



Tutoriels STM32F0

Périphériques standards

**Convertisseur Analogique vers
Numérique CAN**

Arouna DARGA – enseignant chercheur Sorbonne Université
arouna.darga@sorbonne-universite.fr

Table des matières

1.	Introduction.....	3
2.	Le Pin mapping	3
3.	Configuration et test de l'ADC.....	4
4.	Monitoring avec STM Studio	10
4.1	Installation de STM Studio.....	10
4.2	Configuration de STM Studio.....	12
4.3	Commencer le monitoring.....	14
5.	Résumé.....	15

1. Introduction

Les convertisseurs analogique-numérique (CAN) ou Analog to Digital Conversion (ADC) sont impliqués dans plusieurs projets de systèmes embarqués. De de l'interface utilisateur via la capture des valeurs de tension de potentiomètres à la numérisation des données des capteurs. L'utilisation de l'ADC est manifeste dans un certain nombre de cas. Le STM32 peut avoir un ou plusieurs convertisseurs analogique-numérique. Un CAN est un matériel complexe, qui prend de la place sur le silicium et qui nécessite de l'énergie pour fonctionner. Par conséquent, peu d'ADC sont généralement intégrés dans un MCU. Afin de réaliser des conversions sur plusieurs canaux (broches), un seul CAN est connecté séquentiellement (multiplexage) à plusieurs broches d'entrées.

Les périphériques ADC peuvent fonctionner sur un seul canal ou sur plusieurs canaux de manière séquentielle. Vous pouvez effectuer une seule conversion à la demande (ou une seule séquence), ou laisser l'ADC fonctionner en continu.

Dans ce tutoriel, vous allez configurer l'ADC pour qu'il fonctionne en continu sur un seul canal. Si vous comprenez ce qui est fait ici, vous serez en mesure d'adapter la configuration à vos besoins.

2. Le Pin mapping

Toutes les broches du MCU ne peuvent pas être utilisées pour la conversion ADC. Vous devez vous référer à la fiche technique pour découvrir quelles broches sont connectées au multiplexeur d'entrée de l'ADC.

Par exemple, PA0 et PA1 sont connectés respectivement aux entrées 0 et 1 de l'ADC

L2	23	14	10	F8	PA0	I/O	TTa		USART2_CTS, TIM2_CH1_ETR, TSC_G1_IO1, USART4_TX	RTC_TAMP2, WKUP1, COMP1_OUT, ADC_IN0, COMP1_INM6
M2	24	15	11	G7	PA1	I/O	TTa		USART2_RTS, TIM2_CH2, TIM15_CH1N, TSC_G1_IO2, USART4_RX, EVENTOUT	ADC_IN1, COMP1_INP

Dans l'appareil STM32F072RB, il n'y a qu'un seul CAN, mais le multiplexeur d'entrée comporte 16 entrées (0 à 15). Il n'y a aucune raison de préférer un canal parmi d'autres, à moins d'avoir des restrictions sur les broches. Au laboratoire, choisissons arbitrairement le canal 11 de l'ADC. Selon le tableau ci-dessous, le canal 11 est connecté à la broche PC1.

H1	15	8	-	-	PC0	I/O	TTa		EVENTOUT	ADC_IN10
J2	16	9	-	-	PC1	I/O	TTa		EVENTOUT	ADC_IN11
J3	17	10	-	-	PC2	I/O	TTa		SPI2_MISO, I2S2_MCK, EVENTOUT	ADC_IN12
K2	18	11	-	-	PC3	I/O	TTa		SPI2_MOSI, I2S2_SD, EVENTOUT	ADC_IN13

En fin de compte, vous devez vous assurer que le PC1 n'est pas impliqué dans une fonction quelconque au niveau du tableau (telle que LED, bouton, USB...). La meilleure source d'information est le schéma de la carte.

Ici, nous pouvons voir que le PC1 est totalement libre pour notre projet.

PC0	8	PC0
PC1	9	PC1
PC2	10	PC2
PC3	11	PC3
PC4	24	PC4
PC5	25	PC5
PC6	37	PC6
PC7	38	PC7

MCU_LQFP64

3. Configuration et test de l'ADC

Le code ci-dessous définit l'ADC pour une seule conversion continue sur le canal 11. Comme d'habitude, la première étape consiste à configurer le GPIO associé dans le mode correct (analogique ici).

Ensuite, l'horloge de l'ADC est activée, suivie des réglages des registres de l'ADC. Reportez-vous au manuel de référence pour bien comprendre ce qui s'y fait.

