heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Cours - Microinformatique (MUI)

Cours créé par Cédric Bornand, Pierre Favrat et Bertrand Hochet

HEIG-VD février 2016
rev. 0.0

Microinformatique - HEIG-VD

Informations générales

Professeur	Cédric Bornand
Email	cedric.bornand@heig-vd.ch
Téléphone	024 557 27 68
Bureau	Y-Parc, CH-1400 Yverdon-les-Bains

Professeur	Pierre Favrat
Email	pierre.favrat@heig-vd.ch
Téléphone	024 557 27 77
Bureau	Y-Parc, CH-1400 Yverdon-les-Bains Bureau C0.03

Professeur	Bertrand Hochet
Email	bertrand.hochet@heig-vd.ch
Téléphone	024 557 27 68
Bureau	Y-Parc, CH-1400 Yverdon-les-Bains Bureau C0.03

HISTORIQUE - MODIFICATIONS MAJEURES

VERSION	DATE	DESCRIPTION
V 0.0	février 2014	Premier draft
V 1.0	sometime 2014	Version initiale (PFT)

Table des matières

1 Introduction **1**
- 1.1 Place du cours MUI dans le cursus de l'ingénieur . . . 1
- 1.2 Définitions : processeur, microprocesseur, microcontrôleur et système-on-chip . . . 1
- 1.3 Analogie d'un processeur . . . 2
- 1.4 Éléments de systèmes logiques . . . 2

2 GPIO (General Purpose Input Output) **5**
- 2.1 Matériel . . . 5

3 Numération et arithmétique des ordinateurs **7**
- 3.1 Représentation des nombres . . . 7
- 3.2 Opérateurs . . . 7
- 3.3 Exercices . . . 7

4 Interruptions **9**
- 4.1 Processus . . . 9

5 Timers **11**
- 5.1 Principe . . . 11
- 5.2 Cas du MSP430 . . . 12
- 5.3 Timer à usage général : le timer A . . . 13

6 Système d'horloges **17**
- 6.1 Diviseur de fréquences . . . 17

7 CPU **19**
- 7.1 Principe . . . 19

8 Assembleur **21**
- 8.1 Chaine de compilation . . . 21
- 8.2 Debug . . . 21

9 Programmation en C **23**
- 9.1 Mots clés . . . 23

10 Interfaces séries **25**
- 10.1 Notion de pile protocolaire . . . 25

11 AD et DA **27**
- 11.1 AD . . . 27

A Jeux d'instruction du MSP430 **29**
- A.1 Une structure orthogonale . . . 29

Liste des tableaux

1.1 Résultats possibles de fonctions à deux variables 2

Chapitre 1

Introduction

La microinformatique est la science de l'information proche du matériel. Elle s'exprime à l'aide de langages dit de bas niveau tel que l'assembleur et le C. On utilise la microinformatique dans les systèmes embarqués ou dans les ordinateurs lorsque l'on a des contraintes de performances.

1.1 Place du cours MUI dans le cursus de l'ingénieur

Le cours de microinformatique est un cours technologique basé sur des microprocesseurs qui sont le composant principal des microcontrôleurs. Il se situe parmi les autres branches du génie électrique dans la partie de mise en oeuvre (fig. 1.1).

