

Laboratoire PTR

Realtime Oszilloskope

Objectifs du laboratoire

Aujourd'hui, de plus en plus les systèmes temps réel sont réalisés au moyen de systèmes embarqués. Dans ce contexte, les systèmes embarqués doivent toujours accomplir plus de tâches. Une partie de ces tâches doivent être accomplies en temps réel (par exemple des acquisitions de données, des fonctions de contrôle). Les autres tâches sont soumises à des contraintes moins critiques en temps (par exemple Communication, entrée et sortie, affichage).

Pendant ce laboratoire, on veut réaliser un simple oscilloscope à l'aide d'une plaque F7-DISCO. Le but est d'échantillonner un signal à l'aide d'une entrée analogique et de l'afficher à l'écran de façon fiable.

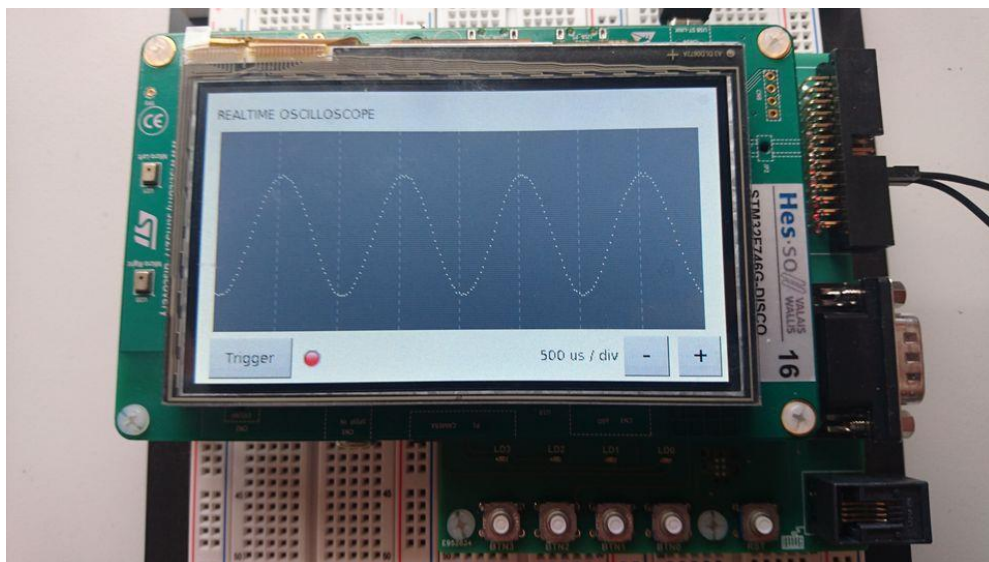




Figure 1: A F7-DISCO based Oscilloscope

Ressources requises

- Le logiciel [STM32CubeMX](#)  est utilisé pour configurer les périphériques du microcontrôleur et pour générer le code pour accéder à ces périphériques au bas niveau (il est déjà installé sur les ordinateurs du laboratoire).
- [System Workbench for Stm32](#)  est un environnement de développement pour systèmes embarqués (aussi installé sur les ordinateurs du laboratoire).
- La plaque de développement **F7-DISCO** et un câble **Micro-USB** pour connecter la plaque de développement à l'ordinateur.
- **Générateur de fonctions.**
- Un fichier avec le nom **work.zip** qui se trouve à l'adresse suivante :

<https://cyberlearn.hes-so.ch/mod/assign/view.php?id=960675>

Ce fichier contient les éléments suivants :

- **src/app** Répertoire avec les classes Factory, Gui et OscilloscopeController
- **src/config** Répertoire avec le(s) fichier(s) de configuration
- **src/mdw** Répertoire de Middleware avec *uGFX* (Display Library) et la librairie *Trace*
- **src/ui-gen** Répertoire avec le code généré (Oscilloscope Display Page)
- **src/platform** Répertoire contenant le code spécifique pour le board et le micro-contrôleur
- **src/xf** XF avec des ports de F7-DISCO pour IDF et FreeRTOS
- **docs** Répertoire avec schémas et datasheets

Outils de développement

STM32CubeMX

Le logiciel [STM32CubeMX](#) de STM est un outil qui sert à configurer les projets pour un environnement de développement et pour un microcontrôleur spécifique.

Il assiste le développeur avec le choix et la configuration des différents périphériques et de les associer aux broches possibles du microcontrôleur. En plus, il est possible de configurer les horloges internes et de finalement générer le(s) fichier(s) du projet pour un IDE spécifique.

Il faut aussi mentionner que ce logiciel peut être utilisé après avoir généré un projet, donc en cours de développement. Ceci au cas où il est nécessaire d'amener des changements concernant les périphériques. Le générateur de code ne réécrit que le code généré par lui-même et laisse le code écrit par le développeur intouché car ce code est placé dans des zones expressément prévues pour. La figure suivante montre une telle zone :

```
3⊕ * @file          : main.c
48 /* Includes -----*/
49 #include "main.h"
50 #include "stm32f7xx_hal.h"
51 #include "cmsis_os.h"
52
53 /* USER CODE BEGIN Includes */
54
55
56 /* USER CODE END Includes */
57
```

Figure 2: STM32CubeMX 'user code' section

Le générateur de code de STM32CubeMX ne touche pas au code inscrit entre les balises 'USER CODE BEGIN' et 'USER CODE END'.

STM32CubeMX peut être exécuté sur toutes les plateformes communes (Windows®, Linux® et macOS®).

System Workbench for Stm32

[System Workbench for Stm32](#) (abrévié **SW4STM32**) est un environnement de développement basé sur Eclipse qui a été (et l'est toujours) développé par la société AC6 en collaboration avec STMicroelectronics. La chaîne d'outils GNU Toolchain et OpenOCD est gérée de façon automatique par SW4STM32.

System Workbench for Stm32 est aussi disponible pour toutes les plateformes communes (Windows®, Linux® et macOS®).

Schéma bloc

Le schéma bloc suivant sert comme base pour l'utilisation des composants matériels (périphériques) du microcontrôleur :

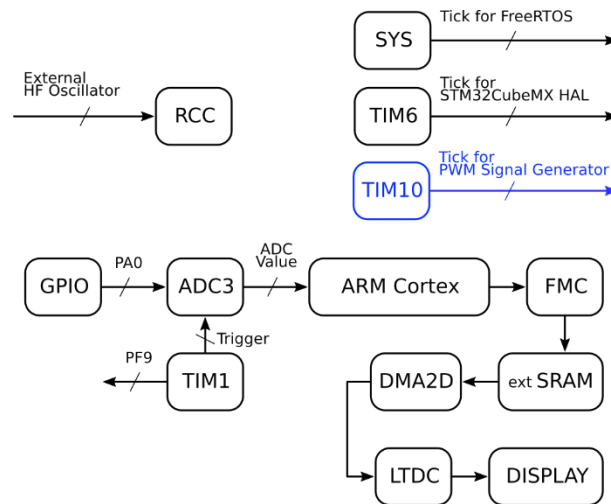


Figure 3: Schéma Bloc

Le ADC3 échantillonne et lit le signal en passant par le GPIO PA0. Le microcontrôleur trace ensuite le signal à l'aide de l'écran LCD.

