



## Compte Rendu

Ecole Nationale Supérieure de l'Electronique et de ses Applications

Microprocesseur

TP4 : Plotting signals

2<sup>eme</sup> Année

Année : 2023 - 2024

Camille Lanfredi

Rémi Weidle

## 1. Préparation

Lors des séances des travaux pratiques nous utilisons plusieurs documents pour étudier dans les moindres fonctionnalités de notre carte Nucléo 144.

A partir du manuel d'utilisation, nous pouvons trouver toutes les caractéristiques intéressantes pour nos travaux pratiques.

Ref : « [https://www.st.com/resource/en/user\\_manual/um1974-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/um1974-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf) »

### 6.1 Nucleo-144 board layout

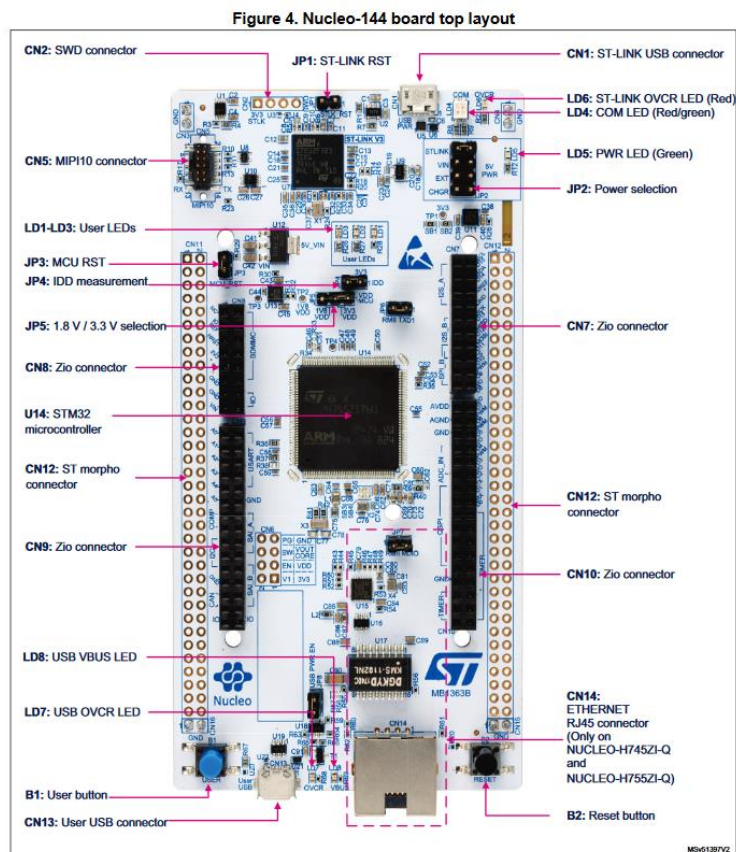


Illustration de la carte Nucléo détaillée

Le microcontrôleur STM32 sur la carte Nucleo-144 dispose de **144 broches au total**. Ces broches comprennent des broches GPIO, des broches d'alimentation, des broches de masse et diverses autres broches de communication et de contrôle.

Les connecteurs ST Morpho sur la carte Nucleo-144 se trouvent sur les côtés de la carte. Ils sont conçus pour faciliter la connexion à des cartes d'extension ou des shields.

Les connecteurs ST Morpho ne sont pas utilisés par défaut sur les cartes Nucleo car ils ne sont pas aussi couramment utilisés que certaines autres interfaces standard telles que les connecteurs Arduino, qui sont plus populaires auprès de la communauté des fabricants et des développeurs.

Ref (page 68) : « [https://www.st.com/resource/en/user\\_manual/um1974-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/um1974-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf) »

## 6.15 ST morpho connector

The ST morpho connector consists in male pin header footprints CN11 and CN12 (OFF by default). They are used to connect the STM32 Nucleo-144 board to an extension board or a prototype/wrapping board placed on top of the STM32 Nucleo-144 board. All signals and power pins of the STM32 are available on the ST morpho connector. This connector can also be probed by an oscilloscope, logical analyzer or voltmeter.

*Table 21* and *Table 22* show the pin assignments of each STM32 on the ST morpho connector.

Les connecteurs Zio sur la carte Nucleo-144 se trouvent généralement du côté opposé des connecteurs ST Morpho (voir page 2 : *Illustration de la carte Nucléo détaillée*). Ces connecteurs offrent un accès à diverses broches d'E/S et fonctionnalités, permettant une plus grande flexibilité pour la connexion de dispositifs externes.

## 6.14 ST Zio connectors

CN7, CN8, CN9 and CN10 are female on top side and male on bottom side connectors. They include support for ARDUINO® Uno V3. Most shields designed for ARDUINO® Uno V3 can fit to the STM32 Nucleo-144 board.

To cope with ARDUINO® Uno V3, apply the following modifications:

- SB138 and SB143 must be ON
- SB140/147/150/157/167/171 must be OFF to connect I<sup>2</sup>C on A4 (pin 5) and A5 (pin 6 of CN9).

**Caution:1** The I/Os of STM32 microcontroller are 3.3 V compatible instead of 5 V for ARDUINO® Uno V3.

**Caution:2** SB12 must be removed before implementing ARDUINO® shield with V<sub>REF+</sub> power being provided on CN7 pin 6. Refer to *Table 12: Solder bridges* for details on SB12.

*Table 13* to *Table 20* show the pin assignment for each STM32 microcontroller on the ST Zio connectors.

Ref (page 36) : « [https://www.st.com/resource/en/user\\_manual/um1974-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/um1974-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf) »

Le signal de port COM virtuel (VCP) est généralement connecté à l'une des interfaces USART (Universal Synchronous Asynchronous Receiver Transmitter) sur le microcontrôleur.

## 6.9 USART communication

The USART3 interface available on PD8 and PD9 of the STM32 can be connected either to ST-LINK or to ST morpho connector. The choice is changed by setting the related solder bridges. By default the USART3 communication is enabled between the target STM32 and both the ST-LINK and the ST morpho connector.

Table 9. USART3 pins

Pin name	Function	Virtual COM port	ST morpho connection
PD8	USART3 TX	SB5 ON and SB7 OFF	SB5 OFF and SB7 ON
PD9	USART3 RX	SB6 ON and SB4 OFF	SB6 OFF and SB4 ON

Ref (page 26) : « [https://www.st.com/resource/en/user\\_manual/um1974-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/um1974-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf) »

## 4. Virtual Com Port and Clocks

Dans un premier temps, nous réalisons la première connexion entre la carte Nucleo et notre ordinateur. Le but est d'envoyer un message depuis la carte Nucleo vers l'ordinateur en utilisant le module USART.

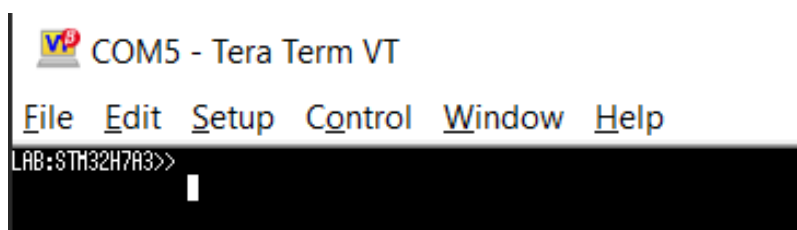
### W1. Etablir la première connexion

Lorsque l'on examine le code avec IOC, USART 3 est actuellement activé, c'est pourquoi nous utilisons "&huart3" pour cela. Le débit en bauds est de 115 200, avec une configuration de 8 bits, aucune parité et 1 bit d'arrêt. Nous configurons TeraTerm en conséquence pour visualiser la communication avec la carte et afficher le message.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART3_UART_Init();
MX_USB_OTG_HS_USB_Init();

/* USER CODE BEGIN 2 */
sprintf(UART_buf, "LAB:STM32H7A3>> \n");
HAL_UART_Transmit(&huart3, (unsigned char *)UART_buf, strlen(UART_buf), 1);
```

Avec TeraTerm sur le Com5 de mon ordinateur, nous pouvons bien lire le message transmis.



Résultat après exécution du programme

## W2. Envoyer des messages plus complexes

Après avoir ajouté le fichier source dans le répertoire "Core/Src" et le fichier d'en-tête (\*.h) dans "Core/Inc", nous pouvons configurer la communication série et ainsi envoyer des messages plus complexes. Il est essentiel de s'assurer d'inclure les fichiers appropriés pour éviter les erreurs liées à des fonctions non définies ou mal définies.

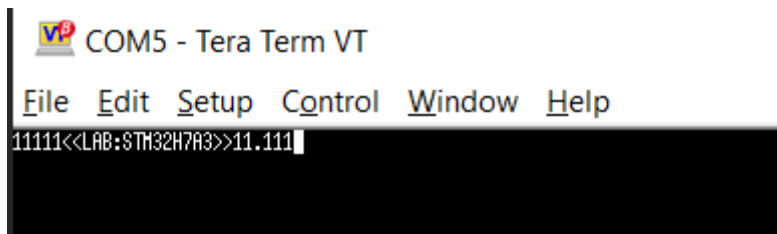
```
SERIAL_Init(&huart3);

SERIAL_SendInt(11111);

SERIAL_SendCharBuf("<<LAB:STM32H7A3>>");

