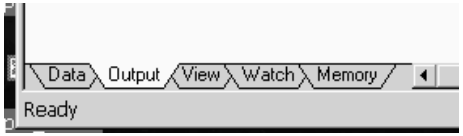
TP 7 : Liaisons Séries et Interruptions

Introduction

Dans ce TP, vous apprendrez comment utiliser le deboggueur NoIce comme terminal alphanumérique pour afficher des chaînes de caractères, et comment utiliser les interruptions pour améliorer la performance des programmes applicatifs. Vous terminerez par un travail de synthèse utilisant vos connaissances acquises sur la programmation embarquée sur le processeur 9s12.

1 Utilisation de Noice comme terminal

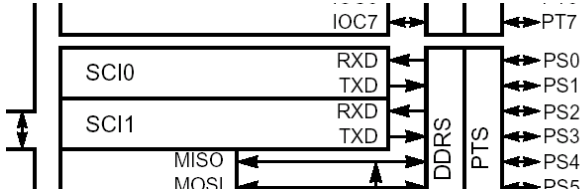
Le processeur est doté de liaisons séries qui lui permettent de communiquer avec des périphériques. Le deboggeur NoIce utilise l'une de ses liaisons, et on peut l'utiliser dans ses programmes, notamment comme terminal alphanumérique via le volet "Output" de la fenêtre inférieure. Celle-ci affiche tous les caractères alphanumériques reçus depuis la carte.



1.1 Fonction d'émission de caractères

Afin d'envoyer des caractères depuis son programme, on va définir une fonction `void outchar( char octet );`
qui réalise l'émission depuis la carte d'un octet vers l'hôte (le PC), et qui sera intercepté et affiché par NoIce.

Au niveau du processeur, les liaisons séries sont gérées par l'une des 2 SCI (Serial Communications Interface). Le processeur en possède 2 (désignées SCI0 et SCI1), complètement indépendantes. Elles sont chacune Full-Duplex (émission et réception simultanée), via leurs broches RxD (Réception) et TxD (émission), et prennent en charge l'implémentation des protocoles "bas niveau" de la communication série.



La transmission d'un octet est relativement simple, et se déroule en 2 étapes :

1. Vérifier (= attendre) que la transmission précédente soit terminée. Ceci est signalé par le passage à 1 du flag TC (Transmission Complete) situé dans le registre SCI0SR1
2. Ecrire l'octet à émettre dans le registre de transmission SCI0DRL

	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
Reset	0	0	0	0	0	0	0	0

FIGURE 1 – Le registre SCIxSR1 *(SCI Status Register 1)*

1.2 Travail à effectuer

Ecrire la définition de la fonction `outchar()`, la déclarer, et la tester avec le programme suivant, qui devra afficher l'alphabet :

```
1  int main()
2  {
3       int i;
4       for( i=0; i<26; i++ )
5            outchar( 'a' + i );
6  }
```

Note : on devrait normalement initialiser la liaison série avant son utilisation, afin de déterminer les paramètres de fonctionnement (vitesse, protocole, etc). Mais comme elle est par défaut initialisée par le moniteur implanté en mémoire et qui permet le fonctionnement du déboggeur, nous n'avons pas besoin de le faire ici.

1.3 Fonction d'émission de chaîne de caractères

Il est pratique de disposer d'une fonction permettant l'envoi d'une chaîne. Vous allez utiliser la fonction 'outchar()' pour écrire une fonction void print(char* msg), qui va réaliser la transmission d'une chaîne vers le terminal, en envoyant les octets un par un. Une chaîne de caractères est définie comme étant une suite d'octets (de codes ASCII) consécutifs en mémoire, terminée par un zéro. Cette fonction va donc passer le caractère courant (*msg) à la fonction `outchar()`, puis incrémenter le pointeur, jusqu'à ce que le caractère courant soit nul :

```
void print( char* msg )
{
     while( *msg != 0 )
     {
          outchar( *msg );
          msg++;          // incrémentation
     }
}
```

Tester cette fonction avec le programme suivant : (pensez à déclarer la fonction `print()`)

```
int main()
{
     print( "Bonjour !\n" );
}
```

1.4 Travail à effectuer

En utilisant les fonctions que vous venez de définir, écrire un programme nommé `tp7_a.c` qui affiche un message détectant l'appui sur l'un des deux boutons poussoirs, et affichant un message indiquant lequel.

2 Fonctionnement en interruptions

2.1 Présentation

Le fonctionnement en interruption est un mécanisme qui permet de déclencher automatiquement l'exécution d'une fonction lors de l'arrivée d'un évènement. Ce mécanisme peut être logiciel (on parle alors d'interruption logicielle), mais sera le plus souvent matériel, c'est à dire prévu au niveau de la puce. L'évènement peut être externe au processeur (signal externe ou requête d'un périphérique) ou interne (division par zéro, reset, fin d'un délai, etc.) Lors de l'arrivée de l'évènement, l'exécution du programme principal s'interrompt automatiquement, et le processeur exécute la routine d'interruption correspondante, puis reprend l'exécution du programme principal.

Dans tous les cas, la survenue de l'évènement est mémorisée, et peut être traitée ultérieurement (lorsque le processeur le peut, il peut être occupé à autre chose de plus important). Cette mémorisation se fait via un flag (bit) dans un registre. La contrepartie de cette mémorisation, c'est qu'il est indispensable d'acquitter cette mémorisation dans la routine d'interruption (en pratique, cela consiste à remettre le flag à zéro).

