TP n°3 : UART, DAC, ADC

Objectifs de la séance

- **Configuration d'une liaison série (UART)**
- **Configuration et utilisation du convertisseur numérique/analogique**
- **Configuration et utilisation du convertisseur analogique/numérique**
- **Application**
- **Utilisation du debugger, visualisation des registres et des mémoires**

Matériel requis :

-Une plateforme **nucleo-board STM32F446RE**

Organisation de la séance :

1) Création d'une liaison série sur carte Nucleo-64
2) Rappels – ADC, DAC
3) Conversion Numérique/Analogique
4) Conversion Analogique/Numérique
5) Conclusion / Bilan

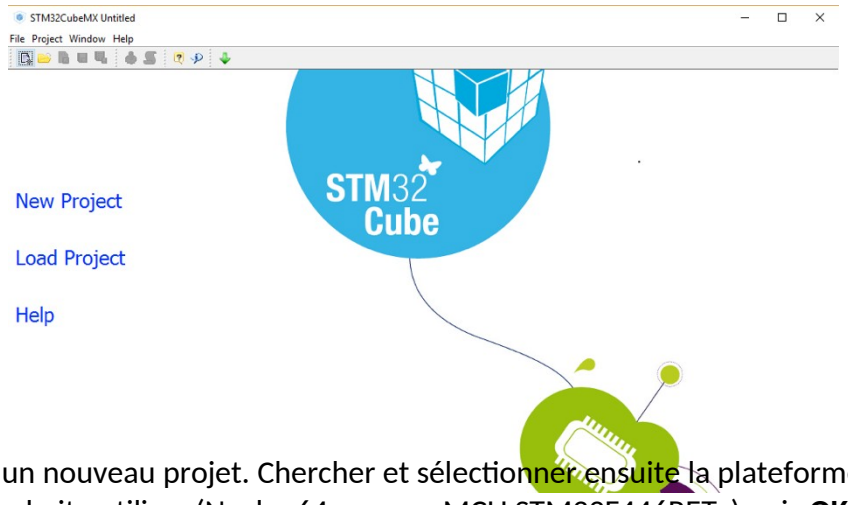
Tout au long de ce TP, des questions vous sont posées. Prenez le temps d'y répondre et de prendre des notes.

Note : une fois la première partie portant sur les liaisons série terminée, si vous ne disposez pas de fils pour tester une connexion entre deux broches, alors votre encadrant vous donnera des instructions pour simuler la sortie et afficher le résultat en passant par TeraTerm.

I. Programmation de la communication série UART

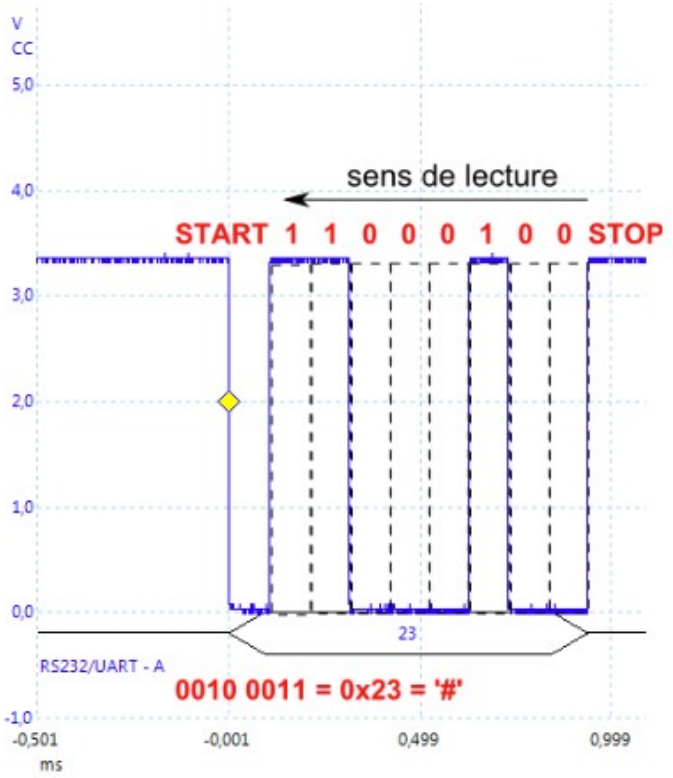
Exercice 1 : Setup du port de communication série

