5 – Interruptions vectorisées

Objectifs : *gestion de périphériques (Timer, GPIO), mise en œuvre d'interruptions vectorisées.*

PREPARATION : LE CODE DES FONCTIONS `led_init` et `led_shift_left` DE LA PARTIE **1.2** DEVRA ETRE PREPARE ET SERA VERIFIE AU DEBUT DE LA SEANCE DE TP.

REALISATION : LES PARTIES **1.2**, **1.3** et **1.4** DOIVENT ETRE VALIDEES PAR L'ENCADRANT AU COURS DE LA SEANCE.

1.1 Description

L'objectif de cette séance est de mettre en œuvre un chenillard, i.e. l'allumage successif de diodes de la droite vers la gauche avec une temporisation à base d'interruptions générées périodiquement par un timer. Dans un deuxième temps, on utilisera une autre interruption générée par un bouton pour inverser le sens de défilement.

1.2 Réalisation du chenillard

Nous allons ici réutiliser les fonctions de gestion du GPIO et compléter celui des LEds.

- Créer un nouveau projet et ajouter les fichiers `gpio.c` et `led.c` écrites au TP3.
- Créer un fichier `main.c`, dans lequel vous écrirez une fonction `main()` appelant deux fonctions : `led_Init(int led_config)` qui devra initialiser le GPIO de façon a pouvoir utiliser les huit LEDs, et `led_shift_left()` qui devra gérer le décalage de LEDs et changer l'état de l'affichage *à chaque appel*.

Remarque : le chenillard peut être réalisé par simple opération de décalage d'une variable `led` (au départ `led` vaut 0x0100 puis chaque appel à `led_shift_left()` provoque un décalage `led << 1` pour allumage de la diode suivante et extinction des autres diodes).

- Ces deux fonctions se trouveront dans un fichier que vous appellerez `led.c` qui devra être ajouté au projet (il faudra également créer un fichier `led.h` contenant le prototype de ces deux fonctions). Le fichier `led.c` réutilisera les fonctions du fichier `gpio.c` et devra également inclure le fichier prototype `LPC210x.h` contenant les adresses des périphériques.
- Complétez ensuite le programme et tester l'affichage des LEDs. A ce stade vous pourrez utiliser une temporisation par une simple boucle for (e.g. `for (i=0; i<5000; i++){}`).

Vous mettrez une copie de vos listings commentés à ce stade dans votre compte-rendu.

1.3 Temporisation par Timer et interruptions

L'objectif est maintenant d'utiliser un des deux Timers disponibles pour contrôler la vitesse défilement. Il s'agit de générer périodiquement (0.1s) des interruptions provoquant le passage à l'état suivant du chenillard tous les dixièmes de seconde. Ce passage consiste simplement à appeler la fonction `led_shift_left()` depuis la routine d'interruption du Timer (`timer_interrupt`), elle-même appelée par la routine de gestion des interruptions

(`IRQ_Handler`). De la même façon qu'au TP2 (GPIO/Interruption), nous utiliserons les interruptions standard (IRQ) du processeur ARM.

Configuration du timer

Dans un fichier `timer.c`, rajoutez une fonction `timer_Init()` qui devra initialiser le Timer1. Les étapes à suivre sont les suivantes :

- Mettre la fréquence `pclk` à `cclk` = 10MHz (registre `SCB_VPBDIV`).
- Configuration du fonctionnement du Timer1 (registres `TCR`, `TC`, `PR`, `PC`).
- Configuration du Timer1 pour la génération d'interruptions (registre `MCR`).

Toujours dans le fichier `timer.c`, rajoutez une autre fonction `timer_Set(int tempo)` permettant de :

- Mettre la valeur du timer à 0 (registre `TC`).
- Spécifier la durée du timer par l'argument `tempo` (qui sera un multiple de 0.1 seconde (registre `MR0`).
- Démarrer le timer (registre `TCR`).

Comme lors du TP2, vous ajouterez un fichier `isr.c` au projet. Il contiendra pour l'instant trois fonctions : **`void`** `isr_Init()`, **`void`** `timer_interrupt()` et `__irq __arm void IRQ_Handler(void)`. Vous devez également écrire un fichier prototype `isr.h` et inclure le fichier `intrinsics.h`.