Le schéma bloc peut être étendu librement.

Tâches

Tâche 1 – Projet STM32CubeMX

Dans STM32CubeMX nous allons créer un nouveau projet avec le micro-contrôleur (MCU) STM32F746NG. Vu que nous utilisons un Discovery Board comme base de notre système embarqué, vous pouvez directement cliquer sur l'onglet *Board Selector* en haut à gauche et chercher l'élément 32F746GDISCOVERY.

Afin de créer le projet, il faut cliquer sur le board souhaité et ensuite sur *Start Project*.

Sur la fenêtre suivante on vous demandera si les périphériques du MCU devraient être initialisés selon les valeurs de défaut pour le board sélectionné.

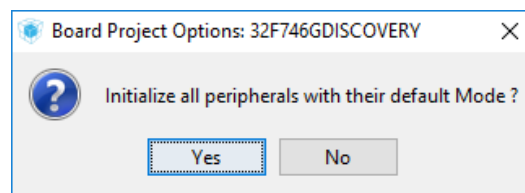


Figure 4: Project Creation – Init Peripherals

Répondez « Oui », ensuite les clocks et le LCD seront déjà configurés correctement.

Une description/schéma du Discovery Board, de la carte d'extension (avec boutons et LEDs) et le datasheets du micro-contrôleur se situent dans le répertoire *docs* dans *work.zip*.

Tâche 2 – Éteindre USB, FatFS et Ethernet

Pour éviter que le projet devienne trop grand, vous devriez éteindre tout ce qui est USB, FatFS et Ethernet. Les périphériques permettant ces fonctionnalités devraient aussi être éteints, afin de réduire considérablement le nombre de fichiers à compiler !

Tâche 3 – Configuration du RCC

Tout d'abord les clocks du MCU doivent être réglés. Pour cela il existe le périphérique RCC (**R**eset and **C**lock **C**ontrol).

Vu que les périphériques sont configurés automatiquement, on ne doit rien faire de plus ici.

Vérifiez que le High-Speed-Clock externe (HSE) est réglé sur 25MHz et que la PLL est utilisée.

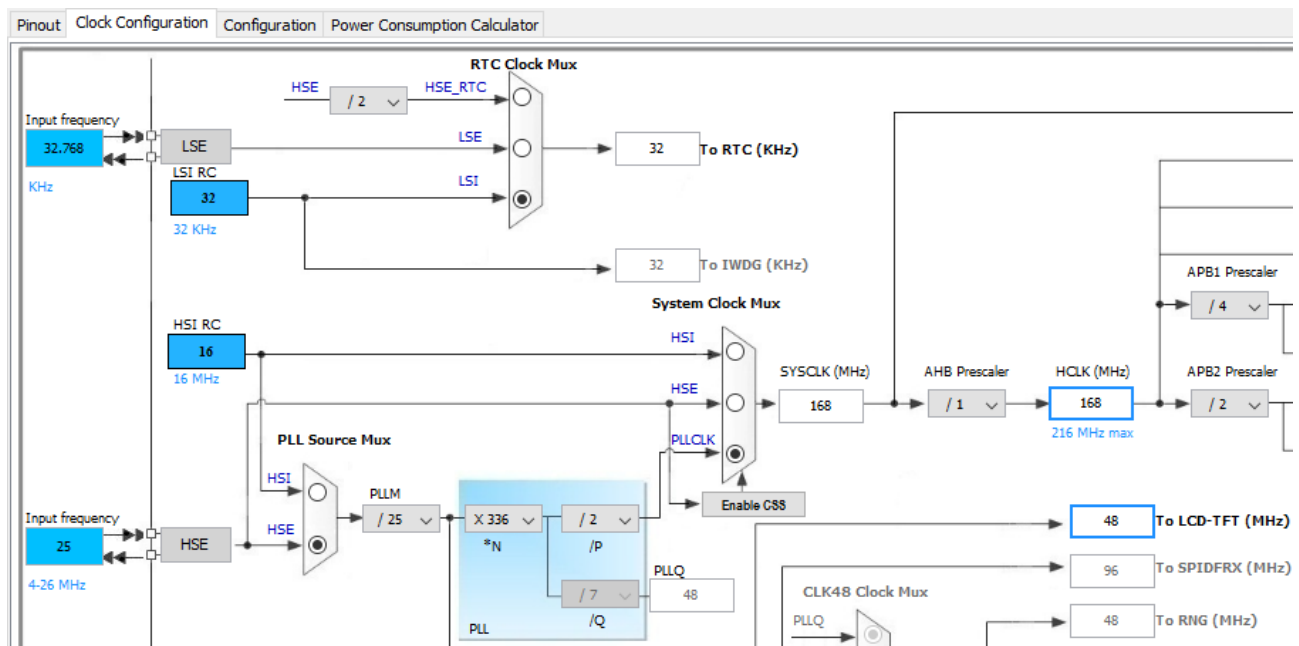


Figure 5: Clock Configuration

Tâche 4 – Configuration de la génératrice de code de CubeMX

Dans cette étape, nous allons configurer la génération de code. Pendant ce pas le nom du projet sera aussi défini (dans notre cas un projet Eclipse) pour lequel le code sera généré en partie par la chaîne d'outils.

Pour effectuer ceci, choisissez **Settings** dans le menu **Project...**

1. Indiquez le nom du projet (par exemple *SW4STM32-RealtimeOscilloscope*)
2. Indiquez le chemin / le répertoire où le projet sera enregistré
3. Comme chaîne d'outils, choisissez *SW4STM32*

La configuration du projet devriez donc s'afficher à peu près comme suit :

