Université Grenoble Alpes
L3 Informatique

Année 2024-2025

Architectures logicielles et matérielles
Semaine 3 TP (2)

Cette première séance de programmation sur carte STM32F4 est consacrée à la prise en mains de l'environnement, et à quelques opérations de base sur les ports GPIO.

Rappels et informations préliminaires.

• On rappelle que les 4 **leds** sont connectées au port **GPIOD**. Plus précisément : la led verte est connectée au pin 12 de ce port, la led orange est connectée au pin 13, la led rouge est connectée au pin 14, et la led bleue est connectée au pin 15.

• Nous allons cross-compiler puis charger l'application sur la carte (en ROM) et, par l'intermédiaire de OpenOCD (http://openocd.org/), être en exécution de l'application dans gdb.
Pour mémoire, quelques commandes utiles de **gdb** (voir par exemple https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf) :
b *(ou break) : positionne un breakpoint dans le programme*
c *(ou continue) : continue l'exécution du programme jusqu'au prochain breakpoint*
n *(ou next) : passe à la ligne suivante dans le source, sans descendre dans les fonctions*
s *(ou step) : même chose, mais en entrant dans les fonctions appelées*
x /x *: affiche le contenu de la mémoire à une adresse donnée, en hexadécimal*
set *: affecte une valeur au contenu de la mémoire à une adresse donnée*

• Nous utiliserons la bibliothèque *libopencm3* pour nos TP (http://libopencm3.org/docs/latest/stm32f4/html/modules.html).
La fonction ***gpio_mode_setup*** de cette bibliothèque est utilisée pour la **configuration** des ports GPIO.

Question 1.
Récupérer le fichier TP_semaine3.tar dans le moodle et placez-le dans le répertoire de votre choix. Désarchiver le contenu. Il contient une petite application pour ces premiers TP, située dans les fichiers *firmware.c* et *bp_leds.c*
Nous allons compléter les fonctions qui permettent d'allumer les **leds**.

Observer le contenu de la fonction *config_leds_pins* dans le fichier *bp_leds.c*. **Pourquoi** l'appel à cette fonction est-il indispensable ? (cf information donnée plus haut)

Vérifier que, dans le fichier *firmware.c*, toutes les lignes de `sortie_int` et `main` sont commentées sauf :

```
void sortie_int(uint16_t v) {
   write_leds_int(v);
}

int main (void) {
   coldstart();
   while (1) {
      sortie_int(3);
   }
   return 0;
}
```

Et de plus décommenter l'appel à la fonction *config_leds_pins*

• Compléter le code de la fonction ***write_leds_int*** dans le fichier *bp_leds.c* avec la fonction développée précédemment en TD (elle prend en paramètre un entier compris entre 0 et 15 et place cette valeur à l'emplacement voulu dans ODR).

Cross-compiler par : `make`
Puis charger et lancer l'exécution par : `make debug`
Une fois dans gdb, faites simplement `continue` (pas nécessaire de positionner des break-points et d'observer en pas à pas ici).
Vérifier que les leds verte et orange s'allument (**pourquoi** ?).
Bien tester avec d'autres valeurs.

• Remplacer l'appel à *sortie_int(3)* par un appel à *sortie_int(n)*, décommenter l'appel à la fonction *plusplus* (observer son effet sur *n*), compiler et exécuter en plaçant un breakpoint sur *plusplus*. Répéter la boucle en continuant de breakpoint en breakpoint.
Qu'observez-vous exactement lors des itérations successives et pourquoi ?

• Pour finir, commenter l'appel à la fonction *sortie_int* et décommenter l'appel à la fonction *chenillard*, compiler et exécuter.
Assurez-vous de comprendre l'animation lumineuse compte tenu des valeurs utilisées.

Question 2.

La fonction *write_leds_int* est censée ne modifier que les bits associés aux leds voulues, et pas les autres. Nous allons vérifier qu'il en est ainsi.

Replacer l'appel à *sortie_int(n)* et commenter celui à *chenillard*. Décommenter toutes les lignes de la fonction *sortie_int* dans le fichier *firmware.c*, compiler et exécuter en mettant un point d'arrêt sur cette fonction, puis en faisant des `next` afin d'observer la valeur de la variable `val` à chaque itération.

Quelles observations faites-vous en conséquence, et est-ce ce qui est espéré ? (noter les valeurs observées et expliquer pourquoi ces valeurs)

Question 3.

Nous allons maintenant écrire une variante de la fonction ***write_leds_int*** :
Ecrire **une variante** de cette fonction, qui prend en paramètre quatre "flags" qui indiquent respectivement si on veut allumer ou éteindre la led correspondante (0 pour éteindre, 1 pour allumer). Autrement dit cette variante a la signature :

```
void write_leds_int (char green, char orange, char red, char blue);
```

(NB. les flags sont des `char` afin de n'utiliser qu'un octet)

Commenter soigneusement le code de cette variante de la fonction *write_leds_int*
Tester cette nouvelle version en vous inspirant de la question 1.

Question 4.

Le principe utilisé pour écrire la fonction *write_leds_int* procède par lecture-modification-écriture dans le registre ODR. Ceci peut poser des problèmes en cas de partage d'un GPIO entre activités concurrentes.

En alternative, nous allons utiliser le registre **BSRR** (Bit Set/Reset Register). C'est un registre 32 bits (les 16 bits de poids faible pouvant être repérés par BSRRL, et les 16 bits de poids fort par BSRRH).
Ecrire un 1 dans l'un des 16 bits de poids faible met à 1 le pin de numéro correspondant, et écrire un 1 dans l'un des 16 bits de poids fort met à 0 le pin de numéro correspondant.

Expliquer selon quel **principe** vous allez concevoir l'unique affectation de BSRR.

Compléter le code de la fonction ***write_leds_int_bsrr*** dans le fichier *bp_leds.c* : cette fonction doit forcer la valeur des leds de façon atomique, en utilisant BSRR.
Tester en remplaçant l'appel précédent à *sortie_int* par un appel à *sortie_int_bsrr*.