Polytech'Nantes

Informatique Industrielle II : Microcontrôleur

David Delfieu

Département Génie Electrique 3^{ieme} année

Acronyms

$ACIC$ Analog Comparator Input Capture Enable. 29

ACO Analog Comparator Output. 29, 30

$COM1A1:0$ Compare Output Mode A 1:0. 32

$COM1x1:0$ Compare Output Mode x(A or B) 1:0. 30, 31, 34, 35

ICF_1 Input Compare Flag 1. 27, 29

$ICNC_1$ Input Capture Noise Canceller 1. 30

OCF_{1A} Output Compare Flag 1 A. 27, 28

OCF_{1B} Output Compare Flag 1 B. 27, 28

$TICIE_1$ Timer Input Capture Interrupt Enable 1. 29

TOV_1 Timer Overflow 1. 27, 29

$WGM1_{3:0}$ Wave Generation Modes 3:0. 2, 29–31

ACS ACD ACO ACI $ACIE$ $ACIC$ $ACIS_1$ $ACIS_0$. 29

$ADCH$ Analog to Digital Converter High. 18

$ADCL$ Analog to Digital Converter Low. 18

$ADMUX$ $REFS_1$ $REFS_0$ $ADLAR$ MUX_4 MUX_3 MUX_2 MUX_1 MUX_0. 19

$ASSR$ Asynchronous Status Register. 44

$DDRB$ Data Direction Register B. 32

$GICR$ General Interrupt Control Register. 11, 12

ICR_1 Input Capture Register. 28–30, 32, 33, 36, 37

$MCUCR$ Management Control Unit Control Register. 8, 11

$MCUCSR$ MCU Control and Status Register. 12

OCR_{1A} Output Compare Register 1 A. 28, 30–33, 36, 37

OCR_{1B} Output Compare Register 1 B. 28, 31

OCR_{1x} Output Compare Register 1 x=A ou B. 32–34, 36

OCR_2 Output Compare Register 2. 39–45

OC_{1A} Sortie *MLI* A du timer 1. 32

OC_{1X} Sorties *MLI* A ou B du timer 1. 34

$SFIOR$ $ADTS2$ $ADTS1$ $ADTS0$ $-$ $ACME$ PUD $PSR2$ $PSR10$. 15

SPI Serial Peripheral Interface. 5

SRAM Static Random Access Memory. 4

SREG I T H S V N Z C. 7

$TCCR_0$ $-$ $-$ $-$ $-$ $-$ CS_{02} CS_{01} CS_{00}. 25

$TCCR_{1A}$ $COM1A_1$ $COM1A_0$ $COM1B_1$ $COM1B_0$ $FOC1_1$ $FOC1_B$ $WGM1_1$ $WGM1_0$. 28, 29, 35

$TCCR_{1B}$ ICN_1 $ICES$ $-$ $WGM1_3$ $WGM1_2$ $CS1_2$ $CS1_1$ $CS1_0$. 28–30, 36

$TCCR_2$ Timer Counter Control Register 2. 40, 43

$TCNT_0$ Timer Counter Timer 0. 25, 26

$TCNT_1$ Timer Counter Timer 1. 28–34, 36, 37

$TCNT_2$ Timer Counter Timer 2. 39–45

$TIFR$ $OCF2$ TOV_2 ICF_1 $OCF1A$ $OCF1_B$ TOV_1 $-$ TOV_0. 24, 25, 28, 37, 39

$TIMSK$ $OCIE_2$ $TOIE_2$ $TICIE_1$ $OCIE_1A$ $OCIE_1B$ $TOIE_1$ $-$ $TOIE_0$. 3, 25, 26, 28, 36, 45

USART Universal Synchronous and Asynchronous Serial Receiver. 5

ACSR Comparateur analogique. 17

ADCH Poids fort de la conversion analogique. 2, 20

ADCL Poids faible de la conversion analogique. 2, 20

ADCSRA Contrôle du convertisseur analogique. 2, 19, 20

ADMUX Multiplexeur analogique. 2, 20

EEPROM Petite mémoire non volatile qui contient des données dynamiques. Elle a un temps d'accès très lent, autour de 5 millisecondes pour un $ATMEGA_8$. 4, 6

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. 4, 6

GICR Gestion des interruptions. 11

ICP1 Patte du timer 1 sur laquelle on peut réaliser une entrée de capture (PB_0). 29

MCUCR Gère notamment les activations des *interruptions* externes. 11

MLI Modulation de Largeur d'Impulsions. 5

OC1A Patte du timer 1 sur laquelle on peut observer une *MLI* (PB_1). 31

OC1B Patte du timer 1 sur laquelle on peut observer une *MLI* (PB_2). 31

OSCCAL Calibration et gestion du résonnateur. 8

PUD Inhibe les résistances de tirage. 15

SFIOR Résistances de tirage et le pré-diviseur des timer 0 et 1. 15, 17

SRAM Mémoire volatile qui contient les données dynamiques. Elle est contient notamment tous les registres du microcontrôleur. 4, 6

SREG Registre qui contrôle les bits d'état du microcontrôleur comme la Carry, le bit Zero, le bit Négatif, ... 7

TCCR0 Pré-diviseur du timer 0. 2, 25

TCNT0 Registre de comptage du timer 0. 2, 25

TIMSK Registre de masquage des interruptions des timers. 2, 25

Contents

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'$ATMEGA_8$, 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 un structure *C* et la manipulation de registres.

La famille des $ATMEGA_8$

Modèle	Flash	EEPROM	RAM	I0	*PWM*	Interfaces	CAN
$ATMEGA_8$	8K	512	1024	23	3	SPI-USART	10 bits
$ATMEGA_{16}$	16K	512	1024	32	4	SPI-USART	10 bits
$ATMEGA_{32}$	32K	1k	2k	32	4	SPI - USART	10 bits
$ATMEGA_{64}$	64K	2k	4k	53	8	SPI - USART(2)	10 bits
$ATMEGA_{128}$	128K	4k	4k	53	8	SPI - USART(2)	10 bits
$ATMEGA_{256}$	256K	4k	8k	53	16	SPI - USART(2)	10 bits

Dans cette famille, l'$ATMEGA_8$ et l'$ATMEGA_{16}$ 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 $ATMEGA_8$ par un $ATMEGA_{16}$.

1.1.1 Les différents blocs mémoire de l'$ATMEGA_8$

La mémoire de l'$ATMEGA_8$ 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. On placera ces variables dans cet espace, lorsqu'elles sont partagées par le prpgramma principal et un sous-programme d'intérruption. On peut utilise alors 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. Les accès à la variable sont plus lent, par contre, il n'y a pas de mise en cache de la variable et donc pas de problème de synchronisation (cf annexe 4.1.4)

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'$ATMEGA_8$ 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 a être modifiée 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'$ATMEGA_8$

L'$ATMEGA_8$ 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'$ATMEGA_8$.

Grâce à une liaison série de l'$ATMEGA_8$ on peut loader un programme à exécuter dans le microcontrôleur