1. Lancer STM32CubeMX



2. Créer un nouveau projet. Chercher et sélectionner ensuite la plateforme que l'on souhaite utiliser (Nucleo64 avec un MCU STM32F446RETx) puis **OK**.
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**.
6. Choisir port USART2. Précisez :
   - Quelle est la broche d'émission ? de réception ?
7. Paramétrer USART2 de manière à avoir :
   - Liaison asynchrone ;
   - une communication série sur 7 bits ;
   - pas de contrôle de flux ;
   - une vitesse de 9600 bauds ;
   - mode full duplex.

Q1. Dessinez dans votre compte-rendu comment le synoptique de la connexion entre le PC hôte et la carte STM32.
Q2. Combien de fils sont nécessaires à cette communication ?
Q3. A quoi correspond les 9600 bauds ?
Q4. Sans transmission à quel état est le bus ?
Q5. Représenter les chronogrammes des caractères I, U et T. A titre indicatif, on donne le chronogramme du caractère dièse ci-dessous.



8. Exporter la configuration sous uVision keil
9. Programmer la partie USER CODE afin de d'envoyer un message de type printf sur la console du PC via un client de type TERATERM. On utilisera pour réaliser cette fonction, la fonction `HAL_UART_Transmit` disponible dans la bibliothèque stm32f4xx_hal_uart.h
10. Faites vérifier vos développements par l'encadrant.

Q6. Consigner dans votre rapport le code écrit.
Q7. Brancher un oscilloscope sur la broche TX de la carte (à côté du connecteur CN4). Relever les chronogrammes pour les caractères 1 et A.
Q8. Mesurer les codes ascii des caractères 1 et A et comparé à la table.
Q9. Mesurer la durée d'un bit. Conclusion.

11. On désire mesurer pratiquement la vitesse réelle de transmission du port série.

Q10. Identifier la partie du code permettant de configurer la vitesse de transmission de la liaison série. Consignez dans votre rapport.
Q11. Quel est le caractère à envoyer, le plus propice à cette mesure ?
Q12. Pour chaque vitesse de transmission du tableau, mesurez à l'aide de l'oscilloscope et d'une sonde la durée d'un bit.

Vitesse V de transmission	Durée d'un bit théorique (BTH) en ms	Durée d'un bit mesuré (BM) en ms

1200 bauds		
2400 bauds		
4800 bauds		
9600 bauds		
19200 bauds		
38400 bauds		
56800 bauds		
115600 bauds		

Q13. Tracer les courbes BTH=f°(V) et BM=f°(V). Quel est l'erreur entre BM et BTH ? En déduire la précision de la vitesse de transmission ?
Q14. A partir d'une transmission d'un caractère, définir comment est réalisé le bit de start ?
Q15. A partir d'une transmission d'un caractère, définir comment est réalisé la condition de stop ?
Q16. Quelles sont les règles à respecter entre un émetteur et un récepteur ?

II. Rappels : ADC, DAC

Un convertisseur analogique-numérique permet de faire l'acquisition de signaux issus du monde extérieur et de les traduire sous forme numérique. Son rôle est donc de convertir une tension V, généralement comprise entre 0V et +Vref, une tension de référence. La valeur numérique issue de la conversion est comprise entre 0 et 2^N-1 avec N le nombre de bits de résolution du convertisseur.

