

TP électronique

Microcontrôleur STM32L152

UART

Chapitre 1	Introduction	p 3
Chapitre 2	Hello world. Communication avec TerraTerm	p 5
Chapitre 3	Utilisation printf et Data Trace dans STM32CubeIDE	p 7

Chapitre 1. Introduction

L'objectif de ce TP est d'envoyer des informations depuis le microcontrôleur vers le PC. Quand on utilise les outils de développement en debug mode, les informations sont envoyées de manière analogue du microcontrôleur vers le PC. Ceci est envoyé par le bus UART.

Vous pouvez utiliser l'émulateur que vous souhaitez. Nous recommandons d'utiliser Tera Term qui est open source et téléchargeable facilement sur internet. Par exemple sur ce site web : <https://tera-term.en.lo4d.com/windows>

Le bus UART est asynchrone, c'est-à-dire sans horloge. Pour communiquer, le maître et l'esclave doivent donc être en accord sur la configuration : baud rate, parité, stop bit...

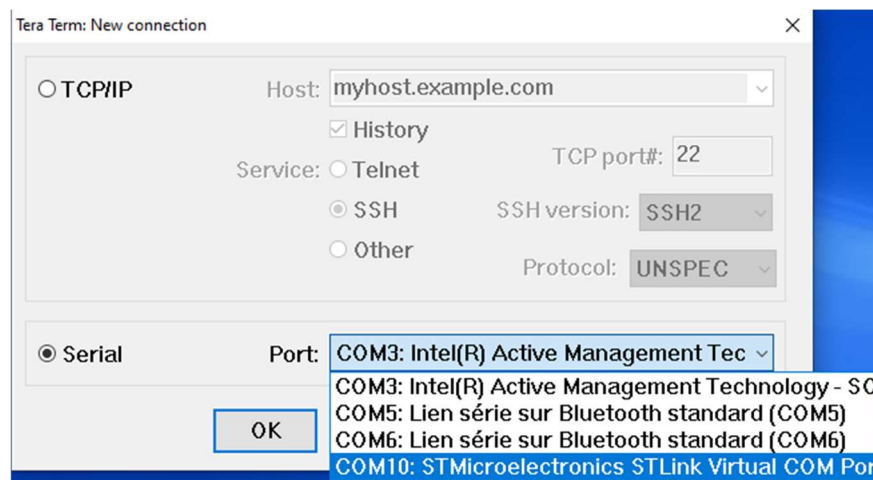
L'UART correspond au port COM sur votre PC. Brancher le Nucleo board et noter sur quel port le Nucleo est connecté.

Dans l'exemple ci-dessous, le board est relié au port COM10.

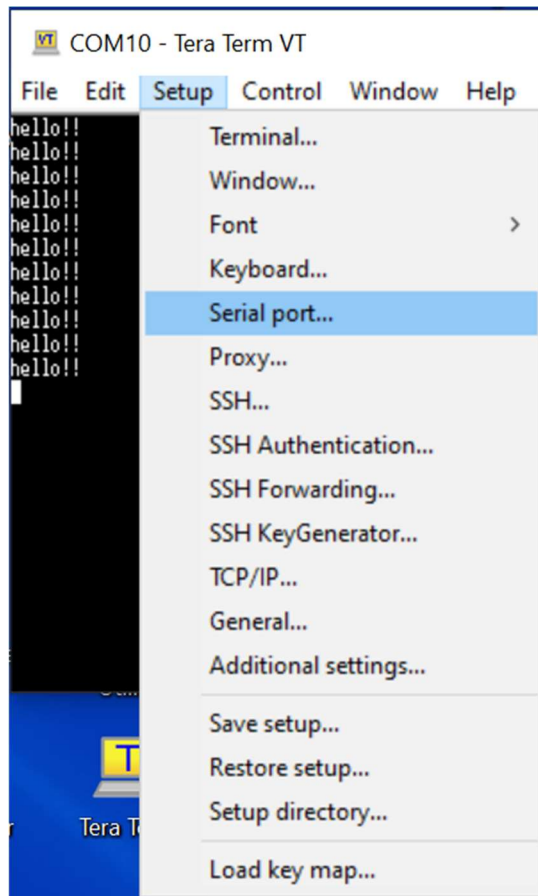
Dans le gestionnaire de périphérique, vérifier



