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

*Ce TD doit vous permettre de comprendre comment effectuer les configurations de base d'un microcontrôleur STM32 en langage C et de réaliser quelques programmes simples :*

- *Structure d'un programme en langage C,*
- *Définition des registres internes,*
- *Configuration des ports d'entrée/sortie,*
- *Opérations d'entrée/sortie,*
- *Exercices pour le TP.*

### **Exercice 1 :** organisation d'un programme en langage C

*Voici un programme permettant de piloter une LED (voyant) à l'aide d'un bouton poussoir :*

```
/*-------------------------------------------------------------------------
 * Name:    Bouton_Led.c
 * Purpose: GPIO usage for STM32
 * Version: V1.00
 * *-------------------------------------------------------------------------*/

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

#define APPUYE 0x0000

unsigned long bouton;

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

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

      while(1)
      {
            bouton = GPIOG->IDR & 0x0100;
            if ( bouton == APPUYE )     //-----------------------------------------------------
             GPIOB->ODR = 0x8000;       //-----------------------------------------------------
            else
             GPIOB->ODR = 0x0000;       //-----------------------------------------------------
      }
}

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-------------------------et CNF=0b00---------------------------------

      GPIOG->CRH = 0x00000004; //Mode=0b00-------------------------et CNF=0b01---------------------------------
}
```

*Q1.1 Repérer le début du programme*

*Q1.2 Repérer la déclaration de la variable utilisée par le programme principale*

*Q1.3 Combien d'octets cette variable occupe-t-elle en mémoire ?*

*Q1.4 Établir un diagramme d'activités (organigramme) illustrant le déroulement de ce programme.*

*Q1.5 Quel est le rôle de l'instruction d'itération :* `while(1) {...}`

## **Exercice 2 :** configuration des ports d'entrées/sorties (E/S)

*La fonction* `Enable_GPIO()` *permet de* ***mettre en service*** *les périphériques que l'on désire utiliser en leur appliquant un signal d'horloge. Cette opération se fera à travers le registre* APB2 Enable Register *des registres de configurations* `RCC (RCC_APB2ENR)`.

*Q2.1 D'après la documentation relative à ce registre, repérer quels sont les bits à positionner afin de mettre en service les ports B et G (GPIOB et GPIOG).*

*Q2.2 Quelle devra être la valeur (hexadécimale) à écrire dans ce registre pour activer ces ports*

*Dans le programme précédent, la fonction* `Enable_GPIO()` *affecte des valeurs au registre* RCC_APB2ENR *:*

*Q2.3 En langage C, l'opérateur* `x<<n` *réalise un décalage de n bit vers la gauche de la valeur binaire de x. Interpréter les valeurs assignées par les deux opérations de la fonction* Enable_GPIO( )

Q2.4 *Que réalise l'opérateur* `y |= x` *?*

*Q2.5 Interpréter le contenu du registre* RCC_AP2ENR *après l'appel de cette fonction.*

*D'autre part, lorsque ces ports sont activés, il convient de définir le mode de fonctionnement pour chacune des broches de ce port. Cette configuration se fait à partir des registres* GPIOx_CRL *et* GPIOx_CRH *(où x représente le nom du port).*

*Q2.6 Quels sont les registres affectés par la fonction* Init_GPIO()*?*

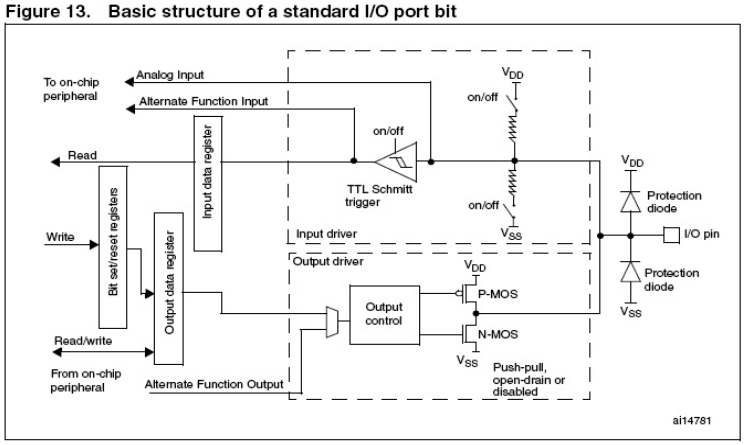
*Q2.7 Quelles sont les bits positionnés* dans ces registres *?*

*Q2.8 D'après la documentation de ces registres, analyser la configuration des broches des ports affectés par cette configuration*

## **Exercice 3 :** utilisation des ports d'E/S

*Chaque broche d'un port est accessible en entrée ou en sortie, suivant sa configuration, par les registres* GPIOx Input Data Register *(GPIOx_IDR) et* GPIOx Output Data Register *(GPIOx_ODR).*