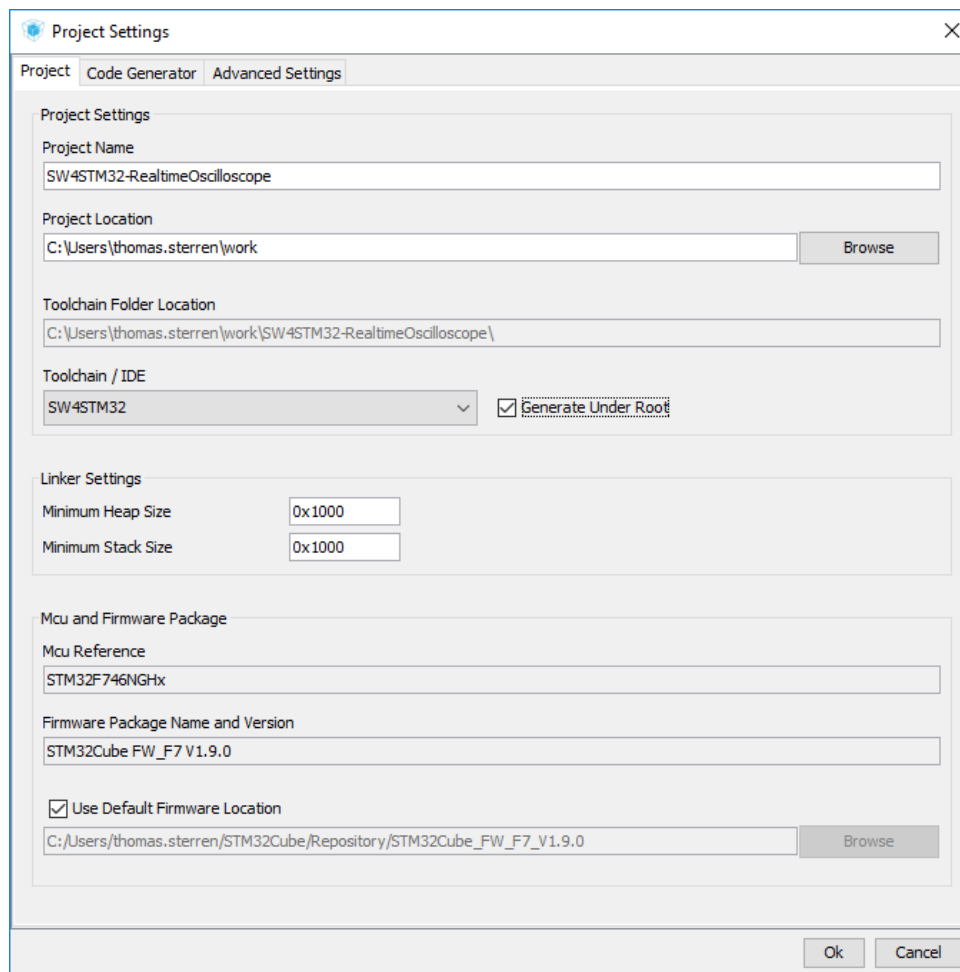


Figure 6: Configuration de la génératrice de code

Rien ne doit être changé sous les onglets **Code Generator** et **Advanced Settings**.

Ensuite, le code peut être généré en choisissant le menu **Project->Generate Code...**. Ouvrez ensuite le projet à l'aide de **System Workbench for STM32** et inspectez le code généré. Pour faire ainsi, vous pouvez cliquer le bouton **Open Project** :

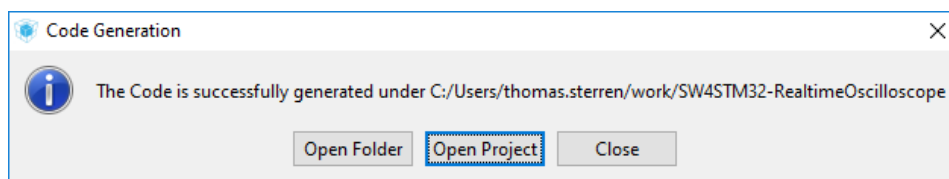


Figure 7: Open Project Window

Entre autre, le logiciel STM32CubeMX a aussi généré un nouveau fichier de projet (.ioc) dans le répertoire choisi. Si vous allez faire des changements concernant la configuration du HAL, dorénavant faut-il uniquement utiliser ce fichier.

Si vous avez déjà enregistré une fois le projet STM32CubeMX, vous allez retrouver deux fichiers .ioc. Le fichier qui avait été enregistré avant la génération du code peut être effacé dans ce cas-là.

Tâche 5 – Build et Debug

Ensuite nous devrons compiler le projet, télécharger sur la cible et déboguer. La création du fichier de programme devrait fonctionner sans problème.

Optimisations de projet:

- Convert to C++ (Clic droit sur Project)
- Enable parallel build (Dans les propriétés du projet)

Pour déboguer, une configuration de Debug doit d'abord être établie. Allez dans le menu **Run->Debug Configurations...** et créez une configuration de debug basée sur AC6 :

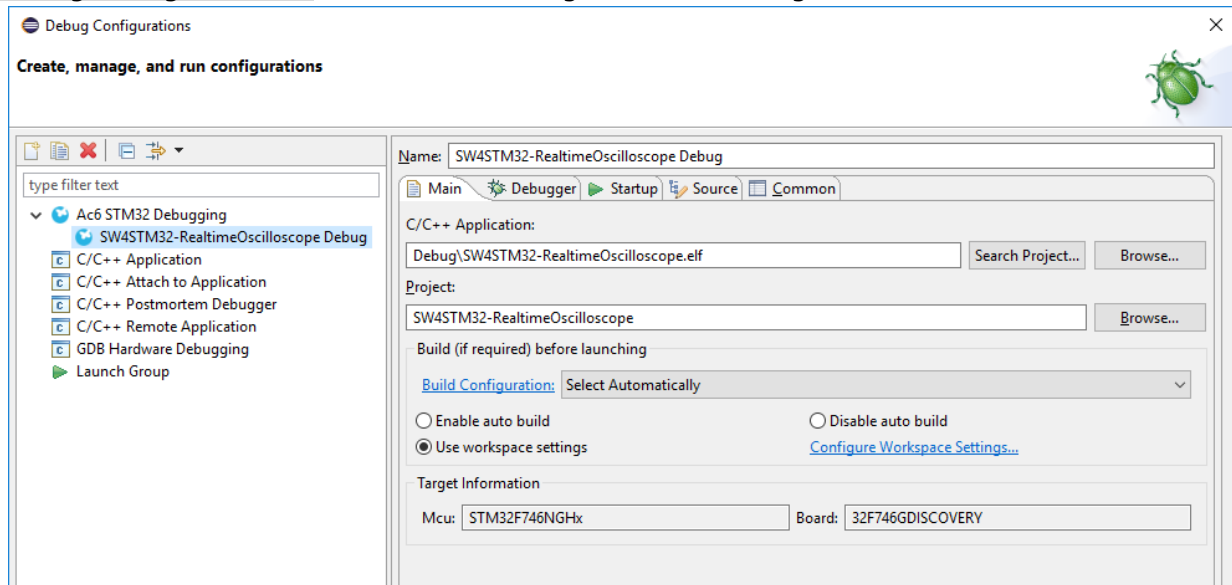


Figure 8: Debug Launch Configuration

Vérifiez si le programme peut être chargé et débogué sur la cible en cliquant sur le bouton **Debug**. Si tout va bien, vous allez vous retrouver dans la fonction `main()` après un petit moment et pour continuer il faut cliquer sur le bouton **Resume**.

Tâche 6 – Configuration du Timing

Comme prochaine tâche, nous voulons analyser les temps d'exécution des différents composants du logiciel. En principe nous avons deux composants qui peuvent être exécutés à deux vitesses différentes :

#	Component	Timing
1	Conversion du signal analogue	1 kHz ou plus
2	Rafraichissement de l'écran	20 à 60 fois par seconde

Question 1 : Est-ce qu'il est possible d'exécuter le composant numéro 1 avec un XF ou bien faut-il un RTOS ? Justifiez votre réponse.

