5 – Interruptions vectorisées

Objectifs : *mise en œuvre d'interruptions vectorisées (Timer, External Interrupt, Nested Vectored Interrupt Controller).*

Préparation: lire la partie **1.1** Description et **1.3** Temporisation par timer et interruptions et **1.4** Deuxième interruption du sujet, ainsi que STM32F4 Series Reference Manual p. 249-259.
En vous basant sur la séance de TP1, écrire le programme de la question **1.2**.

Réalisation : les parties **1.2**, **1.3** et **1.4** doivent être validées par l'encadrant au cours de la séance.

La préparation est obligatoire et vérifiée en début de séance.

1.1 Description

L'objectif de cette séance est de mettre en œuvre la commutation on/off successive des quatre LEDs `LD3`, `LD4`, `LD5`, `LD6` en rotation dans le sens horaire (*clockwise*). On utilisera une temporisation à base d'interruptions générées périodiquement (0.5 seconde) par un timer. Dans un deuxième temps, on utilisera une autre interruption générée par un bouton pour inverser le sens de défilement (*counterclockwise*).

Le principe général (simplifié) des interruptions est illustré à la figure 1. Une interruption (IRQ, Interrupt ReQuest) est un signal envoyé au microcontrôleur par un de ses périphériques pour interrompre le programme en cours et exécuter un sous-programme spécifique (appelé routine d'interruption, e.g. `TIM3_IRQHandler`). Exemple : l'appui sur une touche du clavier provoque une interruption du microprocesseur, la routine d'interruption associée est programmée pour afficher le caractère correspondant à l'écran.

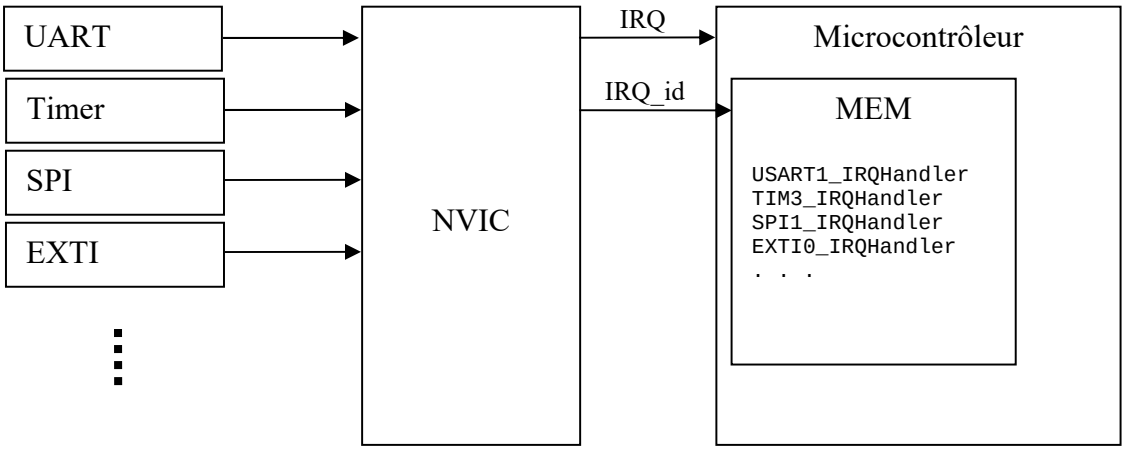

Figure 1. Principe général simplifié des interruptions.

Chaque périphérique peut se servir d'interruptions (Figure 1): une UART pour indiquer qu'elle a reçu un caractère, un bouton lorsque l'on appuie dessus, un timer pour indiquer qu'il a atteint une certaine valeur, etc. Chaque périphérique a donc une routine d'interruption associée qui est prédéfinie par le constructeur, mais dont le comportement est spécifié par le programmeur. Par exemple celle du timer `TIM3` est `TIM3_IRQHandler`. La fonction `TIM3_IRQHandler` peut ainsi être ré-écrite par un utilisateur dans son programme pour décrire une tâche particulière à effectuer lorsqu'une interruption est générée par le timer `TIM3`.

Toutes les sources d'interruptions possibles sont reliées à un composant unique, le contrôleur d'interruptions (NVIC, Nested Vectored Interrupt Controller). Lorsqu'une interruption est reçue par le NVIC, celui-ci envoie au microcontrôleur un signal d'interruption (IRQ) et le numéro de périphérique qui a généré l'interruption (IRQ_id). Le microcontrôleur interrompt le programme en cours et exécute la routine d'interruption correspondante.