Le microcontrôleur STM32F446RE possède notamment :

-3 convertisseurs A/N 12 bits à 16 canaux
-1 convertisseur N/A 12 bits à 2 canaux

Microcontroller features

- STM32F446RET6 in LQFP64 package
- ARM®32-bit Cortex®-M4 CPU with FPU
- Adaptive real-time accelerator (ART Accelerator) allowing 0-wait state execution from Flash memory
- 180 MHz max CPU frequency
- VDD from 1.7 V to 3.6 V
- 512 KB Flash
- 128 KB SRAM System
- 4 KB SRAM Backup
- Timers General Purpose (10)
- Timers Advanced-Control (2)
- Timers Basic (2)
- SPI (4)
- I2S (2)
- USART (4)
- UART (2)
- USB OTG Full Speed and High Speed
- CAN (2)
- SAI (2)
- SPDIF-Rx (1)
- HDMI-CEC (1)
- Quad SPI (1)
- Camera Interface
- GPIO (50) with external interrupt capability
- 12-bit ADC (3) with 16 channels
- 12-bit DAC with 2 channels

Question : Sachant que Vref=3,3V, et que la résolution des convertisseurs est de 12 bits, combien vaut le quantum q, c'est à dire la plus petite tension pouvant être numérisée ?

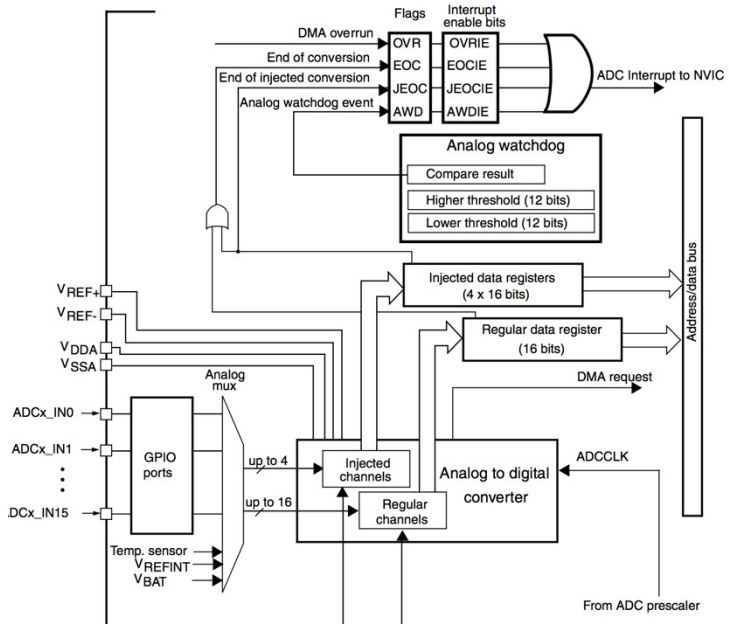
Idéalement pour un CAN, on rappelle que $N = \frac{V}{Vref} 2^N - 1$;

Pour un CNA, le principe est dual, $V_{pleine_{échelle}} = V_{max} - V_{min} = (2^N - 1).q$;

- Les convertisseurs du STM32F446RE
  - CAN (ADC)

    Il est possible de paramétrer le CAN en fonction des besoins de son application.
    Par exemple, on peut choisir :
    -la résolution (12, 10, 8, 6 bits),
    -les canaux utilisés (16 sources externes + des sources internes (Vref, température…)),
    -le mode d'acquisition (unique, continu, discontinu, balayage de plusieurs canaux…),
    -l'utilisation d'un trigger externe (*injected channel*),
    -l'alignement des données dans le,
    -l'utilisation d'une DMA,
    -…

    Sur la figure suivante est illustrée l'architecture d'un ADC du STM32F :



Nous avons pu voir qu'il était très important de définir une fréquence d'échantillonnage supérieure à 2 fois la fréquence maximale (ou bande) du signal d'entrée de manière à respecter le critère de Shannon-Nyquist.
L'horloge de référence est ADCCLK. C'est à partir de sa fréquence et d'une valeur du registre SMP qu'est déterminée le temps d'échantillonnage.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					SMP18[2:0]			SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31: 27 Reserved, must be kept at reset value.

