# TP n°1 : Architecture des systèmes à microprocesseur

**Objectifs de la séance**

**-Prise en main de l'environnement de développement µVision de Kiel**
**-Compréhension des fichiers sources nécessaires à la programmation du microcontrôleur**
**-Utilisation du debugger, visualisation des registres et des mémoires**

**Matériel requis :**

-Une plateforme **nucleo-board STM32F401RE** par binôme

**Organisation de la séance :**

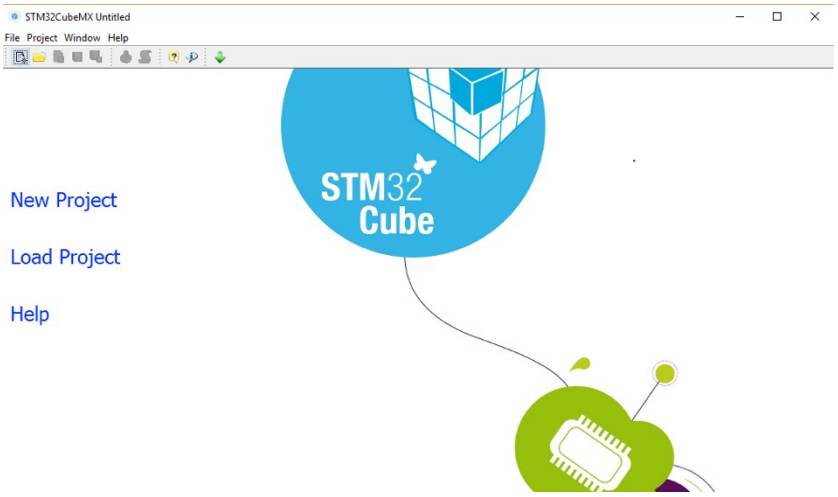
1) Analyse des fichiers
2) Utilisation du debugger
3) Ajout de fonctionnalités
4) Conclusion / Bilan

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

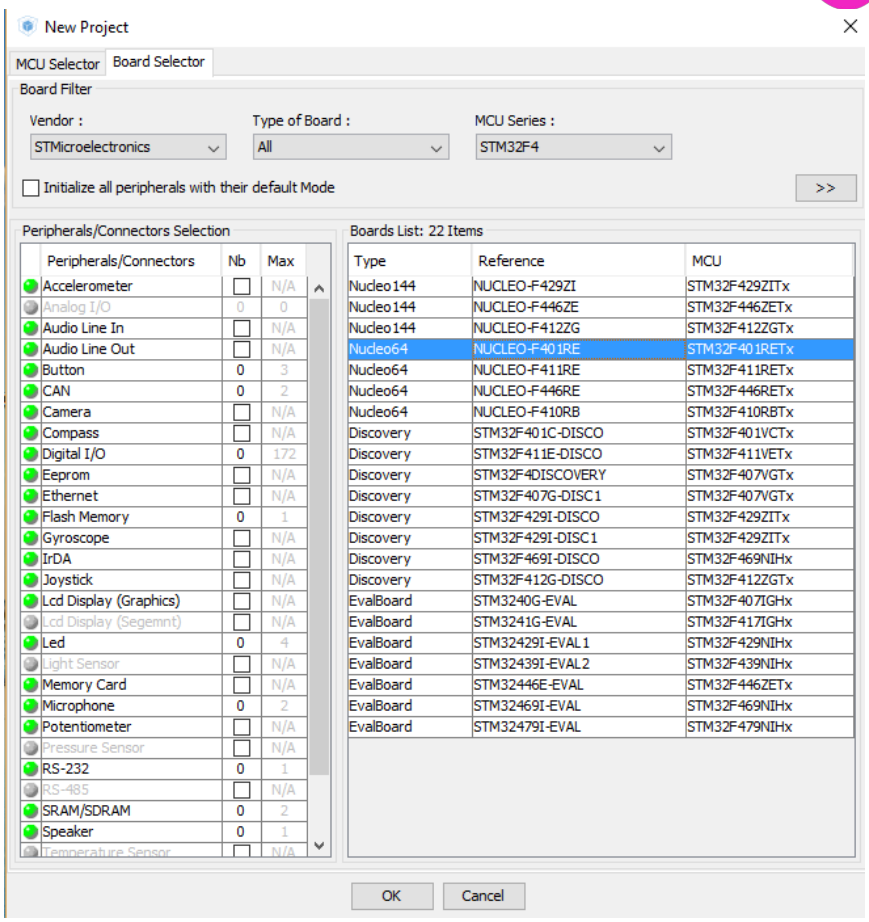
## I. Introduction

On se propose d'utiliser l'outil **STM32CubeMX**. Cet utilitaire permet de configurer graphiquement le microcontrôleur. On y spécifie notamment les sources d'horloge, les périphériques que l'on souhaite utiliser, les sources interruptions, etc.

