Step Débute 2
LED clignote à 2Hz, interruption timer

Documents utiles :
Cortex-M3 Programming Manual (PM0056.pdf)

Le point périph …
Mécanisme d'interruption Timer
Le Timer en comptage modulo N

1. Introduction Interruptions, Timer

La manière de générer le clignotant vu dans le step précédent est à proscrire. Nous allons ici utiliser LA méthode systématiquement utilisée en embarquée qui consiste à utiliser un **composant électronique intégré**, un **compteur modulo N** qui, lorsque le modulo est atteint, est capable de générer une interruption. Ce compteur est inclus dans une unité **périphérique** (à l'instar du GPIO) ayant beaucoup d'autres fonctionnalités appelé **Timer**. Dans la suite du document, on confondra (abus de langage) *Timer* et *compteur modulo N*. Le Timer est indépendant du processeur. Il évolue librement.

1.1. Les interruptions

Pour rappel, voici résumé en image le principe des interruptions sur micro-contrôleur :

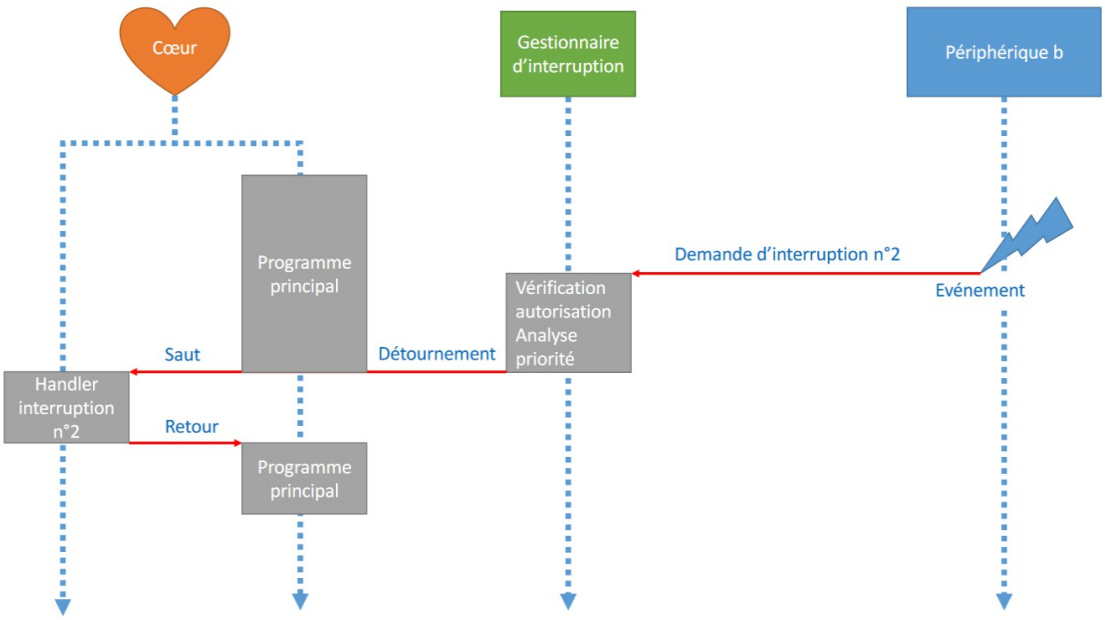


Figure 2 : Phases temporelles d'un traitement d'interruption

Commentaires :

- Le *CPU* (cortex M3), exécute son *programme principal* (le *main* ainsi que toutes les fonctions appelées logiciellement, par un *bl* donc...),
- En même temps le *périphérique b* (peut être un timer) travaille. A un moment donné, ce périphérique demande une interruption (ce peut être le débordement du timer qui atteint le modulo),
- la demande d'interruption arrive au *NVIC* (*Nested Vector Interrupt Controler*). Celui ci a la charge de transmettre, de mettre en attente, ou de ne pas transmettre la demande d'interruption au *CPU*. Pour cela, le *NVIC* aura dû être programmé de manière à ce que chaque ligne d'interruption ait une **priorité** associée.
- Si le *NVIC* décide de transmettre la demande d'interruption, le programme principal est immédiatement interrompu (l'instruction *ASM* en cours doit tout de même se terminer) et le *handler* associé à l'interruption est exécuté. Un *handler* est une fonction très simple (sans argument) qui est propre à une interruption et qui est donc **lancée de manière matérielle.** Idéalement, chaque interruption a son *handler* associé.
- De manière très classique (c'est le cas de ce BE), le *handler* appelle une fonction dite *CallBack*, que l'utilisateur peut choisir.

Qu'on se rassure ! :

Tout ceci est géré par la bibliothèque *DriverJeuLaser.lib*. Voir ici pour l'utilisation de la fonction permettant de configurer les interruptions timer.

