Table des matières

I) UTILISATION DE PRINTF ET SCANF MODE DEBUG DANS KEIL µVISION 2

I-1) CONFIGURATION PREABLABLE DANS L'IDE KEIL µVISION ... 2

I-1-A) MENU MANAGE RUN-TIME ENVIRONMENT (ICONE) ... 2

I-1-B) MENU « OPTION FOR TARGET »(ICONE) ... 3

I-2) EXEMPLE DE CODE UTILISANT LES FONCTIONS PRINTF ET SCANF ... 4

I-3) CONFIGURATION DU DEBUGGER ... 4

I-3-A) MENU « VIEW /SERIAL WINDOWS » ... 4

I-3-B) MENU « VIEW /WATCH 1 » ... 4

II) ENVOI DE MESSAGES SUR UN TERMINAL SERIE VIA LE PORT USB 6

II-1-A) DANS LE SENS STM32 VERS LE TERMINAL SERIE ... 6

II-1-B) DANS LE SENS TERMINAL SERIE VERS STM32 ... 7

II-2) EXEMPLE COMPLET DANS UN ENVIRONNEMENT COMPLEXE ... 8

II-2-A) DEMARRAGE DANS STM32CUBEMX ... 9

II-2-B) DANS L'IDE KEIL µVISION ... 12

II-2-C) RETOUR DANS STM32CUBEMX POUR LE DAC, DMA ET TIM2 ... 14

II-2-D) RETOUR KEIL µVISION POUR LA PROGRAMMATION FINALE ... 19

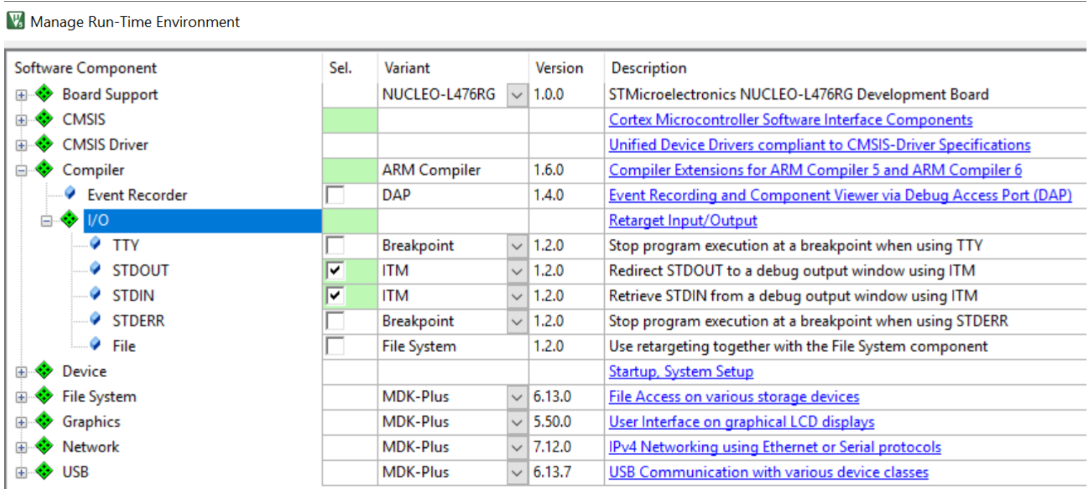
II-3) ECRANS DE RESULTATS ... 21

I) Utilisation de printf et scanf mode debug dans Keil µVision

I-1) Configuration préablable dans l’IDE Keil µVision

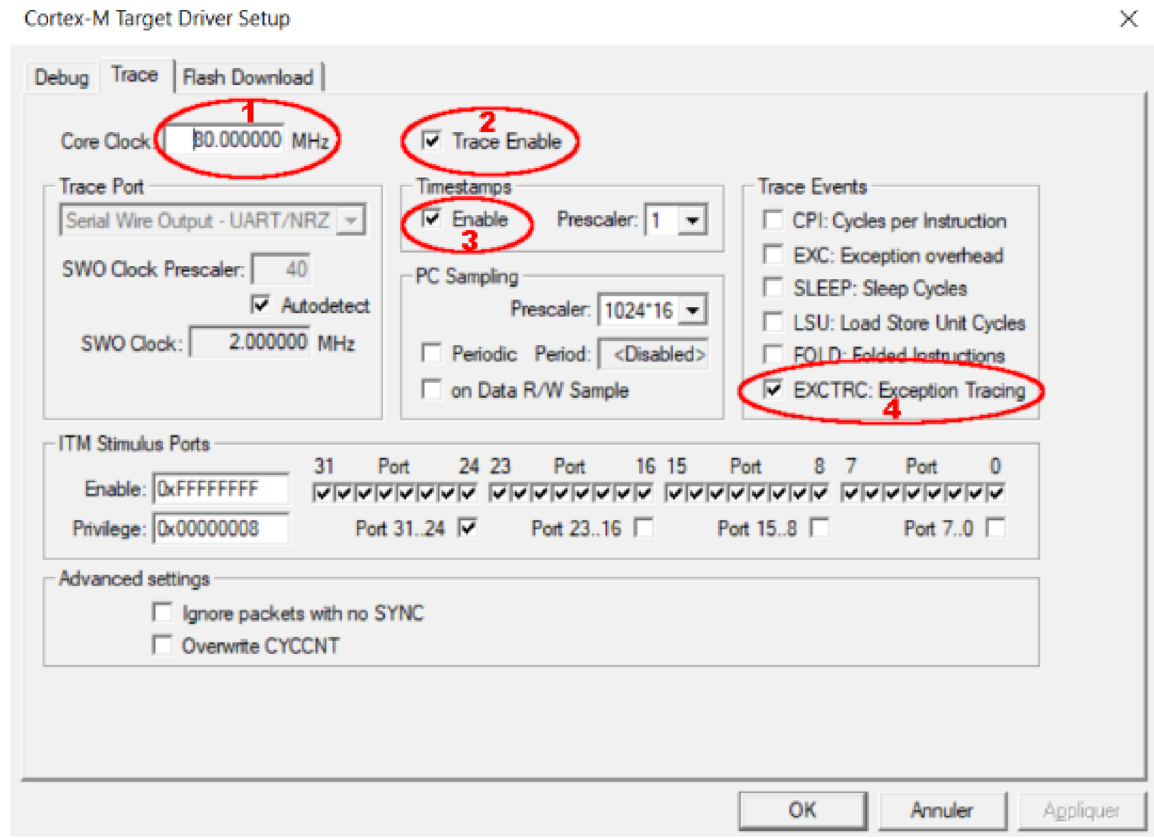
I-1-a) Menu Manage Run-Time Environment (icone)

Dans les options du compilateur, configurer les option I/O de la manière suivant :



I-1-b) Menu « Option for Target »(icone)

- aller dans l'onglet « Debug » ;
- appuyer sur le bouton « Settings » ;
- aller dans l'onglet « Trace » ;
- **1- régler l'horloge** avec celle de notre processeur, dans cet exemple, le core est programmé avec une fréquence de 80 MHz (*Le débogueur utilise cette valeur pour calculer les délais de capture et d'affichage de traces (tels que le courant et la tension par exemple), même lorsque la trace est désactivée. Il configure également le périphérique UART de capture de traces de l'unité de débogage. Il doit être défini sur la fréquence d'horloge système que votre application cible utilise*.);
- **2- cocher** « **Trace Enable** » (*pour capturer la trace via la broche SWO (Serial Wire Output) et pour faire apparaitre les autres options de la fenêtre*) ;
- **3- cocher** « **Enable** » dans la partie « Timestamps » (*ceci permet de créer des informations d'horodatage des traces de debugging et de choisir la fréquence d'horodatage grâce à la liste de choix « Prescaler » adjacente. Ici j'ai mis 1 (fréquence maximale d'horodatage)*) ;
- **4- cocher** « **EXCTRC** » dans la partie « Trace Events » (*la section « Trace Events » permet de suivre les événements spéciaux. La fenêtre Compteurs d'événements affiche le résultat. EXCTRC = Exception Tracing : Le compteur incrémente les exceptions, les entrées d'interruption et les sorties d'interruption. La fenêtre Exceptions de trace affiche les statistiques*).
- **valider** les fenêtres avec le bouton « OK »