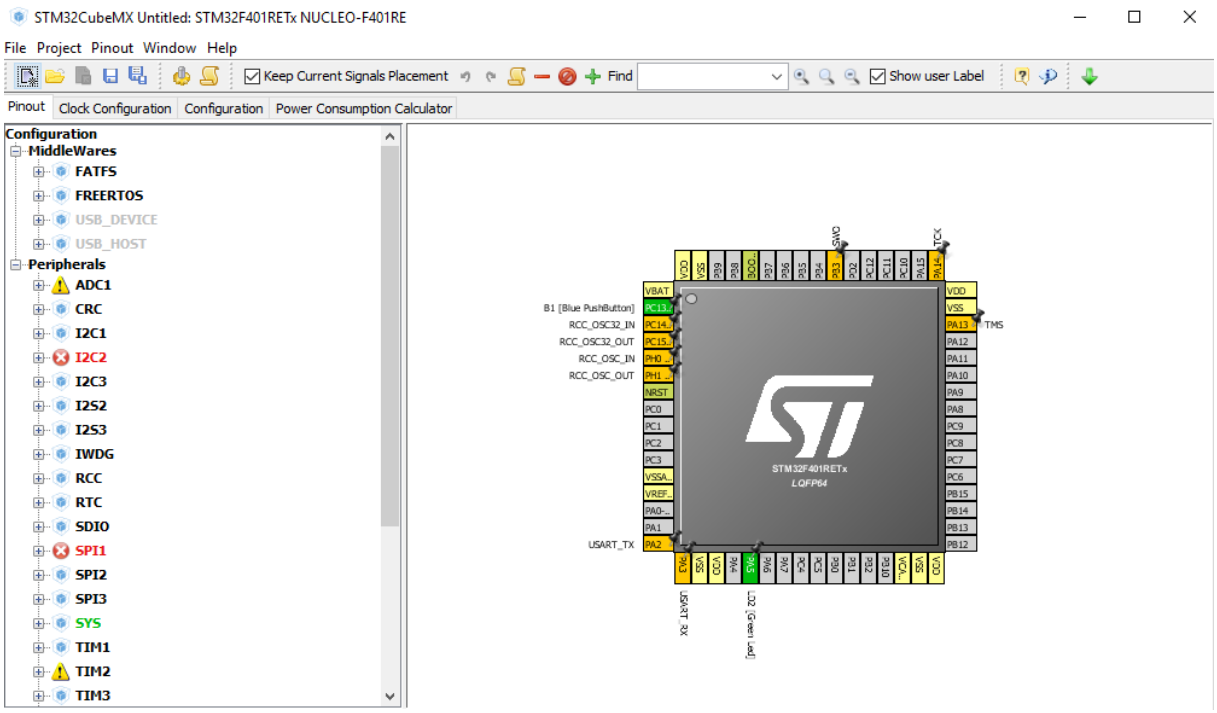
1. Lancer STM32CubeMX



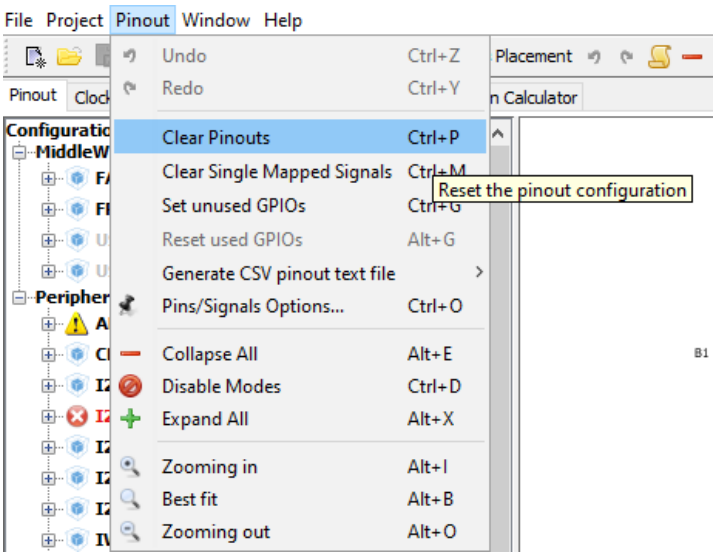

2. Créer un nouveau projet. Chercher et sélectionner ensuite la plateforme que l'on souhaite utiliser (Nucleo64 avec un MCU STM32F401RETx) puis **OK**.




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

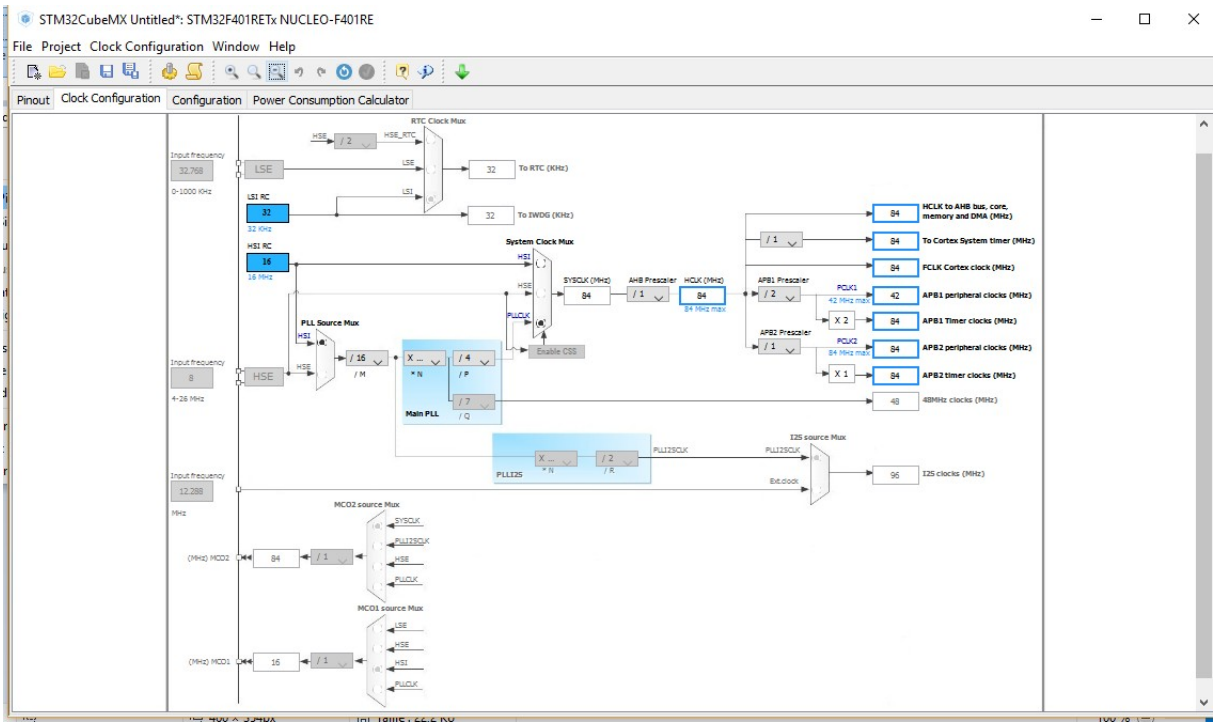



3. Faire une remise à zéro des broches via l'onglet **Pinout/Clear Pinouts**




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.

