

Laboratoire PTR

Oscilloscope Temps Réel

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.

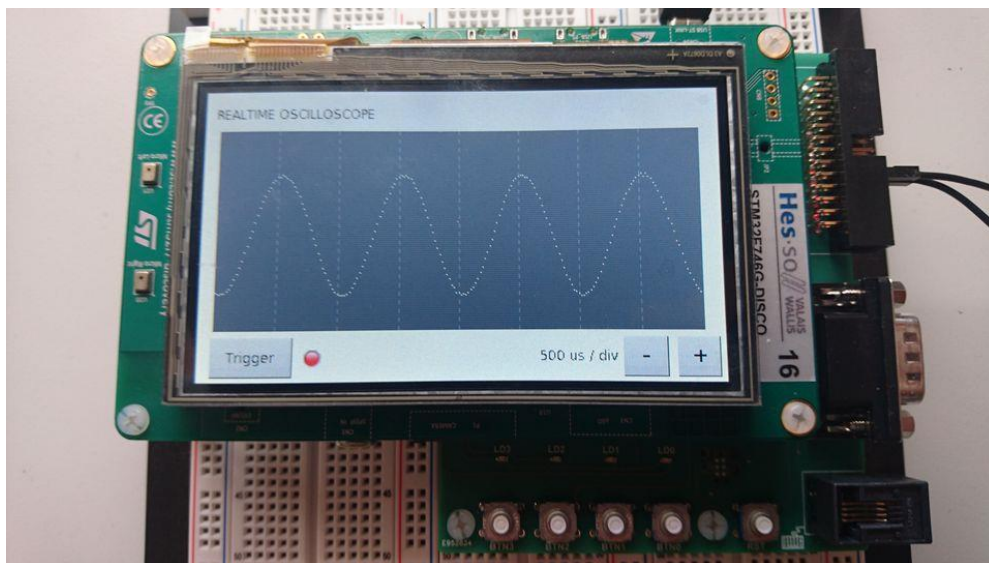



Figure 1: A F7-DISCO based Oscilloscope

Ressources requises

- Le logiciel [STM32CubeIDE](https://www.st.com/en/development-tools/stm32cubeide.html)  est un environnement de développement pour systèmes embarqués qui permet aussi la configuration des périphériques du microcontrôleur pour générer le code et accéder à ses périphériques de bas niveau (il est déjà 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.
- **Un 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 Controller
- **src/config** Répertoire avec le(s) fichier(s) de configuration
- **src/event** Evénements XF utilisés dans le projet
- **src/mdw** Répertoire de Middleware avec la librairie *Trace*
- **src/platform** Répertoire contenant le code spécifique pour le board et le microcontrôleur
- **src/xf** XF avec des ports de F7-DISCO pour IDF et FreeRTOS
- **ide-touchgfx-gen** Répertoire généré par *TouchGFX Designer* contenant le code spécifique à l'affichage de l'oscilloscope sur l'écran
- **docs** Répertoire avec schémas et datasheets

Outils de développement

STM32CubeIDE

[STM32CubeIDE](#) est une plate-forme de développement C/C++ avec des fonctionnalités de configuration de périphériques, de génération de code, de compilation de code et de débogage pour les microcontrôleurs et microprocesseurs STM32. Il est basé sur le framework Eclipse / CDT et la chaîne d'outils GCC pour le développement, et GDB pour le débogage.

STM32CubeIDE intègre les fonctionnalités de configuration STM32 et de création de projet de STM32CubeMX (fourni comme plugin Eclipse) pour offrir une expérience d'outil tout-en-un.

