TP : Utilisation du Timer du 9s12 en génération de signaux

Introduction

Dans ce TP, vous mettrez en oeuvre le Timer du 9s12 en mode "génération de temps". Vous verrez comment réaliser des temporisations calibrées, et comment l'utiliser pour générer un signal sonore, via le buzzer de la carte. Remarque : la consultation du cours sur le Timer est indispensable à ce TP !

1 Utilisation du Timer en comptage

Vérification de la base de temps Vous allez vérifier la validité du tableau de sélection du facteur de division de fréquence, en faisant clignoter les Dels à une fréquence déterminée par le Timer. On utilisera la dernière ligne du tableau de sélection, et on fera clignoter des dels sur le Port B, de façon à "observer" le temps généré. Vous aurez besoin d'utiliser des masques binaires, et il sera pratique de disposer d'un fichier d'en-tête réutilisable qui les définit :

```
1 // bitmask.h : contient les définitions des masques
2 #define BIT0 0x01
3 #define BIT1 0x02
4 ...
5 #define BIT7 0x80
```

En pratique, ce fichier existe déjà dans l'environnement de développement, il devient donc inutile de le créer.

Q1.1. Saisir le programme ci-dessous dans un fichier 'tp6_1.c'.

```
#include <sys/ports.h>
#include "bitmask.h"   // masques de bits
int main()
{
// A - initialisations
     DDRB  = 0xff;  // PORTB tout en sortie
     PORTB = 0xf0;  // 4 dels allumées, 4 éteintes
     TSCR1 |= BIT7; // Activation du Timer
     TSCR2 = ____;  // sélection rapport prescaler
// B - boucle
     while( 1 )
     {
          while( TFLG2 == 0 ) //attente du débordement Timer
               ;
          ..... = ....;        // RAZ de TOF
          PORTB = PORTB ^ 0xff;     // inversion du PortB
     }
}
```

Q1.2. Le compiler et le tester : à l'aide d'une montre, compter le nombre de clignotements en 15 secondes par exemple, et déduisez-en la durée d'un cycle : retrouvez-vous la valeur du tableau ?

Programme de temporisation de 'n' secondes On souhaite maintenant disposer d'une fonction de temporisation permettant de ne rien faire pendant un nombre de secondes donné, qui sera transmis en argument.

```
void tempo( int nb_sec );
```

Q1.3. Enregistrer le programme précédent comme 'tp6_2.c', et le modifier de la façon suivante pour pouvoir tester cette fonction :

```
1     while( 1 ) {
2          PORTB = ...;    // allumage del / PB0
3          tempo( 3 );
4          PORTB = ...;    // extinction del / PB0
5          tempo( 1 );
6     }
```

Dans un premier temps, on pourra considérer que 3 x 349 ms = 1s. La fonction devra donc effectuer 'n' attentes et réinitialisations de TOF, 'n' étant calculé comme 3 fois l'argument transmis.

Q1.4. Ecrire la définition de la fonction (voir cours), et tester le programme.

2 Utilisation des comparateurs

2.1 Principe

Les huit comparateurs du timer peuvent être utilisés pour générer des délais "petits" devant le cycle complet de TCNT (Tmax). Le principe repose sur 4 étapes :

1. lecture de la valeur actuelle de TCNT ;
2. écriture de cette valeur dans TCx, additionnée d'une quantité correspondant au nombre de cycles désirés ;
3. RAZ "initiale" du flag CxF ;
4. Attente de l'activation du flag.

Ceci est illustré par la fig. 1. Par exemple, pour réaliser un délai de 1 ms :

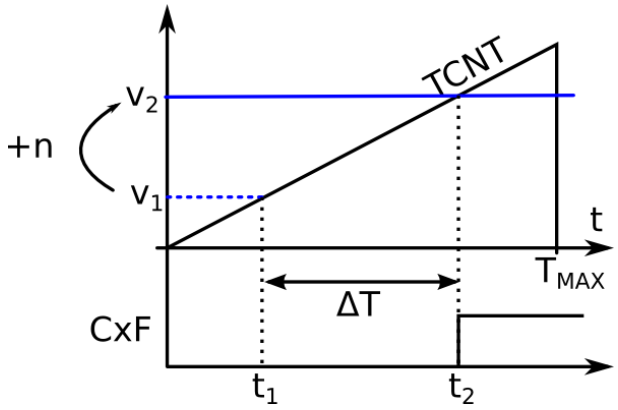

FIGURE 1 – Génération d'une temporisation avec les comparateurs du Timer.

