



Micro 8 bits : l'ATMEGA 8

David Delfieu

Département Génie Electrique 3^{ieme} année

Acronyms

ADCH Analog to Digital Converter High. 20
ADCL Analog to Digital Converter Low. 20
GICR General Interrupt Control Register. 12, 13
MCUCR Management Control Unit Control Register. 9, 12
MCUCSR MCU Control and Status Register. 13
SPI Serial Peripheral Interface. 7
SRAM Static Random Access Memory. 6
SREG I T H S V N Z C. 8
USART Universal Synchronous and Asynchronous Serial Receiver. 7
ACIC Analog Comparator Input Capture Enable. 33
ACO Analog Comparator Output. 32, 33
COM1x1 : 0 Compare Output Mode x(A or B) 1:0. 33–35, 37, 38
ICF₁ Input Compare Flag 1. 30, 32
ICNC₁ Input Capture Noise Canceller 1. 33
OCF_{1A} Output Compare Flag 1 A. 30, 32
OCF_{1B} Output Compare Flag 1 B. 30, 32
TICIE₁ Timer Input Capture Interrupt Enable 1. 32
TOV₁ Timer Overflow 1. 30, 32
WGM_{13:0} Wave Generation Modes 3:0. 32–35
ACS ACD ACO ACI ACIE ACIC ACIS₁ ACIS₀. 33
ADCSRA ADEN ADSC ADATE ADIF ADIE ADPS₂ ADPS₁ ADPS₀. 21
ADMUX REFS₁ REFS₀ ADLAR MUX₄ MUX₃ MUX₂ MUX₁ MUX₀. 21
ASSR Asynchronous Status Register. 47
ICR₁ Input Capture Register. 31–33, 36, 39, 40
OCR_{1A} Output Compare Register 1 A. 31, 32, 35, 36, 39, 40
OCR_{1B} Output Compare Register 1 B. 31, 32, 35
OCR_{1x} Output Compare Register 1 x=A ou B. 36, 37, 40
OCR₂ Output Compare Register 2. 42–48

OC_{1A} Sortie *MLI* A du timer 1. 35

OC_{1X} Sorties *MLI* A ou B du timer 1. 38

SFIOR *ADTS2 ADTS1 ADTS0 – ACME PUD PSR2 PSR10*. 15, 28

TCCR₀ – – – – – *CS₀₂ CS₀₁ CS₀₀*. 27

TCCR_{1A} *COM1A₁ COM1A₀ COM1B₁ COM1B₀ FOC₁₁ FOC_{1B} WGM₁₁ WGM₁₀*. 31, 32, 38

TCCR_{1B} *ICN₁ ICES – WGM₁₃ WGM₁₂ CS₁₂ CS₁₁ CS₁₀*. 31–33, 39

TCCR₂ Timer Counter Control Register 2. 43, 46

TCNT₀ Timer Counter Timer 0. 27, 28

TCNT₁ Timer Counter Timer 1. 31, 32, 34–37, 39, 40

TCNT₂ Timer Counter Timer 2. 42–48

TIFR *OCF₂ TOV₂ ICF₁ OCF_{1A} OCF_{1B} TOV₁ – TOV₀*. 27, 31, 40, 42

TIMSK *OCIE₂ TOIE₂ TICIE₁ OCIE_{1A} OCIE_{1B} TOIE₁ – TOIE₀*. 5, 27, 28, 31, 40, 48

ACSR Comparateur analogique. 19

ADCH Poids fort de la conversion analogique. 4, 22

ADCL Poids faible de la conversion analogique. 4, 22

ADCSRA Contrôle du convertisseur analogique. 4, 21, 22

ADMUX Multiplexeur analogique. 4, 22

EEPROM Petite mémoire non volatile qui contient des données dynamiques. Elle a un temps d'accès très lent, autour de 5 millisecondes pour un *ATMEGA₈*. 6, 8

FLASH Mémoire non volatile qui contient le programme et les données statiques. Par rapport à la *SRAM*, elle a un temps d'accès plus lent ainsi qu'une durée de vie est assez limitée, par contre elle a une consommation faible. 6, 8

GICR Gestion des interruptions. 12

ICP1 Patte du timer 1 sur laquelle on peut réaliser une entrée de capture (*PB₀*). 33

MCUCR Gère notamment les activations des *interruptions* externes. 12

MLI Modulation de Largeur d'Impulsions. 7

OC1A Patte du timer 1 sur laquelle on peut observer une *MLI* (*PB₁*). 35

OC1B Patte du timer 1 sur laquelle on peut observer une *MLI* (*PB₂*). 35

OSCCAL Calibration et gestion du résonnateur. 10

PUD Inhibe les résistances de tirage. 15

SFIOR Résistances de tirage et le pré-diviseur des timer 0 et 1. 15, 19

SRAM Mémoire volatile qui contient les données dynamiques. Elle est contient notamment tous les registres du *microcontrôleur*. 6, 8

SREG Registre qui contrôle les bits d'état du *microcontrôleur* comme la Carry, le bit Zero, le bit Négatif, 8

TCCR0 Pré-diviseur du timer 0. 4, 27

TCNT0 Registre de comptage du timer 0. 4, 27

Contents

1	Présentation générale	4
1.1	Introduction	4
1.1.1	Les différents blocs mémoire de l' <i>ATMEGA₈</i>	4
1.1.2	Fonctionnalités	5
1.1.3	Les modes de communication de l' <i>ATMEGA₈</i>	5
1.2	Architecture	5
1.2.1	Architecture interne	6
1.2.2	Registres Systèmes	7
1.2.3	Horloges et modes des sommeil de l' <i>ATMEGA₈</i>	8
1.3	Les modes interruptifs de l' <i>ATMEGA₈</i>	9
1.3.1	Les <i>interruptions</i>	10
1.3.2	Les <i>interruptions</i> externes	10
1.3.3	Le chien de garde	11
1.4	Les Ports	11
1.4.1	Paramétrage et usage des <i>PORTS</i>	12
1.4.2	Résistance de tirage : Pull-up resistor	13
1.4.3	Registres de manipulation des ports : <i>DDR_x, PIN_x et PORT_x</i>	13
1.4.4	Le Port B	13
1.4.5	<i>PORTC</i>	14
1.4.6	<i>PORTD</i>	14
1.4.7	Lecture et écriture sur un port	15
1.4.8	Lecture d'un Port	15
1.4.9	Fonction de manipulation de bits	15
2	Conversion Analogique	16
2.1	Le Comparateur Analogique	16
2.1.1	Fonctionnement global	16
2.1.2	Les registres	17
2.2	Le Convertisseur Analogique-Numérique : <i>ADC</i>	18
2.2.1	Fonctionnement et caractéristiques	18
2.2.2	Les différentes méthodes de programmation d'une conversion	20
2.2.3	Les Registres de l' <i>ADC</i> : <i>ADMUX, ADCSRA, ADCH, ADCL</i>	20
3	Les Timers	24
3.1	Le <i>timer</i> 0	24
3.1.1	Caractéristiques	24
3.1.2	Les registres associés au <i>timer</i> 0 : <i>TCCR0, TCNT0</i>	25
3.1.3	Le prédiviseur	26
3.2	Le <i>timer</i> 1	28
3.2.1	Caractéristiques générales	28
3.2.2	Les 16 modes du <i>timer</i> 1	32
3.2.3	Les Modes du générateur de formes	33
3.2.4	Inventaire des registres utiles au <i>timer</i> 1	36
3.2.5	Résumé des 16 modes de <i>MLI</i> du <i>timer</i> 1	39
3.2.6	Exemples d'utilisation du <i>timer</i> 1	39

3.3	Le timer 2	40
3.3.1	Aperçu du timer 2	40
3.3.2	Génération de forme : pulse, <i>PWM</i> ,...	41
3.3.3	Les modes du timer 2	42
3.3.4	Les registres utiles au timer 2	44
3.3.5	Les <i>interruptions</i> du timer 2	46
3.3.6	<i>TIMSK</i>	46
3.3.7	TIFR	46
4	La programmation	48
4.1	Langage <i>C</i>	48
4.1.1	Structure d'un programme	48
4.1.2	Entête	48
4.1.3	Main	48
4.1.4	Fonctions	49
4.1.5	Déclaration de variable globales	49
4.1.6	Fonction de manipulation de bits	50
4.1.7	Mise en oeuvre d'une interruption	50
4.2	Interruptions	51

Chapter 1

Présentation générale

1.1 Introduction

Un **microcontrôleur** est un microprocesseur dédié au contrôle, il contient dans un même composant une unité de calcul *CPU*, comme dans un micro-processeur, mais il a, par contre la possibilité d'adresser directement des Ports d'entrées-sorties (impossible pour un microprocesseur). Il a de plus, spécifiquement, des timers, des convertisseurs analogiques, des unités de communication et de la mémoire.

Ce cours présente l'*ATMEGA₈*, **microcontrôleur** de la famille **Arduino** (Microchip anciennement Atmel) sur lequel on développera des programmes en langage *C* dans l'environnement **Arduino**. On n'utilisera pas cependant le langage **Arduino** leur préférant une structure *C* et la manipulation de registres.

La famille des *ATMEGA₈*

Modèle	Flash	EEPROM	RAM	I/O	PWM	Interfaces	CAN
<i>ATMEGA₈</i>	8K	512	1024	23	3	SPI-USART	10 bits
<i>ATMEGA₁₆</i>	16K	512	1024	32	4	SPI-USART	10 bits
<i>ATMEGA₃₂</i>	32K	1k	2k	32	4	SPI - USART	10 bits
<i>ATMEGA₆₄</i>	64K	2k	4k	53	8	SPI - USART(2)	10 bits
<i>ATMEGA₁₂₈</i>	128K	4k	4k	53	8	SPI - USART(2)	10 bits
<i>ATMEGA₂₅₆</i>	256K	4k	8k	53	16	SPI - USART(2)	10 bits

Dans cette famille, l'*ATMEGA₈* et l'*ATMEGA₁₆* sont compatibles broches à broches et il n'y a que très peu de différences au niveau du code. On pourrait sur la carte de *TP* remplacer facilement un *ATMEGA₈* par un *ATMEGA₁₆*.

1.1.1 Les différents blocs mémoire de l'*ATMEGA₈*

La mémoire de l'*ATMEGA₈* est constituée de 1ko de Mémoire vive (SRAM), de 512 octets de EEPROM et de 8ko de mémoire FLASH.

La mémoire de type SRAM contient les registres et la pile système. La mémoire de type Static Random Access Memory (**SRAM**) est un type de mémoire vive "volatile" utilisant des bascules pour mémoriser les données. En l'absence d'alimentation les données sont perdues. Pour placer des variables dans cet espace, on utilisera l'attribut *volatile*. Par exemple : *volatile int i*; Cet attribut assure que la variable sera déclarée dans la SRAM, en dehors de la zone des registres spéciaux. Cela assure à une variable globale de ne pas être perturbée par des modifications de ces registres.

La mémoire FLASH permet 10.000 cycles d'écriture. Elle contient le programme et les données. C'est une mémoire non volatile. Par rapport à la SRAM, elle a un des temps d'accès moins rapide, une durée de vie est assez limitée mais une consommation faible. Elle est même nulle au repos. La FLASH utilise comme cellule de base un transistor *MOS* possédant une grille flottante enfouie au milieu de l'oxyde de grille, entre le canal et la grille. L'information est stockée grâce au piégeage d'électrons dans cette grille flottante. Cette technologie se décline sous deux principales formes : *NOR* et *NAND*, d'après le type de porte logique utilisée pour chaque cellule de stockage. Dans l'*ATMEGA₈* on a une FLASH de type *NOR*. Les mémoires de type *NAND* sont plutôt consacrées aux mémoires de masse externes telles que les Cartes **SD**, **disque dur**,...

Les **mémoires** de type EEPROM sont les plus chères. Elle autorisent 100.000 cycles d'écriture. Elles ont un temps d'accès un peu plus long, et donc on y stocke des données qui n'ont pas vocation à être modifiées souvent. Une autre différence avec la FLASH classique est que l'on y écrit octet par octet.

1.1.2 Fonctionnalités

Les différentes fonctionnalités sont les **timers**, le convertisseur analogique (**ADC**), les possibilités de communication, les **mémoires** et les **PORTS** d'entrées/sorties. Un **timer** peut définir des bases de temps, faire du comptage d'événements, ou bien générer des MLI ou de gérer un watchdog ¹. L'**ADC** permet de convertir en valeurs numériques codées sur 10 bits des tensions entre 0v et 5v. Les **PORTS** permettent d'adresser et de communiquer avec des composants externes.

1.1.3 Les modes de communication de l'ATMEGA₈

L'ATMEGA₈ dispose de manière interne d'un circuit dénommé Universal Synchronous and Asynchronous Serial Receiver (**USART**). A noter qu'on entend couramment parler d'**UART**, mais qu'Atmel a ajouté ici un *S* pour *Synchronous*. Ce qui veut dire que cette interface peut servir à faire aussi bien de la communication série synchrone ou asynchrone, c'est à dire avec des bits de start et de stop, mais aussi synchrone dans laquelle les bits de données sont envoyés de manière cadencée par un signal de clock, piloté par un maître du protocole de communication. Elle utilise deux fils : un pour l'émission et un pour la réception.

La communication série de type **SPI** est un bus de donnée série synchrone baptisé ainsi par Motorola, et qui opère en Full Duplex. Les circuits communiquent selon un schéma maître-esclaves, où le maître s'occupe totalement de la communication. Plusieurs esclaves peuvent co-exister sur un bus, la sélection du destinataire se fait par une ligne dédiée entre le maître et l'esclave appelée *chip select*.

Le bus Serial Peripheral Interface (**SPI**) contient 4 signaux logiques :

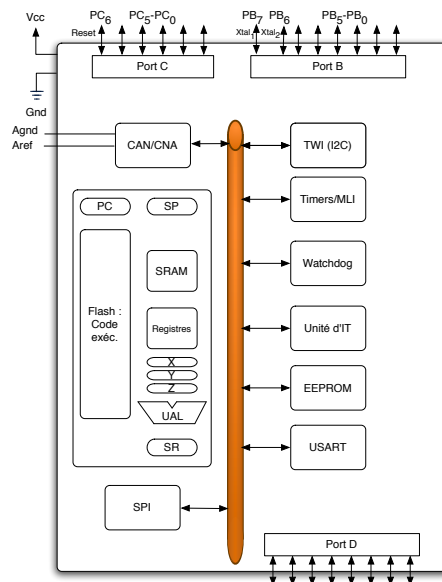
- **SCLK** : Horloge (généré par le maître)
- **MOSI** : Master Output, Slave Input (généré par le maître)
- **MISO** : Master Input, Slave Output (généré par l'esclave)
- **SS** : Slave Select, Actif à l'état bas, (généré par le maître)

La liaison **SPI** est utilisée pour la programmation de l'ATMEGA₈.

Grâce à une liaison série de l'ATMEGA₈ on peut loader un programme à exécuter dans le **microcontrôleur**. Mais on peut aussi communiquer de façon synchrone (**PI**) ou asynchrone (**TWI**) avec des composants externes.

1.2 Architecture

L'architecture du **microcontrôleur** est illustrée dans la figure suivante :



¹Système de surveillance de bon déroulement de programme : Un watchdog est capable de détecter si un programme sort de sa boucle infinie déclenchant alors un reset qui remettra le programme dans sa boucle

1.2.1 Architecture interne

Comme on l’a vu, il y a trois sortes de mémoires :

La mémoire FLASH : stocke le programme (10.000 cycles)

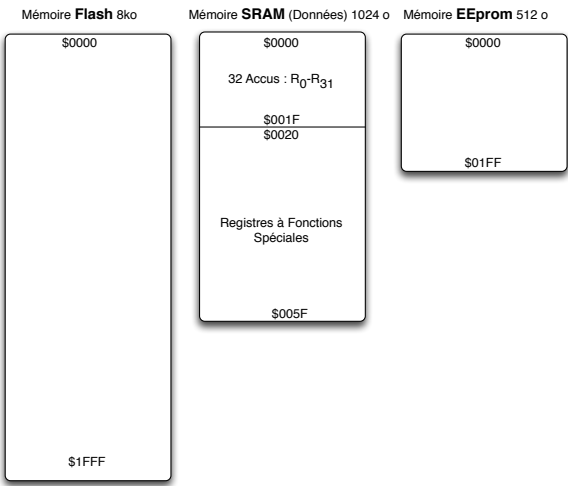
La mémoire SRAM (mémoire donnée) :

- les 32 accus ;
- les registres à fonctions spéciales ;
- la pile.

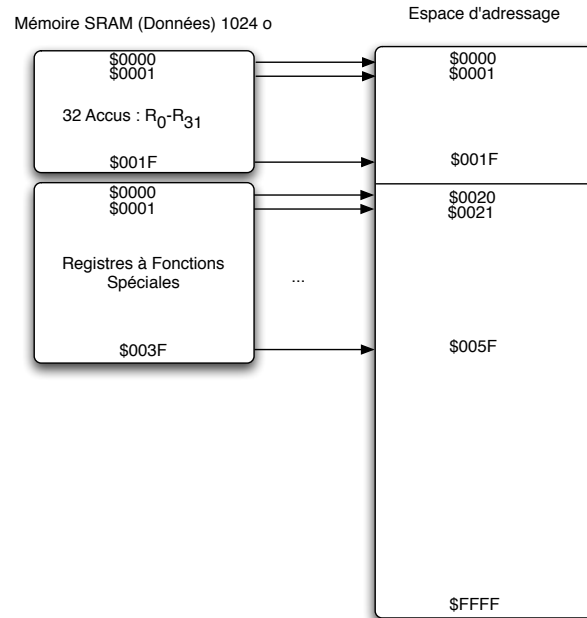
La mémoire EEPROM : on y place des données stratégiques (100 000 cycles)

Les 32 registres internes sont :

Registres	Adresse	Fonction
R_0	\$00	Accumulateur
R_1	\$01	Accumulateur
...	...	accu
R_{25}	\$19	Accumulateur
R_{26}	\$1A	X poids faible
R_{27}	\$1B	X poids Fort
R_{28}	\$1C	Y poids faible
R_{29}	\$1D	Y poids Fort
R_{30}	\$1E	Z poids faible
R_{31}	\$1F	Z poids Fort



Un mapping signifie une projection d’un plan mémoire sur l’espace d’adressage. Dans l’ATMEGA8, certains plans partagent un même espace d’adressage. On distinguera alors l’accès aux variables partageant ce même espace par l’utilisation de modes d’adressage spécifiques.



