Le Timer du processeur 9s12

1 Introduction

Un timer est un **bloc fonctionnel** dont sont dotés tous les microcontroleurs et qui sert :

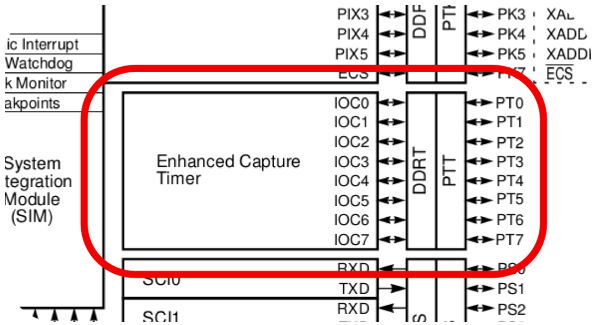
- pour la **mesure** de temps (écart entre 2 fronts d'un signal extérieur). (non traité ici)
- pour la **génération** de temps :
  - génération de signaux électriques calibrés en temps,
  - génération de délais (temporisations).

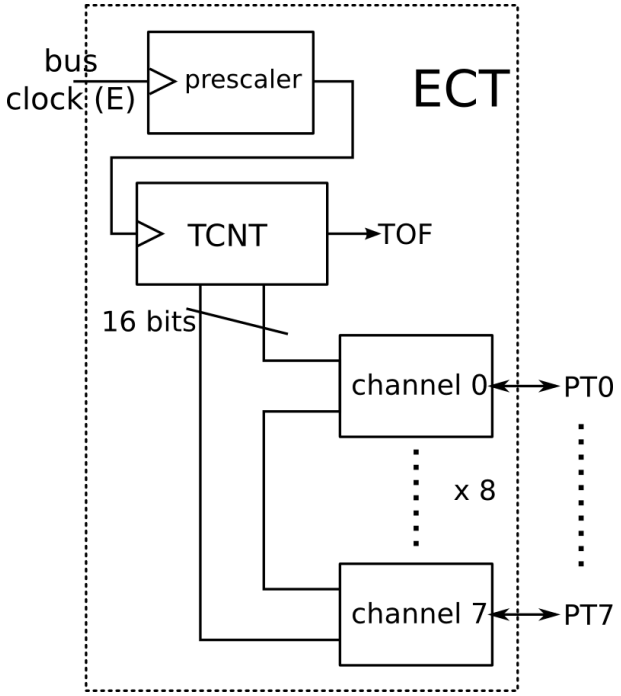
1.1 Architecture du timer du 9s12

Le timer du 9s12 est nommé *Enhanced Capture Timer* (ECT). Il est doté de broches de sorties, qui sont partagées avec le port `PTT` (de type GPIO). Il est composé en interne :

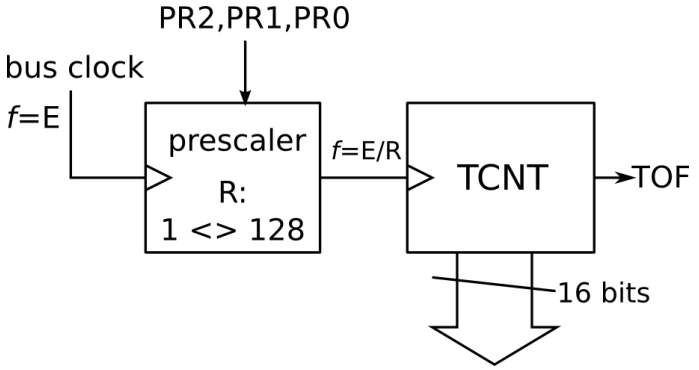
- d'un compteur 16 bits TCNT, piloté par l'horloge interne,
- de 8 canaux internes indépendants. Ces 8 canaux peuvent être utilisés de façon indépendantes, soit en sortie (pour générer des signaux), soit en entrée, pour mesurer des signaux.

