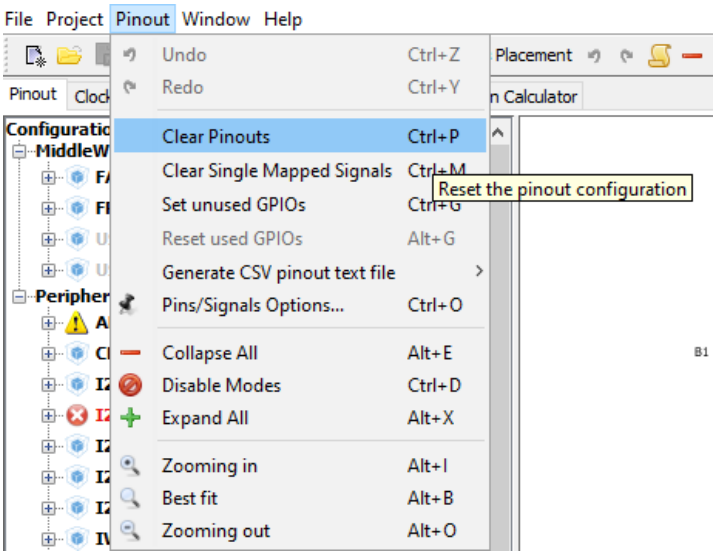
2. Créer un nouveau projet. Chercher et sélectionner ensuite la plateforme que l'on souhaite utiliser (Nucleo64 avec un MCU STM32F446RETx) puis **OK**.



Vous arrivez sur l'écran principal de configuration du MCU.