1.2.2 Registres Systèmes

Ces registres agissent sur le contrôle ou indiquent l'état du processeur.

Registre *SREG*: SREG est un registre crucial dans ce micro-contrôleur. C'est lui qui surveille en permanence le microcontrôleur et positionne ces bits en fonction de la dernière opération arithmétique ou logique. Par exemple si la dernière opération donne un résultat négatif le bit *N* de *SR* passe à un.



- *C*: Carry
- *Z*: Zero
- *N*: Negative
- *V*: overflow = $C_8 \oplus C_7$
- *S*: $V \oplus N$
- *H*: Half carry
- *T*: copy sTorage : bit tampon pour manipuler un bit
- *I*: autorisation générale des *interruptions* : *sei()* – *cli()*

Registre de pile *SP* Ce registre permet les appels de sous-programmes et le passage de paramètres et la sauvegarde de l'état courant d'un programme. En assembleur, une pile se manipule par les instructions :

PUSH : Empile une donnée, décrémente *SP*

POP : Dépile une donnée, incrémente *SP*

Comme les piles Motorola, la pile de l'Atmel, fonctionne par adresses décroissantes. Par défaut *SP* contient : 0x60 et il faudra changer cette valeur par la valeur : 0x1FF

1.2.3 Horloges et modes des sommeil de l'ATMEGA8

Les modes de sommeil : Registre *MCUCR* Ce registre définit les différents modes de sommeil dans lequel le micro peut être plongé :

SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
----	-----	-----	-----	-------	-------	-------	-------

Les bits de configuration des modes de sommeil *SM_i*:

SM2	SM1	SM0	Mode de Sommeil
0	0	0	Mode attente
0	0	1	Mode réduction de bruit pour l'ADC
0	1	0	Mode sommeil (Power down)
0	1	1	Mode économie d'énergie (Power save)
1	0	0	Réservé
1	0	1	Réservé
1	1	0	Non utilisé
1	1	1	En pause

	Domaines d'horloges actives					Oscillateurs		Sources de reveil					
Sommeil	CPU	FLA	IO	ADC	ASY	QTZ	TIM	INT	TWI	T2	EED	ADC	IOs
Power Down								X	X				
En pause						X		X	X				
Power Save					X		X	X	X	X			
Réd. bruit				X	X	X	X	X	X	X	X	X	
Attente			X	X	X	X	X	X	X	X	X	X	X

Calibration et le contrôle de l'horloge OSCCAL

CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0
------	------	------	------	------	------	------	------

Valeur de calibrage de l'oscillateur pour la programmation de la mémoire flash ou de l'eprom. Suivant la configuration de certains bits, on peut utiliser les résonateurs suivants :

- Résonateur externe type céramique ou crystal
- Cristal externe basse fréquence
- Oscillateur externe ou interne de type RC
- Oscillateur calibré interne de type RC
- Horloge externe de type quelconque

La figure suivante (fig. 1.2.3) présente un montage en quartz externe basse fréquence :

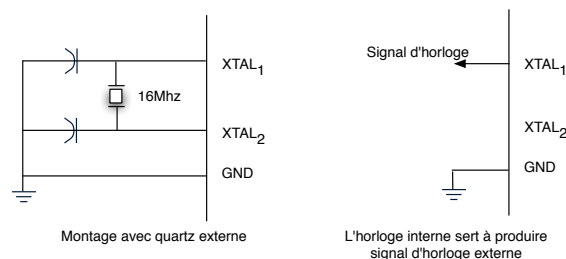


Figure 1.1: Quartz haute fréquence

On parle d'horloge temps réel lorsqu'elle permet de générer des diviseurs ou des multiples entier de la seconde. La figure suivante (fig. 1.2.3) combine le positionnement d'un quartz externe et d'une horloge temps réel :

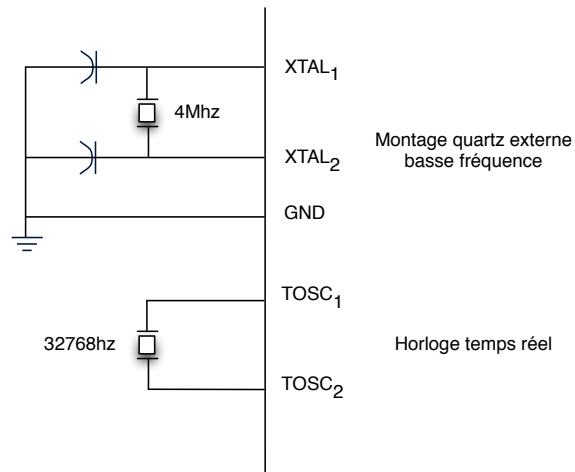


Figure 1.2: Quartz basse fréquence

La figure suivante (fig. 1.2.3) présente un montage en résonateur externe :

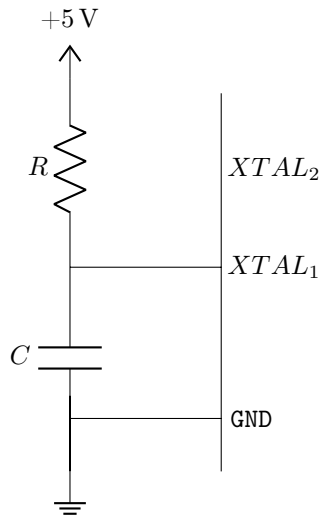


Figure 1.3: Résonateur externe

Tableau récapitulatif de tous les registres systèmes :

Adresse	Nom	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x31 (0x51)	<i>OSCCAL</i>	Registre de calibration et de contrôle de l'horloge							
0x34 (0x54)	<i>MCUCSR</i>	-	-	-	-	WDRF	BORF	EXTRF	PORF
0x35 (0x55)	<i>MCUCR</i>	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
0x37 (0x57)	<i>SPMCR</i>	SPMIE	RWWSB	-	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
0x3D (0x5D)	<i>SPL</i>	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x3E (0x5E)	<i>SPH</i>	-	-	-	-	-	SP10	SP9	SP8
0x3F (0x5F)	<i>SREG</i>	I	T	H	S	V	N	Z	C

1.3 Les modes interruptifs de l'ATMEGA₈

Un **microcontrôleur** fonctionne de façon normale en exécutant la boucle infinie de son programme principal. Il peut aussi être interrompu par un événement unique ou récurrent pour exécuter un sous-programme associé. Il y a deux sortes

d'*interruptions* : Les *interruptions* internes liées aux **timers**, à l'**ADC**, ... ou des *interruptions* externes.

1.3.1 Les *interruptions*

Une *interruption* externe correspond à la prise en compte d'un événement dont l'occurrence possiblement aléatoire. Lors de la survenue de cet événement ou exécute un sous-programme associé. Ces événements peuvent être par exemple :

- La fin d'un délai,
- La fin de conversion,
- Le compteur d'un **timer** atteint un seuil
- Un front sur une patte du **microcontrôleur**

Un front sur une patte peut correspondre au niveau applicatif à :

- Un arrêt d'urgence provoqué par l'appui d'un bouton ad. hoc.
- Capteur de choc ou de contact
- Dépassement de seuil sur un capteur de température, ...

D'un point de vue programmation, il serait donc peu intéressant d'utiliser des boucles d'attentes de cette alarme. Son traitement est donc réalisé en associant un sous-programme à un événement. On associe un événement à un sous-programme par la primitive ISR.

1.3.2 Les *interruptions* externes

L'*ATMEGA8* offre deux *interruptions* externes sur deux pattes : **INT₀(PD₂)** et **INT₁(PD₃)**. Les événements qui peuvent se produire sur ce type de patte sont soit un front montant, soit un front descendant, soit un changement de niveau.

Associer un événement à un sous-programme se fait en utilisant l'instruction ISR auquel on passe un nom d'évènement : *ISR(Nom_evenement)* et du code entre accolades. L'ensemble des événements est données dans les tables de la section 4.2 Exemple :

```
ISR(INT0_vect){
    PORTD ^= 0xF0; // code associe a l'interruption
}
int main(){
    DDRD=0xF0;
    GICR |= 1<<INT0;
    MCUCR |= 1<<ISC01; // Front descendant
    sei();
    while(1){}
```

Si l'évènement **INT₀** (front sur la patte **PD₂**) se produit, quel que soit le comportement de la boucle infinie du programme principal (*main*), le travail est interrompu pour exécuter le sous-programme associé qui est défini entre les deux accolades du bloc *ISR*. A la fin du sous-programme d'*interruption* on revient là où l'on a été interrompu. La mise en oeuvre d'une *interruption* externe se fait grâce aux registre **GICR** et **MCUCR** :

- GICR :

INT1	INT0	-	-	-	-	IVSEL	IVCE
------	------	---	---	---	---	-------	------

- **INT₀** : à un, autorise une interruption externe sur la patte **PD₂**
- **INT₁** : à un, autorise une interruption externe sur la patte **PD₃**

- MCUCR :

SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
----	-----	-----	-----	-------	-------	-------	-------

Considérons une *interruption* externe **INT₀**. Les bits **ISC₀₁**, **ISC₀₀** correspondent alors à la patte **PD₂** et le niveau d'activité est défini par le tableau suivant :

- 00 : niveau bas
- 01 : front montant ou descendant
- 10 : front descendant
- 11 : front montant

Autres bits du registre *GICR*

- *IVSEL* : Interrupt Vector Select
Quand le bit *IVSEL* est mis à zéro, les vecteurs d'interruption sont placés au début de la la mémoire flash. Quand ce bit est mis à un, les vecteurs d'interruption sont déplacés au début de la zone du boot loader de la mémoire flash. L'adresse de cette zone est modifiable par les bits "fusibles" *BOOTSZ*.
- *IVSEL* : à un, autorise le changement du bit *IVSEL*

Parmi les différents modes de fonctionnement d'un micro-contrôleur on peut citer le fonctionnement "chien de garde".

1.3.3 Le chien de garde

Un chien de garde permet de relancer/réinitialiser le programme. En effet lors d'une perturbation électromagnétique, par exemple, le déroulement du programme peut être altéré : Le compteur programme peut alors essayer d'exécuter du code dans une zone mémoire non prévue. Un chien de garde, par exemple, armé toutes les 500 millisecondes, peut alors resetter le programme et le remettre dans un déroulement normal.

Registre *MCUCSR*

-	-	-	-	<i>WDRF</i>	<i>BORF</i>	<i>EXTRF</i>	<i>PORF</i>
---	---	---	---	-------------	-------------	--------------	-------------

- *WDRF* Watchdog Reset Flag: mis à un pour activer le watchdog, raz par un reset ou par une écriture d'un 0
- *BORF* : mis à un lors d'une panne d'électricité partielle, raz par reset ou écriture de 0.
- *EXTR* et *PORF* : Détermine la source d'un reset.

1.4 Les Ports

Dans un système à base de *microcontrôleur* on appelle "*PORT* d'entrées-sorties", des ensembles de 8 connections entre le *microcontrôleur* et l'extérieur. Par ces ports, le système peut réagir à des modifications de son environnement, voire le contrôler. Elles sont parfois désignées par l'acronyme *I/O*, issu de l'anglais *Input/Output* ou encore *E/S* pour *Entrées/Sorties*. Ces *PORTS* sont programmables en entrée ou en sortie.

Des buffers sont associés aux *PORTS*, ils ont la capacité d'être à la fois source ou drain de courant ou en haute impédance. Une ligne d'un *port* d'entrées est essentiellement composé d'un tampon à trois états. Ceux-ci se comportent comme des interrupteurs électroniques qui font apparaître, au moment voulu, soit deux niveaux logiques : zéro ou un et un état de haute impédance. Les niveaux logiques sont mémorisés dans un registre du processeur.

Nom	B7	B6	B5	B4	B3	B2	B1	B0
PORT D								
PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
DDRD	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
PORT C								
PINC	-	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
DDRC	-	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
PORTC	-	PC6	PC5	PC4	PC3	PC2	PC1	PC0
PORT B								
PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
DDRB	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
PORTB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Contrôle des Ports								
SFIOR	-	-	-	-	ACME	PUD	PSR2	PSR10

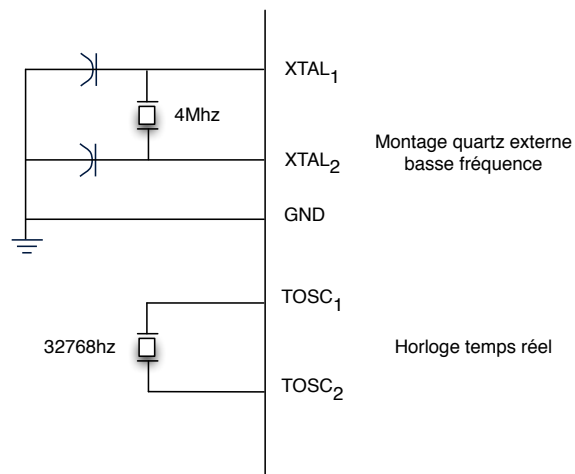


Figure 1.4: Quartz basse fréquence

1.4.1 Paramétrage et usage des *PORTS*

Pour définir le Port en Entrée ou en Sortie on utilise DDR_x ($x = B, C, D$).

- On peut positionner des résistances de tirage sur les lignes d'un port.
- Ecriture sur le Port : Si le Port est en sortie l'écriture d'un "0" ou d'un "1" positionnera cette valeur sur le $PORT_x$. Chaque *port* de 8 bits est limité à un courant total de 200 mA.
- Lecture sur le Port : de la donnée disponible sur le Port est faite dans PIN_x . A chaque bit du registre PIN_x est associée une bascule D qui permet de stabiliser la lecture en la synchronisant sur un front de l'horloge système. Cette stabilisation se fait au détriment d'un délai de lecture qui correspond à une période d'horloge.

1.4.2 Résistance de tirage : Pull-up resistor

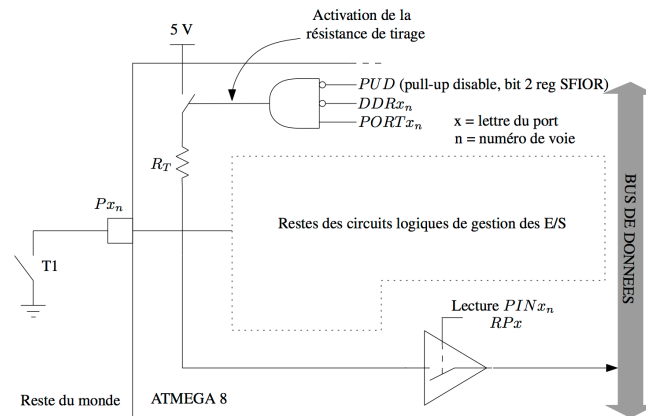


Figure 1.5: Résistances de tirage type "pull-up"

Pour poser des résistances de pull-up il est nécessaire d'avoir les trois conditions suivantes:

- Définir les lignes en entrée ;
- Ecrire un "1" sur ces lignes ;
- PUD mis à zéro dans SFIOR : $SFIO\& = (1 \ll PUD)$;

Résistances de tirage : le registre *SFIO*

ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10
-------	-------	-------	---	------	-----	------	-------

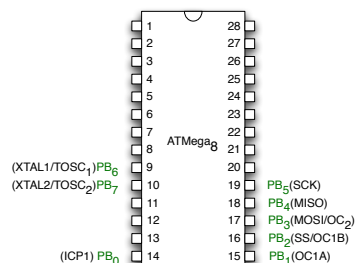
PUD="Pull Up Disable"

- 1: Les résistances de tirages sont désactivées, les entrées sont en mode 3-états.
- 0 : Les résistances de tirages sont activées.