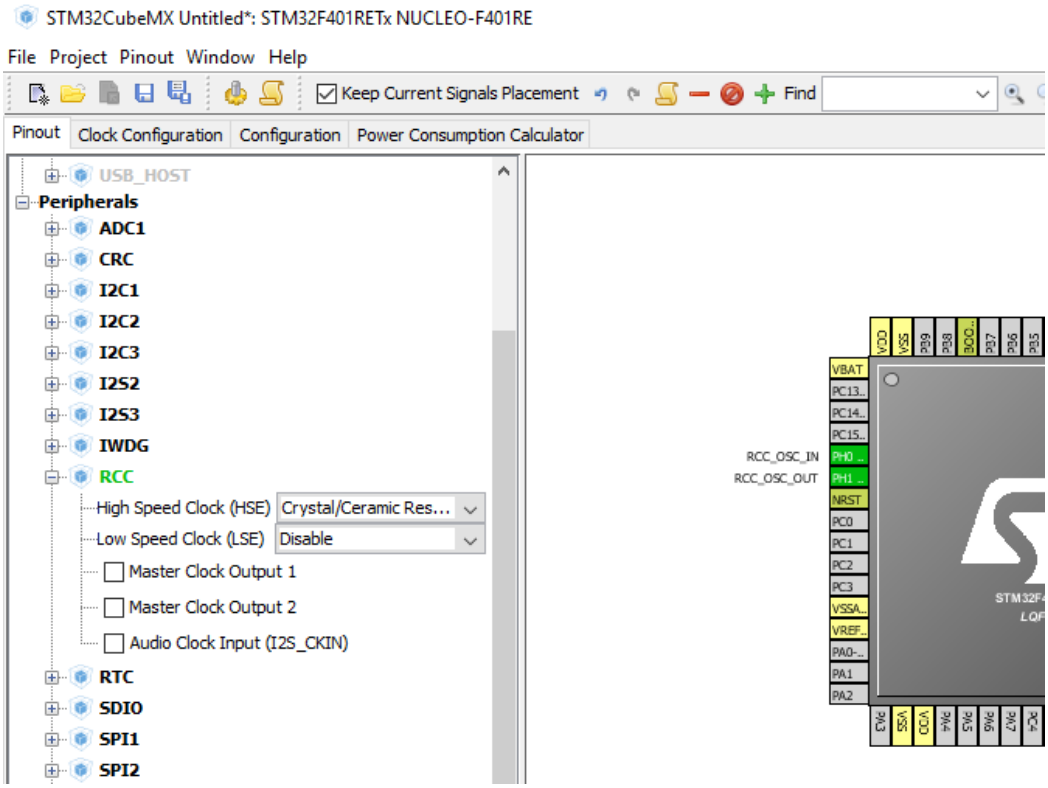
5. A côté de **Pinout** se trouve **clock configuration**.



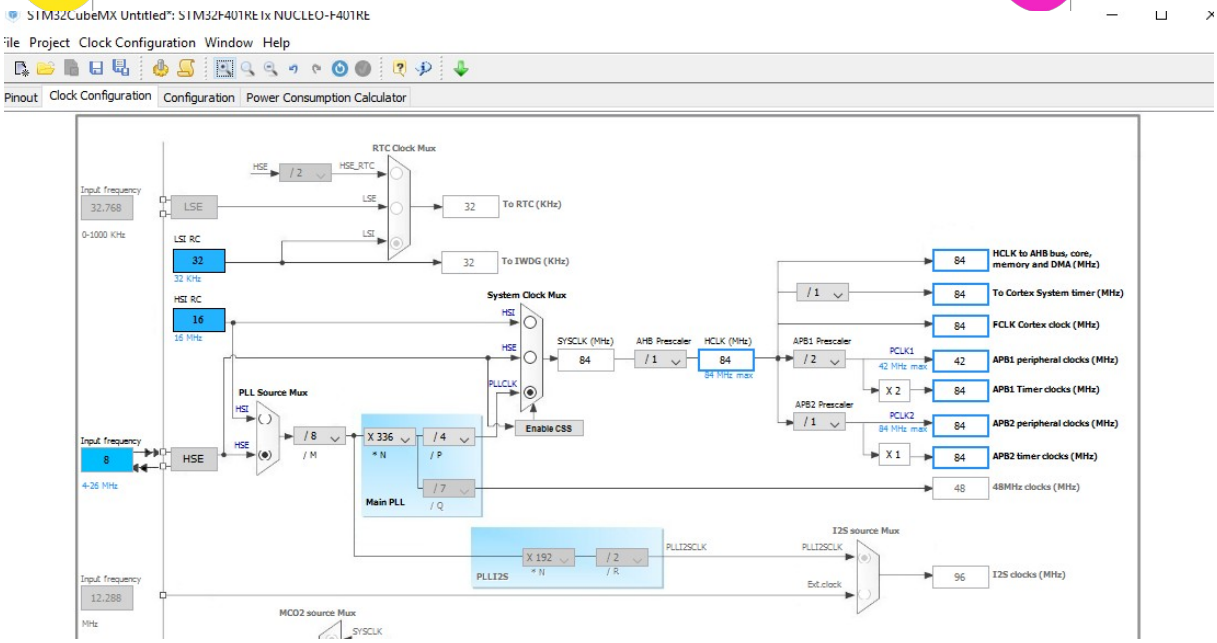

Dans cette fenêtre, vous pouvez graphiquement définir les sources d'horloge pour configurer l'horloge principale (SYSCLK) du microcontrôleur. Vous noterez qu'il existe plusieurs entrées possibles (HSI/HSE et PLLCLK). En d'autres termes, vous pouvez choisir entre une clock interne (HSI - 16MHz), une clock externe (HSE – 4 à26MHz) 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).

Nous allons choisir une source d'horloge externe et générer une horloge système SYSCLK à 84MHz. Nous avons vu aussi qu'il existe un registre spécifique de contrôle au sein du microcontrôleur permettant de configurer la source d'horloge principale. C'est le registre **RCC** (Reset and Clock Control).
Dans la fenêtre **Pinout**, indiquer comme source d'horloge **HSE** (High Speed Clock) que la source d'horloge provient du **Crystal/Ceramic Resonator**.