Ajoutez le code ci-dessous au fichier source **bsp.c** existant dans le projet **my_project**

```

/*
 * ADC_Init()
 * Initialise ADC pour la conversion sur un seul canal
 * canal 11 -> pin PC1
 */

void BSP_ADC_Init()
{
    // Activation de horloge du GPIOC
    // Mettre a '1' le bit b19 du registre RCC_AHBENR
    RCC->AHBENR |= (1<<19);

    // Configure le pin PC1 en mode Analog
    // Mettre à "11" les bits b3b2 du registre GPIOC_MODER
    GPIOC->MODER |= (0x03 <<1);

    // Activation de horloge de ADC
    // Mettre a '1' le bit b9 du registre RCC_APB2ENR
    // voir page 130 pour les autres bits
    RCC->APB2ENR |= (1<<9);

    // Reset de la configuration de ADC
    // Mise a zero des regsitres de configuration de ADC
    ADC1->CR      = 0x00000000;
    ADC1->CFGR1   = 0x00000000;
    ADC1->CFGR2   = 0x00000000;
    ADC1->CHSELR  = 0x00000000;

    // Choix du mode de conversion
    // bit b13 (CONT) du registre ADC1_CFGR1
    // '0' : conversion une seule fois : a la demande
    // '1' : conversion une en continue
    ADC1->CFGR1 |= (1<<13);

    // Choix de la resolution (nombre de bits des data)
    // bits b4b3 (Data resolution)
    //      00: 12 bits
    //      01: 10 bits
    //      10: 8 bits
    //      11: 6 bits
    ADC1->CFGR1 &= ~(0x03 <<4);

    // Choix de la source horloge pour ADC
    // bits b31b30 CKMODE[1:0]: ADC clock mode
    // 00: ADCLK (Asynchronous clock mode), generated at product level
    // (refer to RCC section)
    // 01: PCLK/2 (Synchronous clock mode)
    // 10: PCLK/4 (Synchronous clock mode)
    // 11: Reserved
    ADC1->CFGR2 |= (0x01 <<31UL);

    // Choix de la periodechantillonnage
    // Bits b2b1b0 (SMP[2:0]: Sampling time selection) du registre ADC
    // sampling time register (ADC_SMPR)
    // 000: 1.5 ADC clock cycles
    // 001: 7.5 ADC clock cycles
    // 010: 13.5 ADC clock cycles
    // 011: 28.5 ADC clock cycles
    // 100: 41.5 ADC clock cycles
    // 101: 55.5 ADC clock cycles
    // 110: 71.5 ADC clock cycles

```

```

// 111: 239.5 ADC clock cycles
ADC1->SMPR = 0x03;

// Selectionner le canal 11 pour la conversion
// bit b11 du registre ADC channel selection register (ADC_CHSELR)
// Select channel 11
ADC1->CHSELR |= ADC_CHSELR_CHSEL11;

// Activer ADC
// Mettre a '1' le bit b0 du registre ADC_CR
ADC1->CR |= (1<<0);

// Demarrer la conversion
// Mettre a '1' le bit b2 du registre ADC_CR
ADC1->CR |= (1<<2);
}

```

Ajouter le prototype de déclaration au fichier d'en-tête **bsp.h** :

```

25
26 void    BSP_PB_Init    (void);
27 uint8_t BSP_PB_GetState (void);
28
29 /*
30  * Fonction initialisation USART2 pour etre utiliser comme Console Debug
31  */
32
33 void    BSP_Console_Init (void);
34
35
36 /*
37  * fonctions ADC
38  */
39
40 void BSP_ADC_Init    (void);
41
42
43
44 #endif /* BSP_INC_BSP_H_ */
45

```

Il est maintenant temps de tester :

Extrait du code du fichier **main.c**. Par soucis de clarté, la définition de la fonction **static void SystemClock_Config(void)**; n'est pas présenté.