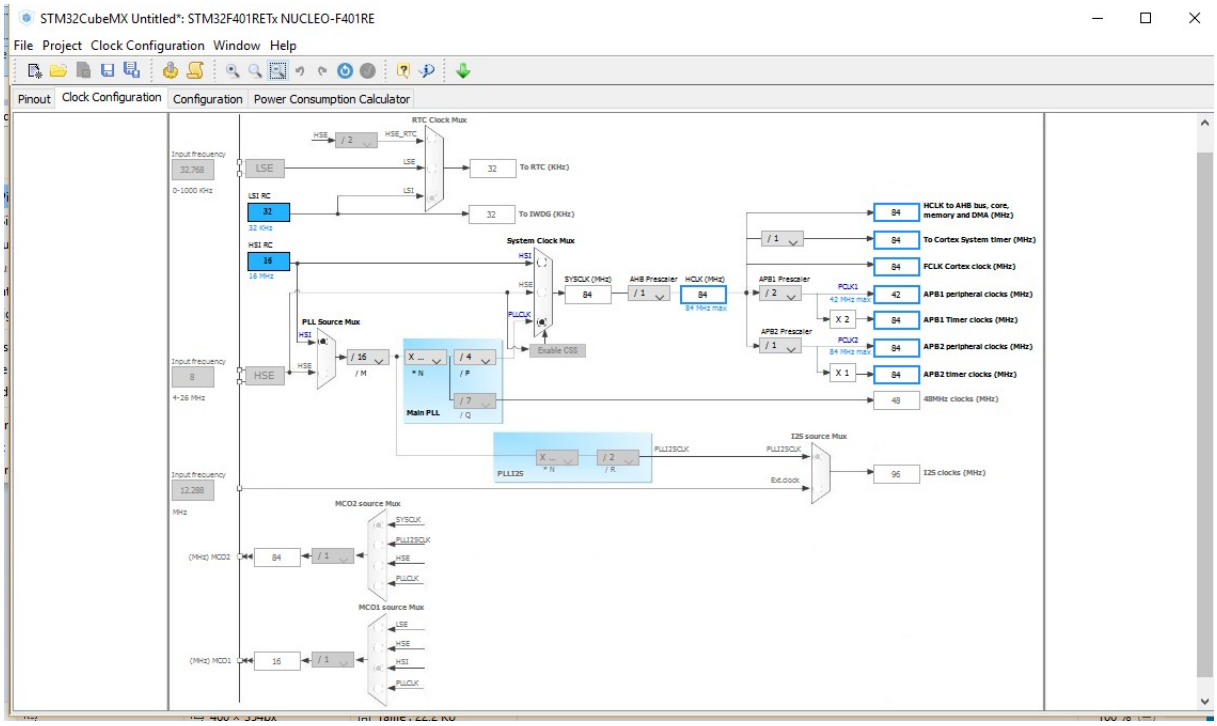


3. Faire une remise à zéro des broches via l'onglet **Pinout/Clear Pinouts**



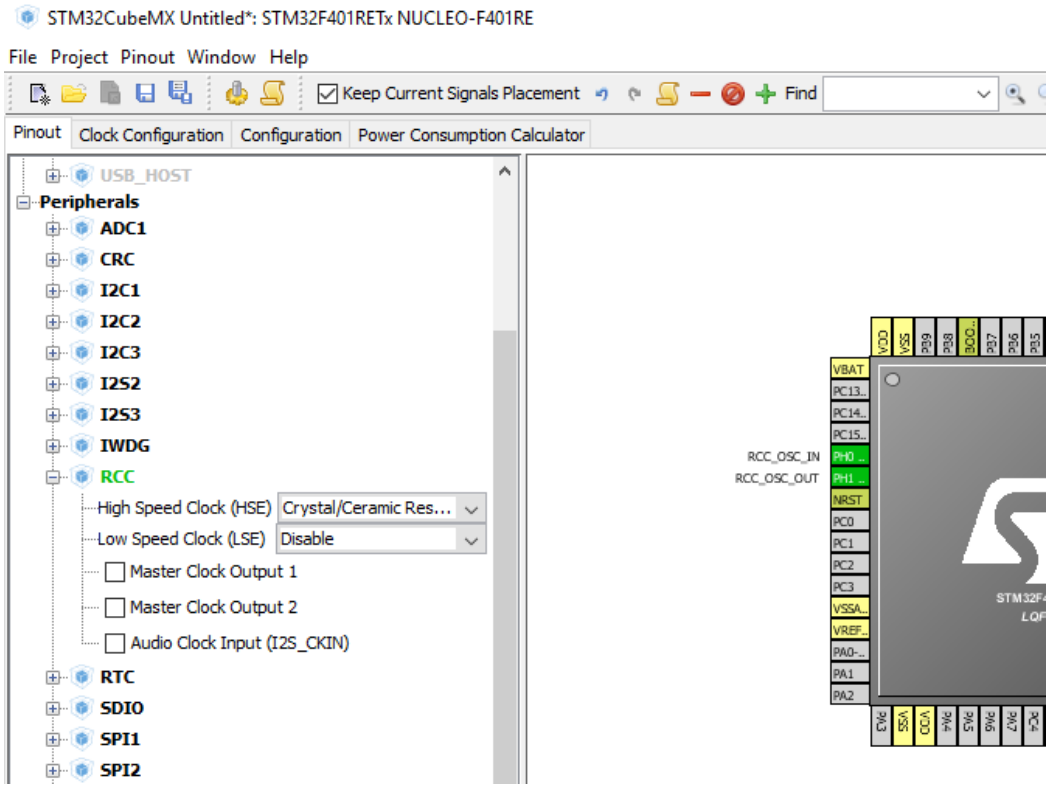
4. Vous pouvez observer dans la fenêtre **Pinout**, l'ensemble des périphériques qui peuvent être utilisés ainsi que les OS temps-réels et autres fonctionnalités logicielles.