SERIAL_SendFloat(11.1111);
```

*Utilisation des fonctions pour communiquer quelques types de variables*



*Résultat après exécution du programme*

On remarque bien que chaque valeur a été parfaitement récupérée dans TeraTerm et qu'il n'y a eu aucun souci en compilation/exécution peu importe le type de donnée ici int, char et float.

## W3. Trouver le tout début du code – Première instruction

Maintenant que la communication est opérationnelle, nous allons étudier les flux de données en interne. Pour ce faire, nous utiliserons l'outil de débogage. En utilisant la fenêtre **SFR** (Special Function Register), qui contient les registres matériels du microprocesseur, nous pouvons observer l'emplacement de la table des vecteurs d'interruption **VTOR** dans la mémoire.

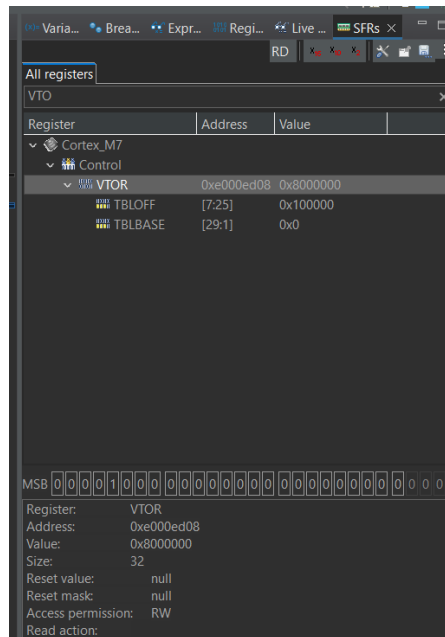


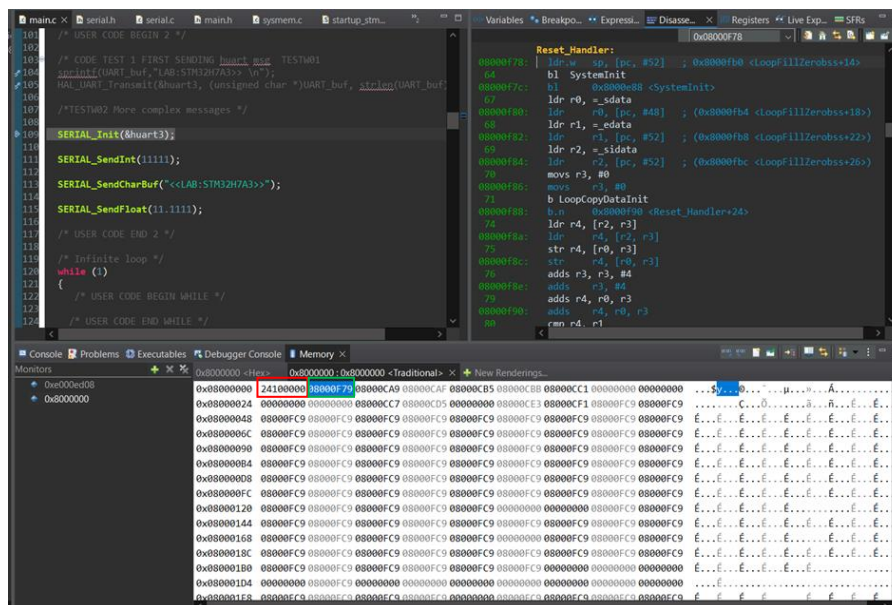
Table des vecteurs d'interruption

La valeur de SCB\_VTOR contient l'adresse du début de la table des vecteurs.

Cependant et d'après la documentation, l'adresse de la première instruction se trouve à SCB\_VTOR+4. Pour l'obtenir nous devons aller dans "Fenêtre" > "Afficher la vue" > "Mémoire" et mettre la valeur en "Hexadécimal" (Hex Integer).

En outre, nous devons soustraire le bit de poids faible la valeur de l'adresse de la première instruction de la valeur à SCB\_VTOR+4.

Nous allons donc ouvrir l'onglet "Fenêtre" > "Afficher la vue" > "Désassemblage".



Cela nous permet d'obtenir :

- Adresse de la pile : 24100000
- Première adresse d'instruction : 08000F78



## W5. Horloge après la réinitialisation

Dans un premier temps, nous allons étudier la commande de l'horloge.

Ref (page 165) : <https://moodle.ensea.fr/mod/resource/view.php?id=1670>

### 8.7.5 RCC clock configuration register (RCC\_CFGR)

Address offset: 0x010

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2SEL[2:0]			MCO2PRE[3:0]				MCO1SEL[2:0]			MCO1PRE[3:0]				Res.	Res.
rw	rw	rw	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
TIMPRE	Res.	RTCPRE[5:0]						STOPKERWUCK	STOPWUCK	SWS[2:0]			SW[2:0]		
rw		rw	rw	rw	rw	rw	rw	rw	rw	r	r	r	rw	rw	rw

**RCC** (Reset and Clock Control) est le registre de contrôle de l'horloge dans un système embarqué. Le champ **SW** (Switch) dans le registre **RCC\_CFGR** est utilisé pour sélectionner la source de l'horloge système.

Ce champ détermine quelle source d'horloge est utilisée comme horloge maître du système.

Désormais, nous nous plaçons avant l'exécution de n'importe quel code et on regarde en 'debug mode' avec **SFR** le registre **RCC\_CDFGR1** et **RCC\_CFGR**.

All registers			
RCC_CDFGR1			
Register	Address	Value	
STM32H7A3			
RCC			
RCC_CDFGR1	0x58024418	0x0	
HPRE	[0:4]	0x0	
CDPPRE	[4:3]	0x0	
CDCPRE	[8:4]	0x0	

Registre RCC\_CDFGR1 avant exécution de code

All registers			
RCC_CFGR			
Register	Address	Value	
STM32H7A3			
RCC			
RCC_CFGR	0x58024410	0x0	
SW	[0:3]	0x0	
SWS	[3:3]	0x0	
STOPWUCK	[6:1]	0x0	
STOPKERWUCI	[7:1]	0x0	
RTCPRE	[8:6]	0x0	
TIMPRE	[15:1]	0x0	
MCO1PRE	[18:4]	0x0	
MCO1SEL	[22:3]	0x0	
MCO2PRE	[25:4]	0x0	
MCO2SEL	[29:3]	0x0	

Registre RCC\_CFGR avant exécution de code

Il est important de noter que tous les paramètres sont initialement à 0 avant l'exécution de tout code. Par conséquent, il est nécessaire de consulter la documentation pour déterminer sur quelle horloge le système est actuellement calibré.

Ainsi d'après la documentation (page 394),

Bits 2:0 **SW[2:0]**: system clock and trace clock switch

Set and reset by software to select system clock and trace clock sources (**sys\_ck** and **traceportck**).

Set by hardware in order to:

- force the selection of the HSI or CSI (depending on STOPWUCK selection) when leaving a system Stop mode
- force the selection of the HSI in case of failure of the HSE when used directly or indirectly as system clock

000: HSI selected as system clock (**hsi\_ck**) (default after reset)

001: CSI selected as system clock (**csi\_ck**)

010: HSE selected as system clock (**hse\_ck**)

011: PLL1 selected as system clock (**pll1\_p\_ck** for **sys\_ck**, **pll1\_r\_ck** for **traceportck**)

others: reserved

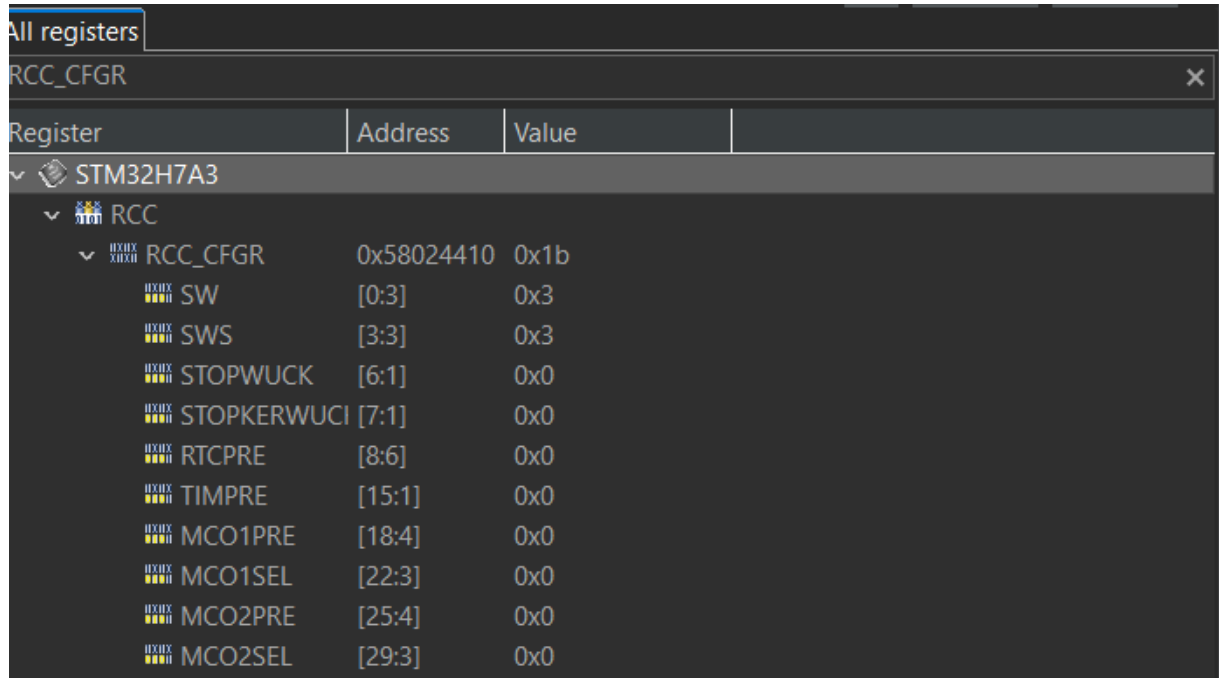
Choix de l'horloge en fonction de SW [2 ; 0]

A 0.0.0 le système est calibré sur l'horloge interne HSI, qui a une fréquence de 64 MHz dans ce cas. Cette configuration n'est pas surprenante, car au démarrage initial, le système peut "essayer" l'horloge interne HSI pour s'assurer qu'une horloge stable est disponible en cas de problème ou de réinitialisation. C'est pourquoi vous voyez (default **after reset**) à côté de 0.0.0, ce qui indique la configuration par défaut après une réinitialisation.

W6. Horloges après l'appel à SystemClock\_Config()



Cette fois-ci, nous avons décidé de faire progresser le mode de débogage en configurant le code pour utiliser une configuration d'horloge spécifique. Les valeurs dans les registres, telles que RCC\_CFGR, changent en conséquence pour refléter cette nouvelle configuration.



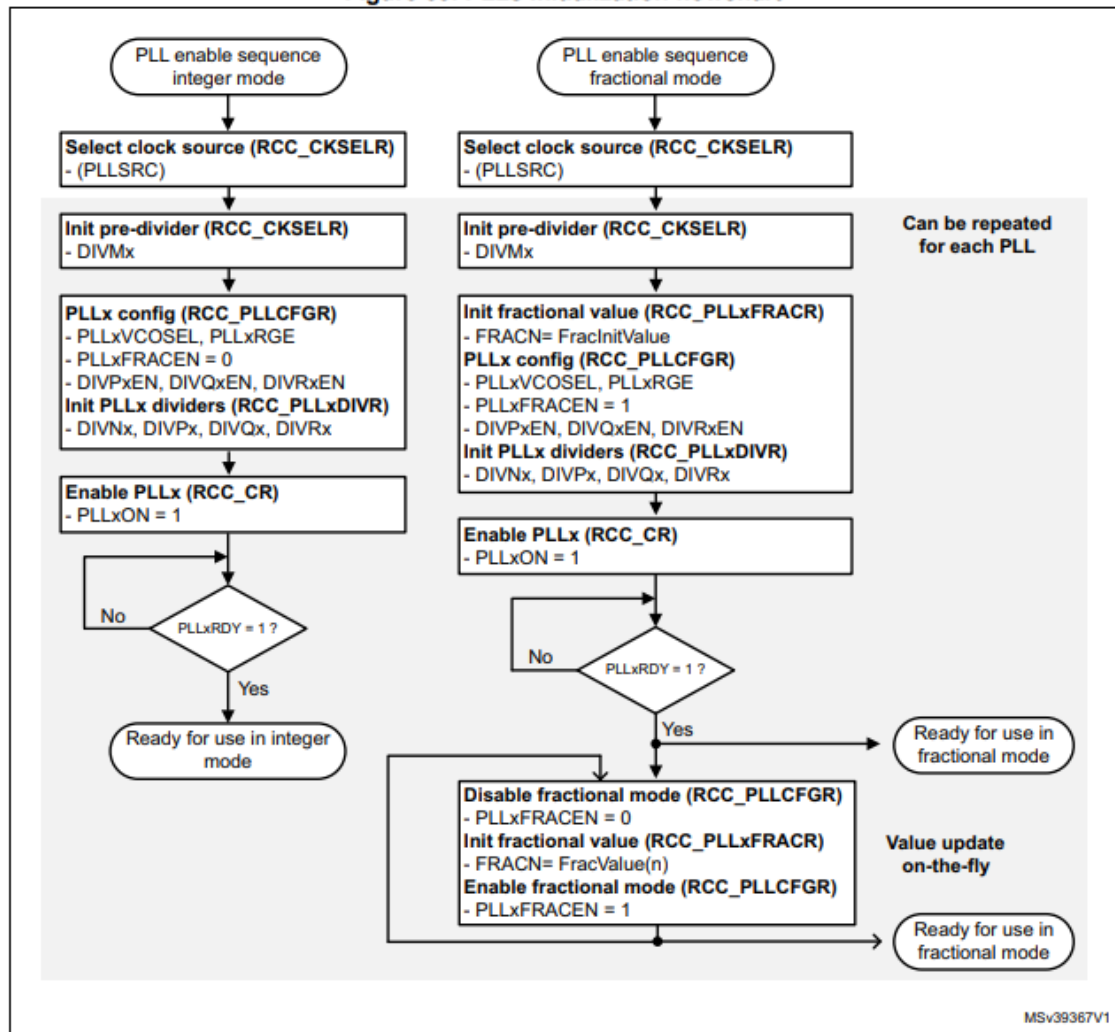
The screenshot shows the 'All registers' window in STM32CubeIDE. The 'RCC\_CFGR' register is selected. The table below shows the configuration of the RCC register and its fields.

Register	Address	Value
STM32H7A3		
RCC		
RCC_CFGR	0x58024410	0x1b
SW	[0:3]	0x3
SWS	[3:3]	0x3
STOPWUCK	[6:1]	0x0
STOPKERWUCI	[7:1]	0x0
RTCPRE	[8:6]	0x0
TIMPRE	[15:1]	0x0
MCO1PRE	[18:4]	0x0
MCO1SEL	[22:3]	0x0
MCO2PRE	[25:4]	0x0
MCO2SEL	[29:3]	0x0

Registre RCC\_CFGR après système\_Clock\_Config

Il est clair que dans ce cas, le champ SW passe de 0.0.0 à 0.1.1. Selon les informations précédemment exposées, le MUX a modifié la source d'horloge à laquelle le système est calibré pour utiliser la PLL (Phase-Locked Loop). Ainsi on va chercher à déterminer la fréquence de la PLL afin de déterminer la fréquence de fonctionnement du système.

Figure 53. PLLs Initialization flowchart



*Diagramme de flux de (page 358)*

Nous remarquons que l'initialisation de la PLL commence dans le registre RCC\_CKSELR, dans lequel on détermine la source de l'horloge ainsi que le 1<sup>er</sup> pré-diviseur de fréquence.

Après un passage au **SRFs**, nous avons dans ce registre les valeurs suivantes :

All registers			
CKSELR			
Register	Address	Value	
STM32H7A3			
RCC			
RCC_PLLCKSELR	0x58024428	0x2020012	
PLLSRC	[0:2]	0x2	
DIVM1	[4:6]	0x1	
DIVM2	[12:6]	0x20	
DIVM3	[20:6]	0x20	

Ensuite, nous allons chercher la valeur **PLLSRC**, qui dans notre cas est 0.1.0, équivalente à 2 en décimal ou 0x2 en hexadécimal. Selon la documentation, il est indiqué que la PLL se synchronise sur la source HSE, généralement une horloge externe issue d'un quartz.

**Bits 2:0 SW[2:0]: system clock and trace clock switch**

Set and reset by software to select system clock and trace clock sources (**sys\_ck** and **traceportck**).

Set by hardware in order to:

- force the selection of the HSI or CSI (depending on STOPWUCK selection) when leaving a system Stop mode
- force the selection of the HSI in case of failure of the HSE when used directly or indirectly as system clock

000: HSI selected as system clock (**hsi\_ck**) (default after reset)

001: CSI selected as system clock (**csi\_ck**)

010: HSE selected as system clock (**hse\_ck**)

011: PLL1 selected as system clock (**pll1\_p\_ck** for **sys\_ck**, **pll1\_r\_ck** for **traceportck**)

others: reserved

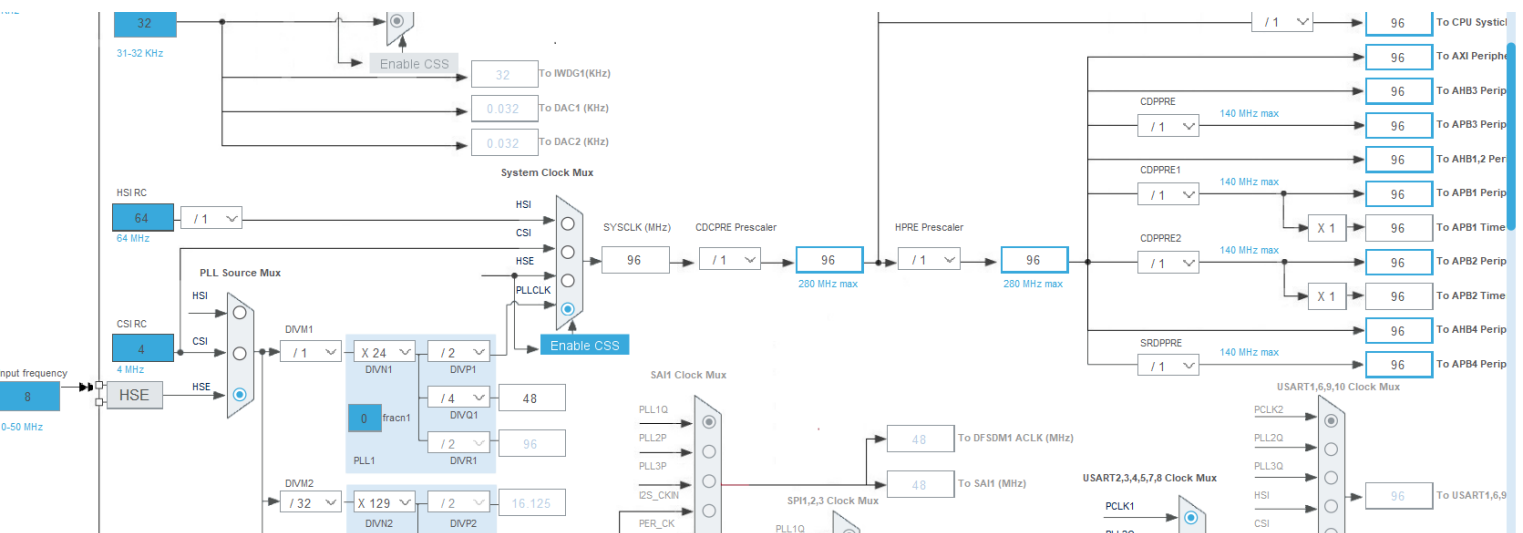
Choix de l'horloge en fonction de SW [2 ; 0]

Source : « [https://moodle.ensea.fr/pluginfile.php/71753/mod\\_resource/content/20/rm0455-stm32h7a37b3-and-stm32h7b0-value-line-advanced-armbased-32bit-mcus-stmicroelectronics.pdf](https://moodle.ensea.fr/pluginfile.php/71753/mod_resource/content/20/rm0455-stm32h7a37b3-and-stm32h7b0-value-line-advanced-armbased-32bit-mcus-stmicroelectronics.pdf) »

Cependant, malgré tous nos efforts, nous n'avons jamais pu localiser le quartz sur la carte afin de déterminer sa fréquence de départ. Nous avons pourtant scruté la documentation matérielle ainsi que la carte elle-même, mais aucun quartz ne semble être connecté. Par conséquent, nous avons pris la décision de retracer les valeurs des pré-diviseurs.

Selon le schéma de l'initialisation de la PLL, trois diviseurs entrent en jeu : DIVM1, DIVN1, et DIVP1. Cette information est confirmée dans l'IOC, où les valeurs respectives sont 1, x24, et /2.

➔ L'horloge est désormais connectée sur HSE, anciennement HSI



D'après le manuel de référence, page 398 : nous pouvons déterminer DIVM1. Et d'après la capture ci-dessus dans le **SFRs** la valeur est de 1.

**Bits 9:4 DIVM1[5:0]: prescaler for PLL1**

Set and cleared by software to configure the prescaler of the PLL1.

The hardware does not allow any modification of this prescaler when PLL1 is enabled (PLL1ON = 1).

In order to save power when PLL1 is not used, the value of DIVM1 must be set to 0.

000000: prescaler disabled

000001: division by 1 (bypass)

000010: division by 2

000011: division by 3

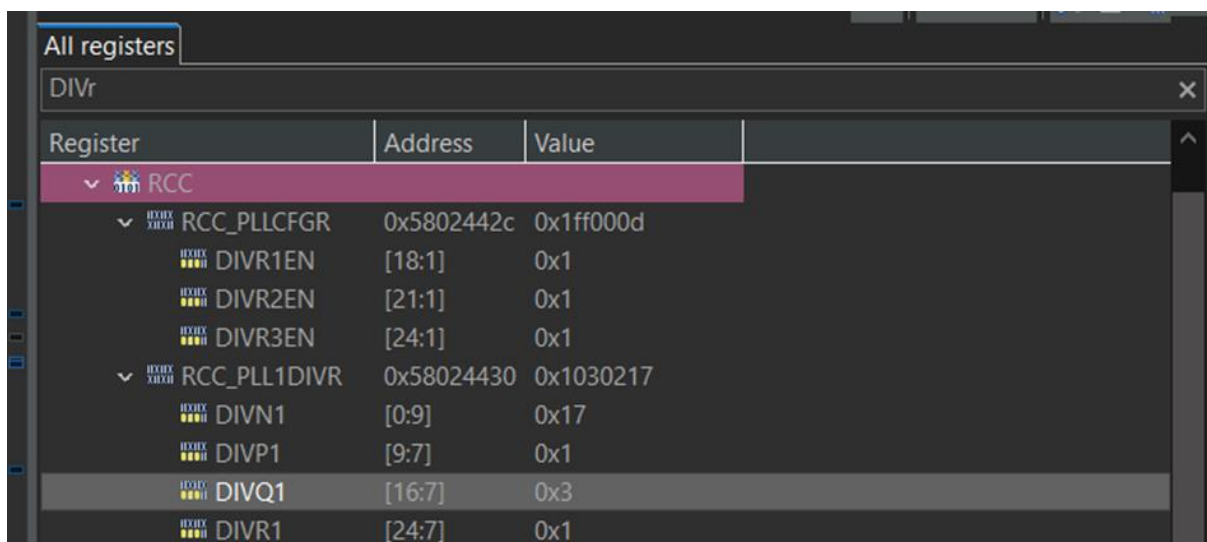
...

100000: division by 32 (default after reset)

...

111111: division by 63

Lorsque la valeur est de 0.0.0.0.0.1, la division est de 1, ce qui signifie que la PLL est contournée (bypass), ce qui confirme ce que nous observons dans l'IOC. Puisque nous travaillons avec la PLL1, conformément au diagramme de l'initialisation de la PLL, il est nécessaire d'examiner le registre RCC\_PLL1DIVR.



Register	Address	Value
<b>RCC</b>		
RCC_PLLCFGR	0x5802442c	0x1ff000d
DIVR1EN	[18:1]	0x1
DIVR2EN	[21:1]	0x1
DIVR3EN	[24:1]	0x1
<b>RCC_PLL1DIVR</b>	0x58024430	0x1030217
DIVN1	[0:9]	0x17
DIVP1	[9:7]	0x1
DIVQ1	[16:7]	0x3
DIVR1	[24:7]	0x1

En utilisant les **SFRs**, nous remarquons que la valeur de DIVN1 est définie comme 0x17, ce qui en binaire est équivalent à 1.0.1.1.1. Nous allons donc consulter la [documentation à la page 402](#) pour effectuer une comparaison.

Bits 8:0 **DIVN1[8:0]**: multiplication factor for PLL1 VCO  
 Set and reset by software to control the multiplication factor of the VCO.  
 These bits can be written only when the PLL is disabled (PLL1ON = PLL1RDY = 0).  
 .....: not used  
 0x006: wrong configuration  
 0x007: DIVN1 = 8  
 ...  
 0x080: DIVN1 = 129 (default after reset)  
 ...  
 0x1A3: DIVN1 = 420  
 Others: wrong configurations

**Caution:** The software must set correctly these bits to insure that the VCO output frequency is between its valid frequency range, that is:

- 128 to 560 MHz if PLL1VCOSEL = 0
- 150 to 420 MHz if PLL1VCOSEL = 1

VCO output frequency =  $F_{ref1\_ck} \times DIVN1$ , when fractional value 0 has been loaded into FRACN1, with:

- DIVN1 between 8 and 420
- The input frequency  $F_{ref1\_ck}$  must be between 1 and 16 MHz.

Effectivement, nous pouvons confirmer que pour la valeur 0x17, la multiplication est de 24, ce qui est en accord avec ce que nous observons dans l'IOC.

Désormais, nous nous intéressons au dernier pré-dividers soit DIVP1. Celui-ci est à 0x1 soit 0.0.0.0.0.1

Bits 15:9 **DIVP1[6:0]**: PLL1 DIVP division factor  
 Set and reset by software to control the frequency of the **pll1\_p\_ck** clock.  
 These bits can be written only when the PLL1 is disabled (PLL1ON = 0 and PLL1RDY = 0).  
 Note that odd division factors are not allowed.  
 0000000: not allowed  
 0000001: **pll1\_p\_ck** = **vco1\_ck** / 2 (default after reset)  
 0000010: not allowed  
 0000011: **pll1\_p\_ck** = **vco1\_ck** / 4  
 ...  
 1111111: **pll1\_p\_ck** = **vco1\_ck** / 128

Nous obtenons donc une division par 2 comme dans l'IOC. Nous avons bien retrouvé toutes les valeurs de divisions de l'IOC dans les registres tout est bien setup. Cependant, comme nous l'avons mentionné, sans connaître la fréquence du quartz connecté à HSE, il est impossible de déterminer la fréquence de la PLL\_CLOCK. Actuellement nous avons :

$$f_{initial} * DIVN_1 * DIVP_1 * DIVM_1 = f_{initial} * 1 * 24 * \frac{1}{2} = 12f_{initial}$$

On retrouve d'ailleurs les différentes valeurs de dividers dans notre code en C lors de la fonction clock\_initialisation.

## W6. Feuille d'évaluation sur la demande