Les registres de contrôle associés sont : TCNT, TSCR1, TSCR2, TFLG1, TFLG2, TIOS, TCTL1, TCTL2

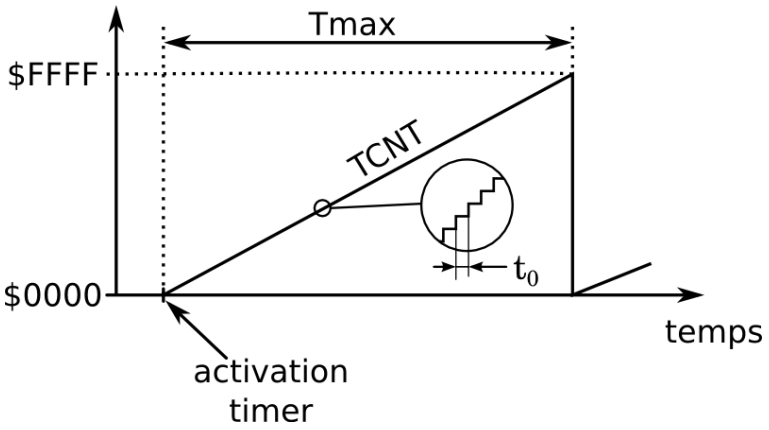




Le compteur 16 bits est accessible directement via le registre TCNT. Son signal d'horloge est issu de l'horloge système, via un prédiviseur de fréquence (*prescaler*), qui sert à pouvoir adapter la cadence de fonctionnement au cahier des charges désiré.



L'évolution temporelle de ce compteur est réprésentée ci-dessous :



Les temps associés seront :

— $t_0 = R/E$ (E : *bus clock*, 24 MHz avec la carte de TP HCS12T)

— $T_{\text{MAX}} = 2^{16} \cdot t_0 = 65536\dfrac{R}{E}$

Le registre **TSCR2** (*Timer System Control Register 2*) permet de spécifier le facteur de division de fréquence R, via le tableau ci-dessous.

	BIT7	6	5	4	3	2	1	BIT0
R	TOI	0	0	0	TCRE	PR2	PR1	PR0
W								
RESET:	0	0	0	0	0	0	0	0

PR2	PR1	PR0	R
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Par défaut le timer est arrêté (économie d'énergie). Avant toute utilisation, On l'activera via le bit TEN (Timer Enable) du registre **TSCR1**. Ceci est à prévoir dans la partie "initialisation" du programme.

	BIT7	6	5	4	3	2	1	BIT0
R	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

— TEN = 1 : timer activé

— TEN = 0 : (défaut) timer désactivé (arrêté)

Le compteur TCNT est doté d'un flag de débordement TOF (*Timer Overflow Flag*), qui est activé (passe à 1) **à chaque** passage de $FFFF à $0000. Ce flag est accessible dans le registre TFLG2 :

	BIT7	6	5	4	3	2	1	BIT0
R	TOF	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

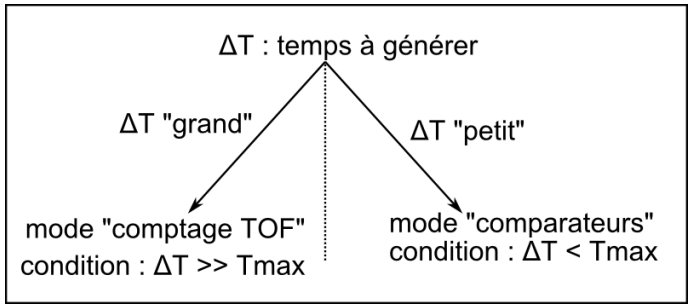
La RAZ de ce flag se fait en écrivant un '1' par dessus : `TFLG2 = 0x80;`

1.2 Utilisation pratique

Le Timer peut s'utiliser de **deux** façons différentes, selon le temps ΔT à générer :

— pour des temps "élevés", on va **compter** les activations du flag TOF.

— pour des temps "faibles", on va utiliser l'un des 8 canaux disponibles.



Il existe une zone de valeurs pour lesquelles on pourra utiliser les deux modes. Le choix adéquat du facteur de division de fréquence R permet de se positionner clairement d'un coté ou de l'autre, mais il est commun pour tout le timer.

1.3 Tableau de sélection de R

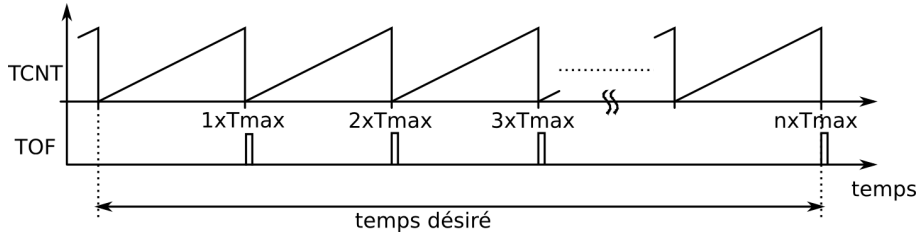
Ce tableau permettra de fixer R en fonction du temps à générer. Il n'est valable que pour une fréquence d'horloge donnée (E=24 MHz).

