TP 4 : UART

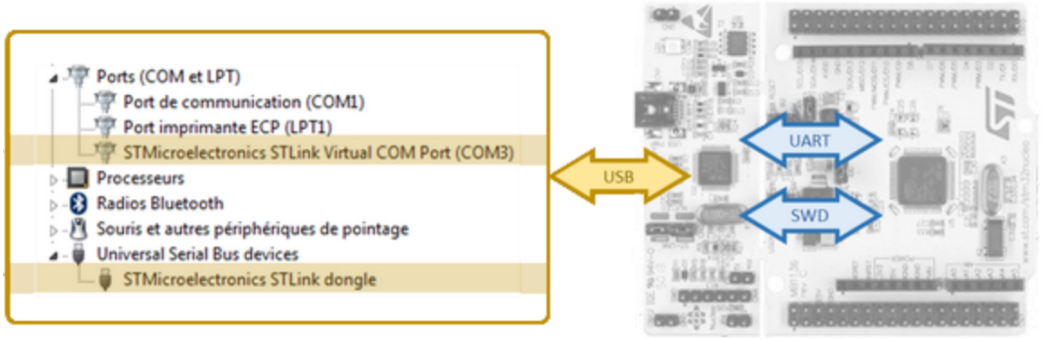
Objectifs :

-Compréhension et mise en œuvre d'un bus de communication série
-Manipulation de périphérique de type UART

Introduction

Dans le cadre de ce TP, on souhaite ajouter de la connectivité à notre microcontrôleur STM32F446RE. Pour cela, il dispose d'un grand nombre de broche d'E/S dont chacune peut être accédée par un périphérique de gestion. Dans notre cas, nous allons mettre en œuvre une liaison série via le contrôleur UART (*Universal Asynchronous Receiver Transmitter*), permettant de transmettre des octets de données sous forme de caractères. L'environnement de programmation et de debug reste STM32CubeIDE.

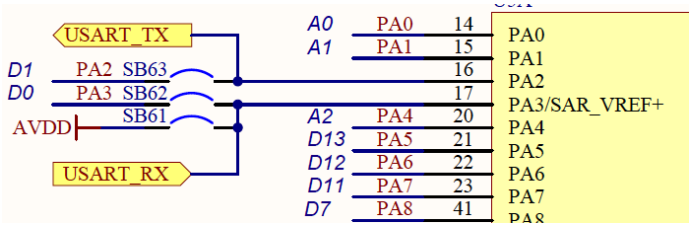
Lorsque vous ouvrez sous Windows votre gestionnaire de périphérique, une fois la carte STM32 connectée en USB, vous pouvez observer ceci :



Le STLink dongle est le device utilisé pour programmer et débugger le microcontrôleur STM32 via l'USB. Le second, est un port **COM** virtuel (ici en l'occurrence COM3 (ssur l'image et sous Windows)) utilisé pour échanger les données utilisateur entre le PC et le STM32.
Généralement, sur le PC, on utilise un programme comme Putty ou encore Tera Term pour communiquer facilement sur ce port sur Linux/Windows. Sur Mac, vous pouvez CoolTerm qui est très bien.

Sur notre carte STM32F446RE, ce port COM virtuel est connecté en UART via les broches de l'USART2 (Universal Synchronous Asynchronous Receiver Transmitter) du microcontrôleur. A noter qu'un USART support le mode synchrone, utilisant un fil supplémentaire d'horloge, ainsi que le mode asynchrone (sans horloge).

D'après [2] et [3], c'est l'USART2 et les broches PA2 (TX) et PA3 (RX) qui sont connectées au port COM. C'est donc le port GPIOA qu'il faudra configurer pour pouvoir activer les fonctions alternatives et l'USART2 (cf. page 192 de [1]).



Pour chaque GPIO (A-I), la liste des fonctions alternatives est disponible dans [3] à partir de la page 57.

Exercice 1 : Transmission de données STM32->PC

Nous allons chercher à pouvoir mettre en œuvre une liaison UART entre le PC et la carte SMT32F446RE. Nous allons donc nous baser sur l'USART2. Les caractères émis/reçus sur cette liaison pourront être simplement visualisés via un émulateur de port COM depuis votre PC car ces deux broches sont redirigées sur l'USB.

-Créer un nouveau projet STM32CubeIDE.
-A travers la configuration graphique, ajouter l'USART2 et configurer le :
    -Mode asynchrone

Q : A quoi correspond le paramètre baud rate ? Qu'est ce que cela implique au niveau du récepteur (ici le PC) ?
A quoi sert l'activation de l'option 'Parity' ?