1.4.3 Registres de manipulation des ports : *DDR_x*, *PIN_x* et *PORT_x*

- *DDR_B*, *DDR_C*, *DDR_D* : Permet de programmer le sens des lignes (1 : sortie, 0 entrée)
- *PORT_B*, *PORT_C*, *PORT_D* : Mémoire l'état des pins, permet l'écriture ou la lecture Ou bien il permet de configurer la résistance de tirage en entrée.
- *PIN_B*, *PIN_C*, *PIN_D* : Permet de lire les valeurs courantes des pattes du port

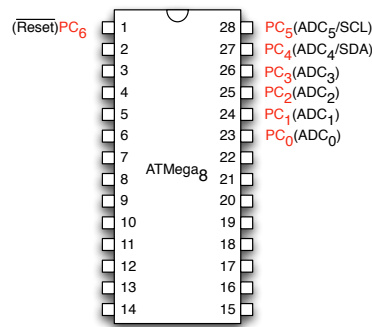
1.4.4 Le Port B



- *DDR_B* : Programme les lignes du port B

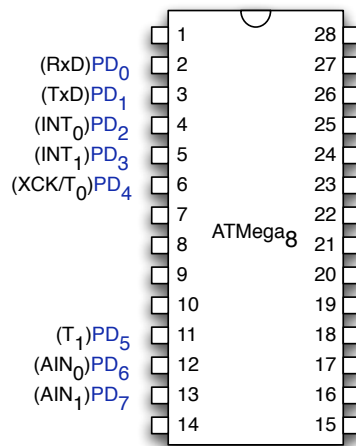
- PIN_B : Lecture du port B
- $PORT_B$: Ecriture sur le port B

1.4.5 $PORTC$



- $DDRC$: Programme les lignes du port C
- PIN_C : Lecture du port C
- $PORT_C$: Ecriture sur le port C
- $PORTC$ est un port d'entrée-sortie bidirectionnel de 7 bits avec des résistances de pull up.
- Le buffer de sortie du $PORTC$ a la capacité d'être à la fois source ou drain de courant.
- En entrée les pattes sont tirés à la masse si les résistances de pull-up sont activées.
- Les pins de C sont en 3-états quand un reset survient.
- $PORTC_6 = \overline{RESET}$ si le fusible $RSTDISBL$ est à 1, alors la ligne est disponible en E/S

1.4.6 $PORTD$



- $DDRD$: Programme les lignes du port C
- $PIND$: Lecture du port D
- $PORTD$: Ecriture sur le port D

1.4.7 Lecture et écriture sur un port

Écriture sur un Port

- Pour écrire sur un *port* on déclare une ou plusieurs ligne en sortie par un registre.
Exemple : `DDRD = 0xFF`; Tout le *port* est déclaré en sortie
- Ensuite on réalise une écriture sur les lignes déclarées en sortie par le registre `PORTD`
Exemple : `PORTD = 0b00000011`; On active les lignes 0 et 1

Exemple d'écriture d'un Port :

```
int main(void) {
    DDRD = 0xFF;      // port D en sortie
    PORTD = 0x7A;     // Allume leds 1,3,4,5,6
    while(1){         // boucle infinie
        PORTD ^= 0x7A; // ^= : ou-exclusif fait clignoter les leds
        _delay_ms(20);
    }
}
```

1.4.8 Lecture d'un Port

- Pour lire sur un *port* on déclare une ou plusieurs ligne en entrée par le registre `DDRx`.
Exemple : `DDRD = 0x00`; Tout le *port* D est déclaré en entrée
- Ensuite on réalise une lecture sur les lignes déclarées en sortie par le registre `PINx` dans une variable de type entier.
Exemple : `int lu = PIND`; On active les lignes 0 et 1

Exemple ducture et d'écriture d'un Port :

```
int main(void) {
    DDRC = 0x00; // \pc\ en entree
    DDRD = 0xFF; // port D en sortie
    int lu;
    while(1){     // boucle infinie
        lu = PINC,
        PORTD = lu; // PORTD recoit PORTC
        _delay_ms(20);
    }
}
```

1.4.9 Fonction de manipulation de bits

```
// Mise a un d'un bit sans affecter les autres bits
PORTD = PORTD | (1<<PORTD4);
PORTD |= (1<<4); // mise a un du bit 4

// Mise a un des 4 bits de poids faible sans affecter les autres bits
PORTB = PORTB | 0x0F // ou
PORTB |= 0x0F;

// Mise a zero d'un bit sans affecter les autres bits
PORTB &=~(1<<PB3); // mise a zero du bit 3

// Mise a zero de 4 bits sans affecter les autres bits
PORTB &= 0x0F; // mise a zero des 4 bits de poids Fort

// Commutation du bit 4 sans affecter les autres bits
PORTD = PORTD^0x10;
PORTD ^= 0x10;
```

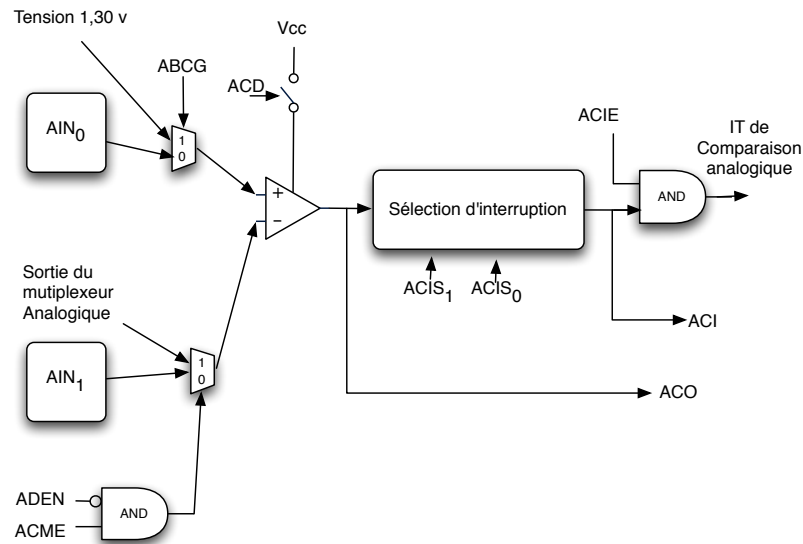
Chapter 2

Conversion Analogique

2.1 Le Comparateur Analogique

2.1.1 Fonctionnement global

Cet élément offre la possibilité de comparer deux valeurs analogiques sur les pattes PD_6 et PD_7 . $AIN_0(PD_6)$ est appelé broche positive et $AIN_1(PD_7)$ broche négative. Lorsque $AIN_0 > AIN_1$ alors $ACO \leftarrow 1$, on peut alors déclencher une *interruption* (avec bit $ACIE=1$) de comparaison.



On a de plus, la possibilité de changer la référence "négative" : AIN_1 par l'une des broches $ADC_0, ADC_1, \dots, ADC_5$. De plus, l'utilisateur peut sélectionner comme évènement de déclenchement, un front montant, un front descendant ou une inversion.

En outre, toute entrée ADC_5, \dots, ADC_0 peut jouer le rôle de AIN_1 . Pour réaliser cela il faut positionner à un le bit de multiplexage $ACME$ du registre $SFIOR$, et mettre $ADEN$ à un du registre $ADCSRA$ on a alors les $MUX_{2..0}$ du registre $ADMUX$.

Les bits *MUX* qui remplacent *AIN₁* selon le tableau suivant :

<i>ACME</i>	<i>ADEN</i>	<i>MUX2:0</i>	<i>AIN0</i>
0	<i>x</i>	<i>xxx</i>	<i>AIN1</i>
1	1	<i>xxx</i>	<i>AIN1</i>
1	0	000	<i>ADC₀</i>
1	0	001	<i>ADC₁</i>
1	0	010	<i>ADC₂</i>
1	0	011	<i>ADC₃</i>
1	0	100	<i>ADC₄</i>
1	0	101	<i>ADC₅</i>
1	0	110	<i>ADC₆</i>
1	0	111	<i>ADC₇</i>

2.1.2 Les registres

Registre ACSR: Analog Comparator Status Register ACSR

<i>ACD</i>	-	<i>ACO</i>	<i>ACI</i>	<i>ACIE</i>	<i>ACIC</i>	<i>ACIS1</i>	<i>ACIS0</i>
------------	---	------------	------------	-------------	-------------	--------------	--------------

- *ACD* Analog Comparator Disable : Bit de mise en marche du comparateur analogique,
0 : mise en marche,
1 : arrêt
- *ACO* Analog Comparator Output : Contient le résultat de la comparaison : si $V_{AIN0} > V_{AIN1}$ alors *ACO* = 1.
- *ACI* : Analog Comparator Interrupt : Flag Bit de demande d'interruption (bit raz ds le sp d'IT)
- Condition IT : *ACIS1* et *ACIS0*). Ce bit est remis à 0 automatiquement après le traitement de l'interruption
- Masque d'IT : *ACIE*.
- *ACIE* Analog Comparator Interrupt enable : Bit de validation de l'interruption *ANA_COMP*.
- *ACIC* : Analog Comparator Input Capture Enable : La mise à 1 de ce bit connecte la sortie du comparateur à l'entrée de capture du Timer1.
- *ACIS1* et *ACIS0* gère le comportement de la sortie *AC₀* :

Activation de <i>AC₀</i>	<i>ACIS1</i>	<i>ACIS0</i>
1 → 0 ou 0 → 1	0	0
non utilisé	0	1
<i>front montant</i>	0	0
<i>front descendant</i>	1	1

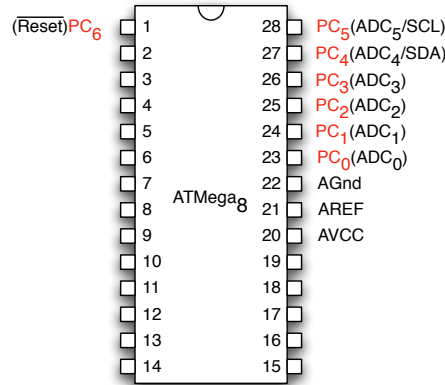
Registre

<i>ADTS2</i>	<i>ADTS1</i>	<i>ADTS0</i>	-	<i>ACME</i>	<i>PUD</i>	<i>PSR2</i>	<i>PSR10</i>
--------------	--------------	--------------	---	-------------	------------	-------------	--------------

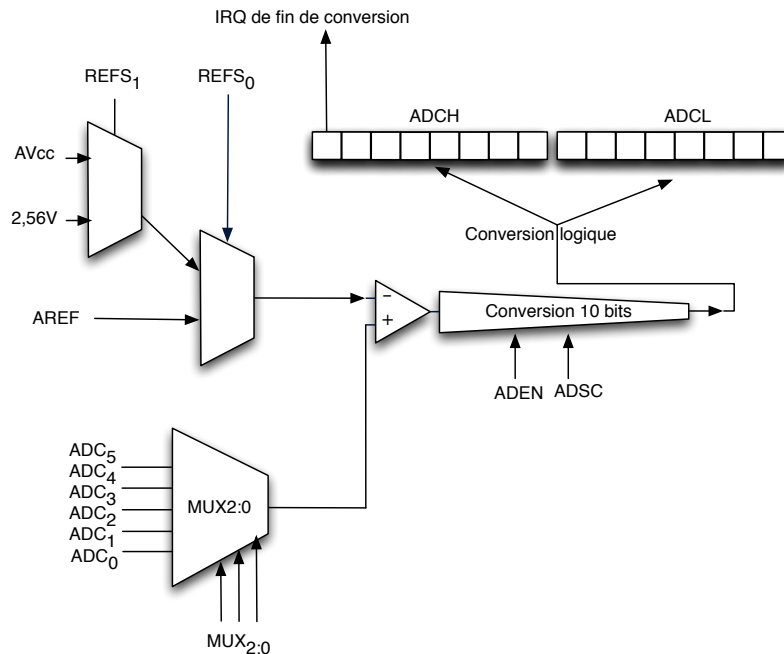
ACME Analog Comparator Multiplexer Enable

- Quand *ACME* = 1 et *ADC* est éteint (*ADEN* = 0 et *ADSC* = 0) alors le multiplexeur *ADC* choisit l'entrée négative du comparateur analogique.
- Quand *ACME* = 0 alors *AIN₁* est appliqué à l'entrée négative du comparateur analogique.

2.2 Le Convertisseur Analogique-Numérique : **ADC**



Le convertisseur (**ADC**) convertit une tension d'entrée analogique en une valeur à 10 bits digitale par approximations successives. Il possède 6 entrées simultanées avec une non-linéarité inférieure à $\pm 2 \text{ LSB}$ avec une erreur à 0 V inférieure à 1 *LSB*. Le résultat de la conversion est positionné dans les registres Analog to Digital Converter High (**ADCH**) et Analog to Digital Converter Low (**ADCL**).



Le temps de conversion prend au minimum 13 cycles d'horloge. De plus, le convertisseur a une alimentation découplée du micro : AV_{cc} tq $AV_{cc} \in [V_{cc} - 0,3, V_{cc} + 0,3]$: V_{cc} est la tension de référence qui peut être externe ou interne : **AREF** ou AV_{cc} ou $= 2,56 \text{ v}$.

2.2.1 Fonctionnement et caractéristiques

On peut le programmer en générant une interruption de fin de conversion. De plus, il est possible de faire fonctionner en limitant le bruit en mode de sommeil. Le résultat de la conversion numérique est donné par la relation :

$$\text{Résultat numérique} = \text{P. Entière}(\text{Tension d'entrée} / \text{Tension de référence} * 1024)$$

Où la tension de référence est soit **AREF** soit AV_{cc} soit $2,56 \text{ v}$.

ADC_0= 2.5 V, AREF = 3.3 V
 ADCH-ADCL = PE(2.5/3.3 x 1024) = 774

Réduction des bruits lors de la conversion Pour réduire au maximum la précision de la conversions on pourra :

- Mettre en sommeil l'unité centrale avant le lancement d'une conversion.
- Découpler soigneusement l'alimentation AV_{cc} avec des condensateurs.
- Règles élémentaires du routage : connexions courtes, plan de masse, ...
- Effectuer un filtre numérique des résultats (amortissement ,moyenne, ...).

La valeur minimale de conversion est GND tandis que a valeur maximale est soit la tension sur la broche $AREF$ soit le tension sur la broche AV_{cc} ou bien une tension interne de 2,56 V (cf bits $REFS_n$).

Chacune des 6 broches d'entrée ADC peuvent être choisies comme des entrées simples de l' ADC . De plus, l' ADC contient un mécanisme d'échantillonneur-bloqueur qui assure que l'entrée est tenue constante pendant 1,5 cycle d'horloge.

$ADCL$ doit être lu en premier puis $ADCH$ pour assurer la cohérence des données qui appartiennent à la même conversion. Une fois $ADCL$ lu, l'accès aux registres de commandes est bloqué afin d'empêcher une nouvelle conversion tant que $ADCH$ n'est pas lu. Quand $ADCH$ est lu, l' ADC est à nouveau opérationnel. L' ADC a sa propre interruption qui peut être déclenchée quand une conversion est achevée. La lecture de l' $ADCL$ et d' $ADCH$ interdit l'accès aux registres de commande de l' ADC , mais si une *interruption* de fin conversion se produit alors que la lecture précédente n'a pas été encore faite le résultat sera perdu. Il faut donc faire attention à lire rapidement un résultat de conversion.

Programmation

L'ensemble des registres à programmer l' ADC est $ADMUX$, $ADCSRA$, $ADCH$, $ADCL$, étudions $ADEN$ $ADSC$ $ADATE$ $ADIF$ ($ADCSRA$) :

$ADEN$	$ADSC$	$ADFR$	$ADIF$	$ADIE$	$ADPS2$	$ADPS1$	$ADPS0$
--------	--------	--------	--------	--------	---------	---------	---------

- L' ADC est activé avec le bit $ADEN$.
- La référence de tension et le choix du canal d'entrée n'entrera pas en vigueur quand $ADEN$ est mis à 1, il faut d'abord désactiver $ADEN$.
- ADC produit un résultat sur 10 bits qui est présenté dans les registres $ADCH$ et $ADCL$.
- Par défaut, le résultat est présenté ajusté à droite, mais peut facultativement être présenté ajusté à gauche en mettant le bit $ADLAR$ à un dans $ADMUX$. Pour un résultat sur 8 bits et un ajustement à gauche, la lecture du registre $ADCH$ est suffisante.

Lancement d'une conversion

Début d'une conversion Une conversion simple est lancée par $ADSC \leftarrow 1$ ($ADSC \in ADCSRA$).

Si l'on change le canal tandis qu'une conversion est en cours, l' ADC finira la conversion actuelle avant l'exécution du changement de canal.

Fin de conversion Dès lors, qu'une conversion est en cours, $ADSC$ reste à un tant que la conversion se réalise et il redescend à 0 quand la conversion est achevée. Quand une conversion est finie, le résultat est écrit dans les registres de données $ADCH$ et $ADCL$ et $ADIF$ est mis à 1. Le programme peut alors lancer $ADSC$ de nouveau et une nouvelle conversion sera amorcée sur le premier front montant de l'horloge.

Durée d'une conversion Une conversion simple commence par $ADSC \leftarrow 1 \in ADCSRA$ puis en attendant le front montant suivant du cycle d'horloge ADC . Une conversion normale prend 13 cycles d'horloge.

La première conversion après l'allumage de l' ADC : $ADEN$ mis alors à 1 dans $ADCSRA$.

2.2.2 Les différentes méthodes de programmation d'une conversion

Conversion échantillonnée

Pour ce mode on va utiliser un **timer** qui va définir une période d'échantillonnage. On va associer à une *interruption* de débordement de ce **timer** dans laquelle on va déclencher la conversion analogique-numérique. Un exemple de codage est donnée dans la section suivante (cf section ??).

Free Running mode : Mode de fonctionnement libre

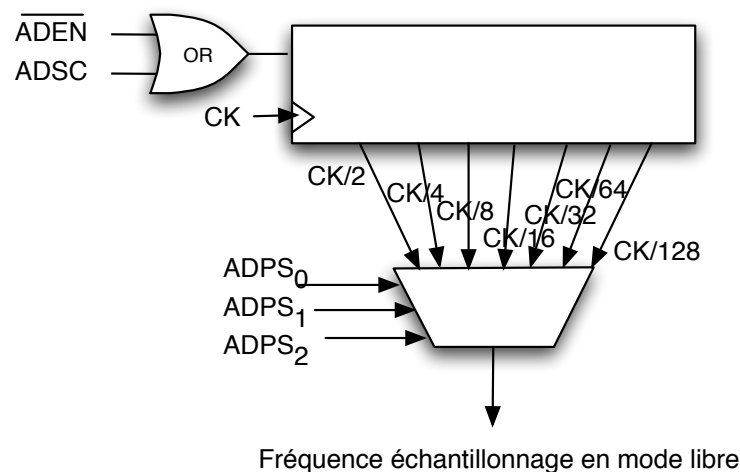
Dans ce mode, l'**ADC** échantillonne en permanence, sans aucune action du programmeur autre que le lancement initial, et met à jour les registres de données (**ADCH** et **ADCL**). Ce mode est positionné par la valeur 1 dans $ADFR \in ADCSRA$. La première conversion doit être lancée par $ADSC \in ADCSRA$. Dans ce mode l'**ADC** effectue des conversions sans se préoccuper du flag d'*interruption* **ADIF**. Un exemple de codage est donnée dans la section suivante (cf section 2.2.3).

Conversion avec attente active

Pour ce mode on va définir une fonction ducture qui retournera un entier image de la conversion. L'attente active est du au fait que, après avoir lancée la conversion par **ADSC**. Un exemple de codage est donnée dans la section suivante (cf section ??).

Aspects numériques Par défaut, la fréquence d'horloge d'entrée est entre 50 *kHz* et 200 *kHz* pour obtenir la résolution maximale. Si une résolution plus basse que 10 bits est nécessaire, la fréquence d'horloge d'entrée de l'**ADC** peut alors être plus haute que 200 *kHz*.