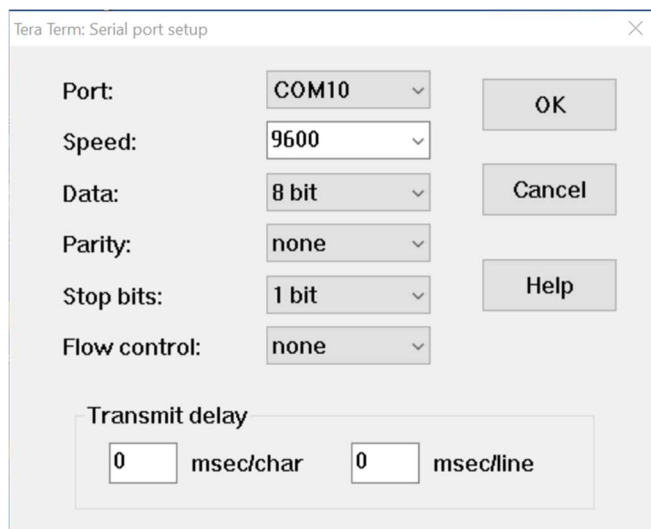
Installer l'application Tera Term, cliquer sur Serial et sélectionner le port COM qui a été lu précédemment.



Cliquer sur Setup/Serial Port .



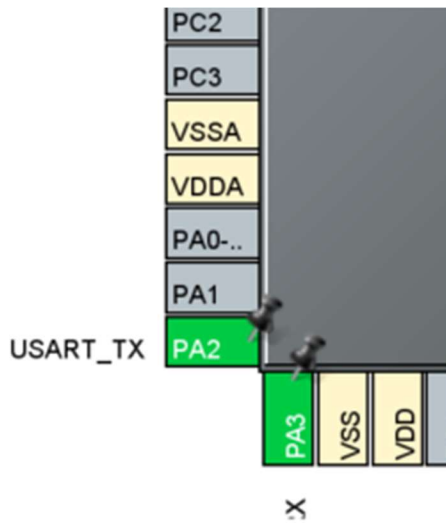
Entrer la configuration suivante. Cette configuration sera utilisée pour programmer le STM32.



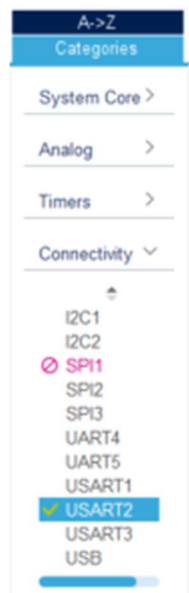
Chapitre 2. Hello

Créer un nouveau projet avec STM32CubeIDE.

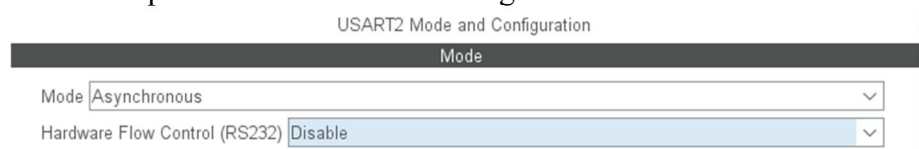
Quand un projet est créé avec le Nucleo-L152, les broches UART utilisées pour communiquer avec l'USB sont configurées correctement. Dans le le pinOut View, les broches sont sélectionnées en vert.

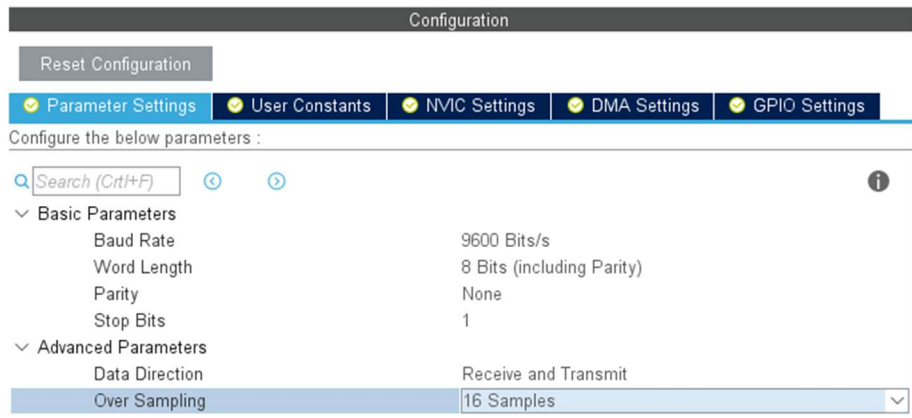


Dans catégorie, sélectionner connectivity et l'USART2



Communiquer l'USART avec la configuration ci-dessous.