```
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI48;
RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
RCC_OscInitStruct.HSI48State = RCC_HSI48_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 1;
RCC_OscInitStruct.PLL.PLLN = 24;
RCC_OscInitStruct.PLL.PLLP = 2;
RCC_OscInitStruct.PLL.PLLQ = 4;
RCC_OscInitStruct.PLL.PLLR = 2;
RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_3;
RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
RCC_OscInitStruct.PLL.PLLFRACN = 0;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
```

### Bases des Microcontrôleurs et Principes Fondamentaux :

Dans cette section initiale, nous avons exploré les principes fondamentaux cruciaux pour comprendre les microcontrôleurs. Un microcontrôleur est un circuit intégré qui sert de cerveau aux systèmes, exécutant des tâches et traitant des données. Son architecture repose sur divers composants, notamment un CPU, une mémoire, des ports d'entrée/sortie (E/S) et des périphériques.

### Composants de Mémoire :

1. **RAM (Mémoire Vive) :** La RAM est une mémoire volatile utilisée pour stocker les données en cours de traitement. Elle offre un accès rapide en lecture et en écriture et est essentielle pour le stockage temporaire des données lors de l'exécution du programme.
2. **ROM (Mémoire en Lecture Seule) :** La ROM contient une mémoire non volatile qui stocke le code du programme et les données essentielles. Elle est généralement utilisée pour stocker le firmware et le code de démarrage qui restent intacts même lorsque le microcontrôleur perd de l'alimentation.
3. **Mémoire Périphérique :** Les microcontrôleurs sont équipés de divers composants périphériques, notamment des minuteurs, des UART, des broches GPIO et des ADC. Ces périphériques interagissent avec des dispositifs externes et fournissent des fonctionnalités supplémentaires au microcontrôleur.

### Niveaux de Code :

1. **Code de Bas Niveau :** Au niveau le plus bas, le code machine ou les instructions en langage d'assemblage sont utilisés pour programmer le microcontrôleur, offrant un contrôle direct sur les composants matériels.
2. **Code de Haut Niveau :** En montant dans la hiérarchie d'abstraction, des langages de haut niveau comme le C dans notre cas sont utilisés pour écrire un code plus lisible par l'homme et plus facile à entretenir.

### HUART (UART Matériel) :



Le module UART matériel, ou HUART, est un périphérique clé qui facilite la communication série. Dans notre cas, c'était le Huart 3 qui était disponible. Il permet au microcontrôleur d'échanger des données avec des dispositifs externes ou d'autres microcontrôleurs en utilisant un protocole standardisé. Dans notre cas avec 115200 bauds, 8bits, pas de parité et 1 bit d'arrêt .

**PLL (Boucle à Verrouillage de Phase) :**

La Boucle à Verrouillage de Phase (PLL) est un composant utilisé pour la gestion de l'horloge dans les microcontrôleurs. Elle permet un contrôle précis de la fréquence d'horloge du microcontrôleur, permettant la synchronisation avec des dispositifs externes et une gestion efficace de l'alimentation. ici avec au moins 3 diviseurs de fréquence comme DIVM1, DIVN1, DIVP1 pour accéder à la PLL\_CLOCK (d'autres sont disponibles encore après. Ceci permet de changer la fréquence facilement pour économiser de l'énergie ou au contraire pour traiter beaucoup de données ou requests.

## 5. Gestion d'un DEL verte via des interrupteur

### Prep8. Horloges (Timer)

Dans un premier temps, nous analysons tous les timers présents sur notre microcontrôleur. Nous utilisons pour cela la documentation technique :

Ref(page26) : « [https://moodle.ensea.fr/pluginfile.php/80103/mod\\_resource/content/2/stm32h7a3zi\\_datasheet.pdf](https://moodle.ensea.fr/pluginfile.php/80103/mod_resource/content/2/stm32h7a3zi_datasheet.pdf) »

**Table 4. Timer feature comparison**

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz) <sup>(1)</sup>
Advanced-control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	140	280
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	140	280
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	140	280
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	140	280
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	140	280
	TIM15	16-bit	Up	Any integer between 1 and 65536	Yes	2	1	140	280
	TIM16, TIM17	16-bit	Up	Any integer between 1 and 65536	Yes	1	1	140	280
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	140	280
Low-power timer	LPTIM1, LPTIM2, LPTIM3	16-bit	Up	1, 2, 4, 8, 16, 32, 64, 128	No	0	No	140	280

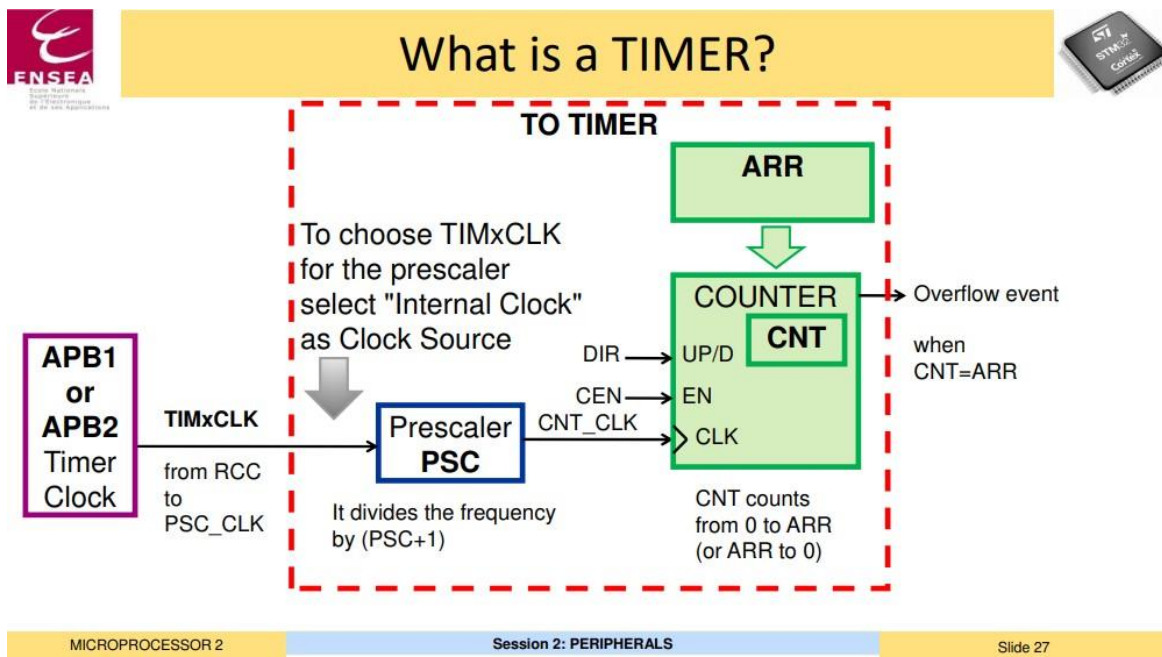
1. The maximum timer clock is up to 280 MHz depending on TIMPRE bit in the RCC\_CFGR register and CDPRE1/2 bits in RCC\_CDCFGFR register.

Pour chaque timer, on peut voir leur fréquence sur le tableau ci-dessus et leurs valeurs maximales pour ARR et PSC sont :

Timers	ARR max	PSC max
TIM1, TIM8	$2^{16} = 65\,536$	$2^{16}$

TIM2, TIM5	$2^{32} = 4\,294\,967\,296$	$2^{16}$
TIM3, TIM4	$2^{16}$	$2^{16}$
TIM12	$2^{16}$	$2^{16}$
TIM13, TIM14	$2^{16}$	$2^{16}$
TIM15	$2^{16}$	$2^{16}$
TIM16, TIM17	$2^{16}$	$2^{16}$
TIM6, TIM7	$2^{16}$	$2^{16}$

En outre, voici comment fonctionne un TIMER :



Les timers pouvant compter sur 32 bits sont les timers TIM2 et TIM5 comme indiqué dans le tableau :

TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	140	280
------------	--------	-------------------	---------------------------------	-----	---	----	-----	-----

Pour obtenir une Fin de Compte 2 fois par seconde, soit une interruption toutes les 0,5 secondes avec une horloge de 96MHz, nous utilisons la relation suivante :

$$ARR * (PSC + 1) = 48M$$

Voici les équations reformulées :

$$ARR+1 = TtimerThor \cdot PSC+1$$

$$ARR = TtimerThor \cdot PSC+1 - 1$$

$$ARR = 1/2 * 1/96 * E6 * 48000 - 1 = 999$$

Pour cela, nous pouvons choisir RCC\_APB1, contenant les timers 2, 3, 4, 5, ...,

### RCC APB1 peripheral clock enable register (RCC\_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UART8 EN	UART7 EN	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved
rw	rw	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
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

Voici leurs adresses :

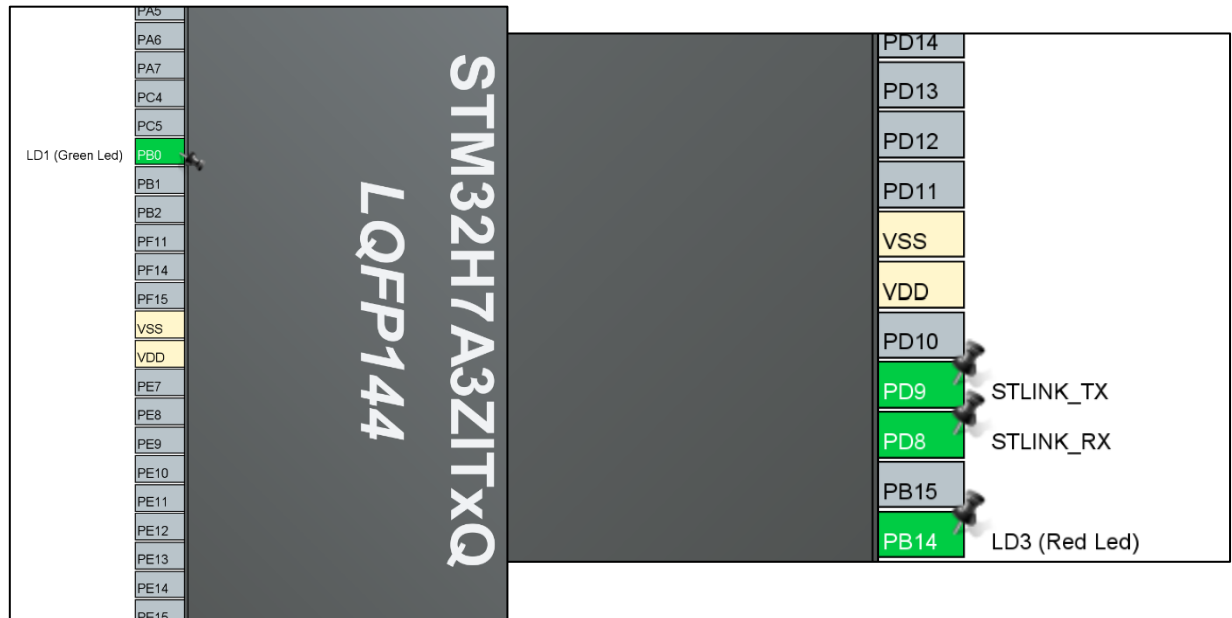
Ref(page 72) : « [https://moodle.ensea.fr/pluginfile.php/59307/mod\\_resource/content/9/RM0385\\_STM32F74xxx.pdf](https://moodle.ensea.fr/pluginfile.php/59307/mod_resource/content/9/RM0385_STM32F74xxx.pdf) »

**Table 1. STM32F4xx register boundary addresses (continued)**

Boundary address	Peripheral	Bus	Register map
0x4000 7400 - 0x4000 77FF	DAC	APB1	<a href="#">Section 14.5.15: DAC register map on page 453</a>
0x4000 7000 - 0x4000 73FF	PWR		<a href="#">Section 5.6: PWR register map on page 149</a>
0x4000 6800 - 0x4000 6BFF	CAN2		<a href="#">Section 32.9.5: bxCAN register map on page 1120</a>
0x4000 6400 - 0x4000 67FF	CAN1		
0x4000 5C00 - 0x4000 5FFF	I2C3		<a href="#">Section 27.6.11: I2C register map on page 875</a>
0x4000 5800 - 0x4000 5BFF	I2C2		
0x4000 5400 - 0x4000 57FF	I2C1		
0x4000 5000 - 0x4000 53FF	UART5		<a href="#">Section 30.6.8: USART register map on page 1021</a>
0x4000 4C00 - 0x4000 4FFF	UART4		
0x4000 4800 - 0x4000 4BFF	USART3		
0x4000 4400 - 0x4000 47FF	USART2		
0x4000 4000 - 0x4000 43FF	I2S3ext		<a href="#">Section 28.5.10: SPI register map on page 928</a>
0x4000 3C00 - 0x4000 3FFF	SPI3 / I2S3		
0x4000 3800 - 0x4000 3BFF	SPI2 / I2S2		
0x4000 3400 - 0x4000 37FF	I2S2ext		<a href="#">Section 21.4.5: IWDG register map on page 713</a>
0x4000 3000 - 0x4000 33FF	IWDG		
0x4000 2C00 - 0x4000 2FFF	WWDG		<a href="#">Section 22.6.4: WWDG register map on page 720</a>
0x4000 2800 - 0x4000 2BFF	RTC & BKP Registers		<a href="#">Section 26.6.21: RTC register map on page 837</a>
0x4000 2000 - 0x4000 23FF	TIM14		<a href="#">Section 19.5.12: TIM10/11/13/14 register map on page 694</a>
0x4000 1C00 - 0x4000 1FFF	TIM13		
0x4000 1800 - 0x4000 1BFF	TIM12		<a href="#">Section 19.4.13: TIM9/12 register map on page 684</a>
0x4000 1400 - 0x4000 17FF	TIM7		<a href="#">Section 20.4.9: TIM6&amp;TIM7 register map on page 708</a>
0x4000 1000 - 0x4000 13FF	TIM6		
0x4000 0C00 - 0x4000 0FFF	TIM5		<a href="#">Section 18.4.21: TIMx register map on page 646</a>
0x4000 0800 - 0x4000 0BFF	TIM4		
0x4000 0400 - 0x4000 07FF	TIM3		
0x4000 0000 - 0x4000 03FF	TIM2		

## Prep9. Horloges, DELs et MLI

Pour la led verte est connectée PB0 qui est régit par D13 Et la led rouge est régit par le



pin PB14.

Cela est confirmé par la documentation technique :

### 6.6 LEDs

**User LD1:** a green user LED is connected to the STM32H7 I/O PB0 (SB65 OFF and SB54 ON) or PA5 (SB65 ON and SB54 OFF) corresponding to the ST Zio D13.

**User LD2:** a yellow user LED is connected to PE1.

**User LD3:** a red user LED is connected to PB14.

De plus, dans le manuel d'utilisateur on trouve :

31	D33	TIMER_D_ <b>PWM1</b>	PB0	TIM3_CH3	-
----	-----	----------------------	-----	----------	---

Donc la LED verte est connectée au timer TIM3\_CH3.

On se base sur l'IOC pour déterminer le timer pour la LED rouge car l'information n'est pas précisée dans le manuel d'utilisation

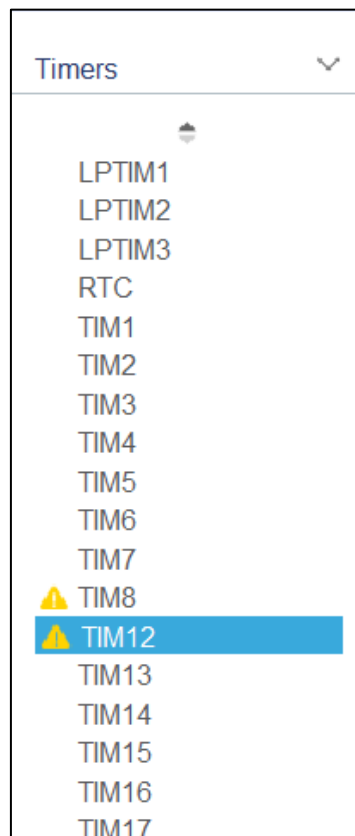


## Prep10. Horloges et Handler code

tim8_brk_it	50	43	TIM8_BRK_TIM12	TIM8 break and TIM12 global interrupts	0x0000 00EC
tim12_gbl_it					
tim8_upd_it	51	44	TIM8_UP_TIM13	TIM8 update and TIM13 global interrupts	0x0000 00F0
tim13_gbl_it					
tim8_trg_it	52	45	TIM8_TRG_COM _TIM14	TIM8 trigger /commutation and TIM14 global interrupts	0x0000 00F4
tim14_gbl_it					
tim8_cc_it	53	46	TIM8_CC	TIM8 capture / compare interrupts	0x0000 00F8

## W11. Vérification des Horloges via le contrôleur IOC

A partir de l'IOC, nous pouvons voir la liste des timers disponibles :

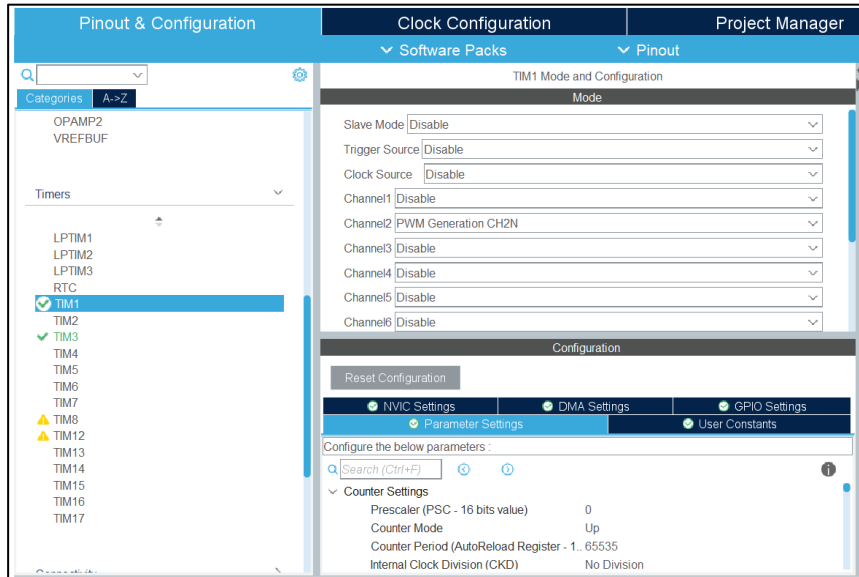


Cependant les timers 8 et 12 ne sont pas complètement disponibles. Nous utiliserons donc le timer 3.

## W12. Contrôle de la Del verte

Ainsi, comme les clocks 8 et 12 sont bloquées on configure les timers 1 et 3 sur les PWM sur les bons channels comme sur les captures ci-dessous.

Pin N...	Signal on...	GPIO out...	GPIO mo...	GPIO Pul...	Maximum...	Fast Mode	User Label	Modified
PB0	n/a	Low	Output P...	No pull-u...	Low	n/a	LD1 (Gr...	✓



La LED verte est connectée à la broche PB0. Le GPIO n'a ni Pull-up ni Pull-down.

Après nous générons le code suite à ces configurations et on a ceci :