Le résultat de la conversion ADC est disponible dans le registre des données **Data Register (DR)**. Il est conseillé de s'assurer que la dernière conversion est effectuée avant de lire ce registre. Pour ce faire, il suffit d'interroger le drapeau **EOC (End Of Conversion)** du périphérique **ADC1**.

```

#include "stm32f0xx.h"
#include "bsp.h"
#include "delay.h"
#include "main.h"

static void SystemClock_Config(void);

int main()
{
    // configuration de horloges du system
    SystemClock_Config();

    // Initialiser la liason serie : Console de Debug
    BSP_Console_Init();

    // test du printf
    mon_printf("La Console est ready!\r\n");

    // Initiaaliser et lancer ADC sur Pin PC1
    BSP_ADC_Init();
    mon_printf("ADC est ready!\r\n");

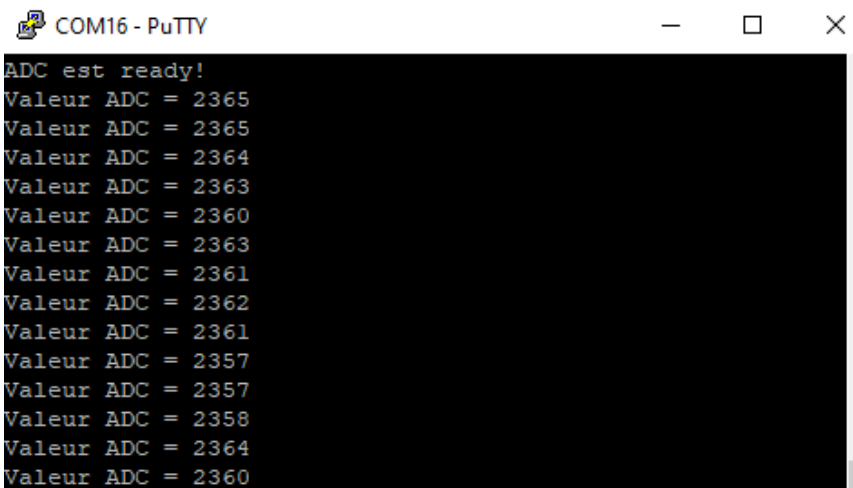
    // Boucle Infini
    while(1)
    {
        // Tant que la conversion est en cours :
        // attendre que le bit b2 du registre ADC interrupt and
        status register (ADC_ISR)
        // passe a '1'
        while ((ADC1->ISR & (1<<2)) != (1<<2));

        // afficher le resultat de la conversion sur la console
        mon_printf("Valeur ADC = %d\r\n", ADC1->DR);

        // attendre environ 200 ms
        BSP_DELAY_ms(200);
    }
}

```

Sauvegardez tout, compilez et téléversez dans le MCU :



```

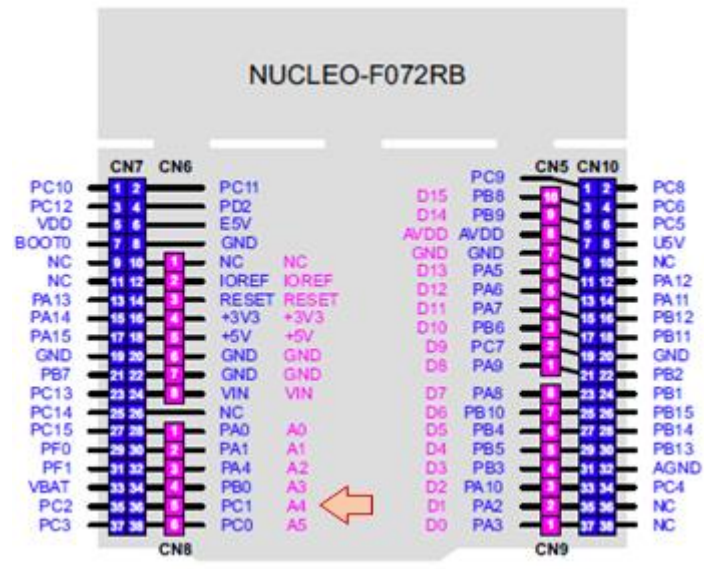
COM16 - PuTTY
ADC est ready!
Valeur ADC = 2365
Valeur ADC = 2365
Valeur ADC = 2364
Valeur ADC = 2363
Valeur ADC = 2360
Valeur ADC = 2363
Valeur ADC = 2361
Valeur ADC = 2362
Valeur ADC = 2361
Valeur ADC = 2357
Valeur ADC = 2357
Valeur ADC = 2358
Valeur ADC = 2364
Valeur ADC = 2360

```

Notez que vous pourriez tout aussi bien utiliser le débogueur pour surveiller les résultats de l'ADC, mais d'une manière moins "en temps réel".

Pour tester davantage le code, il faut faire un peu de câblage.