À tout moment du développement, l'utilisateur peut revenir à l'initialisation et à la configuration des périphériques ou middleware et régénérer le code d'initialisation sans impact sur le code utilisateur tant qu'il se trouve dans certaines zones montrées par la figure suivante :

```
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 STM32CubeIDE ne touche pas au code inscrit entre les balises 'USER CODE BEGIN' et 'USER CODE END'.

STM32CubeIDE est disponible sur 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 :

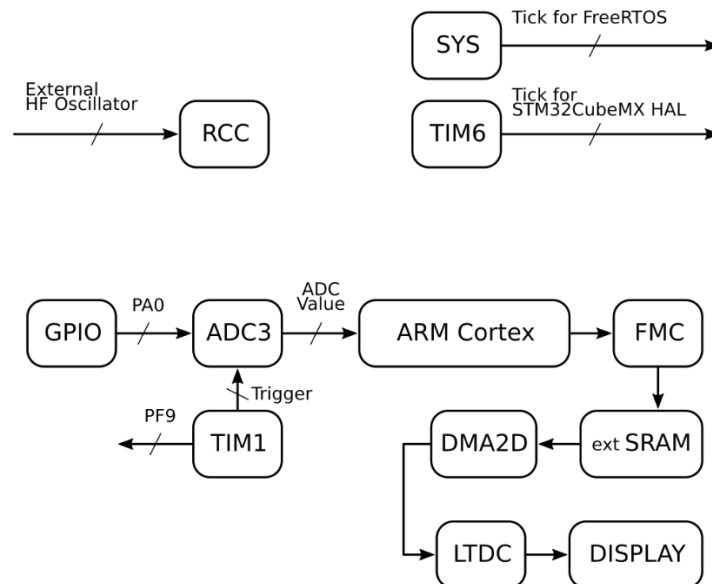


Figure 3: Schéma Bloc

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 STM32CubeIDE

Dans STM32CubeIDE, nous allons créer un nouveau projet, pour cela, nous sélectionnons l'onglet *File->New->STM32 Project*, une nouvelle fenêtre s'affiche. Vu que nous utilisons un Discovery Board comme base de notre système embarqué, nous pouvons directement cliquer sur l'onglet *Board Selector* en haut à gauche et chercher l'élément STM32F746G-DISCO.

Sélectionner le board dans la liste à droite et cliquer sur le Button *Next*.

Sur la nouvelle fenêtre, il est déconseillé d'utiliser la location par défaut. Préférer une location où vous savez que le projet demeurera.

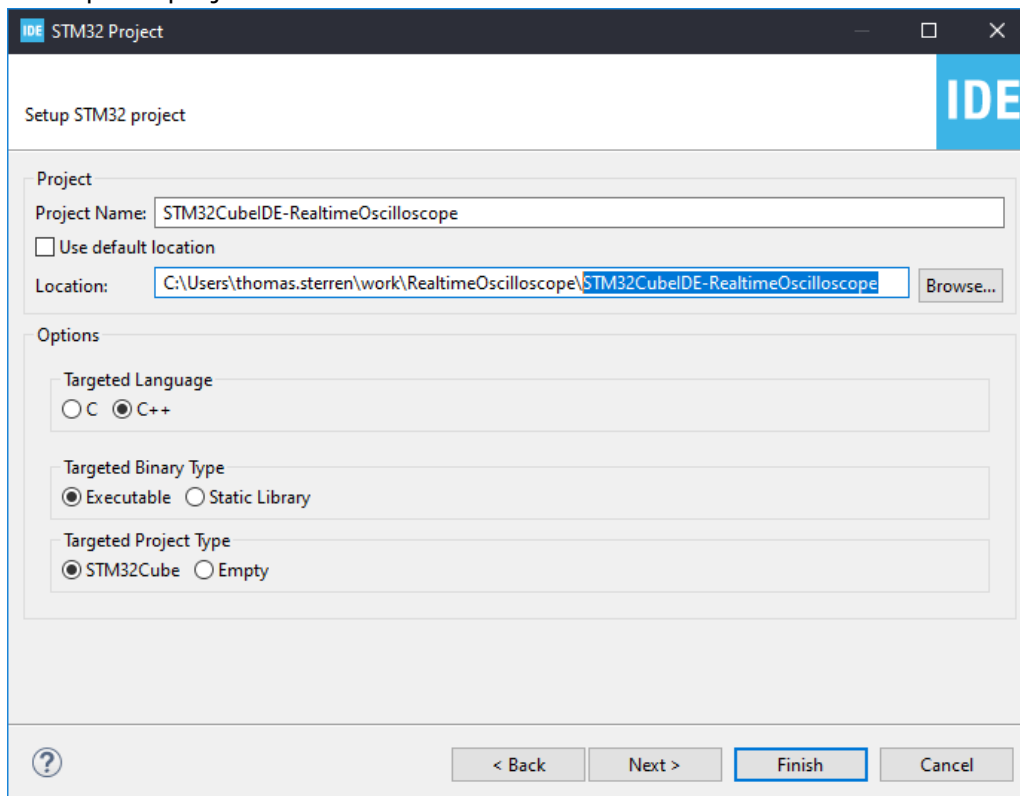


Figure 4: Project Creation - STM32 Project

N'oubliez pas de aussi créer un sous-répertoire avec le même nom que votre projet. Le plus simple est de placer ce projet dans l'endroit où vous avez extrait le fichier work.zip dans « work/RealtimeOscilloscope ».

Choisissez aussi le langage de programmation C++.

Après avoir sélectionné *Finish*, on vous demandera si les périphériques du MCU devraient être initialisés selon les valeurs de défaut pour le board sélectionné.

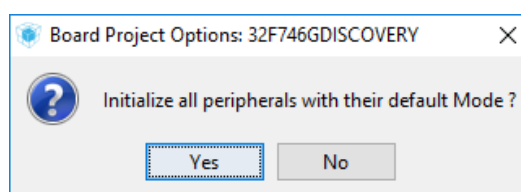


Figure 5: Project Creation – Init Peripherals

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

Si la fenêtre ci-dessous s'affiche, cliquez sur « Yes ». Pour débiter, on ne va pas utiliser FreeRTOS et ce paramètre n'a pas d'importance.

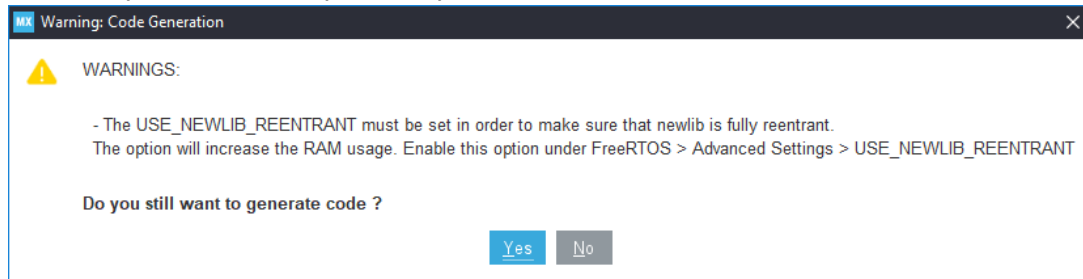


Figure 6: Project Creation – FreeRTOS USE_NEWLIB_REENTRANT

Dans l'arborescence du projet créé, la configuration des périphériques se trouvent dans le fichier avec l'extension *.ioc*.

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