I-2) Exemple de Code utilisant les fonctions printf et scanf

Il faut inclure stdio.h

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stdio.h" // pour la fonction printf
/* USER CODE END Includes */
```

Et l'on peut alors utiliser les fonctions scanf et printf classiquement

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
int n =2020;
printf("Avant d'entrer dans la boucle while(1)...\n"); // pour le mode debug et sa fenêtre Debug Viewer et Watch
while (1)
{
  printf("Saisissez une valeur pour n : "); // pour le mode debug et sa fenêtre Debug Viewer et Watch
  scanf("%d", &n); // pour le mode debug et sa fenêtre Debug Viewer et Watch
  printf("Valeur de n : %i\n",n); // pour le mode debug et sa fenêtre Debug Viewer et Watch
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Il faut ensuite compiler le code et le télécharger dans le microprocesseur, comme à l'habitude.

I-3) Configuration du debugger

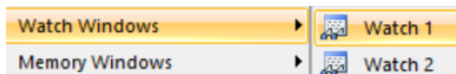
Lancer le debugger par « Crl+F5 ».

I-3-a) Menu « View /Serial Windows »

Afficher la fenêtre « **Debug (printf) Viewer** ».

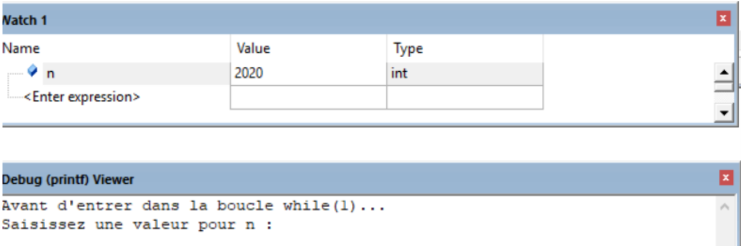
I-3-b) Menu « View /Watch 1 »

Afficher la fenêtre Watch1.

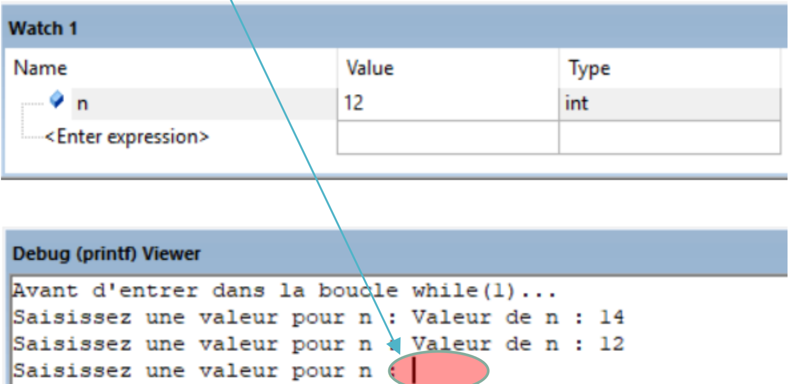


Ajouter la ou les variables dont vous voulez visualiser les valeurs, ici j'ai choisi « **n** » (tapez directement « **n** » dans la fenêtre « Watch 1 »).

Enfin dans la fenêtre « Watch 1 » avec le bouton droit choisir ou pas la visualisation en hexadécimal et disposer les fenêtres Watch 1 et Debug (printf Viewer) à votre guise.
Exécuter le code avec la touche « F5 ».
Avez l'exemple précédent, si vous n'avez pas mis de point d'arrêt, vous devez voir s'afficher :



Mettez votre curseur juste derrière le message « Saisissez une valeur pour n : » et saisissez ce qui vous chante…



II) Envoi de messages sur un terminal série via le port USB

Ce que nous visons dans ce chapitre, c'est la possibilité d'envoyer des données vers un terminal série et de les recevoir à partir d'un terminal série. Nous utiliserons le port USB en port virtuel et un logiciel de terminal série commet TERMITE ou CoolTerm par exemple.

Comme mentionné dans le *user manual* UM1724 concernant ma carte nucleo-L476RG, l'USART2 entre le STM32L476RG et le ST-LINK MCU est activé par défaut afin de prendre en charge le port COM virtuel.

Par conséquent :

- les données en provenance du PC sont reçues via la ligne RX (port PA3) ;
- Les données à destination du PC sont émises via la ligne TX (port PA2).

II-1-a) Dans le sens STM32 vers le terminal série

Pour transmettre des données dans le sens STM32 vers le terminal série, il est commode d'utiliser la classique fonction printf du langage c. Pour ce faire, nous allons redéfinir la fonction **fputc**() du C, car cette fonction est celle sur laquelle s'appuie la fonction **printf**().
Dans le fichier « **main**.c », redéfinirons tout simplement **fputc** en lui demandant d'envoyer les caractères reçus vers **l'USART2**.
Pour connaitre la fonction à appeler pour envoyer les caractères à l'USART2, ouvrons le fichier « stm32l4xx_hal_uart.c » et repérons les commentaires associés aux entrées/sorties.

```
==============================================================
                ##### IO operation functions #####
==============================================================
```

On trouve un peu plus bas, les fonctions à appeler lorsque l'on se trouve en « *Blocking mode* » (*c'est-à-dire lorsque l'on n'utilise ni les interruptions, ni le DMA pour envoyer les données à l'USART2*).
Nous voyons que la fonction qui convient est « **`HAL_UART_Transmit`**» et qu'elle dispose des paramètres suivants :

```
* @param huart   UART handle.
* @param pData   Pointer to data buffer (u8 or u16 data elements).
* @param Size    Amount of data elements (u8 or u16) to be sent.
* @param Timeout Timeout duration.
```

Nous appellerons cette fonction en lui passant :

- le handler du *driver* de l'USART2 (&huart2) ;
- L'adresse du caractère ch (paramètres de notre fonction printf) ;
- 1 (le nombre d'octets à envoyer) ;
- 1000 (le timeout).

Pour utiliser fputc, il faudra commencer par inclure la bibliothèque stdio.

```
/* Private includes -------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
/* USER CODE END Includes */
```

Puis il faudra redéfinir la fonction fputc :

