2 - Timer

Objectifs: *utilisation d'un timer et introduction aux interruptions.*

Préparation: lire la partie **1.1** Description et **1.3** Temporisation par timer et interruptions du sujet, ainsi que STM32F4 Series Reference Manual p. 249-256 et p. 422-427.
En vous basant sur la séance de TP1, écrire le programme de la question **1.2.**
Quelle est le numéro d'interruption attribué au timer `TIM3` ?
Quelle valeur de période du timer faut-il utiliser pour obtenir une temporisation de 0.5 seconde :
a) en supposant que le *prescaler* est à 0
b) en supposant que le *prescaler* est à 10000
Sur combien de bits est codée la période du timer ? Répondez en faisant référence à la documentation (quel document, quelle page).
Peut-on coder la valeur de la période trouvée précédemment avec ce nombre de bits ?
Par la suite, on supposera que le *prescaler* est réglé à10000. Expliquez pourquoi.

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 une temporisation à base d'interruptions générées périodiquement par un timer. On utilisera cette temporisation pour faire clignoter les quatre LEDs `LD3`, `LD4`, `LD5`, `LD6` à chaque seconde (0.5 seconde allumées, 0.5 éteintes).

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.

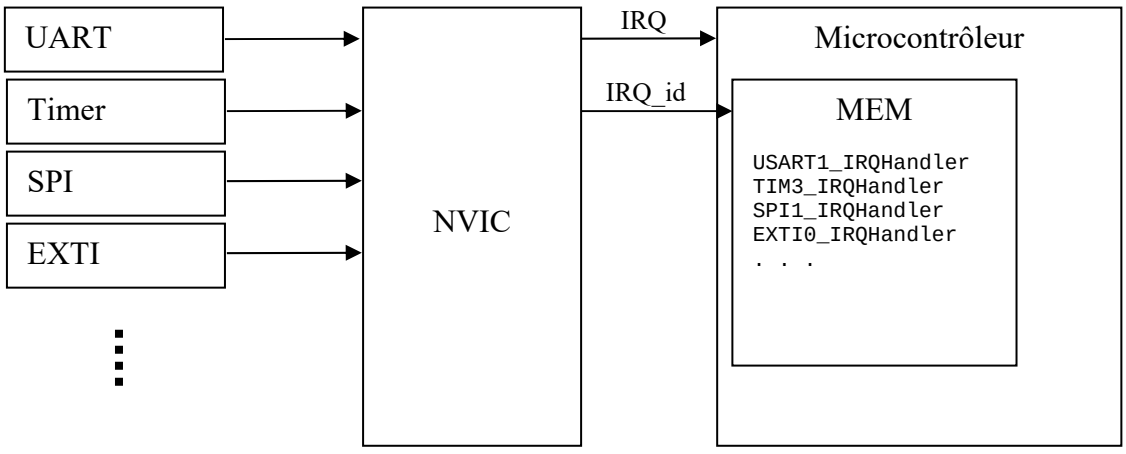

Figure 1. Principe général simplifié des interruptions.

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\TP2` (à créer s'il n'existe pas). **/!\ ne jamais travailler dans Mes Documents, ni sur le Bureau Windows.**
Nom du fichier : TP2_Timer

- 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_timer.c`, puis Add.
- Essayez de compiler, si vous avez l'erreur « function assert param declared implicitely », click 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/
- 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.

1.2 Clignotement

A l'aide des fonctions de la librairie CMSIS `LED.c`, écrire (dans le fichier `main_timer.c`) un programme qui fait clignoter les quatre LEDs `LD3`, `LD4`, `LD5`, `LD6`. Pour que le clignotement

soit visible, on pourra utiliser 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 dans les deux questions suivantes.