1.2. Le Timer

L'architecture matérielle du timer ainsi que l'utilisation de la fonction qui lui est dédiée (bibliothèque *DriverJeuLaser.lib)* est présentée ici.

2. Configuration des périphériques (fonction *main*) 

1. Déployer l'archive *PjtKEIL_StepDeb_2.zip*
2. Ouvrir le projet KEIL *(.uvprojx),* sélectionnez la cible simulation
3. Compléter la configuration du main (balise //** Placez votre code là ** //). Il s'agit ici de :

   - configurer le **Timer 4** en débordement toutes les 100ms, priorité 2,

   - configurer le système d'interruption pour que le débordement du Timer 4 provoque le lancement de la fonction déjà écrite, void timer_callback(void).
4. Compiler et tester le code en simulation uniquement

3. Codage du callback en assembleur

Le travail consiste à produire exactement la même fonctionnalité, mais ici le *callback* sera écrit en assembleur dans le module *Cligno.s* déjà en place dans le projet. Ainsi, la fonction void timer_callback(void) ainsi que la variable globale char FlagCligno, doivent disparaître du fichier *principal.c*, pour apparaître maintenant dans le fichier *Cligno.s* (fichier presque vide, déjà inclus dans le projet).

5. Modifier le fichier *principal.c* en conséquence. Toute la configuration reste inchangée dans le fichier *principal.c*.
6. Coder le *callback* en assembleur. Pour cela suivre le même algorithme que celui utilisé dans initialement dans le fichier *principal.c*.

   NB : afin de pouvoir utiliser les fonctions GPIO, inclure le fichier *DriverJeuLaser.inc* dans *Cligno.s* . On disposera ainsi des adresses des fonctions.

   NB : la fonction assembleur est appelée par le handler associé au timer. La fonction doit donc respecter **ls règles de convention d'appel *AAPCS*** (*ARM Architecture Procedure Call Standard*), voir cours.
7. Tester la solution en simulation.
8. Tester la solution en réel.

4. Point périphérique de micro-contrôleur

4.1. Interruption logicielle périodique

Fonction de configuration :
*void Active_IT_Debordement_Timer(TIM_TypeDef *Timer, char Prio, void (*IT_function)(void));*

Exemple : Active_IT_Debordement_Timer(TIM1, 2, timer_callback);

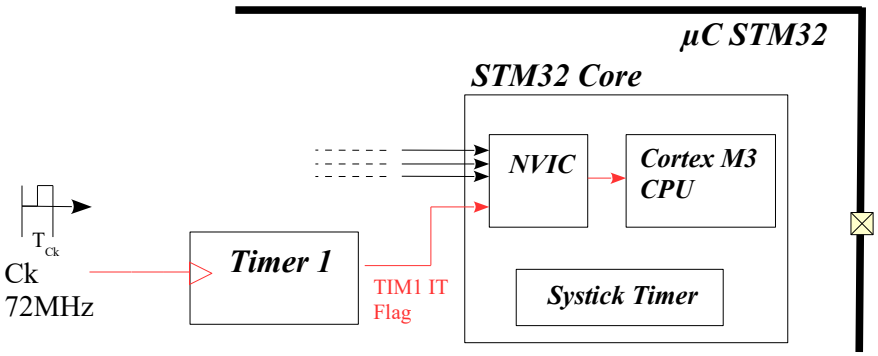
Cette fonction suppose que le timer 1 est déjà lancé avec une certaine périodicité.
Une fois que la ligne donnée en exemple est exécutée :

→ une **demande d'interruption** est faite lors de chaque débordement du timer 1 (mise à '1' d'un signal d'interruption provenant du Timer),
→ la demande d'interruption arrivant au *NVIC* (contrôleur d'interruptions) **est transmise au CPU si la priorité est suffisante**,
→ **le CPU** termine l'instruction *asm* en cours, sauvegarde certains registres, et **exécute la fonction *timer_callback*** via le *handler* d'interruption du timer 1

Les priorités d'interruptions :

- Si deux demandes d'interruptions arrivent en même temps, la plus prioritaire l'emporte. Si elles ont la même priorité, un ordre par défaut est établi par le constructeur.
- Si un *handler* est en cours d'exécution et qu'une nouvelle interruption arrive, celui-ci est interrompu si la priorité de la nouvelle demande est supérieure à celle en cours, sinon, la demande est mise en file d'attente,
- Un niveau de priorité de **0** est **la plus prioritaire**. Le niveau **15**, la moins prioritaire.