5. A côté de **Pinout** se trouve **clock configuration**.



Dans cette fenêtre, vous pouvez graphiquement définir les sources d'horloge pour configurer l'horloge principale (SYSCLK) du microcontrôleur. Vous noterez qu'il existe plusieurs entrées possibles (HSI/HSE et PLLCLK). En d'autres termes, vous pouvez choisir entre une source d'horloge interne (HSI - 16MHz), une source d'horloge externe (HSE – 4 à 26MHz) et une horloge générée par une PLL (PLLCLK). De plus, vous remarquerez que l'horloge des bus et des périphériques dépendent de SYSCLK, hormis l'horloge pour l'USB ainsi que pour l'audio (I2S clock).

Dans la fenêtre **Pinout**, indiquer comme source d'horloge **HSE** (High Speed Clock) que la source d'horloge provient du **Crystal/Ceramic Resonator**.



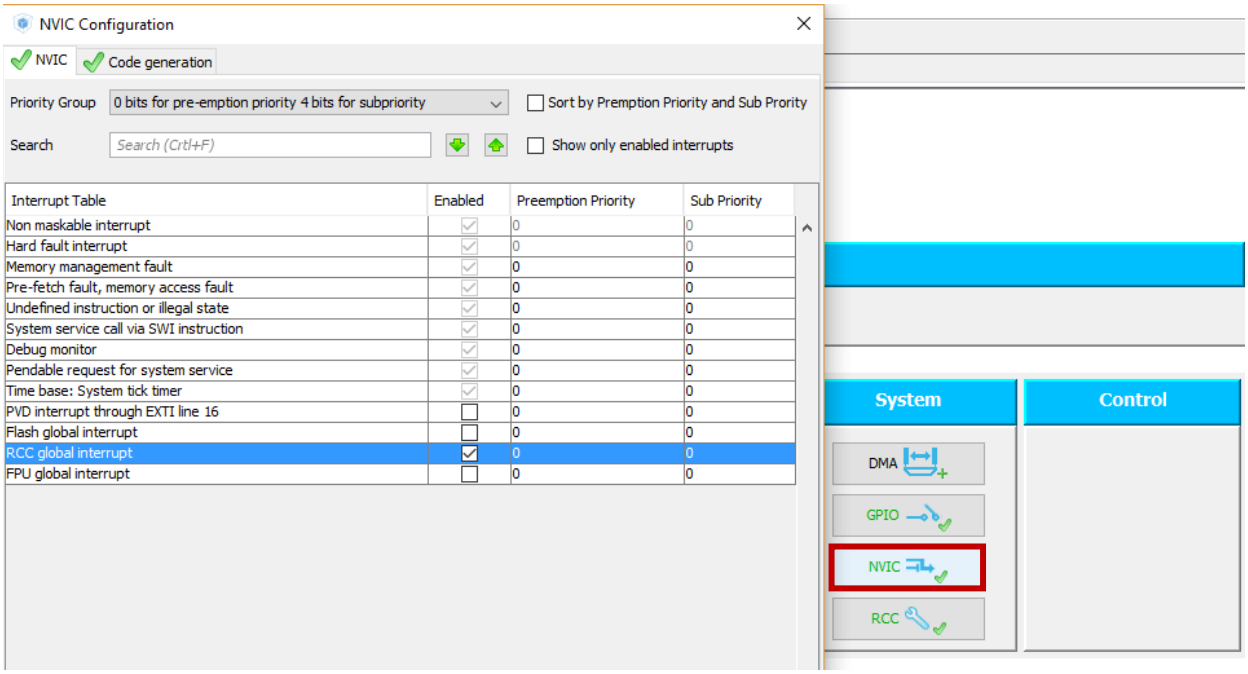
Nous avons vu aussi qu'il existe un registre spécifique de contrôle au sein du microcontrôleur permettant de configurer les sources d'horloge. C'est le registre **RCC** (Reset and Clock Control).

Dans **clock configuration**, modifier certains éléments afin d'obtenir une horloge SYSCLK de 84MHZ.

