# Table des matières

I) UTILISATION DE PRINTF ET SCANF MODE DEBUG DANS KEIL µVISION	2
I-1) Configuration preablable dans l'IDE Keil $\mu V$ is ion	2
I-1-A) Menu Manage Run-Time Environment (icone 🗇 )	2
I-1-в) 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-З-В) 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
<b>II-2-A)</b> DEMARRAGE DANS STM32CUBEMX	9
II-2-A) DEMARRAGE DANS STM32CUBEMX II-2-B) DANS L'IDE KEIL µVISION	9 2
II-2-A) DEMARRAGE DANS STM32CUBEMX II-2-B) DANS L'IDE KEIL µVISION	9 2 4
II-2-A) DEMARRAGE DANS STM32CUBEMX	9 2 4

I) Utilisation de printf et scanf mode debug dans Keil  $\mu$ Vision

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 :

oftware Component	Sel.	Variant		Version	Description
🗉 💠 Board Support		NUCLEO-L476RG	$\sim$	1.0.0	STMicroelectronics NUCLEO-L476RG Development Board
🛛 💠 CMSIS					Cortex Microcontroller Software Interface Components
🛶 CMSIS Driver					Unified Device Drivers compliant to CMSIS-Driver Specifications
🕬 💠 Compiler		ARM Compiler		1.6.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Event Recorder		DAP		1.4.0	Event Recording and Component Viewer via Debug Access Port (DA
					Retarget Input/Output
VTT		Breakpoint	$\sim$	1.2.0	Stop program execution at a breakpoint when using TTY
STDOUT	<b>~</b>	ITM	$\sim$	1.2.0	Redirect STDOUT to a debug output window using ITM
STDIN	<b>~</b>	ITM	$\sim$	1.2.0	Retrieve STDIN from a debug output window using ITM
STDERR		Breakpoint	$\sim$	1.2.0	Stop program execution at a breakpoint when using STDERR
🖉 File		File System		1.2.0	Use retargeting together with the File System component
🛶 Device					Startup, System Setup
🗠 💠 File System		MDK-Plus	$\sim$	6.13.0	File Access on various storage devices
🕬 Graphics		MDK-Plus	$\sim$	5.50.0	User Interface on graphical LCD displays
🗠 💠 Network		MDK-Plus	$\sim$	7.12.0	IPv4 Networking using Ethernet or Serial protocols
a 🚸 USB		MDK-Plus	$\sim$	6.13.7	USB Communication with various device classes

# 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).

Cortex-M Target Driver Setup			×
Debug Trace Rash Download			
Core Clock B0.00000 MHz Trace Port Serial Wire Output - UART/NRZ SWO Clock Prescaler: 40 F Autodetect SWO Clock: 2.000000 MHz ITM Stimulus Ports Enable: 0xFFFFFFFF Privilege: 0x0000008 Port	Image: stamps       Prescaler:       1       Image: stamps         Image: stamps       Prescaler:       1       Image: stamps         PC Sampling       Prescaler:       1024*16 Image: stamps         PC Sampling       Prescaler:       1024*16 Image: stamps         Image: stamps       Prescaler:       1024*16 Image: stamps         Image: stamps       Periodic       Periodic: <disabled>         Image: stamps       Image: stamps       Image: stamps       <disabled>         Image: stamps       Image: stamps       Image: stamp       <disabled>         Ima</disabled></disabled></disabled></disabled></disabled></disabled></disabled></disabled></disabled></disabled></disabled>	Trace Events         CPI: Cycles per Instruction         EXC: Exception overhead         SLEEP: Sleep Cycles         LSU: Load Store Unit Cycles         FOLD: Folded Instructions         EXCTRC: Exception Tracing         Port       8         Port       7         Port       8         Port       7         Port       8         Port       7         Port       8         Port       8         Port       9         Port       8         Port       7         Port       7         Port       8         Port       7         Port       7         Port       7         Port       8         Port       9         Port       9	
Advanced settings	YNC		
		OK Annuler Appliqu	ler