```

406 HAL_GPIO_WritePin(USB_FS_PWR_EN_GPIO_Port, USB_FS_PWR_EN_Pin, GPIO_PIN_RESET);
407
408 /*Configure GPIO pin Output Level */
409 HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD2_Pin, GPIO_PIN_RESET);
410
411 /*Configure GPIO pin Output Level */
412 HAL_GPIO_WritePin(LD2E1_GPIO_Port, LD2E1_Pin, GPIO_PIN_RESET);
413

```

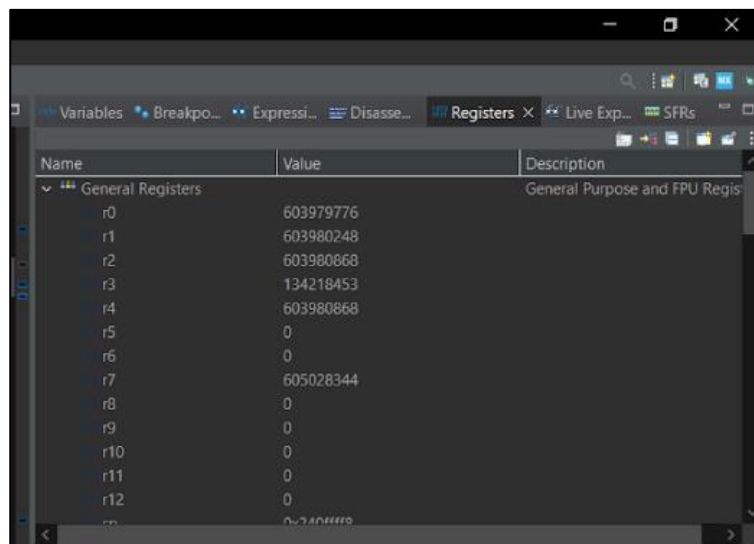
On remarque bien que la LD1 et LD2 ont été set sur le GPIOB. C'est ici que nous irons chercher en premier dans le SFRS.

Le GPIO correspondant à la led verte est le GPIOB0, nous pouvons voir la commande HAL\_GPIO\_Init dans le code suivant :