1.2 Architecture

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

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

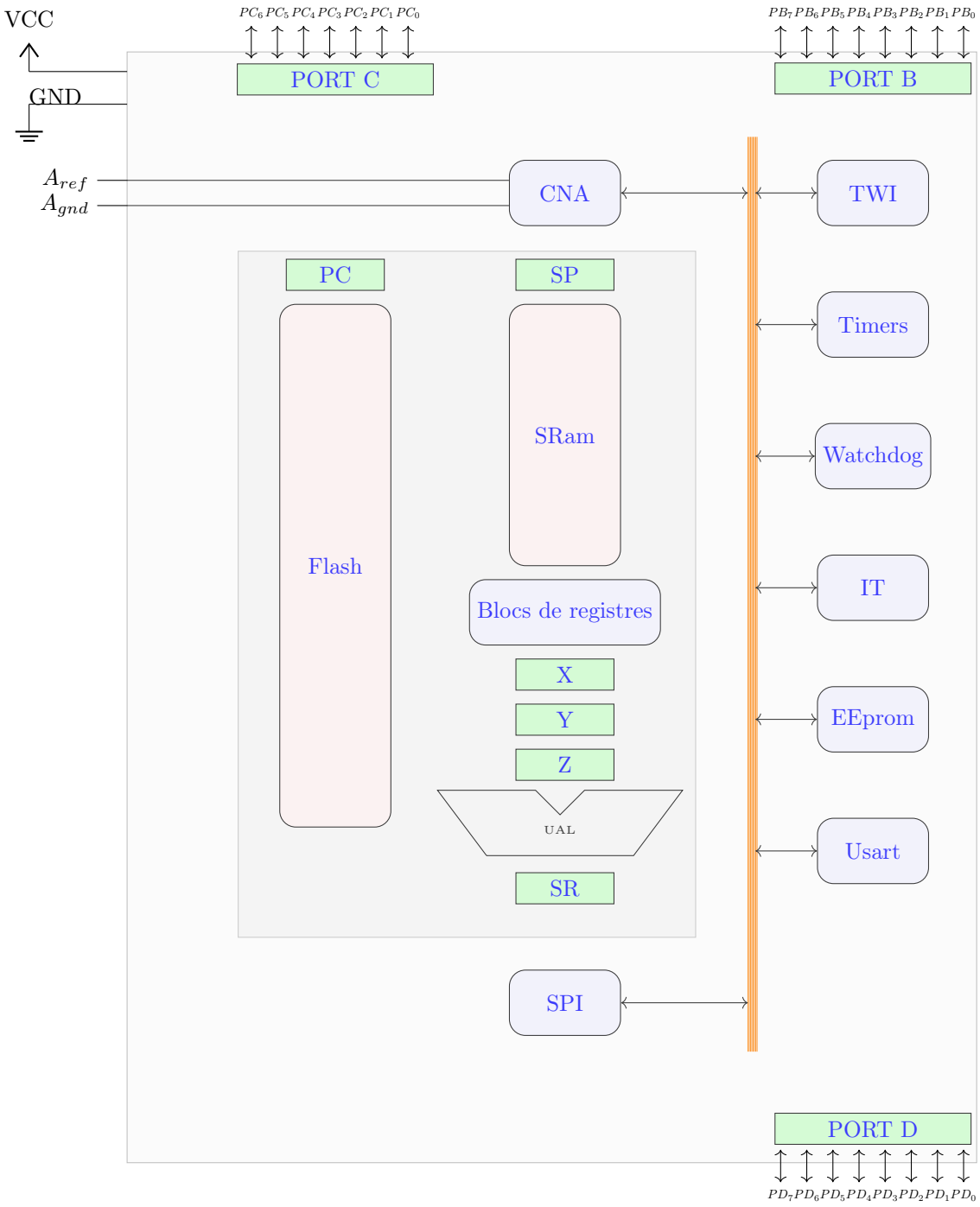


Figure 1.1: synoptique d'un Atmega

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

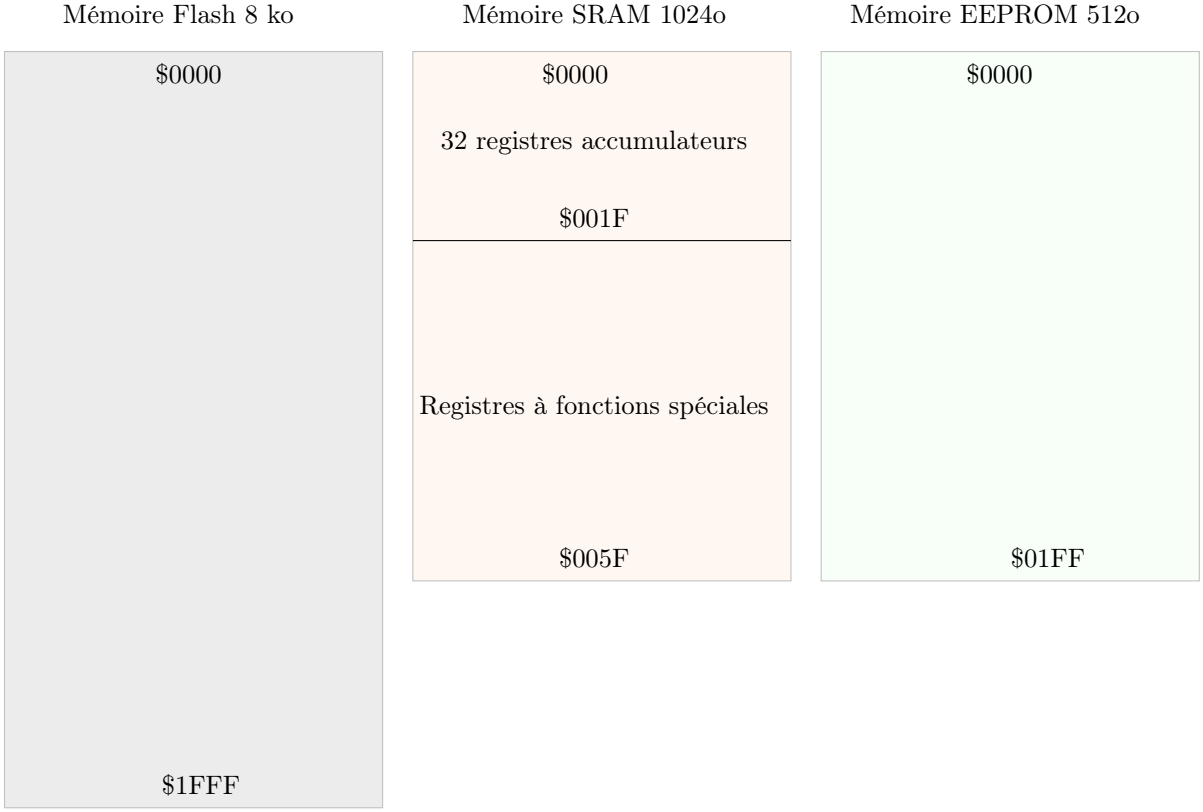


Figure 1.2: synoptique d'un Atmega

Un mapping signifie une projection d'un plan mémoire sur l'espace d'adressage. Dans l'$ATMEGA_8$, 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.

