Le Timer du 9s12

Module Info 2

Sebastien.Kramm@univ-rouen.fr

IUT GEII Rouen

2013-2014

Sommaire

A quoi sert un timer

Un timer est un **bloc fonctionnel** qui sert

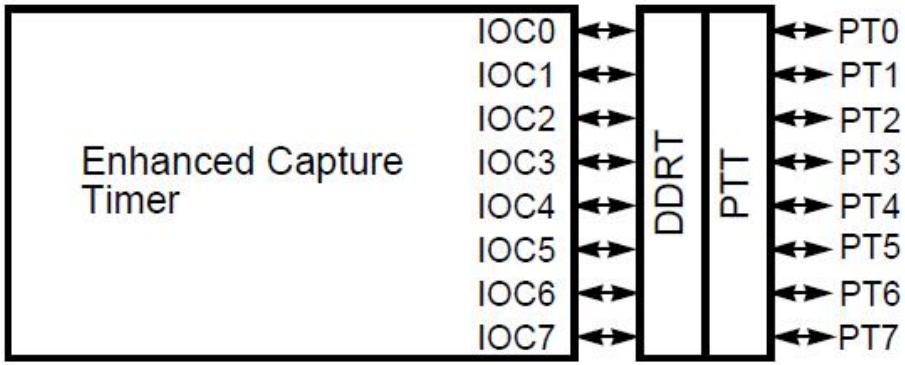
- pour la **mesure** de temps (écart entre 2 fronts d'un signal extérieur). (non traité ici)
- pour la **génération** de temps :
  - génération de signaux électriques calibrés en temps,
  - génération de délais (temporisations).

A quoi sert un timer

Un timer est un **bloc fonctionnel** qui sert

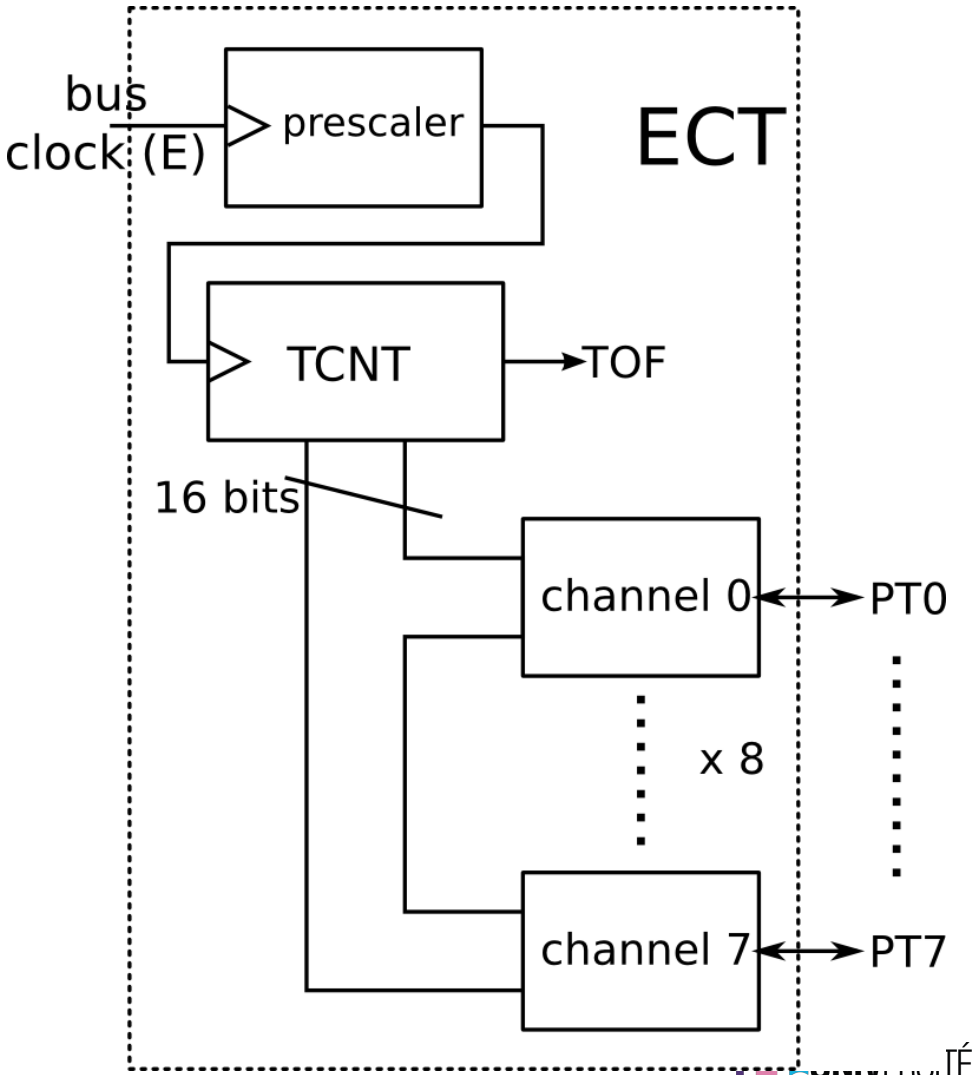
- pour la **mesure** de temps (écart entre 2 fronts d'un signal extérieur). (non traité ici)
- pour la **génération** de temps :
  - génération de signaux électriques calibrés en temps,
  - génération de délais (temporisations).

