Microcontrôleur ARM Cortex M3
TP 1 : GPIO (General Purpose Input Output)

Ce TP doit vous permettre d'utiliser les fonctions de base de l'outil de développement Keil µVision :

- *Créer un projet,*
- *Éditer et compiler un programme,*
- *Programmer un microcontrôleur.*
- *« Debugger » le programme sur une cible (application)*
- *Simuler son fonctionnement, et observer son action*

Afin de commencer à développer vos propres programmes relatifs à l'utilisation des GPIO du microcontrôleur (exercices du TD n°1).

Pour effectuer la suite des opérations reportez-vous au document « Guide d'utilisation de Keil µVision ».

Prise en main de l'IDE Keil µVision :

- *Dans votre espace de travail personnel, vous commencerez par créer un répertoire qui contiendra les sous-répertoires associés à chaque TP. Ces répertoires permettront de recevoir les différents fichiers générés par Keil µVision lors de la configuration et la compilation d'un projet*
- *Ouvrez le tutoriel de prise en main de l'environnement de développement (IDE) Keil µVision et suivez toutes les étapes jusqu'au chapitre « Programmation du microcontôleur ».*

Création de votre premier programme :

*Vous allez reprendre l'exemple du programme « **`Bouton_Led.c`** » du TD1 afin de le tester sur la carte de développement STM32*

Création d'un projet :

- *Lancez Keil µVision.*
- *Créez un nouveau projet dans le répertoire de votre espace de travail, créer votre programme principal (main.c), ajoutez-y le contenu du fichier **`Bouton_Led.c`** puis organisez correctement la structure des répertoires et des fichiers de votre projet.*
- *Paramétrez correctement le compilateur.*
- *Puis compilez (construire) votre projet et corrigez les éventuelles erreurs qui apparaîtrons lors de cette phase.*

Programmation du microcontrôleur :

- *Programmez le microcontrôleur de la carte de développement MCBSTM32.*
- *Testez le bon fonctionnement du programme sur la carte de développement*
- *Que se passe-t-il sur la carte de développement si l'on quitte l'IDE Keil µVision ?*
- *Reprendre la démarche pour le fichier **`Joy_Led.c`*** du TD1 (fichier présent sur moodle)

« Débuggage » des programmes :

- *A partir du projet précédent et du programme* ***Joy_Led.c*** vous allez tester le mode « debug In Situ »

Débuggage In-Situ :

- *Lancer le débuggage In-Situ.*
- *Testez, en exécution « pas à pas », que votre programme fonctionne correctement. Que peut-on observer lors du déroulement du programme ?*
- *Après vous être assurés du bon déroulement de votre programme, lancer le programme (exécution temps réel) et vérifiez son fonctionnement.*

Débuggage en simulation : pour information (FACULTATIVE)

Cette manipulation est facultative pour le TP, mais peut être nécessaire afin d'effectuer la mise au point de vos programmes lors de vos préparations avant les séances de TP.

- *Désactiver maintenant le débuggage In-situ et activer le mode debuggage en simulation.*
- *Observez, par une simulation pas à pas, l'évolution des registres des ports sur lesquels le programme agit (GPIOB et GPIOG). Pour cela, vous utiliserez conjointement les fenêtres Memory, Analysis et aussi les IHM via le menu Peripherals.*
- *Créez un fichier de simulation permettant d'observer l'effet de l'appui sur le bouton BP_USER (PG8) sur le déroulement et les actions du programme.*

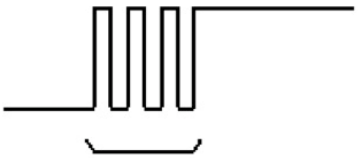
Testez ensuite tous vos programmes du TD1 que vous avez préalablement préparé *(exercices de préparation du TD1)*

IMPORTANT

Gestion d'un bouton poussoir

Un bouton poussoir est constitué d'une lamelle métallique qui, lors de l'appui sur le bouton, établit un contact électrique. Lors de cet appui, la lamelle rebondie et on observe, sur une certaine durée suivant l'appui, une succession d'impulsions électriques dues aux « rebonds » du bouton poussoir :

Appui



L'effet sur le traitement d'un programme qui teste l'appui sur un bouton sera de pouvoir interpréter ces rebonds comme plusieurs appuis successifs sur le bouton. Dès lors le résultat de ce traitement sera erroné.