Tâche 2 – Éteindre USB, FatFS, Ethernet, FreeRTOS, etc...

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

Les périphériques à désactiver se situent dans les onglets *Connectivity* et *Middleware*.

Le projet final utilisera (en incluant [TouchGFX](#)) les périphériques suivants :

- CRC
- DMA2D
- FMC
- I2C3
- LTDC
- ADC3
- SPI2

Dans un premier temps, les tâches seront faites sans utiliser d'OS, ainsi, il faut aussi désactiver FreeRTOS.

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. Pour faire cela, vous devez passer à la vue « Clock Configuration ».

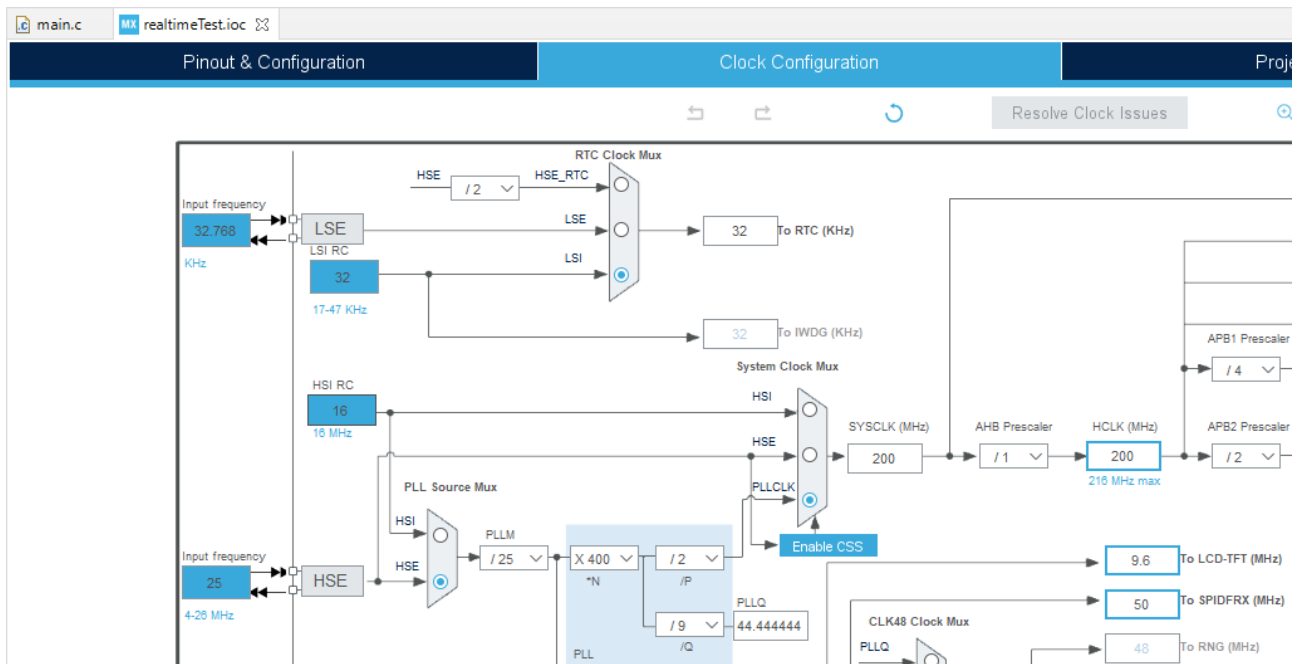


Figure 7: Clock Configuration

Vous pouvez aussi modifier la valeur *To LCD-TFT* à 9.6MHz ou avec une plus basse fréquence, ceci diminuera la fréquence de rafraîchissement de l'écran LCD permettant une meilleure fluidité pour votre future application.

Montez ensuite la stack et le heap pour l'application à `0x1000` :

The screenshot shows the STM32CubeIDE Project Manager. The 'Project' tab is selected, displaying the following settings: Project Name: STM32CubeIDE-RealtimeOscilloscope, Project Location: C:\Users\thomas.sterren\work\RealtimeOscilloscope, Application Structure: Advanced, Toolchain Folder Location: C:\Users\thomas.sterren\work\RealtimeOscilloscope\STM32CubeIDE-RealtimeOscilloscope, Toolchain / IDE: STM32CubeIDE, and Generate Under Root: checked. The 'Code Generator' tab is also visible. The 'Linker Settings' section shows the 'Minimum Heap Size' and 'Minimum Stack Size' both set to 0x1000.

Figure 8: Stack and Heap Sizes

Nous pouvons finalement générer le code du projet, pour cela, nous utilisons le bouton *Device Configuration Tool Code Generation* montré dans la figure suivante.



Figure 9: Code Generation

Tâche 4 – Build et Debug

Nous devons maintenant compiler le projet, télécharger sur la cible et déboguer. La création du fichier de programme devrait fonctionner sans problème.

Optimisations de projet:

- Clic droit sur le projet -> *Properties* -> *C/C++ Build* -> *Behavior* -> *Enable parallel build*

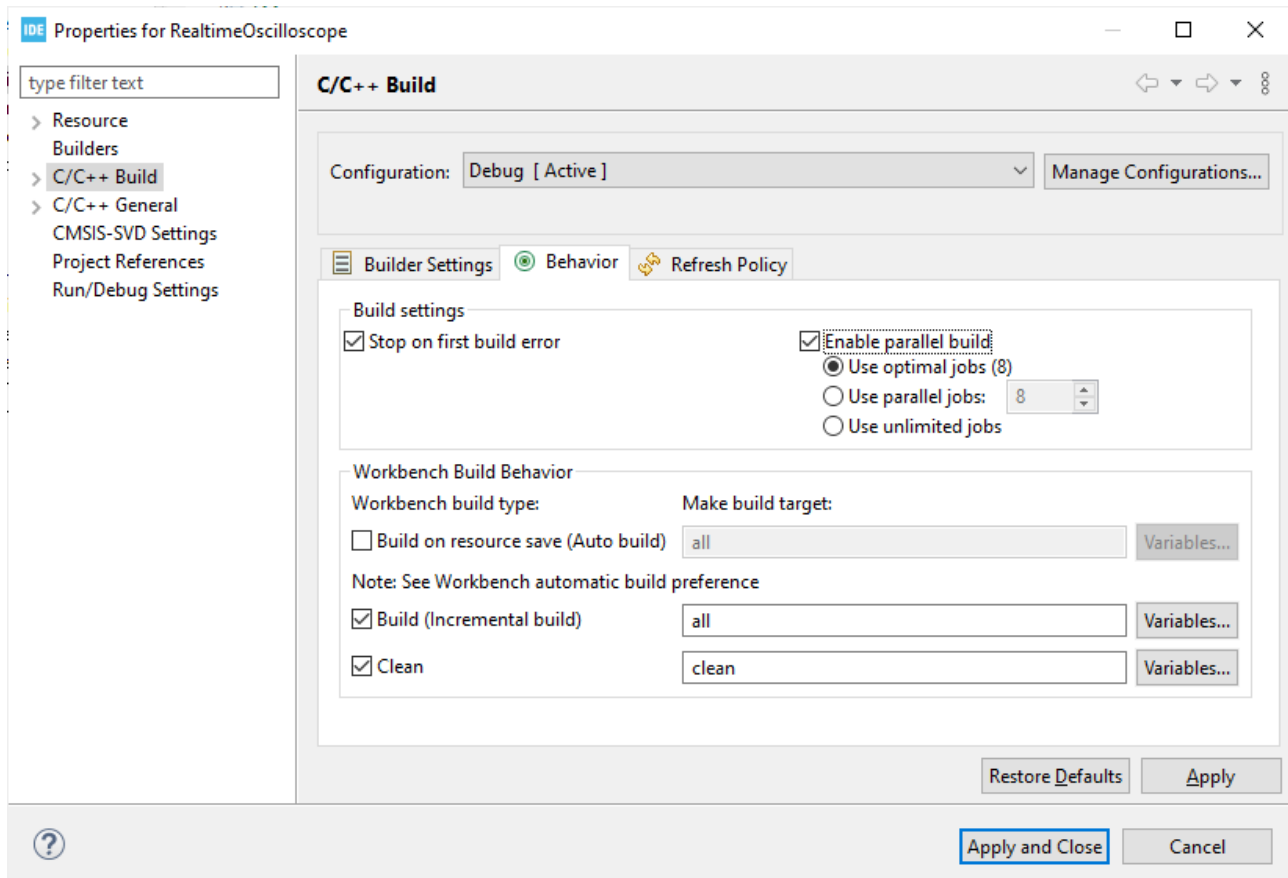


Figure 10: Enable parallel build.

Cette optimisation permet de build le projet beaucoup plus vite, nous pouvons build ce projet avec l'icône en forme de marteau, en faisant un clic droit sur le projet -> *build project* ou tout simplement avec le raccourci **ctrl + b**.