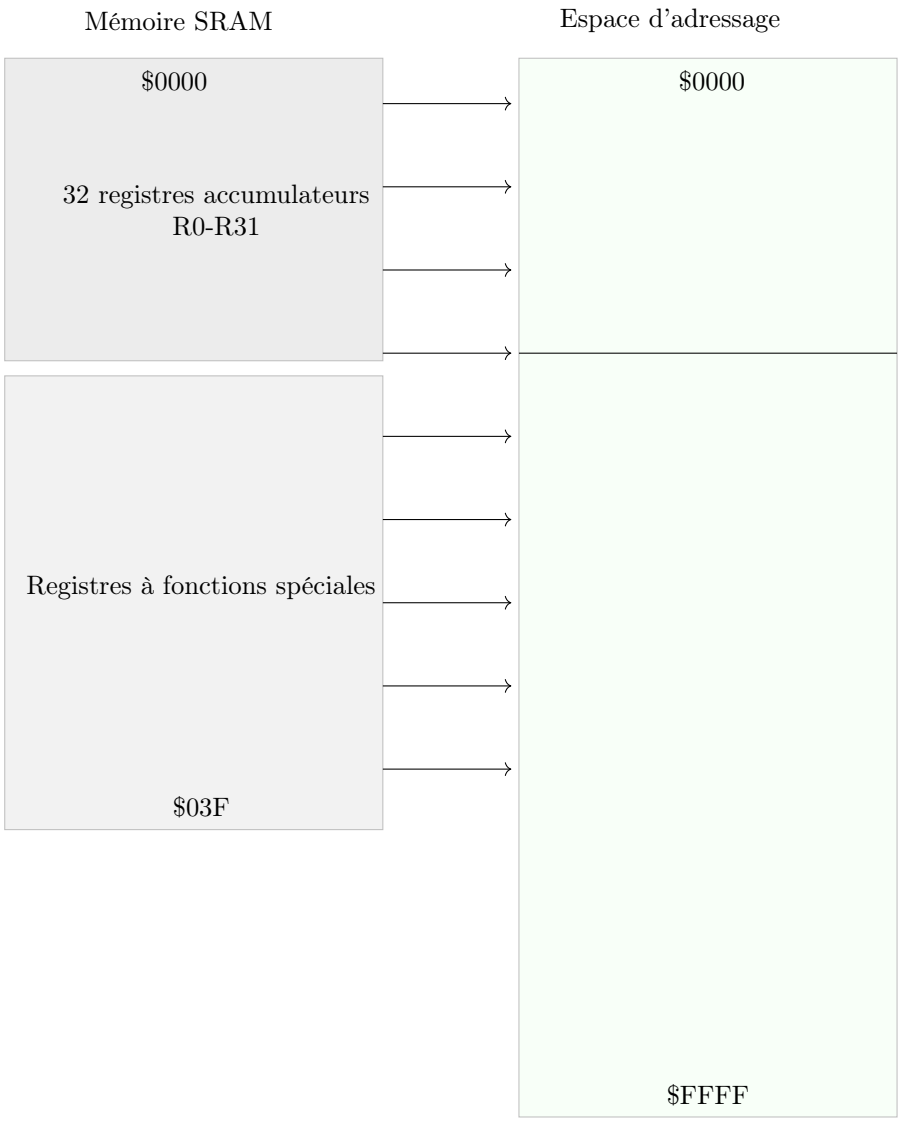


Figure 1.3: synoptique d'un Atmega

1.2.2 Registres Systèmes

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

Registres *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.

I	T	H	S	V	N	Z	C

- 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 : $0x1FF$

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	EEP	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

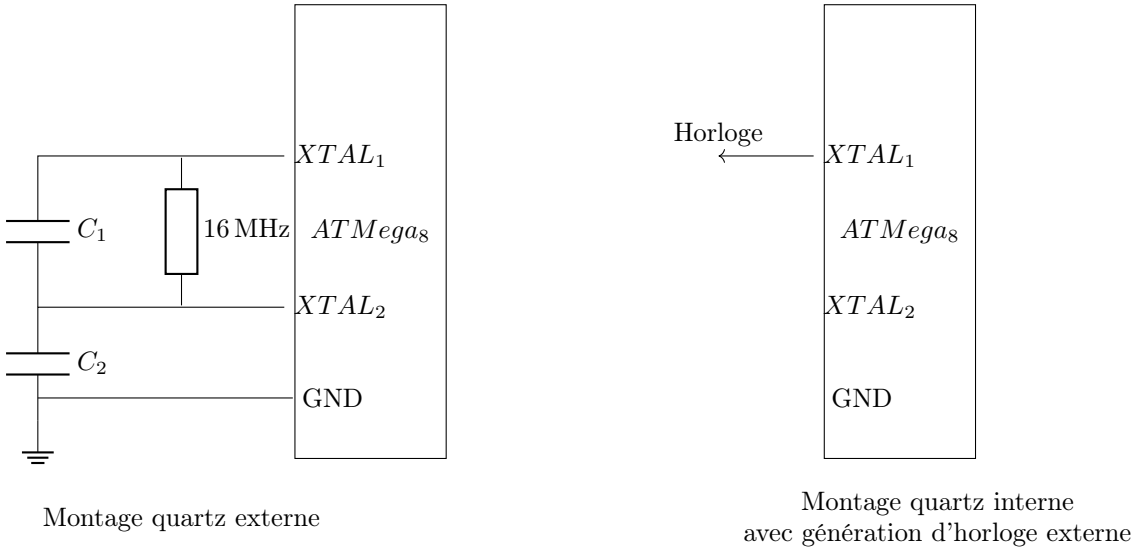
Calibration et le contrôle de l'horloge OSCCAL

CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0

Valeur de calibrage de l'oscillateur pour la programmation de la mémoire flash ou de l'eeprom.
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. **??**) présente un montage en quartz externe basse fréquence :

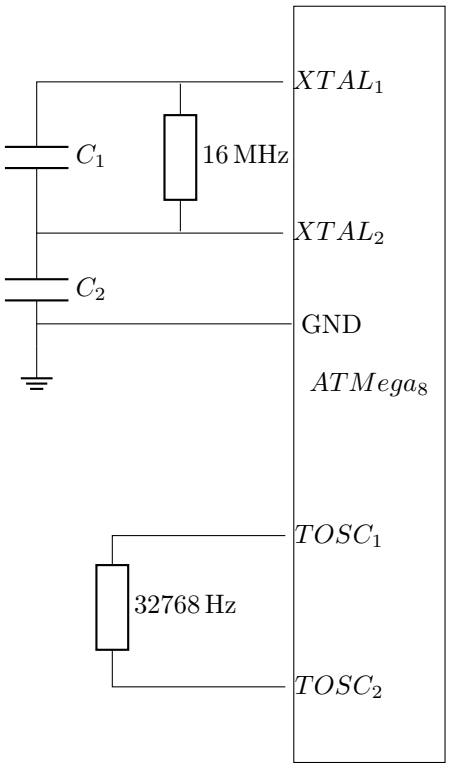


Montage quartz externe

Montage quartz interne
avec génération d'horloge externe

Figure 1.4: synoptique d'un Atmega

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



Génration d'une horloge temps réel

Figure 1.5: synoptique d'un Atmega

La figure suivante (fig. 1.2.2) présente un montage en résonateur externe :

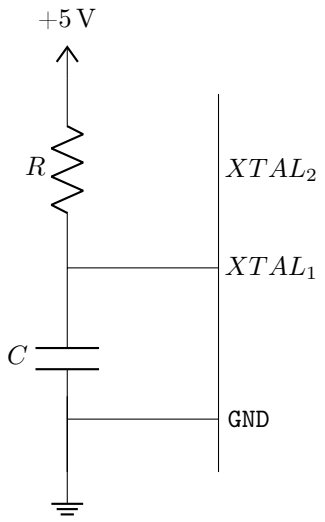


Figure 1.6: Résonateur externe

Tableau récapitulatif de tous les registres systèmes :

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

1.3 Les *interruptions* de l'$ATMEGA_8$

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.

1.3.1 Le concept d'*interruption*

Une *interruption* externe correspond à la prise en compte d'un évènement dont l'occurrence est possiblement aléatoire. Lors de la survenue de cet évènement l'exécution d'un sous-programme associé est lancée automatiquement. 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 intéressent d'utiliser des boucles d'attentes de cette alarme. Par exemple il est déconseillée de faire appel aux fonctions de la bibliothèque Serial (lentes) dans une *interruption*. Son traitement est donc réalisé en associant un sous-programme à un événement. On associe un évènement à un sous-programme par la primitive ISR.

1.3.2 Gestion des *interruptions*

Interruptions imbriquées : Dans la famille Arduino, une *interruption* n'est pas interruptible par défaut par une nouvelle interruption (sauf par un reset), en effet *I* de *SREG* est mis à zéro à l'entrée de l'*interruption*. Toute nouvelle interruption sera alors prise en compte lorsque l'*interruption* en cours sera terminée. Pour autoriser une nouvelle *interruption* dans l'*interruption* en cours il faut donc basculer *I* à un dans l'*interruption* en cours.

Deadlock : Il ne faut pas appeler de fonctions qui se mettent en attente d'une autre interruption. Comme l'interruption est in-interruptible par défaut, la fonction attendra indéfiniment et tout le système se bloquera. C'est ce que l'on appelle un *Deadlock*.

