Informatique embarquée en SAÉ
GPIO et Liaison série

1 GPIO

1.1 Compteur : entier signé avec bit de signe

Écrire un programme qui va compter de -64 à +63. Pour chaque entier, afficher sa valeur binaire grâce au tableau de LED (voir annexe). Si le nombre est négatif, alors allumer la LED la plus à gauche de la carte. Attendre une seconde entre deux affichages.

1.2 Interaction bouton-LED

On désire compter le nombre de fois que le bouton utilisateur a été pressé, et afficher le résultat sous forme binaire en passant par le tableau de LED fourni avec votre carte (voir annexe). On utilisera une variable intermédiaire pour simuler une capture de front, comme illustré dans la Figure 1 ci-dessous :

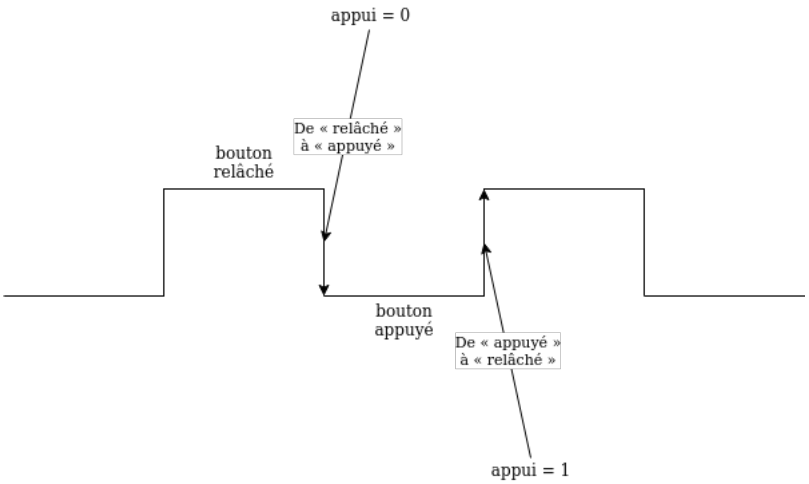

FIGURE 1 – Détection de l'appui (ou du relâchement) d'un bouton.

2 Connexion série

2.1 Premier programme avec liaison série

Créer une variable permettant de créer une connexion pour une liaison série (UART) et permettant de relier votre carte à votre PC. Pour ce faire, on va utiliser la syntaxe suivante. Pour une variable appelée pc, on écrira :

```
Serial pc(SERIAL_TX, SERIAL_RX);
```

Pour transmettre un caractère unique, par exemple, le caractère 'a' via la connexion, on utilisera la fonction putc de la manière suivante :

```
pc.putc('a');    // envoie le caractere 'a' sur la liaison serie
                 // qui connecte la carte au PC.
```

Pour écrire une phrase complète, on utilisera printf (on donne un exemple ci-dessous) :

```
pc.printf("Bonjour. Le caractere choisi est %c\r\n", 'a');
```

En vous inspirant de la question 1 partie 1, écrire un programme qui affichera « bouton appuye X fois » quand le bouton passera de « relâché » à « appuyé ».

2.2 Réception de caractères sur la liaison série

On veut pouvoir réagir instantanément avec la carte, dès qu'on reçoit une commande sous forme d'envoi de caractère unique. Pour cela, on va associer une *interruption* à l'arrivée d'un caractère via la liaison série grâce à la fonction `attach`, comme ceci :

```
#include "mbed.h"

Serial pc(SERIAL_TX, SERIAL_RX);
// autres declarations, etc. ici

void action() {
        // code de l'action ici
}

int main()
{
//      On fournit le nom de la fonction qui devra effectuer l'action voulue
//      lorsqu'on recoit une nouvelle information via la liaison serie.
        pc.attach(&action);
        // reste du code

        while(1) {
                // code
        }
}
```

Écrire un programme qui allume une combinaison de LED différente suivant que les lettres suivantes ont été reçues sur la liaison série :