```
/* Private user code -----------------*/
/* USER CODE BEGIN 0 */
int fputc(int ch, FILE *f)
{
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 1000);
  return ch;
}
/* USER CODE END 0 */
```

II-1-b) Dans le sens terminal série vers STM32

La première idée qui vient à l'esprit est de faire quelque chose de symétrique à ce que nous avons fait pour la fonction **printf** : récrire la fonction **fgetc** sur laquelle s'appuie la fonction **scanf**…
Mais pour la saisie des caractères depuis l'USART2, nous utiliserons **une technique non bloquante** dans laquelle nous réagirons uniquement sur interruption, lorsque l'USART2 nous enverra des caractères.
En cherchant un peu dans le fichier « stm32l4xx_hal_uart.c », nous découvrons une fonction qui s'appelle « __weak void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)». Le commentaire «*the HAL_UART_RxCpltCallback can be implemented in the user file* » nous informe que cette fonction doit être implémentée dans un fichier utilisateur et non pas modifiée dans « stm32l4xx_hal_uart.c ».

C'est ce que nous allons faire dans le fichier main. C'est cette fonction de callback qui sera appelée via l'interruption *USART2 Global interrupt* à chaque caractère reçu.

Pour son fonctionnement, nous doterons cette fonction :

- d'un *buffer* de réception d'une taille de 100 octets ;
- d'un *index* qui pointera sur la position libre dans le buffer ;
- D'une variable globale *BufReady* permettant d'indiquer la fin de réception.

Voici ce que devra faire notre fonction à chaque interruption :

- Vérifier que l'interruption est bien liée à l'USART2 et si c'est le cas :
  - Si l'index est égal à zéro, réinitialiser le buffer avec des '\0' ;
  - Si le caractère reçu est différent de « 0xA » (caractère de fin de ligne).
    - Alors
      - stocker le caractère reçu à la position index du buffer ;

            - Augmenter l'index de 1.
        - Sinon
            - Remettre l'index à zéro
            - Mettre *BufReady* à 1 ;
    - Fin si
    - Réarmer l'interruption dans tous les cas.

Dans le fichier main :

- nous armerons l'interruption en dehors de la boucle infinie ;
- nous bouclerons tant que *BufReady* sera égal à zéro.
- Quand *BufReady* passera à 1 :