Les horloges fournies aux timers sont définies à partir des horloges APB1 TIMER Clocks (TIMER 2,3,4,5,6,7,12,14) et APB2 TIMER Clocks (TIMER 1,8,9,10 et 11).

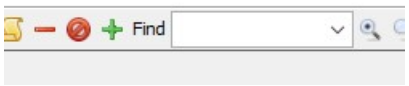
6. Le contrôleur de reset et d'horloge (RCC) permet aussi de configurer les différentes sources de reset :
   - System reset (remise à zéro des registres via NRST pin)
   - Power reset,
   - Backup domain reset.

Pour permettre une interruption de reset, il est nécessaire de l'autoriser au niveau du contrôleur d'interruption NVIC comme indiqué sur la figure suivante, accessible via la fenêtre **configuration** :

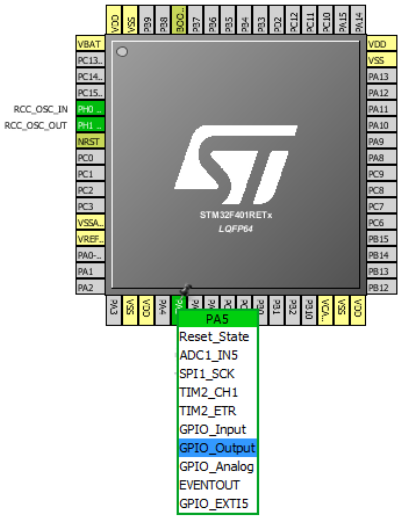


Apply and **OK**.

7. On souhaite aussi pouvoir contrôler la LED. Celle-ci est connectée au Port A d'E/S (GPIOA) et à la broche 5 (PA5). Vous pouvez la chercher via le Find :

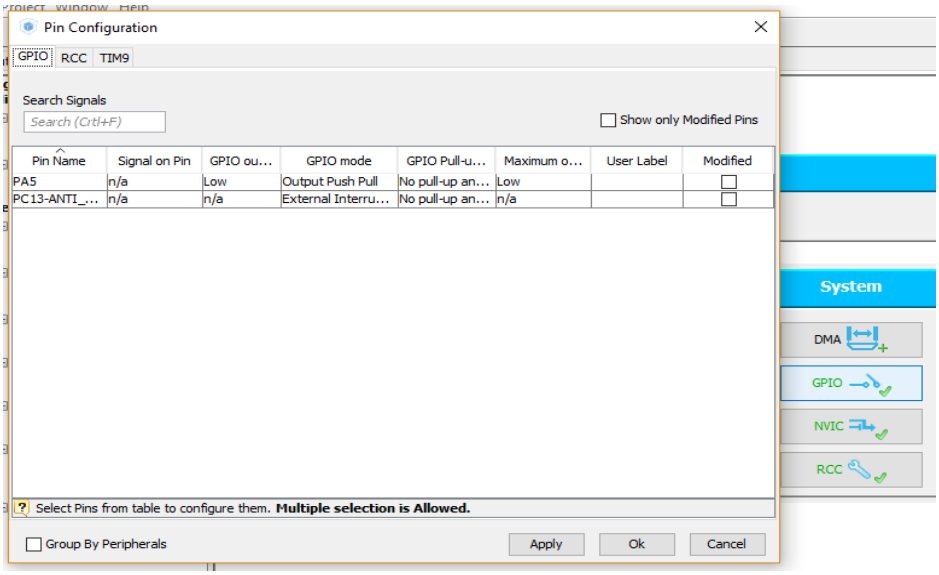


Il faut donc la définir en tant que **GPIO_Output** comme suit :