Temps de conversion Le temps de conversion qui est égal à 13 fois l'horloge système, peut, de plus, être multiplié, par un facteur de pré-division (cf bits $ADPS_{2-0}$. Le module **ADC** contient un pré-diviseur qui produit une fréquence d'horloge acceptable pour l'**ADC** à partir de celle du **Control Process Unit (CPU)**. La Mise en marche pré-diviseur se fait par **ADPS** $\in ADCSRA$ et le pré-diviseur commence à compter dès que l'**ADC** est allumé en mettant le bit **ADEN** à 1 dans **ADCSRA**.



2.2.3 Les Registres de l'ADC : ADMUX, ADCSRA, ADCH, ADCL

- **ADMUX**

$REFS_1$	$REFS_0$	$ADLAR$	-	MUX_3	MUX_2	MUX_1	MUX_0
----------	----------	---------	---	---------	---------	---------	---------

– $REFS_1, REFS_0$

$REFS_1$	$REFS_0$	Tension de Référence
0	0	$AREF$
0	1	AV_{cc} avec capacité externe sur $AREF$
1	0	non utilisé
1	1	2,56 v

– $ADLAR$: ADC Left Adjust Result Ajustement à gauche à 1 ou à droite à 0 du résultat dans le registre $ADCL$ et $ADCH$.

– $MUX_{3,2,1,0}$: Choix du canal ADC .

- $ADCSRA$

$ADEN$	$ADSC$	$ADFR$	$ADIF$	$ADIE$	$ADPS_2$	$ADPS_1$	$ADPS_0$
--------	--------	--------	--------	--------	----------	----------	----------

– $ADEN$: (AD ENable) Mise en marche du convertisseur avec la mise à 1 du bit, l'arrêt avec la mise à 0, la conversion en cours sera terminée.

– $ADSC$: (AD Start Conversion) Lancement de la conversion de la voie sélectionnée (retourne à 0 en fin de conversion). En mode simple conversion, il faut remettre à 1 à chaque nouvelle conversion. En mode libre, la première conversion dure 25 cycles puis les suivantes 15, il n'est pas nécessaire de remettre le bit à 1 à chaque conversion.

– $ADFR$: (Analog Digital Free Running Mode) La mise à 1 de ce bit permet de mettre en le convertisseur en mode conversion libre : mode de fonctionnement où les conversions ont lieu en permanence sans avoir besoin d'être relancées.

– $ADIF$: (AD Interrupt Flag) Passe à 1 une fois la conversion terminée et déclenche l'interruption si $ADIE = 1$. Ce bit repasse automatiquement à 0 lors du traitement de la routine d'interruption.

– $ADIE$: "AD Interrupt Enable" : Validation de l'interruption du convertisseur.

– $ADPS_2, ..., ADPS_0$: Bits de Sélection du "Prescaler": facteur de pré-division de l'horloge interne du convertisseur en fonction du quartz : rien, 2, 4, 8, 16, 32, 64, 128.

- $ADCH$ – $ADCL$:

– Avec $ADLAR = 0$: On cherche un résultat sur 10 bits

-	-	-	-	-	-	ADC_9	ADC_8
ADC_7	ADC_6	ADC_5	ADC_4	ADC_3	ADC_2	ADC_1	ADC_0

```
int L = ADCL;
int H = ADCH;
int res = (H<<8)+L;
```

– Avec $ADLAR = 1$: On cherche un résultat sur 8 bits en laissant tomber $ADCL$

ADC_9	ADC_8	ADC_7	ADC_6	ADC_5	ADC_4	ADC_3	ADC_2
ADC_1	ADC_0	-	-	-	-	-	-

```
int res = ADCH;
```

Programmes relatifs à l' ADC

-1- Mode scrutatif : attente active. La led qui doit clignoter sur PB_0 .

```

volatile int lu; // variable globale

void lecture_analogique_scrutative(){
    char L;
    ADCSRA |= (1<<ADSC);
    while (ADCSRA & (1<<ADSC)) {};
    lu=ADCH; // lecture sur 8 bits
}

int main(void){
    DDRB=0x1F; DDRC=0x00; // C : entree, B : Sortie
    ADMUX=(1<<ADLAR); //ajust Gauch, PC0 entree analog.
    ADCSRA= (1<<ADEN); // Mise On ADC
    do {
        _delay_ms(200);
        lecture_analogique_scrutative();
    } while(1);
    return(0);
}

```

-2- Mode échantillonné : On réalise ici, un déclenchement échantillonné par le **timer 1**, de la conversion analogique

```

volatile int F;

ISR(ADC_vect){char L=ADCL;F=ADCH; }

ISR(TIMER1_OVF_vect) {ADCSRA |= (1<<ADSC);PORTB^=1;}

int main(void){
    DDRB=0x1F;DDRC=0x00;
    ADMUX=(1<<ADLAR); // ajust Gauche, Lecture sur PC0
    ADCSRA= (1<<ADEN)+(1<<ADIE); // mise ON du can, IT CAN
    TCCR1A=0;TCCR1B = (1 << CS11); // prediv du \timer\ Fcpu/8
    TIMSK = (1<<TOIE1); // Validation It de debordement */
    sei(); // toute les its autorisees
    do {
        PORTB^=1;_delay_ms(50);
    }while(1);
    return(0);
}

```

-3- Free Running : On lance l'**ADC** une fois puis on lit à la volée **ADCH** et **ADCL**.

```

int main(void){
    int H,L;
    DDRB=0x1F; DDRC=0x00;
    ADMUX=(1<<ADLAR);
    ADCSRA= (1<<ADEN) + (1<<ADSC)+ (1<<ADFR);
    do {
        PORTB^=1;
        delai(50);
        L=ADCL; H = ADCH;
        Res = (H<<8)+L;
    } while(1);
    return(0);
}

```


-4- Mode boggué : Sans **timer** relance de l'**ADC** dans l'*interruption* de conversion.

```
volatile int F;

ISR(ADC_vect){F=ADCH; ADCSRA |= (1<<ADSC)} // Aie !!!

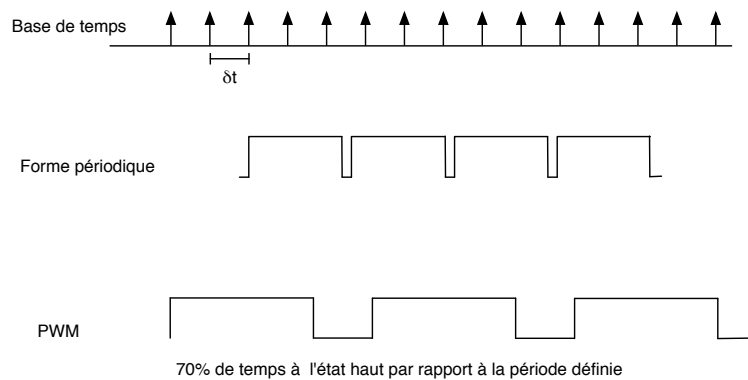
int main(void){
    DDRB=0x1F;DDRC=0x00;
    ADMUX=(1<<ADLAR); // ajust Gauche, Lecture sur PC0
    ADCSRA= (1<<ADEN)+(1<<ADIE); // mise ON du can, IT CAN
    sei(); // toute les its autorisees
    do {
        PORTB^=1;_delay_ms(50);
    }while(1);
    return(0);
}
```

Chapter 3

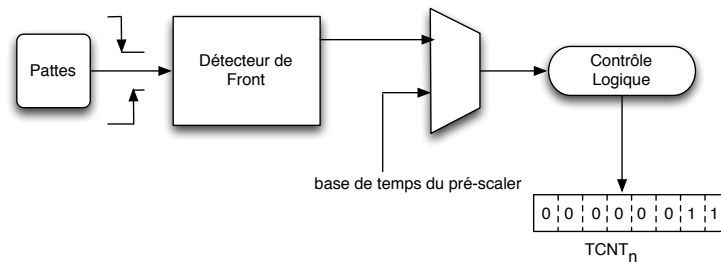
Les Timers

Les fonctions d'un **timer** concernant d'abord les fonctions de temporisation avec la définition de base de temps, la génération de formes (signaux carrés, *MLI*) ainsi que le comptage d'événements. Un **timer** permet aussi de mesurer un temps entre deux événements.

Dans la figure suivante nous illustrons une base de temps et la génération de formes :



Dans cette figure on décrit le comptage ou la capture d'événements :



L'*ATMEGA8* possède 3 **timers** : **timer** 0, **timer** 1 et **timer** 2. Les **timers** 0 et 2 sont des **timers** 8 bits tandis que les **timers** 1 est un **timer** 16 bits. Le **timer** 2 est plus élaboré que le **timer** 0, il permet en plus de faire de la *MLI* et de produire une *interruption* de comparaison.

3.1 Le **timer** 0

3.1.1 Caractéristiques

Ses fonctions de base

- **timer** à sortie unique

- Générateur de fréquences
- Comptage d'événements externes
- Pré-diviseur d'horloge 10 bits
- Le **timer** 0 se manipule à l'aide de seulement 3 registres :
 - $TCNT_0$
 - $TCCR_0$
 - $TIMSK$

Aperçu global du **timer 0** Le **timer** 0 est un **timer** 8 bits. Un débordement provoque l'évènement $TIMER0_OVF_vect$ et la mise à un du drapeau TOV_0 (dans le registre $TIFR$) et possiblement une *interruption* se produit. Le masquage ou l'autorisation de cette *interruption* est réalisé par le bit $TOIE_0$ du registre $TIMSK$. Comme le montre le schéma précédent $TCNT_0$ s'incrémente suivant :

- La patte externe **timer** 0
- Ou le pré-diviseur d'horloge. Ce choix étant fait par les bits $CS0_{2:0} \in TCCR_0$

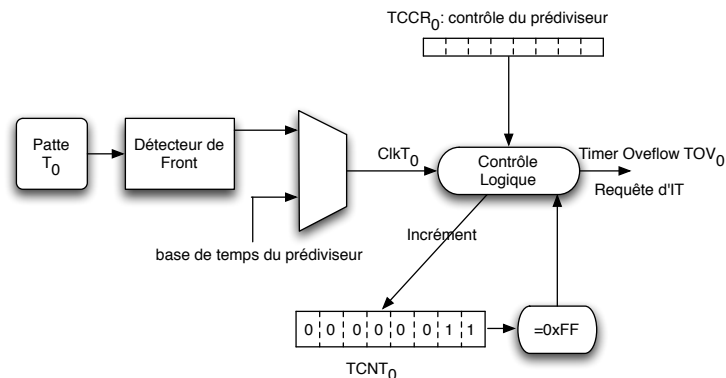


Figure 3.1: Aperçu de **timer** 0

- Le bloc logique ordonne alors l'incrément du registre $TCNT_0$.
- Lorsque $TCNT_0$ atteint la valeur $0xFF$ il repasse à la valeur 0 et dans le même temps le bit TOV_0 passe à un.
- TOV_0 agit comme un 9^{ième} bit.
- Si une *interruption* a été mise en place alors ce bit est **raz** lors de l'exécution du sous-programme d'interruption associé.

3.1.2 Les registres associés au **timer** 0 : $TCCR_0$, $TCNT_0$

- $TCCR_0$

-	-	-	-	-	$CS0_{02}$	$CS0_{01}$	$CS0_{00}$
---	---	---	---	---	------------	------------	------------

<i>CS02</i>	<i>CS01</i>	<i>CS00</i>	<i>Description</i>
0	0	0	T_0 en pause
0	0	1	$clk_{I/O}$: Horloge Système
0	1	0	$clk_{I/O}/8$
0	1	1	$clk_{I/O}/64$
1	0	0	$clk_{I/O}/256$
1	0	1	$clk_{I/O}/1024$
1	1	0	source externe : front descendant sur la patte T0
1	1	1	source externe : front montant sur la patte T0

Remarque 3.1.1 Si par ailleurs, la patte est utilisée en entrée Alors un front sur cette patte affectera quand même **timer** 0 si les modes 6 ou 7 ont été choisis.

- $TCNT_0$: Accessible en lecture ou écriture : Registre de comptage courant.
- $TIMSK$

$OCIE_2$	$OCIE_2$	$TICIE_1$	$OCIE_{1A}$	$OCIE_{1B}$	$TOIE_1$	-	$TOIE_0$
----------	----------	-----------	-------------	-------------	----------	---	----------

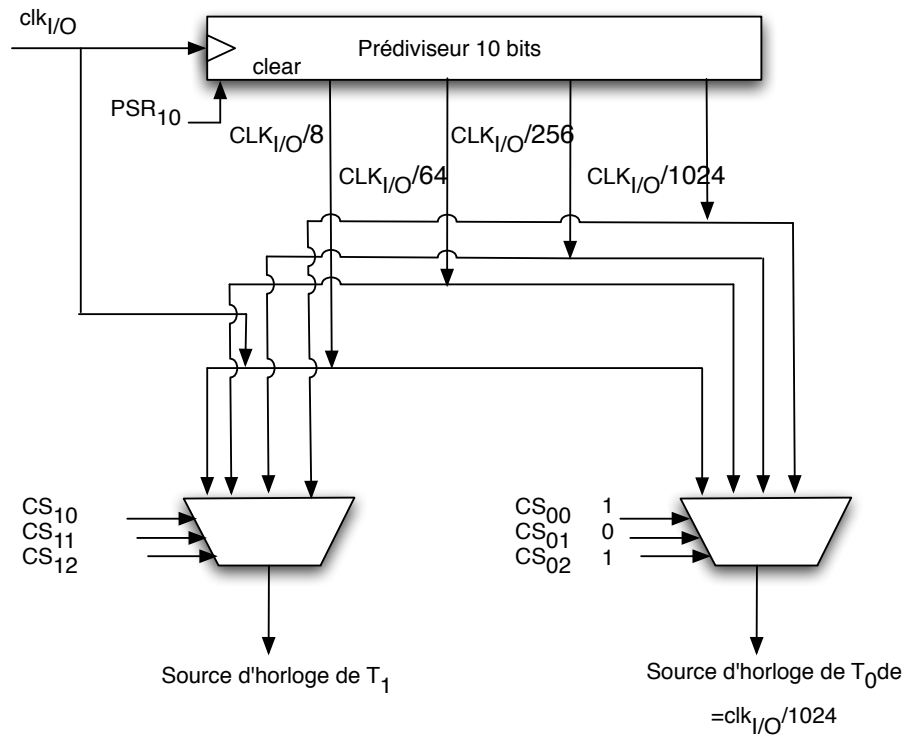
La condition d'autorisation de l'*interruption* de débordement de **timer** 0 est $TOIE_0 = 1$ ET $I = 1 \in SR$). Sur un débordement de $TCNT_0$, on a :

- $TOV_0 \in TIFR$ qui passe à un, **et** le programme associé à l'*interruption* est alors exécuté.
- TOV_0 est alors automatiquement **raz** dès le début du sous-programme d'IT.

Sinon en mode non interruptif, c.a.d. en mode scrutatif il faut remettre a zéro ce bit **EN Y ECRIVANT UN "1"**.

3.1.3 Le prédiviseur

Les timers **timer** 0 et **timer** 1 partagent le même module de pré-division, mais ceux-ci peuvent avoir des fréquences différentes de pré-divisions par rapP à la source d'Horloge Interne. Le Timer/Compteur peut être cadencé directement par l'horloge de système (avec $CS0_{2:0} \in TCCR_0$ (**timer** 0) ou $CS1_{2:0} \in TCCR_1$ à 1) permettant une fréquence d'horloge maximale : $clk_{I/O}$. Autrement, on pourra choisir un facteur de pré-division suivant : $clk_{I/O}/8, clk_{I/O}/64, clk_{I/O}/256, clk_{I/O}/1024$.



Il y a la possibilité d'échantillonner la source externe à travers un détecteur de front qui induira alors un retard de 2.5 à 3.5 cycles d'horloge.

SFIOR

$ADTS_2$	$ADTS_1$	$ADTS_0$	-	ACME	PUD	PSR_2	PSR_{10}
----------	----------	----------	---	------	-----	---------	------------

- Quand $PSR_{10} \leftarrow 1$ le prédiviseur de timer 0 et timer 1 est **raz**.
- Attention ce prédiviseur est partagé par timer 0 et timer 1, et un **reset** du prédiviseur les affectent tous les 2.

Exemple d'utilisation du **timer** zéro avec la mise en place d'une interruption de débordement :

```
ISR (TIMER0_OVF_vect) {
    static int Compteur;
    if (Compteur++ == 50) {
        Compteur=0;
        PortB^=1 // permet de mesurer la periode
    }
}

void configTimer0(){
    TCCR0 = (1<<CS02) + (1<<CS00) ; // clkio/1024
    TIMSK = 1<<TOIE0; // Autorisation IT de debordement
}

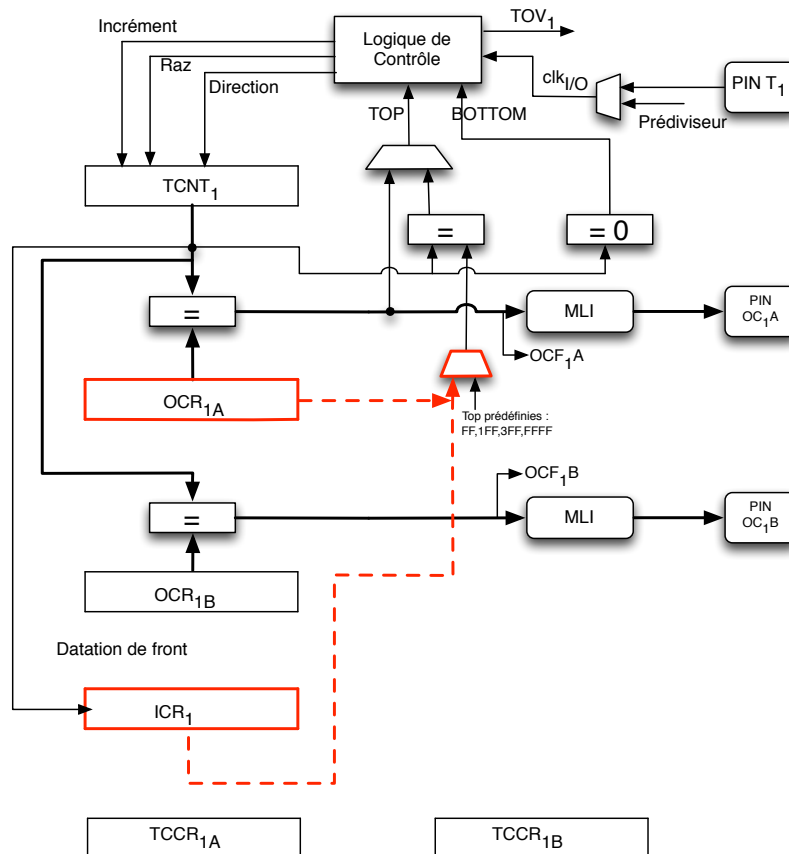
int main() {
    DDRB=0xFF;
    configTimer0();
    sei(); // autorise ttes les interruptions
    while(1);
}
```

3.2 Le **timer** 1

3.2.1 Caractéristiques générales

Ses fonctions de base

- **timer** 16 bits ;
- 2 sorties indépendantes de comparaison ;
- Une entrée de capture avec réduction de bruit ;
- 16 modes de *PWM*
- Générateur de fréquence
- Compteur externe d'événements
- 4 sources d'IT : TOV_1 , OCF_{1A} , OCF_{1B} , ICF_1 .



