TP 2 Pilotes de périphériques

Nous allons continuer à manipuler le programme "blinky" du dernier TP (répertoire `1_blinky/`) et à en isoler des pilotes de périphériques pour allumer et éteindre les LEDs embarquées ainsi que le bouton utilisateur. On rappelle les sources de documentation à utiliser :

— le manuel de notre carte (schéma de la carte, affectation des pattes)
— le manuel de référence de son MCU (documente les périphériques embarqués et les registres les contrôlant)

1 Analyse du programme fourni

Avec l'aide de la documentation du MCU, interprétez chacunes des lignes du programme fourni qui manipulent les registres GPIO :

1. Quel est l'effet de `RCC->AHBENR |= (1 << 21);` ?
2. Quel est l'effet de `GPIOE->MODER |= 0x55550000;` ?
3. Quel est l'effet de `RCC->AHBENR |= (1 << 17);` ?
4. Quel est l'effet de `GPIOA->MODER |= 0x00000000;` ?
5. Quelle est la valeur de `!(GPIOA->IDR & 0x00000001)` ?
6. Quel est l'effet de `GPIOE->ODR ^= 0x0000FF00;` ?

2 Écriture de pilotes

1. Modifiez le pilote des LEDs que vous avez commencé à écrire au TP 1 pour qu'il contienne les fonctions suivantes :

```
void init_leds(); // allume le peripherique et initialise les pins
void set_leds(char num, char v); // num=numero de la LED (0-7),
                                 // v= 1 (allume) ou 0 (eteint)
char get_leds(char num);; //retourne 1 si LED num allumee, 0 sinon
```

2. Écrire un pilote pour le bouton utilisateur : écrire une paire de fchiers `button.h` et `button.c` qui contiennent les fonctions suivantes :

```
void init_button(); // allume le peripherique et initialise les pins
char get_button();  // renvoie 1 si bouton appuye, 0 sinon
```

3. Testez vos pilotes dans `main()` en écrivant un programme qui inverse l'état de chaque LED séquentiellement toutes les ≃100ms, et uniquement quand le bouton est pressé. Le code de main() doit être lisible et portable : il ne doit contenir aucune information spécifique à la plateforme (adresse et structure des registres) mais utiliser uniquement des appels au pilote.
4. Modifiez le pilote du bouton pour détecter les fronts montants, c'est à dire écrire une fonction :

```
char just_pressed(); // renvoie TRUE si le bouton a chang'e d'etat
                     // (de 0 vers 1) depuis le dernier appel a
                     // cette fonction
```

Vous aurez besoin d'une variable d'état (variable globale) supplémentaire.

5. Testez cette fonction en modifiant `main()` de sorte qu'elle inverse l'état de chaque LED séquentiellement à chaque fois que le bouton est pressé.
6. Sur certaines cartes, un appui sur le bouton peut faire avancer l'animation de plusieurs pas car le bouton "rebondit" (quand les deux conducteurs du bouton approchent, le contact est intermittent [1]). Comment résoudre ce problème en software ?

1. Googlez "bouncing switch".