- Tous les microcontroleurs en sont dotés.
- Le timer du 9s12 :



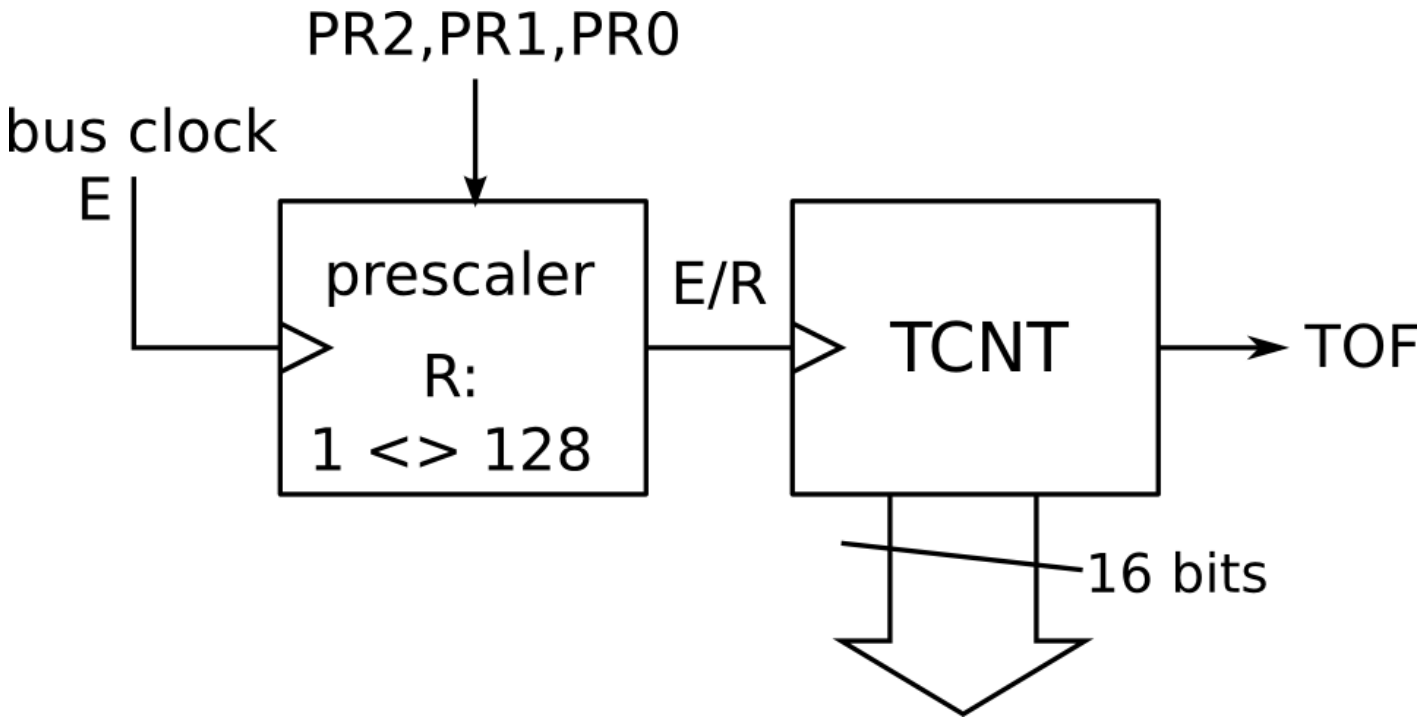
Structure interne

- Le timer du 9s12 est composé :
  - d'un compteur 16 bits TCNT, piloté par l'horloge interne,
  - de 8 canaux internes indépendants, pouvant être utilisés :
    - en sortie (génération de signaux),
    - en entrée (mesure de signaux).