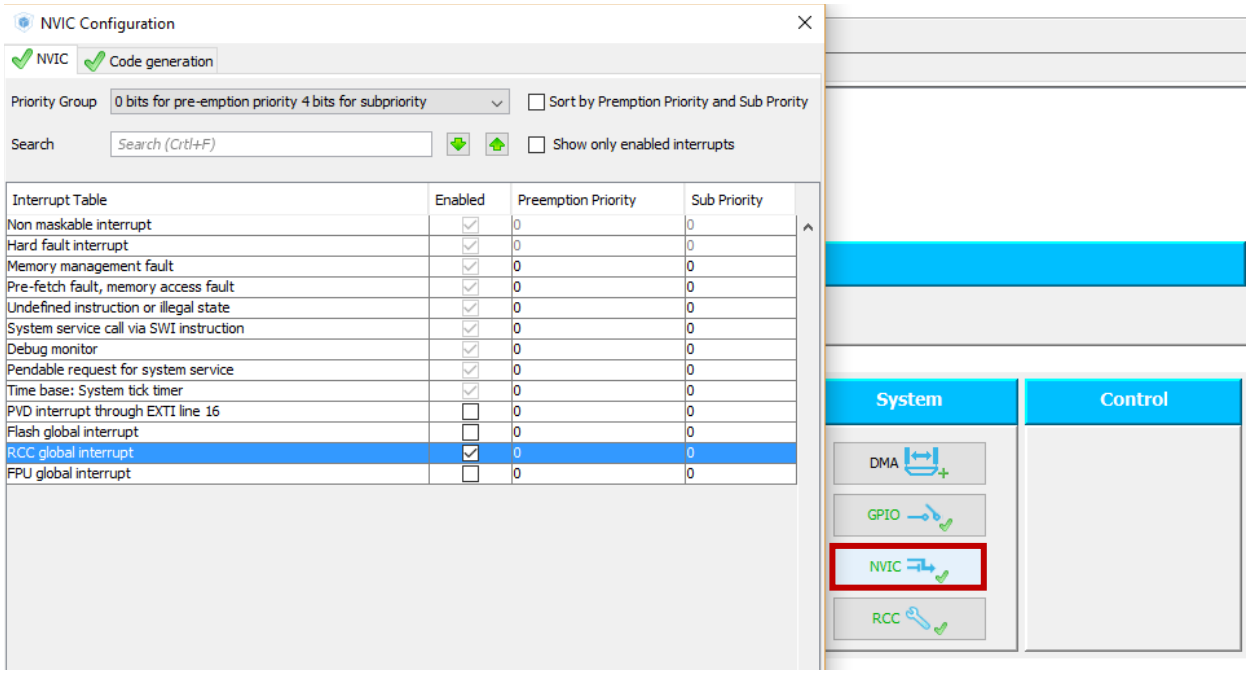



Vous pouvez observer dans la fenêtre **clock configuration** qu'une source d'horloge à 8MHz a été définie. Il reste à configurer les multiplexeurs et le diviseur. En modifiant la valeur du pré-scaleur M de 16 à 8, vous devez obtenir une horloge SYSCLK à 84MHz comme indiqué sur la figure suivante :




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

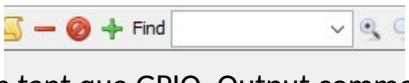
   Il est nécessaire d'autoriser les interruptions au niveau du contrôleur d'interruption NVIC comme indiqué sur la figure suivante, accessible via la fenêtre **configuration** :



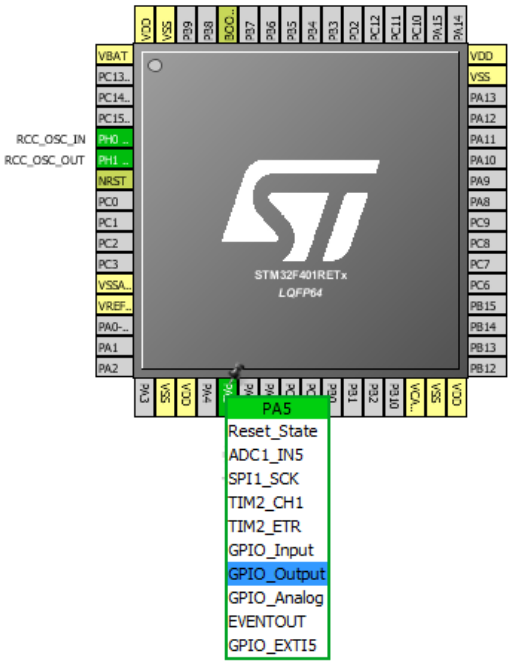

**Apply** and **OK**.

Toujours dans la fenêtre **configuration**, cliquer sur RCC. Sur Vdd Voltage, vous pouvez remarquer que l'on peut directement ajuster la tension d'alimentation du microcontrôleur (1.71<Vdd<3.6V). Dans le sous onglet NVIC Settings, toujours dans le menu RCC, on peut remarquer que les interruptions globales ont été prises en compte (via les actions au point 6).

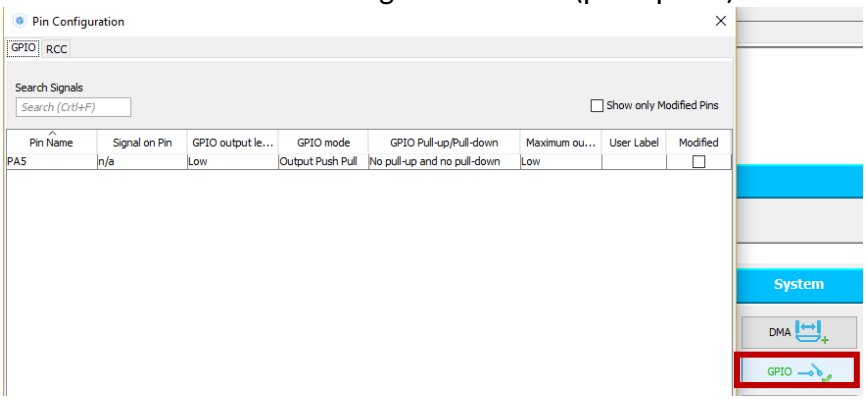
7. On souhaite aussi pouvoir contrôler la LED. Celle-ci est connectée au Port A d'E/S (GPIOA) et à la broche 5 (PA5). Vous pouvez la chercher via le Find :