Question 2 : Est-ce qu'il est possible d'exécuter le composant numéro 2 avec un XF ou bien faut-il un RTOS ? Justifiez votre réponse.

Question 3 : Si l'on combine un timer hardware avec un XF, lequel des deux doit être priorisé ? Justifiez votre réponse.

Tâche 7 – Générateur de Signale PWM

Dans le prochain chapitre, nous utiliserons le convertisseur analogique/numérique (ADC) pour mesurer un signal externe. Si vous n'avez pas un générateur de signaux, nous vous proposons ici de générer un signal PWM pour remplacer le générateur de signaux manquant. Ce qui suit explique comment configurer une timer hardware pour sortir un signal PWM sur un pin du microcontrôleur.

Pour cela, nous utilisons le timer matériel TIM10, qui peut émettre un signal PWM sur le pin PB8. Configurez le TIM10 dans le projet CubeMX comme suit :

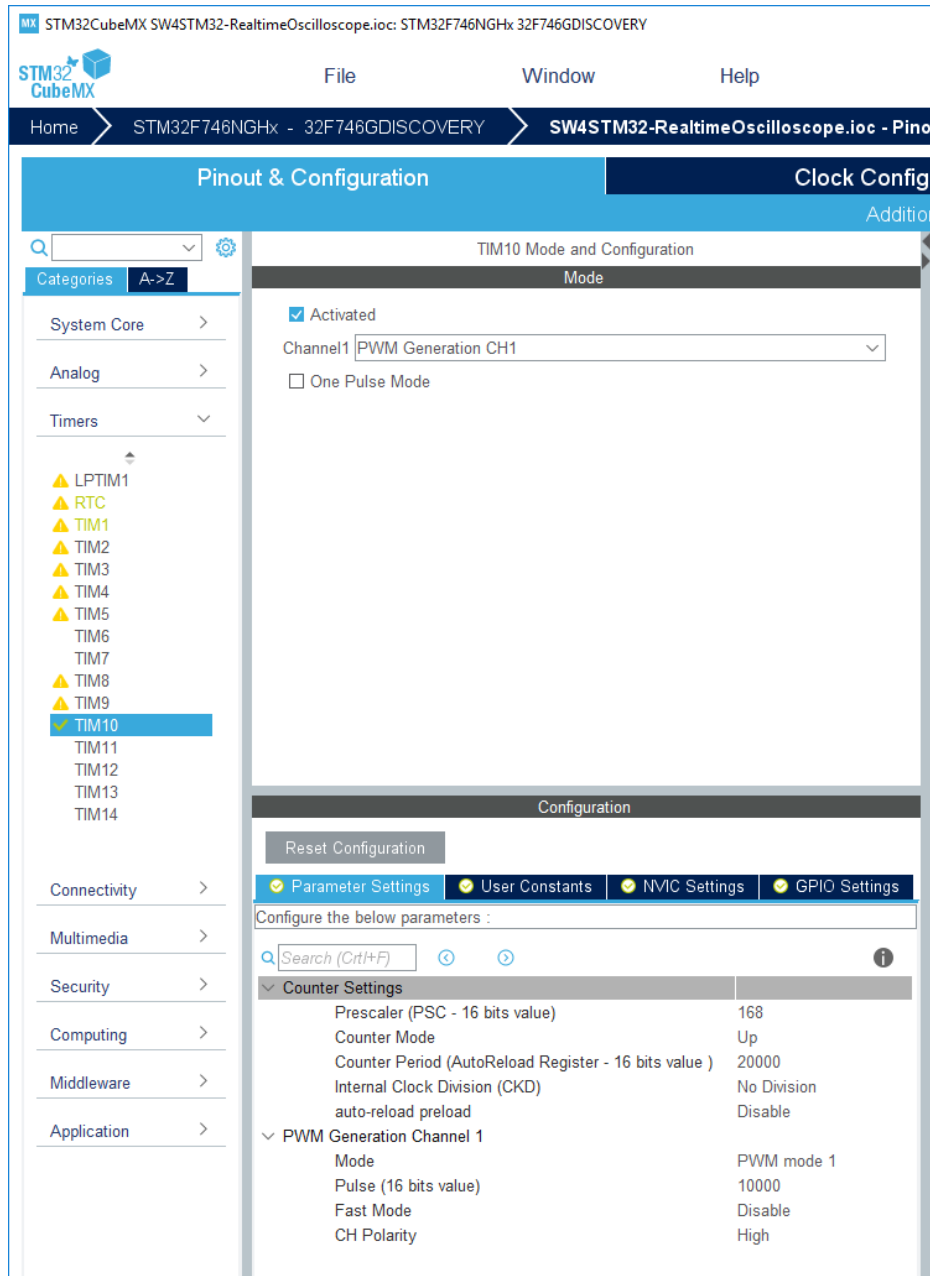


Figure 9: TIM10 Configuration

Avec cette configuration, un signal PWM de 50 Hz est émis sur le pin PB8. Le pin PB8 est tiré sur la carte d'extension sur le pin 6 du connecteur mezzanine :

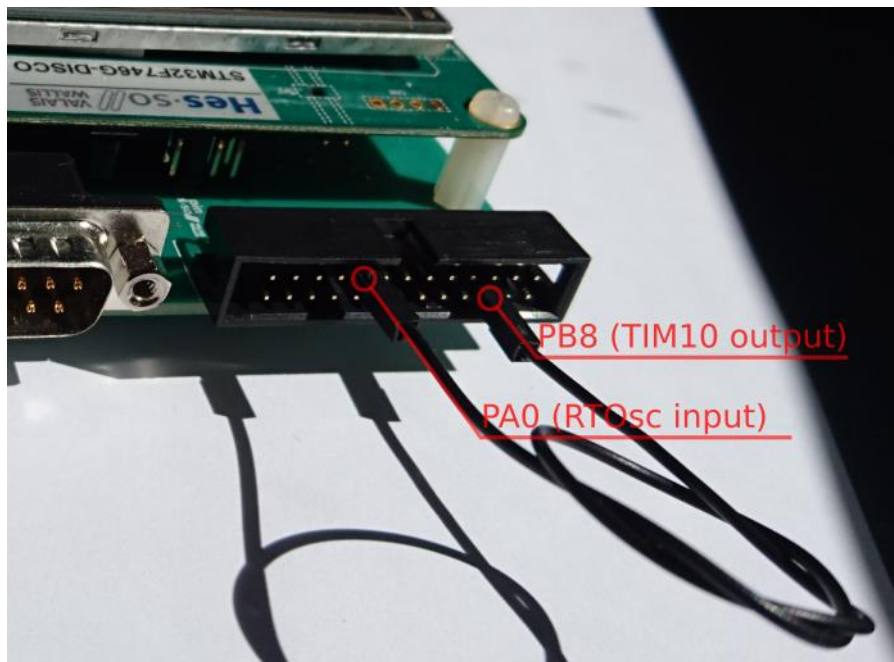


Figure 10: TIM10 Output to PB8 pin

Pour démarrer le timer matériel TIM10, vous devez ajouter le code suivant à votre projet :

```
HAL_TIM_PWM_Start(&htim10, TIM_CHANNEL_1);
```

Vous pouvez maintenant utiliser ce signal comme signal d'entrée pour l'ADC.