Générer le code et écrire le code suivant

Déclarer la variable suivante

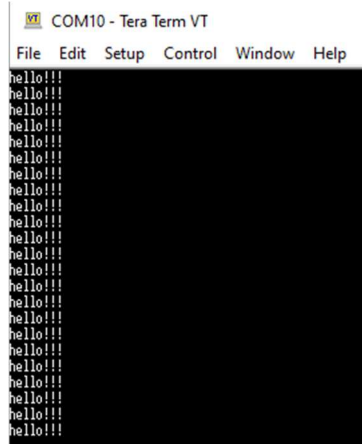
```
uint8_t bufftx[10] = "hello!!!\n\r";
```

Ce buffer sera transmis toutes les 200ms

```
/* USER CODE BEGIN WHILE */
while (1)
{
    HAL_UART_Transmit(&huart2, bufftx, 10, 100);
    HAL_Delay(200);
}
/* USER CODE END WHILE */
```

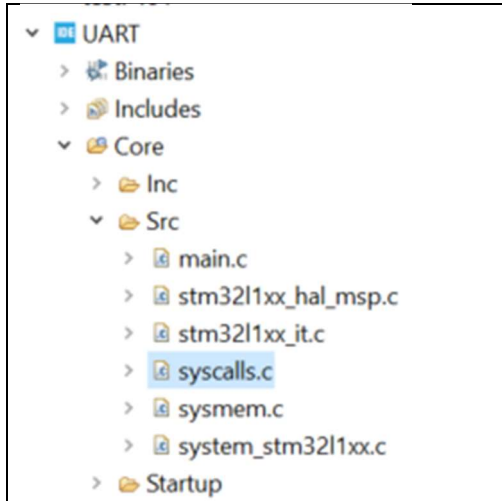
Compiler et lancer le code.

Dans la fenêtre de Tera Term, vous voyez le message apparaître.



Chapitre 3 Utilisation printf et Data Trace dans STM32CubeIDE

Dans le fichier syscalls.c, repérer la fonction `_write`. Mettre en commentaire la fonction `__io_putchar` et la remplacer par la fonction `ITM_SendChar` comme montré ci-dessous.

	<pre>__attribute__((weak)) int _write(int file, char *ptr, int len) { // int i; // for (i=0; i<len; i++) // ITM_SendChar(*ptr++); int DataIdx; for (DataIdx = 0; DataIdx < len; DataIdx++) { //__io_putchar(*ptr++); ITM_SendChar(*ptr++); } return len; }</pre>
---	--

Dans le fichier syscalls.c, ajouter l'inclusion `#include « stm3211xx.h »`.

```
/* Includes */
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>
#include <sys/times.h>
#include "stm3211xx.h"
```

Grâce aux includes définis dans ce fichier, la fonction `ITM_SendChar` définie dans le fichier `core-cm3.h` sera reconnue

Pour information, la fonction `ITM_SendChar` est codée ci-dessous. Il n'y a pas d'utilité dans ce TP à comprendre ce code.

```
/**
 *brief  ITM Send Character
 *details Transmits a character via the ITM channel 0, and
 *        \li Just returns when no debugger is connected that has booked the output.
 *        \li Is blocking when a debugger is connected, but the previous character sent has not been transmitted.
 *param [in]  ch Character to transmit.
 *returns    Character to transmit.
 */
STATIC_INLINE uint32_t ITM_SendChar (uint32_t ch)
{
    if (((ITM->TCR & ITM_TCR_ITMENA_Msk) != 0UL) && /* ITM enabled */
        ((ITM->TER & 1UL) != 0UL) ) /* ITM Port #0 enabled */
    {
        while (ITM->PORT[0U].u32 == 0UL)
        {
            __NOP();
        }
        ITM->PORT[0U].u8 = (uint8_t)ch;
    }
    return (ch);
}
```

On peut simplement remarquer la ligne ITM->PORT[0U] signifie que le bit 0 du port ITM est utilisé. C'est la raison pour laquelle nous utiliserons le bit 0 de ITM stimulus plus tard dans le TP. Cette explication permet de démystifier la configuration de l'IDE qui sera faite plus tard.