Les registres 16 bits

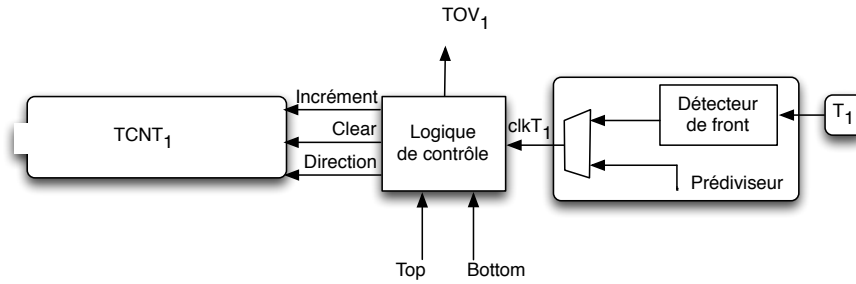
- le registre de comptage $TCNT_1$
- les registre de sortie de comparaison : OCR_{1A} - OCR_{1B}
- le registre de capture d'entrée ICR_1

les registres 8 bits

- Les registres de contrôle du **timer** $TCCR_{1A}$ - $TCCR_{1B}$
- le registre de sortie de comparaison : OCR_{1A} - OCR_{1B}
- le registre de capture d'entrée ICR_1

Les interruptions du **timer 1 :** Tous les signaux d'IT sont visibles depuis $TIFR$. Toutes les *interruptions* sont masquables individuellement dans $TIMSK$

Contrôle de $TCNT_1$: Le **timer** 1 peut être fréquenté de façon interne par le prédiviseur ou par une source externe patte **timer** 1. Le bloc logique détermine comment la source est utilisée pour incrémenter ou décrémenter $TCNT_1$. Le **timer** 1 est inactif quand il n'y a aucune source. La source $clkT_1$ de la figure (en haut à droite de la figure 3.2.1) est sélectionnée par $CS_{12:0} \in TCCR_{1B}$.



Comparaison de seuil : Les deux registres de comparaison OCR_{1A} - OCR_{1B} sont utilisés comme des seuils qui sont comparés avec $TCNT_1$. Le résultat ces comparaisons peut être utilisé pour générer une forme de type *PWM* ou bien une fréquence variable sur les pattes $OC1_B, OC1_B$. Ces comparaisons d'égalité vont positionner les flags OCF_{1A} ou OCF_{1B} qui peuvent alors être utilisés pour générer l'*interruption* correspondante.

La capture : Le registre de capture d'entrée ICR_1 peut capturer un événement (front) sur la patte ICP_1 . Cette partie inclue un filtre réducteur de bruit évitant la capture de fronts parasites.

Le choix de la valeur maximum : La valeur *TOP* est soit le maximum du *timer* ($0xFFFF$) ou bien une valeur définie dans ICR_1 ou OCR_{1A} . L'utilisation de OCR_{1A} pour définir la valeur *TOP*, bloque la génération de *PWM* pour la sortie $OC1A$. Lorsque l'on utilise ICR_1 pour définir la valeur *TOP* les deux sorties *PWM* sont alors disponibles (OCR_{1A} et OCR_{1B}), par contre l'entrée de capture n'est plus disponible.

<i>BOTTOM</i>	<i>BOTTOM</i> = $0x0000$
<i>MAX</i>	<i>MAX</i> = $0xFFFF$
<i>TOP</i>	$TCNT_1$ atteint <i>TOP</i> c.a.d quand il atteint soit : – $0xFF$, $0x1FF$, $0x3FF$ – OCR_{1A} , ou bien ICR_1

Fonctionnement de l'Unité de Comptage

Le registre $TCNT_1$ est piloté par le bloc logique via les signaux *clear*, *increment* ou *decrement* et part la source $clkT_1$. $clkT_1$ peut être généré par une source externe ou interne selon les bits $CS12:0$. Quand $CS12:0 = 0$, le *timer* 1 est arrêté, la valeur de $TCNT_1$ étant toujours accessible par le *CPU*. Une écriture par le *CPU* sur $TCNT_1$ à la priorité sur toute autre opération.

La séquence de comptage peut alors déterminer une forme d'onde (waveform) sur la patte $OC1A$ selon les bits $WGM1_{3:0}$ des registres $TCCR_{1A}$ et $TCCR_{1B}$. Les formes d'onde dépendent en fait des modes de comptage (cf section 3.2.3) sur $TCNT_1$. Le bit de débordement TOV_1 dépend aussi du mode choisi sur $WGM1_{3:0}$.

Unité de capture

L'unité de capture peut capturer des événements externes en leur attachant une étiquette temporelle ¹ attachée à cette occurrence. Le signal externe indiquant l'occurrence d'un ou de plusieurs événements est disponible sur la patte ICP_1 ou par le comparateur analogique. Les étiquettes temporelles peuvent être utilisées pour calculer une fréquence, un rapport cyclique ou d'autres. Ces étiquettes temporelles sont parfois utilisées pour créer un "log" d'évènements.

Aperçu de l'Unité de Capture Quand un événement (front) se produit (cf figure 3.2.1) sur ICP_1 , ou sur ACO , ce signal passe par le réducteur de bruit et le détecteur de front. Si il passe cet étage, il déclenche l'écriture de $TCNT_1$ dans ICR_1 , $TCNT_1$ constitue alors ce que l'on appelle une étiquette temporelle. Dans le même temps le drapeau ICF_1 passe à un.

¹qui est la valeur courante de $TCNT_1$

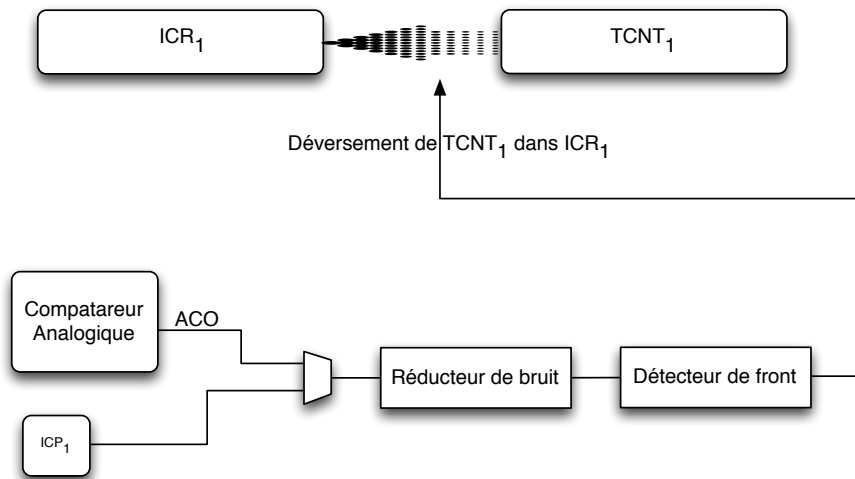


Figure 3.2: Unité de capture

En mode interruptif, Si $TICIE_1$ passe à un, une *interruption* de capture est déclenchée. ICF_1 est automatiquement remis à zéro dans le sous-programme d'IT.

Remarque 3.2.1 ICR_1 ne peut être mis à jour que lorsque l'on utilise le mode qui utilise ICR_1 fixant la valeur TOP . Dans ce cas, les bits $WGM1_{3:0}$ doivent être positionnés avant que ICR_1 soit initialisé avec la valeur TOP . On écrira dans ICR_1 d'abord l'octet de poids faible puis le poids Fort.

La source de capture : La patte de capture est $ICP1$. Cette source peut aussi être connectée au comparateur analogique. Le comparateur est sélectionné avec le bit $ACIC$ du registre ACS .

Remarque 3.2.2 Attention, changer la source peut provoquer une capture, le flag ICF doit donc être **raz** après ce changement.

Les pattes ICP_1 et la patte de sortie ACO sont échantillonnées avec les mêmes techniques déjà évoquées : Le détecteur de front est identique. Quand le réducteur de bruit est activé, de la logique est rajoutée avant le détecteur de front ce qui ralentit le signal de 4 cycles d'horloge. Une entrée de capture peut être déclenchée par programme en contrôlant le $PORTC$ contenant la patte $ICP1$.

Réducteur de bruit C'est en fait un filtre numérique qui délivre la sortie **ssi** les 4 signaux échantillonnés consécutifs sont égaux. Le réducteur est activé par le bite $ICNC_1$ du registre $TCCR1B$.

Capture des événements Tout évènement capturé écrase le précédent même si il n'a pas été traité par le **CPU**. Aussi, dans un sous-programme d'*interruption* il faudra lire ICR_1 au plus tôt. On prendra garde à ne pas changer la valeur TOP lors de l'utilisation de capture. Lors de la mesure d'un signal de type *PWM*, la détection est changée après chaque capture et cela doit donc être fait aussitôt que ICR_1 a été lu.

Unité de Comparaison de sorties

Les sorties $OC1A/B$ sont controlées par les bits $COM1x1 : 0$. Un de ces bits à un provoque la sortie du générateur de forme.

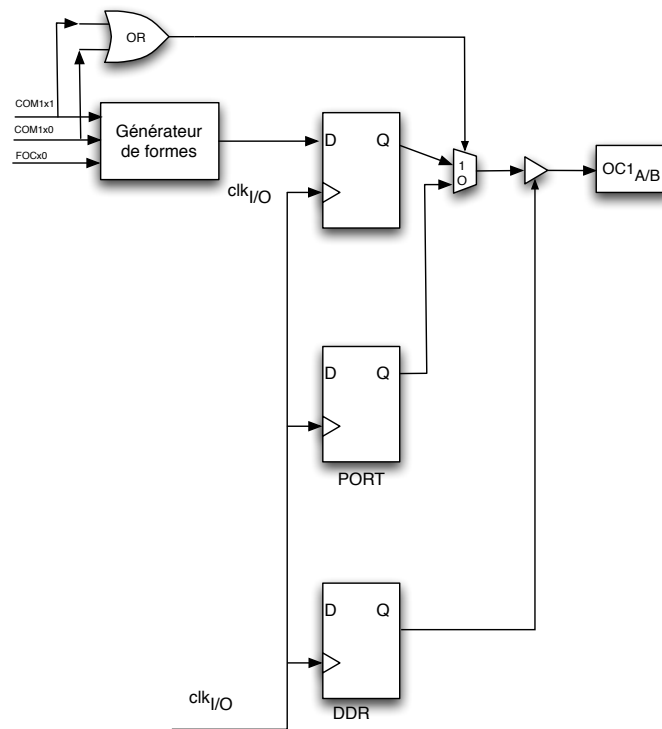


Figure 3.3: Contrôle de la sortie du comparateur

La forme produite sur les pattes **OC1A** ou **OC1B** dépend des bits **WGM13:0** et des modes de comparaison défini par **COM1x1 : 0**, ainsi que des valeurs pour **TOP** et **BOTTOM**. Il faut noter que cette figure montre que les pattes **OC1A** ou **OC1B** doivent avoir été déclarées en sortie à l'aide du registre **DDR**.

3.2.2 Les 16 modes du **timer 1**

Le Comparateur permet de définir la valeur **TOP** de **TCNT1**. Le comparateur peut fonctionner suivant différents modes.

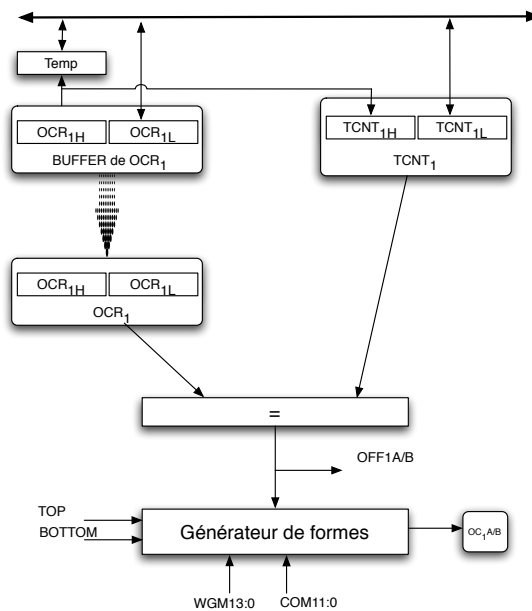


Figure 3.4: Unité de comparaison

Initialisation des registres et pattes de sorties

$TCNT_1$: L'écriture dans **$TCNT_1$** quel que soit le mode va bloquer les comparaisons pour un cycle d'horloge. Il y a des risques à changer **$TCNT_1$** en utilisant le comparateur : Si la valeur écrite dans **$TCNT_1$** est égale à **OCR_{1A}** ou **OCR_{1B}** alors le résultat de la comparaison sera perdu. Il ne faut pas écrire dans **$TCNT_1$** une valeur égale à **TOP** en mode **PWM** à valeur variable. La comparaison avec **TOP** échouera alors et continuera jusqu'à **$0xFFFF$** . Même effet si l'on écrit dans **$TCNT_1$** une valeur égale à **$BOTTOM$** en décomptant.

Concernant les pattes **$OC1A$** et **$OC1B$** , il faut faire attention à positionner **$OC1A$** et **$OC1B$** , en sortie (**PB_6, PB_7**) avant de produire une MLI par l'écriture de **DDR_B** .

Le générateur de formes est défini par les bits **$COM1x1 : 0$** qui permettent de définir 16 modes regroupés en 5 catégories :

- Normal
- **CTC**
- **PWM** rapide
- Phase correcte **PWM**
- Phase et fréquence correcte **PWM**

modes normal, **CTC** , fast **PWM** , ... En mode **PWM** **$COM1x1 : 0$** définissent si la **PWM** est inversée ou non. Les modes sont complètement définis avec : **$WGM1_{3:0}$** et **$COM1x1 : 0$** .

3.2.3 Les Modes du générateur de formes

Les 16 différents modes sont déterminés par les bits : **$WGM1_{3:0}$**

a) Le mode Normal : **$WGM1_{3:0}=0$**

La sortie **OC_{1A}** commute à chaque fois que **$TCNT_1$** atteint la valeur **TOP** (si **$COM1A_{1:0}=1$**)

- Dans ce mode la fréquence max est :

$$f_{OC1A} = \frac{f_{clk_{I/O}}}{2}$$

- Sinon la fréquence est définie par :

$$f_{OC1A} = \frac{f_{clk_{I/O}}}{2 \cdot N \cdot (1 + OCR_{1A})}$$

N représente le facteur de prédivision (1,8,64,256,1024)

Attention à ne pas changer la valeur **TOP** avec une valeur inférieure à la valeur courante du comparateur **OCR_{1A}** .

c) Le mode **PWM** rapide **$WGM1_{3:0}=5,6,7,14$** ou **15**

Cette **PWM** permet de générer de plus hautes fréquences. Elle est dite à simple pente et on peut ici avoir 2 commutations de la sortie sur la même pente. Ici, **$TCNT_1$** compte de **$BOTTOM$** à **TOP** et fait alors une comparaison avec un seuil (**OCR_{1A}**), pendant la montée puis revient à **$BOTTOM$** .

Sortie

- En mode non inversé, la sortie est **raz** sur égalité avec **OCR_{1A}** et **mise à un** sur **$BOTTOM$** .
- En mode inversé, la sortie est à un sur égalité et **raz** sur **$BOTTOM$** .
- Du fait de son fonctionnement en simple pente, on va 2 fois plus vite.

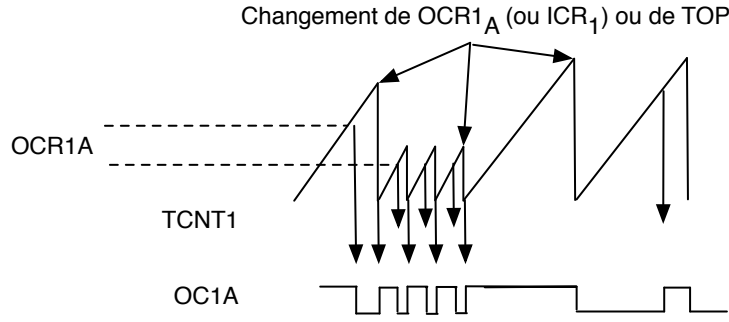


Figure 3.5: Le mode *PWM* rapide

Fréquence Dans ce mode la fréquence est définie par :

$$f_{OC1A\,PWM} = \frac{f_{clk_{I/O}}}{N \cdot (1 + TOP)}$$

N représente le facteur de prédivision (1,8,64,256,1024)

Valeurs particulières Attention à ne pas changer la valeur TOP avec une valeur inférieure à la valeur courante de celle du comparateur. De plus, pour une valeur $TOP = 0$ on aura un quasi front à chaque top d'horloge.

d) Le mode *PWM* à phase correcte $WGM1_{3:0}=1,2,3,10$ ou 11

Ce mode permet d'obtenir une haute résolution de *PWM* à phase correcte. On est cette fois ci sur du double pente avec comparaison de seuil. Le compteur compte de la valeur $BOTTOM$ à TOP puis il décrémente jusqu'à $BOTTOM$ et recommence indéfiniment.