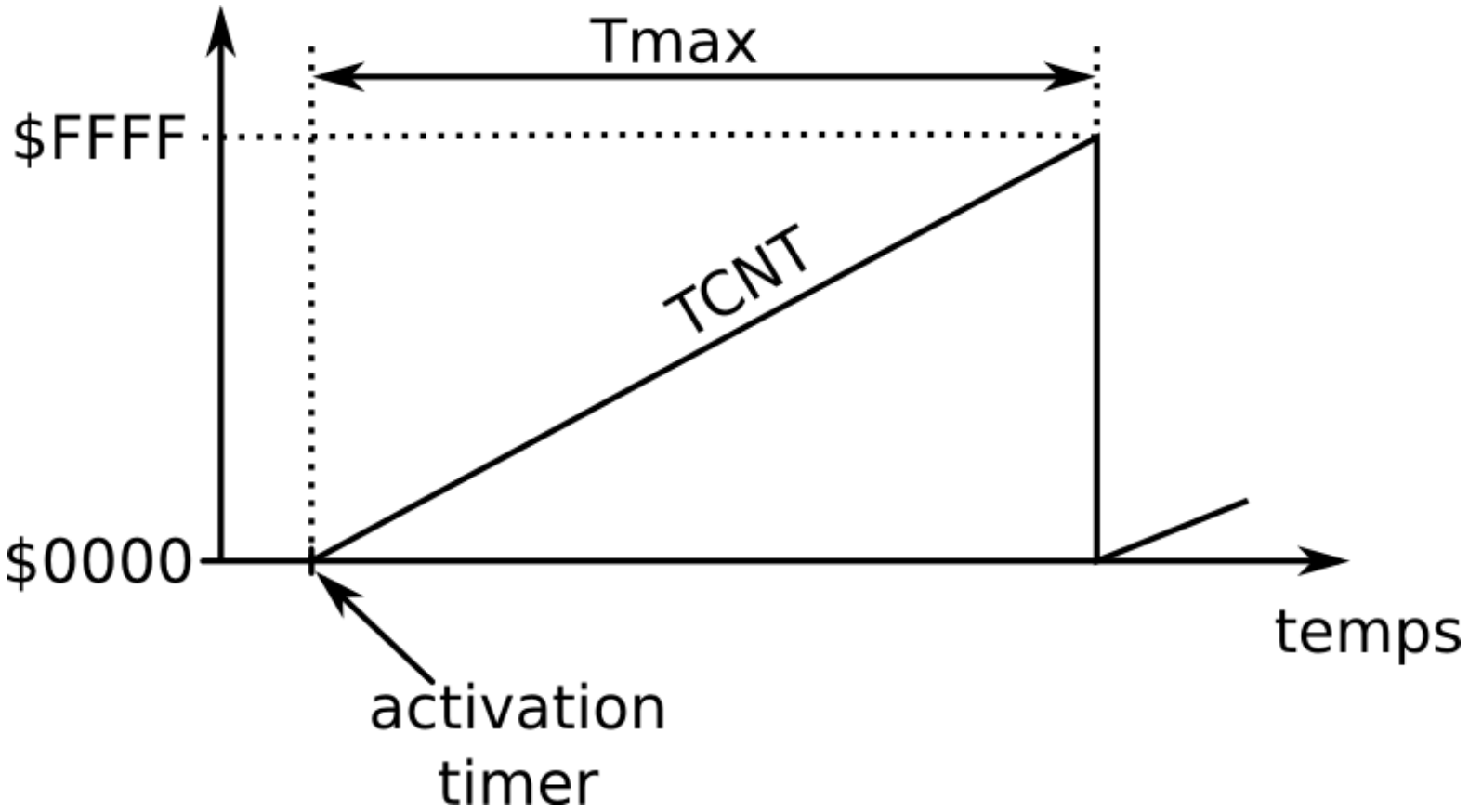


Référence de temps

- Basé sur un compteur 16 bits, accessible via le registre TCNT.
- Signal d'horloge issu de l'horloge système, via un prédiviseur de fréquence.



Illustration



- $t_0 = R/E$ (E : *bus clock*, 24 MHz avec la carte de TP HCS12T)
- $T_{MAX} = 2^{16}.t_0 = 65536.R/E$

Prescaler

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

	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

Activation

- Par défaut le timer est arrêté (économie d'énergie)
- On l'activera via le bit TEN (Timer Enable) du registre TSCR1 (ce qui désactive le port PTT).

	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)
- A prévoir dans l'initialisation.

Flag de débordement

- 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 :

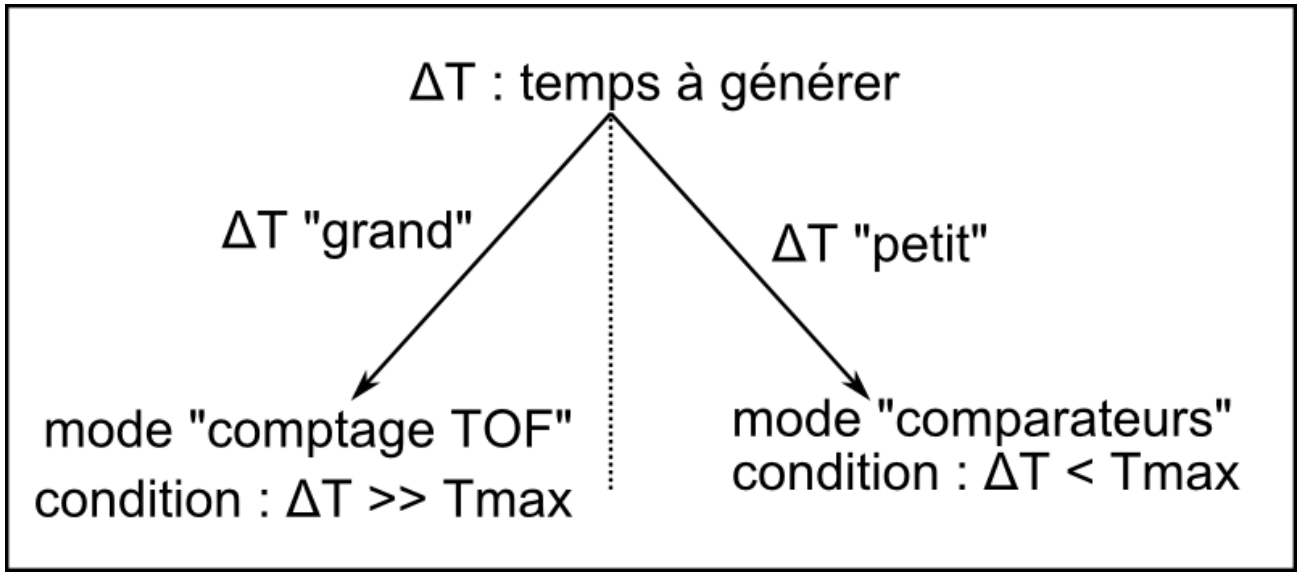
	BIT7	6	5	4	3	2	1	BIT0
R	TOF	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

