Tutoriels STM32F0

Interruptions

Interruptions Timer

Arouna DARGA – enseignant chercheur Sorbonne Université
arouna.darga@sorbonne-universite.fr

Table des matières

1. Introduction

Dans de nombreuses applications impliquant du traitement de signaux, on souhaite que la boucle de calcul soit exécutée à intervalles de temps réguliers, ce qui assure une période d'échantillonnage stable. Une façon courante de procéder consiste à mettre en place une base de temps avec un timer qui déclenche une interruption à chaque événement de mise à jour ou de débordement du compteur du Timer.

2. Utilisation d'un Timer pour générer des interruptions périodiques

2.1 Configuration du périphérique

Dans le fichier **bsp.c**, modifier la fonction **BSP_TIMER_Timebase_Init()**, en ajustant les paramètres de la base de temps du TIM6 afin d'obtenir une période de **200 ms** entre les événements de mise à jour, et activez les interruptions de mise à jour :

```
/*
 * BSP_TIMER_Timebase_Init()
 * TIM6 cadenser a 48MHz
 * Prescaler   = 48000 -> periode de comptage = 1ms
 * Auto-reload = 200  -> periode de debordement    = 200ms
 */
void BSP_TIMER_Timebase_Init()
{
        // activer horloge du peripherique TIM6
        // mettre a '1' le bit b4 (TIM6EN) du registre RCC_APB1ENR
        // voir page 131 du manuel de reference
        RCC->APB1ENR |= (1<<4);          // le bit b4 est defini comme etant
RCC_APB1ENR_TIM6EN
        // Faire un Reset de configuration du TIM6 : mise a zero des registres
        // TIM6_CR1 et TIM6_CR2
        // voir page 543 a 544 du manuel de reference
        TIM6->CR1 = 0x0000;
        TIM6->CR2 = 0x0000;
        // Configuration frequence de comptage
        // Prescaler : registre TIM6_PSC
        // Fck = 48MHz -> /48000 = 1KHz frequence de comptage
        TIM6->PSC = (uint16_t) 48000 -1;
        // Configuration periode des evenements
        // Prescaler : registre TIM6_ARR
        // 1000 /10 = 100Hz
        TIM6->ARR = (uint16_t) 200 -1;
        // Activation auto-reload preload : prechargement
        // mettre a '1' le bit b7 du registre TIM6_CR1
        TIM6->CR1 |= (1<<7);
        // Activation de la demande interruption de debordement
        // mettre a '1' le bit b0 du registre TIM6_DIER
        // page 544
        TIM6->DIER|= (1<<0);

        // Demarrer le Timer TIM6
        // Mettre a '1' le bit b0 du registre TIM6_CR1
        TIM6->CR1 |= (1<<0);
}
```

Ensuite, définissez la priorité et autorisez les interruptions du Timer TIM6 dans la fonction **BSP_NVIC_Init()** :

```
/*
 * BSP_NVIC_Init()
 * Configuration du controleur NVIC pour autoriser et accepter les sources interruptions activer
 */

void BSP_NVIC_Init()
{
	// Mettre en priorite maximuim les lignes interruptions externes EXTI 4 a 15
	NVIC_SetPriority(EXTI4_15_IRQn, 0);

	// Autoriser les lignes interruptions externes EXTI 4 a 15
	// le bouton Bleu est sur PC13 (ligne 13)
	NVIC_EnableIRQ(EXTI4_15_IRQn);

	// Mettre la priorite 1 pour les interruptions du TIM6
	NVIC_SetPriority(TIM6_DAC_IRQn, 1);

	// Autoriser les demandes interruptions du TIM6
	NVIC_EnableIRQ(TIM6_DAC_IRQn);
}
```

A ce stade, nous avons écrit tout le code nécessaire pour avoir un signal d'interruption du périphérique **TIM6** toutes les **200ms**, que le contrôleur **NVIC** va autoriser ou valider.