Plusieurs interruptions en même temps ? Les *interruptions* ont chacune une priorité. Par exemple, les interruptions externes sont plus prioritaires que les interruptions des Timers. L'Arduino exécutera les *interruptions* dans leur ordre de priorité. Dans la table ci-dessous, les priorités les plus petits numéros correspondent aux priorités les plus fortes :

Table 1.1: Tableaux des interruptions de l'$ATMEGA_8$

Priorité	Nom de l'*interruption*	Description
1	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	INT0	External Interrupt Request 0
3	INT1	External Interrupt Request 1
4	TIMER2 COMP	Timer/Counter2 Compare Match
5	TIMER2 OVF	Timer/Counter2 Overflow
6	TIMER1 CAPT	Timer/Counter1 Capture Event
7	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	TIMER1 OVF	Timer/Counter1 Overflow
10	TIMER0 OVF	Timer/Counter0 Overflow
11	SPI, STC	Serial Transfer Complete
12	USART, RXC	USART, Rx Complete
13	USART, UDRE	USART Data Register Empty
14	USART, TXC	USART, Tx Complete
15	ADC	ADC Conversion Complete
16	EE_RDY	EEPROM Ready
17	ANA_COMP	Analog Comparator
18	TWI	Two-wire Serial Interface
19	SPM_RDY	Store Program Memory Ready

1.3.3 Les *interruptions* externes INT_0 et INT_1

L'$ATMEGA_8$ offre deux *interruptions* externes sur deux pattes : PD_2 et PD_3 respectivement appelées INT_0 et INT_1. 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 évènements est données dans les tables de la section 4.2.2
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'événement INT_0 (front sur la patte PD_2) 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

  - INT_0 : à un, autorise une interruption externe sur la patte PD_2
  - INT_1 : à un, autorise une interruption externe sur la patte PD_3

- MCUCR :

SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

  Considérons une *interruption* externe INT_0. Les bits ISC_{01}, ISC_{00} correspondent alors à la patte PD_2 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".

1.3.4 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 $MCUCSR$

-	-	-	-	WDRF	BORF	EXTRF	PORF

- $WDRF$ 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 (cf figures 1.7,1.8,1.9). 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.

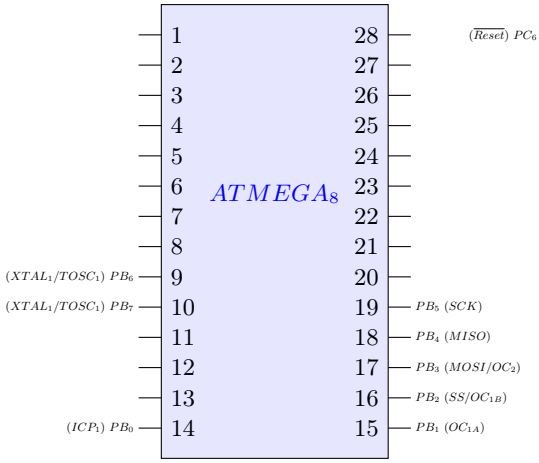


Figure 1.7: Brochage du port *B*

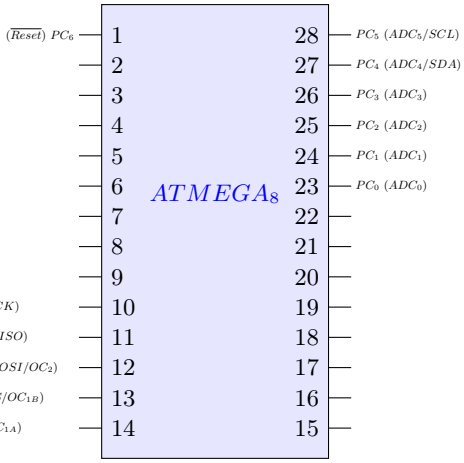


Figure 1.8: Brochage du port *C*

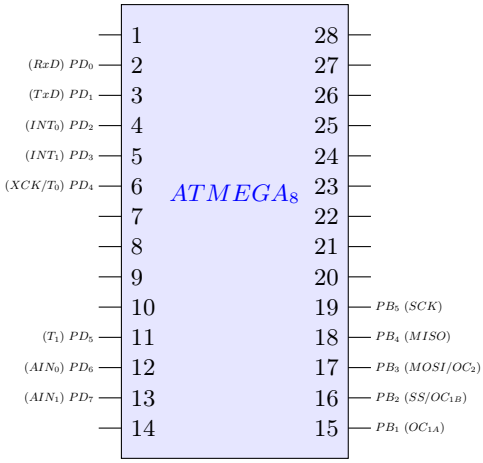


Figure 1.9: Brochage du port *D*

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

1.4.1 Usage des *PORTS*

On peut effectuer les lectures ou des écritures sur les ports. Dans le cas d'une écriture, chaque ligne du port peut fournir 20 mA de courant tandis que chaque *port* de 8 bits est limité à un courant total de 200 mA. Dans le cas d'une lecture sur le Port, le port devient drain de courant et peux accepter aussi 20 mA de courant. C'est le registre PIN_x qui permet la lecture. A chaque bit du registre PIN_x est associée une bascule D qui permet de 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. On peut, par ailleurs, positionner des résistances de tirage sur les lignes d'un port.

Déclaration des lignes en entrées ou en sortie

DDR_B, DDR_C, DDR_D : Permet de programmer le sens des lignes (1 : sortie, 0 entrée)

```
Port B : Seules les pattes 2 et 3 sont en entrées, les autres sont en sortie :
            DDRB = 0b1111 0011;
Port C : Seules la patte 0 est en entrée, les autres sont en sortie :
            DDRC = 0xFE;
Port D : Toutes les pattes sont en sorties :
            DDRD = 255;
```

Ecritures sur les ports

$PORT_B, PORT_C, PORT_D$: Permet d'écrire sur les ports

```
Port B : Les sorties 0,1 4,5,6,7 sont actives :
               PORTB = 0b1111 0011;
Port C : Les sorties 1,2,3,4,5,6,7 sont actives :
               PORTC = 0xFE;
Port D : Toutes les sorties sont actives :
               PORTD = 255;
```

Lecture des ports

PIN_B, PIN_C, PIN_D : Permet de lire les ports

```
Port B : Lecture du PORTB dans la variable i qui a été déclarée sur 16 bits
               int i = PINB;
Port C : Lecture du PORTC dans la variable i qui a été déclarée sur 8 bits
               char c  = PINC;
Port D : Lecture du PORTD dans la variable l qui a été déclarée sur 8 bits
avec masquage des 4 bits de poids fort
               char l = PIND & 0x0F;
```

1.4.2 Codes complets d'écriture et de lecture d'un port

```
int main(void) { //  Ecriture sur un port
  DDRD  = 0xFF;     // PORTD 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);
  }
}
```

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

1.4.3 Fonction de manipulation de bits dans un port (ou un registre)

Plusieurs opérateurs permettent de manipuler des ports de façon globale : | est un *OU* logique et il permet de rajouter des bits à un dans un registre. & correspond au *ET* logique et il permet de mettre des bits à zéro dans un registre. << est un opérateur de décalage à gauche de bits. Cet opérateur permet de positionner un bit au bon endroit avant une mise à "zéro" ou une mise à "un".

```
// 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;
```

1.4.4 Résistance de tirage : Pull Up Resistor

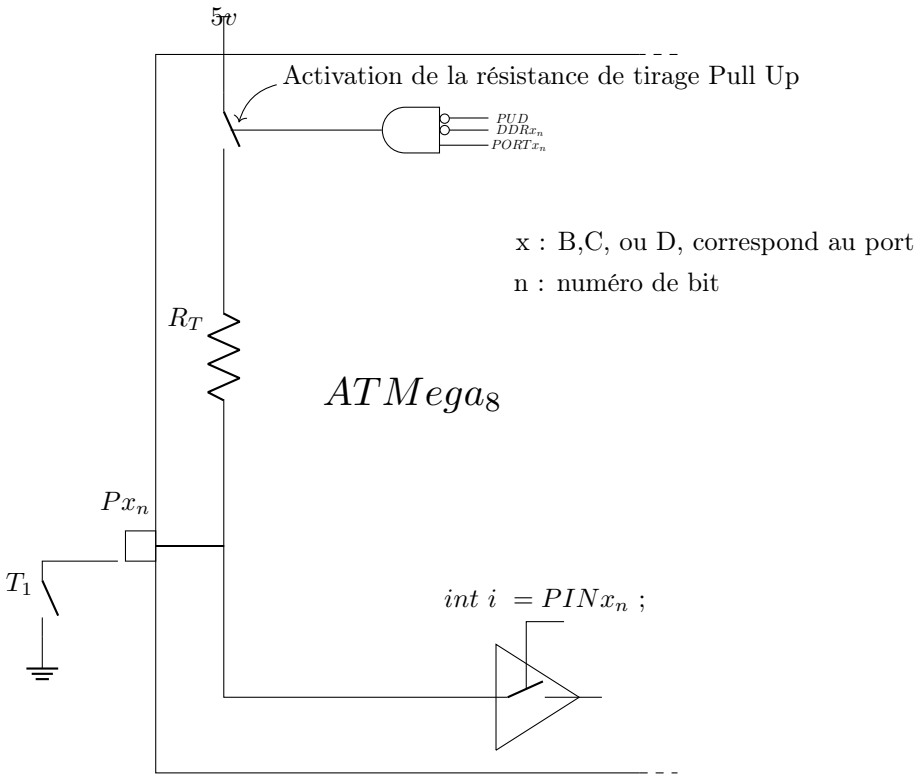


Figure 1.10: Résitance de tirage de type "pull up"