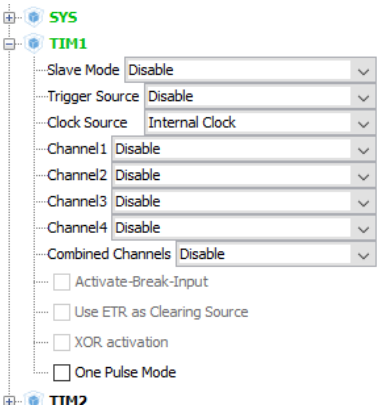
Vous noterez que la broche PA5 possède plusieurs fonctionnalités (pouvant servir à d'autres périphériques comme le TIMER 2, l'ADC, la liaison SPI …). Une fois configurée, retourner dans la fenêtre **configuration**, puis **GPIO**. Vous noterez comment a été configurée la broche (push-pull…)

Optionnel *: de la même façon que PA5, on pourrait configurer la broche* ***PC_13*** *en tant qu'entrée d'interruption* ***GPIO_EXTI13****. On rappelle que cette broche est directement reliée au bouton poussoir utilisateur. Dans la fenêtre configuration, puis GPIO, vous devez obtenir l'image suivante :*

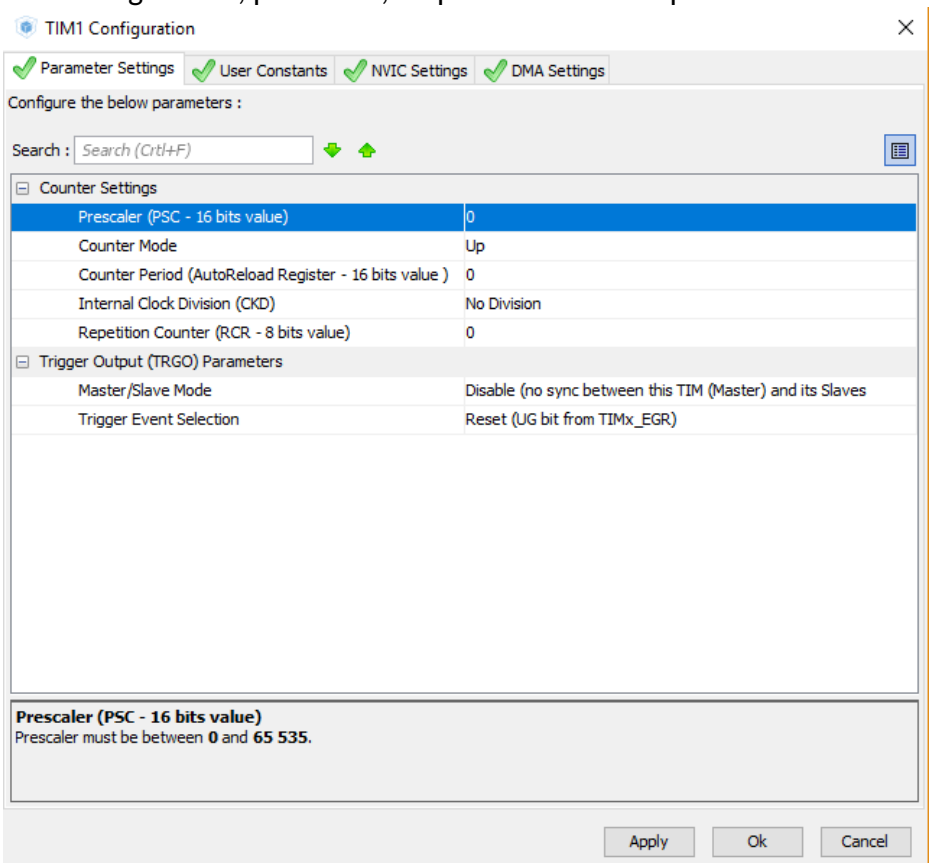


8. Configuration du timer (pour Exo 1)

Ajouter ensuite dans la fenêtre **Pinout**, **le timer 1.** On indiquera seulement la source d'horloge comme étant interne.