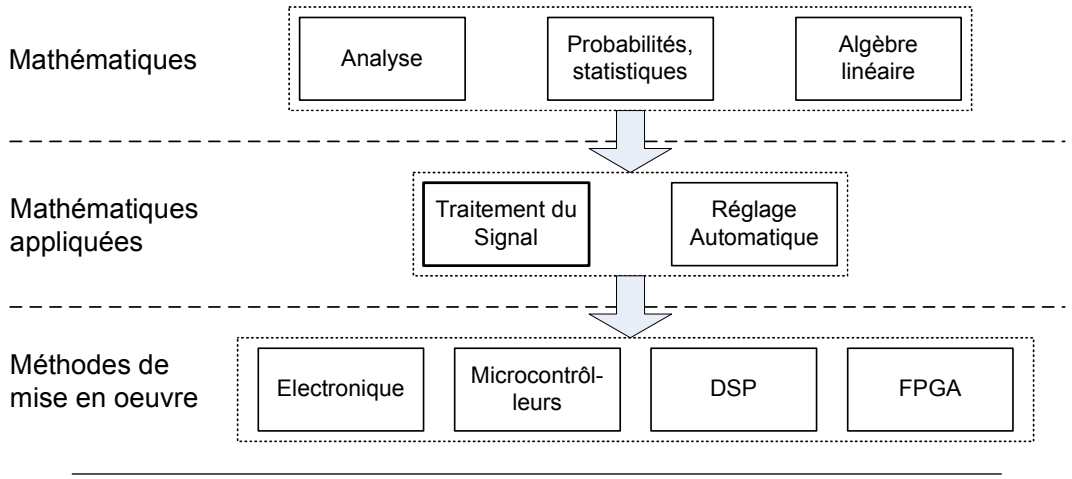

Figure 1.1 – Situation du cours MUI dans le cursus de l'ingénieur

1.2 Définitions : processeur, microprocesseur, microcontrôleur et système-on-chip

Un processeur est un système qui permet l'exécution d'opérations élémentaires tel que des opérations arithmétiques (additions, soustractions, multiplication et division), des opérations logiques (OR, AND, XOR), des tests (égale à, plus petit que, etc.) et des déplacements de données.

Le microprocesseur est un système micro-électronique, aussi appelé circuit intégré ou plus communément "chip" ou "puce", permettant l'exécution d'opérations élémentaires. Contrairement au processeur qui est un concept, le microprocesseur est un composant que l'on place sur un circuit imprimé.

Un microcontrôleur est un système micro-électronique contenant un microprocesseur et des périphériques. Les périphériques essentiels sont les minuteries, les interfaces de communication série et les mémoires (données et instructions).

Le système-on-chip est en ensemble, différent du microcontrôleur, qui inclut tout les composants électroniques d'un ordinateur à part les mémoires. Il se présente sous la forme d'un circuit intégré. La mémoire de données peut dans certain cas être assemblée en dessus du chip pour réduire la taille du système.

1.3 Analogie d'un processeur

Pour expliquer le fonctionnement d'un processeur on peut en faire l'analogie avec un piano mécanique.

1.4 Éléments de systèmes logiques

Pour pouvoir configurer un microcontrôleur de manière efficace, nous avons besoins de quelques éléments de systèmes logiques.

Variables		Fonctions															
A	B	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

$$
\begin{array}{lcllcll}
F0 & = & 0 & F15 & = & \overline{F0} & = 1 \\
F1 & = & A \cdot B & F14 & = & \overline{F1} & = \overline{A \cdot B} \\
F2 & = & A \cdot \overline{B} & F13 & = & \overline{F2} & = \overline{A} + B \\
F3 & = & A & F12 & = & \overline{F3} & = \overline{A} \\
F4 & = & \overline{A} \cdot B & F11 & = & \overline{F4} & = A + \overline{B} \\
F5 & = & B & F10 & = & \overline{F5} & = \overline{B} \\
F6 & = & A \oplus B & F9 & = & \overline{F6} & = \overline{A \oplus B} \\
F7 & = & A + B & F8 & = & \overline{F7} & = \overline{A + B}
\end{array}
$$

TABLE 1.1 – Résultats possibles de fonctions à deux variables

`Théorie et traitement des signaux`
Théorie et traitement des signaux
Théorie et traitement des signaux
Théorie et traitement des signaux

Table Helvetica ou Arial

Variables		Fonctions															
A	B	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Bibliographie

[1] Frédéric de Coulon, *Théorie et traitement des signaux*, Presses polytechniques romandes, 1990

[2] John G. Proakis, Dimitris G. Manolakis : *Digital Signal Processing*, Pearson Prentice Hall, 2007