1. On règle le prédiviseur sur un facteur de division R=4 ⇒ $fe = 24/4 = 6$ MHz ⇒ TCNT s'incrémente toute les 166,7 ns (1 / 6 MHz)
2. On calcule combien de fois il y a 166 ns dans 1 ms : N = 1ms / 166 ns = 6000
3. On initialise : on vient lire à un instant t2 la valeur actuelle de TCNT (N1). On écrit cette valeur augmentée de 6000 dans le registre TCx
4. On attend l'activation du flag CxF : celle-ci se produira 6000 cycles plus tard, soit à l'instant t3 = t2 + 1ms.

Ceci peut s'implémenter par exemple sous la forme d'une fonction `tempo_1ms()`, qui utilise le comparateur n°0 (on suppose le rapport de prédivision déjà correctement sélectionné) :

```
1  void tempo_1ms( void )
2  {
3       TC0 = TCNT + 6000;
4       TFLG1 = TFLG1 | BIT0;          // RAZ initiale du flag
5       while( (TFLG1 & BIT0) == 0)    // attente flag
6            ;
7  }
```

2.2 Génération d'un signal sonore

Nous allons utiliser ce principe pour générer un signal sonore à 1000 Hz avec le buzzer connecté sur la broche PT2. Plutôt que de manipuler directement la broche, on pourra configurer le Timer pour que celle-ci se modifie automatiquement à chaque activation du flag du comparateur : 2 registres (TCTL1 et TCTL2)

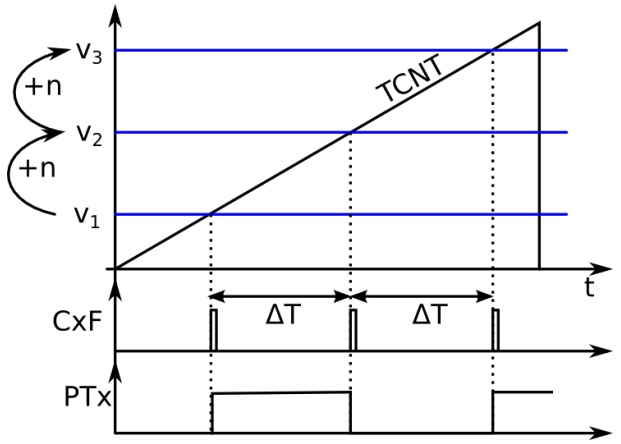

FIGURE 2 – Génération d'un signal sur une broche du Timer.

permettent de spécifier ce qui se passe sur la broche correspondante lorsque le flag passe à 1 (voir cours).

Le plus simple pour générer un signal rectangulaire sera de programmer l'**inversion automatique** de la broche de sortie toutes les 500 μs. pour avoir un signal à 1000 Hz. Comme ce délai sera répété, il devient inutile de procéder à l'initialisation du registre TCx avec la valeur de TCNT : on va reboucler en permanence sur les trois étapes suivantes :

1. attendre l'activation du flag CxF
2. Le ré-initialiser
3. Additionner dans TCx la valeur lue avec le nombre de cycles désirés

La figure 2 illustre ce mécanisme.

Q2.2.1. Avec le rapport de prédivision R=4, déterminer le nombre de cycles correspondant à une demi-période du signal désiré : _____

Q2.2.2. Saisir le programme suivant, l'enregistrer comme 'tp6_3.c', le compléter, et le tester.

```
1  #include <sys/ports.h>
2  #include "bitmask.h"
3  int main()
4  {
5  // A - initialisations
6       TSCR1 |= BIT7; // Activation du Timer
```

```
7       TSCR2 = _____;  // sélection rapport prescaler
8       TCTL1 = _____;  // sélection du mode
9       TCTL2 = _____;  //  de la broche de sortie
10      TIOS  = _____;  // canal concerné en mode "sortie"
11  // B - boucle
12      while( 1 )
13      {
14          while( (TFLG1 & BIT_) == 0 )  // attente
15              ;                         // du flag
16          TFLG1 = ....;                 // RAZ du flag
17          TC_ = TC_ + _____;            // addition
18      }
19  }
```

2.3 Génération d'un signal sonore wobulé

Dans le programme précédent, on vient à chaque activation du flag additionner une constante dans TC2. Or, rien n'interdit d'y additionner une variable et de faire varier celle-ci. On pourra ainsi faire varier **continuellement** la fréquence du signal généré en faisant varier cette variable.

A chaque activation du flag, on pourra incrémenter ou décrémenter cette variable, (qui pourra judicieusement s'appeler `demiper`) Il faudra cependant prévoir une variable `int sens`, qui permettra de déterminer s'il faut incrémenter `demiper` (si `sens` est à 1) ou décrémenter `demiper` (si `sens` est à 0).