> 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 */
finclude "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 */
}
```

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 ».

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

UART #2
UART #3
Debug (printf) Viewer
, ,

#### I-3-b) Menu « View /Watch 1 »

Afficher la fenêtre Watch1.

Watch Windows	•	网	Watch 1
Memory Windows	•	<b>2</b> 2	Watch 2

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 :

Natch 1			×	
Name	Value	Туре		
🔷 n	2020	int		
<enter expression=""></enter>				
Debug (printf) Viewer			×	
Avant d'entrer dans la boucle while(l)				
Saisissez une valeu:	c pour n :			

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

Watch 1						
Name	Value	Туре				
1 n	12	int				
<enter expression=""></enter>						
Debug (printf) Viewer						
Avant d'entrer dans la b	oucle while(1)					
Saisissez une valeur pou	r n : Valeur de n	: 14				
Saisissez une valeur pou	r n 🕴 Valeur de n	: 12				
Saisissez une valeur pou	r n					

## 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, <u>l'USART2 entre le STM32L476RG et le ST-LINK MCU est activé par défaut</u> 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.

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 <u>non pas</u> modifiée dans « stm32l4xx\_hal\_uart.c ».

# C'est ce que nous allons faire dans le fichier main. <u>C'est cette fonction de callback qui sera</u> 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 :
  - $\circ$  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 ;
- o Fin si
- o 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* ;
    - o 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)





7) Par défaut l'arbre d'horloge est bien réglé sur 80 Mhz, il n'est pas nécessaire d'y toucher.

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

Pinout	& Configuration	Clock Co	nfiguration	
		Additiona	l Software	✓ Pinout
Q ~	Ø	USART2 Mode a	and Configuration	I
Categories A->Z		Ma	de	
Connectivity		Mode Asynchronous Hardware Flow Control (R:232) Disable	>	× ×
CAN1 12C1 12C2		☐ Hardware Flow Control (RS485)		
IRTIM		Config	uration	
LPUART1 QUADSPI		Reset Configuration		
SDMMC1		NVIC Settings     O     DMA	Settings	GPIO Settings
SPI2 SPI3 SWPMI1 UART4 UART5 USART1 VSART2		USART2 global interrupt	Preempt	User Constants ion Priority Sub Priority

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



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

<		Project Settings Project Name Tuto_USART2_DMA_DAC Preject Location D:\Documents\Informatique\Programmes\STM32\ Application Structure
	Code Generator	Basic   Do not generate the ma  Toolchain Folder Location  D:\Documents\Informatique\Programmes\STM32\Tuto_USART2_DMA_DAC\  Toolchain / IDE Min Version  MDK-ARM  V5.27  Generate Under
	Advanced Settings	Linker Settings Minimum Heap Size 0x200 Minimum Stack Size 0x400
		Mcu and Firmware Package         Mcu Reference         STM32L476RGTx         Firmware Package Name and Version         STM32Cube FW_L4 ∨1.15.1         ✓ Use latest available version         ✓ Use Default Firmware Location         runo/STM32Cube/Repository/STM32Cube_FW_L4_V1.15.1         Browse

11) Dans la partie « Code Generator », confi	gurer comme ceci
--	------------------

	Project	STM32Cube MCU packages and embedded software packs © Copy all used libraries into the project folder © Copy only the necessary library files © Add necessary library files as reference in the toolchain project configu	
<	Code Generator	Generate peripheral initialization as a pair of '.c/.h' files per peripheral Backup previously generated files when re-generating Keep User Code when re-generating Delete previously generated files when not re-generated	
	Advanced Settings	Set all free pins as analog (to optimize the power consumption)         Enable Full Assert         Template Settings         Select a template to generate customized code         Settings	
12) 13)	Générer le code : Cliquer sur « Open P Code Generation	GENERATE CODE	×
	The Code is suc	cessfully generated under D:/Documents/Informatique/Programmes/STM32/Tuto_USART2_	UMA_DAC

:

# II-2-b) Dans l'IDE Keil µVision

Appuyer sur « F7 » = Project/ Build target

Vous ne devez obtenir aucune erreur :

Build Output	
compiling stm3214xx_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 */
finclude <stdio.h> // USART2 : printf, sprintf, putc
finclude <string.h> // USART2 : strlen
finclude "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 : l si une ligne terminée pa LF (0xA) a été reçue, 0 sinon
uint6_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
```

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
E (
   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 recue 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
   }
L,
  roid 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);</pre>
L,
/* 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 :



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 :

Q ~	0	DAC1	Mode and Configurati	ion
Categories A->Z			Mode	
System Core	>	OUT1 mode Connected to external	l pin only	~
Analog	~	OUT2 mode Disable		~
ADC1 ADC2 ADC3		🗆 Externar migger	Configuration	
COMP1 COMP2		Reset Configuration		
Ø OPAMP1		NVIC Settings	DMA Settings	OFIO Settings
OPAMP2		Parameter Settings		User Constants
	2	Configure the below parameters :		
Timers	>	Q Search (Crt1+F) () ()		0
Connectivity	>	Output Buffer	Enable	
Multimedia	>	Trigger Wave generation mode	Timer 2 Trigg Disabled	er Out event
Security	>	User Trimming Sample And Hold	Factory trimr Sampleandho	ning old Disable
Computing	>			

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

#### Dans l'onglet DMA Settings, configurer comme ceci :

	DAC1 Mode and Configuration					
			Mo	de		
	OUT1 mode Connecte	d to external pin only				$\sim$
	OUT2 mode Disable					~
	🗆 External Trinner					
			Config	uration		
	Reset Configuration					
	🥺 Parameter Settings	🥝 User Constants	📀 NVIC	Settings	🕑 DMA Settin	gs 🎝 GPIO Settings
2	DMA Request	Channel		Di	rection	Priority
<	DAC_CH1	DMA1 Channel 3		Memory To	Peripheral	Low