Chapitre 2

GPIO (General Purpose Input Output)

Bla bla bla

2.1 Matériel

2.1.1 Input

Chapitre 3

Numération et arithmétique des ordinateurs

bla bla bla

3.1 Représentation des nombres

3.1.1 Entiers

3.1.2 Virgule flottante

3.1.3 Virgule fixe

3.2 Opérateurs

3.2.1 Addition

3.2.2 Soustraction

3.2.3 Multiplication

3.3 Exercices

Ex 1

Chapitre 4

Interruptions

Bla bla bla

4.1 Processus

4.1.1 Plouf

Chapitre 5

Timers

Avec le port d'entré/sortie, le timer est le périphérique le plus indispensable du microcontrôleur. En effet, un CPU est très inefficace dès qu'il s'agit de compter le temps, puisqu'il ne peut rien faire d'autre, sous peine de "perdre du temps".

Un timer est donc indispensable dès qu'il s'agit de développer des applications dans lesquelles des contraintes temporelles doivent être respectées, comme :

— génération de délais ou de temps d'attente
— génération de signaux à caractéristiques temporelles définies
— prises de rendez-vous

Le coeur du timer est un *compteur synchrone*. En général, des fonctionnalités y sont ajoutées, qui permettent d'enrichir les possibilités du timer.

5.1 Principe

Le plus souvent, le timer est construit autour d'un *incrémenteur*. C'est un circuit séquentiel synchrone, construit avec un registre de N bits, et un incrémenteur combinatoire (fig. 5.1). La structure détaillée de ce type de circuit séquentiel est étudiée au cours "Systèmes Logiques".

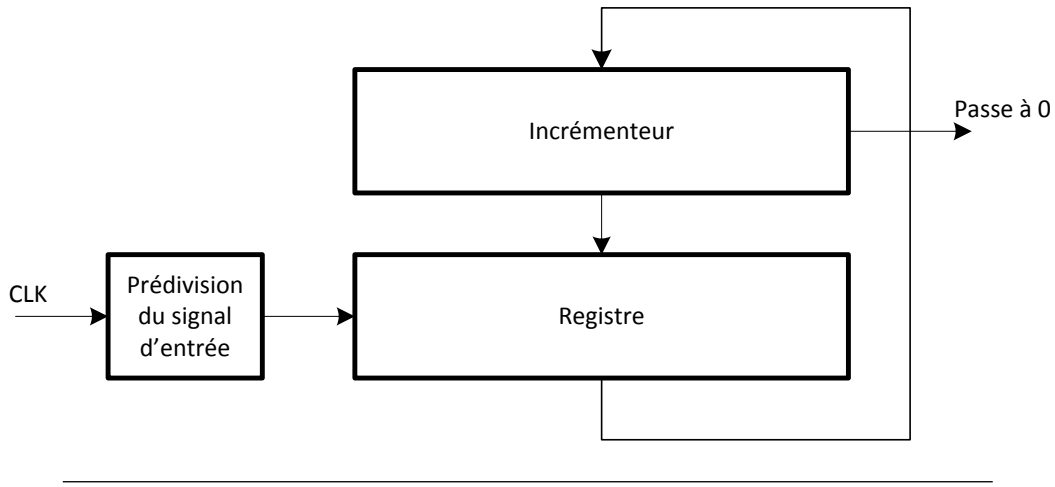

Figure 5.1 – Schéma de base d'un timer

L'incrémentation se fait au rythme d'un signal souvent appelé CLK (fig. 5.1), qui est soit :

— périodique, auquel cas il mesure le temps ;
— apériodique quelconque, auquel cas il compte simplement les flancs de ce signal tant qu'il est actif et on parle plutôt de *compteur*.