Pour déboguer, une configuration de Debug doit d'abord être établie. Allez dans le menu **Run->Debug Configurations...** et créez une nouvelle configuration de debug STM32 Cortex-M C/C++ Application

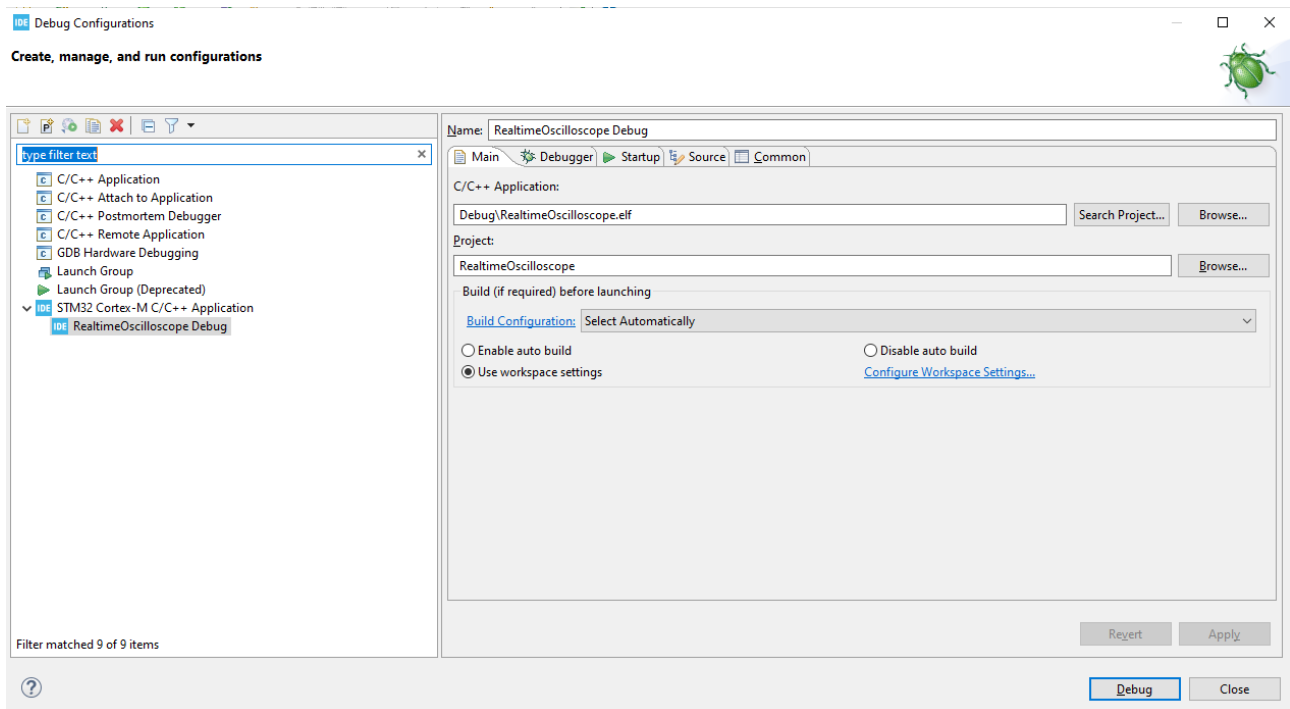



Figure 11: Debug Launch Configuration

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 5 – 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 :

#	Composant	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 un RTOS ? Justifiez votre réponse. **Aucun des deux, il faut un timer hardware**
- Question 2 : Est-ce qu'il est possible d'exécuter le composant numéro 2 avec un XF ou un RTOS ? Justifiez votre réponse. **XF est suffisant**
- Question 3 : Si l'on combine un timer hardware avec un XF, lequel des deux composants doit être priorisé ? Justifiez votre réponse. **Le timer hardware. Si le XF s'occupe principalement de l'affichage, il n'est pas prioritaire**

Tâche 6 – Générateur de Fréquence Externe

Dans le prochain chapitre, nous utiliserons le convertisseur analogique/numérique (ADC) pour mesurer un signal externe. Pour générer ce signal, un petit générateur de fonctions est prévu :

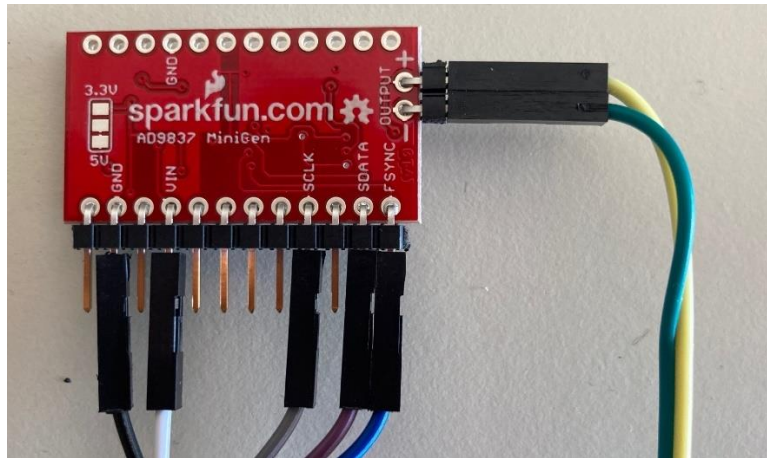


Figure 12: Signal Generator "MiniGen"

Ce générateur de fréquence (Sparkfun-AD9837 / MiniGen) peut générer 3 formes de signaux différents (triangle, rectangle et sinus) à des fréquences différentes. Ce signal servira de signal d'entrée pour notre oscilloscope.

Connectez le MiniGen à la carte F7-DISCO selon les chiffres ci-dessous :

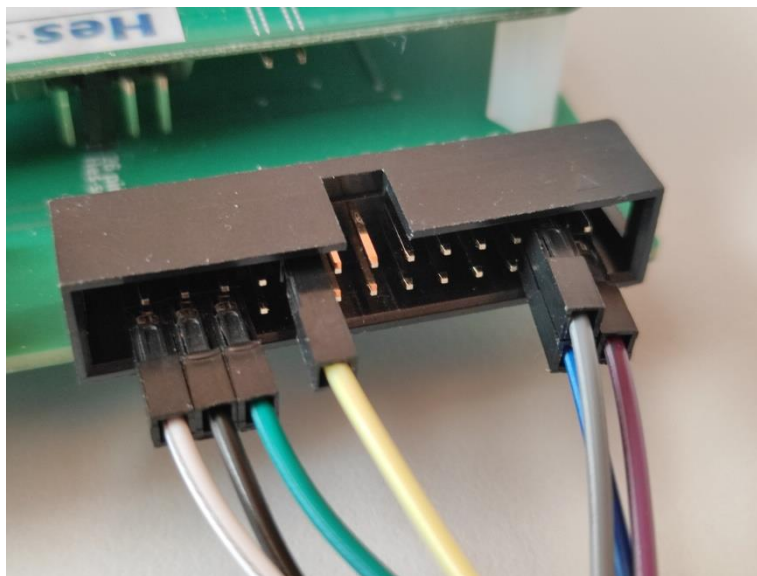


Figure 13: Wires on 26-pin Header

Veuillez-vous assurer que le fil jaune est connecté à la cinquième broche en partant de la gauche sur la rangée supérieure. Cela correspond à la broche 17 (ADC3_IN0) du connecteur J3 à 26 broches !

Cette broche est le signal d'entrée pour la périphérie ADC3 du microcontrôleur.

MiniGen Signals	Pin-Header 26 Pins	Pin-Header 26 Signals	Wire Color
GND	#22	GND	green
VIN	#26	3V3	white
SCLK	#3	SCK	grey
SDATA	#2	MOSI	violet
FSYNC	#4	nSS_OUT	blue
OUTPUT-	#24	GND	black
OUTPUT+	#17	ADC3_IN0	yellow

Figure 14: Table with Signals and Pins for 26-pin Header

Tâche 7 – 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 ? **Au minimum, le double, soit 2kHz**

