MONNERET Nicolas

MI43
Rapport de TP

MI43 – P07
M. LACAILLE

Introduction

Le but du TP est de mettre en place un mécanisme de commutation de tâches en utilisant un Timer système qui va générer des interruptions à intervalle régulier. On veut pouvoir exécuter plusieurs tâches en passant de l'une à l'autre à chaque interruption. On va donc d'abord mettre en place le Timer système, puis les différentes tâches. Enfin, une fonction d'endormissement de tâche sera créée.

1. Mise en place d'un Timer système

On souhaite mettre en place un Timer système qui génère une interruption de l'application toutes les 10 millisecondes.
On va donc, en langage C, initialiser l'AIC, initialiser le Timer 1, initialiser le PIO pour pouvoir afficher le résultat sur les leds et activer l'horloge de périphérique.
Nous allons également créer la routine de traitement des interruptions en langage Assembleur.

Pour définir les différentes structures et adresses que nous allons utiliser, on inclut le fichier « m55800.h » fourni.

```
#include "include/m55800.h"
```

Fonction d'initialisation du Timer et de l'AIC :

```
void activeInterruptTimer(void){
        StructAIC* aic;
        StructAPMC* apmc;
        /*Utilisation du block 0 des contrôles timers (afin d'utiliser le timer 1)*/
        StructTCBlock* tcb0;
        /*----------
        Configuration de l'horloge
        ----------*/
        apmc = APMC_BASE;
        apmc->APMC_PCER = 0x0080 ;
        /*----------
        Configuration du timer 1
        ----------*/
        tcb0 = TCB0_BASE;
        /*Valeur de comparaison à 2500 afin de générer une interruption toute les 10ms*/
        tcb0->TC[1].TC_RC=2500;
        /*Division de l'horloge par 128 (250KHz) et mode WAVE, trigger sur TC_RC*/
        tcb0->TC[1].TC_CMR = TC_CLKS_MCK128|TC_CPCTRG|TC_WAVE
        /*Autorisation des interruption sur comparaison de TC_RC*/
        tcb0->TC[1].TC_IER = TC_CPCS;
        /*Activation de l'horloge de TC1*/
        tcb0->TC[1].TC_CCR=TC_CLKEN|TC_SWTRG;

        /*----------
        Configuration de l'AIC
        ----------*/
        aic = AIC_BASE;
        /*Interruption sur timer 1 (Interruption source 7) avec une priorité de 7*/
        aic->AIC_SMR[7] = AIC_SRCTYPE_INT_LEVEL_SENSITIVE | 7;
```

```
	/*Adresse de la fonction de traitement des interruptions*/
	aic->AIC_SVR[7] = (int)routine_asm;
	aic->AIC_IECR = 1 << 7;
}
```

Fonction d'initialisation du PIO pour affichage sur les leds :

```
void activeLeds(void){
	StructPIO* piob;
	int leds;

	piob = PIOB_BASE;
	leds = PB8 | PB9 | PB10 | PB11 | PB12 | PB13 | PB14 | PB15;

	/*Activation des leds mise à zero de l'affichage*/
	piob->PIO_PER = leds;
	piob->PIO_OER = leds;
	piob->PIO_SODR = 0xFF << 8;
}
```

Fonction d'affichage sur leds:

```
void affSurLeds(unsigned char i){
	StructPIO* piob;
	piob = PIOB_BASE;

	piob->PIO_CODR = i << 8;
	piob->PIO_SODR = ~(i << 8);
}
```

Fonction de traitement des interruptions en C (appelée par l'Assembleur) :

```
void routine_c(void){
	StructTCBlock* tcb0;
	static int compteur = 0;
	tcb0 = TCB0_BASE;

	/*Lecture et acquittement du timer*/
	if(tcb0->TC[1].TC_SR){
		/*Compter jusqu'à 100 afin d'entrer toutes les secondes (interruption chaque 10ms)*/
		if(compteur >= 100){
			/*valeur affichée sur les leds (définie globale)*/
			val++;
			affSurLeds(cmp);
			compteur = 0;
		}
		else{
			compteur++;
		}
	}
}
```

Main, appel des différentes fonctions :

```
int main(void){
	StructTCBlock* tcb0;
	tcb0 = TCB0_BASE;

	val = 0;
	activeInterruptTimer();
	activeLeds();

	while(1);

	return 0;
}
```

Fonction Assembleur de traitement des interruptions :

```
AIC_EOICR equ 0xFFFFF130

	EXPORT routine_asm
	AREA routine, CODE, READONLY
	IMPORT routine_c

routine_asm
	; Sauvegarde du registre de liens dans la pile
	SUB lr, lr, #4
	STMFD sp!, {lr}
	; Sauvegarde du spsr
	MRS r14, spsr
	STMFD sp!, {r14}
	; IRQ autorisées
	MSR cpsr_c, #0x13
	; Sauvegarde des registres de base
	STMFD sp!, {r0-r3, r12, lr}

	; Appel de la routine c
	BL routine_c

	; Restauration des registres de base
	LDMFD sp!, {r0-r3, r12, lr}
	MSR cpsr_c, #0x92
	; Acquittement de l'AIC
	LDR r14, =AIC_EOICR
	STR r0, [r14]

	LDMFD sp!, {r14}
	MSR spsr_c, r14
	; Retour d'interruptions
	LDMFD sp!, {pc}^

	END
```

2. Commutation de tache

On doit ici créer plusieurs tâches (fonctions C) et les faire tourner alternativement. Pour cela, on va utiliser le timer créé précédemment pour générer des interruptions et passer d'une tâche à l'autre à chacune de ces interruptions. Nous allons donc créer un ordonnanceur qui va se charger d'alterner les processus.
Les tâches seront implémentées en C par une structure et décrites par un descripteur de tâche (PCB), qui sera un simple pointeur.