1.3 Temporisation par timer et interruptions

Nous utiliserons ici le timer `TIM3`. Le principe de fonctionnement des interruptions pour notre programme est le suivant. Le timer est clocké à 84MHz. Pour réaliser une temporisation de 0,5 secondes, il doit être configuré pour compter de 0 à 42000000 périodiquement. Lorsqu'il atteint 42000000, il doit générer une interruption. Cette interruption est analysée par le contrôleur d'interruptions (NVIC) qui envoie un signal d'interruption (IRQ) au microcontrôleur. Celui-ci interrompt sa tâche en cours et exécute la routine d'interruption prédéfinie associée au timer (`TIM3_IRQHandler`). La fonction `TIM3_IRQHandler` doit être programmée pour allumer ou éteindre les quatre LEDs en fonction de leur état courant. On pourra utiliser par exemple une variable globale `unsigned int led` : si `led=0` (LEDs éteintes), on allume les LEDs et on met à jour `led=1` ; sinon (LEDs allumées), on éteint les LEDs et on met à jour `led=0`.

Utilisation du *prescaler* : le timer compte à sa fréquence de fonctionnement de 84MHz quand le *prescaler* est réglé à 0. Cela signifie qu'il s'incrémente à chaque cycle d'horloge (`1/84000000` secondes). Si le prescaler est réglé à 1, il s'incrémente tous les 2 cycles d'horloge (`2 * 1/84000000` secondes), S'il est réglé à 2, il s'incrémente tous les 3 cycles d'horloge (`3 * 1/84000000` secondes), etc.

En s'aidant de la documentation STM32F4 Series Reference Manual décrivant le fonctionnement et les registres des périphériques, l'objectif consiste à écrire les trois fonctions suivantes qui permettront de manipuler le timer `TIM3`:

- `void TIM3_Initialize(void)` qui permet d'initialiser et de démarrer le timer `TIM3`. Les registres à configurer sont :
  - `RCC_APB1ENR` pour activer l'horloge périphérique `RCC APB1` sur lequel est connecté le timer `TIM3`.
  - `TIM3_CR1` pour configurer tous les bits à 0 (comptage croissant, division d'horloge = 1, etc).
  - `TIM3_PSC` pour définir le prescaler du timer `TIM3`.
  - `TIM3_ARR` pour spécifier la valeur de la période.
  - `TIM3_DIER` pour activer les interruptions sur dépassement (configuration : *update interrupt*)
  - `TIM3_CR1` pour démarrer le compteur.
- `void NVIC_Initialize(void)` qui permet d'initialiser les interruptions pour le timer `TIM3`. Configurer pour cela le registre :
  - `NVIC_ISER0` pour que le contrôleur d'interruption autorise les interruptions du timer `TIM3`. Cette étape consiste à mettre à 1 le bit n° `X` du registre `NVIC_ISER0` (où `X` est le numéro d'interruption attribué au timer `TIM3` trouvé en préparation). Dans le code il faut utiliser la notation `NVIC->ISER[0]` car c'est un tableau.
- `void TIM3_IRQHandler(void)` qui est la routine d'interruption du timer `TIM3`. L'interruption est indiquée par le bit `UIF` du registre `TIM3_SR`, qui doit être testé au début (`update interrupt pending == 1`) et réinitialisé à la fin (`update interrupt pending = 0`) pour que les interruptions fonctionnent. Cette routine d'interruption doit décrire les actions à faire à chaque interruption du timer (clignotement).

1.4 Utilisation de la librairie standard CMSIS

L'objectif est maintenant d'utiliser les fonctions de la librairie CMSIS pour réaliser le programme précédent (`stm32f4xx_rcc.h`, `stm32f4xx_tim.h`, `core_cm4.h`). Il faut donc ré-écrire (dans un autre fichier `main_timer_cmsis.c`) les deux fonctions suivantes:

- `void TIM3_Initialize(void)` qui permet d'initialiser et de démarrer le timer `TIM3`. Les fonctions à utiliser sont:
  - `RCC_APB1PeriphClockCmd` pour activer l'horloge périphérique `RCC APB1` sur lequel est connecté le timer `TIM3`.
  - `TIM_TimeBaseInit` pour configurer le timer (prescaler, mode comptage, période, division d'horloge).
  - `TIM_ITConfig` pour activer les interruptions 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 `TIM3`.
- `void TIM3_IRQHandler(void)` qui est la routine d'interruption du timer `TIM3`. L'interruption est indiquée par le bit `UIF` du registre `TIM3_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 (clignotement).