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

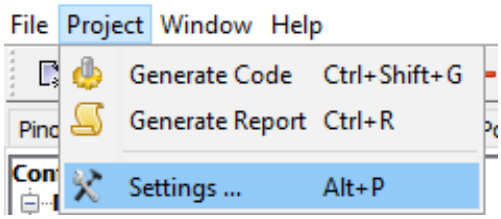



Vous noterez que la broche PA5 possède plusieurs fonctionnalités (pouvant servir à d'autres périphériques comme le TIMER, l'ADC, la liaison SPI …). Une fois configuré, retourner dans la fenêtre **configuration**, puis **GPIO**. Vous noterez comment a été configuré la broche (push-pull…)



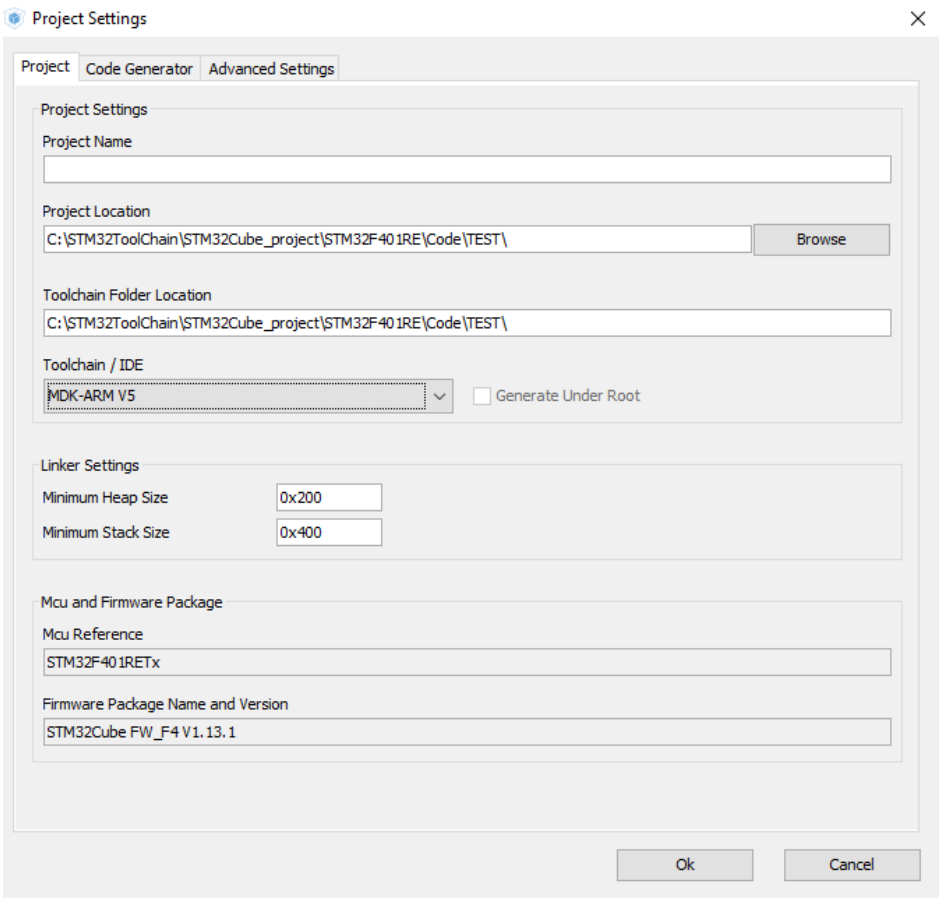
8. *Génération du code et du projet pour Keil uVision*

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

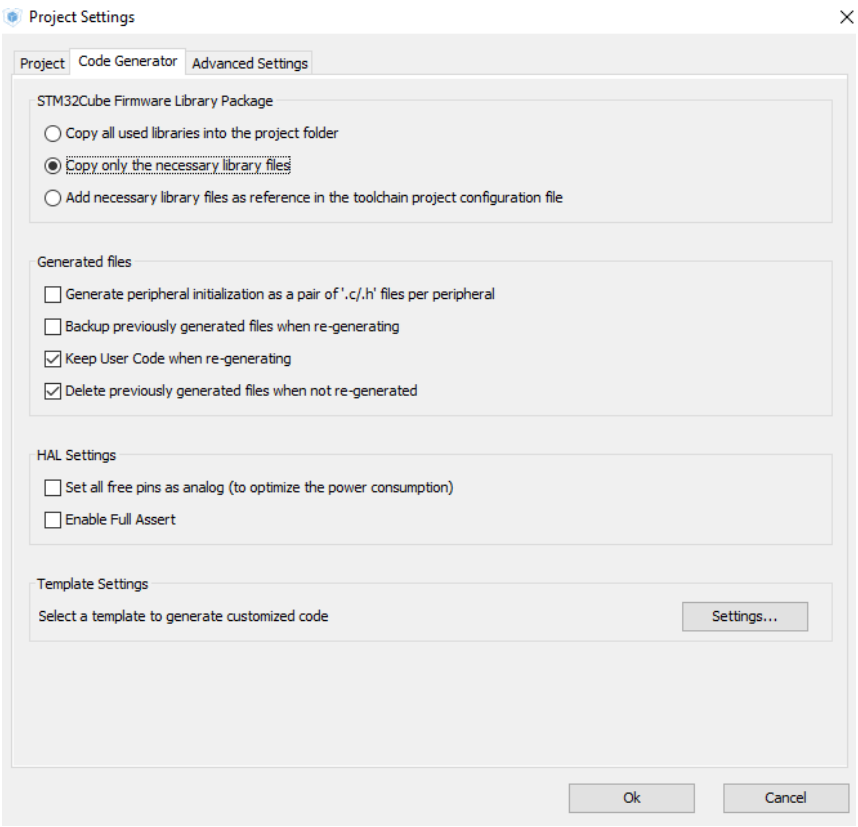



Dans **Project Name** indiquer le nom de votre projet.
Dans **Project location**, indiquer le répertoire de votre projet où sera générer le code.
Dans **Toolchain/IDE**, indiquer **MDK-ARM v5**.
Dans **Linker Settings**, vous pouvez observer la taille des zones de Heap et de Stack (Pile). La zone 'Heap' est une mémoire allouée pour stocker des variables créées dynamiquement au cours de l'exécution. La Pile est une mémoire spécifique permettant la sauvegarde d'information (contenu de registre, adresse de sous-programme, …).