Q2.3.1. Enregistrer le programme précédent comme 'tp6_4.c', et le modifier conformément à l'algorigramme de la fig. 3. On y a ajouté des tests permettant de faire basculer la variable `sens` si on est arrivé aux valeurs maximales et minimales de demiper. On définira à cet effet deux constantes symboliques `DP_MIN` et `DP_MAX` (avec la directive `#define`).

On pourra prendre 8 comme rapport de prédivision, ce qui donne une fréquence pour l'horloge de 3 MHz, et un "pas" entre 2 itérations de TCNT $t_0 = 333$ns. Proposer des valeurs pour `DP_MAX` et `DP_MIN` permettant de faire varier la fréquence du signal entre 800 Hz et 3 kHz.

Calcul de dp, nombre de cycles d'horloge correspondant à une demi période :

$$dp = \frac{1}{2}\frac{T}{t_0} = \frac{1}{2}\frac{1/f}{R/f_{CLK}} = \frac{f_{CLK}}{2\,R\,\cdot f}$$

Avec :
- $t_0 = R/f_{CLK}$: période du signal d'horloge de TCNT,
- R : facteur de prédivision choisi,
- $f = 1/T$: fréquence désirée.

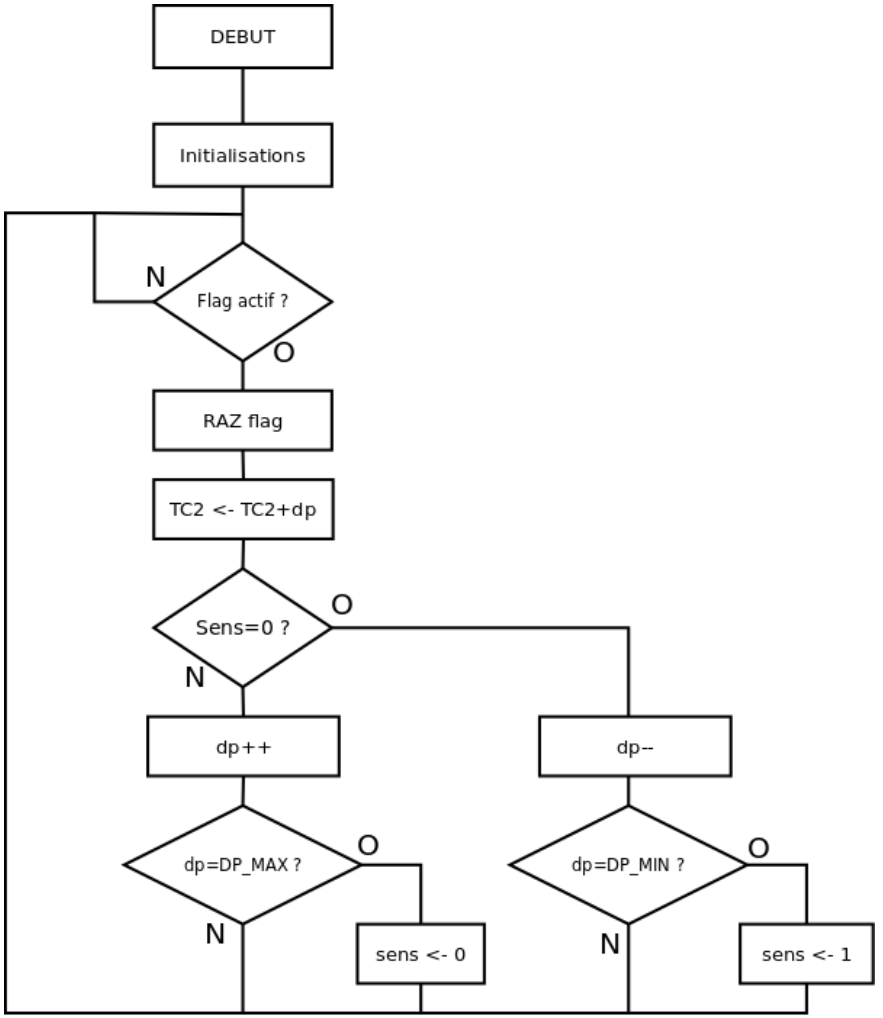

FIGURE 3 – Algorigramme pour générer un signal wobulé.

2.4 Génération d'un signal sonore intermittent

Proposer un programme qui va générer un signal sonore intermittent (« bip-bip »). On demande f=1kHz, $t_{ON} = t_{OFF} = 500$ ms.

Principe : on réalise un **comptage** des activations du flag C2F, et dès qu'on atteint le nombre de valeurs correspondant à 500ms, on désactive la sortie via TCTL1/TCTL2. On recommence ensuite le cycle, pour la réactiver au bout de 500ms. En pratique, il faudra inverser à chaque fois le bit gérant la sortie (voir les bits OMx/OLx dans TCTL1/TCTL2).