2.1. Sauvegardes / restauration de contexte

On modifie la routine de traitement des interruptions Assembleur afin de sauvegarder le contexte de la tâche en cours et de restaurer le contexte de la tâche suivante afin de passer d'une tâche à la suivante. La routine assembleur appelle la fonction C d'ordonnancement qui va se charger de passer à la tâche suivante.

Code de la routine Assembleur :

```
AIC_BASE    equ 0xFFFFF000
AIC_EOICR   equ 0x130
TC1_BASE    equ 0xFFFD0040
TC_SR       equ 0x20

        EXPORT asm_handler
        IMPORT handler_C
        IMPORT      PPCB
        AREA timer_handler, CODE, READONLY

asm_handler
        ; Sauvegarde du registre le lien dans la pile irq
        SUB         lr, lr, #4
        STMFD       sp!, {lr}
        ; Sauvegarde des registres communs à tous les modes dans le PCB courant
        LDR         r14, PPCB
        ADD         r14,r14,#8
        STMIA       r14,{r0-r7}
        ; Récupération du spsr et du pc pour sauvegarde
        LDMFD       sp!, {r1}
        STR         r1,[r14, # ((15*4)+4)]
        MRS         r0, SPSR
        STR         r0,[r14, # (15*4)]
        ; Pointeur sur le champ reg[8] la structure PCB
        ADD         r3, r14 , #(8*4)
        ; Mode svc
        MSR         cpsr_c, #0x93
        ; Sauvegarde des registres
        STMIA       r3!,{r8-r14}
        ; Sauvegarde du spsr  du mode
        MRS         r4, SPSR
        STR         r4, [r3,#8]
```

```
        ; On réautorise les irq
        MSR             cpsr_c, #0x13
        ; On acquitte le timer
        LDR             r14, =TC1_BASE
        LDR             r14,[r14, #TC_SR]

        ; On appelle le sheduler
        BL              handler_C

        ; On bloque les irq
        MSR             cpsr_c, #0x93
        ; Récupération de psr et pc
        LDR             r0, PPCB
        ; On ajuste le pointeur sur psr
        ADD             r0, r0, #(8+(15*4))
        ; On récupère PSR, PC et SPSR
        LDMIA           r0,{r4-r6}
        ; Restauration du SPSR du mode svc
        MSR             spsr_cxsf, r6
        ; Restauration r8-r14
        LDMDB           r0!, {r8-r14}
        ; Passage en mode irq
        MSR             cpsr_cxsf, #0x92
        ; Rrestauration du psr de la tâche
        MSR             spsr_cxsf, r4
        ; L'adresse de retour dans la pile
        STMFD           sp!, {r5}
        ; Restauration du reste des registres
        MOV   r14, r0
        LDMDB           r14!, {r0-r7}
        ; Retour d'interruption
ack
        ; On acquitte l'AIC
        LDR             r14, =AIC_BASE
        STR             r0, [r14, #AIC_EOICR]
        ; Retour d'interruption (restauration de la nouvelle tâche)
        LDMFD           sp!, {pc}^

        END
```

2.2. Création des taches et ordonnanceur

Une tâche est définie par la structure PCB suivante :

```
typedef struct s_PCB{
	unsigned handle;
	/*Etat de la tâche (en cours, en attente)*/
	unsigned state;
	/*Les 15 registres de base*/
	unsigned reg[15];
	/*Registre contenant le statut de la tâche*/
	unsigned psr;
	/*Adresse des traitements effectués par la tâche*/
	void* pc;
	/*Registre de sauvegarde du statut*/
	unsigned spsr;
	/*Pointeur sur la tâche suivante*/
	struct s_PCB* next;
} PCB_t;
```

Les instructions exécutées par les tâches sont définies dans des fonctions. Chaque fonction contient une boucle infinie et affiche des leds différentes :

```
void tache1(){
	while(1){
		affSurLeds(255);
	}
}
```

On définie le pointeur sur la tâche exécutée ainsi que les pointeurs sur les différentes tâches :

```
PCB_t *PPCB;
PCB_t pcb_idle, pcb_tache1, pcb_tache2, pcb_tache3;
```

La fonction appelée par le code Assembleur est la suivante :

```
void handler_C(){
	StructTCBlock* tcb0;
	tcb0 = TCB0_BASE;

	/*Lecture et acquittement du timer*/
	if(tcb0->TC[1].TC_SR){
		cptSeconde++;
		/*changement de tâche toutes les secondes (interruption chaque 10ms)*/
		if(cptSeconde >=100){
			scheduler();
			cptSeconde=0;
		}
	}
}
```