-Après génération du code, expliquer les paramètres suivants de la structure UART_HandleTypeDef associée à huart2 au niveau de la configuration de l'USART2 (MX_USART2_UART_Init()) :

-Ouvrir le fichier **stm32f4xx_hal_msp.c** et analyser la fonction **HAL_UART_MspInit()** ;
Q : Qu'est-ce qui est réalisé ? En quoi est-ce important ? Expliquer chaque ligne de code

-Dans la fonction main, en faisant appel aux fonctions de l'uart, modifier votre programme pour :
    -Envoyer la chaîne suivante : "Hello CY !"
    -Puis de permettre un affichage des nombres 0 à 9 s'incrémentant toutes les secondes

-Tester sur la carte ; Pour visualiser la communication entre la carte STM32 et votre PC, il suffit d'utiliser un utilitaire Putty, Tera Term ou encore CoolTerm pour Mac)
-Faire valider par l'enseignant

-Consigner vos développements dans votre rapport

Exercice 2 : Réception de données PC->STM32 ; écho

On souhaite implémenter une fonction d'écho. A chaque réception d'un caractère sur la liaison série RX, on renverra la donnée sur la voie TX.

-Créer un nouveau projet STM32CubeIDE et appelé le TP4_ex2.c
-A partir du code développé dans l'exercice 1, utiliser l'utilitaire graphique pour générer le code d'initialisation de l'USART2
-Modifier le programme principal pour pouvoir réaliser l'écho
-Tester et faire valider par l'enseignant
-Consigner vos développements dans votre rapport

Faites ensuite évoluer votre programme pour ne plus utiliser la scrutation mais bien le mécanisme d'interruption.
-Modifier la configuration via l'outil graphique
-Modifier le code nécessaire
-Tester et faire valider par l'enseignant
-Consigner vos développements dans votre rapport

Exercice 3 : Le nombre mystère

On souhaite développer un petit jeu interactif via l'USART2. L'objectif est de faire deviner un nombre mystère, tiré aléatoirement et compris entre 0 et 9. Le programme sur le microcontrôleur devra inviter le joueur à saisir une valeur. Celle-ci sera analysée et une indication sera fournie pour le programme, comme par exemple "Le chiffre est supérieur" ou "Le chiffre est inférieur". La partie s'arrête lorsque le joueur a trouvé le bon chiffre. Le programme retourne alors un message de fin de partie indiquant le nombre de tentative. Le joueur rentrera les valeurs en tapant le chiffre via un des émulateurs de port série que vous utilisez (Putty, Tera Term ou encore CoolTerm).

A noter que la solution devra forcément mettre en œuvre les interruptions
-Créer un nouveau projet STM32CubeIDE et appelé le TP4_ex3.c
-Modifier la configuration via l'outil graphique
-Modifier le code nécessaire
-Tester et faire valider par l'enseignant
-Consigner vos développements dans votre rapport

Exercice 4 : Fonction printf

On souhaite disposer d'une fonction printf redirigeant les caractères vers l'UART et non la sortie standard console en C.
Pour cela, nous allons nous devoir redéfinir la fonction l'implémentation de la fonction __io_putchar(int ch) ou fputc en fonction des .

-Copier le code suivant dans la section en dessous de /* USER CODE BEGIN 0*/:

```
#ifdef __GNUC__
/* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small
printf
   set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */
```

- Ajouter ensuite le code liée à l'implémentation de cette fonction

```
PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the USART2 and Loop until the end of
transmission */
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
}
```

-Modifier le fichier main.c afin d'y tester la fonction printf().
-Ouvrir un terminal sur TeraTerm et valider le fonctionnement.
-Consigner vos développements dans votre rapport.

-Faire le test avec un nombre flottant ; Par défaut, ce type ne devait pas être supporté.
-Rajouter le flag –u_printf_float dans les propriétés de votre projet puis dans C/C++Build -> Settings -> Tool Settings -> MCU GCC Linker -> Miscellaneous.

-Re-tester maintenant votre fonction printf avec des flottants.

Avoir la notion du temps :

Selon vous, si l'on considère une vitesse de transmission de l'UART de 115200 bauds. Combien de temps cela prendrait pour transmettre le message "Hello Console\r\n" ?

Si vos calculs sont bons, vous devez obtenir environ 1.4 ms ! Cela est très long en comparaison de notre processeur qui possède une horloge système de 84MHz (environ 12ns de période), soit un rapport supérieur à 100 000.
Ayez donc toujours en tête qu'utilisez la liaison série prend du temps, qu'il faut éviter les messages trop longs et privilégier l'utilisation du printf lors du debug uniquement et non dans l'application finale !!
Heureusement, il existe des mécanismes pour permettre d'optimiser l'utilisation de l'UART, notamment le périphérique DMA (Direct Access Memory) qui permet d'effectuer les transferts mémoires vers l'UART sans intervention du processeur (uniquement la configuration initiale), qui peut s'attacher à d'autres tâches.

