TP 4 Interface utilisateur

Nous proposons ici de concevoir une application complète de clignotement complexe des LEDs de notre carte, permettant à l'utilisateur de choisir indépendemment le motif et la vitesse de clignotement. On partira pour cela du code que vous avez écrit au TP 2, comprenant le driver pour les LEDs et le driver pour le bouton. Pour commencer, ne gardez que la boucle infinie dans le `main()`.

1 Clignotements par timer

1. Dans le `main()`, appelez la fonction `HAL_Init()` pour initialiser le timer `SysTick`. Définissez une fonction `void SysTick_Handler()` qui ne fait pour l'instant qu'allumer une des LEDs. Testez que cette LED s'allume bien.
2. La fonction `SysTick_Handler()` est en fait appelée toutes les millisecondes, interrompant temporairement `main`. Modifiez cette fonction de façon à faire clignoter la LED toutes les 128ms. [1]
3. Codez maintenant les motifs de clignotement suivants. Conservez le code pour chaque motif dans une fonction séparée `clign1(), clign2()` etc., appelée toutes les 128ms par `SysTick_Handler()`.
   (a) toutes les LEDs allumées, puis éteintes, puis allumées...;
   (b) les LEDs paires allumées, puis les impaires, puis les paires...
   (c) les LEDs 0 et 4 allumées, puis les LEDs 1 et 5, puis 2 et 6, puis 3 et 7...
   (d) chacune des LEDs allumées successivement : 0, puis 1, puis 2,...puis 7, puis 0, ...

   On tentera d'écrire le code le plus concis possible, nécessitant le moins de variable d'état (i.e. variables globales ou statiques). Utilisez les opérations bit-à-bit !

2 Détection d'appui sur le bouton

1. Dans le pilote pour le bouton utilisateur, détecter les fronts montants et descendants, c'est à dire écrire deux fonctions :

   ```
   bool just_pressed();
   bool just_released();
   ```

   qui renvoie `true` si le bouton a changé d'état (respectivement de 0 vers 1 et de 1 vers 0) depuis le dernier appel à ces fonctions. Vous aurez besoin d'une variable d'état (variable globale) supplémentaire.
2. Testez cette fonction en modifiant `SysTick_Handler` de sorte qu'elle passe au motif de clignotement suivant à chaque pression sur le bouton : `clign1()`, puis `clign2()`, puis `clign3()`, puis `clign4()`, puis `clign1()` etc.

1. Coup de pouce : ajoutez une variable `unsigned int counter` qui compte les millisecondes depuis l'allumage ; votre `SysTick_Handler` ressemblera alors à `if (counter % 100 == 0) {...}`.

3. Sur certaines cartes, un appui sur le bouton peut faire avancer l'animation de plusieurs pas à la fois, car le bouton "rebondit" : quand les deux conducteurs du bouton approchent, le contact est intermittent et plusieurs fronts montants peuvent être détectés [2].

   Pour corriger cela, on va appliquer dans nos fonctions un algorithme de *debouncing* : on garde maintenant en mémoire les 8 dernières mesures du bouton, sous la forme de huit bits (un `char`). À chaque appel, on décale tous les bits vers la gauche, et on introduit à droite (bit de poids faible) notre nouvelle mesure. Maintenant, modifiez `just_pressed()` pour qu'elle ne renvoit `true` que si l'historique des mesures est égal à `0b01111111`, et `just_released()` renvoit `true` s'il est `0b10000000`. Testez.

3 Interface utilisateur

On cherche maintenant à coder le comportement suivant pour notre application :

— le programme est soit en mode *normal*, soit en mode *sélection de fréquence*. On passe d'un mode à l'autre par pression longue (>512ms) sur le bouton. À l'allumage, on est en mode normal.

— en mode sélection de fréquence, une des 8 LEDs seulement est allumée ; elle indique la vitesse de clignotement sélectionnée. Chaque LED correspond à une vitesse : 8, 16, 32, 64, 128, 256, 512, ou 1024 ms. Une pression courte sur le bouton sélectionne la vitesse suivante. À l'allumage, la vitesse 128ms est sélectionnée.

— en mode normal, les LEDs affichent un des 4 motifs de clignotement, tournant à la vitesse sélectionnée. une pression courte sur le bouton passe au motif suivant.

1. Modifiez `SysTick_Handler()` de façon à détecter les pression longues sur le bouton. Une pression longue, c'est quand :
   — il s'est écoulé 512ms depuis le dernier appel à `just_pressed()` ayant renvoyé `true`,
   — aucun appel à `just_released()` n'a renvoyé `true` depuis (sinon c'est un appui court).
   Vous aurez besoin d'une variable d'état supplémentaire.
2. Isolez maintenant dans une paire de fichiers `ui.c/ui.h` toute la logique de l'interface utilisateur. On y définira trois nouvelles variables d'état : `bool mode`, `char freq` et `char pattern`. On en exportera deux fonctions, `read()` et `display()`, qui respectivement a) interroge l'état du bouton et éventuellement met à jour `mode` ou `freq`, et b) met à jour l'état des LEDs en fonction de l'état courant.
3. Appelez ces deux fonctions dans `SysTick_Handler()`, et testez votre programme.

2. Googlez "bouncing switch".