STMicroelectronics

Info2 : STM32

TP5 L'ADC

Ce qui est écrit dans le programme BUT1

Apprentissage visé : être capable de concevoir un programme organisé en fonctions et implantable dans une cible de type microcontrôleur
Programmation sur cible :
– Convertisseurs
Contenu : ADC.

Le but de ce TP5 Info2

Pour l'apprentissage visé : Configurer l'ADC3 pour qu'il convertisse par interruption et sur 8 bits la tension présente sur le bit de port PF6. Cet octet sera affiché en décimal sur l'écran.

Matériel utilisé

Kit de développement : STM32F429 discovery : 32-Bit Cortex®-M4F - 180Mhz – 2MB Flash – 3x12 bits 16 channels 2.4MSPS ADC – 2x12 bits DAC – 12 x 16 bits timers and 2x32bits timers - SPI - I²C - 500 µA/MHz
Ecran : QVGA : 320x240 pixels - 262K color TFT - 24-bit RGB Parallel Pixel Output 8 bits-per-pixel (RGB888) – 4 wires resistive touch screen

M. Garcier

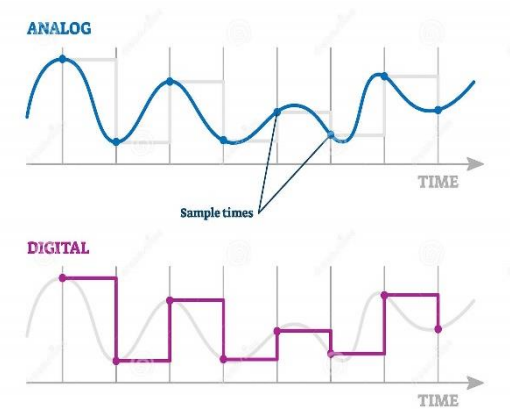
Table des matières

A. ADC

Le STM32 que nous utilisons dispose de 3 ADC 12 bits appelés ADC1, ADC2 et ADC3.
Pour ce TP nous utiliserons l'ADC3.

Un ADC (Analogic To Digital Converter) est un composant qui convertit un signal analogique en différentes valeur numériques.

Voyons d'abord la différence entre signal analogique et signal numérique :

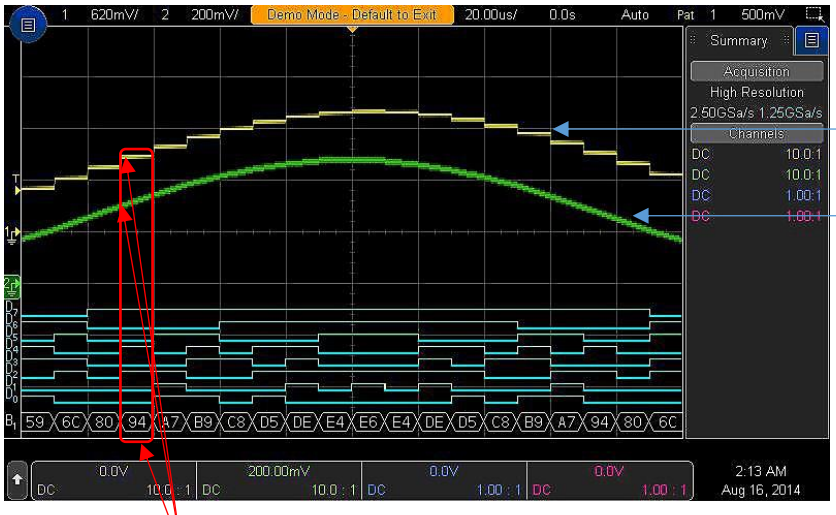


Le signal en bleue est un signal analogique. Sa valeur évolue au cours du temps et est constituée d'une infinité de valeurs. Le son capté par un micro, ou reproduit par un haut-parleur est un signal analogique.

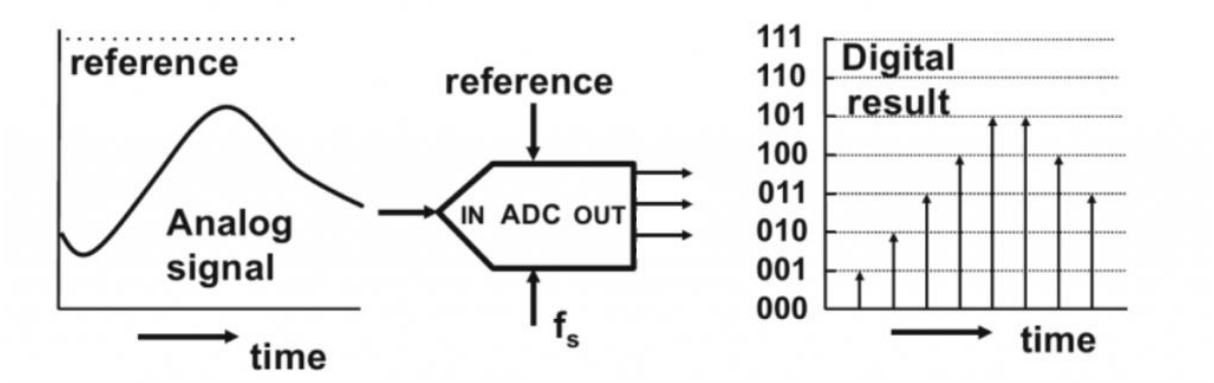
Le signal en violet est « l'équivalent » numérique du signal analogique. On l'appelle numérique car à intervalles réguliers (sample times), une mesure du signal analogique est faite. La mesure/le nombre correspondant est mémorisé jusqu'à la mesure suivante. En électronique on appelle ces mesures des échantillons (sample en anglais). C'est un signal numérique.