- la RAZ se fait en écrivant un '1' par dessus :

```
TFLG2 = 0x80;
```

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.

Selection de R

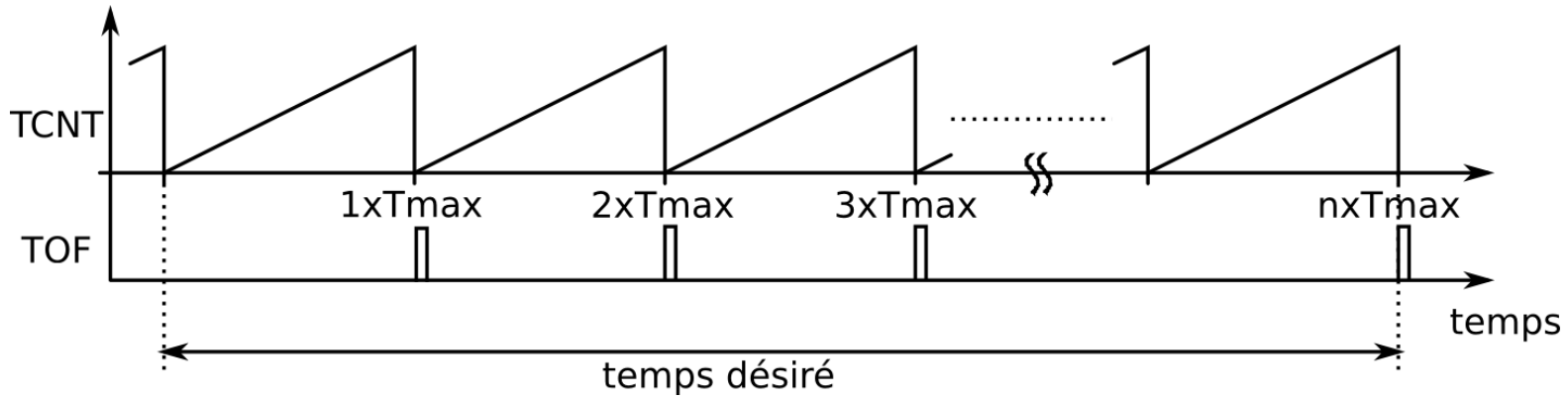
- Tableau de sélection (valable pour E=24 MHz)

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

Sommaire

Principe

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



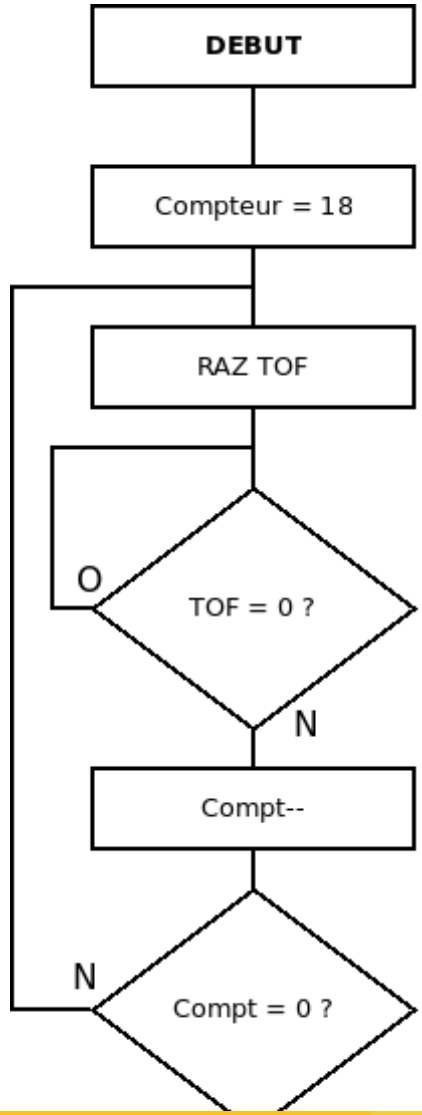
- Attention, on ne pourra compter que des "briques" entières !
  $\Rightarrow$ problème de précision pour de faibles valeurs de comptage.