Tâche 8 – Configuration de l'ADC (Software Triggered)

Pour commencer, on désire pouvoir mesurer des signaux jusqu'à une fréquence de 1 kHz.

Question 1 : Combien de mesures [Samples/s] le convertisseur A/D doit-il effectuer par seconde pour pouvoir échantillonner des signaux avec des fréquences jusqu'à 1 kHz ?

Question 2 : Faut-il un filtre ? Si oui, quelle sera la fréquence de coupure de ce filtre ?

Question 3 : Est-ce que la fréquence donnée par le théorème d'échantillonnage ou devrait-elle être plus élevée ?

Pour notre démarche, il nous faut le périphérique ADC3 du microcontrôleur. Les périphériques ADC1 et ADC2 ne seront pas utilisés dans ce projet.

Question 4 : Lequel des canaux du ADC3 doit être utilisé pour pouvoir mesurer / échantillonner le signal à l'aide de la broche *PA0* ?

Utilisez le projet STM32CubeMX pour configurer le ADC3. Vous pouvez effectuer les changements dans les onglets *Pinout* et *Configuration*. Lisez dans le manuel *stm32f746xx-reference-manual.pdf* le chapitre 15 (ADC). Vous devriez au moins maîtriser les termes suivants :

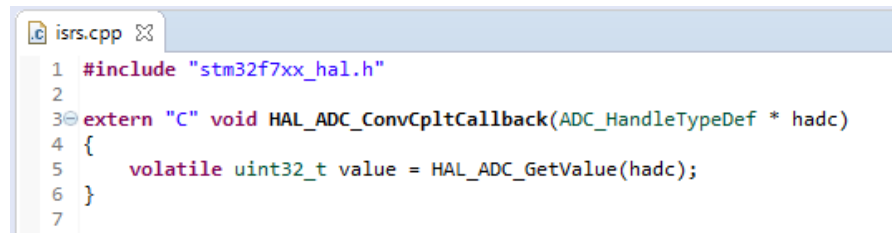
- Single or Continuous Conversion Mode
- Scan Mode
- Discontinuous Mode

Pour commencer, nous allons démarrer la conversion par le logiciel et lire le résultat dans le ISR (Interrupt Service Routine) correspondant.

Le HAL du CubeMX met à disposition une routine d'interruption. Cette dernière est appelée à la fin de chacune des conversions. La routine porte le nom `HAL_ADC_ConvCpltCallback()` et il

est possible de l'adapter à ses propres buts. Pour cela créez le fichier *isrs.cpp* dans le répertoire *Src*.

Dans ce fichier, implémentez la fonction et le code permettant de réaliser la lecture du ADC :

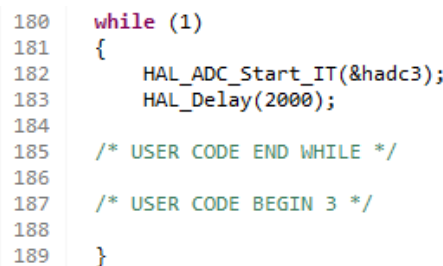


```
1 #include "stm32f7xx_hal.h"
2
3 extern "C" void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef * hadc)
4 {
5     volatile uint32_t value = HAL_ADC_GetValue(hadc);
6 }
7
```

Figure 11: ADC Callback Function

Pour démarrer la conversion, on doit appeler la fonction `HAL_ADC_Start_IT()`.

Voici un exemple :



```
180 while (1)
181 {
182     HAL_ADC_Start_IT(&hadc3);
183     HAL_Delay(2000);
184
185     /* USER CODE END WHILE */
186
187     /* USER CODE BEGIN 3 */
188
189 }
```

Figure 12: `HAL_ADC_Start_IT()` in *main.c*

Testez, si vous pouvez lire la valeur échantillonnée dans la routine d'interruption.

Question 5 : Est-ce que le ADC pourrait-il éventuellement effectuer des mesures à des intervalles réguliers à l'aide de ses propres moyens ?

Attention : Limitation tension d'entre

Remarquez bien que la génératrice des signaux sort un signal avec une amplitude de 5 volts. Les GPIO du microcontrôleur tolèrent des tensions jusqu'à 5 volts. Des tensions plus grandes vont endommager les périphériques du microcontrôleur !

Tâche 9 – Configuration du timer Hardware (TIM)

Maintenant qu'on est capable de lire les valeurs AD, on souhaiterait maintenant le faire à des intervalles réguliers. Nous pouvons vérifier s'il est possible

Da wir jetzt im Stande sind AD-Werte einzulesen, möchten wir dies nun in einem regelmässigen Zeitintervall durchführen. Comme vous l'avez déjà peut-être découvert il n'est pas possible de faire 1000 mesures par seconde avec le XF ou un RTOS. Pour le faire, un timer hardware doit être utilisé, et ce timer va démarrer la conversion (trigger).

Pour cela nous allons utiliser le périphérique TIM1 du MCU. Le but est de laisser tourner le timer hardware à une fréquence de base de **1MHz**. Ensuite nous devons régler la période du compteur afin de pouvoir configurer des fréquences plus basses que 1MHz.

Pour vérifier la fréquence, on utilisera l'interrupt Output-Compare et dans la routine de service des interruptions on va piloter un GPIO, ensuite on peut contrôler la fréquence avec un oscilloscope.

Prenez du temps à lire le Chapitre 22 (TIM) sur le timer Hardware dans le datasheet *stm32f746xx-reference-manual.pdf*.

1. Configurez le TIM1 pour qu'il fonctionne à une fréquence de base de 1MHz
2. Réglez la période du timer pour une fréquence de 10kHz

3. Configurez l'interruption *Capture-Compare* du TIM1
4. Implémentez une fonction de callback dans le fichier *Src/isrs.cpp*
`HAL_TIM_OC_DelayElapsedCallback()`
5. Configurez le GPIO **PF9** en tant que sortie, afin de pouvoir l'utiliser dans la fonction de callback.
6. Vérifiez le temps d'intervalle du TIM1 à l'aide de l'oscilloscope.

Remarque :

Si vous n'avez pas d'oscilloscope sous la main, ne tenez pas compte des deux derniers points 5. et 6.

Tâche 10 – Configuration du ADC (Timer Triggered)

Maintenant nous avons un timer hardware qui peut être utilisé pour déclencher la lecture du convertisseur AD.

1. Modifiez la configuration du ADC dans le projet STM32CubeMX afin de démarrer la conversion avec le TIM1 (en hard maintenant et non en soft). Utilisez le registre *Capture Compare* pour ceci.
2. Dans la configuration du TIM1, vérifiez que la sortie bascule aussi.
3. On doit démarrer le timer TIM1 avec la fonction `HAL_TIM_OC_Start_IT()` dans le *main.c*.
4. La fonction `HAL_ADC_Start_IT()` doit seulement être appelée une fois dans la fonction *main()* afin d'enclencher l'interruption du ADC. Alors n'oubliez pas de mettre la fonction `HAL_ADC_Start_IT()` en commentaire ou de la supprimer dans la boucle while du *main()*.