Instant de commutation de $OC1_x$ Cette commutation dépend du registre $OCR1_x$ et de la valeur TOP . On voit ces instants de commutation dans la figure 3.2.3 ou deux événements se produisent lors des montées et des descentes :

- En mode non inversé, $OC1_x$ est **raz** sur l'égalité de $TCNT1$ avec $OCR1_x$ en comptant, et **mis à un** sur l'égalité de $TCNT1$ avec $OCR1_x$ en décomptant.
- En mode inversé, $OC1_x$ est **mis à un** sur l'égalité de $TCNT1$ avec $OCR1_x$ en comptant, et **raz** sur l'égalité de $TCNT1$ avec $OCR1_x$ en décomptant.

Remarque 3.2.3 $OCR1_x$ sont mis à jour sur TOP . Cela signifie que si $OCR1_x$ est modifié pendant une pente, sa nouvelle valeur n'est prise en compte que sur le prochain TOP

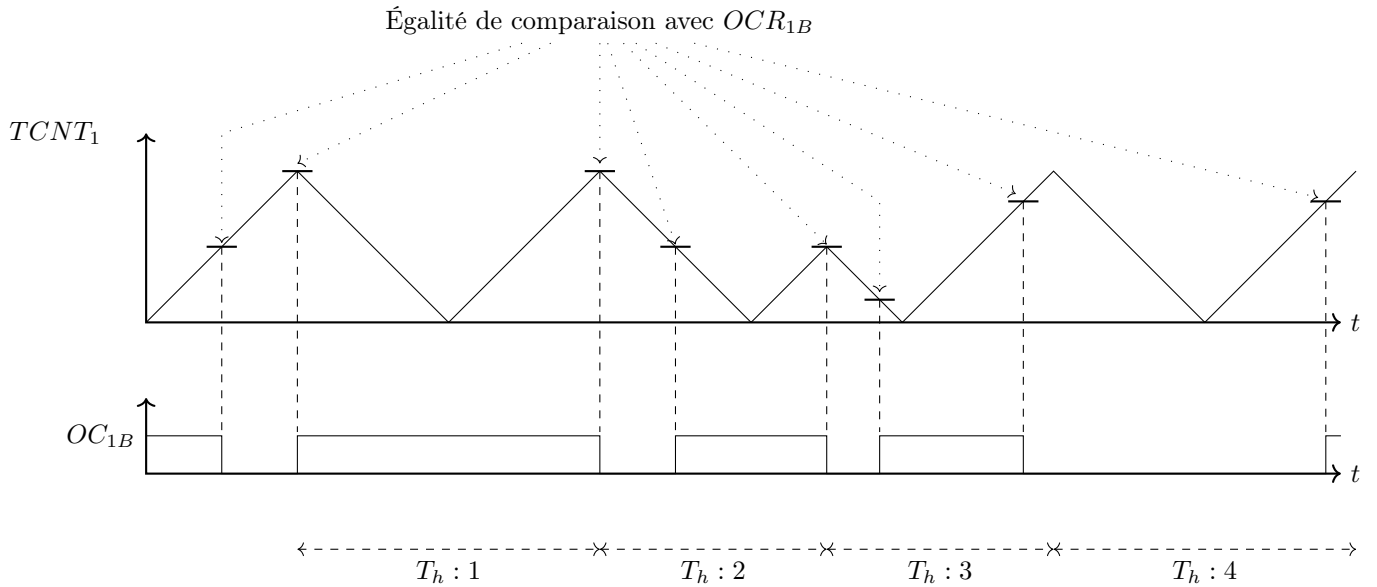
Ce mode à double pente à une fréquence de hachage maximale évidemment deux fois plus petite que celle à simple pente, mais son caractère symétrique le rend plus utilisé dans les applications de commande de moteurs.

Résolution La résolution varie de la valeur 2 bits à 16 bits, elle dépend de la valeur de $OCR1_A$ ou $ICR1$. La résolution de la *PWM* est donnée par l'équation :

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

Dans ce modu compteur est incrémenté jusqu'à ce que la valeur atteigne soit une des valeurs : 0x00FF, 0x01FF, 0x03FF ($WGM1_{3:0} = 1, 2$, ou 3), soit la valeur dans $ICR1$ ($WGM1_{3:0} = 10$), soit la valeur dans $OCR1_A$ ($WGM1_{3:0} = 11$).

Dès que $TCNT1$ a alors atteint TOP alors $OC1A$ commute et $TCNT1$ décompte dès lors.



Attention Là encore, lorsque l'on change la valeur TOP on doit s'assurer qu'elle est supérieure à la valeur courante de $TCNT_1$

Fréquence Dans ce mode la fréquence est définie par :

$$f_{OC1A/PCPWM} = \frac{f_{clk_{I/O}}}{2.N.(1 + TOP)}$$

N représente le facteur de prédivision (1,8,64,256,1024)

Mode inversé ou non En positionnant les bits $COM1x1:0$ à 2 on aura du non inversé. En positionnant les bits $COM1x1:0$ à 3 on aura de l'inversé

e) Le mode PWM à phase et fréquence correcte : "MLI Centrée" $WGM1_{3:0} = 8$ ou 9

Ce mode fournit une haute résolution à phase et fréquence correcte. Ce mode comme le précédent est basé sur une double pente. Il compte indéfiniment de $BOTTOM(0x0000)$ à TOP et de TOP à $BOTTOM$

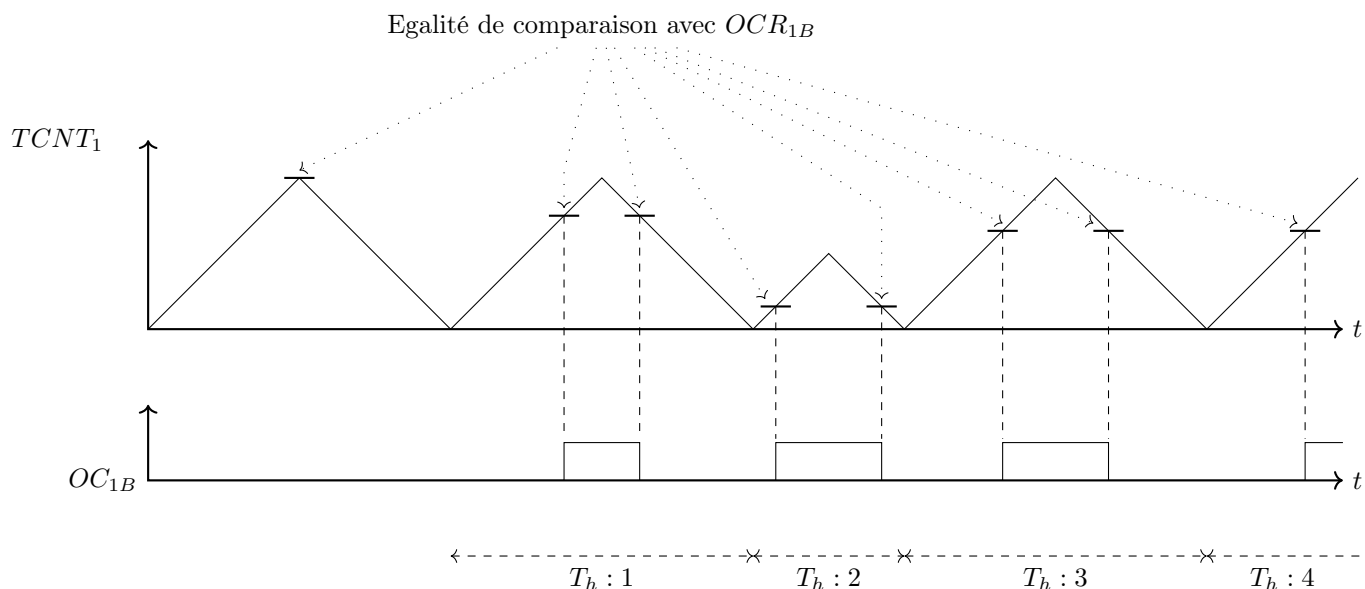
En mode non inversé (cf figure suivante) les sorties $OC1A$ et $OC1B$ sont remises à zéro sur l'égalité entre $TCNT_1$ et OCR_{1x} en comptant et mis à un sur égalité en décomptant. En mode inversé c'est l'inverse.

Remarque 3.2.4 OCR_{1x} sont mis à jour sur $BOTTOM$. Cela signifie que si OCR_{1x} est modifié dans une pente alors sa nouvelle valeur n'est prise en compte que sur le prochain $BOTTOM$.

L'opération en double pente donne une fréquence max plus basse comparée à celle en simple pente. Cependant son caractère très symétrique en fait le modu plus utilisé en terme de contrôle moteur.

Résolution La résolution de la PWM est donnée par l'équation :

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$



Au contraire du mode précédent il y a une symétrie dans le signal quand on examine sa période. De plus, OC_{1X} ne sera visible sur le q que si le $PORTB$ a été programmé en sortie : avec DDR_B .

Inversé/non inversé En positionnant les bits $COM1x1 : 0$ à 2 on aura du non inversé. En positionnant les bits $COM1x1 : 0$ à 3 on aura de l'inversé.

Fréquence Dans ce mode la fréquence est définie par :

$$f_{OC_{1A}PFCPWM} = \frac{f_{clk_{I/O}}}{2 \cdot N \cdot (1 + TOP)}$$

N représente le facteur de prédivision (1,8,64,256,1024)

3.2.4 Inventaire des registres utiles au **timer 1**

Le registre $TCCR_{1A}$

COM_{1A1}	COM_{1A0}	COM_{1B1}	COM_{1B0}	FOC_{1A}	FOC_{1B}	WGM_{11}	WGM_{10}
-------------	-------------	-------------	-------------	------------	------------	------------	------------

Bits $COM1x1, COM1x0$ en mode *normal* et *CTC*

$COM1x1$	$COM1x0$	Description
0	0	P libre $OC1x$ déconnectés
0	1	changement d'état quand $OC1x = TCNT1$
1	0	raz sur égalité
1	1	mis à un sur égalité

Bits $COM1x1$, $COM1x0$ en mode fast PWM

$COM1x1$	$COM1x0$	Description
0	0	P libre $OC1_x$ déconnectés
0	1	$WGM1_{3:0} = 15$ $OC1_A$ commute sur égalité $OC1_B$ déconnectés
1	0	raz $OC1_x$ sur égalité mis à un sur $BOTTOM$
1	1	mis à un sur égalité raz sur $BOTTOM$

Bits $COM1x1$, $COM1x0$ en mode PWM à phase correcte et PWM à phase et fréquence correcte

$COM1A_1$ $COM1B_1$	$COM1A_0$ $COM1B_0$	Description
0	0	P libre $OC1_x$ déconnectés
0	1	$OC1_A$ commmute sur égalité, $OC1_B$ déconnecté
1	0	raz $OC1_x$ sur égalité en incrémentant mis à un en décrémentant sur égalité
1	1	mis à un sur égalité en incrémentant raz sur égalité en décrémentant

Bits $FOC1_x$ Ces bits $FOC1_x$ ne sont actifs que si on n'est pas en PWM mode. Cependant ces bits doivent être **raz** en PWM mode. $FOC1_x \leftarrow 1$ force à l'instant choisi, une comparaison entre $TCNT_1$ et OCR_{1A} (par ex.). Ce sont des bits d'échantillonnage. En lecture ces bits valent zéro.

$WGM1_{1:0}$ Waveform Generation Mode Ces bits déterminent le sens du comptage, TOP et le type de forme et on l'a vu précédemment, ils déterminent aussi le mode : normal, CTC , et les trois modes PWM :

- PWM rapide : Fast PWM
- PWM à phase correcte : Phase correct PWM .
- PWM à phase et fréquence correcte : Phase and frequency correct PWM .

Le registre $TCCR_{1B}$

$ICNC_1$	$ICES_1$	-	$WGM1_3$	$WGM1_2$	CS_{12}	CS_{11}	CS_{12}
----------	----------	---	----------	----------	-----------	-----------	-----------

$ICNC_1$: Réduction de bruit La réduction de bruit se fait par $ICNC_1$: Input Capture Noise Canceler. Si $ICNC_1 \leftarrow 1$ cela active le réducteur de bruit de l'entrée de capture.

Quand le réducteur de bruit est activé l'entrée de capture ICP_1 est filtrée. Le filtre affecte sa sortie si on a 4 sorties successives égales ce qui induira un délai de 4 cycles d'horloges.

$ICES_1$: La sélection du type d'évènement La sélection du type d'évènement se fait par $ICES_1$: Input Capture Edge Select. Ce bit sélectionne quel type de front permet de déclencher la capture. Quand $ICES_1 = 0$ c'est un front descendant, lorsque $ICES_1 = 1$ c'est un front montant.

Quand une capture est déclenchée, $TCNT_1$ est vidé dans ICR_1 . $ICF1$ est positionné à un et peut générer une *interruption* . Quand c'est ICR_1 qui détermine TOP alors ICP_1 est déconnectée et le mécanisme de capture ne fonctionne plus.

$WGM13 : 2$ Waveform Generation Mode (cf 3.2.2)

Bit CS12:0 Clock Select Ces trois bits sélectionnent la prédivision : division de $Cllk_{io}$ par $N = (1, 8, 64, 256, 1024)$.

Les Registres $TCNT_1, OCR_{1x}, ICR_1$

$TCNT_1$ est accessible en lecture et en écriture. Modifier $TCNT_1$ pendant le comptage peut produire un effet indésirable si la nouvelle valeur est supérieure aux valeurs de comparaison : on manque alors une comparaison.

OCR_{1x} , les registres de comparaison contiennent une valeur 16 bits qui est continuellement comparée avec la valeur courante de $TCNT_1$. L'égalité est utilisée pour générer une *interruption* ou une forme sur la patte $OC1_x$.

ICR_1 est mis à jour avec la valeur du compteur $TCNT_1$ à chaque fois qu'un événement se produit sur la patte $ICP1$. Ce registre peut aussi être utilisé pour définir la valeur TOP .

Le registre $TIMSK$

$OCIE_2$	$OCIE_2$	$TICIE_1$	$OCIE_{1A}$	$OCIE_{1B}$	$TOIE_1$	-	$TOIE_0$
----------	----------	-----------	-------------	-------------	----------	---	----------

$TICIE_1$: "Timer/Counter1, Input Capture Interrupt Enable" :

Quand $TICIE_1 \leftarrow 1$, le bit I du registre d'état est mis à un, toutes les *interruptions* sont donc globalement autorisées et en particulier l'interruption de capture. L'*interruption* ad hoc est alors exécutée lorsque $ICF_1 \in TIFR$ reçoit un.

$OCIE_{1A}$: "Timer/Counter1, Output Compare A Match Interrupt Enable" : Quand $OCIE_{1A} \leftarrow 1$, le bit I du registre d'état est mis à un, toutes les *interruptions* sont donc globalement autorisées et en particulier l'interruption de comparaison. L'*interruption* ad hoc est alors exécutée lorsque $OCF_{1A} \in TIFR$ reçoit un.

$OCIE_{1B}$: "Timer/Counter1, Output Compare B Match Interrupt Enable" : Quand $OCIE_{1A} \leftarrow 1$, le bit I du registre d'état est mis à un, toutes les *interruptions* sont donc globalement autorisées et en particulier l'interruption de comparaison. L'*interruption* ad hoc est alors exécutée lorsque $OCF_{1B} \in TIFR$ reçoit un.

$TOIE_1$: "Timer/Counter1, Overflow Interrupt Enable" :

Quand $TOIE_1 \leftarrow 1$, le bit I du registre d'état est mis à un, toutes les *interruptions* sont donc globalement autorisées et en particulier l'interruption de comparaison. L'*interruption* ad hoc est alors exécutée lorsque $TOV_1 \in TIFR$ reçoit un.

Le registre $TIFR$

OCF_2	TOV_2	IC_1	OCF_{1A}	OCF_{1B}	TOV_1	-	TOV_0
---------	---------	--------	------------	------------	---------	---	---------

ICF_1 : "Timer/Counter1, Input Capture Flag" :

ICF_1 est un drapeau (flag) qui reçoit un quand la patte $ICP1$ reçoit un signal. Quand ICR_1 est utilisé pour stocker TOP avec un mode de $WGM1_3 : 0$, ICF_1 est positionné à un quand le compteur atteint TOP . ICF_1 est automatiquement **raz** par le sous-programme d'*interruption* est exécuté. Sinon ICF_1 peut être **raz** en y écrivant un un.

OCF_{1A} : "Timer/Counter1, Output Compare B Match Flag" :

OCF_{1A} est un drapeau (flag) qui reçoit un quand l'égalité de OCR_{1A} avec $TCNT_1$ se produit. OCF_{1A} est automatiquement **raz** par le sous-programme d'*interruption* est exécuté. Sinon OCF_{1A} peut être **raz** en y écrivant un un.

OCF_{1B} : "Timer/Counter1, Output Compare B Match Flag" :

Ce bit est un drapeau (flag) qui reçoit un quand l'égalité de OCR_{1B} avec $TCNT_1$ se produit. OCF_{1B} est automatiquement **raz** par le sous-programme d'*interruption* est exécuté. Sinon OCF_{1B} peut être **raz** en y écrivant un un.

TOV₁: "Timer/Counter1, Overflow Flag" :

Dans les modes normal et *CTC* modes, **TOV₁** est mis à un sur un dépassement de capacité **TOV₁** est automatiquement **raz** par le sous-programme d'*interruption* est exécuté. Sinon **TOV₁** peut être **raz** en y écrivant un un.