Exemple de mise en oeuvre

- On souhaite écrire une fonction de tempo d'une durée de 200 ms.
- On a E = 24 MHz, on choisit R=4
  F = 24 MHz / 4 = 6 MHz
- Un cycle complet de TCNT dure :
  T_{MAX} = 65536 * 1/6MHz = 10,9 ms
- pour avoir T=200ms, if faut attendre 'n' cycles de 10,9ms :
  n = 200 / 10,9 = 18
  $\Rightarrow$ on va compter 18 activations du flag TOF.

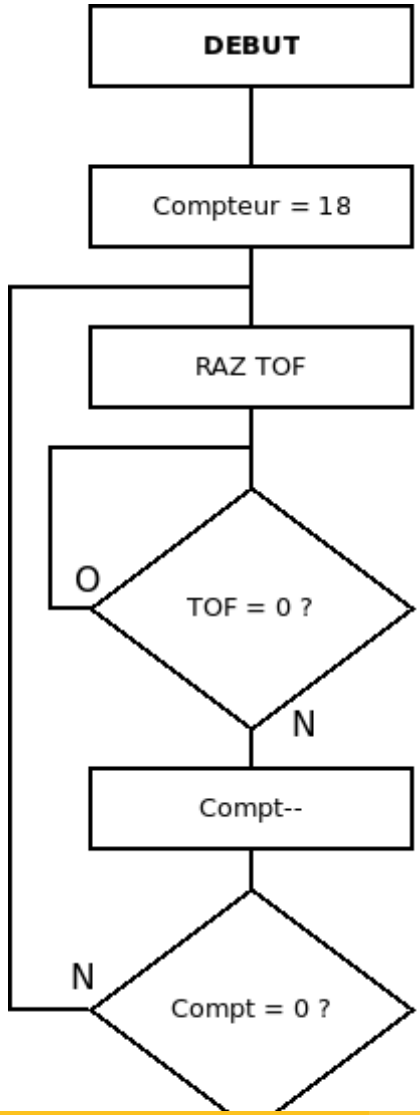
Implémentation pratique

- Algorigramme



Implémentation pratique

- Algorigramme



- Implémentation

```
1  void tempo_200ms(void)
2  {
3    int compt = 18;
4    while( compt != 0 )
5    {
6       TFLG2 = BIT7; // RAZ
7       while( TFLG2 == 0 )
8         ;
9       compt--;
10   }
11 }
```

Remarque : on fait la RAZ de TOF **avant** l'attente.

Sommaire

Mode de fonctionnement des 8 canaux

- 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*).

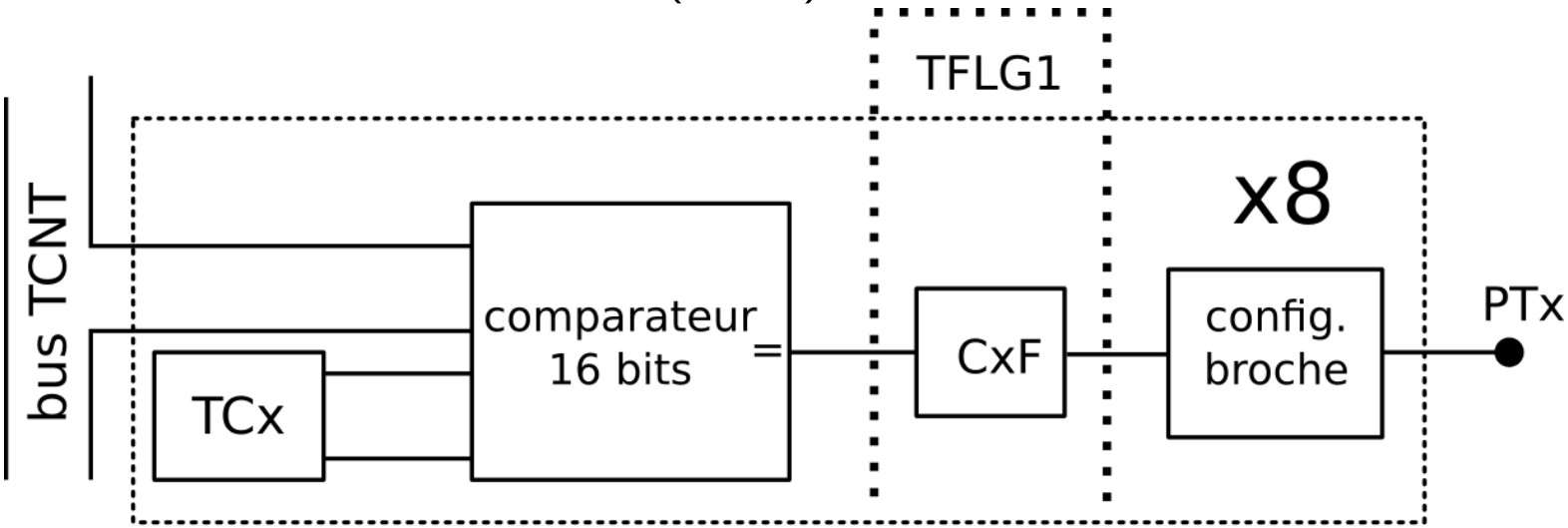
	BIT7	6	5	4	3	2	1	BIT0
R W	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
RESET:	0	0	0	0	0	0	0	0

- 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).