Pour poser des résistances de pull-up il est nécessaire d'avoir les trois conditions suivantes (porte AND du schéma) :

- Définir les lignes en entrée avec une action sur $DDRx_n$;
- Ecrire un "1" sur ces lignes, $PORT_x = 1$;
- PUD mis à zéro dans SFIOR : $SFIOR \;\&= \; (1 << PUD)$;

Si tel est le cas, même si le transistor T_1 est ouvert alors on aura $5v$, et donc un "un logique", lors de lecture de $PINx_n$.

Résistances de tirage : le registre *SFIOR*

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.

Chapter 2

Le traitement des valeurs Analogiques

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 $AC0 \leftarrow 1$, on peut alors déclencher une *interruption* (avec bit $ACIE = 1$) de comparaison.

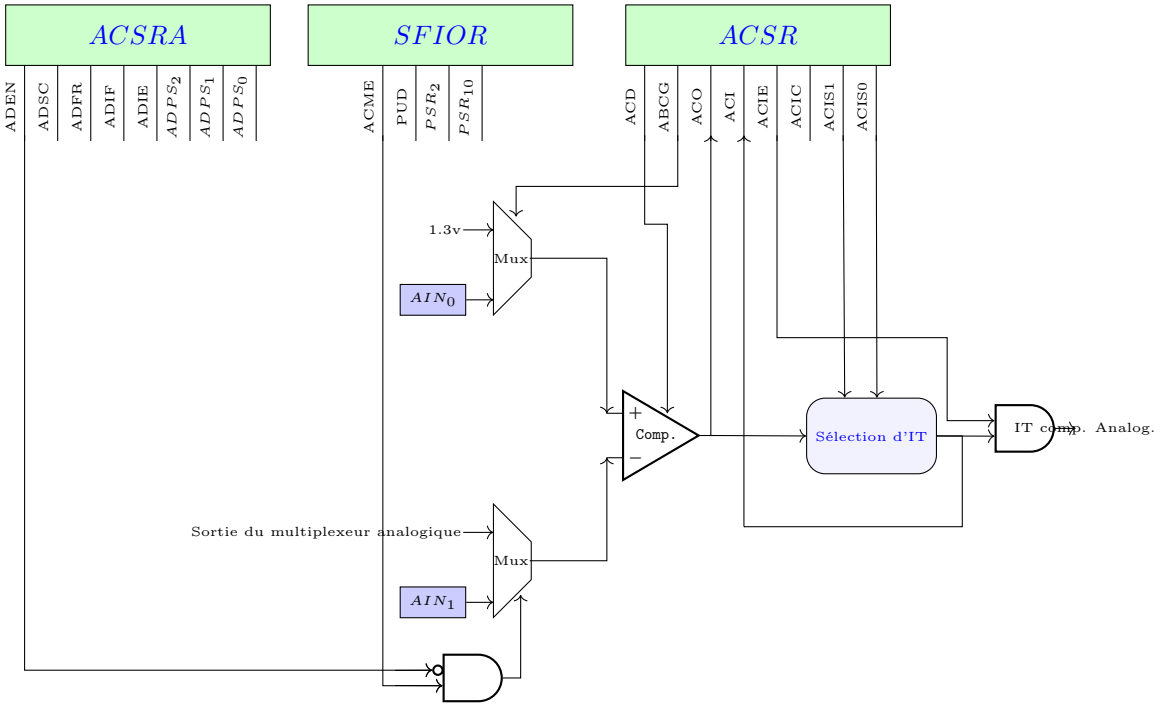


Figure 2.1: Comparateur

On a de plus, la possibilité de changer la référence "négative" : AIN_1 par l'une des broches $ADC_0, ADC_1, ..., ADC_5$ une des sorties du multiplexeur analogique. Les bits $ACIS1, ACIS0$ peremettent à l'utilisateur de sélectionner comme évènement de déclenchement, un front montant, un front descendant ou une inversion.

En outre, toute entrée $ADC_5, ..., ADC_0$ peut jouer le rôle de AIN_1. Pour réaliser cela il faut positionner à un le bit de multiplexage $ACME$ du registre $SFIOR$ quand l'ADC est OFF ($ADEN$ à zéro) les bits $MUX2, MUX1, MUX0$ du registre $ADMUX$ permettent de modifier la source AIN_1.

Les bits MUX qui remplacent AIN_1 selon le tableau suivant :

$ACME$	$ADEN$	$MUX2:0$	$AIN1$
0	x	xxx	$AIN1$
1	1	xxx	$AIN1$
1	0	000	ADC_0
1	0	001	ADC_1
1	0	010	ADC_2
1	0	011	ADC_3
1	0	100	ADC_4
1	0	101	ADC_5
1	0	110	ADC_6
1	0	111	ADC_7

2.1.2 Les registres

Registre ACSR: Analog Comparator Status Register ACSR

ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0

- 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 $VAIN_0 > VAIN_1$ alors $ACO = 1$.
- ACI : Analog Comparator Interrupt : Flag Bit de demande d'interruption (bit raz ds le sp d'IT)
- Condition IT : $ACIS1$ et $ASCI0$). 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 $ACSI0$ gère le comportement de la sortie AC_0 :

Activation de AC_0	ACIS1	ACIS0
$1 \rightarrow 0$ *ou* $0 \rightarrow 1$	0	0
non utilisé	0	1
front montant	0	0
front descendant	1	1

Registre SFIOR

ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10

$ACME$ 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 AIN_1est 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 $+/-2$ LSB avec une erreur à 0 V inférieure à 1 LSB. Le résultat de la conversion est positionné dans les registes Analog to Digital Converter High ($ADCH$) et Analog to Digital Converter Low ($ADCL$).