Bits 26:0 **SMPx[2:0]:** Channel x sampling time selection
These bits are written by software to select the sampling time individually for each channel.
During sampling cycles, the channel selection bits must remain unchanged.

Note: 000: 3 cycles
001: 15 cycles
010: 28 cycles
011: 56 cycles
100: 84 cycles
101: 112 cycles
110: 144 cycles
111: 480 cycles

Le CAN est basé sur la technologie SAR (Registre à Approximations Successives).
Le temps de conversion est donc égal au nombre de division à effectuer, correspondant à la résolution du convertisseur (ici 12 bits donc 12 cycles).
Le temps total de conversion se détermine de la manière suivante :

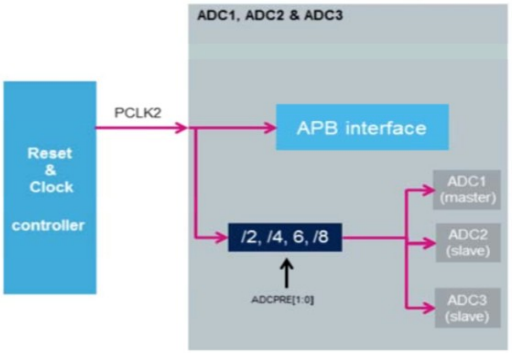
T_total_conversion = Sampling_time + 12

Exemple :

Si ADCCLK = 30MHz, et que le temps d'échantillonnage est de 3 cycles (SMPx[2 :0]=000), le temps total de conversion vaut :

T_total_conversion = 3 + 12 = 15 cycles, soit 0,5 us (2 Méch/s).

Il existe deux domaines d'horloge au sein du DAC. L'un permet l'interface avec le microcontrôleur (registres) et est régit par PCLK2, l'autre sert à la partie analogie et est régit par ADCCLK. L'horloge de la partie analogique est réglable par l'intermédiaire d'un prescaler.



Interruption :
Selon les besoins, il est possible de déclencher une interruption à chaque fin de conversion (flag EOC).

Finalement, on résume la configuration de l'ADC par les commandes suivantes :

1)Choisir l'entrée
-Activation et configuration de la broche d'entrée analogique
-Choix des canaux réguliers
2) Réveil de l'ADC
-Activation de l'ADC
3) Choix du mode de fonctionnement et du canal
4) Configuration de l'interruption
-Permettre ou non la génération d'une interruption en fin de conversion EOC
5) Déclenchement de la conversion
6) Récupération de la donnée numérisée

- CNA (DAC)

Les microcontrôleurs de la famille STM32 intègrent pour la plupart des DAC avec différentes configurations en fonction de la série STM32Fx. Dans le microcontrôleur STM32F446RE, il y a un DAC 12bits avec deux canaux de sortie.

Ses principales caractéristiques sont rappelées :