Tâche 11 – XF Integration**XF Package**

Dans la prochaine étape nous voudrions intégrer le XF inclus dans le *work.zip* dans le projet. Copier alors le répertoire *src* au complet dans le répertoire où votre projet *SW4STM32-RealtimeOscilloscope* se trouve :

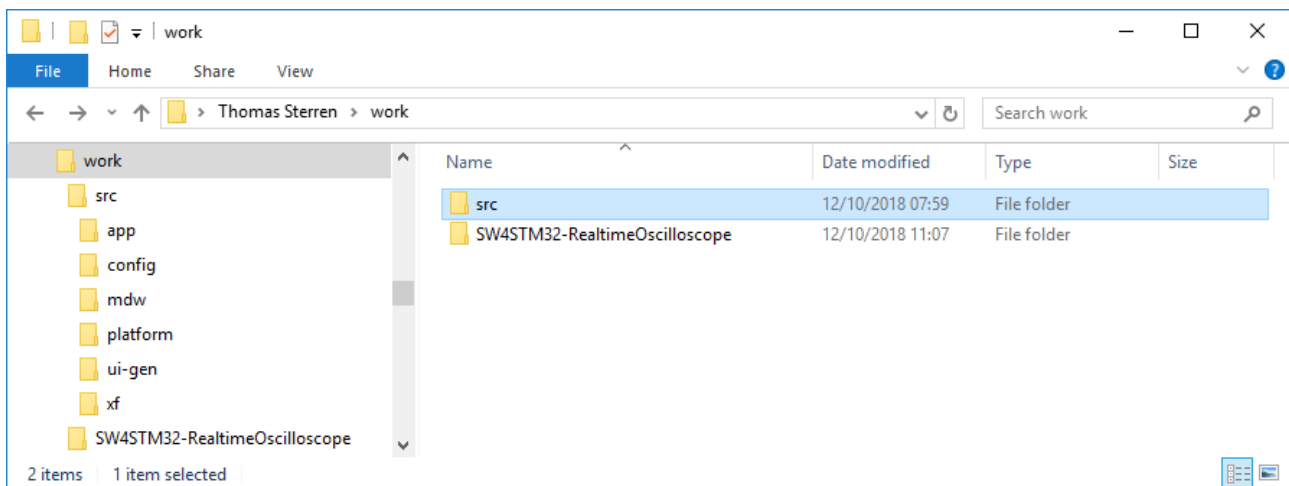


Figure 13: Work Folder Structure

Pour référencer le dossier *xf* dans le projet *SW4STM32*, un nouveau répertoire doit être créé dans le projet dont l'option *Linked Folder* doit référencer le dossier *src/xf* :

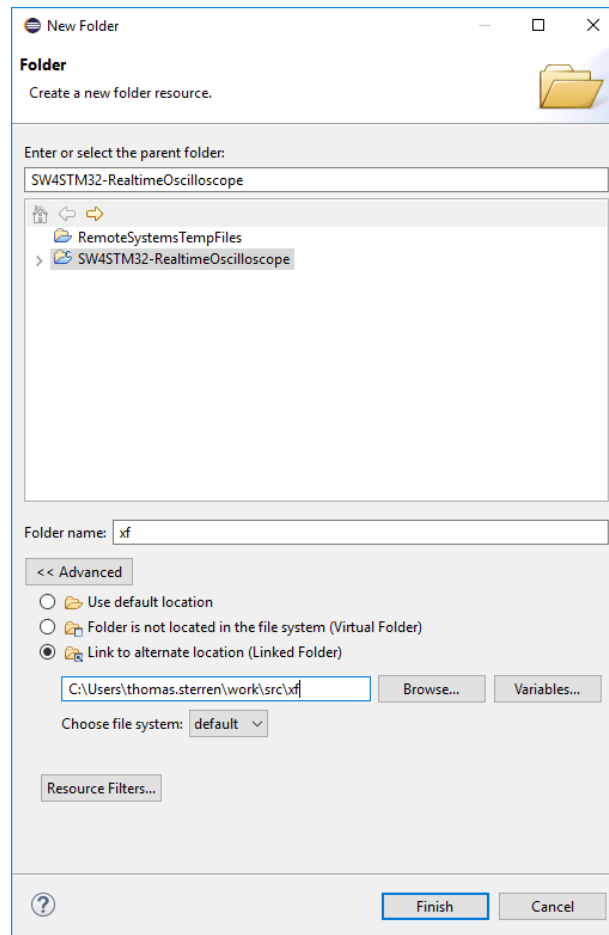


Figure 14: Project Link Creation to src/xf Folder

Dans la prochaine étape on doit spécifier les sous-répertoires du XF, qui seront compilés. Pour cela on spécifie ces répertoires comme en tant que *Source Folders*:

- *xf/core*
- *xf/include*
- *xf/port*
- *xf/port/default-idf*

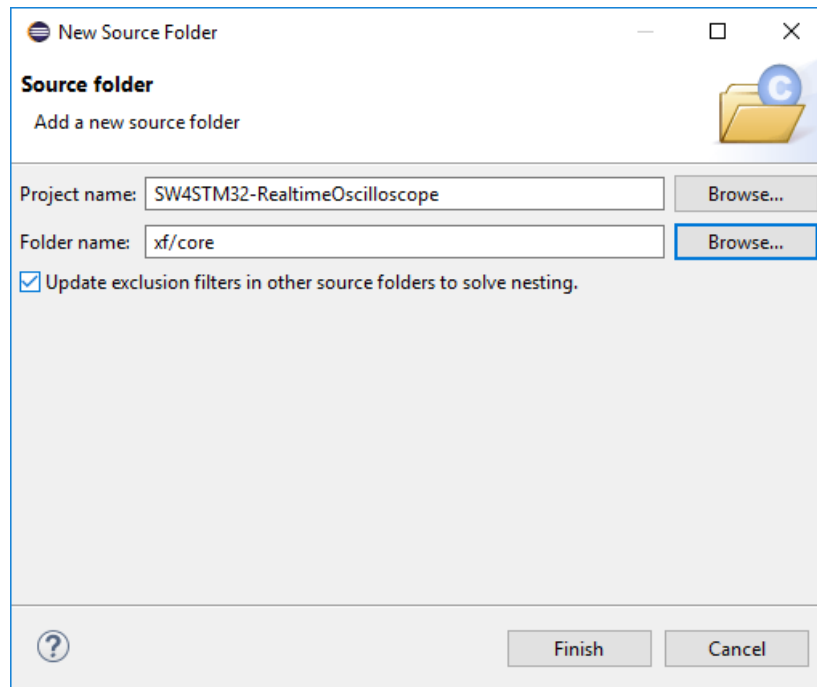


Figure 15: Example – Add 'xf/core' as Source Folder

Dans les propriétés du projet sous *Include* on doit rajouter les chemins d'accès suivants :

```
"${ProjDirPath}"
"${ProjDirPath}\..\src"
"${ProjDirPath}\..\src\config"
"${ProjDirPath}\..\src\xf\include"
"${ProjDirPath}\..\src\xf\port"
"${ProjDirPath}\..\src\xf\port\default-idf"
```

XF Package Dependencies

Le XF nécessite les composants software de *mcu* et *trace*:

Component	Location
mcu	src/platform/f7-disco-gcc
trace	src/mdw

Créez les liens entre les répertoires *src/platform* et *src/mdw* et le projet.