Dans la fenêtre configuration, puis TIM1, on peut observer les paramètres suivants :

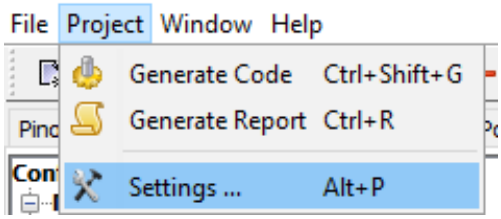


Modifier les paramètres du **Prescaler** et de **Counter Period** pour que le compteur du puisse avoir une période de comptage d'1s.

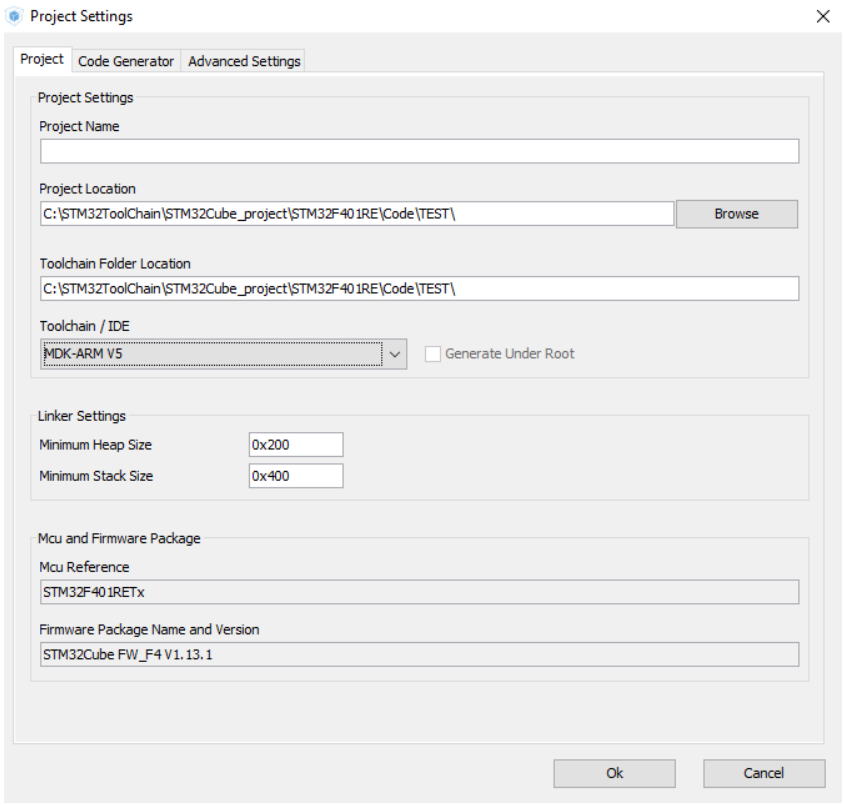
Dans l'onglet **NVIC Settings**, cochez la case **Enable** pour **TIM1 update and TIM10 global interrupt**. De cette manière, une interruption sera générée à l'issue de chaque cycle.
Apply et **OK**.