Add Delete			
DMA Request Settings			
		Peripheral	Memory
3 Mode Circular	Increment Address		
	Data Width	4 Word ~	Word ~

- 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 :

Q ~	٢	TIM2 Mode and Configuration	
Categories A->Z		Mode	
System Core	>	Slave Mode Disable	$\sim$
Analog	>	Trigger Source Disable 1	$\sim$
		Clock Source Internal Clock	~
Timers	~	Channel1 Disable	~
\$		Channel2 Disable	$\sim$
LPTIM1		Configuration	
LPTIM2		Deset Configuration	
RTC		Reset Conliguration	
		📀 Parameter Settings 🛛 📀 User Constants 🛛 😔 NVIC Settings 🛛 😒 DMA Settings	
TIM3		Configure the below parameters :	
TIM4		Q Search (CrtI+F) ③ ③	0
TIM6		V Counter Settings 2	
TIM7		Prescaler (PSC - 16 bits value)	
TIM8		Counter Mode Up 3	
TIM15		Counter Period (AutoReload Register - 32 bit. 100-1	
11M16 TIM17		Internal Clock Division (CKD) No Division	
1114117		auto-reload Disable	
		✓ Trigger Output (TRGO) Parameters	
Connectivity	>	Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)	
		Trigger Event Selection TRGO	

- 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:
- $ARR = \frac{f_{TIM}/2}{f_{PWM}} 1 \rightarrow ARR = \frac{f_{TIM}/3}{f_{PWM}} 1 \rightarrow ARR = \frac{f_{TIM}/4}{f_{PWM}} 1 \rightarrow \dots$

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

Mada	
Mode	-
Slave Mode Disable	$\sim$
Trigger Source Disable	$\sim$
Clock Source Internal Clock	$\sim$
Channel1 Disable	$\sim$
Channel2 Disable	~
Configuration	
Reset Configuration	
📀 Parameter Settings 🛛 📀 User Constant 💭 📀 NVIC Settings 🌛 DMA Set	ttings
NVIC Interrupt Table Enabled Preemption	Priority Sub Priority
TIM2 global interrupt 🛛 🗸 🔽 🔾 0	0

Une fois tout ce paramétrage effectué dans STM32CubeMX, il faut regénérer le code → Bouton

GENERATE CODE	
« Generate Code » :	
Répondre « Close » à	
Code Generation	×
The Code is successfully generated under D:/Documents/Informatique/Programmes/STM32/Tuto_USART2_D	MA_DAC
Open Folder Open Project Close	

Sauver le projet par File/Save Project/.

Et retourner dans Keil  $\mu$ Vision pour accepter la mise à jour des fichiers source.

μVision	×	<
0	D:\Documents\Informatique\Programmes\STM32\Tuto_USART2 _DMA_DAC\Src\main.c File has been changed outside the editor. reload ?	
	Oui Non	

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 :



choix de l'utilisateur ;

Et enfin la boucle principale qui récupère le choix utilisateur :

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
ſ
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
   if (BufReady)
   ł
     sprintf(buffer, "%s", BufRecept);
     BufReady=0;
     //AITENTION, pour que ce programme fonctionne, il faut que votre terminal sèrie
//ajoute uniquement "OxOA" 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 OXOD OxOA, il faudra changer la condition du test "strlen(buffer)==2" etc
     HAL_DAC_Stop_DMA(shdac1, 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 DACl = 100 Hz (100 échantillons)\r\n", buffer);
              n = 100 ;
             break:
        }
     else
      ł
        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(shdac1,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 :

✓ CoolTerm_0 *				
<u>File Edit Connection View</u>	<u>W</u> indow <u>H</u> elp			
New Open Save	t Disconnect Clear Data			
Entrez un des choix ci-dessous :				
A	> 25 ichantillons = 400,0 Hz			
B	> 50 ichantillons = 200,0 Hz			
С	> 75 ichantillons = 133,3 Hz			
<> A et <> B et <> C	> 100 ichantillons = 100 Hz			
Votre choix ? > A				
Option A choisie ==> friquence D/	AC1 = 400 Hz (25 ichantillons)			
Votre choix 2 > B				
Option B choisie ==> friquence D	AC1 = 200 Hz (50 ichantillons)			
Votre choix 2 > C				
Option C choisie ==> friquence D	AC1 = 133,3 Hz (75 ichantillons)			
L'option choisie " D " est diffirente	de A et de B et de C ==> friquence DAC1 = 100 Hz (100 ichantillons)			
Votre choix ? >				

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