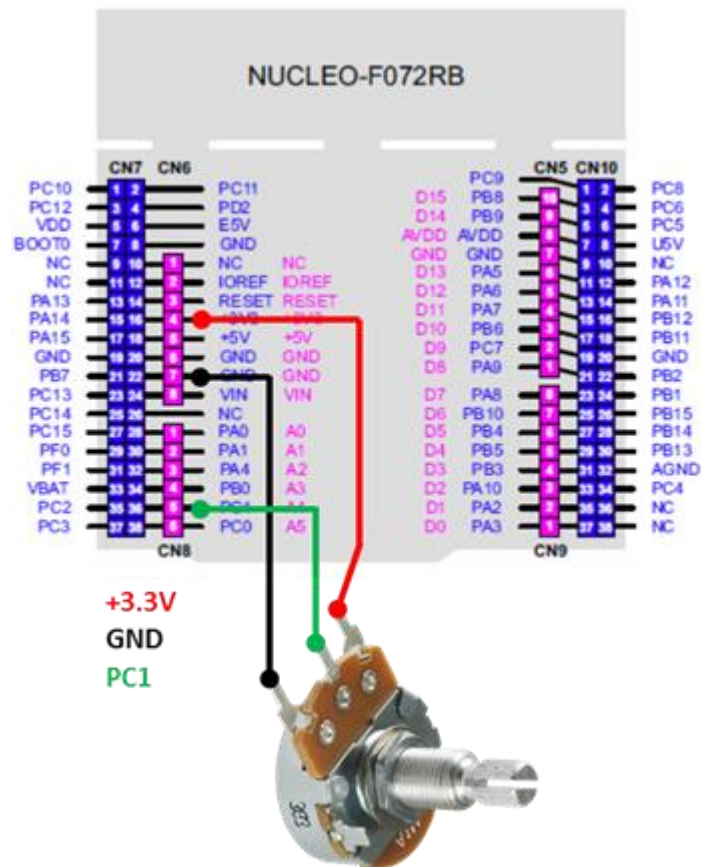
Tout d'abord, localisez la broche **PC1** reporté sur la carte Nucleo :



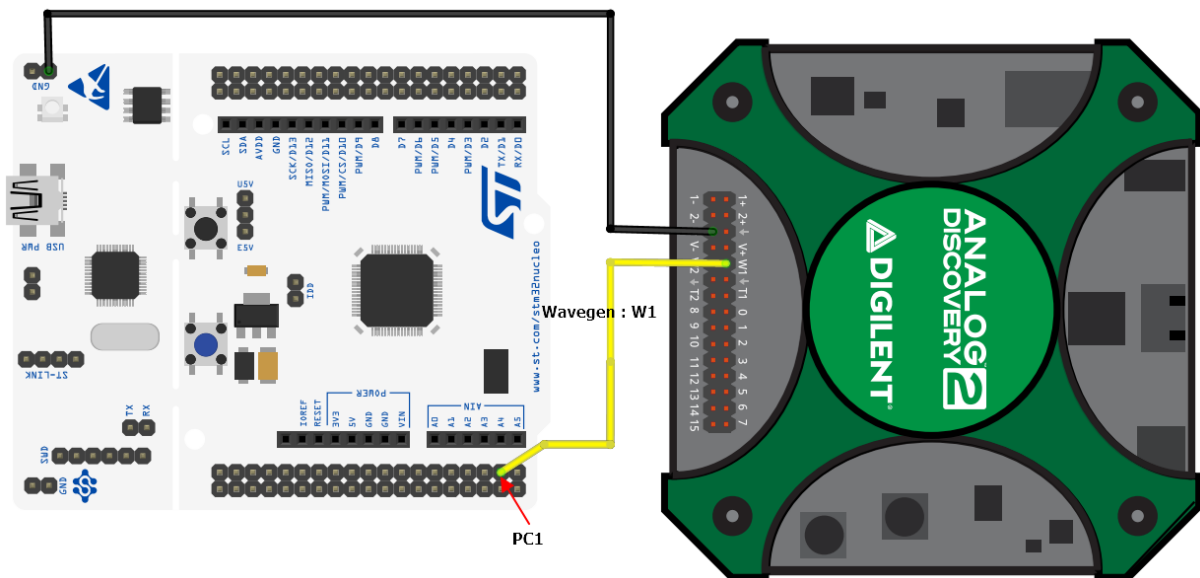
PC1 correspond à la broche **A4** "Arduino". Laisser le **PC1** flottant (sans relier à quelque chose) peut produire des résultats de conversion aléatoires. La connexion de **PC1** à **GND** devrait afficher **0** pour la **valeur ADC**. La connexion de **PC1** à la broche **+3,3V** devrait afficher une valeur proche de **4095 (0x0FFF)**, la plage complète pour une valeur de 12 bits).