La figure 5.2 montre l'évolution de la valeur du registre en fonction du temps, pour un *timer* de 4 bits. La pente de la courbe dépend du rythme auquel le registre est incrémenté, donc de la fréquence du signal CLK. Lorsque le registre atteint sa valeur maximale, égale à $2^N - 1$, il repasse à 0. Durant ce passage à 0, un signal (qui est la retenue sortante du circuit incrémenteur) passe à '1' et émet éventuellement une requête d'interruption.

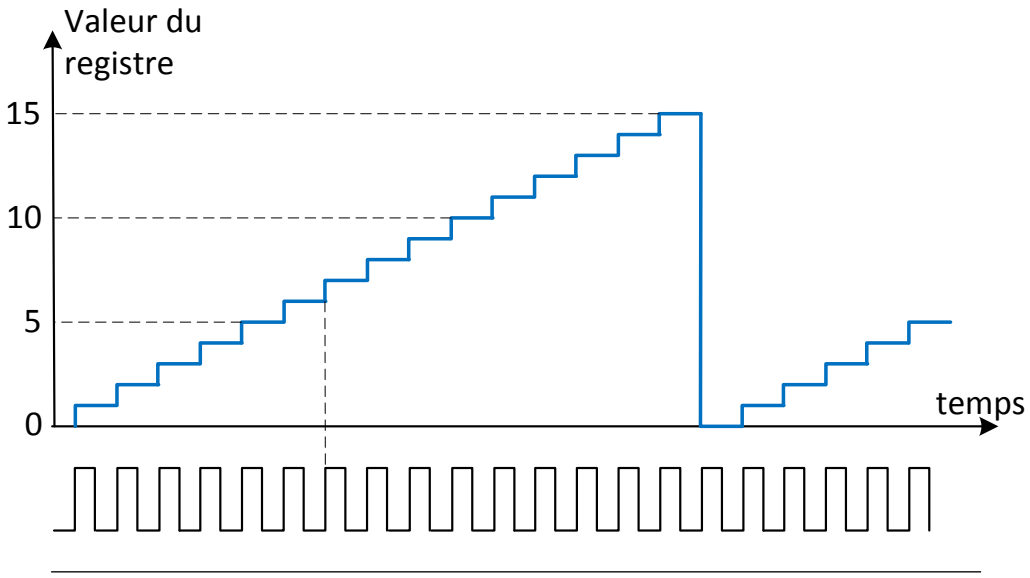

FIGURE 5.2 – Comportement temporel du timer (N=4)

Dans le cas d'un compteur, le signal CLK est apériodique. La figure 5.3 montre l'évolution de la valeur du registre en fonction du temps, pour un *compteur* de 4 bits. De même, la retenue sortante du circuit incrémenteurémet éventuellement une requête d'interruption lors du passage par 0 de la valeur du registre.