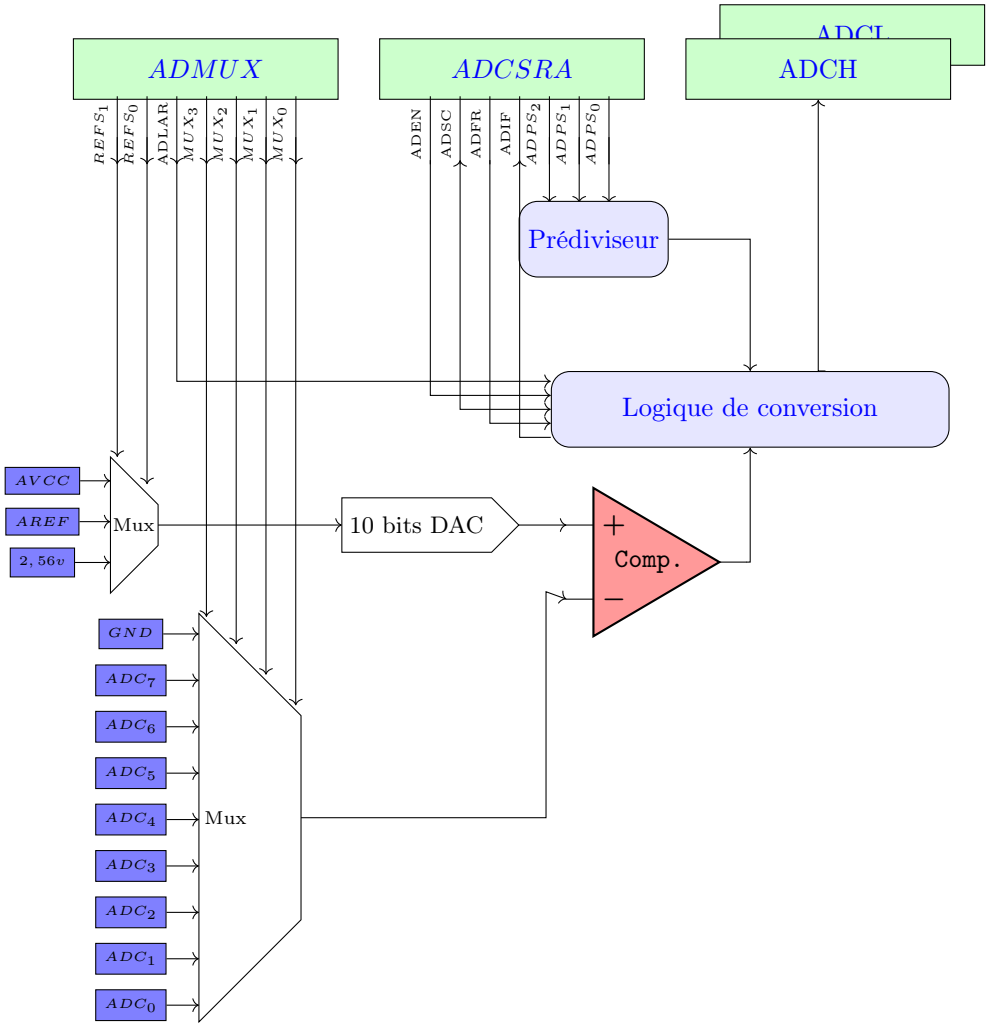


Figure 2.2: ADC

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

2.2.1 Fonctionnement et caractéristiques

On peut le programmer en générant une interruption de fin de conversion. De plus, il est possible du faire fonctionner en limitant le bruit en mode de sommeil. Le résultat d'une conversion numérique sur 10 bits est donné par la relation :

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

Ou la tension de référence est soit $AREF$ soit AV_{cc} soit 2,56v.

```
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, ...).
- Pour minimiser le bruit on peut ajuster la tension de référence

Ajustement de la tension On peut ajuster la tension de référence : 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 $REFS_n$).

Choix de l'entrée à convertir 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 d'abord $ADCSRA$:

ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0

- L'ADC est activé avec le bit $ADEN$ à 1.
- L'ADC est déclenché par $ADSC$ à 1.
- 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 avec un ajustement à gauche, on ne lit alors que le registre $ADCH$.

Cycle de conversion

Début d'une conversion Une conversion simple est lancée en positionnant $ADSC$ à 1.

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.

Durée d'une conversion Une conversion normale prend 13 cycles d'horloge.

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. $ADCL$ doit être lu en premier puis $ADCH$. Une fois que $ADCL$ est lu, l'accès aux registres est bloqué afin d'empêcher une nouvelle conversion. Quand l'$ADCL$ et $ADCH$ sont lus, alors l'ADC est à nouveau être opérationnel.

Le programme peut alors lancer $ADSC$ de nouveau et une nouvelle conversion sera amorcée sur le premier front montant de l'horloge.

2.2.2 Les différentes méthodes de programmation d'une conversion

Conversion échantillonnée

Pour réaliser une conversion échantillonnée, on doit 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 2.2.3).

Free Running mode : Mode de fonctionnement libre

Dans ce mode, on n'utilise pas d'interruption et l'ADC échantillonne en permanence, sans aucune action du programmeur autre que le lancement initial et la mise à jour les registres de données (*ADCH* et *ADCL*) est automatique. 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

L'attente active est du au fait que, après avoir lancée la conversion par *ADSC*, on realise une boucle qui attend que *ADSC* retombe à zéro. Un exemple de codage est donnée dans la section suivante (cf section 2.2.3).

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 pré-diviseur 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 en positionnant une valeur sur les bit $ADPS \in ADCSRA$ et le pré-diviseur commencera à compter dès que l'ADC est allumé en mettant le bit *ADEN* à 1 .

Les bits $ADPS2, ADPS1, ADPS0$ sélectionnent l'horloge :

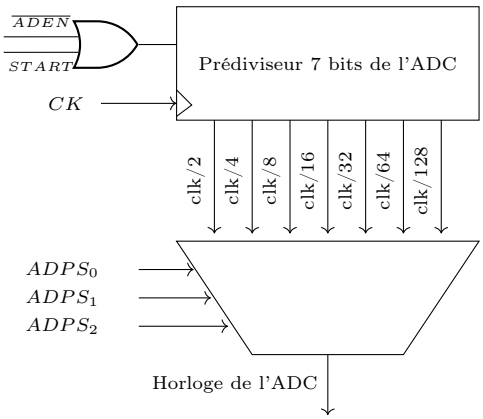


Figure 2.3: Pré-diviseur

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

– $REFS_1, REFS_0$

$REFS1$	$REFS0$	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 ou les conversions ont lieu en permanence sans avoir besoin dus re-lancer.

– $ADIF$: (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 facteur de pré-division de l'horloge interne du convertisseur en fonction du quartz (cf figure 2.3)

- $ADCH - ADCL$:

– 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 : on réalise des conversions en mode mode "attente active".

```
volatile int lu; // variable globale

void lecture_analogique_scrutative(){
  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 échantilloné 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=0x01;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 seule fois, puis on lit à la volée *ADCH* et *ADCL*.

```
int main(void){
    int H,L;
    DDRB=0x01; 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 beuggué : Sans timer relance de l'ADC dans l'*interruption* de conversion. Le programme principal n'a plus le temps de s'exécuter.

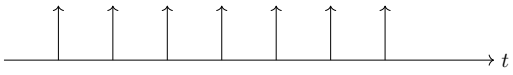
```
volatile int F;

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

int main(void){
  DDRB=0x01;DDRC=0x00;
  ADMUX=(1<<ADLAR);  // ajust Gauche, Lecture sur PC0
  ADCSRA= (1<<ADEN)+(1<<ADCS)+(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

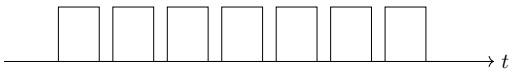
Les Timers

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*) ansi que le comptage d'événements. Un timer permet aussi de mesurer un temps entre deux événements.

Dans la figure suivante (figure 3) nous illustrons une base de temps et la génération de formes et le comptage d'évènement. Pour le comptage, la troisième partie de la figure 3 indique que l'on peut compter des fronts qui viennent de la patte T_1.

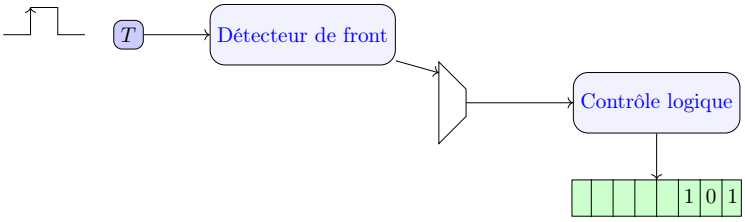


Base de temps



Forme périodique, *MLI*

Comptage d'événements



TCNT

L'$ATMEGA_8$ 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.

Drapeaux : Un drapeau n'est pas un bit au sens usuel. Il a pour rôle de signaler un évènement et est généralement associé à une interruption. Ce qui le distingue d'un bit est qu'un drapeau ne peux pas être mis à un ou remis à zéro de façon simple. Dans l'$ATMEGA_8$, le registre $TIFR$ contient 7 drapeaux qui correspondent à des interruptions liées aux timers 0,1 et 2.