Sous-sommaire

Fonctionnement des 8 comparateurs

- 2 entrées 16 bits, 1 sortie (1 bit)



- Comparaison entre :
  - la valeur de TCNT
  - et la valeur (fixe) du registre 16 bits TCx (TC0, TC1, ... TC7)
- Si égalité, le flag CxF passe à 1, et reste à 1.
- Ce flag pourra générer une action sur la broche de sortie de façon automatique.

Registre TFLG1

- Le registre TFLG1 (Main Timer Interrupt Flag 1) contient les 8 flags de chaque canal (CxF)

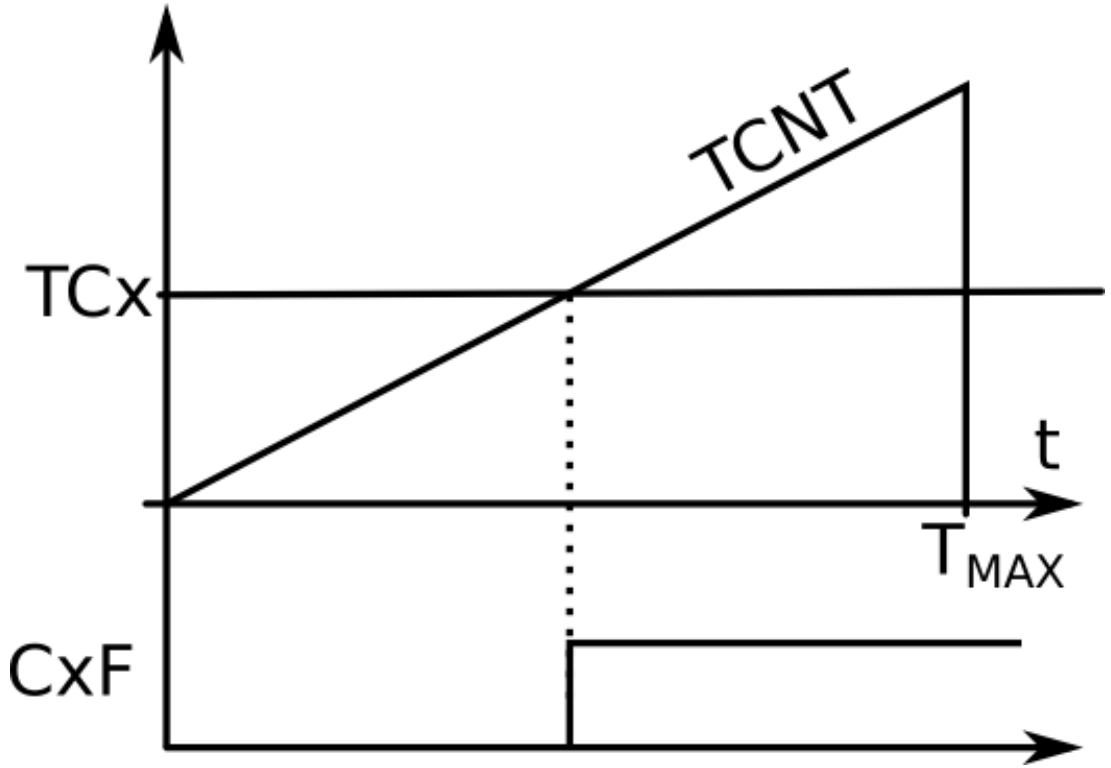
	BIT7	6	5	4	3	2	1	BIT0
R W	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
RESET:	0	0	0	0	0	0	0	0