Aspect matériel



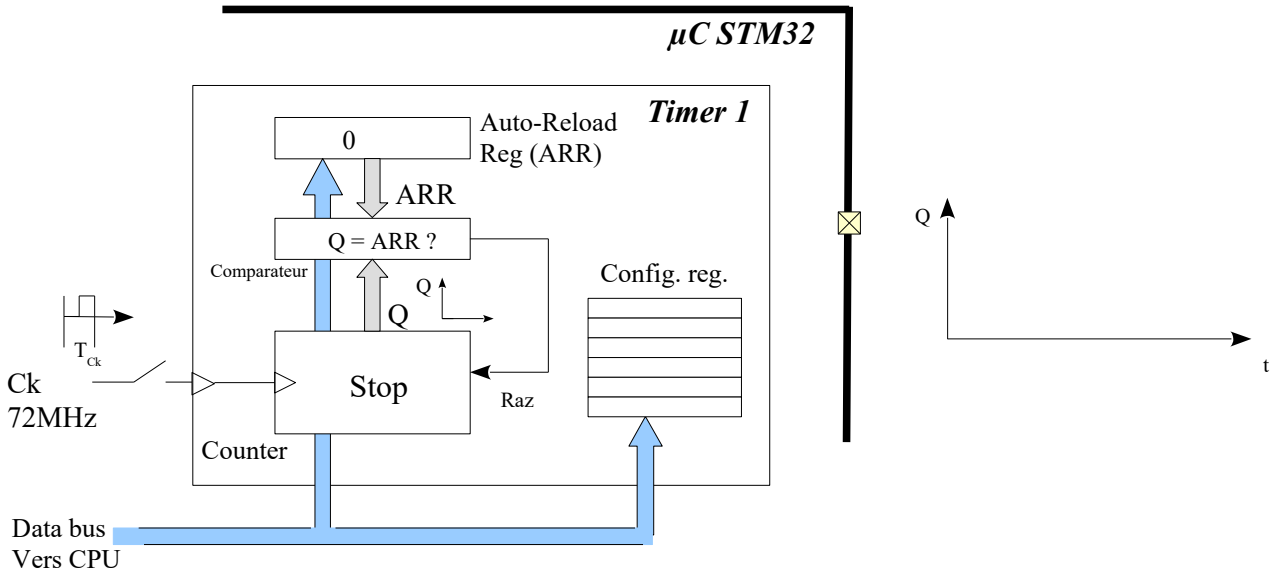
Retour

4.2. Le Timer

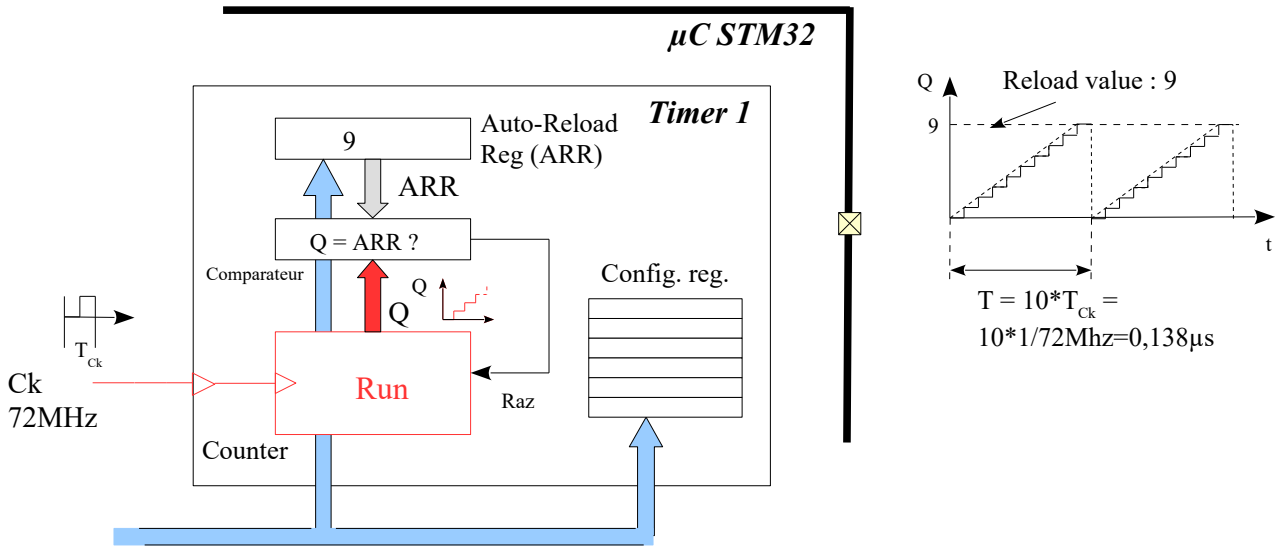
Fonction de configuration :
*void Timer_1234_Init_ff(TIM_TypeDef *Timer, u32 Duree_ticks)*

La fonction permet au Timer considéré de compter modulo *Duree_ticks*.
Exemple : *Timer_1234_Init_ff(TIM1, 10);*

Avant l'exécution *: Le timer 4 est arrêté, non configuré, non clocké*



Après l'exécution *: Le timer 4 est clocké, lancé, il cycle avec 10 périodes ticks (T_{ck})*



Retour