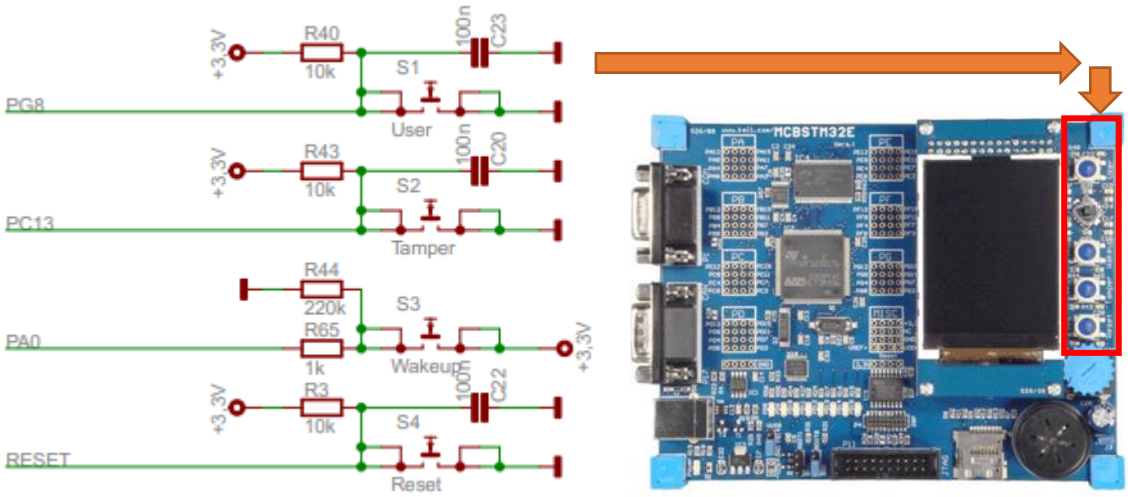
**Figure 13. Basic structure of a standard I/O port bit**




Q3.1 Que réalise l'instruction :

```
bouton = GPIOG->IDR & 0x0100
```

*Q3.2 Sur le kit d'évaluation voici le schéma de câblage des boutons du kit*




Q3.2 Déduire le nom du bouton poussoir utilisé (user, Tamper, Wakeup, Reset) au regard de la variable bouton :

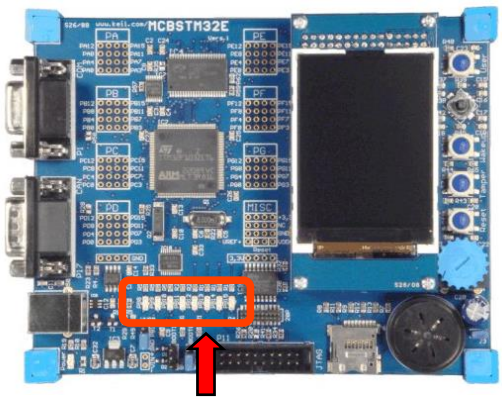
```
bouton = GPIOG->IDR & 0x0100
```

*Les boutons présentent un niveau logique 0 lorsqu'ils sont actionnés et un niveau 1 lorsqu'il sont au repos.*

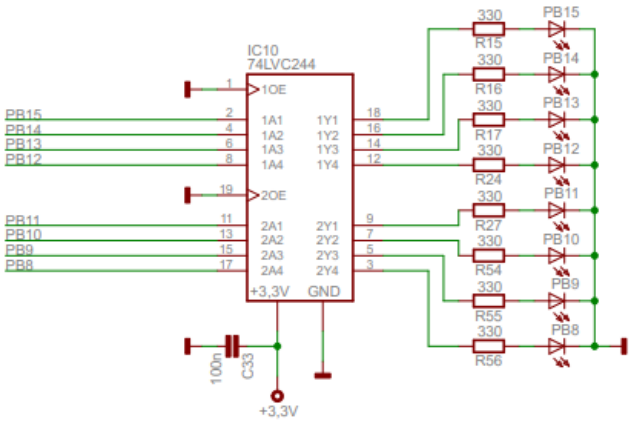
*Q3.3 Définir la valeur que prendra la variable bouton en fonction de l'action sur le bouton poussoir.*

*Q3.4 Analyser alors le résultat sur la condition de test.*

*Sur le port B, 8 LEDs sont connectées aux broches PB8 à PB15. Si une broche est au niveau logique 1, la LED est allumée, sinon (niveau logique 0) elle est éteinte.*




LEDS sur le kit STM32 des TPs




Q3.4 Analyser les instructions de test (if()…) du programme et décrire ce qu'il devrait se passer en fonction de l'appui sur le bouton poussoir.