3.2.5 Résumé des 16 modes de *MLI* du **timer 1**

<i>v</i>	<i>WGM₁₃</i>	<i>WGM₁₂</i>	<i>WGM₁₁</i>	<i>WGM₁₀</i>	Mode des Timers	<i>TOP</i>	Mise à jour de <i>OCR1_x</i>	Mise à un de <i>TOV₁</i>
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM Phase Correcte 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM Phase Correcte 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM Phase Correcte 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	<i>OCR1_A</i>	immediate	MAX
5	0	1	0	1	Fast PWM 8-bit	0x0FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM 9-bit	0x1FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM 10-bit	0x3FF	BOTTOM	TOP
8	1	0	0	0	PWM Phase, freq. correctes	<i>ICR₁</i>	BOTTOM	BOTTOM
9	1	0	0	1	PWM Phase, freq. correctes	<i>OCR1_A</i>	BOTTOM	BOTTOM
10	1	0	1	0	PWM Phase Correcte	<i>ICR₁</i>	TOP	BOTTOM
11	1	0	1	1	PWM Phase Correcte	<i>OCR1_A</i>	TOP	BOTTOM
12	1	1	0	0	CTC	<i>ICR₁</i>	immediate	MAX
13	1	1	0	1	Réservé	—	—	—
14	1	1	1	0	Fast PWM	<i>ICR₁</i>	BOTTOM	TOP
15	1	1	1	1	Fast PWM	<i>OCR1_A</i>	BOTTOM	TOP

3.2.6 Exemples d'utilisation du **timer 1**

-1- *MLI* centrée sur la patte *PB₂(OC1_B)*.

```
int main (void) {
    int i;

    DDRB = 0xFF;                //Port B en sortie

    // TCCR1A : Mli centree : Mode 9, RAZ sur egal. en incr. et SET sur egal.
    // en decr.
    TCCR1A= (1<<COM1B1)+(1<<WGM10); //0b00100001;

    // TCCR1B : suite des bits de WGM et predivision = 1
    TCCR1B=(1<<WGM13)+(1<<CS10); //0b00010001;

    // OCR1A : valeur de Thash
    OCR1A=10000;
    // OCR1B : valeur tu Ton
    OCR1B=5000;
    while(1);
}
```

-2- Interruption de débordement du **timer 1**:

```
#define LedToggle PortB^=1
volatile byte Compteur;

ISR (TIMER1_OVF_vect)
{ byte i;
  TCNT1 = 0;
  if (Compteur++ == 50) {
    Compteur=0;
    LedToggle; // mesurer la periode sur PB0
  }
```

```

    }
}

void configTimer1(){ //
    TCCR1A = 0;      // Mode normal
    TCCR1B = 1<<CS10; // clkio
    TIMSK = 1<<TOIE1;
}

int main() {
    DDRB=0xFF;
    configTimer1();
    sei();           // autorise les interruptions
    while(1);
}

```

3.3 Le timer 2

Les fonctions du **timer 2** sont la génération de fréquences, le comptage, le mode **raz** de $TCNT_2$ sur comparaison avec rechargement automatique et la fonction génération de formes (waveform). Celle-ci permet les modes suivant :

- mode sans erreur ;
- mode *PWM* à phase correcte ;

3.3.1 Aperçu du timer 2

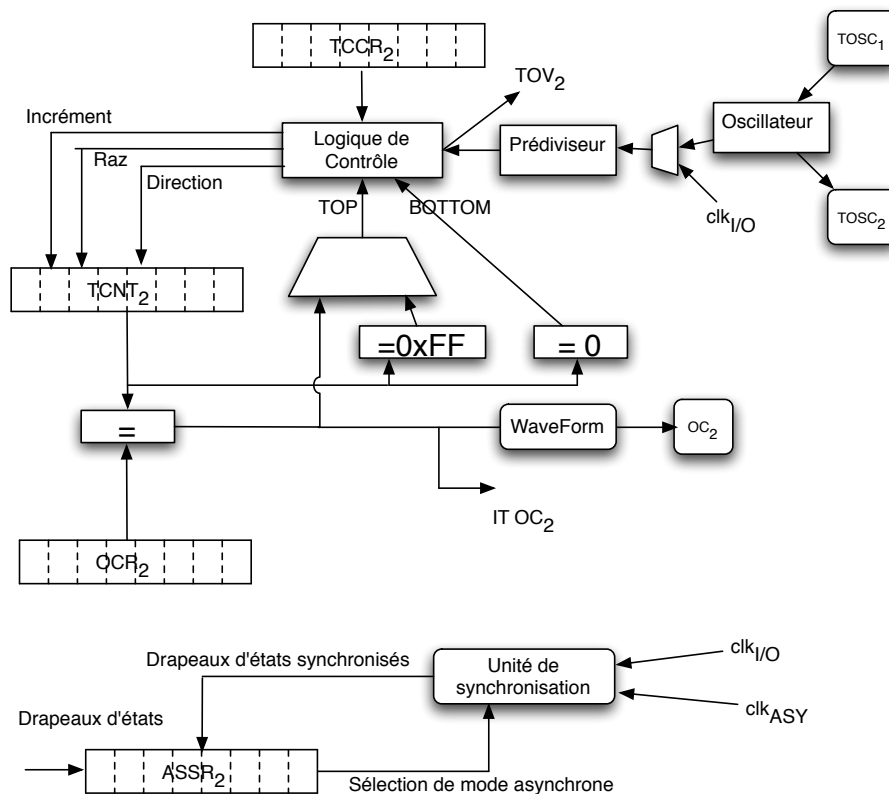


Figure 3.6: Aperçu du **timer 2**

Le **timer 2** est un compteur 8 bits aussi, $TCNT_2$ et OCR_2 sont des registres 8-bit. Les ITs sont visibles dans $TIFR$ et masquables dans $TIMSK$. Le **timer** peut être clocké de façon interne par le prédiviseur ou de façon asynchrone par $TOSC_1, TOSC_2$. Cette synchronisation est contrôlée par le registre $ASSR$. Le block logique sélectionne la source

d'incrémation. OCR_2 est comparé à tout moment à $TCNT_2$ et le test de cette égalité peut être utilisé à tout moment pour générer une forme de type *PWM* ou pour faire varier une fréquence sur OC_2 . Le débordement génère sur une *interruption* par le flag OCF_2 On a $BOTTOM = 0$, $MAX = 0xFF$, de plus TOP peut être soit MAX soit la valeur placée dans OCR_2

Contrôle du comptage

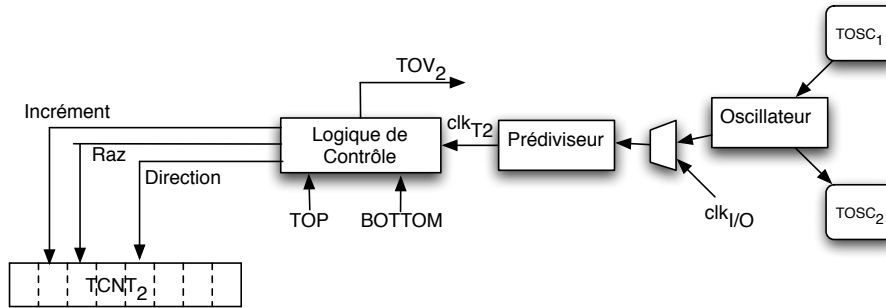


Figure 3.7: Contrôle du comptage

- Comptage : incrémation ou décrémation ;
- clk_{T2} horloge d'incrément
- TOP : signale que $TCNT_2$ a atteint sa valeur haute ;
- $BOTTOM$: signale que $TCNT_2$ a atteint 0 ;

Contrôle de la fréquence

- $CS22 : 0$ sélectionne la clock ;
- si $CS22 : 0 = 0$ le *timer* est au repos ;
- $WGM2_1 WGM2_0 \in TCCR_2$ permettent de déterminer la séquence de comptage.

3.3.2 Génération de forme : pulse, *PWM*,...

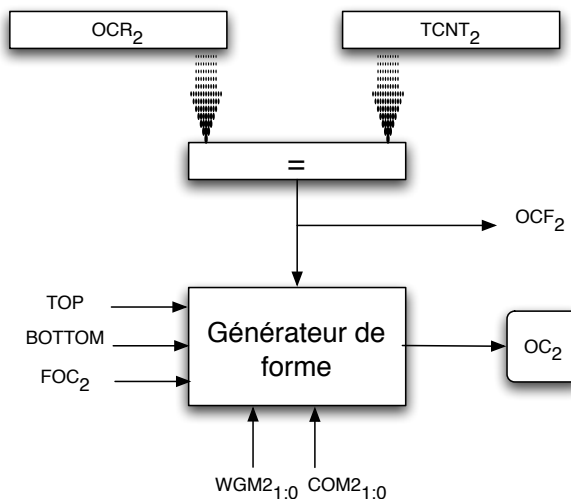


Figure 3.8: Génération de formes

OCR_2 est un registre à buffer double en mode *PWM*. On n'a pas la double bufferisation en mode normal et *CTC*. Quand on est en double buffer le *CPU* accède au buffer de OCR_2 , quand ce mode est désactivé le *CPU* accède au registre lui-même.

Mise à jour du buffer et de OCR_2 :

La mise à jour d' OCR_2 est retardée au moment où $TCNT_2$ atteint soit *TOP* ou *BOTTOM* évitant ainsi de créer des *PWM* non symétriques rendant la sortie "Glitch Free".

En mode non *PWM*, on peut forcer la comparaison par FOC_2 si il y a égalité OC_2 prend une valeur. C'est $COM2_{1:0}$ qui définissent ce qui se passe alors sur OC_2 (1, 0 ou commutation). Toute écriture du *CPU* bloque toute comparaison avec $TCNT_2$ ce qui permet de positionner une même valeur d'initialisation sur OCR_2 et $TCNT_2$. Par contre, si on met une valeur dans OCR_2 égale à $TCNT_2$ le test d'égalité va être perdu.

Unité de comparaison

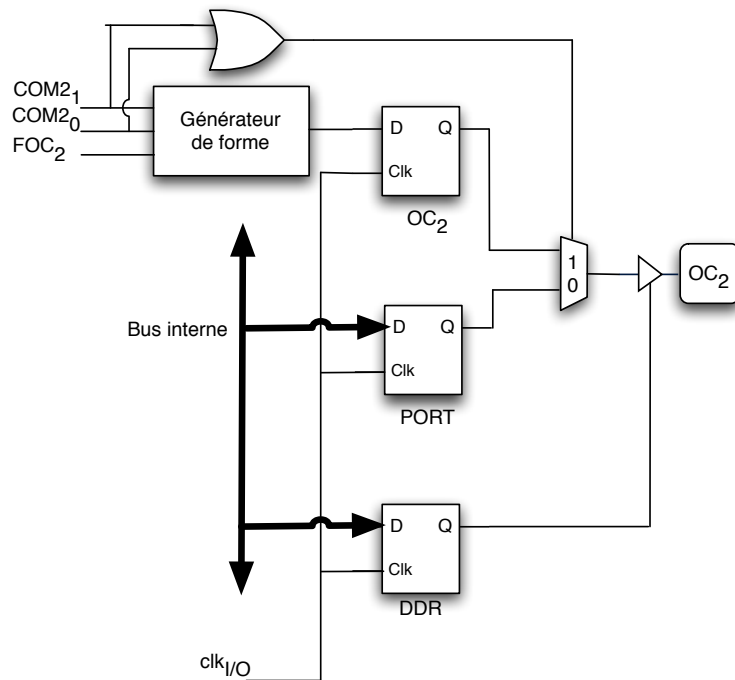


Figure 3.9: Unité de comparaison

- $COM2_{1:0}$ contrôle 2 fonctions :
 - La sortie de comparaison du générateur de formes
 - Ce que reçoit OC_2 dans le cas où l'un des deux bits est à un.
- Cette patte reçoit soit :
 - La sortie du générateur de formes
 - soit un bit d'un *port*, suivant la configuration du Port fixé par *DDR*

3.3.3 Les modes du timer 2

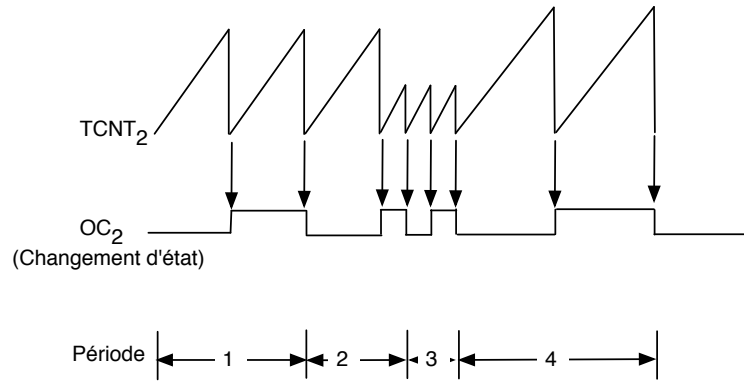
Les modes sont programmés par les bits $WGM2_{1:0}$ et $COM2_{1:0}$. Les bits $COM2_{1:0}$ contrôlent si la *PWM* est inversée ou non. Pour les modes non *PWM* (tel que le mode comparaison), les bits $COM2_{1:0}$ contrôlent aussi, si OCF_2 doit être **raz**, **mis à un**, ou commuté (toggle). Enfin, $WGM2_{1:0}$ contrôlent les modes.

Mode normal $WGM2_{1:0} = 0$

Dans ce mode $TCNT_2$ s'incrémente jusqu'à 0xFF puis recommence depuis 0. TOV_2 passe à un quand $TCNT_2$ retombe à zéro. TOV_2 sert alors de 9^{ième} bits, il sert aussi à déclencher une *interruption*. L'*interruption* est **raz** automatiquement par l'exécution du sous-programme d'*interruption* et en mode non interruptif il faut faire une **raz** par une **mise à un**.

Mode Clear **timer** on Compare *CTC* : $WGM2_{1:0} = 2$

Dans ce mode OCR_2 définit la résolution. $TCNT_2$ est **raz** quand $TCNT_2$ devient égal à OCR_2 qui définit la valeur *TOP* (cf figure 3.3.3).



Interruption: Une *interruption* peut être générée à chaque *TOP*. Dans le sous-programme d'*interruption* on peut changer le *TOP*. Attention : ne pas changer *TOP* avec une valeur à $TCNT_2$

Fréquence

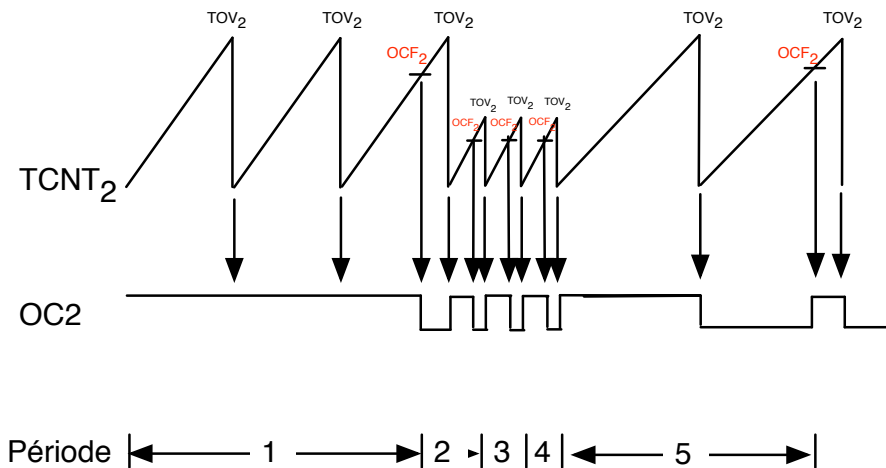
$$f_{OC_2} = \frac{f_{clk_{I/O}}}{2.N.(1 + OCR_2)}$$

Mode *PWM* a fréquence rapide avec $WGM2_{1:0} = 3$

C'est un mode à simple front. TOV_2 est **mis à un** quand $TCNT_2$ atteint *MAX* l'*interruption* peut mettre à jour OCR_2 . $TCNT_2$ compte de *BOTTOM* à *MAX* et repart de *BOTTOM* ; En mode non inversé, $OC2$ est **raz** sur égalité de $TCNT_2$ et OCR_2 , et **mis à un** sur *BOTTOM*

Fréquence

$$f_{OC_2 PWM} = \frac{f_{clk_{I/O}}}{N.(1 + OCR_2)}$$



Chronogramme de la *PWM* rapide

Le mode *PWM* à phase correcte

Dans ce mode $WGM2_{1:0} = 1$, on a une *PWM* à phase correcte. Elle est basée sur un mode double front. TOV_2 est mis à un quand $TCNT_2$ atteint *BOTTOM* et l'interruption peut mettre à jour OCR_2 . $TCNT_2$ compte de *BOTTOM* à *MAX* et repart de *MAX* jusqu'à *BOTTOM* ;

inversé/non inversé En mode non inversé, en comptant $OC2$ est **raz** sur égalité de $TCNT_2$ et OCR_2 , tandis que en décomptant, $OC2$ est mis à un sur sur égalité de $TCNT_2$ et OCR_2 . On a un mode ici symétrique : "phase correcte".

Fréquence

$$f_{OC2PWM} = \frac{f_{clkI/O}}{2.N.(1 + OCR_2)}$$

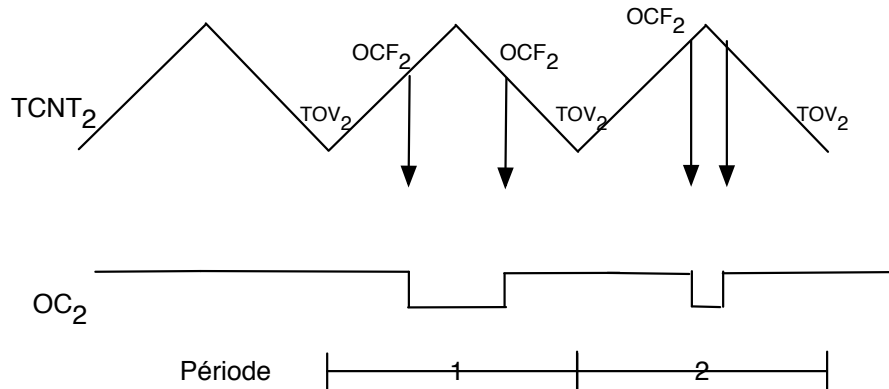


Figure 3.12: le timer 2 : Mode *PWM* phase correcte

Chronogramme de la *PWM* à phase correcte

3.3.4 Les registres utiles au timer 2

Le registre $TCCR_2$

Le registre $TCCR_2$

FOC_2	WGM_2	$COM2_1$	$COM2_0$	$CS2_2$	$CS2_1$	$CS2_0$	$TCCR_2$
---------	---------	----------	----------	---------	---------	---------	----------

Bit 7 : FOC_2 : Force Output Compare FOC_2 est actif quand on est pas en mode *PWM*. Cependant ce bit doit être **raz** quand $TCCR_2$ est mis à jour en mode *PWM*. Quand $FOC_2 \leftarrow 1$ une comparaison immédiate est forcée selon la valeur des bits de $COM2_{1:0}$. $OC2$ sera mis à jour. FOC_2 est un bit provoquant un échantillonnage.

