Muhammad Haziq KAMARUL AZMAN
Ludovic MIRANDA 16/5/2012

Systèmes à Microprocesseurs

Travaux Pratiques n°2 : GPIO

Manipulation

Fonction de gestions des GPIOs

```
/*** gpio.c ***/

/* Fonction GPIO_Init qui initialise le port parallel avec la configuration des LEDs à utiliser */


void GPIO_Init(int config)

{

	GPIO_IODIR = config; /* On charge la configuration des LEDs à utiliser */

}


/* Fonction GPIO_Read qui retourne l'état du registre GPIO_IOPIN */


short GPIO_Read()

{

	return GPIO_IOPIN;

}


/* Fonction GPIO_Write qui écrit la configuration dans le GPIO */


void GPIO_Write(int mot)

{

	GPIO_IOSET = 0x00000000 ;

	GPIO_IOCLR = mot;
```

```
}
```

```
/*** main.c ***/

int main(void)
{
	int i,j;
	GPIO_Init(0x0000FF00);
	for(i=0;i<1000000;i++)
	{
		GPIO_Write(0x00000F00);
		for(j=0;j<3000;j++);
	}
}
```

Nous avons testé le programme et nous voyons que les LED 1 à 4 clignotent. Nous avons remarqué aussi que l'émulateur n'est pas si vite sous Seehau, donc nous avons enlevé les boucles de retard.

Fonction de gestions des LEDs

```
/*** led.c ***/
/* Fonction led_Init qui initialise les LEDs à utiliser */

void led_Init(int led_conf)
{
	GPIO_Init(led_conf);
}

/* Fonction led_Blink qui réalise le clignottement des LEDs */
```

```
void led_Blink()
{
	int i;
	int conf = (~GPIO_Read());
	while(1)
	{
		GPIO_Write(conf);
		for(i=0; i<300000; i++); /* Boucle de retard */
	}
}

/* main.c */
int main()
{
led_Init(0x0000FF00);
GPIO_Write(0x00000100);
led_Blink();

return 0;
}
```

Nous avons testé le code, et nous avons le LED 1 qui clignote. Par contre, nous n'avons pas tout intégrer dans une seule fonction, d'où le GPIO_Write (servant à déclarer l'état des registres GPIO à la configuration que l'on souhaite) dans le main.

Ensuite, nous avons modifié le programme pour prendre en compte la gestion des interruptions. En fait, l'interruption agit sur une variable globale *run* qui arrête la boucle lorsqu'une interruption est détectée.

Au niveau d'implémentation dans le code, nous l'avons mis comme condition de démarrage de la fonction *led_Blink*, ce qui fait maintenant que *led_Blink* prend en entrée un paramètre de type entier *run* dont la valeur est testée avant de lancer la procédure. Si *run* == 1 alors on boucle et si *run* == 0 on ne boucle pas. Nous avons dû donc enlever la boucle dans la fonction *led_Blink* et le mettre dans le programme principal.

Nous avons ensuite déclaré dans main.c une fonction de gestion des intérruptions :

```
___irq __arm void IRQ_Handler(void)
{
    if( run == 1 ) /* Si le boucle tourne, on l'arrête */
        run = 0;
    else /* Sinon, on met run = 0 pour assurer que run ne prend pas d'autres valeurs */
        run = 0;
}
```

Utilisation d'une interruption extérieure (EXTINT2)

Dans cette partie de la manipulation, nous avons écrit un code de gestion des intérruptions extérieures (dans notre cas générées par un bouton).

```
/* main.c */
#include "LPC210X.h"
#include "fonctions.h"
#include <intrinsics.h>

int run=1;

__irq __arm void IRQ_Handler(void){
    if(run == 1){ // S'il y a clignotement
```

```
                run = 0; // on l'arrête
        }
        else{  // sinon
                run = 0; // il n'y a pas clignotement !
        }
}

void isr_init(void){
        __disable_interrupt(); // Désactiver les interruptions
        PCB_PINSEL0 = 0x00000001;
        VICIntSelect = 0x00000000; // 0 assigned this register to the IRQ category
        VICIntEnable = 0x00008000; // on met le bit 16 à 1, d'après la table 44
        VICDefVectAddr = (unsigned int) IRQ_Handler;
        __enable_interrupt();
}

int main()
{
        led_Init(0x0000FF00);
        GPIO_Write(0x00000100);

  while(run){
        led_Blink();
  }

        return 0;
}
```