```
/*Configure GPIO pins : LD1_Pin LD3_Pin */
GPIO_InitStruct.Pin = LD1_Pin|LD3_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

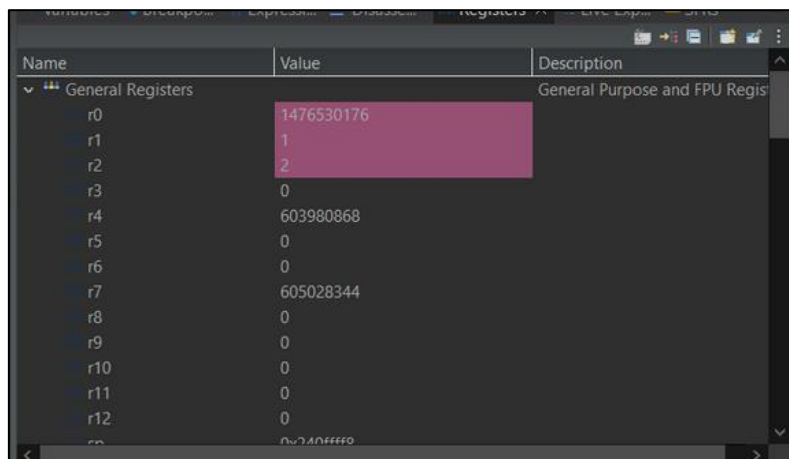
On regarde avant et après les configurations :

AVANT :



Name	Value	Description
General Registers		
r0	603979776	General Purpose and FPU Registers
r1	603980248	
r2	603980868	
r3	134218453	
r4	603980868	
r5	0	
r6	0	
r7	605028344	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	

APRES :

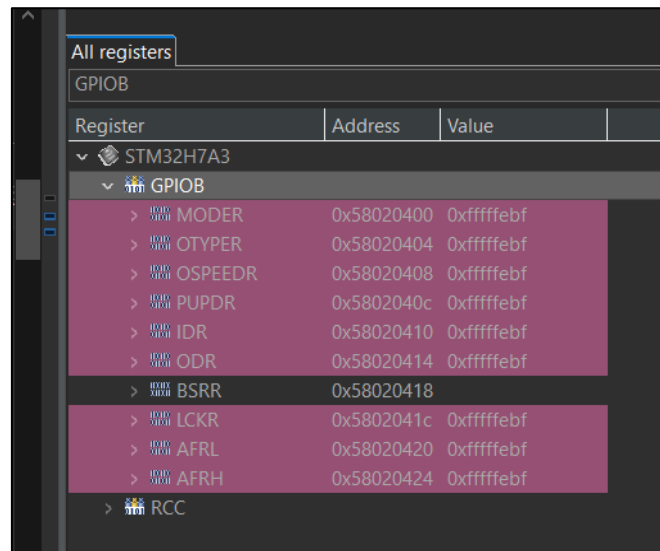


Name	Value	Description
General Registers		
r0	1476530176	General Purpose and FPU Registers
r1	1	
r2	2	
r3	0	
r4	603980868	
r5	0	
r6	0	
r7	605028344	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	

On a donc bien la configuration de R1 et R2.

Puis, dans le GPIOB on regarde avant et après la configuration :

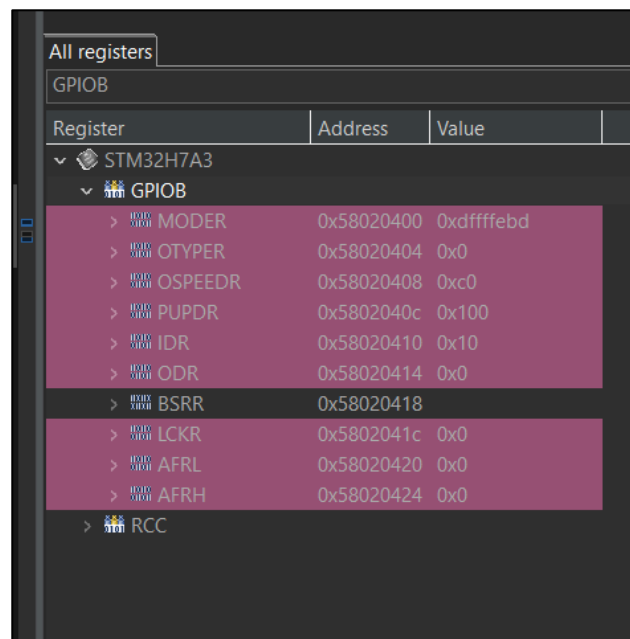
AVANT :



The screenshot shows the 'All registers' window in STM32CubeIDE. The 'GPIOB' register block is expanded, showing a list of registers with their addresses and current values. The values for most registers are 0xffffffff, indicating they are in their default state.

Register	Address	Value
MODER	0x58020400	0xffffffff
OTYPER	0x58020404	0xffffffff
OSPEEDR	0x58020408	0xffffffff
PUPDR	0x5802040c	0xffffffff
IDR	0x58020410	0xffffffff
ODR	0x58020414	0xffffffff
BSRR	0x58020418	
LCKR	0x5802041c	0xffffffff
AFRL	0x58020420	0xffffffff
AFRH	0x58020424	0xffffffff

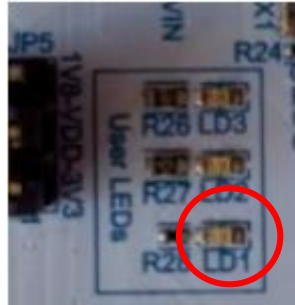
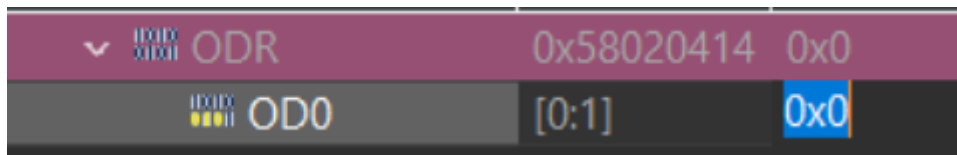
APRES :



The screenshot shows the same register view as before, but the values have been updated to reflect the configuration. The MODER register is now 0xdffffebd, and the OTYPER, OSPEEDR, PUPDR, IDR, ODR, LCKR, AFRL, and AFRH registers are all 0x0.

Register	Address	Value
MODER	0x58020400	0xdffffebd
OTYPER	0x58020404	0x0
OSPEEDR	0x58020408	0xc0
PUPDR	0x5802040c	0x100
IDR	0x58020410	0x10
ODR	0x58020414	0x0
BSRR	0x58020418	
LCKR	0x5802041c	0x0
AFRL	0x58020420	0x0
AFRH	0x58020424	0x0

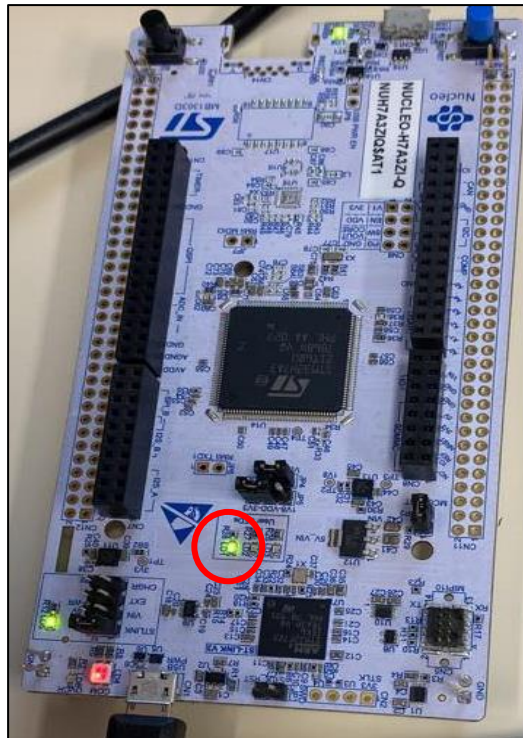
Dans la vue SFR, nous pouvons basculer manuellement le registre ODR pour allumer et éteindre la LED et l'éteindre.



On voit que tout a été bien initialisé. En plus si on joue avec le OD0 et qu'on le set à 1 la led verte s'allume.

All registers			
GPIOB			
Register	Address	Value	
> OSPEEDR	0x58020408	0xc0	
> PUPDR	0x5802040c	0x100	
> IDR	0x58020410	0x11	
▼ ODR	0x58020414	0x1	
OD0	[0:1]	0x1	
OD1	[1:1]	0x0	
OD2	[2:1]	0x0	

Résultat sur la carte :



Nous avons bien la LED LD1 verte qui s'allume.



## W13. Configuration de l'Horloge A

Pour configurer proprement les horloges on doit définir le PSC et l'ARR. On sait que SYSCLK est à 96MHz et que l'on veut que la LED s'allume à la fréquence de 1Hz.

### PHOTO

$$T_{IT} = (PSC + 1) * (ARR + 1) * \frac{1}{f_{SYSCLK}} \Leftrightarrow 0,5s * 96 MHz = (PSC + 1) * (ARR + 1)$$

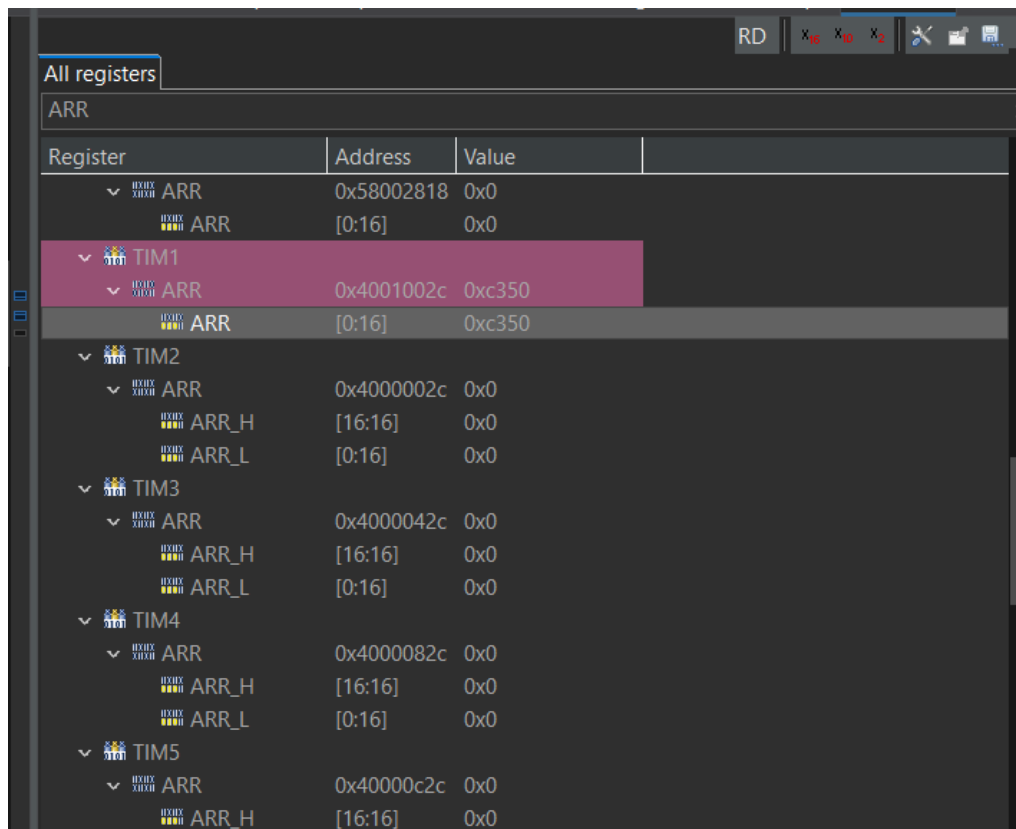
$$PSC = T_{IT} * ARR$$

$$PSC = \frac{1}{2} * \frac{1}{96 * 10^6} * 48000 - 1 = 999$$

En résumé, assurez-vous que ces équations sont bien adaptées à votre application spécifique, et n'oubliez pas de consulter la documentation de votre microcontrôleur pour garantir une configuration correcte du timer.

On peut donc choisir ARR = 49 999 et PSC = 999 pour que cela soit simple.

On définit donc le TIMER avec le PSC et le ARR



On voit bien dans la mémoire que cela a été set à 0x3C0 soit 960 pour le prescaler.

Pour la mémoire en ARR on a 0XC350 ce qui correspond bien à 50000.

On remarque aussi qu'avant et après l'initialisation de TIM1 le registre CNT ne change absolument pas. Ceci semble cohérent puisque nous n'avons pas du tout set le CEN donc le NCT ne bouge pas nous le voyons dans la doc :

The counter is clocked by the prescaler output CK\_CNT, which is enabled only when the counter enable bit (CEN) in TIMx\_CR1 register is set (refer also to the slave mode controller description to get more details on counter enabling).

Par conséquent, il suffit de changer la valeur de CEN pour le timer 1 pour que le CNT finisse par fonctionner.

Pour la suite du TP, nous utilisons le Timer2 configuré de la même manière puisque le timer1 ne possède pas de global interrupt.