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

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

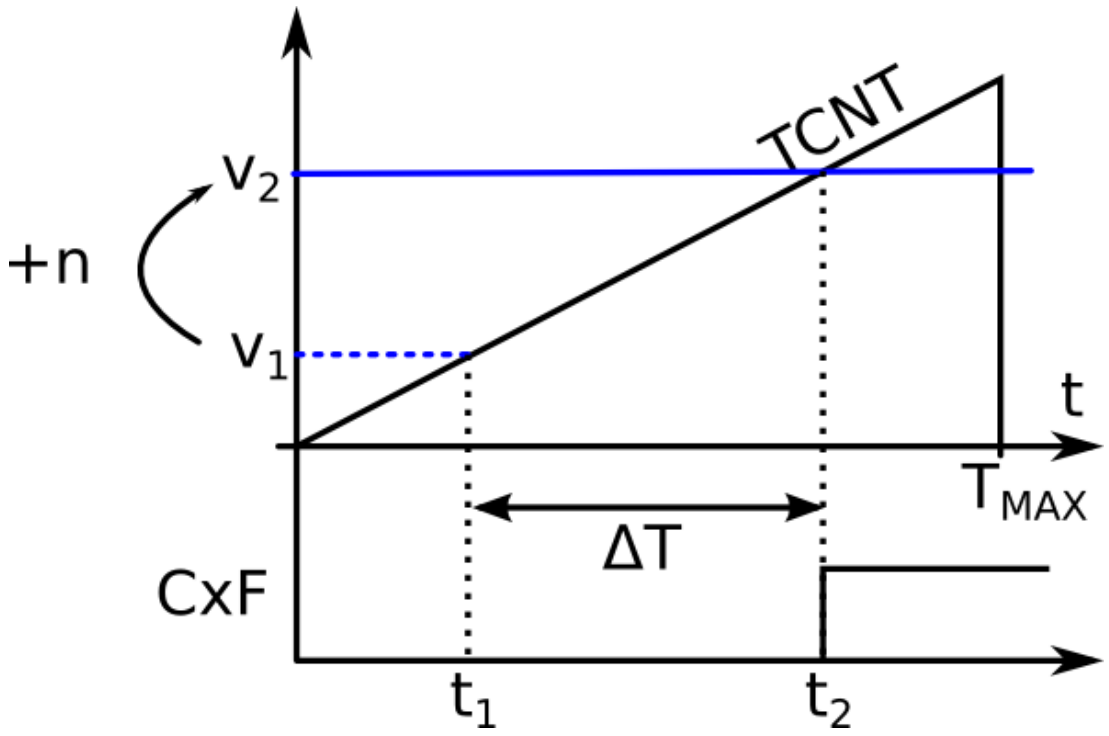
Fonctionnement

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



- On pourra générer des délais de valeur **inférieure** à T_{MAX}.

Principe de génération de temps



- Principe : on **associe** 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.

Sous-sommaire

Exemple 1

- On souhaite une fonction de temporisation de 1 ms.
- Calcul préliminaires :
  - On fixe R à 8
    $\Rightarrow f_0 = 24\text{MHz} / 8 = 3$ MHz
    $\Rightarrow$ TCNT s'incrémente toutes les 333ns (1/3MHz)
  - On calcule combien de fois il y a 333ns dans 1ms :
    1ms / 333ns (=1ms x 3MHz) = 3000

$\Rightarrow$ 1ms correspond à 3000 cycles d'horloge

(avec **ce** facteur de division de fréquence.)

Exemple en utilisant le canal 0

```
1  void tempo_1ms()
2  {
3    TC0 = TCNT + 3000; /* ecriture dans TC0 de la valeur courante
4                          de TCNT + le nombre de cycle désiré */
5    TFLG1 = TFLG1 | BIT0; // RAZ initiale
6    while( (TFLG1 & BIT0) == 0) // attente de l activation du flag
7        ;
8  }
```

Exemple en utilisant le canal 0

```
1  void tempo_1ms()
2  {
3    TC0 = TCNT + 3000; /* ecriture dans TC0 de la valeur courante
4                          de TCNT + le nombre de cycle désiré */
5    TFLG1 = TFLG1 | BIT0; // RAZ initiale
6    while( (TFLG1 & BIT0) == 0) // attente de l activation du flag
7        ;
8  }
```

- Remarque 1 : si on veut utiliser le canal 1, remplacer TC0 par TC1, BIT0 par BIT1.
- Remarque 2 : Cette fonction ne s'exécute correctement **que** si le timer a été initialisé dans la fonction `main()`.

Sous-sommaire

Action sur les broches de sortie

- 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, 2 bits OM et OL :