    - nous afficherons le *buffer* ;
    - nous remettrons *BufReady* à zéro ;

II-2) Exemple complet dans un environnement complexe

En guise d'exemple, nous allons reconstruire le programme de l'étape 3 du tutoriel « *Expérimentation DAC-DMA-TIMER Nucléo-L476RG*».

Par défaut le nombre d'échantillons de la sinusoïde sera de 100, mais à tout moment l'utilisateur pourra choisir d'entrer une des 3 options suivantes :

- A ➔25 échantillons ;
- B ➔50 échantillons ;
- C ➔75 échantillons.
- Autre valeur saisie ➔ 100 échantillons.

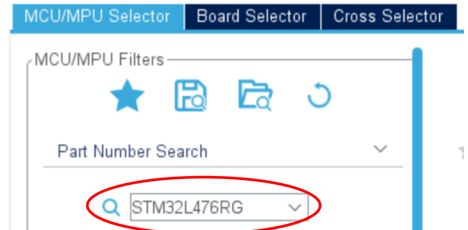
Nous utiliserons donc la fonction **printf** pour afficher les messages ci-dessus et nous utiliserons **le mécanisme non bloquant** pour la prise en compte des choix de l'utilisateur.

A la fin de ce tutoriel, nous aurons donc un exemple de mise en œuvre de la carte NUCLEO-L476RG dans l'environnement technique suivant : {USART2 + USART2 Global Interrupt + DAC1 + DMA + Tim2 + printf vers USART2 + mécanisme de récupération des données via USART2}.

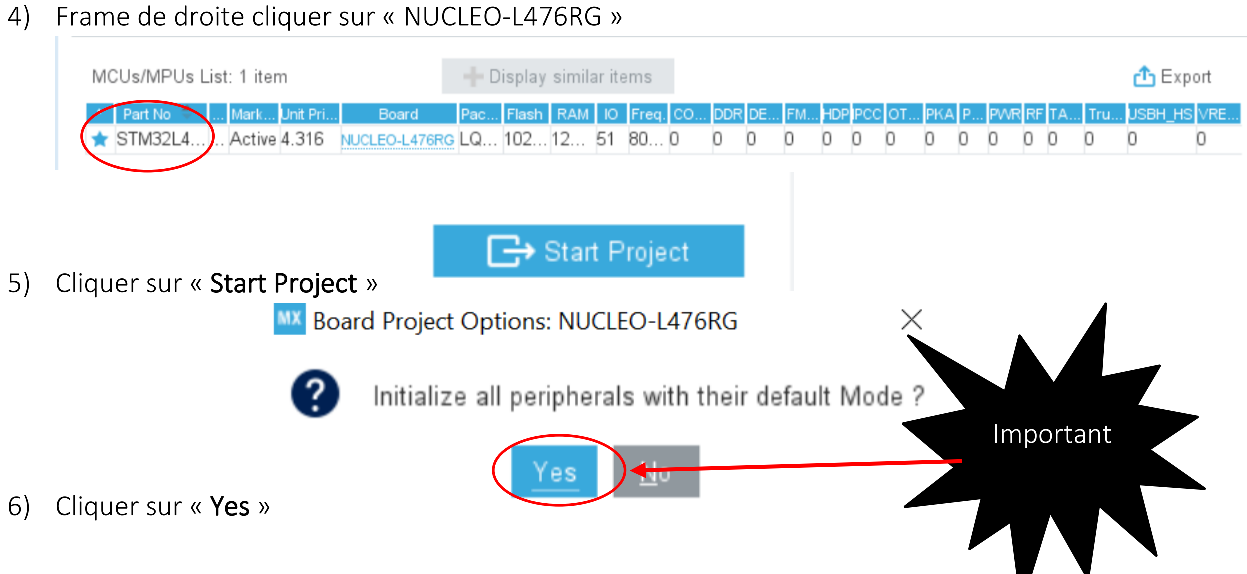
Nous récrivons entièrement le programme, ce qui rend ce tutoriel autonome. Il suffit de suivre les étapes ci-dessous.

II-2-a) Démarrage dans STM32CubeMX

1) Lancer STM32CubeMX
2) File / New Project (ou Ctrl+N)

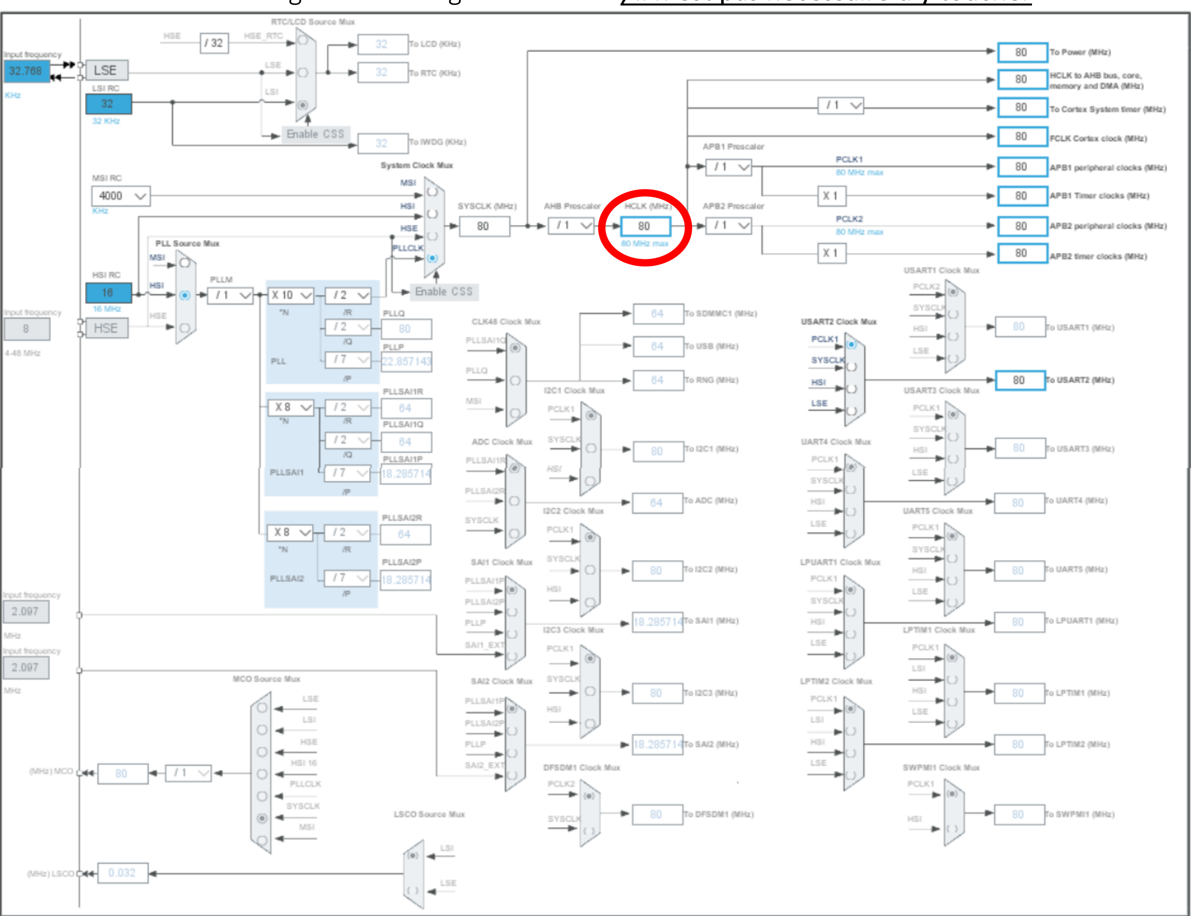


3) Rechercher notre carte Nucléo :
4) Frame de droite cliquer sur « NUCLEO-L476RG »

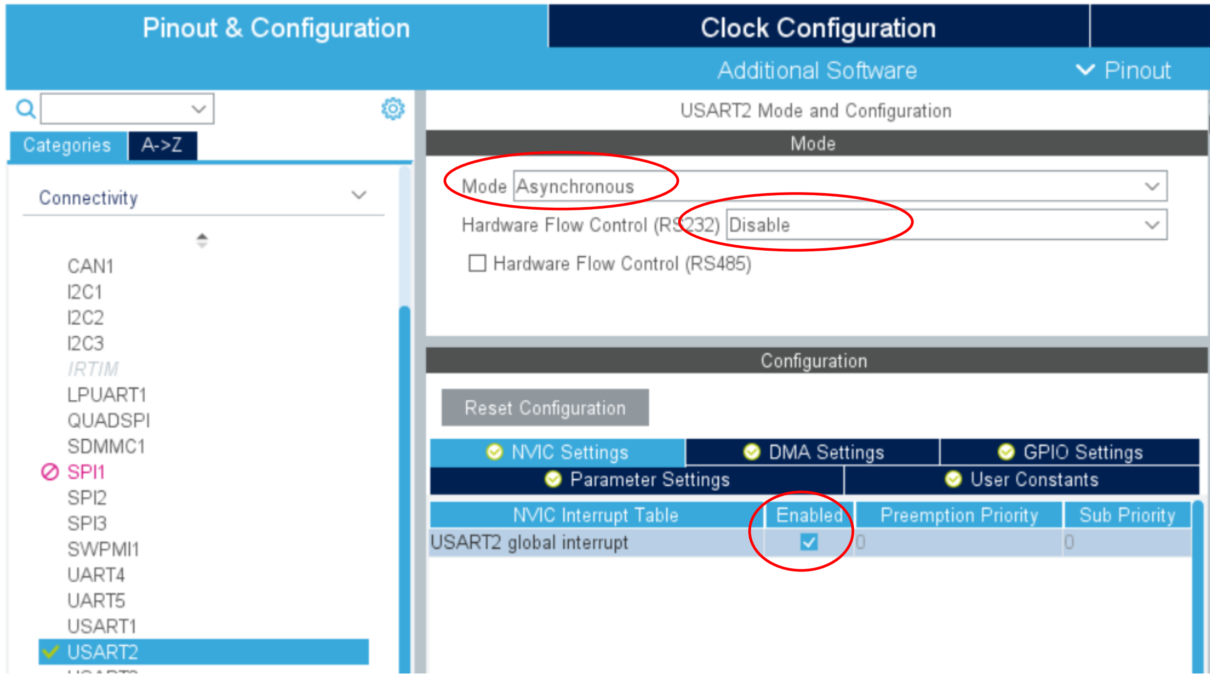


5) Cliquer sur « **Start Project** »
6) Cliquer sur « **Yes** »

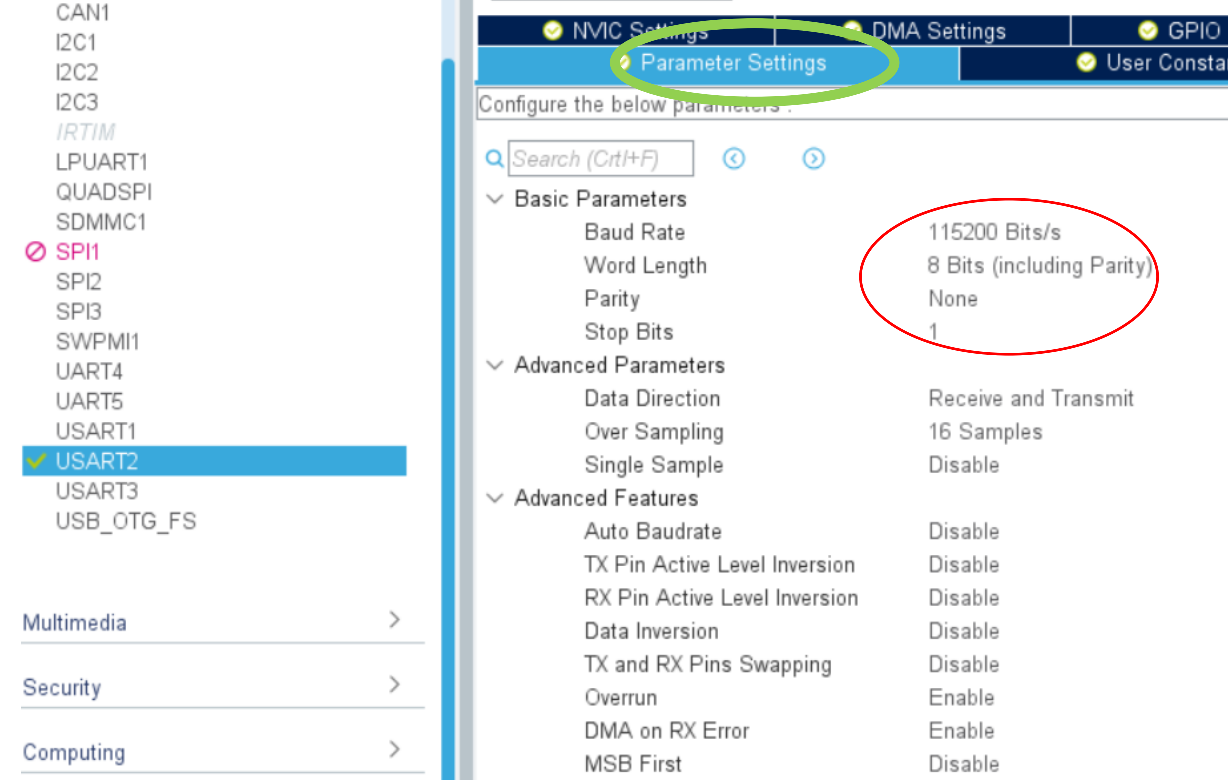
7) Par défaut l’arbre d’horloge est bien réglé sur 80 Mhz, <u>il n’est pas nécessaire d’y toucher</u>.



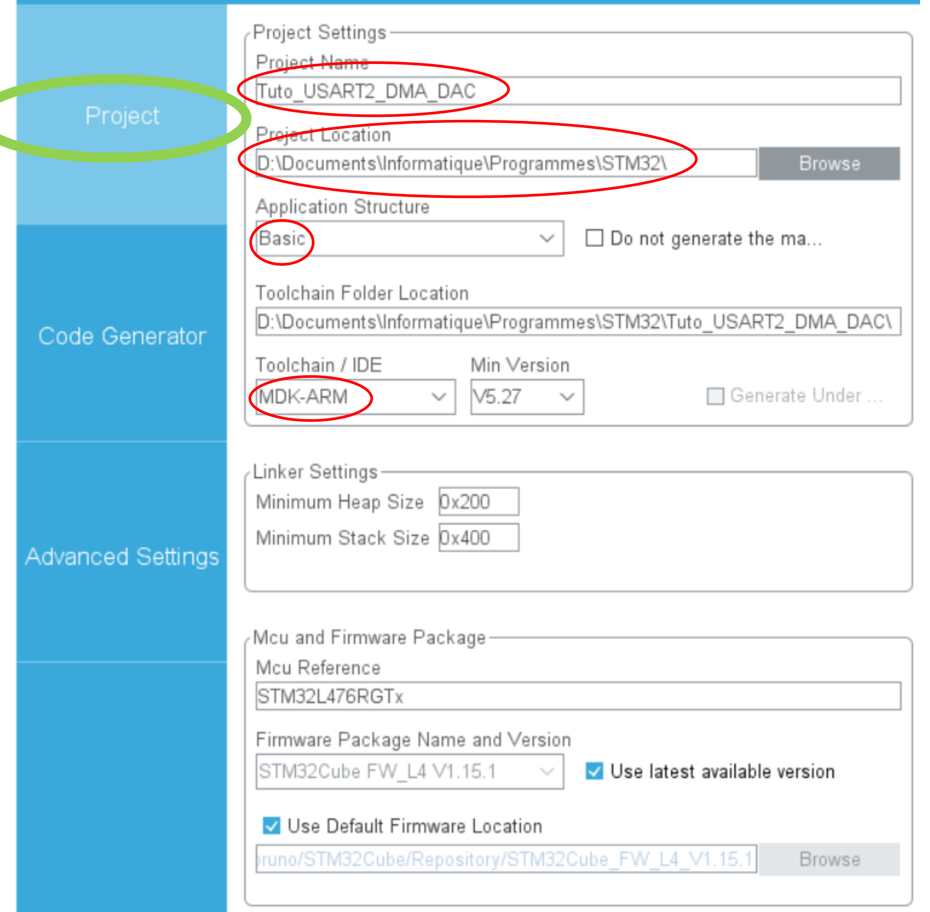
8) Dans « **Pinout & Configuration** », sélectionnez **USART2** et configurer comme ceci :



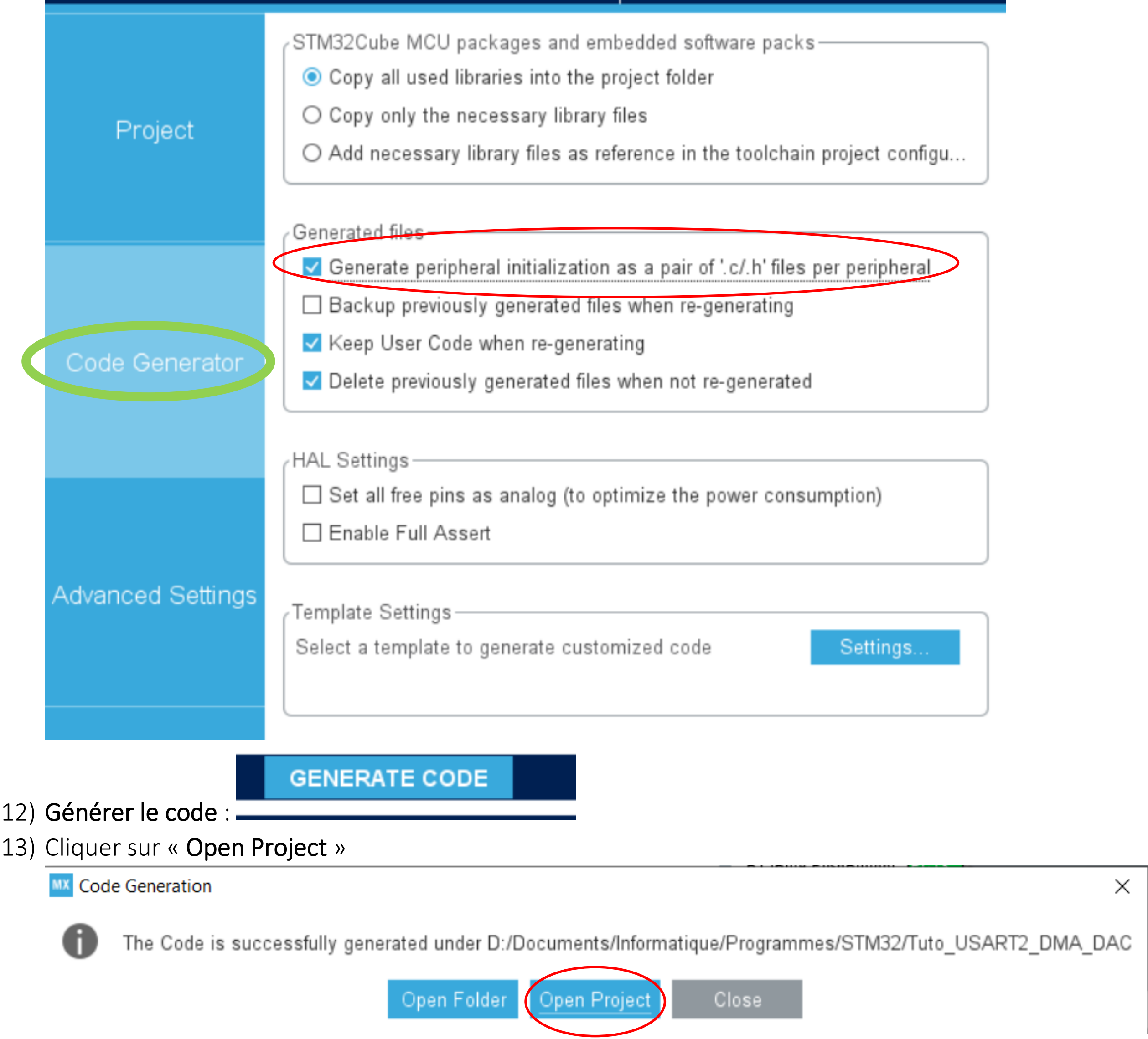
9) Dans l'onglet « **Parameter Settings** », configurer comme ceci :



10) Onglet « **Project Manager** », configurer comme ceci dans la partie «Project » :



11) Dans la partie « Code Generator », configurer comme ceci :



12) **Générer le code** :

13) Cliquer sur « **Open Project** »

II-2-b) Dans l'IDE Keil µVision

Appuyer sur « **F7** » = **Project/ Build target**

Vous ne devez obtenir aucune erreur :