R	t_0	T_{MAX}
1 2	41,7 ns	2,73 ms
4 8		
16 32		
64 128		

2 Utilisation en comptage

2.1 Principe

On utilise les intervalles de temps T_{MAX} comme des "briques" à partir desquelles on construit le temps désiré.

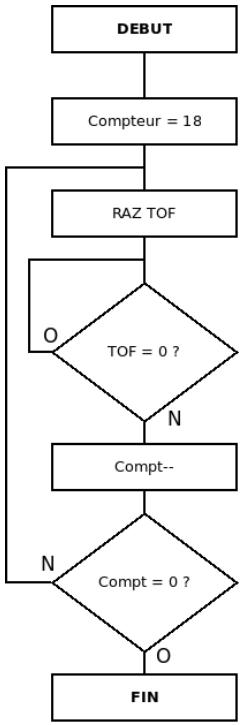


On ne pourra cependant compter que des "briques" entières : il peut y avoir un problème de précision pour de faibles valeurs de comptage.

2.2 Exemple de mise en oeuvre

On souhaite écrire une fonction de tempo d'une durée de 200 ms. On commence par examiner le tableau de sélection pour choisir un facteur de division de fréquence. Il faut avoir TMAX < 200ms. Afin d'avoir une précision correcte, on choisit R=4, ce qui correspond à TMAX=10.9 ms.

Pour avoir T=200ms, if faut attendre 'n' cycles de 10,9ms :
n = 200 / 10,9 = 18
⇒ Il faudra compter 18 activations du flag TOF (18 "attentes" de son activation), puis sortir. L'algorigramme correspondant est ci-dessous.




```
void tempo_200ms(void)
{
  int compt = 18;
  while( compt != 0 )
  {
     TFLG2 = BIT7; // RAZ
     while( TFLG2 == 0 )
       ;
     compt--;
  }
}
```

Remarque : on fait la RAZ de TOF **avant** l'attente, de façon à garantir que le flag est bien à 0 avant d'attendre qu'il passe à 1.

3 Utilisation des comparateurs

Chaque canal du timer peut fonctionner en deux modes :

— en génération de temps, via un **comparateur** binaire,
— en capture de temps, via un **verrou** binaire (*Latch* en anglais).

La sélection entre ces deux modes se fait pour chaque canal via un bit dans le registre TIOS (*Timer Input capture / Output compare Select register*).

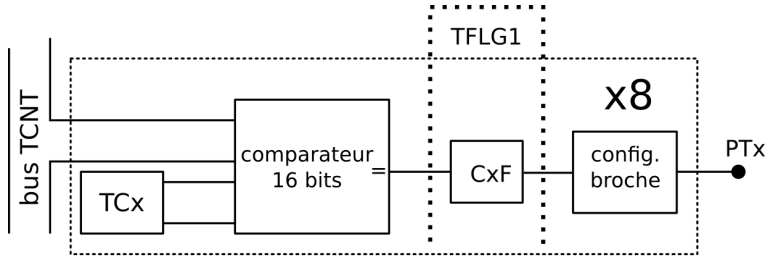
	BIT7	6	5	4	3	2	1	BIT0
R W	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
RESET:	0	0	0	0	0	0	0	0

Chacun de ces bits correspond à une broche du Timer

— 1 : broche en mode "Output Compare" (comparateur),
— 0 : broche en mode "Input Capture" (pas traité ici).

3.1 Principe de fonctionnement des 8 comparateurs

Chaque comparateur a 2 entrées 16 bits et 1 sortie (1 bit) et effectue la comparaison entre la valeur de TCNT et la valeur (fixe) d'un registre 16 bits associé, nommé TCx (TC0, TC1, ... TC7).