Dans le main.c, écrire le programme ci-dessous

```
/* Includes -----
#include "main.h"
#include "stdio.h"

/* Private includes -----
/* USER CODE BEGIN PV */

uint8_t count = 0;

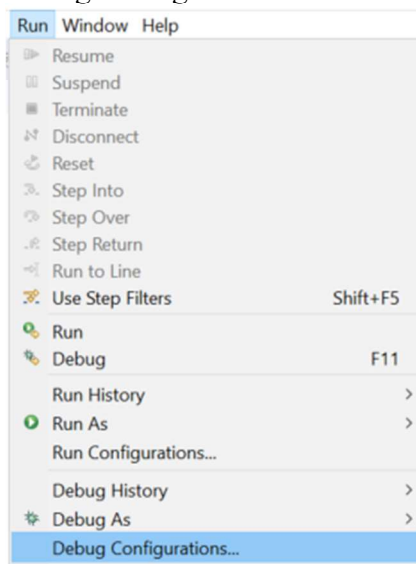
/* USER CODE END PV */

while (1)
{
    printf("hello count = %d \n ", count);
    count++;
    HAL_Delay(200);

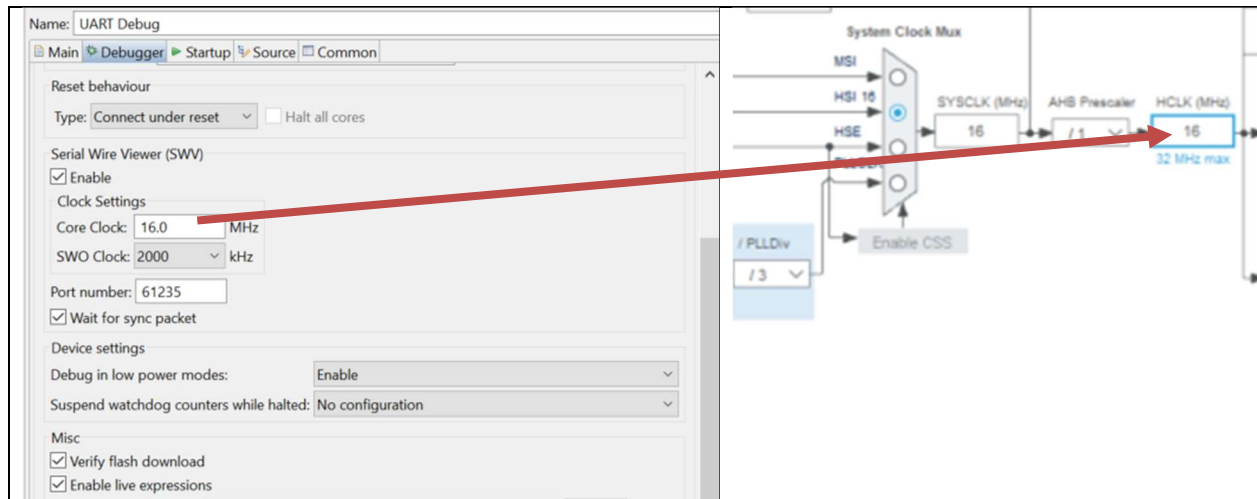
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Pour configurer le mode debug et obtenir les informations désirées, cliquer sur le menu Run et Debug Configurations

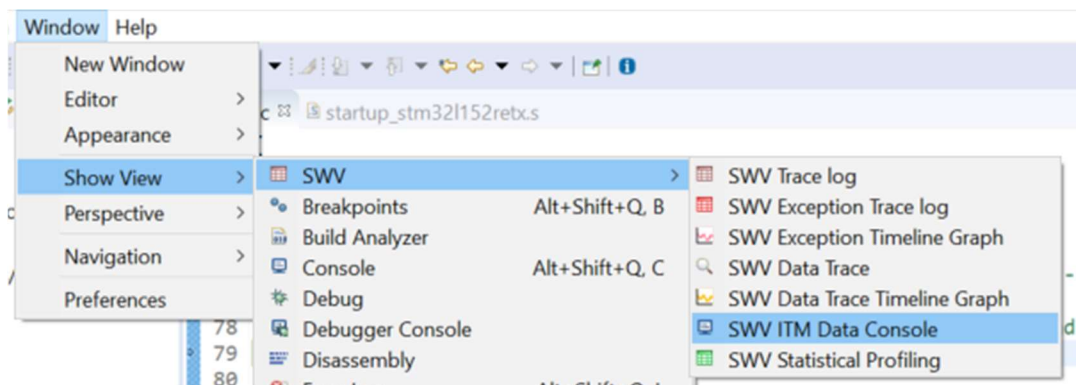


Dans la tabulation Debugger, Enable SWV et mettre l'horloge à la même valeur que celle prgrammée dans le cœur



Lancer le programme en mode debug

Ouvrir les fenêtres SWV ITM Data Console et SWV Data Trace Timeline Graph

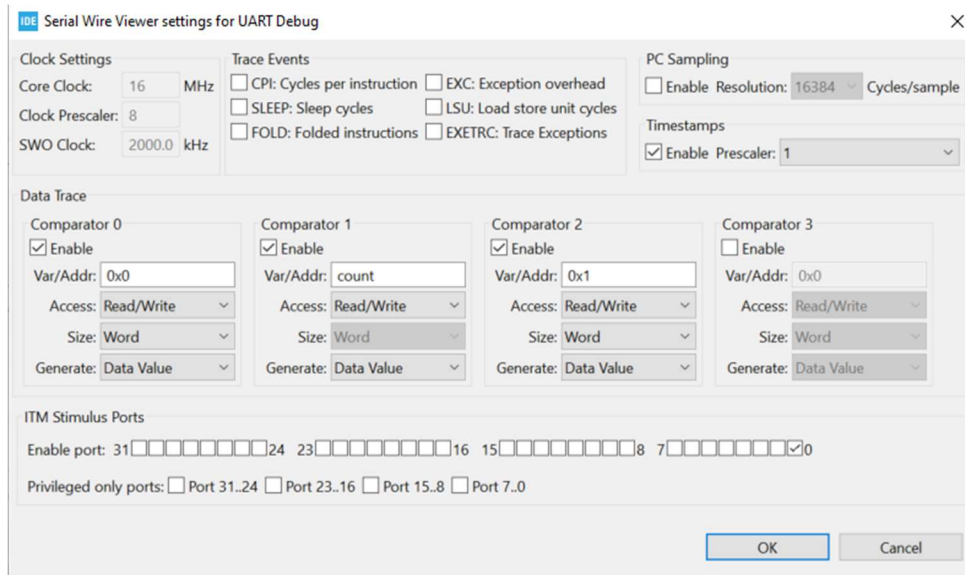