## **Exercice 4 :** utilisation de masques

*Pour ne modifier que certains bits d'un registre, il est nécessaire de réaliser un masque. Donner, pour chaque instruction, le résultat équivalent (en commentaire) en hexadécimale sur le contenu des registres et ou de la variable et la configuration qui en découle. Si on ne connait pas la valeur précédente du/des bits on placera la valeur X au lieu de 0 ou 1.*

Q4.1

```
temp = GPIOG->CRL & 0x0FFFFFFF;//temp=0x……..
temp |= 0x40000000; //temp=0x……..
GPIOG->CRL = temp; // GPIOG->CRL =0x……..
```

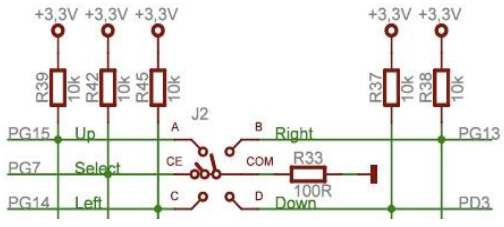
Q4.2 Comment devra être déclarée (typage) la variable temp (char, int, long, double, float….)?

Q4.3 Donner l'équivalence en hexadécimal (comme en Q4.*1*) des lignes ci-dessous (en commentaire) en tenant compte de la valeur initiale imposée par la première ligne

```
GPIOB->ODR &=0xAAAA ;
GPIOB->ODR &= ~0x8000; // GPIOB->ODR = 0x…….
GPIOB->ODR |= 0x8000; // GPIOB->ODR = 0x…….
GPIOB->ODR ^ = 0x8000; // GPIOB->ODR = 0x…….
GPIOB->ODR &= ~(1<<8); // GPIOB->ODR = 0x…….
GPIOB->ODR |= (1<<8); // GPIOB->ODR = 0x…….
GPIOB->ODR |= (0<<8); // GPIOB->ODR = 0x…….
```

*Sur la carte de développement utilisée en TP, un joystick est relié au port G et D de la façon suivante :*

| | |
|---|---|
| UP | PG15 |
| DOWN | PD3 |
| RIGHT | PG13 |
| LEFT | PG14 |
| SELECT | PG7 |




Q4.4 Ecrire les séquences d'instructions qui permettront de configurer correctement les broches de ces ports.

## **Exercice 5 :** adressage des registres

*Tous les registres du microcontrôleur sont accessibles par des adresses imposées par le constructeur. Pour accéder à ces adresses, on initialise un pointeur sur une adresse absolue grâce à une transformation de type (CAST) :*

*Ceci est un nombre :* `0x40012000`

*Un CAST transforme ce nombre en un pointeur sur un entier :*

```
(volatile unsigned long *) 0x40012000
```

*Et on accède au contenu de ce pointeur :* `*(volatile unsigned long *) 0x40012000`

*Afin de rendre le programme plus lisible, on préfèrera associer un nom avec l'adresse de chaque registre par une équivalence :*

```
#define CRL_PORTG *(volatile unsigned long *) 0x40012000
```

*=> La valeur 0x40012000 est donc transformée en un pointeur sur un entier. CRL_PORTG est équivalent au contenu de ce pointeur, donc au contenu de l'adresse 0x40012000.*

*Les registres de chaque port sont accessibles par une adresse de base et un offset. D'après la documentation du constructeur, les registres des périphériques GPIO sont organisés de la façon suivante :*

| **Adresse de base** | **Périphérique** |
|---|---|
| 0x40010800 | GPIO Port A |
| 0x40010C00 | GPIO Port B |
| 0x40011000 | GPIO Port C |
| 0x40011400 | GPIO Port D |
| 0x40011800 | GPIO Port E |
| 0x40011C00 | GPIO Port F |
| 0x40012000 | GPIO Port G |

| **Registre** | **Offset** |
|---|---|
| GPIOx_CRL | 0x00 |
| GPIOx_CRH | 0x04 |
| GPIOx_IDR | 0x08 |
| GPIOx_ODR | 0x0C |

Q5.1 Définir une équivalence qui permettra d'accéder aux broches du Port B en sorties.

Q5.2 Même chose pour les broches du Port G en entrées.

## **Exercice 6 :** préparation du TP

*Pour le TP relatif aux GPIO, vous préparerez les exercices suivants :*

### *1) Test des boutons et des LEDs :*

*Ecrire un programme qui permettra d'allumer une LED différente pour chaque bouton actionné sur la carte de développement :*

| **Bouton** | **Broche** | **LED** |
|---|---|---|
| BP_WAKEUP | PA0 | PB8 |
| BP_TAMPER | PC13 | PB9 |
| BP_USER | PG8 | PB10 |
| JOYSTICK_LEFT | PG14 | PB11 |
| JOYSTICK_RIGHT | PG13 | PB12 |
| JOYSTICK_UP | PG15 | PB13 |
| JOYSTICK_DOWN | PD3 | PB14 |
| JOYSTICK_SELECT | PG7 | PB15 |

### *2) Quelques petits programmes :*

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

*- Ecrire un programme qui permet de décaler l'allumage d'une LED à droite ou à gauche suivant que l'on actionne le bouton JOYSTICK_LEFT ou JOYSTICK_RIGHT.*

*- Réaliser une application qui permet d'allumer une LED lors de l'appui sur BP_WAKEUP puis son extinction par l'appui sur PB_TAMPER*

***Ces applications devront être préparées avant le TP afin d'être testées, validées et rédigées au cours de celui-ci***