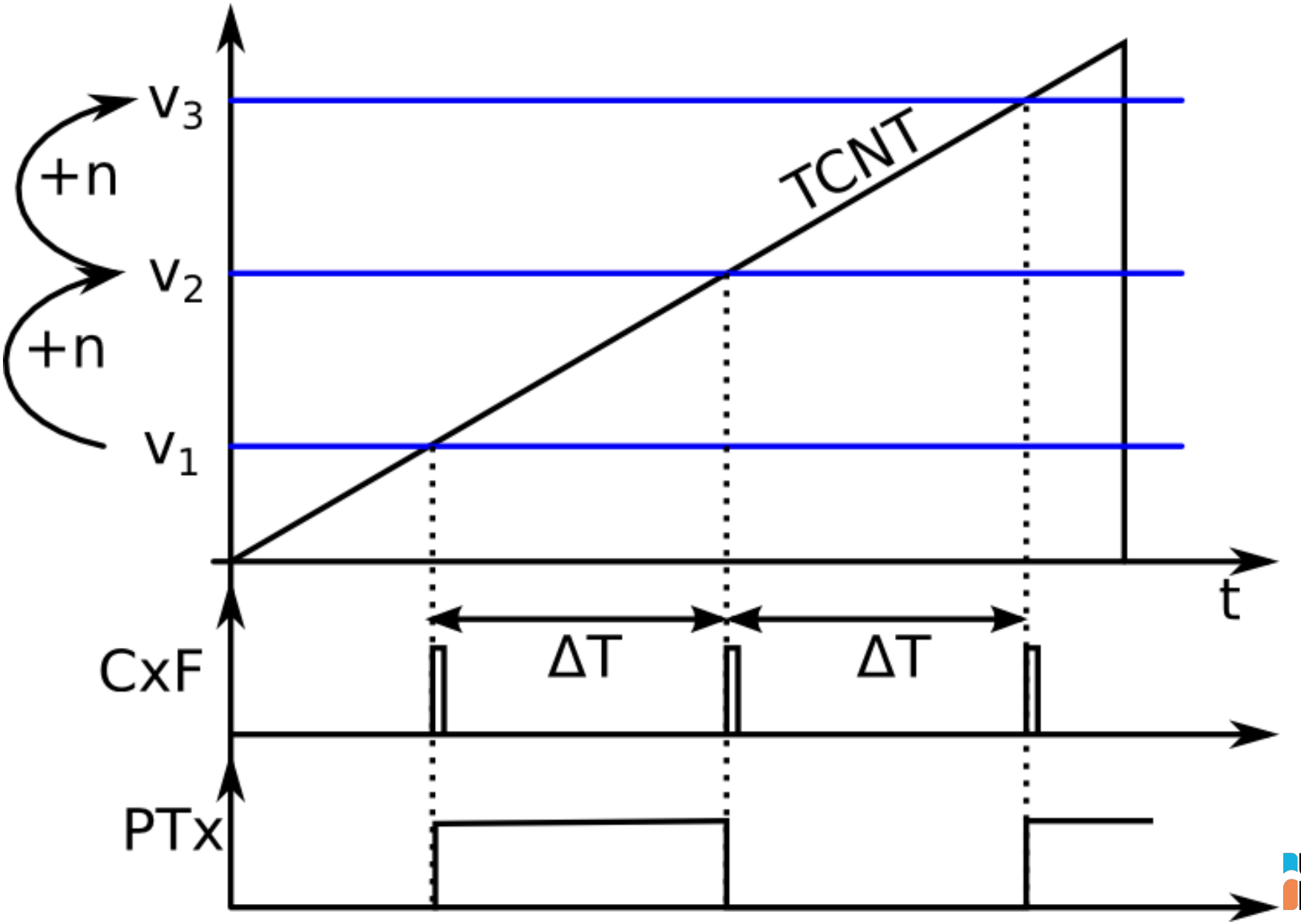
OMx	OLx	Action
0	0	Timer déconnecté de la broche de sortie
0	1	inversion de PTx (mode "Toggle")
1	0	PTx = 0
1	1	PTx = 1

Génération de signal périodique

- Initialisation : on configure TCTL1/TCTL2 pour avoir le mode "Toggle" : chaque activation du flag va **inverser** la broche de sortie.
- On doit répéter dans une boucle infinie les étapes :
  1. Ajout dans TCx du nombre de cycles correspondant au temps désiré
  2. RAZ flag CxF
  3. Attente de son activation

UNIVERSITÉ DE ROUEN

Illustration



Exemple : signal 1kHz sur la broche PT0

- On choisit R=8.
- Le temps à considérer est la **demi-période**.

```
1  int main()
2  {
3  //   initialisations
4    TSCR1 =
5    TSCR2 =
6    TIOS =
7    TCTL1 =
8    TCTL2 =
9
10 // boucle  infinie
11   while( 1 )
12   {
13     while( (TFLG1 & BIT0) == 0 ) // attente flag
14       ;
15     TFLG1 = TFLG1 | BIT0; // RAZ flag
16     TC0 = TC0 +          ; // addition
```

Sommaire

Exemple de cas réel

- Il arrive qu'on doive surveiller à la fois
  - le temps qui passe...,
  - et un autre évènement.
- Par exemple : *"Allumer une del si l'utilisateur n'a pas appuyé sur un BP au bout de 2s."*

Exemple de cas réel

- Il arrive qu'on doive surveiller à la fois
  - le temps qui passe...,
  - et un autre évènement.
- Par exemple : *"Allumer une del si l'utilisateur n'a pas appuyé sur un BP au bout de 2s."*

```
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 )
```

```
1     int encore = 1; // flag
2     int compt = 0; // compteur de Tmax
3     TFLG2 = 0x80;  // RAZ TOF
4     do
5     {
6        if( TFLG2 == 0x80 )
7        {
8           TFLG2 = 0x80; // RAZ TOF
9           compt++;
10       }
11       if( compt == NBCYCLES_2S )
12       {
13          encore = 0;
14          PORTB = ... // Del ON
15       }
16       if( "appui sur le BP" )
17          encore = 0;
18    }
19    while( encore );
20   }
21 }
```