TP 9 I^2C

Le but de ce TP est de communiquer avec l'accéléromètre embarqué sur votre carte (référence LSM303DLHC) au moyen du bus I^2C : le configurer, récupérer et traiter ses mesures à intervalle régulier. On se servira de la datasheet du composant, qui indique l'adresse et la signification de chacun de ses registres internes. [1] L'adresse I^2C par défaut de l'accéléromètre est 0x32 (sur 7 bits), et son espace d'adressage est d'un octet.

1 Initialisation et configuration

1. Écrire le code d'initialisation du périphérique I2C1 et le router vers les pattes PB6 (SCL) et PB7 (SDA). Il faut :
   — activer les horloges GPIOB et I2C1
   — configurer PB6 et PB7 à l'aide du GPIO en *alternate function open drain*, activer le *pull down*, la vitesse maximum et sélectionner la fonction alternative correspondant à I2C1 (à chercher dans `stm32f3xx_hal_gpio_ex.h`).
   — déclarer une `I2C_HandleTypeDef` et y remplir les champs : `Instance`, `Mode`, `Timing` (mettre à `0x00902025`), `AddressingMode`.
   — passer cette *handle* à la fonction d'initialisation `HAL_I2C_Init()`.
2. Écrire des fonctions

   ```
   void AccelWrite(uint8_t reg, uint8_t data)
   uint8_t AccelRead(uint8_t reg)
   ```

   qui vont respectivement écrire une valeur `data` (8 bits) dans le registre `reg`, et retourner la valeur courante du registre `reg`. On rappelle que :
   — pour écrire `data` dans `reg`, l'accéléromètre attend un message composé de `reg` (8 bits) suivi de `data` (8 bits).
   — pour lire le contenu de `reg`, on doit envoyer un message simplement composé de `reg` (8 bits), puis on attend la réception d'un message contenant la valeur.
3. Après l'initialisation du bus I^2C, il faut configurer l'accéléromètre en écrivant la bonne valeur dans son registre `CTRL_REG1_A` (consultez la datasheet pour connaître sa signification). Activer tous les axes, et réglez la vitesse de capture sur 50Hz.
4. Tester ce code en faisant afficher toutes les 20 millisecondes les 8 bits lus au registre d'adresse 0x29 sur les 8 LEDS. A quoi correspond ce registre ?
5. Isoler un *driver* pour l'accéléromètre, dont voici le `.h`.

   ```
   typedef struct {
     float x; // -1..1
     float y; // -1..1
     flat z;  // -1..1
   } AccelData;
   ```

1. http://www.st.com/content/ccc/resource/technical/document/datasheet/56/ec/ac/de/28/21/4d/48/DM00027543.pdf/files/DM00027543.pdf/jcr:content/translations/en.DM00027543.pdf

```
typedef enum {
  ACCEL_RATE_OFF = 0b0000,
  ACCEL_RATE_1Hz = 0b0001,
  ACCEL_RATE_10Hz = 0b0010,
  ACCEL_RATE_100Hz = 0b0101,
  ACCEL_RATE_400Hz = 0b0111,
} AccelRate;

void AccelInit(AccelRate rate);
void AccelGetData(AccelData* data);
```

Tester lors de l'appel à `AccelGetData` que les données sont bien converti en un flottant entre -1 (carte complètement penchée dans une direction) et 1 (carte complètement penchée dans l'autre). [2]

2 Traitement des données

On va maintenant traiter les coordonnées x et y venant du capteur pour afficher la direction vers laquelle la carte est penchée (comme dans le programme de démonstration qui venait avec les cartes). Les données lues sont en coordonnées cartésiennes ; il faudra les convertir en polaires : un angle θ et un rayon R :

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \qquad R = \sqrt{x^2 + y^2}$$

1. Calculer θ à l'aide de la fonction `atanf` (attention aux divisions par zéro). On rappelle qua la fonction `atanf` travaille en radian : elle renvoit un flottant dans $[-\frac{\pi}{2}; \frac{\pi}{2}]$.
2. Après chaque calcul, allumer uniquement la LED la plus proche de la direction dans laquelle la carte est penchée.
3. Comme vous pouvez le constater, les LEDs clignotent de façon déplaisante quand on est entre deux valeurs : il y a du bruit dans la mesure. Filtrons le bruit à l'aide d'un filtre passe-bas à un pôle, qui va "lisser" les valeurs de chaque lecture.
   (a) déclarer un nouveau vecteur de coordonnées `AccelData filtered_data` (une variable globale). C'est ce vecteur qui doit être utilisé pour calculer θ.
   (b) à chaque lecture, calculer :

   ```
   filtered_data.x += c * (data.x - filtered_data.x);
   filtered_data.y += c * (data.y - filtered_data.y);
   ```

   où `data` est la dernière lecture, et `c` est le *coefficient* du filtre (une constante dans $[0; 1]$).
   (c) ajuster le coefficient `c` de façon à filtrer au maximum le bruit sans introduire de "lag" entre le mouvement de la carte et la valeur affichée.

2. On pourra ou bien récupérer les deux registres de de poids faible et fort, et les combiner (16 bits de résolution), ou bien se contenter du registre de poids fort (8 bits).