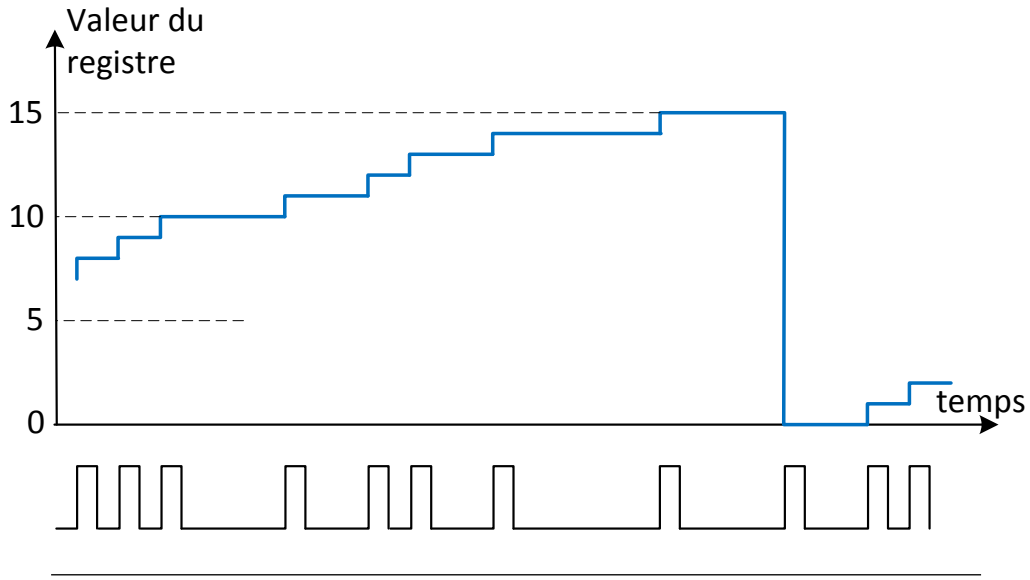

FIGURE 5.3 – Comportement temporel d'un compteur (N=4)

Comme nous le verrons dans la suite, certains timers utilisent un décrémenteur au lieu d'un incrémenteur, voire les deux. Finalement, on peut rencontrer différents types de timers dans un même microcontrôleur, chacun étant adapté à un besoin spécifique.

5.2 Cas du MSP430

Les microcontrôleurs de la famille MSP430 embarquent jusqu'à 4 types de timers différents :

— Real Time Clock, ou *Horloge Temps Réel* (RTC) pour la création d'horloges
— WatchDog Timer. ou *Chien de Garde* (WDT) pour la prévention de plantées d'origines diverses
— Timers à usage général

5.2.1 Real Time Clock

Après initialisation, le RTC maintient à jour les informations horaires les plus courantes :
— année
— numéro du mois dans l'année
— jour du mois
— jour de la semaine
— heure
— minute
— seconde

Ce type de timer nécessite une horloge dont la fréquence est 1Hz, dérivée d'une horloge standard à 32768Hz. Cette dernière fréquence fut introduite aux débuts de la montre à quartz car il est aisé de réaliser des quartz à cette fréquence. On obtient l'horloge à 1Hz par une simple division de 32768 par 2^{15}.

5.2.2 WatchDog Timer

La "plantée" d'un programme est la plupart du temps due à un défaut dans le programme ou à un concours de circonstances qui fait que le programme est bloqué dans un boucle sans fin. Le cas le plus courant est l'attente d'un évènement qui n'arrive pas et qui n'arrivera jamais. Le microcontrôleur ne réagit plus. Le seul moyen est de le *resetter*.

5.3 Timer à usage général : le timer A

Dans le MSP430, le timer à usage général est un périphérique complexe auquel le CPU peut soutraiter des fonctionnalités sophistiquées. En plus des fonctionnalités usuelles, on trouve :
— capture de l'état du registre de comptage lors d'un évènement
— requêtes d'interruption dès que le registre de comptage a atteint une valeur donnée (comparaison)
— génération de signaux PWM sur une patte externe, sans aucune aide du CPU autre que l'initialisation

La figure 5.4 illustre la structure du timer A, en mettant en évidence les registres et les signaux d ?interruption.