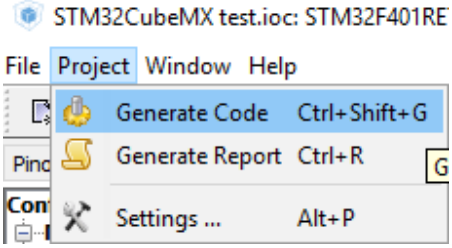
Typiquement, si l'on définit une taille miminale de 0x200, cela revient à réserver 512 octets de mémoire RAM pour la zone de 'heap'.
Pour la Pile, la zone minimale est deux fois plus grande (environ 1Ko).



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




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



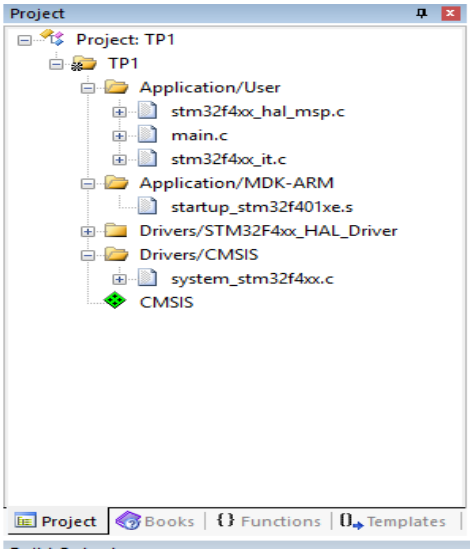

Cliquer ensuite sur **Open Project** pour ouvrir le projet généré sous µVision.

## II. Analyse des fichiers générés dans µVision de Kiel

Avant de s'intéresser à la programmation du microcontrôleur, on se propose d'étudier les fichiers sources qui ont été générés afin de bien comprendre leurs rôles respectifs.

Dans la fenêtre **Project**, vous pouvez observer plusieurs répertoires :

Application/MDK-ARM
Drivers/CMSIS
Drivers/STM32F4xx_HAL_Driver
Application/User




Le répertoire **Application/MDK –ARM** contient un fichier nommé **startup_stm32F401xe.s**.
Question :

-Selon vous à quoi sert ce fichier ? Retrouver les informations suivantes :

.Taille de la Pile ?

.Contenu à l'adresse 0 de la table des vecteurs d'exceptions/interruptions ?

-Quelle action est réalisée lors du reset par le Reset_Handler ?

Dans le répertoire **Drivers/CMSIS,** regarder le fichier nommé **system_stm32f4xx.c.** Vous pouvez observer que ce fichier contient plusieurs fonctions réalisant notamment l'initialisation du CPU, la configuration de l'horloge système (SYSCLK).
Vous aurez noté, que la fonction *SystemInit* est la fonction appelée directement à la suite du reset.

Dans le répertoire **Drivers/STM32F4xx_HAL_Drivers** sont présents les drivers permettant le contrôle de nombreux périphériques du microprocesseur (DMA, Timer, GPIO, …).

Dans le dernier dossier appelé **Application/User**, se trouve 3 fichiers.

-main.c
-stm32f4xx_hal_msp.c
-stm32f4xx_it.c

Question : Que contiennent les fichiers stm32f4xx_it.c et stm32f4xx_hal_msp.c ?

Ouvrer le fichier **stm32f401xe.h** (étendez le fichier main.c)

Question : Que contient ce fichier ? Retrouver le mapping mémoire.

-A quelles adresses commence et termine la FLASH ? la SRAM ? la mémoire allouée pour les périphériques ?
-A quelle adresse commence la zone mémoire pour l'ADC ?

## III. Ajout de fonctionnalités

Comme vous pouvez le remarquer, le code n'effectue aucun traitement spécial après les étapes d'initialisation et de configuration. Nous allons donc rajouter quelques traitements.

### 1) Initialisation de deux tableaux

#### a. Ecriture de la fonction

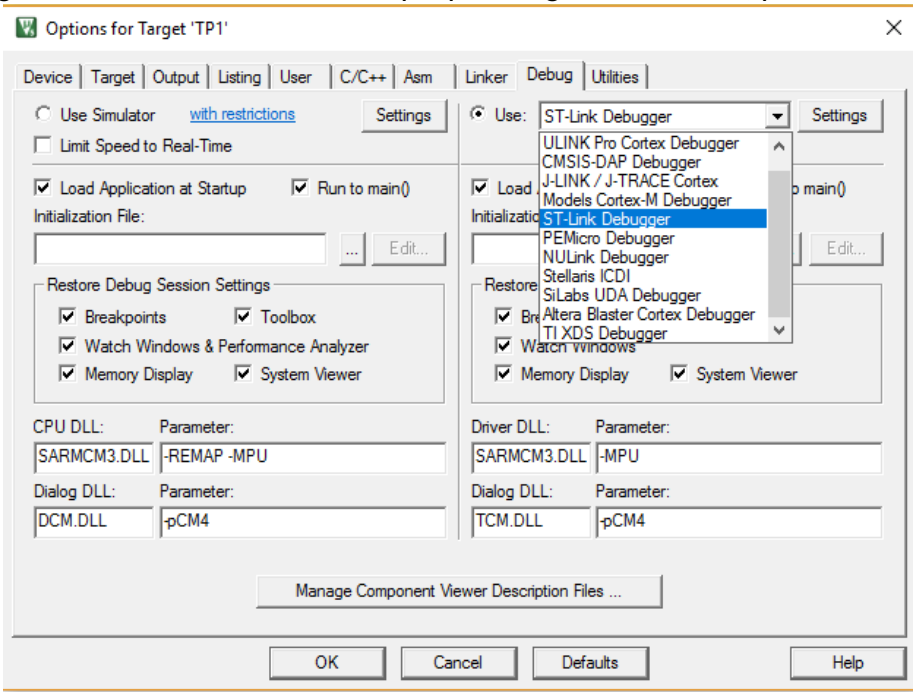
Dans le fichier main.c, réaliser une fonction **void init_tab()** qui réalise l'initialisation de deux tableaux de 10 entiers non signés avec les valeurs suivantes :