Configuration du projet en C:

- Ouvrir Keil µVision.
- Project → Create new µVision project

Sélectionner le répertoire `C:\users\elec3\TP_microprocesseurs\TP5` (à créer s'il n'existe pas). **/!\ ne jamais travailler dans Mes Documents, ni sur le Bureau Windows.**
Nom du fichier : TP5_NVIC

- Dans la fenêtre Select Device for Target 'Target 1', cliquer dans STMicroelectronics et sélectionner STM32F4 Series / STM32F407 / STM32F407VG.
- Dans la fenêtre Manage Run-Time Environment, sélectionner: Board Support → STM32F4-Discovery (à sélectionner dans le menu déroulant), CMSIS → CORE, Device → Startup
- Pour pouvoir utiliser les librairies CMSIS pour les timers, cocher Device → StdPeriph Drivers → TIM, Framework et RCC.
- Pour pouvoir utiliser les librairies CMSIS pour les LEDs et le GPIO, cocher aussi Board Support → STM32F4-Discovery → LED et Device → GPIO dans le menu, puis faites OK.
- Dans le menu projet à gauche, clic droit sur Source Group 1, puis Add New Item to Group 'Source Group 1' / C File (.c) / Name: `main_nvic.c`, puis Add.
- Essayez de compiler, si vous avez l'erreur « function assert param declared implicitely », clic droit dans Target1 → Options for Target 'Target 1'… → C/C++ → Preprocessor Symbols → Define : `USE_STDPERIPH_DRIVER`
- Remplacer le fichier `system_stm32f4xx.c` par celui qui se trouve sur la page http://users.polytech.unice.fr/~bilavarn/ rubrique Systèmes à Microprocesseurs.
- Clic droit dans Target1 → Options for Target 'Target 1'… → C/C++ → Preprocessor Symbols → Define et ajouter : `HSE_VALUE=8000000`.

Configuration du debugger ST LINK :

- Dans le menu Options → Debug → ST Link Debugger → Settings → Port: SW.

Option à décocher pour utiliser la gestion des priorités dans les interruptions :

- Clic droit dans Target1 → Options for Target 'Target 1'… → C/C++ → décocher `One ELF Section per Function`

1.2 Fonction de commutation des LEDs

A l'aide des fonctions de la librairie CMSIS (`LED.h`), écrire une fonction `LED_shift_clockwise` qui met en œuvre la commutation on/off successive des quatre LEDs `LD3`, `LD4`, `LD5`, `LD6` en rotation dans le sens horaire (*clockwise*). **A l'appel de la fonction `LED_shift_clockwise`, la LED allumée est éteinte et la LED suivante est allumée.** Pour que le clignotement soit visible, on utilisera *provisoirement* une boucle `for` entre deux clignotements: `for (i=0; i<10000000; i++);` Une temporisation rigoureuse de 0.5 secondes par timer et interruptions sera ensuite réalisée à la question suivante.

1.3 Temporisation par Timer et interruptions

Nous utiliserons ici le timer `TIM4` (voir TP2). A l'aide des fonctions de la librairie CMSIS (`stm32f4xx_rcc.h`, `stm32f4xx_tim.h`, `core_cm4.h`), écrire les trois fonctions suivantes:

- `void TIM4_Initialize(void)` qui permet d'initialiser et de démarrer le timer `TIM4`. Les fonctions à utiliser sont:
  - `RCC_APB1PeriphClockCmd` pour activer l'horloge périphérique `RCC APB1` sur lequel est connecté le timer `TIM4`.
  - `TIM_TimeBaseInit` pour configurer le timer (prescaler, mode comptage, période, division d'horloge).
  - `TIM_ITConfig` pour activer les interruptions du timer sur dépassement (*update interrupt*).
  - `TIM_Cmd` pour démarrer le compteur.
- `void NVIC_Initialize(void)` pour initialiser le contrôleur d'interruption, en faisant appel à :
  - `NVIC_EnableIRQ` pour que le contrôleur d'interruption autorise les interruptions du timer `TIM4`.
- `void TIM4_IRQHandler(void)` qui est la routine d'interruption du timer `TIM4`. L'interruption est indiquée par le bit `UIF` du registre `TIM4_SR`, qui doit être testé au début (`TIM_GetFlagStatus`) et réinitialisé à la fin (`TIM_ClearFlag`) pour que les interruptions fonctionnent. Cette routine d'interruption doit décrire les actions à faire à chaque interruption du timer (commutation des LEDs).