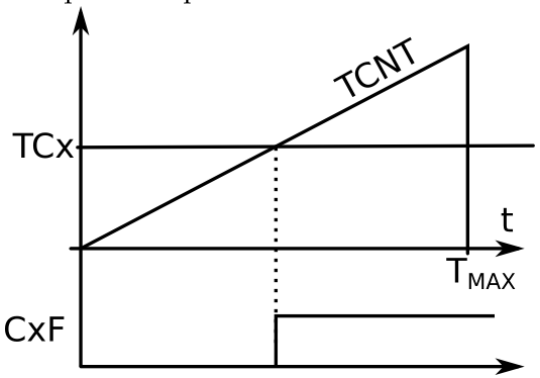
Dès qu'il y a égalité entre les deux valeurs, le flag CxF passe à 1, et reste à 1. Ce flag pourra générer une action sur la broche de sortie de façon automatique.
Le registre TFLG1 (Main Timer Interrupt Flag 1) contient les 8 flags de chaque canal (CxF) :

	BIT7	6	5	4	3	2	1	BIT0
R W	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
RESET:	0	0	0	0	0	0	0	0

Ces *flags* sont remis à zéro par écriture d'un 1 par dessus. Par exemple (pour le bit 3) :

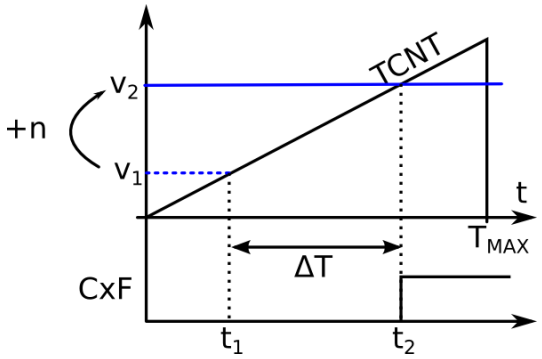
```
TFLG1 = TFLG1 | BIT3;
```

Dès que le compteur TCNT atteint la valeur de TCx, le flag est activé :



Ces comparateurs permettrons de générer des délais de valeur strictement inférieure à T_{MAX}.

3.2 Principe de génération de temps



On va **associer** le temps à une valeur numérique, grace à la pente de TCNT.

- A $t = t_1$, on vient lire la valeur courante de TCNT, on lui ajoute le nombre de cycle désiré n, et on enregistre cette valeur dans TCx.
- A $t = t_2$, le flag CxF est activé : il s'est écoulé un temps Δt, **proportionel** à n.

3.3 Exemple pratique

On souhaite une fonction de temporisation de 1 ms. Dans le tableau de sélection (voir p. 2), il faut choisir une valeur telle que TMAX<1ms. On choisit R=8, ce qui nous donne une fréquence de base f_0 de 24MHz / 8 = 3 MHz.

TCNT s'incrémente donc toutes les 333ns (1/3MHz). On calcule combien de fois il y a 333ns dans 1ms :
1ms / 333ns (=1ms x 3MHz) = 3000

⇒ 1ms correspond donc à 3000 cycles d'horloge, avec **ce** facteur de division de fréquence.

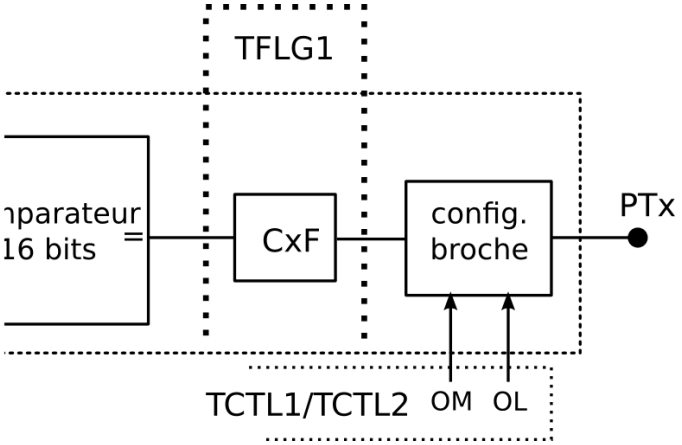
Le code correspondant est ci-dessous, en utilisant le canal 0. Si on veut utiliser le canal 1, il faut remplacer TC0 par TC1, BIT0 par BIT1.

```
void tempo_1ms()
{
  TC0 = TCNT + 3000; /* ecriture dans TC0 de la valeur courante
                        de TCNT + le nombre de cycle désiré */
  TFLG1 = TFLG1 | BIT0; // RAZ initiale
  while( (TFLG1 & BIT0) == 0) // attente de lactivation du flag
    ;
}
```

Remarque : Cette fonction ne s'exécute correctement **que** si le timer a été initialisé dans la fonction `main()`.