Intérêt des interruptions Sur un système réel, les évènements n'ont pas tous la même importance : une panne d'alimentation est un évènement plus important qu'un appui sur une touche du clavier. Sans interruptions, lorsque plusieurs sources sont à surveiller ou lors du traitement d'une tâche, il est possible de rater un évènement (ou de le traiter trop tard, ce qui revient au même). Avec les interruptions, en gérant correctement les priorités, on peut faire en sorte d'avoir une réactivité idéale : chaque évènement, selon son importance, peut être géré en temps utile.

2.2 Exemple d'application

Dans le TP n° 6, vous aviez écrit un programme réalisant la génération d'un signal sonore avec le buzzer. Le programme principal ne faisait qu'attendre le passage à 1 du flag pour incrémenter le registre TC2, il était difficile d'occuper le programme à autre chose, sous peine de "rater" l'activation du flag (revoir le TP6).

Vous allez donc implémenter une interruption qui va se déclencher **automatiquement** à chaque activation du flag, permettant ainsi d'occuper le programme principal à autre chose.

2.3 En pratique

Pour avoir un programme fonctionnant en interruptions, 4 points sont à considérer :

1. le programme principal, réalisant la "tâche de fond" de l'application
2. la routine d'interruption, qui doit traiter l'évènement
3. le bloc d'initialisation, qui va configurer le processeur pour activer les interruptions
4. Le vecteur d'interruption

2.3.1 Programme principal

Pour l'instant, il sera réduit à une boucle infinie :

```
     while( 1 )
          ;
```

2.3.2 Routine d'interruption (*Interrupt Service Routine*)

C'est une fonction d'un type particulier. Elle sera définie à la suite du main(), et devra réaliser l'acquittement de l'interruption (RAZ du flag O2F) et l'incrémentation de TC2 (voir TP6) :

```
1 void isr_ect_2( void )
2 {
3       TFLG1 = TFLG1 | BIT2;
4       TC2 += 1500;
5 }
```

Cette fonction (comme toute fonction...) doit être déclarée. Ceci est fait automatiquement par le fichier "vector.h" (voir plus loin).

2.3.3 Bloc d'initialisations

Il doit effectuer les mêmes initialisations que dans le TP6, plus :
- Autoriser les interruptions au niveau du timer
- Autoriser les interruptions de façon globale pour le processeur (elles sont inhibées par défaut)

Le premier point se fait en positionnant le bit correspondant à la sortie désirée dans le registre TIE. A chacun des 8 canaux du Timer correspond un bit dans ce registre.

	BIT7	6	5	4	3	2	1	BIT0
R W	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
RESET:	0	0	0	0	0	0	0	0

FIGURE 2 – Registre TIE (*Timer Interrupt Enable register*)

Le deuxième point se fait par une instruction assembleur qui met à zéro le bit I du registre CC (voir document "modèle du programmeur"), et s'appelle `cli` (*CLear Interrupt mask*).

Pour insérer une instruction assembleur dans un source en C, on utilise le mot-clé C "asm". La syntaxe complète est donc :

```
asm( "cli" );
```

2.3.4 Programmation du vecteur d'interruption

La programmation des vecteurs se fera pour nous en incluant un fichier d'entête particulier, "vector.h", qui contient la déclaration et l'initialisation d'un tableau de pointeurs, qui sera sauvegardé dans le fichier .S19 à l'adresse de la table des vecteurs (de $ff80 à $ffff pour nous). Il faut juste préciser **AVANT** la ligne `#include "vector.h"` quelle interruption on souhaite utiliser en déclarant une constante du nom de l'interruption. Ici, ce sera :

```
#define USE_ISR_ECT_2
```

On pourra bien sûr avoir un programme utilisant plusieurs routines d'interruptions, il faudra alors déclarer les constantes correspondantes, et définir les routines correspondantes.

2.4 Travail à effectuer

En reprenant le travail fait au TP6 (bloc d'initialisations notamment), réécrire le programme en le renommant en `tp7_b.c`, de façon à générer un signal sonore à 1000 Hz en interruption.

3 Exercice de synthèse

3.1 Interruptions sur TOF

Ecrire un programme `tp7_c1.c` fonctionnant en interruptions qui va faire clignoter les 8 dels du Port B toutes les secondes. Vous utiliserez le débordement du compteur TCNT (signalé par l'activation du flag TOF) comme base de temps. Cet évènement peut générer une interruption, à la condition que le bit TOI (Timer Overflow Interrupt Mask), situé dans le registre TSCR2 soit à 1 (voir énoncé TP 6 pour TSCR2). En choisissant un rapport de prédivision de 8, le compteur est soumis à une horloge de 3 MHz, il va donc déborder au bout d'un temps égal à $65536/3$ MHz, soit 21,8 ms. La routine d'interruption devra **compter** les intervalles de 21,8 ms, et au bout de 23 fois (il y a 23 fois 21,8 ms dans 500 ms), elle devra **inverser** les bits du port B, et réinitialiser le compteur.

Il faudra définir la constante `USE_ISR_ECT_TOF`, et la routine devra s'appeler :

```
void isr_ect_tof()
```

Attention, le compteur dans le routine d'interruption devra être déclaré comme variable globale (sinon le contenu est perdu chaque fois qu'on sort de la fonction). La routine devra aussi à chaque appel réinitialiser le flag TOF.

3.2 Deux routines d'interruption

Copier le programme précédent en `tp7_c2.c`, et lui ajouter une routine d'interruption sur le canal 2 du Timer, de façon à avoir un signal sonore intermittent :
- Dels allumées ⇒ signal sonore actif
- Dels éteintes ⇒ signal sonore inactif

Ceci se fera en basculant alternativement de 0 à 1 le bit autorisant la broche de sortie à commuter (voir table OM/OL des registres TCTL1 & TCTL2) dans la routine sur TOF.