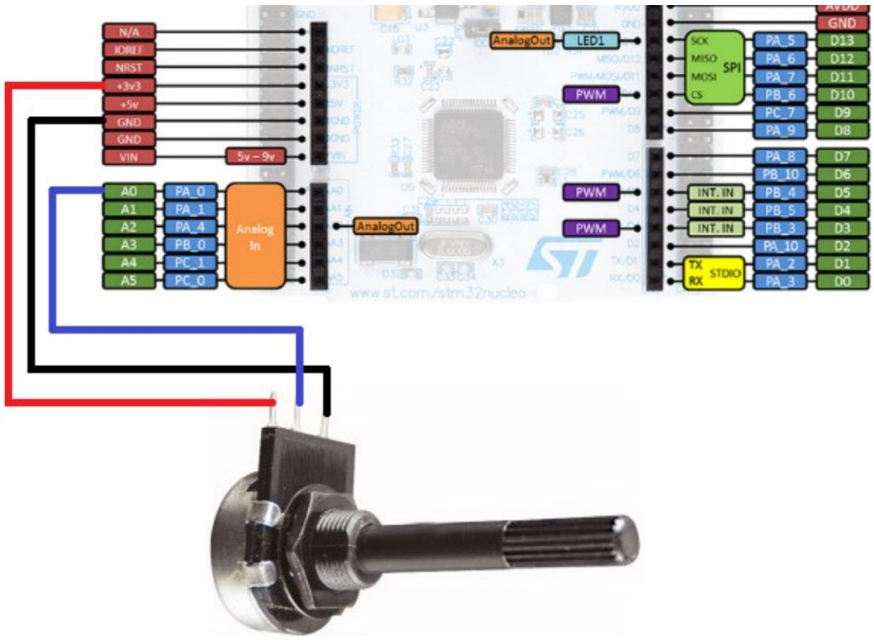
.left or right data alignment in 12-bit mode
.synchronized update capability
.noise-wave generation
.triangular-wave generation
.dual DAC channel independent or simultaneous conversions
.DMA capability for each channel
.external triggers for conversion

Pour toute information complémentaire, vous pouvez vous référer à la documentation constructeur du STM32F446RE :
http://www.st.com/content/ccc/resource/technical/document/datasheet/65/cb/75/50/53/d6/48/24/DM00141306.pdf/files/DM00141306.pdf/jcr:content/translations/en.DM00141306.pdf

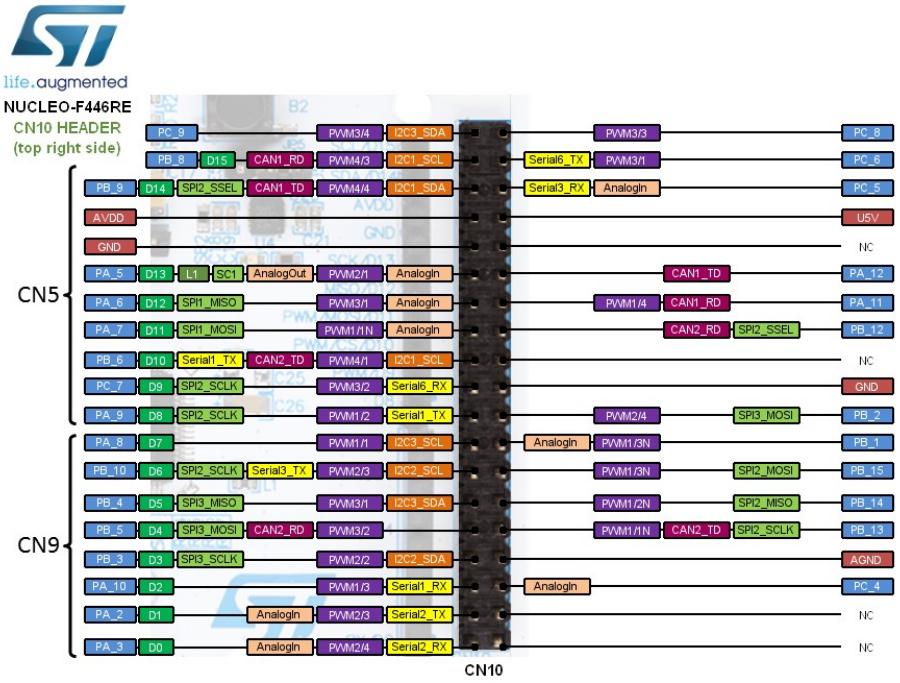
III. Exercice 1 : Conversion analogique/numérique

On se propose de manipuler dans un premier temps le convertisseur analogique numérique.