Pour pallier ce problème, on préfèrera attendre le relâchement du bouton poussoir avant de réaliser l'action que cet appui devait déclencher :

Aussi pour les programmes que vous avez préalablement préparés il est fort à parier que cela ne fonctionne pas correctement dû à ce problème.

En réalité il faut traiter un appui par interruption et en déclenchant des « timers de « débounce » ». Les Timers seront vus au cours du TD2 et du TP2 et les interruptions au cours du TD3 et TP3.

Aussi pour l'instant on peut utiliser une méthode que l'on dira de type cochon pour traiter ces rebonds (bounce).

Pour cela on fera appel à une variable mémoire tampon et une temporisation logicielle (pas bien !!)

Exemple pour l'exercice 6a du TD1 que vous aviez à préparer :

main.h :

```
// main.h
#ifndef __MAIN_H
    #define __MAIN_H

    #define JOY_DOWN   3
    #define JOY_UP     15
    #define JOY_RIGHT  13
    #define JOY_LEFT   14
    #define JOY_SELECT 7
    #define BP_WAKEUP  0
    #define BP_TAMPER  13
    #define BP_USER         8

    #define LED_8                8
    #define LED_9                9
    #define LED_10          10
    #define LED_11          11
    #define LED_12          12
    #define LED_13          13
    #define LED_14          14
    #define LED_15          15

    #define APPUYE 0x0000

    #define ALL_LED_ON 0xFF00
    #define ALL_LED_OFF 0x0000

#endif
```

main.c

```
/*-------------------------------------------------------------------------
/*-------------------------------------------------------------------------
Concevoir un programme qui incrémente le port B (LEDs) à chaque appui sur le
bouton BP_USER

 * *-------------------------------------------------------------------------
-*/



#include <stm32f10x.h>        // STM32F10x Library Definitions

#include "main.h"

unsigned long bpUser;

unsigned int memoire = 1; // état relâché par défaut

unsigned int nombre = 0;



void Enable_GPIO(void);
void Init_GPIO(void);

int main (void)
{
    Enable_GPIO(); // Mise en service des ports
    Init_GPIO();  // Initialisation des Ports

    GPIOB->ODR = ALL_LED_OFF;    // LED PB15 OFF

    while(1){
     bpUser =  GPIOG->IDR & (1<<BP_USER); // lecture de l'état du bouton
d'incrémentation

     // Si le bouton a un état différent de celui enregistré ET
    // que cet état est "appuyé"
    if((bpUser != memoire) && (bpUser == APPUYE)){
       // on incrémente la variable qui indique
       // combien de LED devrons s'allumer
```

```
            nombre++;
            nombre %= 256;

            unsigned long tempo = 1000000;          //tempo cochon pour éviter
les rebonds (environ 14ms)
            while(tempo--);
    }
      // on enregistre l'état du bouton pour le tour suivant
      memoire = bpUser;
      GPIOB->ODR = (nombre << LED_8);          //on affiche le nombre sur les
leds 8 à 15 donc décalalé de 8bits
     }
}

void Enable_GPIO(void)
{

    RCC->APB2ENR |= (1 << 3); // Enable GPIOB clock
    RCC->APB2ENR |= (1 << 8); // Enable GPIOG clock
}

void Init_GPIO(void)
{
    GPIOB->CRH  = 0x33333333; // Mode=0b11 (50MHz) et CNF=0b00 (Push-Pull)
pour PB8 à PB15 (Leds 8 à 15)

    GPIOG->CRH |= 0x00000004; // Mode=0b00 (Input Mode) et CNF=0b01 (Floating
Input) pour  PG8
```