Configuration des interruptions

En vous aidant du User Manual, compléter la fonction `isr_Init` pour configurer le contrôleur d'interruption (VIC). Les étapes à suivre sont :

- Désactiver les interruptions (fonction `__disable_interrupt()`).
- Configuration du VIC pour permettre les IRQ du timer et affectation de l'adresse de la routine de gestion des interruptions `IRQ_Handler` (registres `VICIntSelect`, `VICIntEnable` et `VICDefVectAddr`).
- Autoriser les interruptions (fonction `__enable_interrupt()`).

a) Routine de gestion des interruptions :

Compléter la fonction `IRQ_Handler` qui appellera la fonction `timer_interrupt` pour exécuter la routine d'affichage du chenillard et effectuer un reset des interruptions (registre `IR`).

Tester et inclure le programme à ce stade dans vos comptes-rendus et expliquer en quelques lignes votre façon de procéder pour obtenir une temporisation au dixième de seconde.

1.4 Gestion d'une deuxième interruption

On souhaite maintenant contrôler le sens de défilement par l'appui sur un bouton (SW4). Du fait de la possibilité de deux interruptions, il faut un arbitrage. Un niveau de priorité est donc attribué aux interruptions dont la gestion est confiée à une unité spécifique (VIC, Vectorized Interrupt Controller).

Le fonctionnement est le suivant : un canal est attribué à chaque interruption, la plus prioritaire étant le canal 0, la moins prioritaire le canal 15. Lorsqu'une interruption survient, le contrôleur d'interruption résout les conflits éventuels en renvoyant l'adresse de la routine d'interruption la plus prioritaire par le registre `VICVectAddr`.

Dans notre programme, l'interruption prioritaire sera celle du bouton SW4 qui devra changer le sens de défilement *même si le processeur est entrain de traiter une interruption liée à la temporisation*. Pour cette raison, l'interruption liée à SW4 (`eint2_interrupt`) sera associée au canal 0 et l'interruption du timer (`timer_interrupt`) au canal 1.

a) Modifier le code nécessaire dans la fonction `isr_Init` pour initialiser le VIC. Les étapes à suivre sont les suivantes :
   - Ajouter la configuration du bouton SW4 pour qu'il génère une interruption EINT2 (registre `PINSEL0`).
   - Modifier la configuration originale du VIC pour permettre les interruptions du timer ***et*** du switch SW4 (registres `VICIntSelect`, `VICIntEnable`).
   - Compléter/modifier la configuration du VIC (registres `VICVectCntl0`, `VICVectAddr0`, `VICVectCntl1`, `VICVectAddr1`, `VICDefVectAddr`). Il faudra en particulier spécifier les adresses des deux routines d'interruptions (`timer_interrupt` et `eint2_interrupt`) aux canaux correspondants.

b) Le mécanisme d'appel à la routine d'interruption adéquate (`eint2_interrupt` ou `timer_interrupt`) est décrite dans la fonction `IRQ_Handler`, qui reste la routine principale exécutée lors d'une interruption. Celle-ci doit être modifiée comme suit :

```
__irq __arm void IRQ_Handler(void)
{
  void (*interrupt_function)();
  unsigned int vector;
  vector = VICVectAddr;
  interrupt_function = (void(*)())vector;
  (*interrupt_function)();
  VICVectAddr = 0;
}
```

Analyser et expliquer le fonctionnement de cette routine, en lui ajoutant des commentaires.

c) Ajouter et écrire la fonction `eint2_interrupt`, puis tester votre programme. Vous pourrez dans un premier temps mettre au point une version qui arrête le défilement par appui sur SW4, puis la version qui inverse le défilement (il faudra alors ajouter une fonction `led_shift_right()`, ainsi qu'une variable pour la gestion du sens de défilement).

Rajouter le code nécessaire, tester votre programme et inclure le code source de vos programmes dans vos comptes-rendus, avec un petit paragraphe d'explication du fonctionnement des interruptions dans votre programme.