On peut observer que l'on passe d'une infinité de valeurs sur la courbe bleue à seulement 8 samples sur la courbe violette. On peut penser que le signal numérique ne ressemble pas beaucoup au signal analogique. En réalité des « règles » qui seront vues l'an prochain permettent de définir la fréquence d'échantillonnage (c'est-à-dire le nombre de points par seconde) F_s qui sera suffisante pour que le signal numérique représente au mieux le signal analogique.

Voilà un aperçu d'un signal sinusoïdal et du signal numérique correspondant sur un oscilloscope :



Voilà l'exemple d'un ADC 3bits.



Un temps de conversion existe toujours. C'est le temps qu'il faut l'ADC pour prendre un sample puis obtenir la valeur binaire correspondante. Afin d'éviter d'attendre dans le main que l'ADC ait fini de convertir on activera l'interruption dans le NVIC de Cube MX, ce qui permettra au programme de venir récupérer le résultat de la conversion dans l'ISR de l'ADC.

Dans ce TP, le signal sera appliqué au bit de port **PF6** qui correspond à l'entrée de l'ADC3.

B. Projet Cube MX

Voilà maintenant plusieurs projets que vous créez. Vous commencez donc à avoir une certaine habitude et vous serez désormais moins guidés.
Vous reprendrez le sujet du TP1 pour créer un nouveau projet dans le Workspace I:\INFO2_STM32 et dont le nom de votre projet : Info2_TP5_1

Dans cube MX vous chercherez comment :

- Autoriser l'ADC3 sur IN4 (qui correspond au bit de port PF6).
- Prescaler ⇒ laisser le réglage par défaut.
- Résolution ⇒ 8bits (l'ADC est sur 12 bits mais par commodité pour la suite du TP on le limite à 8 bits).
- Les autres paramètres seront laissés à leur valeur par défaut.
- Penser à autoriser l'interruption dans NVIC.

Générez le code, compilez-le et vérifiez que vous n'avez pas d'erreur.

Attention : Nombreux sont ceux qui oublient de continuer le paramétrage du projet dans STM32 Cube IDE et en particulier, le clic droit sur le nom du projet pour les include. Si vous ne vous souvenez plus comment faire, reportez-vous sur sujet du TP1

C. Codage

Voilà des étapes qui vont vous aider à atteindre l'objectif d'afficher la valeur en base 10 de l'ADC sur l'écran :

- Créer une variable sur 8 bits dans votre main.c appelée ResultatADC. Comme son nom l'indique, vous vous servirez plus tard de cette variable pour récupérer la valeur de l'ADC dans l'ISR.
- Mettre une valeur en dur dans ResultatADC et l'afficher sur l'écran grâce à la fonction itoa comme vous l'avez déjà fait dans les TP précédents. Compilez puis tester.
- Afin de pouvoir utiliser la variable ResultatADC en dehors du fichier main.c il est nécessaire d'ajouter la ligne de code suivant dans le fichier main.h qui se trouve dans le dossier Inc

  extern char ResultatADC ;
- L'ISR de l'ADC3 se trouve dans le fichier stm32f4xx_it.c
  Voici la fonction qui pourra être utilisée dans votre main et/ou ISR :
  HAL_ADC_Start_IT(…);
  HAL_ADC_GetValue (…) ;

  En vous servant de l'aide STM32F439xx_HAL.chm avec Sumatra.exe et éventuellement d'internet trouvez comment utiliser ces 2 fonctions.
- Modifier le main() pour que le programme n'affiche plus une valeur en dur mais la valeur réelle de l'ADC

D. Câblage et test

Pour finaliser les tests une résistance variable servant de pont diviseur variable, appliquera une tension comprise en 0 et 3V sur le bit de port PF6. 3 Câbles Dupont permettront de faire la connexion entre la carte du STM32 et la résistance variable que vous irez chercher dans les casiers à composants au fond du couloir. Vous trouverez les Dupont en GE210.
En faisant varier la valeur de la résistance variable à l'aide d'un petit tournevis, vous devez voir la valeur sur votre écran évoluer entre 0 et 255