$TIFR$

OCF_2	TOV_2	ICF_1	OCF_{1A}	OCF_{1B}	TOV_1	-	TOV_0

- OCF_2: "Output Compare Match Flag T_2"
- TOV_2: "TiOutput Compare Match Flag T_2"

- ICF_1: "Input Capture Flag T_1"
  - Ce drapeau reçoit un quand la patte ICP_1 reçoit un front.
  - ICF_1 est **raz** quant l' '*interruption* est exécuté.
- $OCF1_{A/B}$: "Output Compare A ou B Match Flag T_1"
  - Ce drapeau reçoit un lors de l'égalité de $OCR1_B$ et $TCNT_1$.
  - OCF_1B est **raz** quant l'*interruption* est exécuté.
- $TOV_{1/0}$: "Timer Overflow $T_{1/0}$"
  - $TOV_{1/0}$ est mis à un sur un dépassement de capacité
  - $TOV_{1/0}$ est **raz** quant l'*interruption* est exécuté.

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 par la patte externe timer 0 ou bien le prescaler (pré-diviseur d'horloge). Ce choix étant fait par les bits $CS0_{2:0} \in TCCR_0$

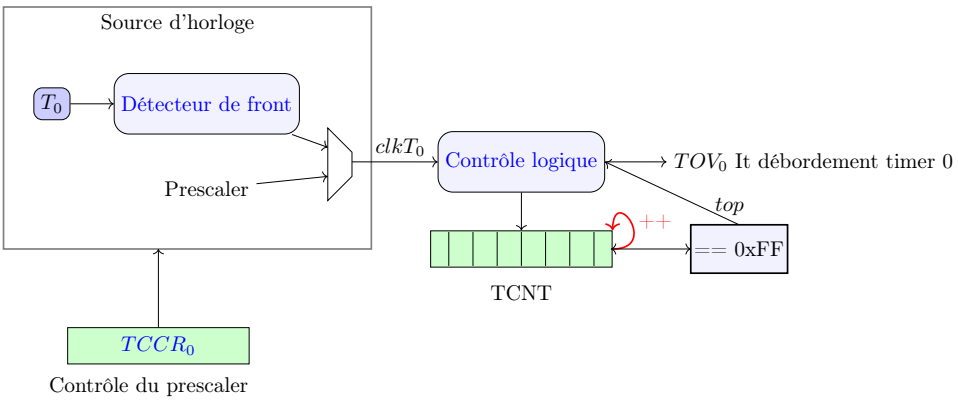


Figure 3.1: Aperçu de timer 0

Suivant la fréquence $clkT_0$, le bloc logique provoque l'incrément du registre TCNT0. Et 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 $9^{\text{ième}}$ bit. Si une *interruption* a été mise en place alors ce bit est remis à zéro (**raz**) lors de l'exécution du sous-programme d'interruption associé.

3.1.2 Les registres qui pilotent le timer 0 : TCCR0, TCNT0 TIMSK

- $TCCR_0$

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

$CS02$	$CS01$	$CS00$	*Description*
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 T_0 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 remis à zéro dés 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"**.

Exemple d'utilisation du timer zéro avec la mise en place d'une interruption de débordement. Dans ce exemple, la variable compteur sert à diviser la fréquence de commutation du port B par 50.

```
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);
}
```

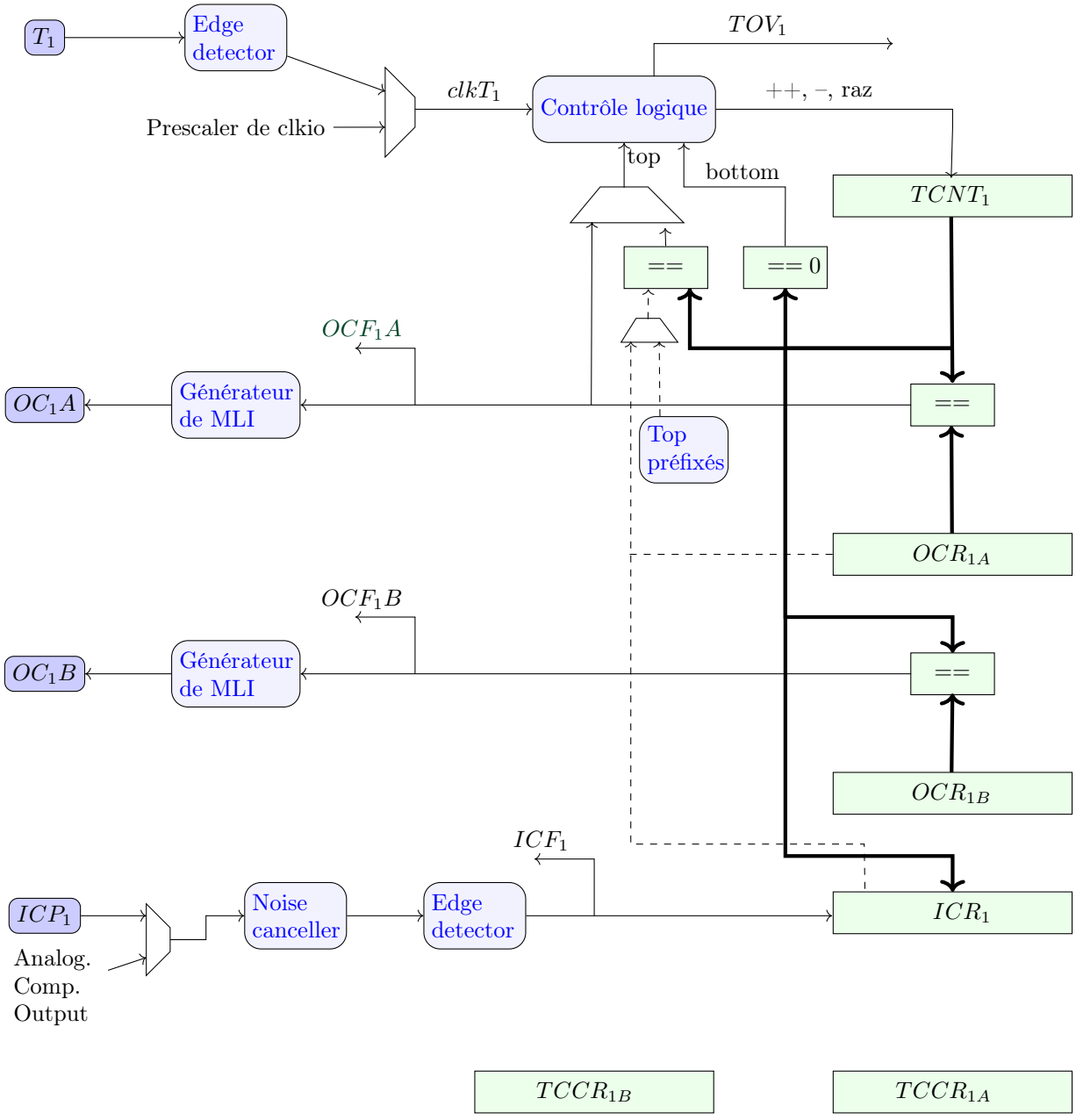
3.2 Le timer 1

3.2.1 Caractéristiques générales

Ses fonctions de base

Le timer 1 de l'$ATMEGA_8$ possède 16 bits, il possède 2 sorties indépendantes $OC1A$ et $OC1B$, sur lesquelles il peut générer 16 formes d'ondes différentes dont 12 *MLI*. De plus, il a une entrée de capture $ICP1$, avec annulation de bruit, lui permettant de compter des évènements externes. Il a aussi 4 sources d'it TOV_1, OCF_{1A}, OCF_{1B}, ICF_1

La figure suivante illustre tous les éléments et registres du timer 1 :



Registres à programmer pour le pilotage du timer 1

Figure 3.2: Aperçu de timer 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 $TIMSK$ Les 4 ITs du timer1 sont :

- TOV_1 : *Interruption* de débordement du 1 ;
- $OCF1_A$: *Interruption* de égalité entre $TCNT_1$ et $OCR1_A$;
- $OCF1_B$: *Interruption* de égalité entre $TCNT_1$ et $OCR1_B$;
- ICF_1 : *Interruption* de capture de front sur ICP_1 ;

Ces signaux d'*interruption* sont visibles dans avec des drapeaux contenus dans $TIFR$. De plus, ces *interruptions* sont masquables individuellement dans $TIMSK$.