## W14 : Gestion de la LED via l'interrupteur horloge A

On met le NVIC avec la priorité à 1 dans l'ioc.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD and PVM interrupts through EXTI line	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
EXTI line[9:5] interrupts	<input type="checkbox"/>	0	0
TIM2 global interrupt	<input checked="" type="checkbox"/>	1	0
USART3 global interrupt	<input type="checkbox"/>	0	0
FPU global interrupt	<input type="checkbox"/>	0	0
HSEM1 global interrupt	<input type="checkbox"/>	0	0
ECC diagnostic Global Interrupt	<input type="checkbox"/>	0	0

☒ Enabled    Preemption Priority 1    Sub Priority 0

On voit bien qu'après avoir correctement réglé notre NVIC la génération de code fonctionne :

```
static void MX_NVIC_Init(void)
{
    /* TIM2_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(TIM2_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(TIM2_IRQn);
}
```

En allant regarder dans le SFRS et les registres on a bien :

<input checked="" type="checkbox"/> 1010 0101	NVIC_ISER0	0xe000e100	0x10000000
<input checked="" type="checkbox"/> 1010 0101	NVIC_ICER0	0xe000e180	0x10000000
> 1010 0101	NVIC_IPR7	0xe000e41c	0x10

Concrètement :

- NVIC\_ISER0 permet d'activer les interruptions 0 à 31. La valeur 0x10000000 indique que l'interruption TIM2 est activée.
- NVIC\_ICER0 permet de désactiver les interruptions 0 à 31. La valeur 0x10000000 indique que l'interruption TIM2 est désactivée.
- NVIC\_IPR7 définit la priorité de l'interruption TIM2. La valeur `0x10` signifie que la priorité est définie à 1.

Les deux images suivantes montrent le code utilisé pour faire clignoter la led verte. Le premier montre l'initialisation tandis que la seconde concerne la réalisation du code.

Nous ajoutons donc dans le global interrupt le code à exécuter. Donc le changement d'état de la led :

```

11 /**
12  * @brief This function handles TIM2 global interrupt.
13  */
14 void TIM2_IRQHandler(void)
15 {
16     /* USER CODE BEGIN TIM2_IRQn 0 */
17     HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
18     /* USER CODE END TIM2_IRQn 0 */
19     HAL_TIM_IRQHandler(&htim2);
20     /* USER CODE BEGIN TIM2_IRQn 1 */
21
22     /* USER CODE END TIM2_IRQn 1 */
23 }
24

```

Après avoir ajouté dans le main la boucle while(1) pour ne pas quitter le programme et faire planter le microprocesseur.

## 6 Plotting signals

### Preparation 15: Very basic DSP

Dans notre test, nous enverrons 50 échantillons à la fois. Si F est de 20Hz, combien de périodes du signal avec 50 échantillons ? Si F est de 50Hz, combien de périodes du signal devons-nous voir avec 50 échantillons ?

Combien de périodes du signal devons-nous observer avec 50 échantillons ?

La période d'un signal sinusoïdal est la durée nécessaire pour effectuer un cycle complet, soit  $1/F$ .

Avec une fréquence d'échantillonnage de 1kHz (ou 1000 échantillons par seconde), nous prenons 50 échantillons à la fois.

Pour comprendre combien de périodes du signal F sont capturées dans ces 50 échantillons, nous pouvons calculer la durée couverte par ces échantillons :

#### Période d'échantillonnage x Nombre d'échantillons

Nous savons alors combien de périodes correspondent à la période du signal

- 1)  $F = 20\text{Hz}$   
 $T = 1/F = 50\text{ms}$   
 $1\text{ms/échantillon} * 50 \text{ échantillons} = 50\text{ms}$   
 Nombre de périodes =  $50\text{ms}/50\text{ms} = \underline{\underline{1 \text{ période}}}$
- 2)  $F = 50\text{Hz}$   
 $T = 1/F = 20\text{ms}$   
 $1\text{ms/échantillon} * 20 \text{ échantillons} = 20\text{ms}$   
 Nombre de périodes =  $50\text{ms}/20\text{ms} = \underline{\underline{2,5 \text{ périodes}}}$

## W16 : Timer with $\mu$ s accuracy

Dans cette question, on cherche à générer un temps de référence pour calculer notre sinus. On crée donc deux Timers noté TIMA et TIMB.

Configuration du TIMA : Dans le TP sera le Timer 5

- Doit avoir comme temps de référence  $1\mu$ s
- Possède un global interrupt
- Utilise ce global interrupt pour vérifier que le TIMB ait un temps de référence de  $1\mu$ s.

Configuration TIMB : Dans le TP sera le Timer 2

- Doit avoir un temps de référence de  $1\mu$ s.
- Sert juste à compter donc n'a pas de global interrupt.

Nous avons décidé de choisir le TIM2 et TIM5 car ce sont les deux sur 32 bits ce qui donne plus de flexibilité. On choisit donc,

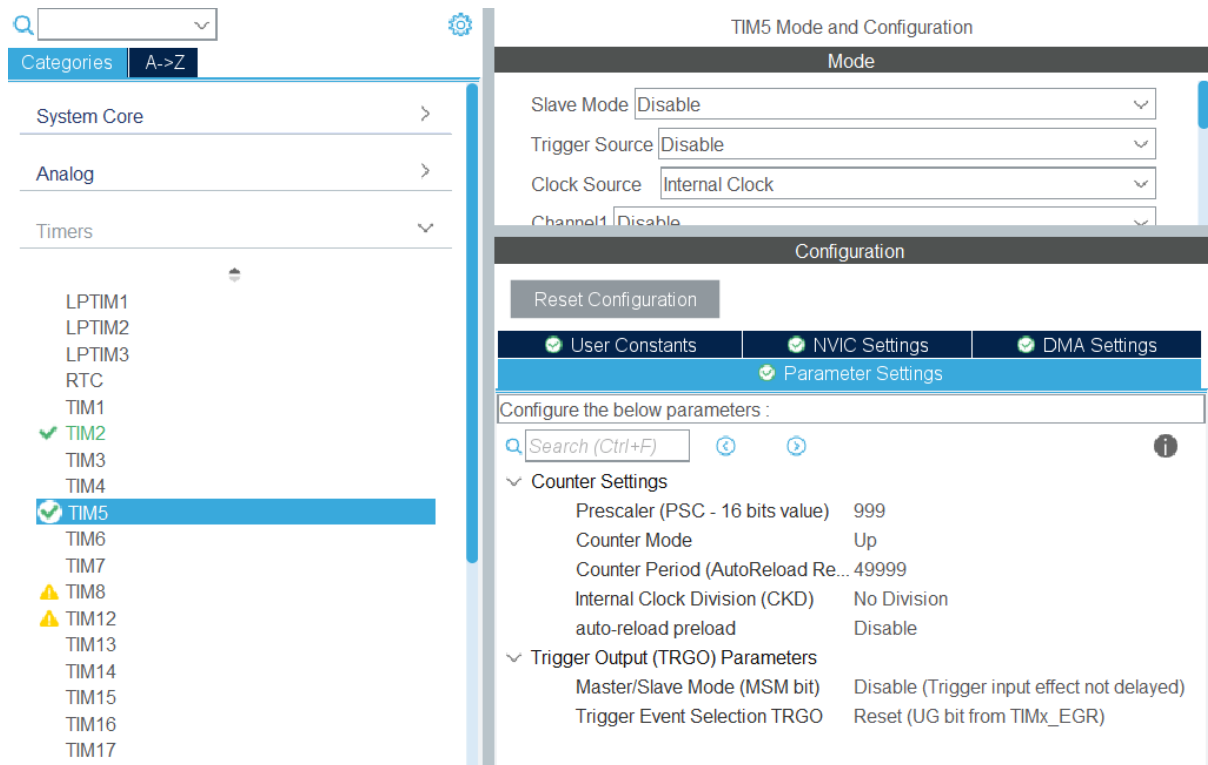
Pour le TIM2 réglé à  $1\mu$ s un ARR = 999999 et PSC = 95

The screenshot shows the STM32CubeMX Pinout & Configuration window. On the left, the 'Timers' category is expanded, and TIM2 is selected. The right pane shows the 'TIM2 Mode and Configuration' settings. The 'Mode' section has 'Slave Mode' set to 'Disable', 'Trigger Source' set to 'Disable', 'Clock Source' set to 'Internal Clock', and 'Channel1' set to 'Disable'. The 'Configuration' section has a 'Reset Configuration' button. Below this are tabs for 'User Constants', 'NVIC Settings', and 'DMA Settings', with 'Parameter Settings' currently active. The 'Parameter Settings' section shows the following configuration for TIM2:

Configure the below parameters :	
Search (Ctrl+F)	
Counter Settings	
Prescaler (PSC - 16 bits value)	95
Counter Mode	Up
Counter Period (AutoReload Re...)	999999
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	

Pour le TIM5 réglé à 1s un ARR = 49 999 et PSC = 999





Une fois cela fait nous générons le code et y ajoutons le STAMP et récupérons la valeur du compte de TIM2 :

```

/**
 * @brief This function handles TIM5 global interrupt.
 */
void TIM5_IRQHandler(void)
{
    /* USER CODE BEGIN TIM5_IRQn 0 */
    int t = TIM2->CNT;
    SERIAL_SendCharBuf("<<Debut LAB16>>");
    SERIAL_SendNL();
    SERIAL_SendInt(t);
    SERIAL_SendNL();

    ITM_Port32(31)=1;
    /* USER CODE END TIM5_IRQn 0 */
    HAL_TIM_IRQHandler(&htim5);
    /* USER CODE BEGIN TIM5_IRQn 1 */

```

On a donc:

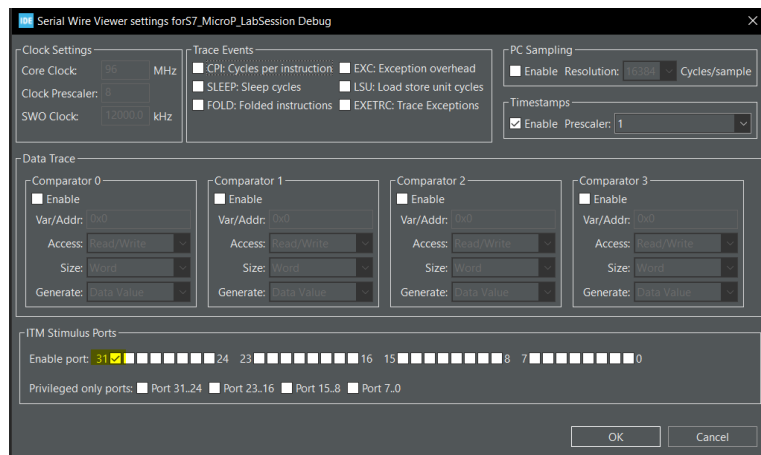
fonction TIM5 Handler

Récupération de la valeur de compte de TIM2

Ecriture avec Serial qui sera lu dans TeraTerm

Ajout du Stamp sur le port 31 avec en écriture 1

Afin que le Stamp fonctionne correctement, on active le port 31 afin de le lire dans la fenêtre SWV.



Nous avons surligné en jaune l'activation du port 31.

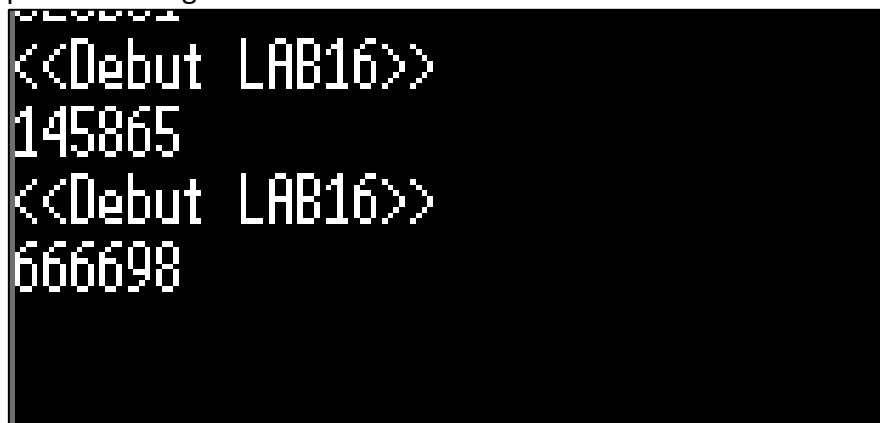
Ensuite nous lançons TeraTerm avec bon nombre de Baud (115200) afin que tout soit bien affiché dedans.

Ainsi on obtient avec le stamp :

57	ITM Port 31	1	718747840	7.486957 s
58	Sync	48	722747838	7.528623 s
59	Sync	48	738747830	7.695290 s
60	Sync	48	756747821	7.882790 s
61	ITM Port 31	1	768747816	8.007790 s

Le laps de temps entre deux Stamps est de 0,520833s ce qui est cohérent car le temps de référence est de 1s.

Or sur cette période on regarde dans TeraTerm :



Le compteur a compté 520833 sur ce laps de temps. Afin de déterminer le temps de référence on a donc  $\frac{0,520833}{520833} = 0,000001 = 1\mu s$

Le temps de référence de TIMB est donc bien validé à exactement

## W17 : Sending samples one after the other (1kHz sampling)

Cette fois ci on cherche à obtenir un trigger à 1ms. Par conséquent on recalcule le PSC et l'ARR du Timer :

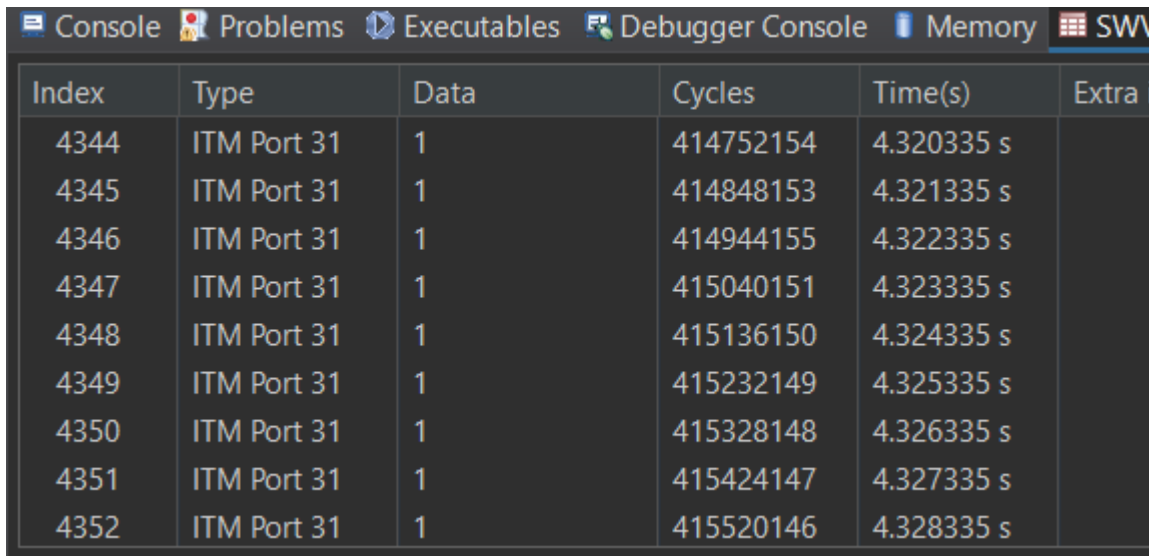
$$T_{IT} = (PSC + 1) * (ARR + 1) * \frac{1}{f_{SYSCLK}} = 0.001s$$

$$(PSC + 1) * (ARR + 1) = 96000$$

Donc PSC = 95 et ARR = 999

On le configure via l'IOC et on garde en TIM2 le timer réglé sur un décompte à 1µs.