Conclusion

A travers ce TP, vous avez pu découvrir la programmation de périphériques pour la communication série vers le monde extérieur. Vous avez aussi mis en œuvre le mécanisme fondamental d'interruption dans ce contexte.

Références

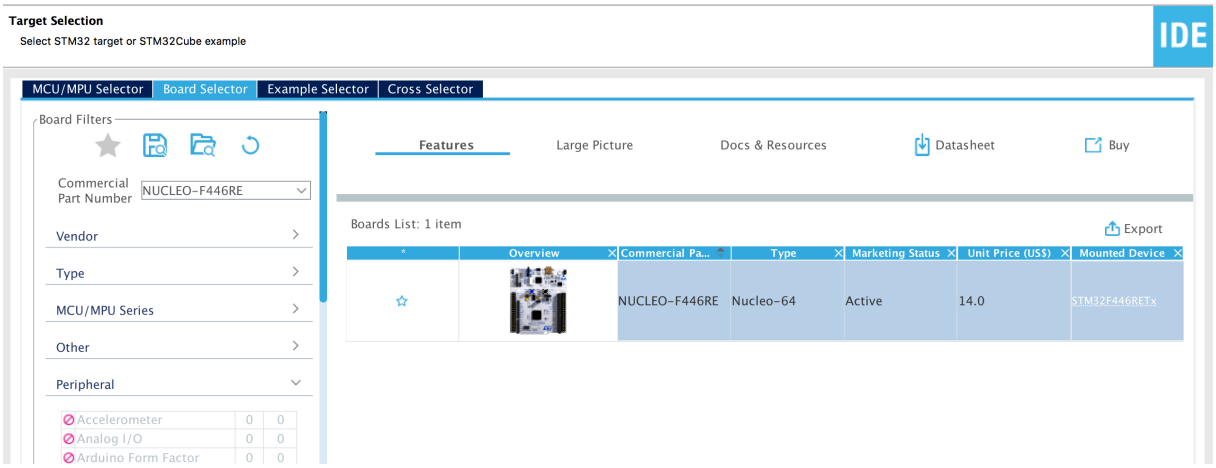
[1]STMicroelectronics, '*STM32F446xx advanced ARM –based 32-bit MCUs*', RM0390 Reference Manual revision 4 February 2018, https://www.st.com/resource/en/reference_manual/dm00135183-stm32f446xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[2] STMicroelectronics, 'STM32 Nucleo-64 boards (MB1136)', UM1724 User Manual rev 14, Aug. 2020, https://www.st.com/resource/en/user_manual/dm00105823-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf

[3] STMicroelectronics, 'STM32F446xC/E, Datasheet DS10693 Rev 8, July 2020, https://www.st.com/resource/en/datasheet/stm32f446mc.pdf

ANNEXE 1 : Création d'un projet STM32CubeIDE (sans configuration graphique) en C

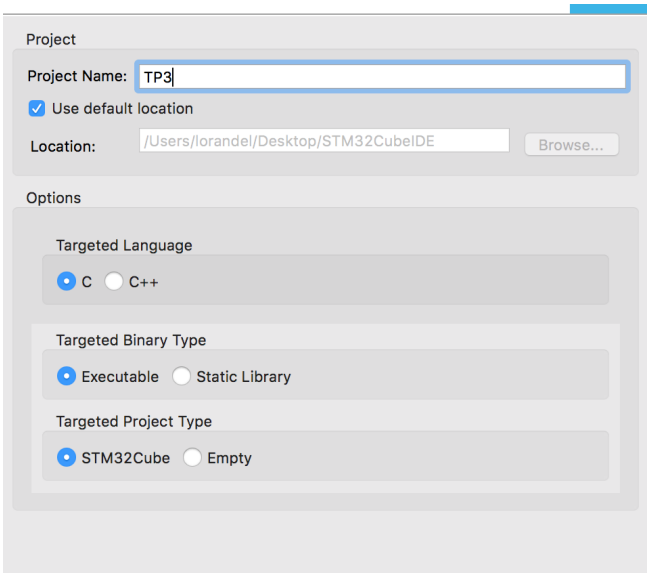
Après avoir lancé STM32CubeIDE, dans l'onglet **File** > **New** > **STM32 Project**, sélectionner la bonne plateforme de développement, via l'onglet **Board Selector.** Pour cela, à l'aide du filtre, tapez **NUCLEO-F446RE** puis **Next**.



Une fenêtre « STM32 Project s'ouvre. Nommer simplement votre projet, et pour l'option « **Targeted Project Type** » choisir soit :

-Empty -> Projet simple sans configuration graphique via STM32CubeMX

-STM32Cube -> si génération d'un projet avec configuration graphique du microcontrôleur ; <u>Choisir cette option par défaut.</u>



Puis **Finish**.

L'utilitaire de création vous demandera si vous souhaitez initialiser les périphériques à leur mode par défaut, choisir **non**.