Objectif : On souhaite faire l'acquisition d'une tension issue d'un potentiomètre via une des broches analogiques disponibles (Ax). En fonction de la tension issue du potentiomètre, on allumera plus ou moins de LEDS D0 à D2 qui seront connectées à l'aide d'une carte fille dédiée (cf. plus bas). Pour cela, vous réaliserez le montage comme indiqué sur la figure suivante :



Tout le matériel sera fourni par l'enseignant.



❑ 1ière partie : Configuration sous STM32CubeMx

-Lancer STM32CubeMx
-Créer un nouveau projet pour **notre board Nucleo64, STM32F446RE**.
-Effectuer de suite une remise à zéro de la configuration des broches via **Pinout/Clear Pinouts.**

-Dans Clock Configuration, vérifier que les horloges SYSCLK est à 84MHz. Relever les fréquences des horloges desservant les périphériques, notamment PCLK1, PCLK2.

Question : Selon vous, quelle est l'horloge qui est envoyée vers les périphériques comme les ADC ?

-Configurer l'ADC1 de manière à pouvoir faire l'acquisition du signal sur l'entrée IN0.
-Dans le sous-menu **configuration**, cliquer sur **ADC1**. Les paramètres de configuration de l'ADC sont visualisables et modifiables à partir de cette fenêtre.

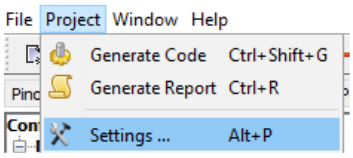
Question : Qu'est ce qu'un prescaler ? Quelle est l'horloge de référence qui est reliée à l'entrée du prescaler ?

-Dans le sous-menu **configuration**, cliquer sur **ADC1**. Les paramètres de configuration de l'ADC sont visualisables et modifiables à partir de cette fenêtre.
-Configurer l'ADC pour pouvoir effectuer des conversions continues.
-Autoriser les interruptions générées par l'ADC au niveau du contrôleur NVIC.

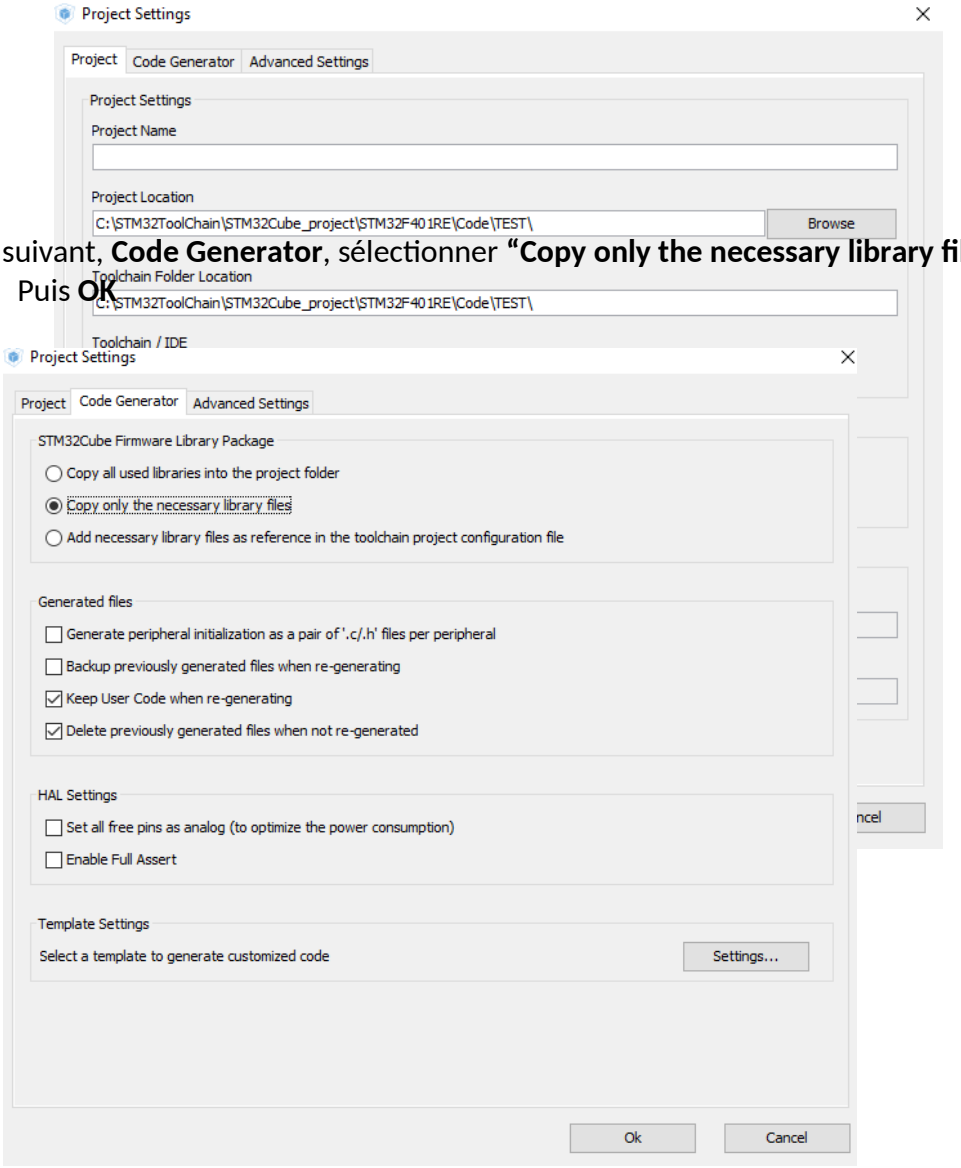
-Dans le contrôleur RCC, autoriser les interruptions (RCC global interrupt).