On décide d'ajouter un TimeStamp pour vérifier que notre timer se trigger toute les ms.



Index	Type	Data	Cycles	Time(s)	Extra
4344	ITM Port 31	1	414752154	4.320335 s	
4345	ITM Port 31	1	414848153	4.321335 s	
4346	ITM Port 31	1	414944155	4.322335 s	
4347	ITM Port 31	1	415040151	4.323335 s	
4348	ITM Port 31	1	415136150	4.324335 s	
4349	ITM Port 31	1	415232149	4.325335 s	
4350	ITM Port 31	1	415328148	4.326335 s	
4351	ITM Port 31	1	415424147	4.327335 s	
4352	ITM Port 31	1	415520146	4.328335 s	

Ainsi on veut envoyer les données de cosinus et de sinus sur 50 samples :

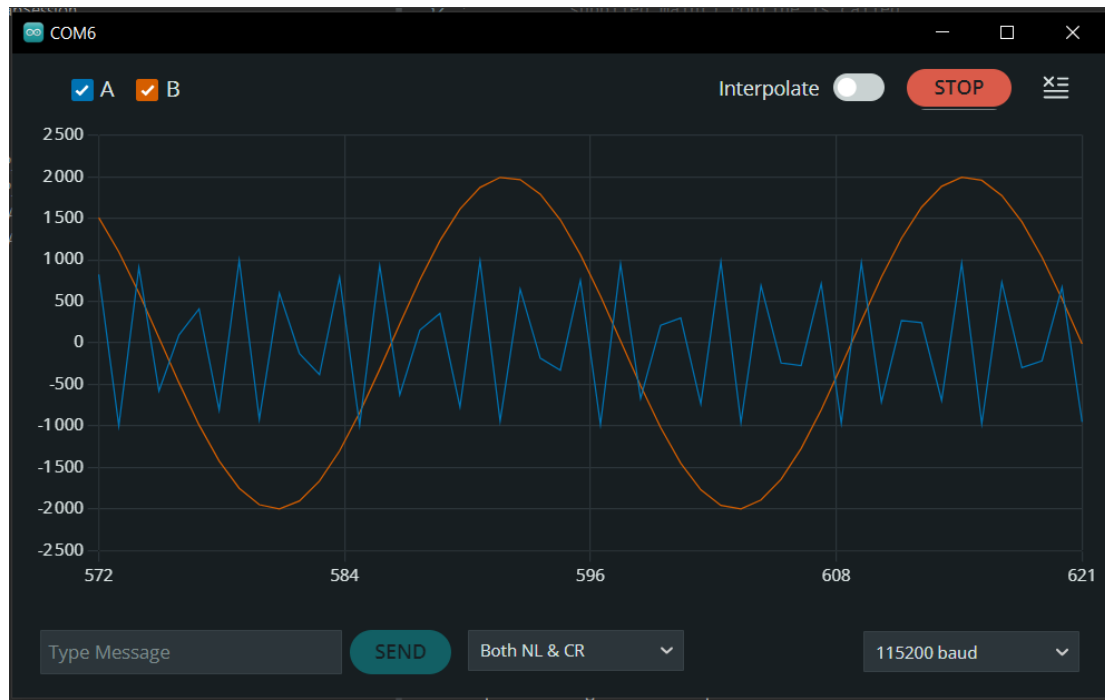
```

221 void TIM5_IRQHandler(void)
222 {
223     /* USER CODE BEGIN TIM5_IRQn 0 */
224
225     int t = TIM2->CNT;
226     //SERIAL_SendInt(t);
227     int A = 1000;
228     int B = 2000;
229
230     float Fa = 20;
231     float Fb = 50;
232
233     float sig1 = A*sin(2*3.14*Fa*t);
234     float sig2 = B*cos(2*3.14*Fb*t);
235
236     if(count<100){
237         ITM_Port32(31)=1;
238         SERIAL_SendNL();
239         SERIAL_SendCharBuf("A:");
240         SERIAL_SendFloat(sig1);
241         SERIAL_SendCharBuf(", B:");
242         SERIAL_SendFloat(sig2);
243         count++;
244         ITM_Port32(31)=2;
245     }
246

```

On envoi bien la donnée de la forme A: XXXXXXXXX, B: XXXXXXXX

Le cosinus est réglé à 50 Hz donc comme on prend uniquement 50 samples donc 50ms on est censé voir 2 périodes et demi. Pour le sinus à 20 Hz on est censé en voir qu'une seule période.



Comme convenu on voit bien les 2 périodes et demi du cosinus (SignalB en rouge) mais par contre le sinus est inexploitable. De même le cosinus est segmenté ce qui peut provenir d'un sous-échantillonnage des signaux. Pour cela on ajoute un Timestamp dans le if d'affichage afin de déterminer le temps d'exécution.

```

if(count<100){
    ITM_Port32(31)=1;
    SERIAL_SendNL();
    SERIAL_SendCharBuf("A:");
    SERIAL_SendFloat(sig1);
    SERIAL_SendCharBuf(", B:");
    SERIAL_SendFloat(sig2);
    count++;
    ITM_Port32(31)=2;
}

/* USER CODE END TIM5_IRQn 0 */
HAL_TIM_IRQHandler(&htim5);
/* USER CODE BEGIN TIM5_IRQn 1 */

/* USER CODE END TIM5_IRQn 1 */

```

Type	Data	Cycles	Time(s)	Extra info
ITM Port 31	1	18650466	194.275687...	
ITM Port 31	2	18853908	196.394875...	
ITM Port 31	1	18864271	196.502823...	
ITM Port 31	2	19084281	198.794594...	
ITM Port 31	1	19097549	198.932802...	
ITM Port 31	2	19301150	201.053646...	
ITM Port 31	1	19311125	201.157552...	
ITM Port 31	2	19522842	203.362938...	
ITM Port 31	1	19534751	203.486990...	

Puis on fait la différence entre 1 et 2 : On voit que l'exécution est de plus de 2s alors que l'on trigger le timer toute les ms. Par conséquent des valeurs sont perdues, l'échantillonnage est très mauvais et donc les courbes sont peu lisibles.

Si on enlève l'affichage du script on obtient :

```

239     if(cntEchantillon<=100){
240         ITM_Port32(31) = 1;
241     /* SERIAL_SendNL();
242        SERIAL_SendCharBuf("A: ");
243        SERIAL_SendFloat(sig1);
244        SERIAL_SendCharBuf(", B: ");
245        SERIAL_SendFloat(sig2); */
246         ITM_Port32(31) = 2;
247         cntEchantillon++;
248     }

```

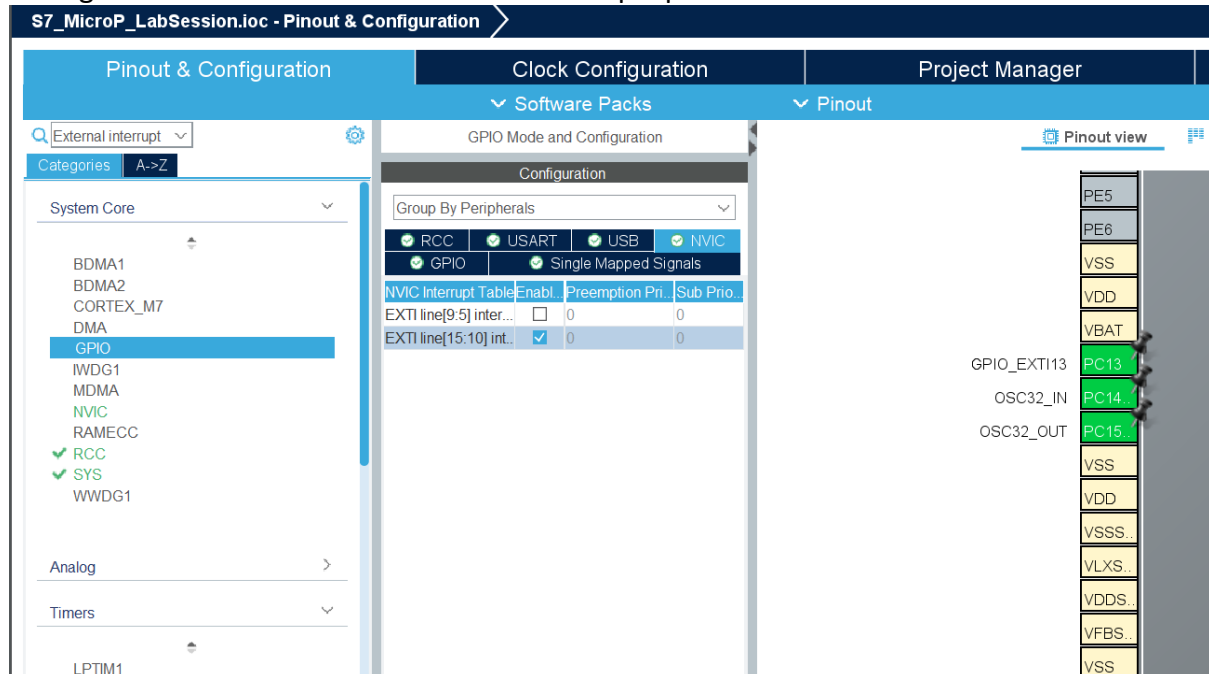
Index	Type	Data	Cycles	Time(s)
192	ITM Port 31	2	9171602	95.537521 ...
193	ITM Port 31	1	9267860	96.540208 ...
194	ITM Port 31	2	9267867	96.540281 ...
195	ITM Port 31	1	9363584	97.537333 ...
196	ITM Port 31	2	9363591	97.537406 ...
197	ITM Port 31	1	9459573	98.537219 ...
198	ITM Port 31	2	9459580	98.537292 ...
199	ITM Port 31	1	9555251	99.533865 ...
200	ITM Port 31	2	9555258	99.533938 ...

D'après les valeurs on voit que le script s'exécute en 73μs. Par conséquent, pour éviter un sous-échantillonnage, l'affichage doit se faire en 926μs maximum.

## W18 : Re-triggering a sending when pushing blue button

On ne souhaite pas améliorer le processus mais juste ajouter 50 samples en plus avec le bouton bleu.

On remarque que le bouton est lié au PC13 de la carte par conséquent on va dans l'IOC pour configurer le NVIC avec l'activation des interrupts par EXTI-LINE.



Maintenant que le bouton bleu a été configuré nous allons l'utiliser pour améliorer notre processus. En effet actuellement nous envoyons les données une à une pour les plots ensuite ce qui force la communication pour chaque point ce qui est long. Une façon d'améliorer et de préparer 50 points dans un tableau que l'on enverra en un seul coup au plot pour y observer le résultat.

On déclare donc les tableaux pour le cosinus et le sinus.

```

45 /* Private variables -----
46 /* USER CODE BEGIN PV */
47 int cntEchantillon;
48 int dataA[50];
49 int dataB[50];
50 /* USER CODE END PV */
51

```

Dans l'interruption du bouton on reset le compte des échantillons qui a été déclaré en variable globale.



```

096 void EXTI15_10_IRQHandler(void)
10 {
11     /* USER CODE BEGIN EXTI15_10_IRQn 0 */
12     cntEchantillon = 0;
13     /* USER CODE END EXTI15_10_IRQn 0 */
14     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
15     /* USER CODE BEGIN EXTI15_10_IRQn 1 */
16
17     /* USER CODE END EXTI15_10_IRQn 1 */
18 }
19

```

Puis on remplit les tableaux tant qu'ils ne sont pas pleins. S'ils sont pleins on les affiche ce qui réduit drastiquement le temps de traitement et donc améliore le programme.