```
Build Output
compiling stm32l4xx_hal_pwr_ex.c...
linking...
Program Size: Code=7196 RO-data=496 RW-data=16 ZI-data=1152
FromELF: creating hex file...
"Tuto_USART2_DMA_DAC\Tuto_USART2_DMA_DAC.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:49
```

Pour le programme et les besoins communication avec l'USART2, nous aurons besoin des librairies suivantes :

- **sdtdio** (pour printf, sprintf et putc notamment) ;
- **string** (pour strlen notamment).

Pour remplir le tableau d'échantillons de la forme d'onde sinusoïdale, nous aurons aussi besoin de la librairie **math**.

Dans les commentaires du programme j'ai fait précéder de **USART2**, tout ce qui a été ajouté pour les besoins de la communication série avec l'USART2 et de **DAC1**, tout ce qui a été ajouté pour la mise en œuvre de l'étape 3 du tutoriel « *Expérimentation DAC-DMA-TIMER Nucléo-L476RG* ».

Nous allons inclure ces librairies dans le code :

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>  // USART2 : printf, sprintf, putc
#include <string.h> // USART2 : strlen
#include "math.h"   //   DAC1 :pour utilisation de la fonction sinus
/* USER CODE END Includes */
```

Pour la fonction générant les échantillons, nous aurons besoin de la constante PI, nous allons la définir à l'endroit prévu pour :

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define PI 3.1415926 // DAC1 : pour utilisation de la fonction décrite dans AN3126
/* USER CODE END PD */
```

Pour réaliser le reste du code utilisateur, nous aurons besoin de déclarer un certain nombre de variables :

```
/* USER CODE BEGIN PV */
char BufIndex ;             // USART2 : index qui pointera sur la position libre dans le buffer de reception
char BufRecept[100] ;       // USART2 : buffer  de réception d'une taille de 100 octets
char buffer[100];           // USART2 : buffer de manoeuvre utilisé dans la boucle principale
uint8_t BufReady=0 ;        // USART2 : booléen : 1 si une ligne terminée pa LF (0xA) a été reçue, 0 sinon
uint8_t BufUSART2[2];       // USART2 : buffer utilisé par les fonctions HAL dédiées à l'USART2
uint32_t sine_val[100];     //   DAC1 : tableau 100 positions max, pour les valeurs de la forme d'onde sinusoidale
int n = 100 ;                 //   DAC1 : Nb échantillons
/* USER CODE END PV */
```

Voici maintenant les 3 fonctions utilisateur nécessaires à ce tutoriel :

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int fputc(int ch, FILE *f) { // USART2 : redéfinition de la fonction putc sur laquelle s'appuie printf
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 1000); // chaque caractère est redirigé vers l'USART2
  return ch;
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) //USART2 : fonction callback appelée à chaque réception de caractère
{
  int i ;
  if(huart->Instance==USART2) //UART en cours
  {
    if (BufIndex==0) {for (i=0;i<100; i++) BufRecept[i]=0;}//Effacer Rx_Buffer avant de recvoir des nouvelles données

    if (BufUSART2[0]!=0x0A) //La donnée reçue est différente de LineFeed
    {
      BufRecept[BufIndex++]=BufUSART2[0]; //ajoute la donnée au buffer
    }
    else
    {
      BufIndex=0; // index prêt pour une nouvelle ligne à récevoir
      BufReady=1; //les données sont prêtes à lire
    }
    HAL_UART_Receive_IT(&huart2, BufUSART2, 1);//réactiver l'interruption à chaque fois
  }
}
void get_sineval() // DAC1 : usage unique pour valorisation des samples ; n = nombre d'échantillons
{// Cf formule note d'application AN3126
    for (int i=0;i<n;i++) sine_val[i]=((sin(i*2*PI/n)+1)*2048);
}
/* USER CODE END 0 */
```

La première : **fputc** sert à réorienter la fonction **printf** vers l'USART2 ;
La deuxième : **HAL_UART_RxCpltCallback** sert à récupérer les lignes reçues via l'USART2 ;
La troisième : **get_sineval** sert à valoriser le tableau d'échantillons de la forme d'onde.