Rajoutez le répertoire *platform/f7-disco-gcc* aux *Source Folder* du projet.

Rajoutez les éléments suivants aux chemin d'accès *Include* du projet :

```
"${ProjDirPath}\..\src\platform\f7-disco-gcc"
"${ProjDirPath}\..\src\platform\f7-disco-gcc\mcu"
"${ProjDirPath}\..\src\mdw"
```

XF Tick in SysTick_Handler

Le XF nécessite un timer hardware pour que le port IDF gère les timeouts. Le timer SysTick est déjà réglé pour une période de 1 ms dans STM32CubeMX. Nous utiliserons celui-ci afin de faire fonctionner le XF.

Rajoutez le code suivant dans le fichier *Src/stm32f7xx_it.c* dans la fonction `SysTick_Handler()`:

```
// SysTick handler gets called every millisecond (is given by code
// generated by STM32CubeMX). Check which interval is needed by the
// XF (typically slower) and call XF_tick() accordingly.
#if (PORT_IDF_STM32CUBE != 0)
    if ((HAL_GetTick() % XF_tickIntervalInMilliseconds()) == 0)
    {
        XF_tick();
    }
#endif // PORT_IDF_STM32CUBE
```

N'oubliez pas de relier le fichier header *xf.h*:

```
#include "xf/xf.h"
```

Board Dependencies

Dans le dossier *platform/board* la classe `LedController` utilise les LEDs de la carte d'extension. Dans le projet STM32CubeMX les GPIO User Label suivants doivent être assignés aux GPIOs correspondants :

User Label	GPIO Name
LED0	PA15
LED1	PH6
LED2	PA8
LED3	PB4

Maintenant l'intégration du XF dans le projet est terminée. Vous devriez pouvoir compiler et linker le projet sans erreurs.

Tâche 12 – Application

Application Package

Prochainement nous allons rajouter le répertoire *app* au projet. Procédez de la même manière qu'auparavant: rajouter au projet et définir comme *Source Folder*.

Le répertoire *app* contient trois classes : `Factory`, `Gui` et `OscilloscopeController`.

La classe `Factory` est responsable pour la création et l'initialisation des composants nécessaires pour le programme.

La classe `Gui` représente le display de l'oscilloscope. Elle est responsable de l'affichage sur le display et elle gère les événements tactiles du display.

La classe `OscilloscopeController` est responsable du comportement de l'oscilloscope. Elle décide quelles données afficher et à quel moment.

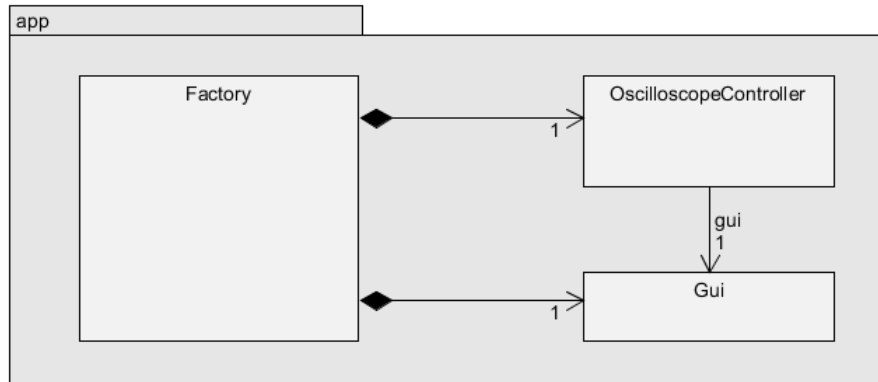


Figure 16: App Package – Class Model Diagram

Le XF et la classe Factory doivent être appelées/intégrées dans le *Src/main.c* généré. Les fonctions C correspondants pour initialiser, construire et démarrer sont accessibles. Intégrez les deux classes de cette manière dans le fichier *main.c*.

Les fonctions rajoutées avant pour démarrer l'ADC et le TIM peuvent être effacés. Celles-ci sont désormais appelées par la classe **Factory**.

Display Packages

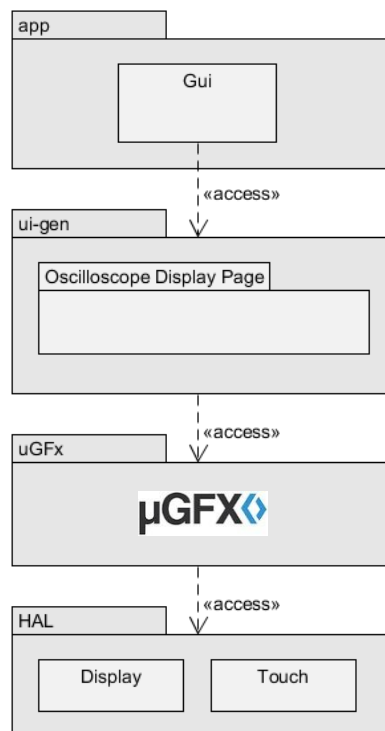


Figure 17: GUI Software Layers

Les répertoires suivants doivent être rajoutés au projet afin de pouvoir continuer à compiler :

Répertoire	Description
mdw/ugfx	Display library
ui-gen	Display pages (Oscilloscope Graphic Elements)

Les paths suivants doivent être rajoutés comme des *Includes Paths* pour le compilateur C/C++ :

```
"${ProjDirPath}..\src\config"  
"${ProjDirPath}..\src\xf\include"  
"${ProjDirPath}..\src\xf\port"  
"${ProjDirPath}..\src\mdw\ugfx"  
"${ProjDirPath}..\src\mdw\ugfx\boards\base\STM32F746-Discovery"  
"${ProjDirPath}..\src\mdw\ugfx\drivers\gdisp\STM32LTDC"  
"${ProjDirPath}..\src\mdw\ugfx\src\gdisp\mcufont"  
"${ProjDirPath}..\src\ui-gen"
```

...