'a'	→	LED 1
'p'	→	LED 8
'w'	→	LED 2
'm'	→	LED 7
'f'	→	LED 1, LED 2, LED 3, LED 4
'k'	→	LED 5, LED 6, LED 7, LED 8

Tout autre caractère provoquera l'extinction de toutes les LED.
Rappel : on peut utiliser le mot-clé `switch` lorsqu'on a le choix de plusieurs actions en fonction d'une unique valeur lue. Pour lire un caractère reçu via le port série, on peut utiliser `getc` :

```
char car;
// plus de code ici...
car = pc.getc();
```

A Fonctions utiles pour le TP

DigitalIn nom(num_broche) : déclare une variable qui configure la broche numéro `num_broche` en tant qu'entrée numérique, et l'associe à la variable appelée `nom`.
Exemple :

```
// PC_13 : broche associee au bouton utilisateur de la STM32F446RE
// Alias de PC_13 : USER_BUTTON
DigitalIn bouton(PC_13);
```

Attention, le bouton utilisateur de la STM32F446RE est actif à l'état bas.

DigitalOut nom(num_broche) : déclare une variable qui configure la broche numéro `num_broche` en tant que sortie numérique, et l'associe à la variable appelée `nom`.
Exemple :

```
DigitalOut led1(PB_6);
// PA_5 (alias : LED1) : broche associee a la LED de la STM32F446RE.
DigitalOut led2(PA_5);
```

Déclarer un tableau d'entrées ou sorties numériques :

```
DigitalIn  nom_tab[nb_broches]   = { num_broche1, ..., num_brocheN  };
DigitalOut nom_tab2[nb_broches2] = { num_br1, num_br2, ..., num_brK };
```

Le code ci-dessus crée un tableau `nom_tab` de `nb_broches` cases de type `DigitalIn`. Chaque case est associée à une broche, dans l'ordre des numéros `num_broche1`, `num_broche2`, ..., `num_brocheN`. On crée aussi un tableau appelé `nom_tab2` de `nb_broches2` cases de type `DigitalOut`, qui sont associées aux broches numéro `num_br1`, `num_br2`, ..., `num_brK`.
Lire le contenu d'une broche déclarée dans un tableau :

```
// lit le contenu (0 ou 1) de la 5e broche du tableau nom_tab2.
int valeur = nom_tab2[4]; :w
```

Fonction d'attente/temporisation : `wait`.
Exemple :

```
float val = 1.5;
wait(val); // le programme attend <val> secondes avant de continuer.
```

B Exemple complet

```
#include "mbed.h"

DigitalIn  button(PC_13);
DigitalOut leds[8] = { PB_8,PB_9,PA_5,PA_6,PA_7,PB_6,PC_7,A_9 };

/* Allume les LED du tableau <leds> une par une */
void leds_on() {
    for (int i = 0; i < 8; ++i) {
        leds[i] = 1;
    }
}

/* Eteint les LED du tableau <leds> une par une. */
void leds_off() {
    for (int i = 0; i < 8; ++i) {
        leds[i] = 0;
    }
}

int main() {
        int direction = 0;
    while(1) {
        if (button == 0) { // bouton appuye
            direction = !direction;
            if (direction == 1)
                leds_on();
            else
                leds_off();
            wait(1.0);
        }
    }
}
```

C Fonction utile : afficher un nombre en binaire sur des LED

```
/* A chaque iteration, on affecte la valeur du poids faible de
 * <num> a <leds[i]>, puis on decale <num> d'un rang vers la droite.
 * A la fin, <num> vaut 0, et toutes les LED sont configurees.
 */
void update_leds(unsigned num) {
  for (int i = 0; i < 8; ++i) {
    leds[i] = num & 0x1; // <leds[i]> <- bit de poids faible de <num>
    num     = num >> 1;  // decale <num> de un rang vers la droite.
  }
}
```