2.2 Implémentation de l'ISR

Comme indiqué précédemment, lorsque le signal d'interruption est validé et autorisé par le contrôleur **NVIC**, l'exécution du programme se branche automatiquement sur une adresse mémoire de code spécifique. En regardant dans le fichier assembleur **startup_stm32f0xx.S**, vous verrez que le nom de la fonction correspondant au gestionnaire d'interruption TIM6 (ou ISR) est : **TIM6_DAC_IRQHandler()**. Si vous vous demandez pourquoi le nom inclut "**DAC**", c'est parce que TIM6 a une fonction spécifique associée au DAC que les autres timer n'ont pas. Nous ne l'utilisons pas pour l'instant, vous pouvez donc considérer TIM6 comme un Timer de base ordinaire.

Écrivons quelque chose de simple dans le fichier **stm32f0xx_it.c** comme ISR (Interrupt Service Routine) du Timer **TIM6**:

```
/*
 * This function handles TIM6 interrupts
 */

extern uint8_t timebase_irq;

void TIM6_DAC_IRQHandler()
{
  // Verifier que cest le debordement de Timer TIM6 qui a declencher interruption
  if ((TIM6->SR & TIM_SR_UIF) == TIM_SR_UIF)
  {
    // CAcquitter ou valider la demande interruption
    TIM6->SR &= ~TIM_SR_UIF;

    // Tache a realiser
    timebase_irq = 1;
  }
}
```

Les timers peuvent être utilisés pour déclencher des interruptions dans de nombreuses situations. Par conséquent, et cela devrait être une pratique automatique. Chaque fois que vous écrivez un gestionnaire d'interruption, commercez par tester quel événement a déclenché la routine d'interruption. Comme nous n'avons activé l'interruption qu'en fonction de l'événement "**update**", il ne devrait y avoir aucune autre raison d'être là... mais il faut quand même le faire en vérifiant que le drapeau **UIF** est activé, puis l'effacer pour réactiver le signal d'interruption suivant.

N'oubliez pas que **l'ISR doit être court**. Ici, nous ne mettons à "1" qu'une variable globale ***timebase_irq***. Cette variable sera testée dans la fonction **main()**.

2.3 La fonction main()

La boucle principale continue désormais à tester les variables globales pour savoir s'il y a eu des interruptions. En fonction de la valeur de ces variables, des actions sont effectuées :

```c
#include <math.h>
#include "stm32f0xx.h"
#include "bsp.h"
#include "delay.h"
#include "main.h"

static void SystemClock_Config(void);

// variable globale
uint8_t button_irq = 0;
uint8_t timebase_irq = 0;

int main(void)
{
	// Configuration horloge du  systeme : 48MHz avec 8MHz HSE
	SystemClock_Config();
	//Initialisation de pin PC13 pour bouton Bleu avec interruption
	BSP_PB_IT_Init();
	// Initialisation USART pour console liaison serie
	BSP_Console_Init();
	// test console : message accueil
	mon_printf("La Console est Ready!\r\n");
	//Initialisation de Timer TIM6 pour delai avec interruption
	BSP_TIMER_Timebase_Init();
  // initialisation du controleur interruption NVIC
  BSP_NVIC_Init();
// boucle principale des applications
	while(1)
	{
		// Traitement de la tache lier a la detection du front descendant du bouton
		if(button_irq == 1)
		{
			mon_printf("#");
			button_irq=0;
		}
		// Exemple de tache importante a faire
		// envoyer "." au PC par USART tout les 200 ms
		// Maintenant il y a pas de delai attente 200ms
		if (timebase_irq == 1)
		{
			mon_printf(".");
			timebase_irq = 0;
		}
	}
}
```

	Sauvegarder votre code main et programmer le MCU. Vérifier que programme est plus réactif au Bouton et traite de manière fiable la tâche importante du main.

3. Résumé

Dans de nombreuses situations, l'UCM sera invitée à traiter les données à intervalles réguliers. L'utilisation de délais dans les boucles doit être évitée car un délai est quelque chose qui occupe l'UCM (comptage). L'utilisation de timers en même temps que l'interruption doit être votre premier choix. La configuration présentée dans ce tutoriel peut être utilisée comme configuration de base pour de nombreuses applications.