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

Question 3 : Est-ce la fréquence donnée par le théorème d'échantillonnage ou devrait-elle être plus élevée ? **C'est la fréquence du théorème, pour éjecter les fréquences inutiles**

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 ? **INO**

Utilisez le fichier .ioc du projet 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 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 fins. Pour cela créez le fichier *isrs.cpp* dans le répertoire *Core/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 15: 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 16: `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 éventuellement effectuer des mesures à des intervalles réguliers à l'aide de ses propres moyens ?

Oui, continuous. Mais pour être vraiment régulier, il

Attention : Limitation de la tension d'entrée faut utiliser un timer pour activer la conversion

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 8 – Configuration du timer Hardware (TIM)

Maintenant qu'on est capable de lire les valeurs AD, on souhaiterait le faire à des intervalles de temps réguliers. 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 devrons 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 la 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 *Core/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 9 – 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 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 de l'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 10 – XF Integration

XF Package

Dans la prochaine étape, nous voudrions intégrer le XF inclus dans le *work.zip*. Copier alors le répertoire *src* au complet dans le répertoire où votre projet STM32CubeIDE-*RealtimeOscilloscope* se trouve :

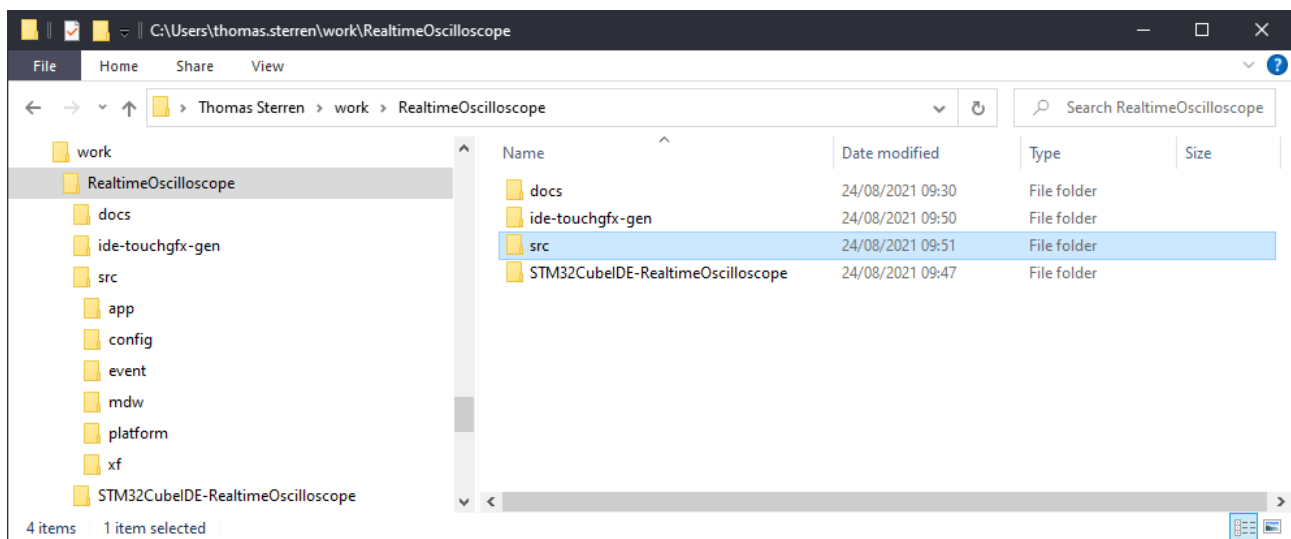


Figure 17: Work Folder Structure

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

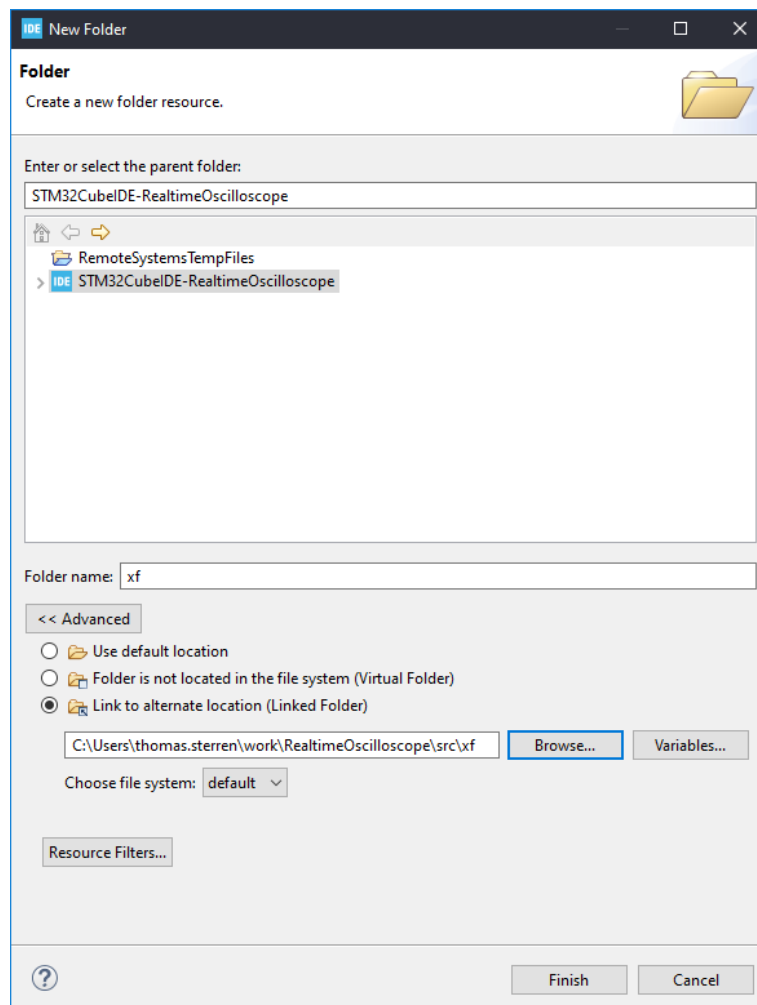


Figure 18: Project Link Creation to src/xf Folder

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

- *xf/core*
- *xf/include*
- *xf/port*

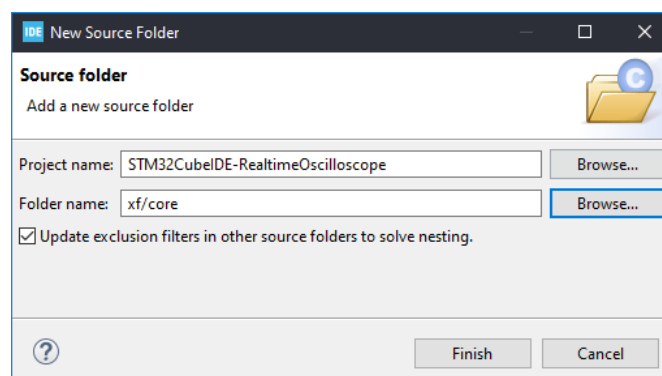


Figure 19: Example – Add 'xf/core' as Source Folder

Dans les propriétés du projet sous *C/C++ Build -> Settings -> Include paths* 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"
```