Tab0 = {0,1,2,3,4,5,6,7,8,9} ;
Tab1 = {10,11,12,13,14,15,16,17,18,19} ;

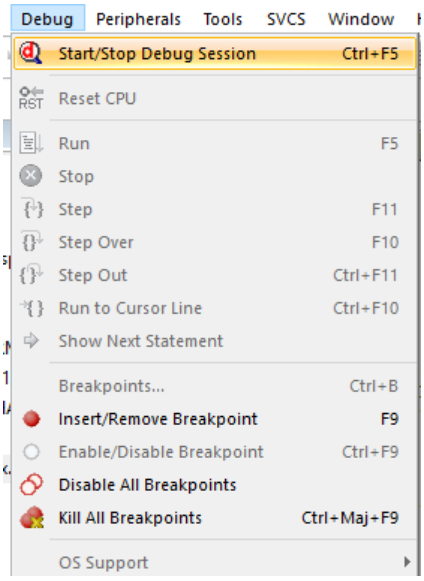
Une fois le code complété, compiler via la touche F7 ou l'icône : 
S'il y a une erreur, consulter les fenêtres **Build Output** et **Error list** pour savoir d'où elle provient et la corriger.
Recompiler jusqu'à ce qu'il n'y ait plus d'erreur. Si c'est le cas, le fichier exécutable a été créé. On peut passer à l'étape de debug.

#### b. Debug

Connecter la plateforme via le lien USB. Il est possible que le driver de debug ne soit pas spécifié. Vérifier dans l'onglet **Flash/Configure Flash Tools,** puis dans **Debug,** vérifier que ST-Link Debugger est bien choisi comme indiqué par la figure ci-dessous, puis OK :



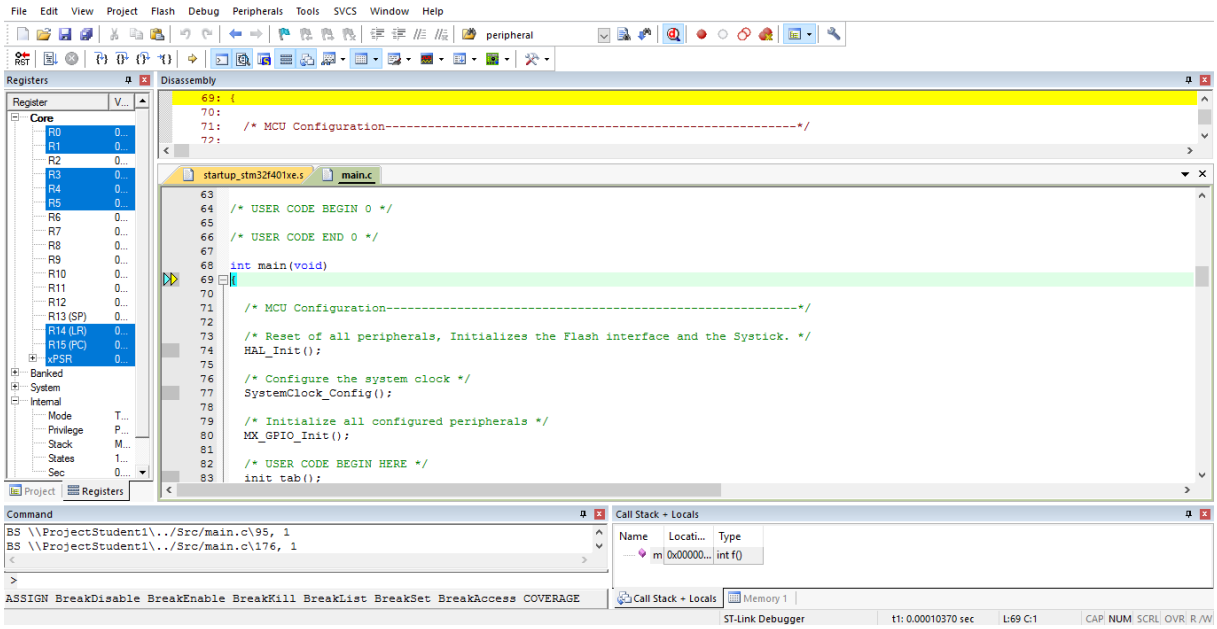
On va maintenant s'intéresser au debugger. Lancer le via l'onglet **Debug/Start-Stop Debug session**.




Un message indiquant l'utilisation de la version d'évaluation (limitant la taille du fichier exécutable) peut apparaître. Cliquer sur OK.

- Les fenêtres de visualisation

Plusieurs fenêtres sont alors visibles comme le montre la figure suivante :




- Registers -> affiche le contenu courant des registres du microcontrôleur.
- Disassembly -> affiche le code assembleur généré à partir de votre code C.
- Command -> permet à l'utilisateur de rentrer des commandes lors du debug

- Call Stack + locals -> permet d'observer l'adresse de la fonction courante exécutée ainsi que les variables locales utilisées (adresses et contenus)

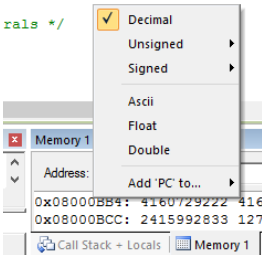
Questions :

-Quelle est l'adresse du registre PC au lancement du debug ? Dans quelle mémoire votre fonction main a-t-elle donc été placée ?
-Même question pour le pointeur de pile (Stack Pointer)?
-Que contient le registre xPSR ?

uVision est très riche en outils permettant le debug efficace d'un code. Des fenêtres supplémentaires peuvent être ajoutées via l'onglet **View**. Par exemple, dans Serial Windows, on peut observer les données envoyées sur la liaison série (UART, printf). µVision dispose aussi d'un analyseur logique **(View/Analysis Windows/Logic Analyzer**) permettant d'observer l'évolution d'un signal au cours du temps.
Lors de l'exécution d'un programme, il peut être intéressant d'enregistrer les traces d'exceptions et d'évènements qui se sont produits durant l'exécution (**View / Trace / Trace exceptions ou Event counters**).
Lors de l'utilisation de périphérique, tous les registres correspondants peuvent être visualisés via **View/System Viewer/**.