Bit 6,3 : $WGM2_{1:0}$: Waveform Generation Mode Ces Bits contrôlent la séquence de comptage, le maximum et le type de forme :

v	$WGM2_1$	$WGM2_0$	Mode	TOP	maj OCR_2	$TOV_2 \leftarrow 1$?
0	0	0	Normal	0xFF	Immédiatement	MAX
1	0	1	PWM phase correcte	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR_2	Immédiatement	MAX
3	1	1	PWM rapide	0xFF	BOTTOM	MAX

Bit $COM2_{1:0}$: Compare Match Output Mode En mode de comparaison

$COM2_1$	$COM2_0$	description
0	0	OC_2 est deconnectée du P
0	1	OC_2 est commutée sur égalité
1	0	OC_2 est raz sur égalité
1	1	OC_2 est mise à un sur égalité

Bit $COM2_{1:0}$: Compare Match Output Mode En mode de comparaison et *PWM* rapide

$COM2_1$	$COM2_0$	description
0	0	OC_2 est deconnectée du P
0	1	réservé
1	0	OC_2 est raz sur égalité, mise à un sur <i>BOTTOM</i>
1	1	OC_2 est mise à un sur égalité, raz sur <i>BOTTOM</i>

Bit 5:4 : $COM2_{1:0}$: Compare Match Output Mode En mode de comparaison et *PWM* à phase correcte

$COM2_1$	$COM2_0$	description
0	0	OC_2 est deconnectée du P
0	1	réservé
1	0	OC_2 est raz sur égalité en comptant, mise à un sur égalité en décomptant
1	1	OC_2 est mise à un sur égalité en comptant, raz sur égalité en décomptant

Bit 2:0 : $CS2_{2:0}$: Clock Select

$CS2_2$	$CS2_1$	$CS2_0$	description
0	0	0	pas d'horloge, le timer 2 est stoppé
0	0	1	$clk_{I/O}$
0	1	0	$clk_{I/O}/8$
0	1	1	$clk_{I/O}/32$
1	0	0	$clk_{I/O}/64$
1	0	1	$clk_{I/O}/128$
1	1	0	$clk_{I/O}/256$
1	1	1	$clk_{I/O}/1024$

ASSR

Le registre $ASSR$

-	-	-	-	AS_2	$TCN2U_B$	$OCR2U_B$	$TCR2U_B$
---	---	---	---	--------	-----------	-----------	-----------

Bit AS_2 : Asynchronous Timer/Counter2 Quand $AS_2=0$, le **timer** 2 est cadencé par $clk_{I/O}$ Quand $AS_2=1$, le **timer** 2 est cadencé par le quartz connecté à $TOSC_1$

Bit $TCN2U_B$: Timer/Counter2 Update Busy Quand le **timer** 2 fonctionne en mode asynchrone, et que $TCNT_2$ est **mis à jour** alors $TCN2U_B \leftarrow 1$. Un niveau bas sur ce bit indique que $TCNT_2$ est prêt à être mis à jour avec une nouvelle valeur.

Bit 1 : $OCR2U_B$: Output Compare Register2 Update Busy Quand le **timer** 2 fonctionne en mode asynchrone, et que OCR_2 est **mis à jour** alors $OCR2U_B \leftarrow 1$. Un niveau bas sur ce bit indique que OCR_2 est prêt à être mis à jour avec une nouvelle valeur.

Bit 0 : $TCR2UB$: Timer/Counter Control Register2 Update Busy Quand le **timer** 2 fonctionne en mode asynchrone, et que $TCCR_2$ est mis à jour alors $TCCR2UB \leftarrow 1$. Un niveau bas sur ce bit indique que $TCCR_2$ est prêt à être mis à jour avec une nouvelle valeur.

Definition 3.3.1 Accès aux registres "busy" Si une écriture est réalisée sur un des trois registres : $TCNT_2$, OCR_2 ou $TCCR_2$ alors que les flags correspondants sont busy alors on peut corrompre les calculs ou provoquer une interruption non désirée.

Précautions sur le passage en mode asynchrone Lors des changements synchrones-asynchrones les valeurs des registres $TCNT_2$, OCR_2 ou $TCCR_2$ peuvent être erronées. La procédure de change vers le mode asynchrone est la suivante :

- Désactiver les interruptions relatives à le **timer** 2 par **raz** de $OCIE_2$ et $OCIE_2$
- Sélectionner la source avec AS_2
- Ecrire les nouvelles valeurs dans $TCNT_2$, OCR_2 ou $TCCR_2$
- Passer en mode Asynchrone : Attendre que les bits $TCN2UB$, $OCR2UB$ ou $TCR2UB$ passe de "busy" à "libre" (valeur 0).
- **raz** les flags d'interruption relatifs à le **timer** 2
- Ré-autoriser les ITs
- L'oscillateur est optimisé pour utiliser un quartz de 32,768 khz.
- Attention la fréquence du quartz principal doit être au moins 4 fois supérieure à celle du quartz que l'on utiliserait en mode asynchrone connecté à $TOSC_1$.
- Lors d'une écriture dans $TCNT_2$, OCR_2 ou $TCCR_2$, la nouvelle valeur passe par $TEMP$ et n'est accessible que après deux fronts positifs sur $TOSC_1$. Il faudra à l'utilisateur scruter les bits "busy" déjà évoqués avant d'écrire de nouvelles valeurs.

3.3.5 Les interruptions du **timer** 2

3.3.6 $TIMSK$

- Bit 7 : $OCIE_2$: Timer/Counter2 Output Compare Match Interrupt Enable
Quand $OCIE_2 \leftarrow 1$ et que $I = 1 (\in SREG)$ alors l'interruption de comparaison du **timer** 2 est active. Si une égalité entre $TCNT_2$ et OCR_2 se produit alors $OCF_2 \leftarrow 1 (\in TIFR)$
- Bit 6 : $OCIE_2$: Timer/Counter2 Overflow Interrupt Enable
Quand $TOIE_2 \leftarrow 1$ et que $I = 1 (\in SREG)$ l'interruption de débordement est active et $TOV_2 \leftarrow 1 (\in TIFR)$

3.3.7 $TIFR$

- Bit 7 : OCF_2 : Output Compare Flag 2
 $OCF_2 \leftarrow 1$ quand une égalité de comparaison se produit entre $TCNT_2$ et OCR_2 .
 - $OCF_2 \leftarrow 1$ par le hard dans le sous-programme d'interruption.
 - Sinon, $OCF_2 \leftarrow 0$ en écrivant un **un**.
 - C'est quand on a $I = 1 (\in SREG)$ et $TOCIE_2=1$ et $OCF_2=1$ que le sous-programme d'interruption est exécuté.
- Bit 6 : TOV_2 : Timer/Counter2 Overflow Flag
 - $TOV_2 \leftarrow 1$ quand un overflow se produit.
 - $TOV_2 \leftarrow 0$ par le hard dans le sous-programme d'IT.
 - Sinon, $TOV_2 \leftarrow 0$ en écrivant un **un**.
- C'est quand on a $I = 1 (\in SREG)$ et $TOIE_2 = 1$ et $TOV_2=1$ que le sous-programme d'IT est exécuté.

- En mode *PWM* ce bit reçoit un quand il y a un changement de direction à 0x00

Exemple d'utilisation du **timer 2** avec la mise en place d'une interruption de débordement :

```
#define LedToggle PortB^=1
volatile byte Compteur;

ISR (TIMER2_OVF_vect)
{ byte i;
  TCNT2 = 0;
  if (Compteur++ == 50) {
    Compteur=0;
    LedToggle; // mesurer la periode
  }
}

void configTimer2(){
  TCCR2A = 0; // Mode normal
  TCCR2B = (1<<CS22) + (1<<CS21) ; // clkio/256 est incremente toutes les 16uS
  TIMSK2 = 1<<TOIE2; // TOIE2
}

int main() {
  DDRB=0xFF;
  configTimer2();
  sei(); // autorise les interruptions
  while(1);
}
```

Chapter 4

La programmation

4.1 Langage C

4.1.1 Structure d'un programme

Un programme C est composé de trois parties. Les entêtes, les fonctions nécessaires et le programme principal (main).

```
//-----ENTETE-----
#include <avr/io.h>
#define F_CPU 12E6 // reglage fclkIO sur fquartz
#include <util/delay.h> //utilisation de _delay_ms
#include <avr/interrupt.h>
#define Max_del 262
//-----FONCTIONS-----
void sleep(int dzs){
    int i=0;
    for(i=0;i<10*dzs;i++) _delay_ms(10);
}
//-----MAIN-----
int main(void)
{
    DDRC = 0x00;    // \pc\ en entree
    DDRB = 0x01;    // PB0 en sortie
    DDRD = 0b1111011; // Port D en sortie, PORTD2 en entree
    do {
        sleep(10);
    } while(1);
    return(1);
}
```

4.1.2 Entête

Sous l'IDE Arduino , il n'y a pas besoin d'inclure de bibliothèques, et par contre, on peut y définir des constantes non modifiables ou des fichiers à inclure.

```
#define Max_del 262
#include "mesfonctions.c" // Inclusions de fichier c
#include "interface.h" // Inclusion de bibliotheque utilisateur
```

4.1.3 Main

Un main commence par des initialisations et se termine généralement par une boucle infinie :

```

void fonction1(){...}
int fonction1(){int a; ... ; return a}
int main(void)
{
    DDRC = 0x00;    // PORTC en entree
    DDRB = 0x01;    // PB0 en sortie
    DDRD = 0b1111011;    // PORTD en sortie, PD2 en entree
    int j;
    do {
        sleep(10);
        fonction1();
        j=fonction2();
    } while(1);
    return(1);
}

```

4.1.4 Fonctions

Elles apparaissent de préférence (sinon on mettra les prototypes) avant le main :

```

void sleep(int dzs){
    int i=0;
    for(i=0;i<10*dzs;i++) _delay_ms(10);
}

void entier(int nbtours, int sens){ // passage de variables
int i,j;
int Tentier[4]={1,2,4,8};

    for(j=0;j<nbtours*12;j++) {
        for(i=3;i>=0;i--){
            delai2();
            PB=Tentier[i];
        }
    }
}

```

4.1.5 Déclaration de variable globales

Pour les variables globales, on utilisera l'attribut volatile qui est une directive qui indique au compilateur de déclarer une variable dans la RAM et non dans la zone de registre. Cette zone peut produire, en effet, des modifications de variables inattendus. On pourra aussi utiliser cet attribut pour les variables des sous-programmes d'IT.

Cet attribut dégrade temps d'accès à la variable.

```

volatile int F;

void delai(){
    int i,deltaT=F;
    for(i=0;i<=deltaT;i++) _delay_ms(1);
}

int main(void){
    F=10;
    DDRB=0x1F;
}

```

```

DDRC=0x00;

do {
    PB^=1;
    delai();
}while(1);
return(0);
}

```

4.1.6 Fonction de manipulation de bits

- Mise à un par masque avec opérateur “ou” |:
- Mise à zéro par masque avec opérateur “et” &:
- Inversion avec opérateur “xor” ^:

```

// Mise a zero des 4 bits de poids faible
PORTB=PORTB & 0xF0; // ou bien
PORTB &= 0xF0;

// Mise a un des 4 bits de poids faible d'un port
PORTB = PORTB | 0x0F // ou bien
PORTB |= 0xF0;

// Le Pipe : On utilise un pipe quant on veut modifier un registre
// Si l'on veut simplement initialiser un registre on utilise l'affectation '=',
// Mise a un du bit 4 de PORTD et seulement lui !
PORTD |= 1<<PORTD4;

//Mise a zero du bit PORTD4 et seulement lui !
PORTD &= ~(1<<PORTD4);

// Commutation du bit PORTD4 et seulement lui !
PORTD ^= (1<<PORTD4);

```

4.1.7 Mise en oeuvre d'une interruption

On utilise la directive ISR qui associe un évènement à du code :

```

ISR(TIMERO_OVF_vect){ // routine d'interruption
    PORTD = PORTD^0x10; // Commutation du but num 4
    TCNT0=0;
}
int main(void)
{
    DDRD = 0b1111011; // Port D en sortie, PORTD2 en entree
    TCNT0= 0; // valeur initiale du compteur
    TCCR0 = 5; // facteur de predivision de clk/I0
    sei(); // Toutes les ITs sont possibles
    TIMSK |= (1<<TOIE0); // Validation de l'It de debordement
    do {} while(1);
    return(1);
}

```

4.2 Interruptions

Nom de l'interruption	Description
ADC_vect	ADC Conversion Complete
ANALOG_COMP_0_vect	Analog Comparator 0
ANALOG_COMP_1_vect	Analog Comparator 1
ANALOG_COMP_2_vect	Analog Comparator 2
ANALOG_COMP_vect	Analog Comparator
ANA_COMP_vect	Analog Comparator
EE_RDY_vect	EEPROM Ready
EE_READY_vect	EEPROM Ready
EXT_INT0_vect	External Interrupt Request 0
INT0_vect	External Interrupt 0
INT1_vect	External Interrupt Request 1
IO_PINS_vect	External Interrupt Request 0
LCD_vect	LCD Start of Frame
LOWLEVEL_IO_PINS_vect	Low-level Input on Port B
OVRLT_vect	timer Overrun
CAN_vect	
PCINT0_vect	Pin Change Interrupt Request 0
PCINT1_vect	Pin Change Interrupt Request 1
PSC0_CAPT_vect	PSC0 Capture Event
PSC0_EC_vect	PSC0 End Cycle
PSC1_CAPT_vect	PSC1 Capture Event
PSC1_EC_vect	PSC1 End Cycle
PSC2_CAPT_vect	PSC2 Capture Event
PSC2_EC_vect	PSC2 End Cycle
SPI_STC_vect	Serial Transfer Complete
SPM_RDY_vect	Store Program Memory Ready
SPM_READY_vect	Store Program Memory Read
TIM0_COMPA_vect	Timer/Counter Compare Match A
TIM0_COMPB_vect	Timer/Counter Compare Match B
TIM0_OVF_vect	Timer/Counter0 Overflow
TIMER0_CAPT_vect	ADC Conversion Complete
TIMER0_COMPA_vect	TimerCounter0 Compare Match A
TIMER0_COMPB_vect	timer Counter 0 Compare Match B
TIMER0_COMP_A_vect	Timer/Counter0 Compare Match A
TIMER0_COMP_vect	Timer/Counter0 Compare Match
TIMER0_OVF0_vect	Timer/Counter0 Overflow
TIMER0_OVF_vect	Timer/Counter0 Overflow
TIM1_CAPT_vect	Timer/Counter1 Capture Event
TIM1_COMPA_vect	Timer/Counter1 Compare Match A
TIM1_COMPB_vect	Timer/Counter1 Compare Match B
TIM1_OVF_vect	Timer/Counter1 Overflow
TIMER1_CAPT1_vect	Timer/Counter1 Capture Event
TIMER1_CAPT_vect	Timer/Counter Capture Event
TIMER1_CMPA_vect	Timer/Counter1 Compare Match 1A
TIMER1_CMPB_vect	Timer/Counter1 Compare Match 1B
TIMER1_COMP1_vect	Timer/Counter1 Compare Match
TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
TIMER1_COMPB_vect	Timer/Counter1 Compare MatchB
TIMER1_COMPC_vect	Timer/Counter1 Compare Match C
TIMER1_COMPD_vect	Timer/Counter1 Compare Match D
TIMER1_COMP_vect	Timer/Counter1 Compare Match
TIMER1_OVF1_vect	Timer/Counter1 Overflow
TIMER1_OVF_vect	Timer/Counter1 Overflow

Nom de l'interruption	Description
TWI_vect	2-wire Serial Interface
TXDONE_vect	Transmission Done, Bit timer Flag 2 Interrupt
TXEMPTY_vect	Transmit Buffer Empty, Bit Itmer Flag 0 Interrupt
UART0_RX_vect	UART0, Rx Complete
UART0_TX_vect	UART0, Tx Complete
UART0_UDRE_vect	UART0 Data Register Empty
UART1_RX_vect	UART1, Rx Complete
UART1_TX_vect	UART1, Tx Complete
UART1_UDRE_vect	UART1 Data Register Empty
UART_RX_vect	UART, Rx Complete
UART_TX_vect	UART, Tx Complete
UART_UDRE_vect	UART Data Register Empty
USART0_RXC_vect	USART0, Rx Complete
USART0_RX_vect	USART0, Rx Complete
USART0_TXC_vect	USART0, Tx Complete
USART0_TX_vect	USART0, Tx Complete
USART0_UDRE_vect	USART0 Data Register Empty
USART1_RXC_vect	USART1, Rx Complete
USART1_RX_vect	USART1, Rx Complete
USART1_TXC_vect	USART1, Tx Complete
USART1_TX_vect	USART1, Tx Complete
USART1_UDRE_vect	USART1, Data Register Empty
USART2_RX_vect	USART2, Rx Complete
USART2_TX_vect	USART2, Tx Complete
USART2_UDRE_vect	USART2 Data register Empty
USART3_RX_vect	USART3, Rx Complete
USART3_TX_vect	USART3, Tx Complete
USART3_UDRE_vect	USART3 Data register Empty
USART_RXC_vect	USART, Rx Complete
USART_RX_vect	USART, Rx Complete
USART_TXC_vect	USART, Tx Complete
USART_TX_vect	USART, Tx Complete
USART_UDRE_vect	USART Data Register Empty
USI_OVERFLOW_vect	USI Overflow
USI_OVF_vect	USI Overflow
USI_START_vect	USI Start Condition
USI_STRT_vect	USI Start
USI_STR_vect	USI START
WATCHDOG_vect	Watchdog Time-out
WDT_OVERFLOW_vect	Watchdog timer Overflow
WDT_vect	Watchdog Timeout Interrupt