Rappels de microcontrôleurs

L'objectif de ce TP d'introduction est double :

D'une part vous familiariser avec la couche d'abstraction matérielle (HAL pour *Hardware Abstraction Layer*).
Il s'agit d'un ensemble de fonctions permettant d'accéder aux périphériques sans avoir à manipuler les registres. Ça permet, en principe, de rendre le code plus facile à écrire, plus lisible, et surtout plus portable.
Vous constaterez néanmoins que pour programmer les périphériques, il faut quand même comprendre leur fonctionnement.

D'autre part, l'objectif est de vous apprendre à lire cette @&#$! de documentation (ces @&#$!s de documentations, pour être précis).

Ce TP est prévu pour la version **1.16.0** de STM32CubeIDE.

1 Premiers pas

Prenez de bonnes habitudes : créez un dépot git, et travaillez dedans.

Créer un projet pour la carte NUCLEO-G431RB. Une pop-up s'ouvre, vous pouvez accepter après avoir compris de quoi il s'agit.

1. Certaines broches sont en rouge, d'autres en orange. Pourquoi ?

   Une fois le projet créé, dans l'onglet "Pinout & Configuration" :
   — Configurer le quartz de l'horloge :
   `System Core > RCC > High Speed Clock (HSE) = Crystal/Ceramic Resonator`
   — Configurer le système de programmation et de débug :
   `System Core > SYS > Debug = Serial Wire`
   — Notez les modifications du pinout
   Configurer ensuite l'horloge du microcontrôleur. Pour cela, aller dans l'onglet "Clock Configuration". Mettre HCLK à 170MHz **en utilisant l'oscillateur externe HSE.**

   Enfin, configurer le projet pour éviter d'avoir un `main.c` de 14km de long. Dans l'onglet Projet Manager :
   — Dans "Code Generator", cliquez sur "Generate peripheral initialization as a pair of '.c/.h' files peripheral"

   Sauvegarder et générer le code.
2. Où se situe le fichier `main.c` ?

3. À quoi servent les commentaires indiquant `BEGIN` et `END` ?

On pourrait compiler et programmer la carte, mais il ne se passerait rien de très palpitant.
Pour rendre ce TP plus excitant, faisons clignoter une LED ! Quelques exemples de fonctions utiles :
— `HAL_Delay`
— `BSP_LED_On`
— `BSP_LED_Off`
— `BSP_LED_Toggle`
Historiquement (l'an dernier) les fonctions suivantes étaient utilisées :
— `HAL_GPIO_WritePin`
— `HAL_GPIO_TogglePin`

4. Quels sont les paramètres à passer à `HAL_Delay` et `BSP_LED_XXX` ? Vous pouvez accéder au code d'une fonction en maintenant la touche `<Ctrl>` et en cliquant dessus.
5. Dans quel fichier les ports d'entrée/sorties sont-ils définis ?
6. Écrire un programme permettant de faire clignoter la LED.
7. Modifier le programme pour que la LED s'allume lorsque le bouton USER est appuyé.

2 Avec des interruptions

Dans cette section nous verrons comment :
— Utiliser un timer pour faire clignoter les LED,
— Configurer le GPIO pour déclencher une interruption lors de l'appui sur un bouton.
Nous utiliserons, disons, le timer TIM2.

1. Rappeler les étapes nécessaires pour configurer un timer.
2. Sur quel bus TIM2 est-il connecté ? Pourquoi cette information est-elle nécessaire ? (Bon, en vrai, avec ce modèle précis on s'en fout, mais répondez à la question quand même, s'il vous plait).
3. Comment configurer TIM2 pour déclencher une interruption 10 fois par secondes ? Pourquoi peut-on laisser le prescaler à 0 ?
4. Dans quel fichier trouve-t-on les serveurs d'interruption ? Quel est le nom du serveur d'interruption de TIM2 ?
5. Écrire le code du serveur d'interruption permettant de changer l'état de la LED à chaque interruption.
6. Quelle fonction permet de démarrer le timer en mode interruption ? Où cette fonction doit-elle être appelée ? Que doit-t-on lui passer comme paramètres ?
7. Démarrer le timer en mode interruption. Compiler, programmer, s'extasier.

8. Que se passe-t-il si l'on supprime la ligne `HAL_TIM_IRQHandler(&htim2);` du serveur d'interruptions de TIM2 ?

Avec du code dans le fichier définissant les vecteurs d'interruptions, notre code devient moins facile à lire, et donc à maintenir. La HAL STM32 fournit des fonctions de *callback*, c'est à dire des fonctions appelées automatiquement lors d'une interruption. On peut redéfinir ces fonctions n'importe où, et donc mieux organiser notre code.

9. Quel est le nom de la fonction callback du timer ? Déplacer le code dans le fichier `main.c`

Nous allons maintenant passer à l'utilisation du bouton poussoir par interruption.

10. Configurer l'entrée du bouton USER pour déclencher une interruption.
11. Changer la fréquence de clignottement de la LED lors de l'appui sur le bouton USER. Les interruptions EXTI seront utilisées.

3 La liaison série

Le port USB par lequel nous programmons la carte permet également – entre autres – d'émuler une liaison série. Plus besoin de vieux câble RS232. En plus, la liaison série est déjà configurée !

1. Cherchez la documentation de la carte ainsi que celle du microcontrôleur.
2. Quel périphérique est utilisé ? Quelles sont les broches concernées ? Quel est le baudrate de la ligne ?

Attention, bien que le périphérique s'appelle USART2 (oups, c'est la réponse à la question précédente...), il faut utiliser le driver UART et pas USART.
Ne me demandez pas pourquoi. (Edit 2024 : Maintenant je sais, demandez-moi)

3. Concevoir, sans interruptions, un programme renvoyant chaque caractère qu'il reçoit.

Mettons les interrutions de côté pour l'instant. À la place nous allons construire un mini-shell qui nous aidera par la suite.
(C'est un moyen de vous occuper, c'est assez complexe, on passera plusieurs heures dessus plus tard)

4. Proposer une solution pour que la fonction printf utilise notre liaison série.

Pour la suite, travailler dans un nouveau fichier (shell.c par exemple).

5. Concevoir un shell permettant d'appeler des fonctions préalablement enregistrées.