-Vérifier dans le contrôleur NVIC que les interruptions sont bien autorisées.

Une fois que tout est configuré, générer le code du projet de la même manière que les TPs précédents :

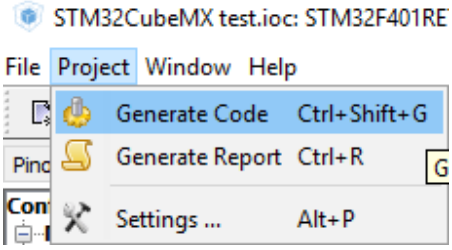


Dans **Project Name** indiquer le nom de votre projet.
Dans **Project location**, indiquer le répertoire de votre projet où sera générer le code.
Dans **Toolchain/IDE**, indiquer **MDK-ARM v5**.
Dans **Linker Settings**, vous pouvez observer la taille des zones de Heap et de Stack (Pile).



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.

- 2nde partie : Développement du code sous Kiel µVision

-Compléter le code généré pour réaliser le programme demandé.
-Faire valider par l'enseignant

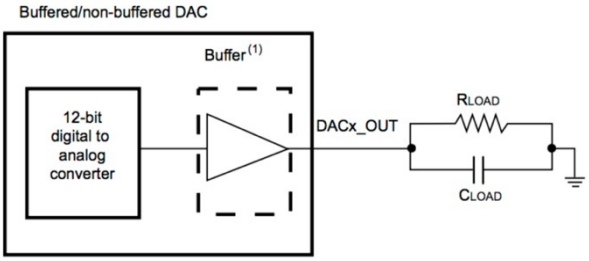
IV. Exercice 2 : DAC - Génération de signal triangulaire

Nous avons vu dans le cours que le DAC permet la génération de signaux comme du bruit, un signal PWM, etc. Dans cet exercice, on souhaite configurer le convertisseur numérique analogique pour générer un signal triangulaire. L'amplitude max du signal sera la moitié de la pleine échelle et la période du signal sera de 81,92 ms.
La période du signal sera calculée par l'intermédiaire du timer. Celui-ci générera un Update Event à chaque fin de cycle de comptage qui lancera la conversion d'un nouvel échantillon au niveau du DAC. La configuration du Timer devra être précise (valeur du compteur, fréquence de comptage via le prescaler..).