3.4 Action automatique sur les broches

La broche de sortie peut-être modifiée automatiquement dès que le flag du canal (CxF) devient actif :



Deux registres permettent de configurer ce qui va se passer lors de l'activation d'un flag CxF :

TCTL1 :

	BIT7	6	5	4	3	2	1	BIT0
R W	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
RESET	0	0	0	0	0	0	0	0

TCTL2 :

	BIT7	6	5	4	3	2	1	BIT0
R W	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
RESET	0	0	0	0	0	0	0	0

Pour chaque broche, deux bits OM et OL spécifient ce qui ce passe au moment où le flag s'active :

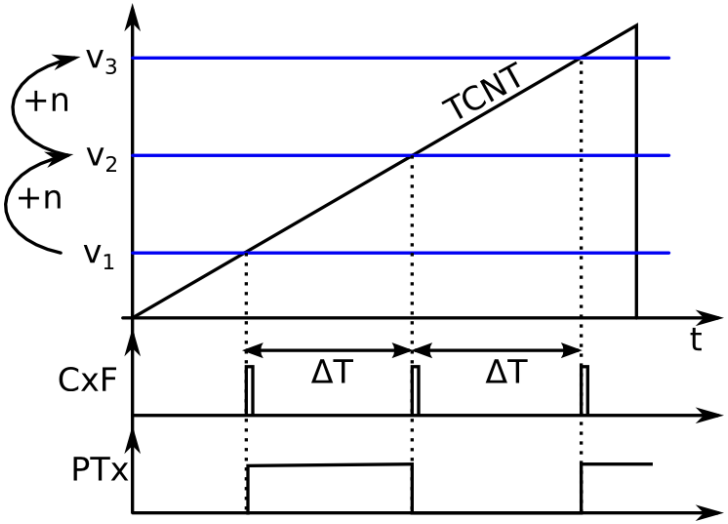
OM	OL	Action
0	0	Timer déconnecté de la broche de sortie
0	1	Inversion de PTx (mode "toggle")
1	0	PTx = 0
1	1	PTx = 1

3.5 Génération de signal périodique

On configure TCTL1/TCTL2 pour avoir le mode "Toggle" : chaque activation du flag va **inverser** la broche de sortie. On doit ensuite répéter dans une boucle infinie les 3 étapes suivantes :

1. Ajout dans TCx du nombre de cycles correspondant au temps désiré
2. RAZ flag CxF
3. Attente de son activation

Ceci est illustré par le chronogramme ci-dessous, sur lequel on peut remarquer que le temps à générer correspond à la **demi-période** du signal.



3.6 Exemple : signal 1kHz sur la broche PT0

On choisit ici R=8.

```
int main()
{
// initialisations
  TSCR1 =
  TSCR2 =
  TIOS =
  TCTL1 =
  TCTL2 =
```

```
// boucle infinie
  while( 1 )
  {
    while( (TFLG1 & BIT0) == 0 ) // attente flag
      ;
    TFLG1 = TFLG1 | BIT0; // RAZ flag
    TC0 = TC0 +             ; // addition
  }
}
```

Exercice : compléter le listing ci-dessus de façon à avoir effectivement un signal à 1kHz.

4 Cas pratique

Il arrive qu'on doive surveiller à la fois l'écoulement du temps et un autre évènement. Par exemple, un cahier des charges peut spécifier : *"Allumer une del si l'utilisateur n'a pas appuyé sur un BP au bout de 2s."*

Il faudra alors dans la boucle surveiller les deux évènemenents, comme dans le pseudo-code ci-dessous :

```
faire
{
   // incrementer un compteur à chaque Tmax
   // SI compt = 2 s, alors allumer la del et fin
   // SI appui sur BP, alors fin
 }
 tant que ( pas fini )
```

L'implémentation correspondante est ci-dessous. On peut voir que la structure commence à être complexe, mais la programmation en interruption permettra de simplifier la gestion de ce type de spécification.

```
  int encore = 1; // flag
  int compt = 0; // compteur de Tmax
  TFLG2 = 0x80;  // RAZ TOF
  do
  {
    if( TFLG2 == 0x80 )
    {
      TFLG2 = 0x80; // RAZ TOF
      compt++;
    }
    if( compt == NBCYCLES_2S )
    {
      encore = 0;
      PORTB = ...  // Del ON
    }
    if( /* appui sur le  BP */ )
      encore = 0;
  }
  while( encore );
 }
}
```