PWM Signal Generator

Sur la vue de l'oscilloscope sur l'écran LCD, il y a des boutons bleus pour changer le signal de sortie du générateur de signaux PWM entre 2 fréquences :

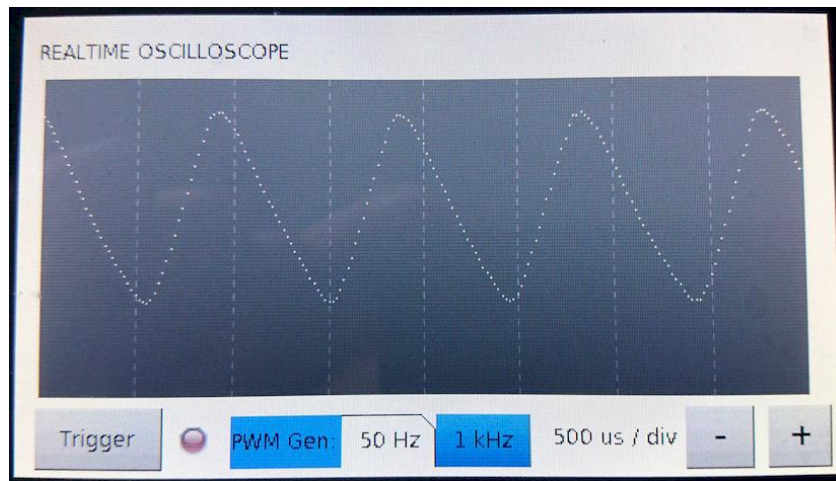


Figure 18: PWM Signal Generator Controls

Pour l'activer, mettez le define `ENABLE_SIGGEN_PWMGENERATOR` dans le fichier `config/siggen-pwmgenerator-config.h` à **1**.

Tâche 13 – Oscilloscope GUI

La classe `OscilloscopeController` est responsable d'afficher les valeurs lues du ADC.

La méthode `doShowAnalogSignal()` est régulièrement appelée par la machine d'états. Créez un array, dans lequel les valeurs lues du ADC sont chargées. Afficher les valeurs lues sur le display.

Tâche 14 – Sample-Rate Tuning

Essayer d'incrémenter la fréquence d'échantillonnage (Sampling-Rate).

Question 1: Quelle fréquence d'échantillonnage peut être atteinte?

Question 2: Quel(s) composant(s) limite(nt) le système?

Tâche 15 – RTOS Integration

Dans la prochaine étape nous analysons comment notre application se comporte avec un système d'opération.

Le XF ainsi que la librairie uGFX supportent FreeRTOS. Activez le système d'opération FreeRTOS dans le projet STM32CubeMX. Après on règle les paramètres suivants dans la fenêtre des configurations:

Paramètre	Valeur
USE_COUNTING_SEMAPHORES	Enabled
USE_TIMERS	Enabled
TOTAL_HEAP_SIZE	16'384 bytes
MINIMAL_STACK_SIZE	512 Words

La fenêtre suivante s'affiche pendant que le code du projet *System Workbench* est généré :

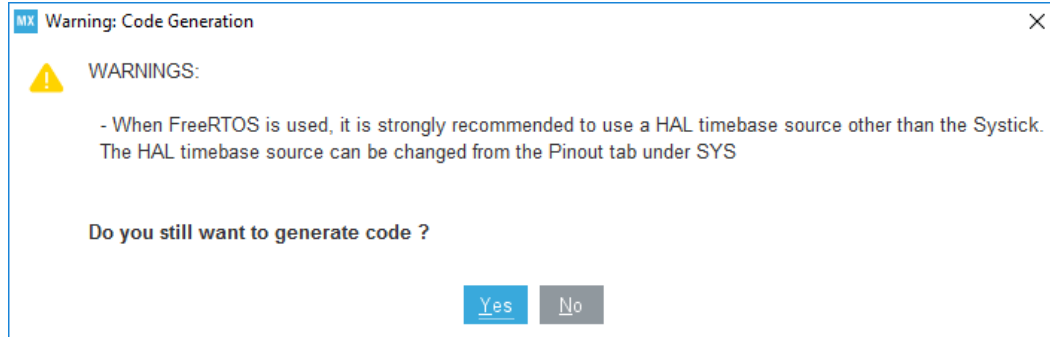


Figure 19: CubeMX Generation with FreeRTOS HAL timebase window

Ici, vous pouvez appuyer sur le bouton "Yes". Le tique pour le CubeMX HAL peut être laissée à SysTick.

Configuration XF et uGFX

Modifiez le fichier de configuration du XF (`config/xf-port-config.h`) ainsi que la librairie uGFX (`config/gfxconf.h`) afin d'utiliser FreeRTOS.

Le XF a également besoin du répertoire "xf/port/default-cmsis-os" pour utiliser FreeRTOS. Ajoutez ceci au projet.

Ajoutez les include paths nécessaires.

Settings pour Factory et XF

CubeMX a généré du code dans le fichier `main.c` qui lance un thread nommé `defaultTask`. Ce thread exécute la fonction `StartDefaultTask`, qui est également implémentée dans le fichier `main.c`.

Vous devez déplacer le code d'initialisation et de démarrage de la Factory et du XF dans la fonction `StartDefaultTask` ! Cela est nécessaire car la bibliothèque `uGFX` utilise les ressources de FreeRTOS, qui ne fonctionnent que lorsque FreeRTOS est en cours d'exécution.

Exécutez quelques recherches avec cette configuration.

Question 1: À quelle fréquence maximale peut-on régler l'échantillonnage?

Question 2: Quels avantages voyez-vous à utiliser FreeRTOS dans cette application?

Question 3: Donnez un exemple où un RTOS serait particulièrement nécessaire?

Tâches facultatives

Dans les tâches précédentes, nous avons créé la base pour un oscilloscope fonctionnel. Mais il reste des fonctionnalités qui seraient "nice to have" que l'on pourrait ajouter au projet :

1. Une fonction pour trigger le signal à un certain niveau
2. Régler l'axe du temps de l'affichage
3. Sauvegarder les valeurs échantillonnées à l'aide du DMA

FA1 : Fonction Trigger

Une fonction qui permette de déterminer le niveau du signal échantillonné doit être implémentée. Ensuite, on compare ce niveau avec un seuil. Le signal est affiché à partir du moment où le seuil est atteint ou dépassé.

Le but de cette fonction est d'afficher le signal d'une façon à ce qu'il reste toujours au même endroit sur l'écran et d'éviter qu'il se déplace à travers de l'écran.

FA2: Display – Axe du temps

Développer le programme afin d'utiliser les boutons '-' et '+' respectivement pour modifier l'axe du temps affiché à l'écran de l'oscilloscope.

FA3 : Enregistrer les valeurs échantillonnées à l'aide du DMA

La tâche d'enregistrer les échantillons de façon explicite dans la routine d'interruption peut être prise en charge par le DMA. Utiliser le DMA à l'avantage que le microcontrôleur doit moins travailler et consomme ainsi moins d'énergie.

Modifiez le code de façon que le ADC3 utilise le DMA.

Rendu

Vous devez rendre un fichier .zip qui contient les éléments suivants :

- Un petit rapport qui montre l'état du projet
 - Quelles tâches ont été réalisées ?
 - Quels travaux doivent encore être accomplis ?
 - Toutes les réponses aux questions posées
- Votre projet Eclipse. Le contenu du répertoire <MyProjectName>/Debug peut être effacé.