❑ 1ière partie : Configuration sous STM32CubeMx

-Sous STM32CubeMX, créer un nouveau projet.
-Configurer l'horloge et le reset (RCC).
-Ajouter ensuite un DAC.

-Dans **Parameter settings** du DAC, vous pouvez observer trois paramètres de configuration :
1) Output buffer,
2) Trigger
3) Wave generation mode (si un trigger est sélectionné)



Le premier permet l'utilisation ou non d'un buffer en sortie du convertisseur, permettant d'adapter l'impédance de sortie sans avoir recours à un ampli op.
Le second permet de spécifier si un trigger de synchronisation (matériel ou logiciel) est utilisé.
Le troisième paramètre n'est disponible que si un trigger est utilisé. Il permet de choisir le type de signal généré (triangle ou bruit) ainsi que l'amplitude maximale de la sortie. La génération du signal sera donc déclenchée et synchronisée à partir du signal de trigger.

-Générer le code et ouvrir Kiel µvision

❑ 2nde partie : Configuration sous STM32CubeMx

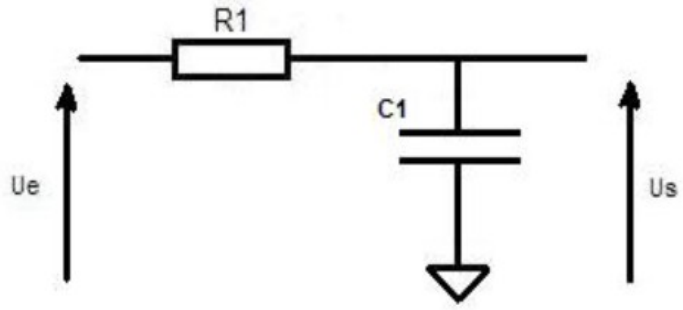
-Compléter le code généré pour réaliser le programme.
-Faire valider par l'enseignant.

V. Exercice 3 : Traitement de signaux

Nous allons reprendre le signal triangulaire généré lors de l'exercice 2. Sous uVision, on configurera le Timer 2 avec un prescaler de 2 et une période de comptage de 5 (respectivement valeurs 3 et 6 sous STM32CubeMx). Le générateur de signal triangle occupera la pleine échelle du convertisseur soit 4096 pts (DAC_TRIANGLEAMPLITUDE_4095).

Question : Quelle sera la période du signal triangulaire ?

On connectera ensuite un réseau RC comme indiqué ci-dessous. On prendra R1 = 1k et C1= 220nF. Le signal triangulaire sera appliqué en entrée (Ue) et on observera la sortie (Us).



Sous STM32CubeMX :

-Ajouter un ADC pour récupérer le signal de sortie qui sera connecté à une entrée analogique (A0 par exemple). On autorisera les interruptions indiquant la fin de conversion.

-Ajouter un timer (Timer 1) qui permettra de déclencher périodiquement une interruption déclenchant la conversion d'une valeur par l'ADC. On choisira une valeur de Prescaler et de période pour avoir une UEV toutes les 0,1 ms (f=10kHz).

-Générer le code et lancer uVision

Sous uVision :

-Modifier le code pour pouvoir lancer le timer 1 en mode interruption et le sous-programme d'interruption permettant de démarrer les conversions numériques analogiques.

-Dans le sous-programme d'interruption de l'ADC, **récupérer la valeur lue dans une variable globale**.

-Lancer le debug, chercher à visualiser le signal numérisé **à l'aide de l'analyseur logique**.

Question : Expliquer la forme du signal de sortie (Us). Si besoin, regarder la FFT du signal d'entrée puis du signal de sortie à l'aide d'un oscilloscope présent dans la salle.