9. Génération du code et du projet pour Keil uVision

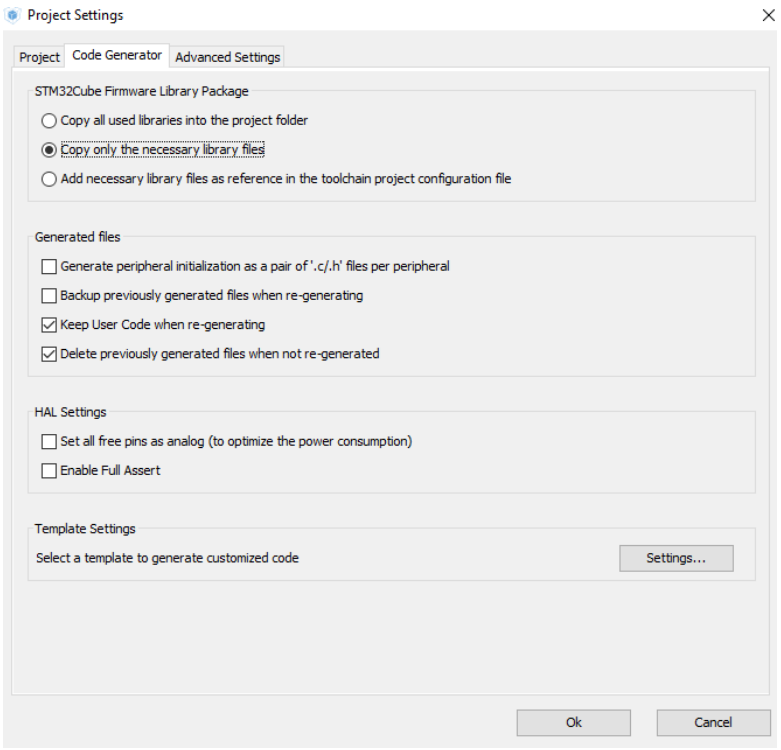
C'est la dernière étape de configuration du projet. Aller dans **Project/Settings**.



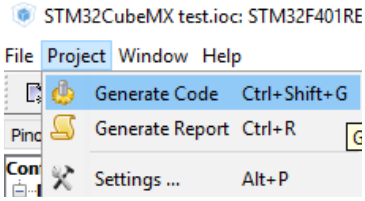
Dans **Project Name** indiquer le nom de votre projet.
Dans **Project location**, indiquer le répertoire de votre projet où sera générer le code.
Dans **Toolchain/IDE**, indiquer **MDK-ARM v5**.



Dans l'onglet suivant, **Code Generator**, sélectionner **"Copy only the necessary library files".**
Puis **OK**

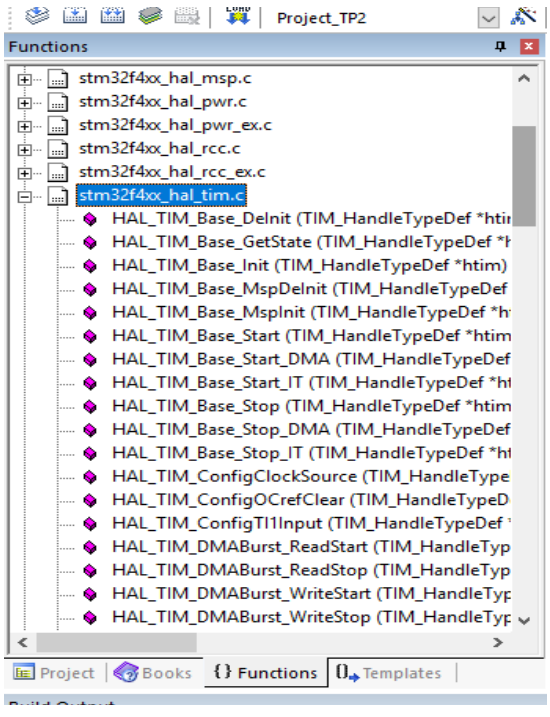


Enfin, dans l'onglet **Project, Generate Code**.



Cliquer ensuite sur **Open Project** pour ouvrir le projet généré sous µVision.

- **Sous uVision (Kiel)**



Compléter le code généré pour réaliser la fonctionnalité souhaitée.

-> Pour cela, regarder les fonctions disponibles dans la librairie.

<u>En cas de problème, servez-vous du debugger afin d'observer les registres du timer.</u>

IV. Références

ST Microelectronics,"STM32F446xx advanced ARM based 32-bit MCUs", RM0390 Reference Manual, Jan 2016, http://www.st.com/content/ccc/resource/technical/document/reference_manual/4d/ed/bc/89/b5/70/40/dc/DM00135183.pdf/files/DM00135183.pdf/jcr:content/translations/en.DM00135183.pdf