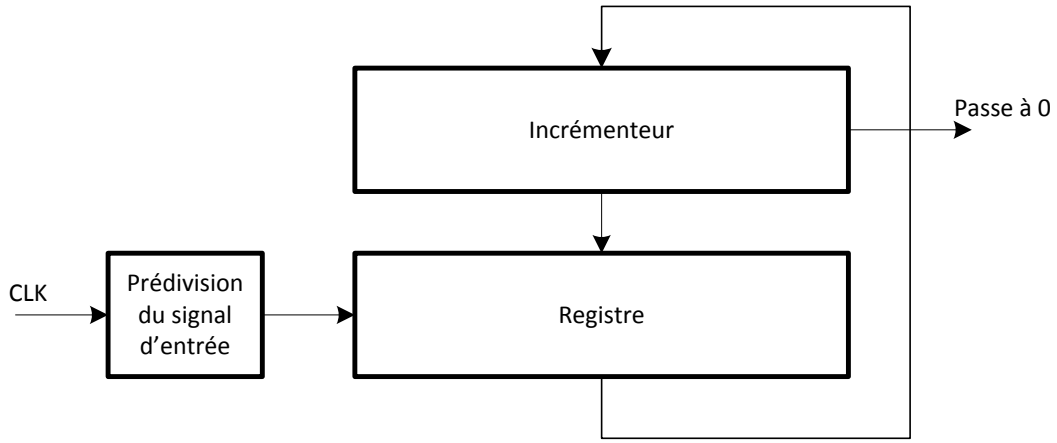

FIGURE 5.4 – Structure du timer A

Le timer est organisé en tranches, correspondant chacune à un bloc fonctionnel. Chaque rectangle représente un « registre de donnée », c ?est à dire un registre qui contient une information représentant une grandeur. Les circuits de comparaison/capture peuvent aussi générer automatiquement des signaux particuliers sur une patte du microcontrôleur. Ces fonctionnalités ne sont pas représentées sur la figure 5.4. A gauche de chaque « registre de donnée » est noté le nom de son registre de contrôle. Tous ces registres sont sur 16 bits.

5.3.1 Bloc de comptage

TAxR : Timer A Register (n° x) ou registre de comptage du timer n° x. Sa valeur évolue entre 0 et 0xFFFF ou le contenu de TAxCCR0.

TAxCTL : Timer A Control (n° x) ou registre de contrôle du comptage du timer n° x. Il permet de spécifier comment TAxR compte. Il est composé de 5 champs, pour contrôler :

- horloge de référence (champ TASSEL)
- diviseur de l?horloge de référence (champ ID)
- contrôle de mode (champ MC), qui définit si le registre de comptage TAxR compte jusqu?à 0xFFFF ou le contenu de TAxCCR0, ou s'il décompte après avoir atteint le contenu de TAxCCR0
- reset du registre de comptage TAxR
- contrôle des interruptions (TAIE) issues du registre de comptage. A l'évidence, ces interruptions ne peuvent être générées qu'au moment particulier où le registre de comptage TAxR est à 0, puisqu'on ne connait pas la valeur maximale que peut prendre le registre de comptage TAxR.

Un dernier champ (TAIFG) contient le flag d'état de l'interruption issue du registre de comptage TAxR.

5.3.2 Bloc de capture/comparaison n°0

TAxCCR0 : registre de comparaison/capture n° 0 pour le Timer A (n° x). Ce bloc fonctionnel est en étroite interaction avec le registre de comptage TAxR, puisqu'il peut (selon la valeur du champ MC) définir la borne supérieure du comptage. Comme TAxR, TAxCCR0 est un « registre de donnée » ; leurs valeurs peuvent être comparées (mode comparaison) ou le contenu de TAxR peut être copié dans TAxCCR0 (mode capture). Lors de ces deux événements (égalité des deux registres ou transfert de TAxR dans TAxCCR0), une requête d'interruption est émise; c'est le signal appelé TAxCCR0_CCIFG, que nous nommerons CCIFG0. Ce circuit de capture/compare, centré autour du registre TAxCCR0, est contrôlé par le registre de contrôle TAxCCTL0.

TAxCCTL0 : registre de contrôle du circuit de capture/compare n° 0 pour le Timer A (n° x). Il permet de spécifier comment TAxCCR0 se comporte. Il est composé de 6 champs, pour contrôler :

- sélection de la fonction Capture ou Comparaison (CAP)
- mode de capture (sur flanc montant, descendant, etc...) (CM)
- sélection du signal de capture (CCIS)
- synchronisation ou non du signal de capture avec l'horloge de comptage (SCS)
- mode de sortie (OUTMOD)
- autorisation des interruptions émises par le bloc de capture comparaison n° 0 (CCIE)

D'autres champs contiennent différents signaux tels que le flag de l'interruption issue du bloc (CCIFG0).