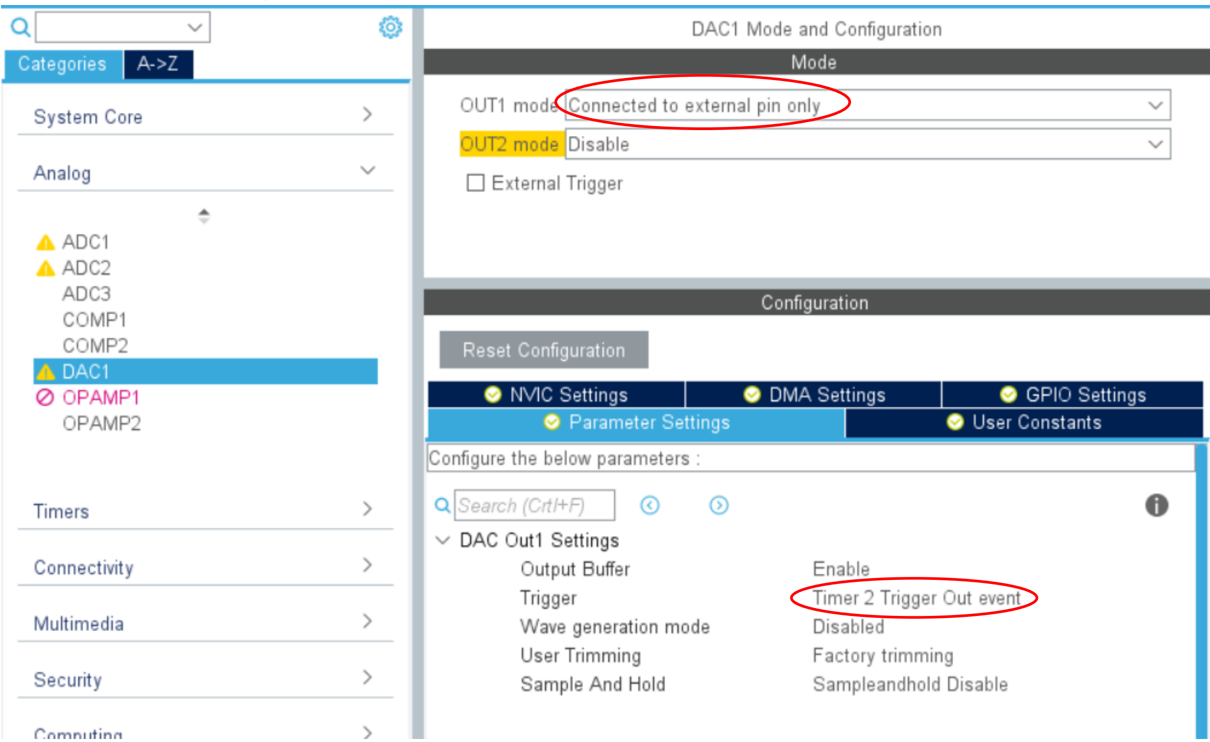
Compilons le code tel qu'il est pour vérifier qu'il n'y a pas d'erreur :

```
Tuto_USART2_DMA_DAC\Tuto_USART2_DMA_DAC.axf: Warning: L6989W: Could not apply patch sdcomp-29491-629360 to instruction VPOP
Program Size: Code=7196 RO-data=496 RW-data=16 ZI-data=1152
Finished: 0 information, 1 warning and 0 error messages.
FromELF: creating hex file...
"Tuto_USART2_DMA_DAC\Tuto_USART2_DMA_DAC.axf" - 0 Error(s), 1 Warning(s).
Build Time Elapsed:  00:00:05
```

Il y a juste un warning qui est dû à un bug connu du compilateur, nous n'y sommes pour rien et n'y pouvons rien, nous vivrons donc avec…

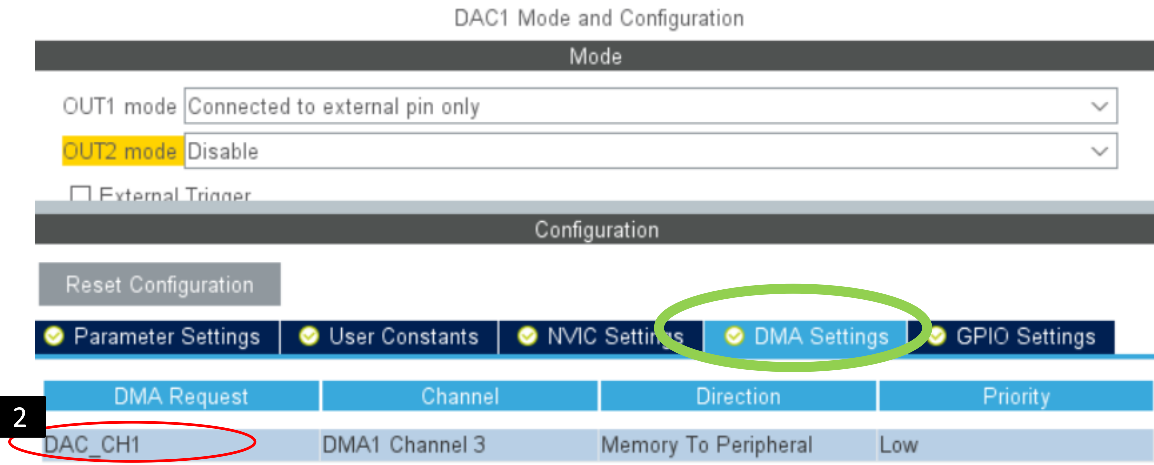
II-2-c) Retour dans STM32CubeMX pour le DAC, DMA et tim2

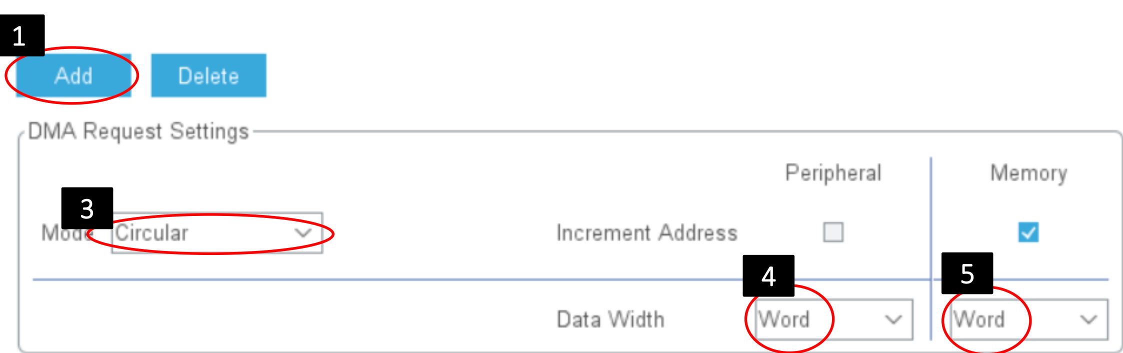
Dans « **Pinout & Configuration** », configurer le **DAC1** comme ceci :