Faites cela pour les compilateurs suivants :

- MCU GCC Compiler
- MCU G++ Compiler

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 au 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 HAL_TIM_PeriodElapsedCallback

Le XF nécessite un timer hardware pour que le port IDF gère les timeouts. Le timer TIM6 est déjà réglé pour une période de 1 ms dans le fichier *.ioc* du projet pour servir un temps de base pour la librairie HAL fourni par STM32CubeIDE. Nous utiliserons celui-ci afin de faire fonctionner le XF. Rajoutez le code suivant dans le fichier *Core/Src/main.c* dans la fonction *HAL_TIM_PeriodElapsedCallback()* :

```
#if (PORT_IDF_STM32CUBE != 0)
    if (htim->Instance == TIM6)
    {
        // STM32 HAL tick handler gets called every millisecond (is given by code
        // generated by STM32CubeIDE). Check which interval is needed by the
        // XF (typically slower) and call XF_tick() accordingly.
        if (XF_isRunning() &&
            (HAL_GetTick() % XF_tickIntervalInMilliseconds()) == 0)
        {
            XF_tick();
        }
    }
#endif // PORT_IDF_STM32CUBE
```

Mettez le code ci-dessous entre les balises */* USER CODE BEGIN Callback 1 */* et */* USER CODE END Callback 1 */* afin qu'il ne soit pas écrasé la prochaine fois que le code sera généré.

N'oubliez pas d'inclure le fichier header *port-functions.h* dans le fichier *main.c* :

```
#include "xf/port/port-functions.h"
```

Board Dependencies

Dans le dossier *platform/board*, les classe *LedController* et *ButtonsController* utilisent les LEDs et les boutons de la carte d'extension. Dans le projet STM32CubeIDE, les GPIO User Label suivants doivent être assignés aux GPIOs correspondants :

User Label	GPIO Name
LED0	PA15
LED1	PH6
LED2	PA8
LED3	PB4
BUTTON0	PI2
BUTTON1	PI3
BUTTON2	PG7
BUTTON3	PG6

Ajoutez aussi les interruptions des boutons, spécifier leur mode (*External Interrupt Mode with Rising/Falling edge trigger detection*) ainsi que le fait que ceux-ci sont des pull-up.

Activez les interruptions EXTI2, EXTI3 et EXTI9:5 correspondantes dans le périphérique *NVIC*.

Ajoutez au fichier *Core/Src/isrs.cpp* le code suivant qui relie les interruptions avec la class *ButtonsController* :

```
#include "board/buttonscontroller.h"

extern "C" void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
```



```
{
    switch (GPIO_Pin)
    {
    case BUTTON0_Pin:
    case BUTTON1_Pin:
    case BUTTON2_Pin:
    case BUTTON3_Pin:
        ButtonsController::getInstance().onIrq();
        break;
    default:
        break;
    }
}
```

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

Tâche 11 – Application

Application Package

Pour la prochaine étape, 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 entre autres trois classes : `Factory`, `Gui` et `oscilloscope::Controller`.

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 `oscilloscope::Controller` est responsable du comportement de l'oscilloscope. Elle décide quelles données afficher et à quel moment.

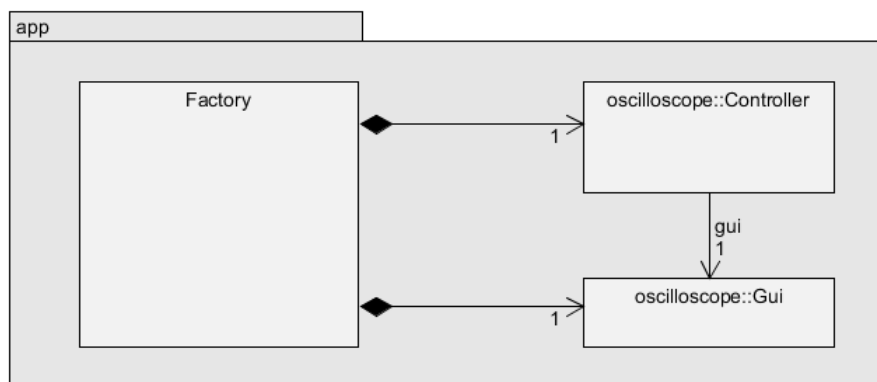


Figure 20: App Package – Class Model Diagram

Le XF et la classe `Factory` doivent être appelés/intégrés dans le *Src/main.c* généré. Les fonctions C correspondantes 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 effacées. Celles-ci sont désormais appelées par la classe `Factory`.

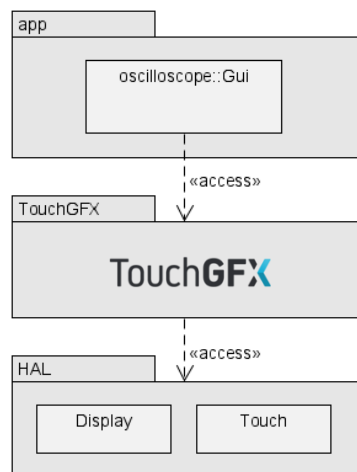
Display Packages

Figure 21: GUI Software Layers

[TouchGFX Designer](#) est un logiciel permettant de générer de manière très simple du code pour afficher divers éléments sur un écran pour les microcontrôleurs STM32. Vous pouvez ajouter les répertoires suivants dans le projet de la manière suivante :

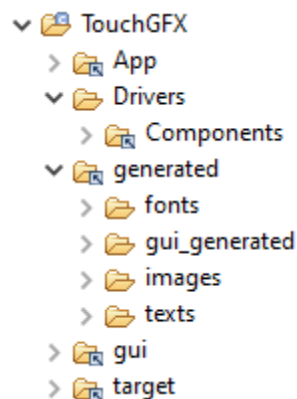


Figure 22: TouchGFX – Folder Structure

Ces répertoires se situent dans le dossier ide-touchgfx-gen :

Répertoire	Remarque
TouchGFX/App	
TouchGFX/gui	
TouchGFX/generated	Ne pas ajouter le répertoire <i>simulator</i> au projet
TouchGFX/target	Le code du répertoire fut modifié pour fonctionner sans RTOS
Drivers/Components	

Les chemins d'accès suivants doivent être rajoutés comme des *Includes Paths* pour le compilateur C :

```
"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/App"
"${ProjDirPath}/../ide-touchgfx-gen/Drivers/Components"
```