Le bouton réinitialise le compteur et donc remplace le tableau avec les nouvelles valeurs ce qui "met à jour" le graph.

```

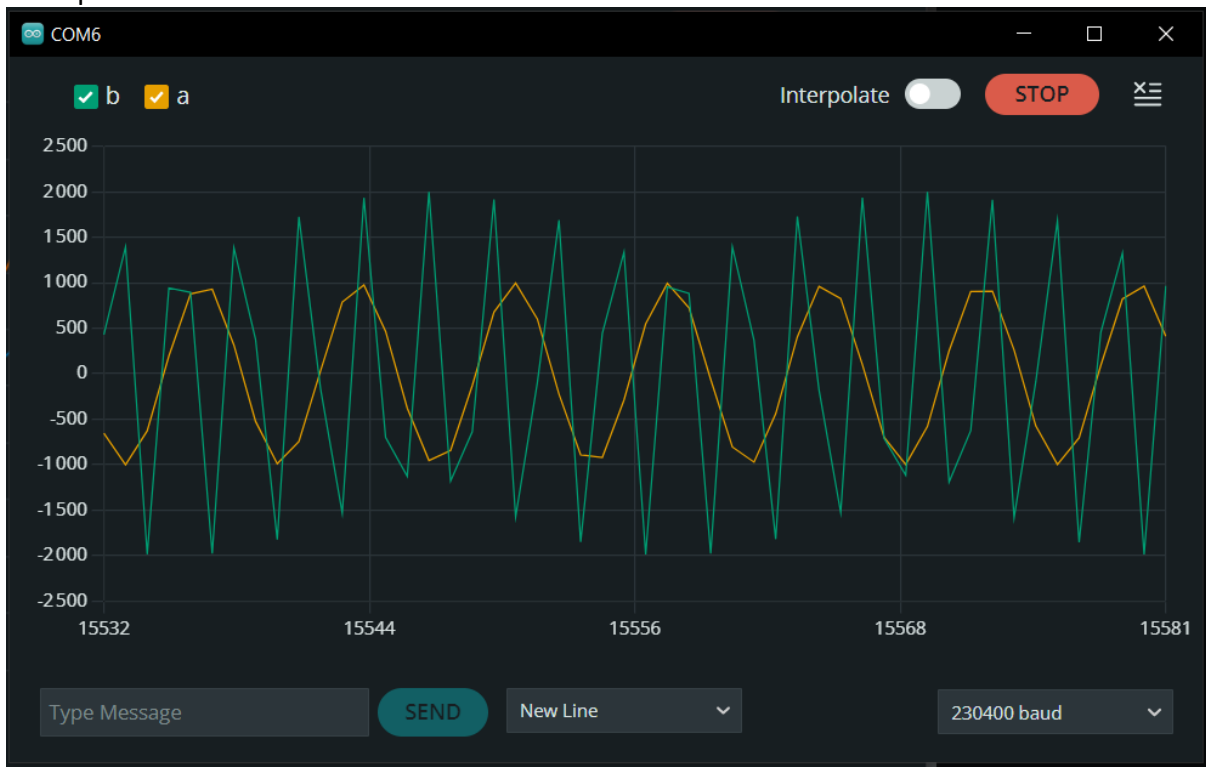
223 void TIM5_IRQHandler(void)
224 {
225     /* USER CODE BEGIN TIM5_IRQn 0 */
226
227     int t = TIM2->CNT;
228     //SERIAL_SendInt(t);
229     int A = 1000;
230     int B = 2000;
231
232     float Fa = 20;
233     float Fb = 50;
234
235     float sig1 = A*sin(2*3.14*Fa*t);
236     float sig2 = B*cos(2*3.14*Fb*t);
237
238     if(cntEchantillon<=100){
239         dataA[cntEchantillon] = sig1;
240         dataB[cntEchantillon] = sig2;
241         cntEchantillon++;
242     }
243
244     if(cntEchantillon == 100){
245         SERIAL_SendToPlot(dataA, dataB, cntEchantillon);
246     }
247
248     /* USER CODE END TIM5_IRQn 0 */
249     HAL_TIM_IRQHandler(&htim5);
250     /* USER CODE BEGIN TIM5_IRQn 1 */
251
252     /* USER CODE END TIM5_IRQn 1 */
253 }
254

```

Vidéo qui le montre : [https://drive.google.com/file/d/1HF\\_zwyUfezu7-AFNQ2vgOH7GLtid5ADS/view?usp=drive\\_link](https://drive.google.com/file/d/1HF_zwyUfezu7-AFNQ2vgOH7GLtid5ADS/view?usp=drive_link)

### W19 : : Improvement ver1 with faster serial communication

Quand on augmente le baud rate cela améliore le signal A donc le sinus Mais améliore pas le cosinus donc le signal B. Nous ne savons pas expliquer ce qu'il se passe. Cependant le résultat n'est pas satisfaisant car on souhaite deux courbes bien tracées.



Quand on utilise les STAMPS pour voir si cela s'exécute plus rapidement on a

```

252
253 if(cntEchantillon == 100){
254     ITM_Port32(31) = 1;
255     SERIAL_SendToPlot(dataA, dataB, cntEchantillon);
256     ITM_Port32(31) = 2;
257 }
258
259

```

Index	Type	Data	Cycles	Time(s)	Extra info
0	Sync	48	3999998	41.666646 ...	
1	ITM Port 31	1	9549336	99.472250 ...	
2	Sync	48	21549330	224.472188...	
3	ITM Port 31	2	22082428	230.025292...	
4	Sync	48	38082420	396.691875...	
5	Sync	48	54082412	563.358458...	
6	Sync	48	72082403	750.858365...	
7	Sync	48	89993395	917.524949...	

L'affichage prend 131 ms ce qui est bien mieux que 2s avant. De plus même si c'est supérieur à 1 ms étant donné que l'on calcule d'abord les 50 valeurs avant de les envoyer cela est déjà mieux que de faire point par point.

### Preparation 20: Very basic DSP at 10kHz sampling

Le signal à générer est de cette forme :  $A \cdot \sin(2 \cdot \pi \cdot F \cdot t)$ . La fréquence d'échantillonnage est de 10 kHz. Le taux d'échantillonnage est de 10kHz. Dans notre test, nous envoyons 50 échantillons à la fois.

Si F est de 200Hz, combien de périodes du signal devrions-nous voir avec 50 échantillons ?

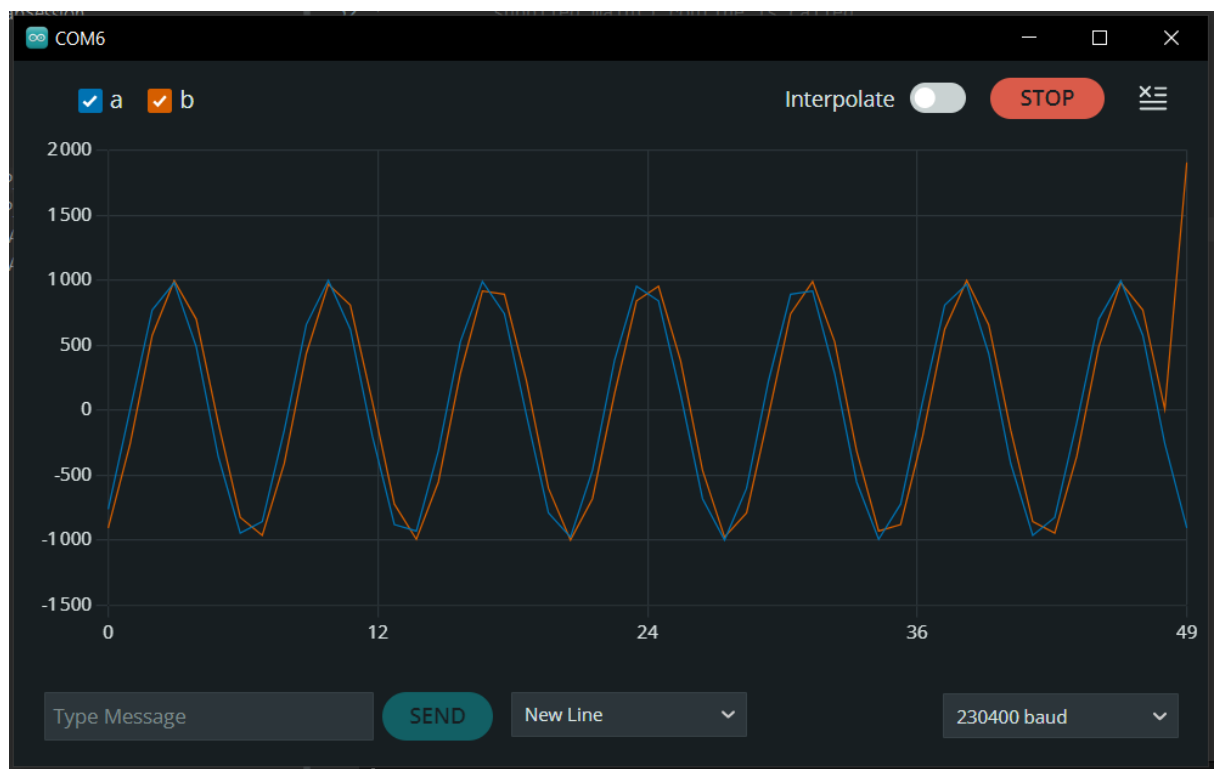
Si F est de 500Hz, combien de périodes du signal devrions-nous voir avec 50 échantillons ?

Identique à la préparation 15 :

- 1) F = 500Hz  
 $T = 1/F = 2\text{ms}$   
 $0,1\text{ms/échantillon} \cdot 50 \text{ échantillons} = 5\text{ms}$   
 Nombre de périodes =  $5\text{ms}/2\text{ms} = \underline{\underline{2,5 \text{ périodes}}}$
- 2) F = 200Hz  
 $T = 1/F = 5\text{ms}$   
 $0,1\text{ms/échantillon} \cdot 50 \text{ échantillons} = 5\text{ms}$   
 Nombre de périodes =  $5\text{ms}/5\text{ms} = \underline{\underline{1 \text{ période}}}$

### W21 : Plotting sine signals (10kHz sampling)

Après avoir réglé la fréquence dans l'IOC ainsi que celle du sinus et cosinus on trace dans le serial plotter A et B.



On remarque que les deux signaux sont bien mieux ils sont encore segmentés mais déjà moins qu'avant et commencent à avoir la forme d'un sinus et cosinus. Ce résultat reste

perfectible mais on voit bien que le problème de sous-échantillonnage commence à être résolu.

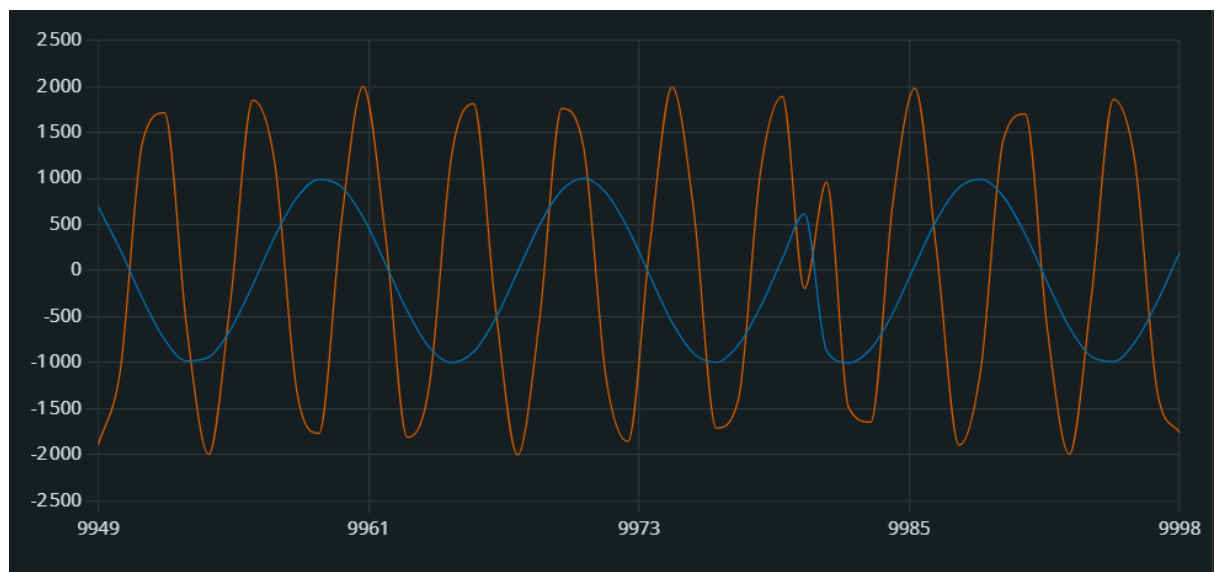
### W22 : Improvement ver2 with better implementation strategy

C'est actuellement la méthode utilisée pour envoyer les données. Remplir les tableaux de data, Envoyer une fois pleins  
Attendre une update du tableau.  
Ceci augmente grandement les performances puisque l'on envoie qu'une seule fois les données dans le plot.

### W23 : Limits of ver2

On pourrait augmenter la taille afin d'avoir plus de données. Cependant cela ajoute du traitement car il faut calculer chaque point. Puisque le trigger est de 1ms si on a 50 points on a 50ms avant affichage. Donc si on est prêt à attendre 1 secondes on peut mettre 1000 points....

Ceci a amélioré la résolution mais pas totalement la forme du sinus et cosinus ce qui est un peu décevant. Nous n'avons pas eu le temps de corriger tous les soucis...





## Preparation 24: ADC pins

Choose one Zio connector: CN7, CN8, CN0 or CN10. What are the pins that can be used on your chosen connector to acquire analog data?

D'après la fiche technique relative à l'utilisation des broches dans le user manuel, on choisit le connecteur ZIO CN9.

Les PINS que l'on peut utiliser pour recevoir les signaux analogique sont les pins: PA3, PC0, PC3\_C, PB1, PC2\_C, ou PB9 et PB8.

On choisit donc les pins PA3 et PC0. Les ADC correspondant sont respectivement ADC1, ADC2 et ADC3 pour PC0. Les channels correspondants sont le 15 pour PA3 et le 10 pour PC0.

**Table 17. NUCLEO-H745ZI-Q and NUCLEO-H755ZI-Q pin assignments (continued)**

Connector	Pin	Pin name	Signal name	STM32H7 pin	Function	Remark
CN9	1	A0	ADC	PA3	ADC_12_INP15	ARDUINO® support
	3	A1	ADC	PC0	ADC_123_INP10	ARDUINO® support
	5	A2	ADC	PC3_C	ADC_3_INP1	ARDUINO® support
	7	A3	ADC	PB1	ADC_12_INP5	ARDUINO® support
	9	A4	ADC	PC2_C or PB9	ADC3_INP0 (PC2_C) or I2C1_SDA (PB9)	ARDUINO® support
	11	A5	ADC	PF11 or PB8	ADC1_INP2 (PF11) or I2C1_SCL (PB8)	ARDUINO® support
	13	D72	COMP1_INP	PB2 <sup>(4)</sup>	COMP	NC by default
	15	D71	COMP2_INP	PE9	COMP	-
	17	D70	I2C_B_SMB	PB5 <sup>(5)</sup>	I2C_2	NC by default
	19	D69	I2C_B_SCL	PF14	I2C_2	-
	21	D68	I2C_B_SDA	PF15	I2C_2	-
	23	GND	GND	-	ground	-
	25	D67	CAN_RX	PD0	CAN_1	-
	27	D66	CAN_TX	PD1	CAN_1	-
	29	D65	I/O	PR14 <sup>(6)</sup>	I/O	-