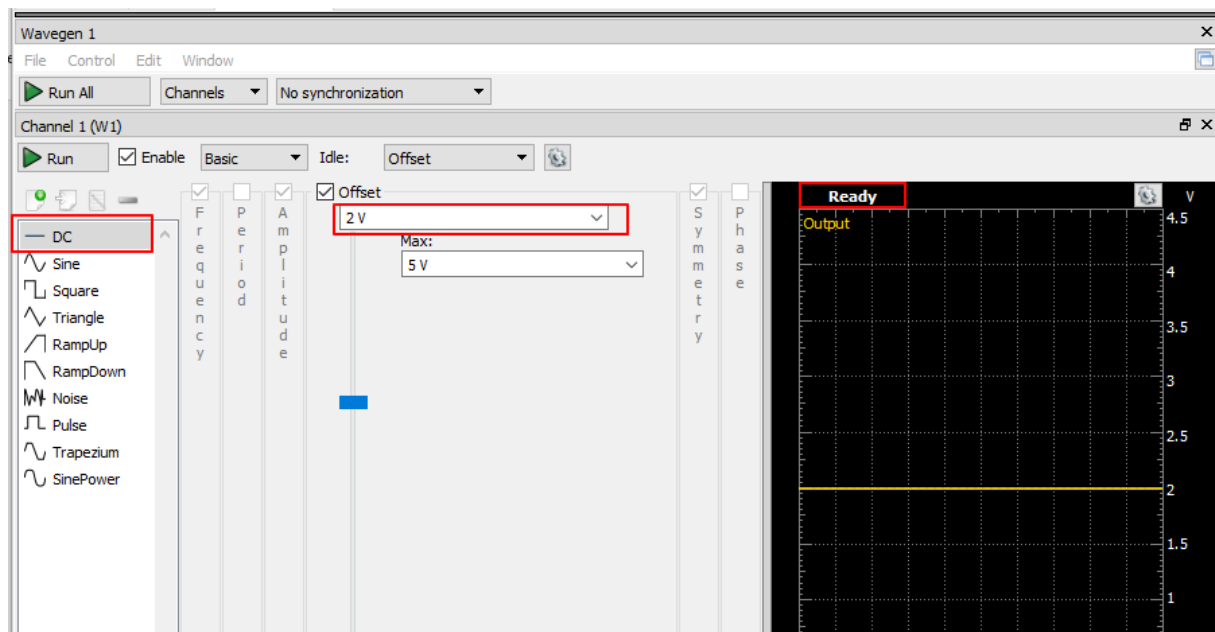
Si ces tests simples réussissent, alors il semble que tout fonctionne bien.

Pour modifier l'entrée de l'ADC en continu, vous pouvez câbler un potentiomètre comme indiquer sur la figure ci-dessous :



De même on peut utiliser le boîtier **Analog Discovery 2**, en réalisant le branchement ci-dessous :





4. Monitoring avec STM Studio

4.1 Installation de STM Studio

STM Studio® est un logiciel qui peut afficher les valeurs des variables d'un programme en temps réel. Bien entendu, il y a quelques limites. En général, vous pouvez surveiller :

- Variables allouées statiquement (c'est-à-dire globales) uniquement
- À une fréquence d'échantillonnage maximale de 1 kHz (1 ms)

C'est encore un moyen incroyablement puissant pour trouver des bugs ou pour régler la boucle de calcul dans les applications de contrôle (par exemple, régler un contrôleur PID).

Tout d'abord, vous devez installer STM Studio. Vous pouvez obtenir STM Studio sur le site web de ST :

- http://www.st.com/content/st_com/en/products/development-tools/software-...
- STM-STUDIO-STM32 version 3.5.1

Comme nous ne pouvons monitorer que des variables globales, ajoutons une variable globale pour stocker le résultat de l'ADC :

```
#include "stm32f0xx.h"
#include "bsp.h"
#include "delay.h"
#include "main.h"

static void SystemClock_Config(void);

// variables globales
uint16_t ADC_result;

int main()
{
    // configuration de horloges du system
    SystemClock_Config();

    // Initialiser la liason serie : Console de Debug
    BSP_Console_Init();

    // test du printf
    mon_printf("La Console est ready!\r\n");

    // Initiaaliser et lancer ADC sur Pin PC1
    BSP_ADC_Init();
    mon_printf("ADC est ready!\r\n");

    // Boucle Infini
    while(1)
    {
        // Tant que la conversion est en cours :
        // attendre que le bit b2 du registre ADC interrupt and
        // status register (ADC_ISR)
        // passe a '1'
        while ((ADC1->ISR & (1<<2)) != (1<<2));

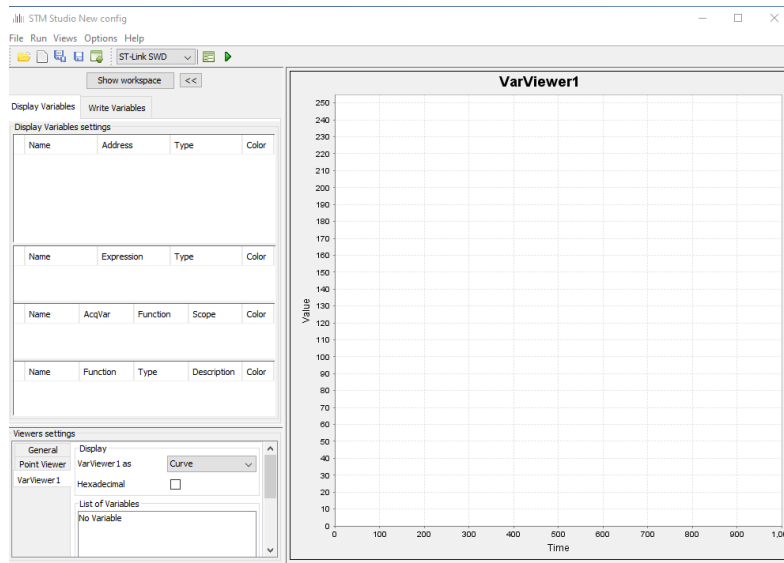
        // afficher le resultat de la conversion sur la console
        ADC_result = ADC1->DR;
        mon_printf("Valeur ADC = %d\r\n", ADC_result);

        // attendre environ 200 ms
        BSP_DELAY_ms(200);
    }
}
```