Les chemins d'accès suivants doivent être rajoutés comme des *Includes Paths* pour le compilateur C++ :

```
"${ProjDirPath}/../ide-touchgfx-gen"
```

```

"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/App"
"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/target/generated"
"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/target"
"${ProjDirPath}/../ide-touchgfx-gen/Drivers/Components/ft5336"
"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/generated/fonts/include"
"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/generated/images/include"
"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/generated/texts/include"
"${ProjDirPath}/../ide-touchgfx-gen/Middlewares/ST/touchgfx/framework/include"
"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/gui/include"
"${ProjDirPath}/../ide-touchgfx-gen/TouchGFX/generated/gui_generated/include"

```

Il ne manque plus qu'à ajouter la librairie de TouchGFX dans le linker comme montré dans les figures suivantes :

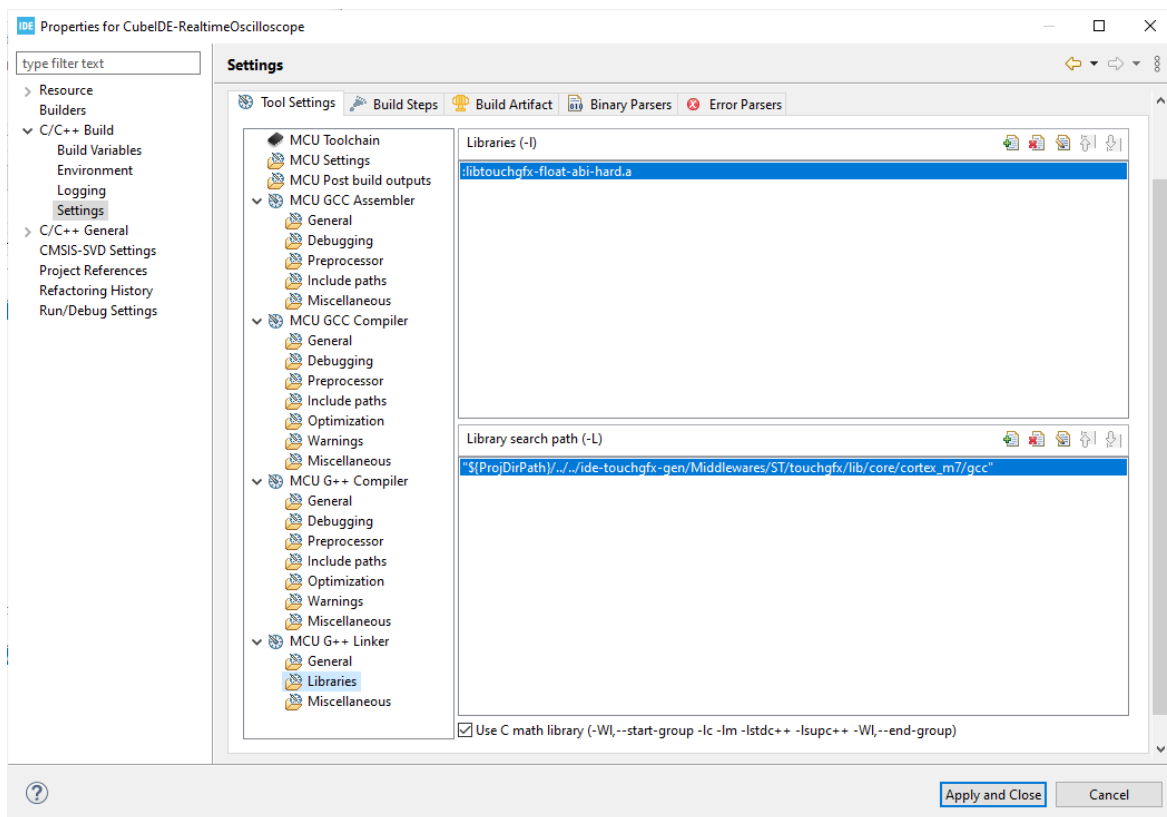


Figure 23: TouchGFX – Library

Nom de la librairie :

`:libtouchgfx-float-abi-hard.a`

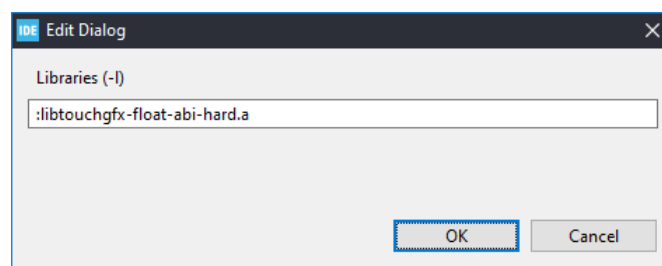


Figure 24: TouchGFX – Library Name

Et chemin de la librairie :

```
"${ProjDirPath}/../ide-touchgfx-gen/Middlewares/ST/touchgfx/lib/core/cortex_m7/gcc"
```

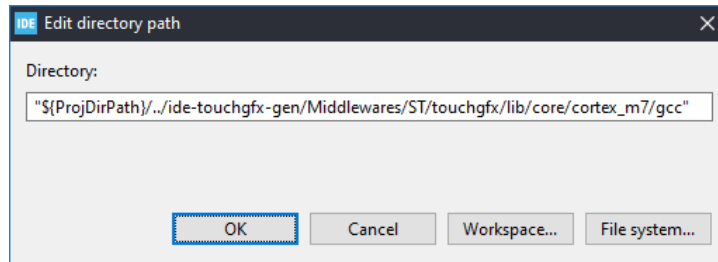


Figure 25: TouchGFX – Library Path

Vous trouverez dans le fichier *ide-touchgfx-gen/Core/Src/main.c* un exemple de main généré par le logiciel TouchGFX Designer. Comparez-le avec votre *main.c* et faites les modifications nécessaires (pensez à contrôler la fonction *MX_FMC_Init()*).

Dans l'exemple généré, TouchGFX utilise l'OS FreeRTOS, il vous faudra donc supprimer toutes références à cet OS ainsi qu'aux threads.

Générateur de Fréquence – MiniGen – Mode d'Emploi

Le module générateur de fréquence MiniGen, connecté en externe, peut être commandé par les trois premiers boutons de la carte d'extension F7-DISCO :

- BTN3 : Modifie la forme du signal (triangle, rectangle, sinus)
- BTN2 : Passage à la fréquence inférieure suivante
- BTN1 : Passage à la fréquence supérieure suivante

Sur la vue de l'oscilloscope sur l'écran LCD, il y a un indicateur bleu qui décrit le signal actuellement généré :