5.3.3 Bloc de capture/comparaison n°1,2,3...

Ces circuits sont des copies conformes du circuit de capture/comparaison n° 0. La seule différence est que leur coeur (TAxCCRy) n'a pas d?influence sur la borne supérieure de TAxR. Ils intéragissent toutefois avec TAxR en mode capture. Pour le circuit n° y, le registre de donnée est nommé TAxCCRy, le registre de contrôle est TAxCCTLy. Bien entendu, dans le registre de contrôle TAxCCTLy, les champs portent les même noms (CAP, CM, CCIS, SCS, etc?).

5.3.4 Différence entre Capture et Comparaison

La figure 5.5 illustre le comportement du registre TAxR dans le mode « continuous ». Une fois le processus de comptage lancé, TAxR évolue en dent de scie, comme illustré. Le circuit de capture/comparaison utilise ce signal pour son opération.

En mode « comparaison », le contenu du registre TAxCCRy est une entrée. Dès que TAxR atteint TAxCCRy, le signal noté « Pulse » est généré, qui peut déclencher l?exécution d'une routine de service d'interruption.

En mode « capture », le contenu du registre TAxCCRy est une sortie. Un signal « Pulse » doit être fourni, qui déclenche la copie de TAxR dans TAxCCRy au moment où la pulse apparaît. En fait, c?est le flanc actif de la pulse qui déclenche le processus de capture, qui peut donc se produire sur le flanc montant, descendant ou les deux, du signal « pulse ». Typiquement, le signal « pulse » est une entrée du microcontrôleur. Au moment où la capture est exécutée, une requête d?interruption peut être émise.

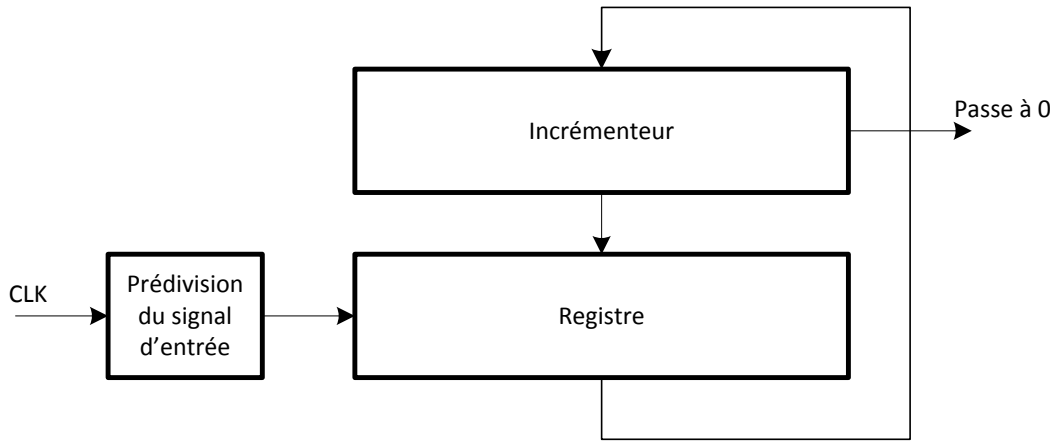

FIGURE 5.5 – Comparaison : TAxR est l'entrée / Capture : Pulse est l'entrée

Chapitre 6

Système d'horloges

Bla bla bla

6.1 Diviseur de fréquences

6.1.1 Zip

Chapitre 7

CPU

Bla bla bla

7.1 Principe

7.1.1 Tada

Chapitre 8

Assembleur

Bla bla bla

8.1 Chaine de compilation

8.1.1 Kaboum

8.2 Debug

Chapitre 9

Programmation en C

Bla bla bla

9.1 Mots clés

9.1.1 Youpla

Chapitre 10

Interfaces séries

Bla bla bla

10.1 Notion de pile protocolaire

10.1.1 Modèle OSI

Chapitre 11

AD et DA

Bla bla bla

11.1 AD

11.1.1 ploutch

Annexe A

Jeux d'instruction du MSP430

Bla bla bla

A.1 Une structure orthogonale

A.1.1 Vroum

zzz