Construisez le projet, vérifiez que vous n'avez pas d'erreurs ou d'avertissements, puis affichez votre console série. Vérifiez dans votre terminal que le programme fonctionne bien, comme avant.

4.2 Configuration de STM Studio

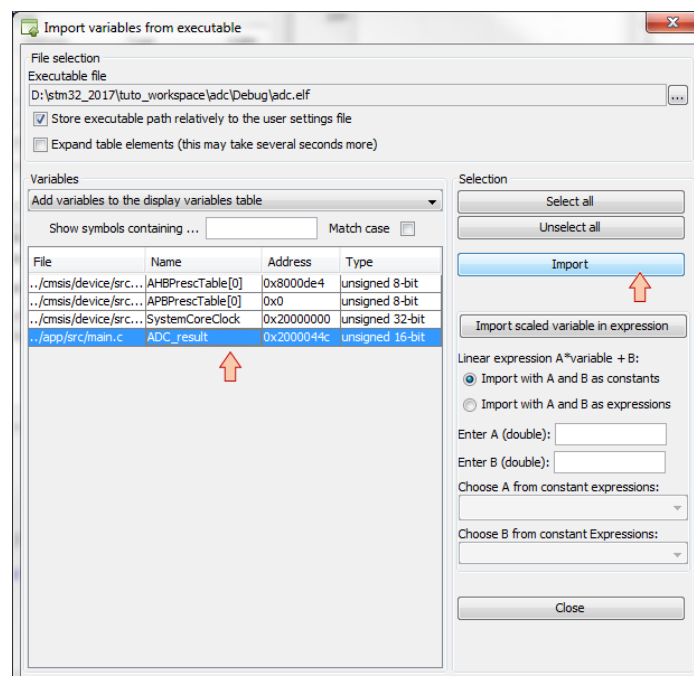
Lancez l'application STM Studio à partir du menu de démarrage de Windows.