1.4 Deuxième interruption sur bouton B1

On souhaite maintenant contrôler le sens de défilement par l'appui sur le user button B1 (bleu). Le bouton B1 est par construction connecté au port PA.0 qui peut générer une interruption `EXTI0` (STM32F4 Series Reference Manual p. 259).

Du fait de la possibilité de deux interruptions, il faut un arbitrage qui décide quelle interruption doit être traitée si les deux interruptions se produisent en même temps (le processeur ne peut exécuter qu'une routine d'interruption à la fois). Un niveau de priorité est attribué à chaque interruption, la plus prioritaire correspond au niveau 0. Lorsqu'une interruption survient, le contrôleur d'interruption analyse les sources d'interruption et renvoie la plus prioritaire au microprocesseur pour qu'il exécute la routine d'interruption associée.

Dans notre programme, l'interruption prioritaire sera celle du bouton B1 (`EXTI0`) qui devra changer le sens de défilement *même si le processeur est entrain de traiter une interruption de temporisation*. Pour cette raison, l'interruption `EXTI0` liée à B1 aura une priorité de 0 et l'interruption du timer `TIM4` aura une priorité de 1.

Configuration du projet :

Project → Manage → Run-Time Environment…

- Pour utiliser les librairies CMSIS des interruptions externes, cocher Device → EXTI et Device → StdPeriph Drivers → SYSCFG.

A l'aide des fonctions de la librairie CMSIS (`misc.h`), écrire, modifier ou ajouter les quatre fonctions suivantes:

- `void EXTIO_Initialize(void)` qui permet d'initialiser l'utilisation du bouton B1. Les fonctions à utiliser sont:
  - `RCC_AHB1PeriphClockCmd` pour activer l'horloge périphérique `RCC AHB1` sur lequel est connecté le bouton B1.

  - `RCC_APB2PeriphClockCmd` pour activer l'horloge du contrôleur de configuration système (`SYSCFG`). Voir `stm32f4xx_rcc.c` et `stm32f4xx_rcc.h`.
  - `GPIO_PinConfigure` pour configurer la broche PA.0 en mode: Input, Output Push Pull, Output Speed 50MHz, et GPIO No Pull up/down.
  - `SYSCFG_EXTILineConfig` pour connecter `EXTI Line 0` à la broche PA.0. Voir `stm32f4xx_syscfg.c` et `stm32f4xx_syscfg.h`.
  - `EXTI_ConfigureLine` pour activer les interruptions `EXTI Line 0` sur front montant (*trigger rising*). Voir `EXTI_STM32F4xx.c` et `EXTI_STM32F4xx.h`.
- `void NVIC_Initialize(void)` pour initialiser le contrôleur d'interruption, en faisant appel à :
  - `NVIC_Init` pour configurer les deux interruptions `EXTI0_IRQn` et `TIM4_IRQn` (Numéro de canal, activation de l'interruption, priorités préemption : 0, sous priorités canal : 0 pour `EXTI0` et 1 pour `TIM4`). Voir fichiers `misc.c` et `misc.h`.
- `void EXTI0_IRQHandler(void)` pour inverser le sens. On pourra utiliser une variable globale `turn` qui vaudra `0` ou `1` :
  - L'interruption est indiquée par le bit `0` du registre `EXTI_PR`, qui doit d'abord être réinitialisé (`EXTI_ClearPendingBit(0)`) pour que les interruptions fonctionnent à nouveau.
  - Si `turn = 0` → `turn = 1` ; si `turn = 1` → `turn = 0`
- `void TIM4_IRQHandler(void)` pour tourner dans le sens horaire ou antihoraire :
  - Si `turn = 0` → `LED_shift_clockwise`; si `turn = 1` → `LED_shift_counterclockwise`;
  - Ne pas oublier de tester et de réinitialiser le bit `UIF` (voir **1.3**)
- Il faudra donc rajouter et tester une fonction `LED_shift_counterclockwise`.

Complétez vos compte-rendus avec une copie du code, des explications sur l'utilisation des fonctions CMSIS dans votre programme.
Expliquez par ailleurs dans un paragraphe l'utilisation des interruptions dans votre programme (pas le principe général des interruptions qui est expliqué dans le sujet).