-Rajouter une fenêtre de visualisation de la mémoire via **Memory Windows/ Memory 1**.
-Taper dans le champ **Address** , la valeur **PC.**

➔ On peut donc visualiser le contenu des registres à leur emplacement en mémoire ainsi que pour toutes les variables de l'application. A noter que l'affichage des données (decimal, signed, unsigned int/short.. ) peut être modifié via un clic droit dans la zone mémoire.




-Effectuer une nouvelle recherche pour **tab0.**

Questions :

-Dans quelle mémoire sont tab0 et tab1 ?

Mettre un point d'arrêt sur la ligne d'appel à la fonction init_tab() dans le fichier main.c (en cliquant sur la zone grise à gauche du numéro de ligne).

```
68  int main(void)
69  {
70
71    /* MCU Configuration--------------------------------
72
73    /* Reset of all peripherals, Initializes the Flash inte
74    HAL_Init();
75
76    /* Configure the system clock */
77    SystemClock_Config();
78
79    /* Initialize all configured peripherals */
80    MX_GPIO_Init();
81
82    /* USER CODE BEGIN HERE */
83    init_tab();
```

On lance ensuite l'exécution du code jusqu'à cette ligne en cliquant sur F5 ou via l'icône suivante .

On dispose aussi de plusieurs commandes pour effectuer du pas-à-pas :

1) 2) 3) 4)

NB : 1-Step -> permet de rentrer dans la prochaine étape d'exécution de la commande C
2-Step over -> permet de continuer l'exécution jusqu'à la prochaine ligne
3-Step out -> permet de sortie de la fonction courante
4-Run cursor to line -> permet de lancer l'exécution du code jusqu'à la ligne indiquée

Avant toute autre chose, relever la valeur du PC et du LR. Chercher aussi en mémoire l'adresse de init_tab() et la relever.

Avant fonction init_tab() : PC = LR =

Puis rentrer dans la fonction init_tab() en effectuant un pas (icône **Step** ou via la touche **F11**). Noter à nouveau la valeur de PC et LR.

Après entrée dans la fonction PC = LR =

Regarder aussi dans la fenêtre **call stack + Locals** (cf. fenêtre en bas à droite), l'adresse et le contenu de la variable d'index de boucle (si vous avez utilisé une boucle for par exemple).

-Question optionnelle : déterminer dans quel registre de travail est stockée la variable d'index de la boucle

On va maintenant vérifier **en pas-à-pas** (icône **Step Over** ou touche F10) que l'initialisation du tableau se déroule correctement. On pourra vérifier simultanément l'évolution du contenu mémoire via la fenêtre **Memory 1**, **address** = tab0 ou tab1.

A la fin de la fonction init_tab() et avant de revenir au programme principal, noter à nouveau les valeurs de PC et LR.

Fin de fonction init_tab() PC = LR =

Avancer de nouveau à l'aide de **Step Over (F10).**

Retour au programme principal PC = LR =

Conclure sur le fonctionnement du registre PC et du registre LR. Si besoin, reprendre les étapes précédentes en visualisant le code assembleur via la fenêtre **Disassembly.**
(Note BL et BX indique des branchements cf. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0553a/CIHJJEIH.html puis l'instruction set). La remise à 0 du microcontrôleur peut s'effectuer **via Debug/Reset CPU**.

IMPORTANT : si le contenu de tab0 et tab1 à la fin de la fonction d'initialisation ne sont pas corrects, modifier votre code et ré-itérer le debug.

### 2) Additions des contenus des deux tableaux

On souhaite maintenant faire l'addition des deux tableaux tab0 et tab1. Dans le fichier main.c, décrire maintenant la fonction add_tab() qui permet de réaliser l'opération suivante :

Tab1[i] = Tab0[i] + Tab1[i] ;

Compiler et s'il n'y a pas d'erreur, lancer le debugger. Vérifier **en pas à pas**, le bon fonctionnement de votre programme et observer les résultats en mémoire. Pour cela, refaire comme précédemment, placer un point d'arrêt sur lors de l'appel à la fonction add_tab() et exécuter en pas à pas les instructions.

### 3) Détection de somme paire

On souhaite maintenant détecter si la somme du contenu de tab1 est paire. Si c'est le cas, on allumera la led LD2. Celle-ci est connectée physiquement à la broche 5 du port GPIOA (Pin A5).
Pour pouvoir manipuler des éléments externes au coeur, il est nécessaire de configurer les ports d'entrée-sortie. Dans notre cas, il faudra configurer le port GPIOA et la pin 5. Ceci a déjà été réalisé par l'utilitaire STM32CubeMX qui a décrit la fonction MX_GPIO_Init(). On ne cherchera pas forcément à comprendre toute cette fonction (Cours sur les GPIOs à venir).

Dans un premier temps, écrivez la fonction **unsigned int calcul_somme()** qui retourne la somme de tous les éléments de Tab1. Vérifier son fonctionnement.

Dans un second temps, modifier votre code pour mettre à 1 la led si la somme est paire, si non la laisser à 0.

Utiliser la ligne suivante pour allumer la led :

```
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
```

Pour l'éteindre :

```
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);
```

Une fois que tout fonctionne, vous pouvez tester l'autre cas possible. En mode debug, placer vous après la fonction **calcul_somme()**. En accédant directement aux registres et à la mémoire et **sans modifier le code existant,** chercher à ne pas allumer la led.

## IV. Bilan et conclusion

Lors de ce TP, vous avez pu vous analyser les différents fichiers nécessaires à l'initialisation et la configuration du microcontrôleur.
Le debugger est un élément indispensable lors de tout développement de code pour un processeur ou un microcontrôleur. µVision dispose d'un environnement très riche dont vous avez vu quelques possibilités. Ainsi, pouvoir accéder directement aux registres, visualiser le contenu des mémoires, (etc.) sont autant d'éléments indispensables pour bien comprendre, tester et valider une implémentation.