Dans la fenêtre SWV ITM Data Console, cliquer sur Configure Trace



Configurer comme indiqué ci-dessous. Sélectionner bit0 de ITM Stimulus Port.

Avec cette configuration, nous pourrons lire le printf dans le fenêtre SWV ITM Data Console et un graphe de count dans SWV Data Trace Timeline Graph

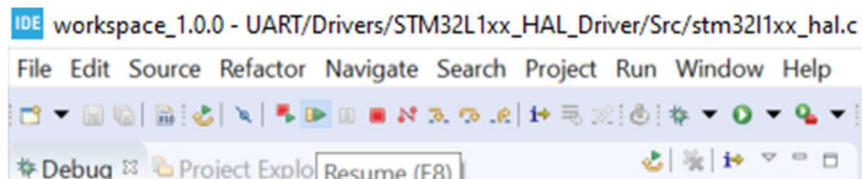


Cliquer sur OK

Puis cliquer sur Start Trace pour commencer à enregistrer les données



Lancer le programme avec l'icône 'resume'



Les valeurs s'affichent bien dans les 2 fenêtres.

Port 0

```
hello count = 8
hello count = 9
hello count = 10
hello count = 11
hello count = 12
hello count = 13
hello count = 14
hello count = 15
hello count = 16
hello count = 17
hello count = 18
hello count = 19
hello count = 20
hello count = 21
hello count = 22
hello count = 23
hello count = 24
hello count = 25
hello count = 26
hello count = 27
```