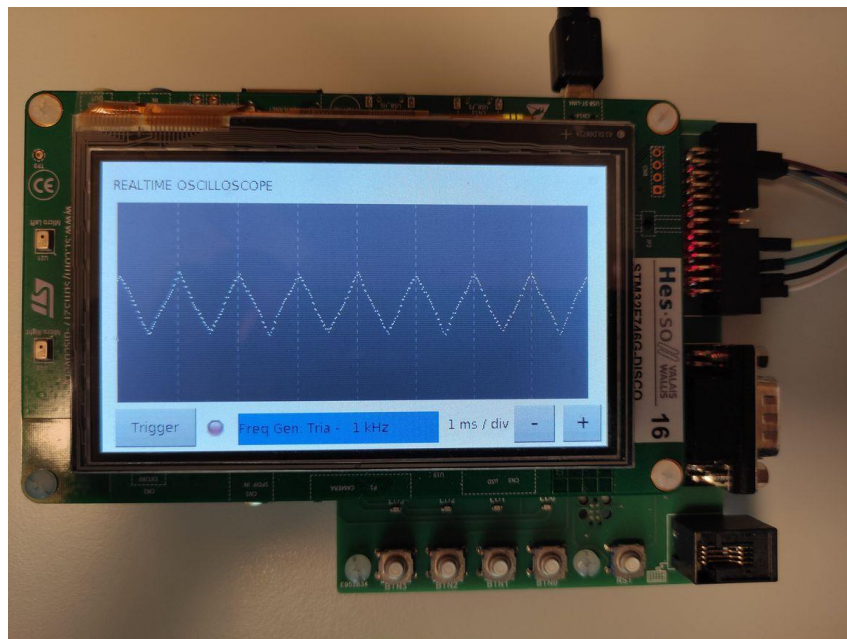


Figure 26: MiniGen - Signal Generation Info

Tâche 12 – Oscilloscope GUI

La classe `oscilloscope::Controller` est responsable d'afficher les valeurs lues de l'ADC.

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

Tâche 13 – 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 14 – RTOS Integration

Dans cette étape, nous allons analyser le comportement de notre application avec un système d'exploitation.

Le XF ainsi que la librairie [TouchGFX](#) supportent [FreeRTOS](#). Activez le système d'exploitation FreeRTOS dans le fichier `.ioc` :

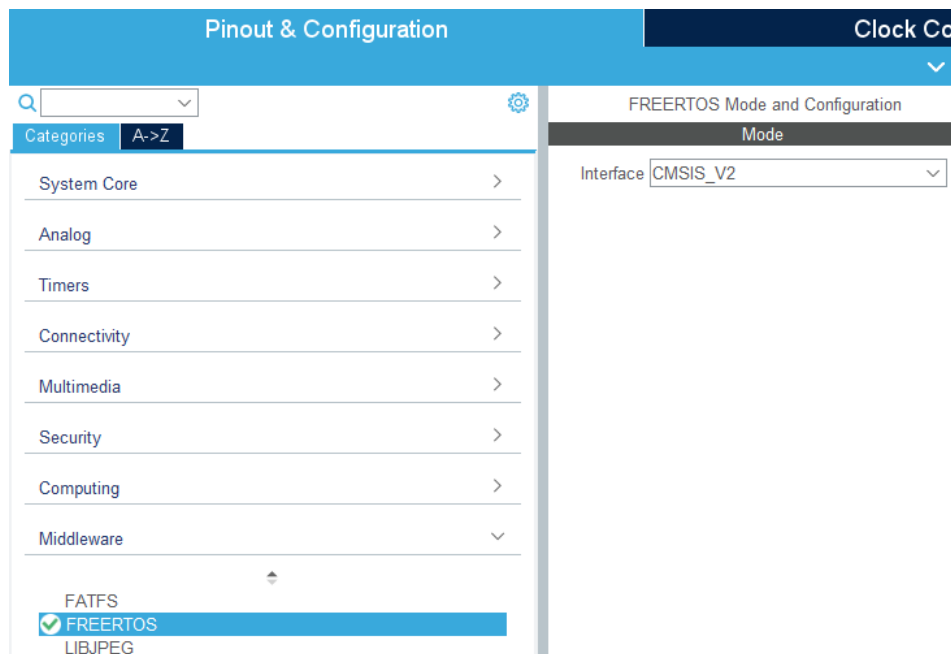


Figure 27: FreeRTOS Configuration CMSIS_V2

Après, on règle les paramètres suivants dans la fenêtre des configurations :

Paramètre	Valeur
USE_COUNTING_SEMAPHORES	Enabled
USE_TIMERS	Enabled
TOTAL_HEAP_SIZE	65'536 bytes
MINIMAL_STACK_SIZE	2048 Words

Activez également le paramètre `USE_NEWLIB_REENTRANT` :

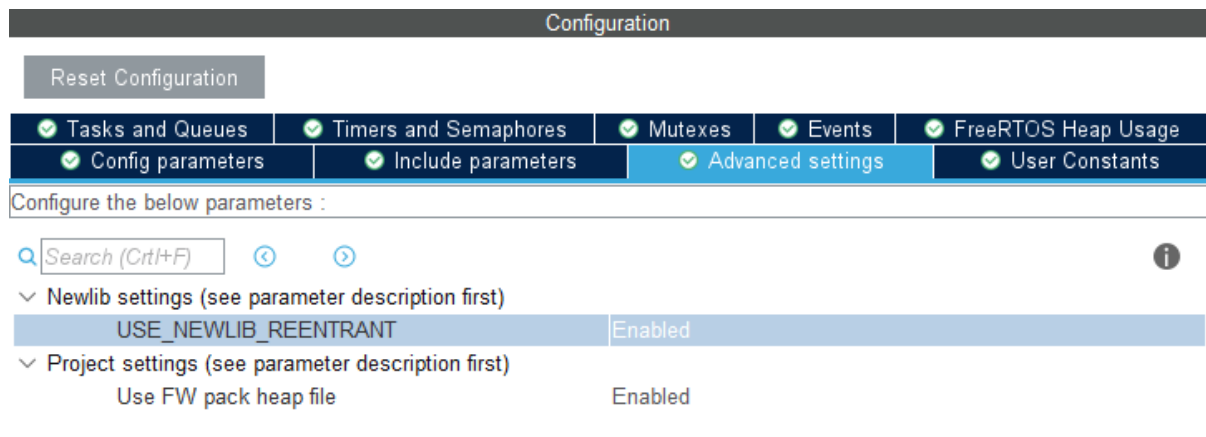


Figure 28: FreeRTOS USE_NEWLIB_REENTRANT

Configuration XF

Modifiez le fichier de configuration du XF (*config/xf-port-config.h*) afin d'utiliser FreeRTOS. Le XF a également besoin du répertoire "xf/port/default-cmsis-os" pour utiliser FreeRTOS.

Ajoutez ceci au projet, si il n'est pas encore mis comme répertoire de source.

Ajoutez les include paths nécessaires.

Configuration TouchGFX

Modifiez le fichier de configuration pour TouchGFX (*config/touchgfx-config.h*) afin d'utiliser FreeRTOS.

Settings pour Factory et XF

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

Pour que la librairie graphique utilise correctement les fonctionnalités de FreeRTOS, vous pouvez de nouveau regarder le fichier de base généré par TouchGFX Designer dans *ide-touchgfx-gen/Core/main.c*. C'est la fonction `StartDefaultTask` qui doit gérer l'appelle `MX_TouchGFX_Process()` de TouchGFX.

Effectuez 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 permet 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 au 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.