Out1 mode ➔ Connected to external pin only
Trigger ➔Time 2 Trigger Out event

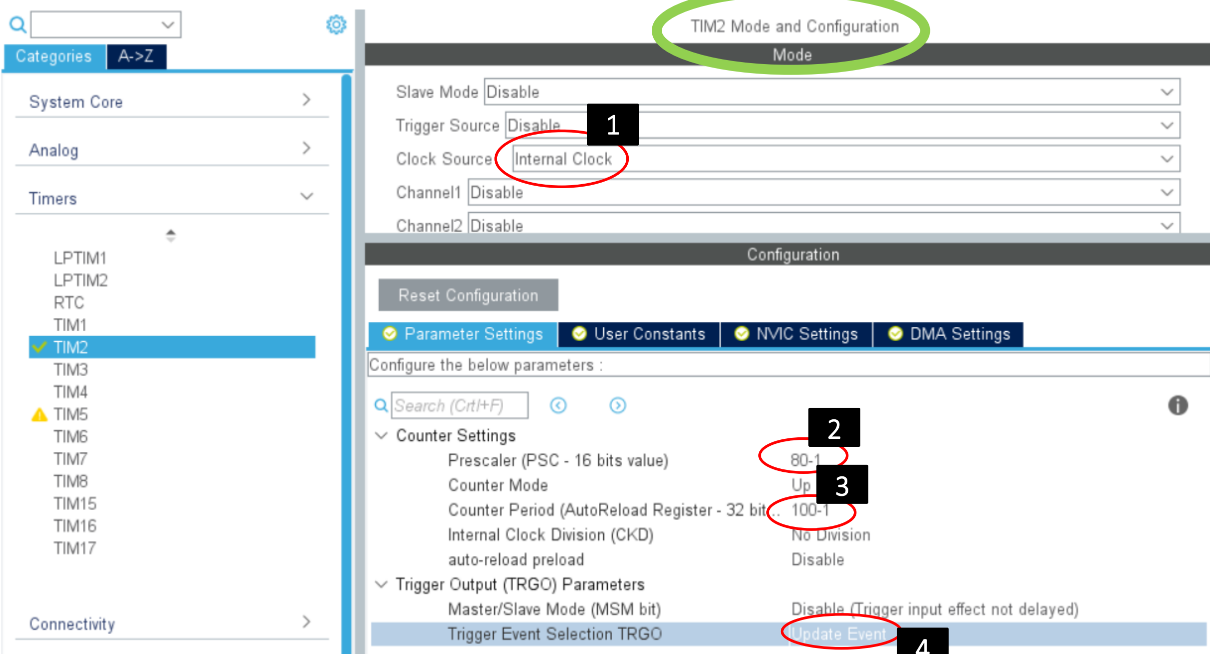
Dans l’onglet DMA Settings, configurer comme ceci :





1- Appuyer sur **Add** et
2- Choisir « **DAC_CH1** » ;
3- Choisir le Mode **Circular** (*Le mode circulaire permet d’avoir un flux de données continu. La source et les adresses de destination et le nombre de données à transférer sont automatiquement rechargés après chaque transfert complet.*);
4- Choisir **Word** pour **Peripheral Data Width** (*taille des données vers le DAC*) ;
5- Choisir **Word** pour **Memory Data Width** (*taille des données en mémoire*).

Il reste à configurer le **timer2**, le choisir dans l'onglet « **Pinout & Configuration** » et configurer ainsi :



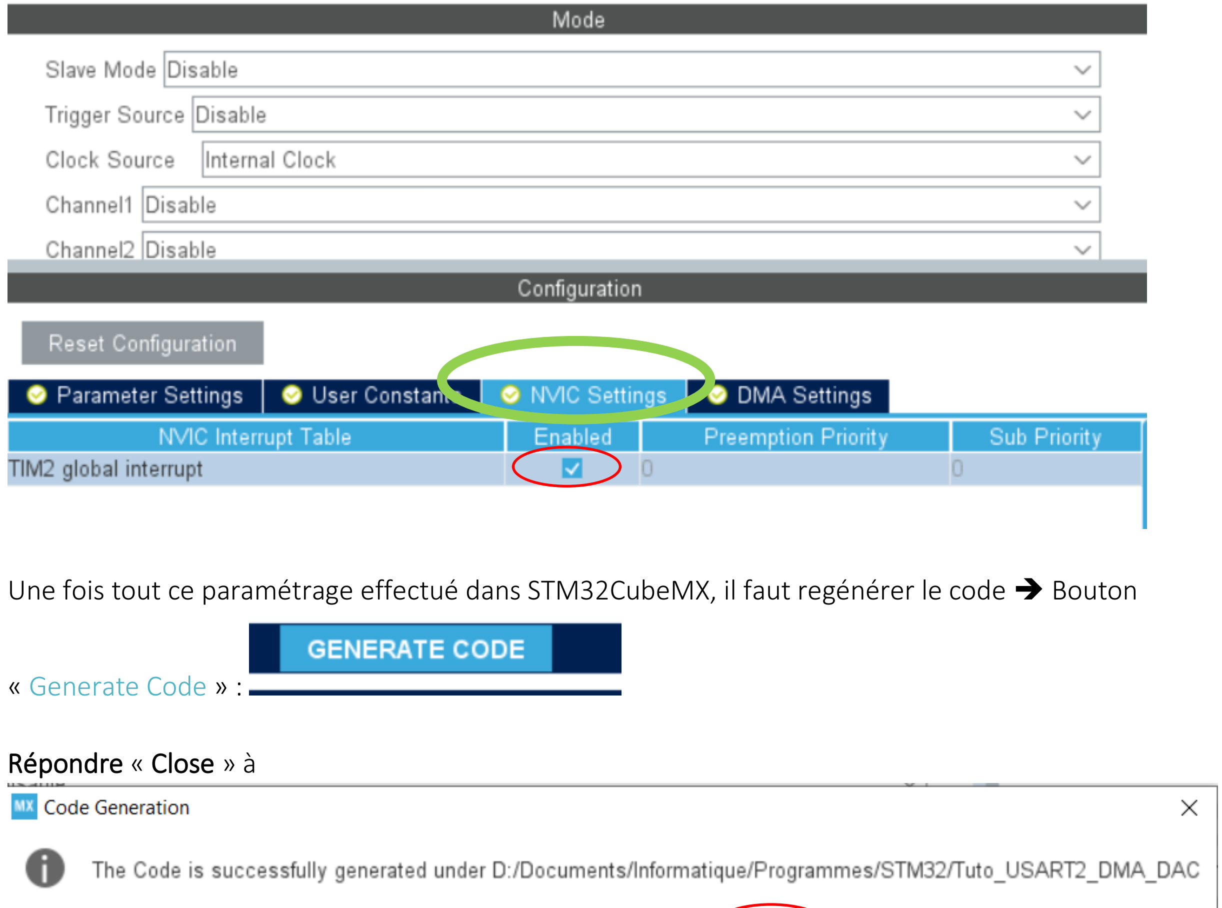
1- Clock Source ➔ Internal Clock ;
2- Prescaler (PSC – 16 bits value) ➔ 80-1 (Pour diviser la master clock de 80 Mhz par 80 ➔ 1 MHz) ;
3- Counter Period (AutoReload Register – 32 bits) 100-1 (*on répète le comptage 99 fois pour avoir 100 séquences de comptage avant un « Update Event » et donc on obtient un « Update Event » ayant une fréquence de 1MHz/100* ➔ *10 kHz. Si l'on dessine notre forme d'onde avec 100 échantillons, ça nous fera une fréquence de 10 kHz/100 = 100 Hz pour notre onde*.) ;
4- Trigger Event Selection TRGI ➔ Update Event

Notons qu'il est possible de faire plus simple dans l'écran ci-dessus, nous aurions pu laisser le paramètre *Prescaler* à **0** et n'utiliser que le paramètre *AutoReload Register* en le positionnant à **7999**, ce qui aurait donné le même résultat. J'ai préféré utiliser les deux paramètres pour montrer les différentes possibilités d'action. La documentation présente sur le site de FT (en.STM32L4_WDG_TIMERS_GPTIM.pdf) donne la formule suivante pour trouver la fréquence PWM :

PWM frequency set-up

- Defined with auto-reload (ARR, in TIMx_ARR) and clock prescaler (PSC, in TIMx_PSC):
- $f_{PWM} = \frac{f_{TIM}}{(ARR+1)\times(PSC+1)}$
- Practically, one must start with PSC = 0 (no prescaler):
- $ARR = \frac{f_{TIM}}{f_{PWM}\times(PSC+1)} - 1 \rightarrow ARR = \frac{f_{TIM}}{f_{PWM}} - 1$
- If it yields a value above the 16-bit (or 32-bit) range, PSC must be increased until ARR fits:
- $ARR = \frac{f_{TIM}/2}{f_{PWM}} - 1 \rightarrow ARR = \frac{f_{TIM}/3}{f_{PWM}} - 1 \rightarrow ARR = \frac{f_{TIM}/4}{f_{PWM}} - 1 \rightarrow \ldots$

Il faut maintenant aller dans l’onglet « NVIC Settings » et cocher « Enabled » en face de « TIM2 blobal interrupt » :