Aller dans le menu principal puis **File → Import variables** ou cliquer 

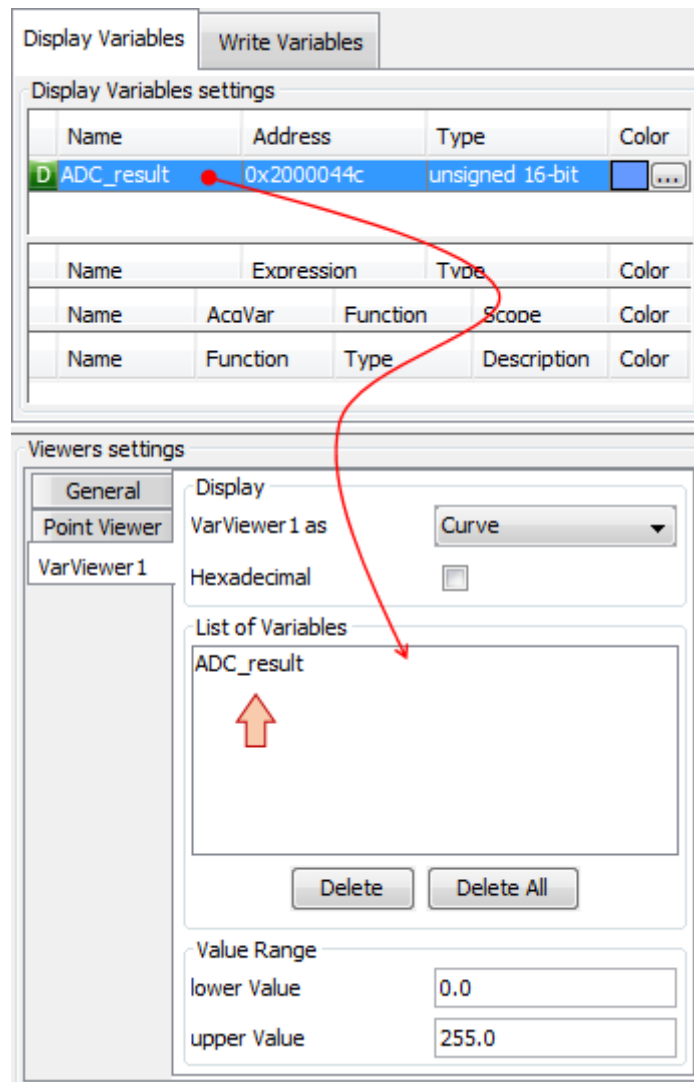
Dans la fenêtre **Import variables from executable**, utilisez le bouton  à droite du champ Fichier exécutable pour rechercher le fichier exécutable de votre projet **.elf**. Vous trouverez le fichier **.elf** dans le sous-répertoire suivant du dossier de votre projet, sous le dossier **\Debug**.

STM Studio vous présente ensuite une liste de toutes les variables que vous pouvez "espionner" :

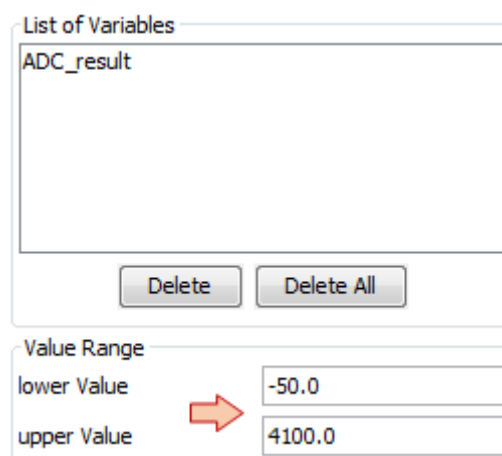



Sélectionner la variable **ADC_result** puis cliquez sur le bouton Import. Puis fermer cette fenêtre.

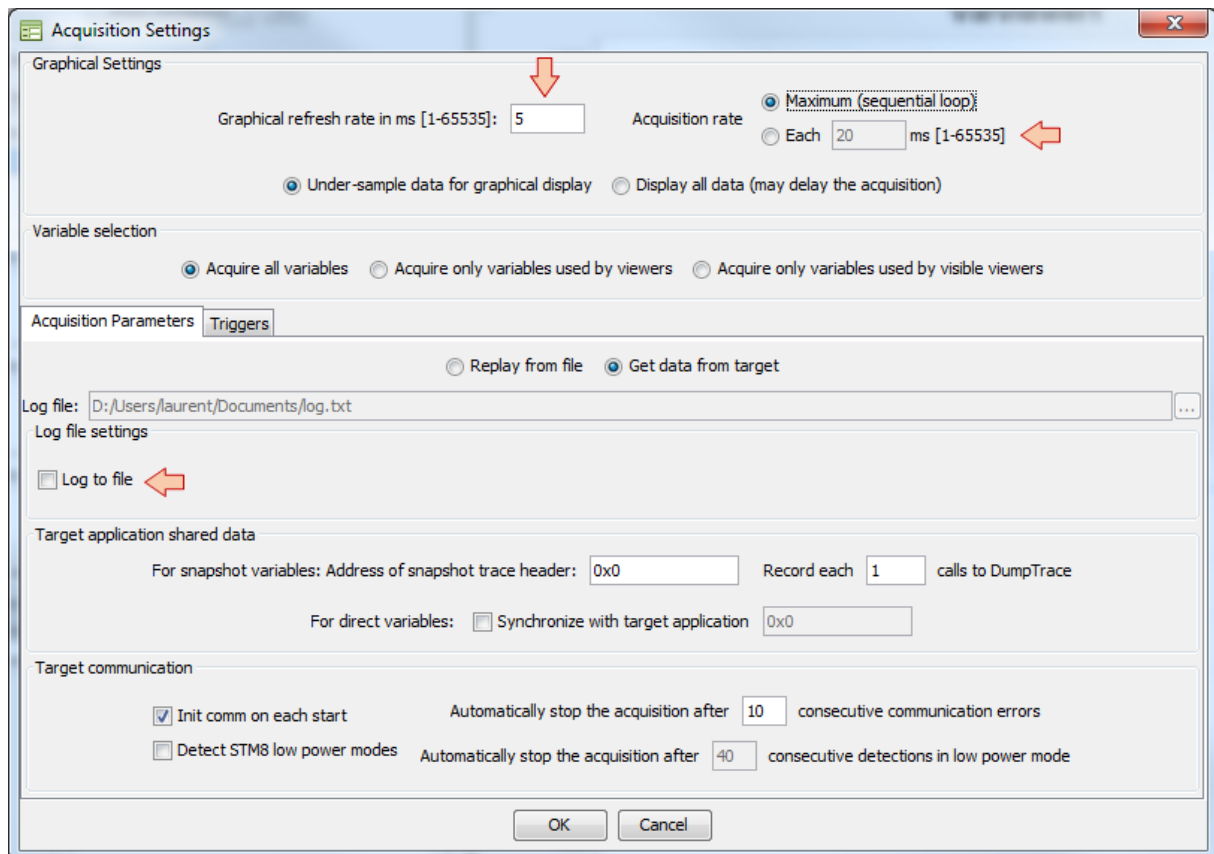
Glissez et déposez la variable **ADC_result** de la zone Variable settings à la zone **VarViewer 1** :