Fonction d'ordonnancement (passage à la tâche suivante) :

```
void scheduler(){
	PPCB = PPCB->next;
}
```

Dans le Main, on initialise le Timer, l'AIC, le PIO pour les leds et les différentes tâches :

```
int main(void){
	StructTCBlock* tcb0;
	tcb0 = TCB0_BASE;

	/*On place le pointeur de tâche sur la tâche principale*/
	PPCB = &pcb_idle;

	/*Initialisation de la tâche principale*/
	/*tâche suivante = tâche 1*/
	pcb_idle.next = &pcb_tache1;
	/*adresse de la tâche*/
	pcb_idle.pc = (void*) idle;
	pcb_idle.psr = 0x13;
	pcb_idle.spsr = 0x93;
	/*allocation de la mémoire pour la pile*/
	pcb_idle.reg[13] = (unsigned)malloc(50*sizeof(unsigned)) ;
	/*placement du pointeur de pile au sommet de celle-ci*/
	pcb_idle.reg[13] += 50*4 ;

	/*Initialisation de la tâche 1*/
	pcb_tache1.next = &pcb_tache2;
	pcb_tache1.pc = (void*) tache1;
	pcb_tache1.psr = 0x13;
	pcb_tache1.spsr = 0x93;
	pcb_tache1.reg[13] = (unsigned)malloc(50*sizeof(unsigned)) ;
	pcb_tache1.reg[13] += 50*4 ;

	/*Initialisation de la tâche 2*/
	pcb_tache2.next = &pcb_tache3;
	pcb_tache2.pc		= (void*) tache2;
	pcb_tache2.psr		= 0x13;
	pcb_tache2.spsr = 0x93;
	pcb_tache2.reg[13] = (unsigned)malloc(50*sizeof(unsigned)) ;
	pcb_tache2.reg[13] += 50*4 ;

	activeInterruptTimer1();
	activeLeds();

	while(1);

	return 0;
}
```

2.3. Endormissement / réveil

On souhaite mettre en place une fonction d'endormissement de tâche durant n cycles : sleep(n). Cette fonction pourra être appelée par les différentes tâches. Pour cela, on va rajouter à la structure PCB un champ attente qui contiendra le temps d'attente jusqu'au prochain réveil de la tâche. Ainsi, lors d'une interruption (toutes les 10ms), on va diminuer ce temps d'attente. Si une tâche possède un temps d'attente de 0, alors on va exécuter la tâche.

La structure PCB contient désormais :

```
unsigned attente;
```

A l'initialisation de la tâche dans le Main, on rajoute met l'attente à 0 :

```
pcb_tache1.attente = 0;
```

La fonction de traitement devient :

```
void handler_C(){
	StructTCBlock* tcb0 = TCB0_BASE;

	/*pointeurs temporaires qui permettent de parcourir les tâches*/
	PCB_t *temp = PPCB;
	PCB_t *temp2 = PPCB;

	if(temp2 == &pcb_idle){
		temp2 = PPCB->next;
		temp = PPCB->next;
	}

	if(tcb0->TC[1].TC_SR){
		cptSeconde++;
		/*On diminue le temps d'attente des différentes tâches à chaque interruption*/
		do{
			if((temp->attente) > 0) (temp->attente) --;
			temp=temp->next ;
		}
		while(temp != temp2);

		if(cptSeconde >=100){
			scheduler();
			cptSeconde=0;
		}
	}
}
```

L'ordonnanceur devient :

```
void scheduler(){
	/*Pointeur temporaire qui permet de parcourir les tâches*/
	PCB_t *temp = PPCB ;

	/*Tant que les tâches sont en attente, on passé à la tâche suivante*/
	do{
		PPCB = PPCB->next;
	}
	while((PPCB->attente) > 0 && PPCB != temp);

	/*Si l'attente de la tâche est égale à 0, alors la tâche est exécutée*/
	/*Sinon, on place le pointeur de tâche sur la tâche principale*/
	if(PPCB == temp && (PPCB->attente) > 0){
		PPCB = &pcb_idle;
		PPCB->next = temp->next ;
	}
}
```

Enfin, on met en place la fonction sleep(n), qui modifie le champ attente de la tâche. La variable n passée en paramètre est un temps en secondes, on la multiplie par 100 car l'attente est en centième de secondes.

```
void sleep(int n){
	PPCB->attente = n*100;
}
```

Conclusion

Nous avons créé trois tâches et la tâche principale, le Timer lance bien une interruption toutes les 10ms et l'ordonnanceur passe d'une tâche à l'autre comme prévu. Toutes les secondes, nous pouvons vérifier le fonctionnement du mécanisme en regardant le changement d'affichage des leds suivant la tâche exécutée. La fonction d'endormissement mise en place pour terminer fonctionne également. En plaçant par exemple, dans une des tâches cette fonction avec un endormissement de 5 secondes, alors on constate qu'elle n'est pas exécutée pendent une durée de 5 secondes.