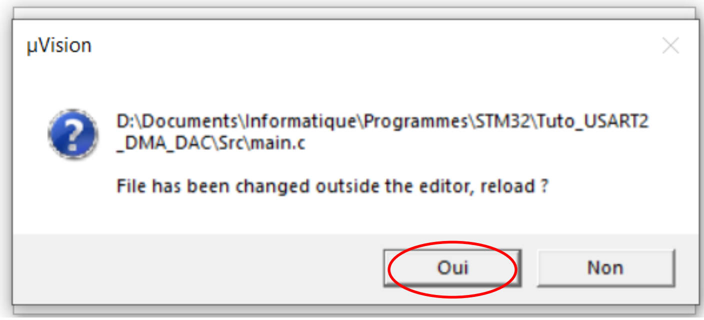


Une fois tout ce paramétrage effectué dans STM32CubeMX, il faut regénérer le code → Bouton « Generate Code » :

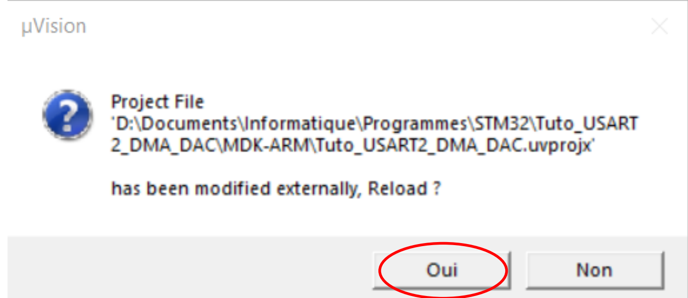
Répondre « Close » à

Sauver le projet par File/Save Project/.

Et **retourner dans Keil µVision** pour accepter la mise à jour des fichiers source.



Repondre « Oui »



Répondre « Oui »

Appuyer sur « F7 » = Project/ Build target

Vous devez toujours avoir le même « warning » que précédemment et toujours zéro erreur :

```
Tuto_USART2_DMA_DAC\Tuto_USART2_DMA_DAC.axf: Warning: L6989W: Could not apply pat
Program Size: Code=9536 RO-data=496 RW-data=16 ZI-data=1312
Finished: 0 information, 1 warning and 0 error messages.
FromELF: creating hex file...
"Tuto_USART2_DMA_DAC\Tuto_USART2_DMA_DAC.axf" - 0 Error(s), 1 Warning(s).
Build Time Elapsed:  00:01:05
```

II-2-d) Retour Keil µVision pour la programmation finale

Si rien n'a été oublié dans les étapes précédentes, STM32CubeMx devrait avoir généré ceci automatiquement :

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART2_UART_Init();
  MX_DAC1_Init();
  MX_TIM2_Init();
```

Il nous reste à :

- **Remplir** la table **sine_val** avec les formes d'ondes ;
- **Activer** la réception des interruptions de l'**USART2** ;
- **Activer** le **timer2** ;
- **Démarrer** le **DMA** associé au **DAC1** ;
- **Afficher** un message de choix **sur le terminal série** ;

La partie de code juste avant la boucle infinie on voit l'utilisation du **printf** :

```
/* USER CODE BEGIN 2 */
get_sineval(); // Onde sinusoïdale avec 100 échantillons (n initialisé à 100)
HAL_UART_Receive_IT(&huart2, BufUSART2, 1); // Activation du mécanisme d'interruption pour la réception USART2
HAL_TIM_Base_Start(&htim2) ; // Démarrage du timer 2
HAL_DAC_Start_DMA(&hdac1,DAC1_CHANNEL_1,sine_val,100,DAC_ALIGN_12B_R); // Démarrage du DAC1, on passe en paramètre
                                                                       // le tableau et le nombre d'échantillons
printf("\nEntrez un des choix ci-dessous :\n");
printf("\t A \t\t\t-->  25 échantillons = 400,0 Hz\n");
printf("\t B \t\t\t-->  50 échantillons = 200,0 Hz\n");
printf("\t C \t\t\t-->  75 échantillons = 133,3 Hz\n");
printf("\t <> A et <> B et <> C \t--> 100 échantillons = 100 Hz\n");
printf("\nVotre choix ? > ");
/* USER CODE END 2 */
```

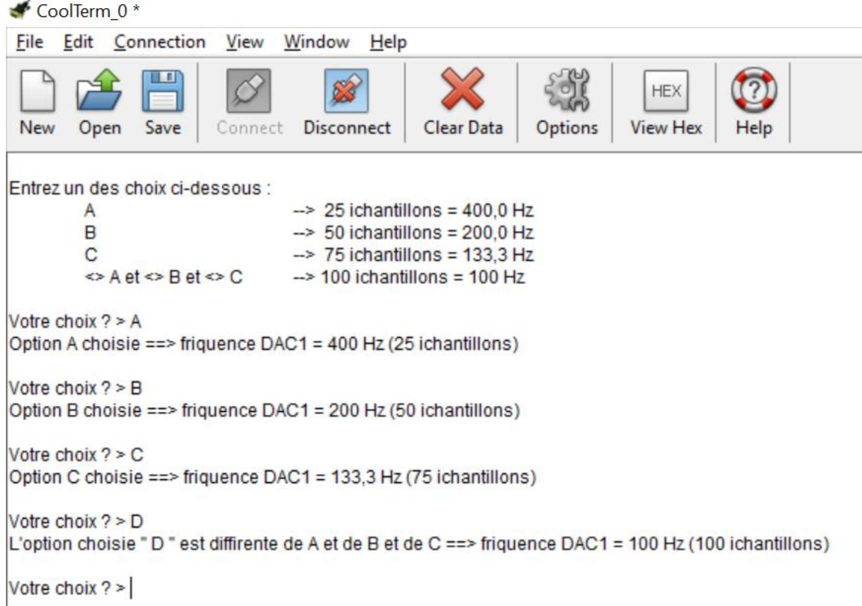
- **Ecrire** le contenu de la boucle « while (1) » qui sera chargé de **prendre en compte le choix de l'utilisateur** ;

Et enfin la boucle principale qui récupère le choix utilisateur :

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  if (BufReady)
  {
    sprintf(buffer, "%s", BufRecept);
    BufReady=0;

    //ATTENTION, pour que ce programme fonctionne, il faut que votre terminal série
    //ajoute uniquement "0x0A" après que la touche return ait été appuyée
    //Par exemple dans les options du terminal "Coolterm", il faut régler "Enter Key" Emulation sur LF
    //Si votre terminal ajoute 0X0D 0x0A, il faudra changer la condition du test "strlen(buffer)==2" etc

    HAL_DAC_Stop_DMA(&hdac1,DAC1_CHANNEL_1); // Arrêt du DAC1
    if(strlen(buffer)==1)
    {
      switch (buffer[0])
      {
        case 'A': //Option A
          printf("Option A choisie ==> fréquence DAC1 = 400 Hz (25 échantillons)\r\n");
          n = 25 ;
          break;
        case 'B': //Option B
          printf("Option B choisie ==> fréquence DAC1 = 200 Hz (50 échantillons)\r\n");
          n = 50 ;
          break;
        case 'C': //Option C
          printf("Option C choisie ==> fréquence DAC1 = 133,3 Hz (75 échantillons)\r\n");
          n = 75 ;
          break;
        default:
          // instructions à exécuter si expression n'est égale à aucune des valeurs des case
          printf("L'option choisie \" %s \" est différente de A et de B et de C ==> fréquence DAC1 = 100 Hz (100 échantillons)\r\n", buffer);
          n = 100 ;
          break;
      }
    }
    else
    {
      printf("L'option choisie \" %s \" est différente de A et de B et de C ==> fréquence DAC1 = 100 Hz (100 échantillons)\r\n", buffer);
      n = 100 ;
    }
    printf("\nVotre choix ? > "); // On réaffiche la demande pour un nouveau choix
    get_sineval(); // On reconstruit la forme d'onde avec le nombre d'échantillons choisi
    HAL_DAC_Start_DMA(&hdac1,DAC1_CHANNEL_1,sine_val,n,DAC_ALIGN_12B_R); // On redémarre le DAC1
  }
}
/* USER CODE END 3 */
```

II-3) Ecrans de résultats

1) CoolTerm :



2) Oscilloscope (respectivement A, B, C, D) :