Définissez ensuite la plage de visualisation en fonction de ce que vous attendez de la variable. L'ADC 12 bits fournit des valeurs de 0 à 4095.



Allez maintenant dans le menu **Option** → **Acquisition Settings** ou cliquez 

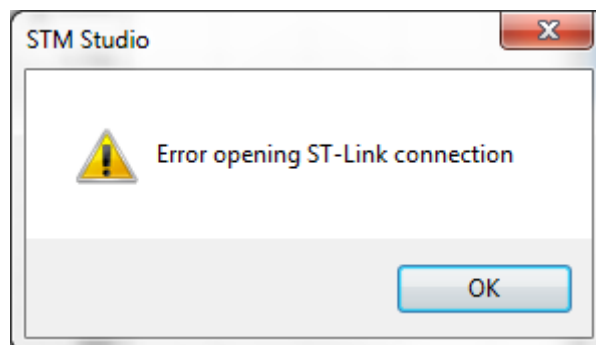


Réglez le taux de rafraîchissement graphique (**Graphical refresh rate**) sur 5 ms et décochez l'option "**Log to file**". Vous pouvez également modifier le taux d'acquisition en conséquence. Cliquez sur OK pour fermer la fenêtre.

4.3 Commencer le monitoring

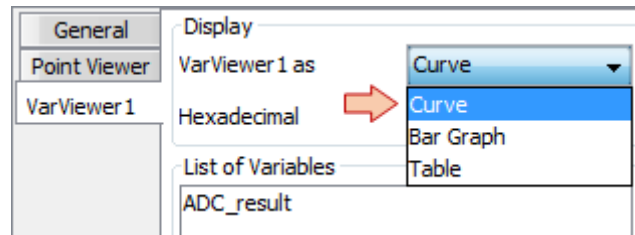
Allez dans le menu **Run → Start** ou cliquez sur le bouton 

Si vous obtenez la fenêtre ci-dessous, assurez-vous de terminer la session de **Debug sous STM32CubeIDE** et de réessayer. Vous ne pouvez pas avoir les deux connectés au ST-Link en même temps




Vous pouvez maintenant surveiller les fluctuations de la tension appliquée sur la broche PC1, comme avec un oscilloscope. Vous pouvez même vous assurer que STM Studio et les données que vous imprimez dans la console sont les mêmes (comme il se doit).

Vous pouvez également jouer avec les différentes options dont vous disposez pour afficher les données (courbe, graphique à barres, tableau) :



Vous pouvez également modifier les paramètres d'acquisition, vous pouvez zoomer (in/out) sur les courbes, et même enregistrer les données dans un fichier à des fins d'analyse ou de rapport (Excel, Matlab...).

Notez que vous ne pouvez pas flasher le code pendant que STM Studio est en cours d'exécution.

Il n'est pas nécessaire de fermer STM Studio, il suffit d'appuyer sur le bouton  avant de faire clignoter un nouveau code, puis de relancer le processus de surveillance.

STM Studio est désormais lié au fichier **.elf** et mettra automatiquement à jour les adresses variables si nécessaire.

5. Résumé

Dans ce tutoriel, vous avez configuré l'ADC pour une conversion continue sur un seul canal et vous utilisez STM-Studio pour surveiller en temps réel le résultat de la conversion.