$TIMSK$ Ce registre regroupe toutes les interruptions liés aux timers 2,1 et 0.
$OCIE_2$ $TOIE_2$ $TICIE_1$ $OCIE_1A$ $OCIE_1B$ $TOIE_1$ $-$ $TOIE_0$ ($TIMSK$)

- $TICIE_1$: "Timer 1, Input Capture Interrupt Enable" Dès que $TICIE_1$ et I sont à un, l'*interruption* de capture est activée. L'*interruption* ad hoc est alors exécutée lorsque $ICF1 \leftarrow 1$
- $OCIE1_{A/B}$: "Timer 1, Output Compare A/B Match Interrupt Enable". Dès que $OCIE1_{A/B}$ et I sont à un, l'*interruption* de comparaison est activée. L'*interruption* ad hoc est alors exécutée lorsque $OCF1_{A/B} \leftarrow 1$
- $TOIE_{1/0}$: "Timer 1/0, Overflow Interrupt Enabled" Dès que $TOIE_{1/0}$ et I sont à un, l'*interruption* de débordement est activée.

Contrôle de $TCNT_1$: $TCNT_1$ peut être incrémenté ou décrémenté de façon interne par le prédiviseur ou par une source externe patte T_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 **??**) est sélectionnée par $CS_{1_{2:0}} \in TCCR_{1B}$.

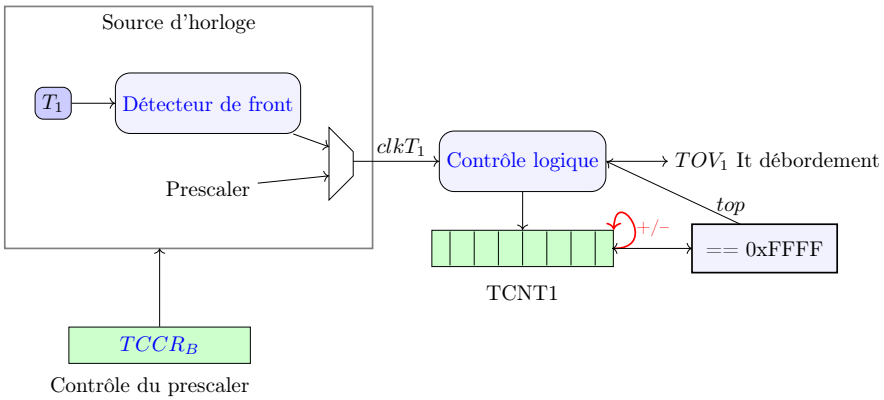


Figure 3.3: Controle du comptage du timer 1

Le choix de la valeur maximum de $TCNT_1$: 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 disponible (OCR_{1A} et OCR_{1B}), par contre l'entrée de capture n'est plus disponible.

$BOTTOM$	$BOTTOM = 0x0000$
MAX	$MAX = 0xFFFF$
TOP	$TCNT_1$ atteind TOP c.a.d quand il atteint soit : – $0xFF$, $0x1FF$, $0x3FF$ – $OCR1A$, ou bien ICR_1

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 utilisé pour générer l'*interruption* correspondante.

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 évènements 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 **??**) sur $ICP1$, 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.

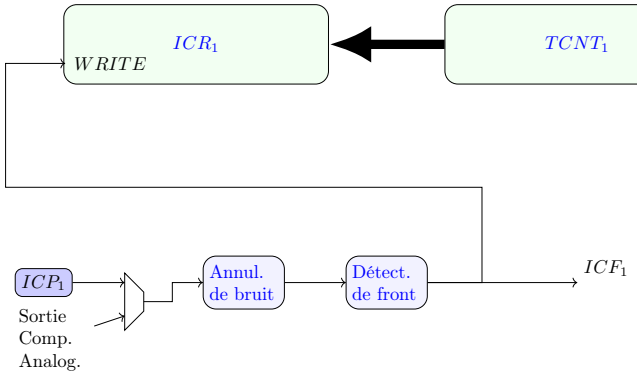


Figure 3.4: Capture d'évènement

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_1ne peut être mis à jour que lorque 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.*

[1] qui est la valeur courante de $TCNT_1$

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 $TCCR_{1B}$.

Capture des événements Tout évènement capturé écrase le précédent même si il n'a 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 aprés chaque capture et cela doit donc être fait aussitôt que ICR_1 a été lu.

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 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.

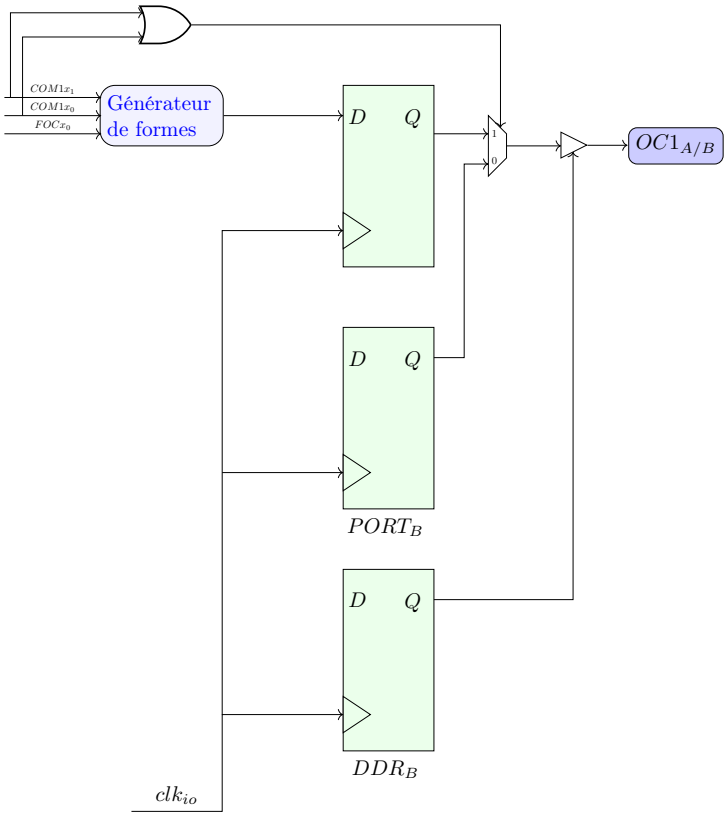


Figure 3.5: Comparateur

La figure indique que le port doit être configuré en sortie à l'aide du registre DDR_B et que la sortie sur les pattes $OC1A$ ou $OC1B$ deviendrons une sortie du générateur de forme (à la place de $PORT_B$) en fonction des modes de comparaison défini par $COM1x1:0$, ainsi que du bit FOC_x.

3.2.2 Les 16 modes du timer 1

Les 4 bits $WGM1_{3:0}$ permettent de définir 15 modes (le mode 13 est inutilisé) regroupés 5 catégories :

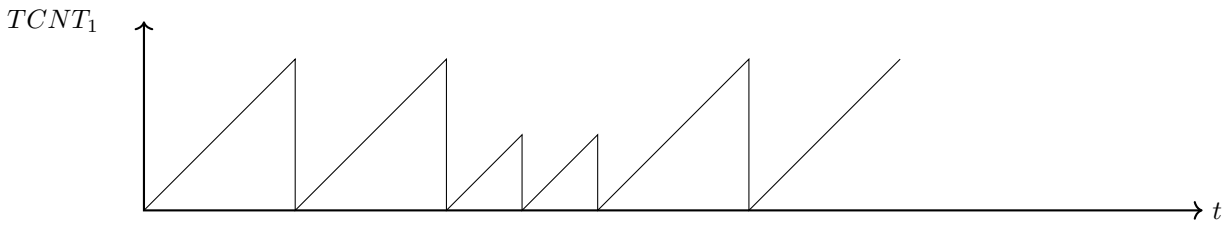
- 1 mode Normal
- 2 mode *Clear To Compare*
- 5 modes *PWM rapide*
- 5 modes à *phase correcte PWM*
- 2 modes à *phase et fréquence correcte PWM*(MLI centrée)

3.2.3 Les 16 Modes du générateur de formes : $WGM1_{3:0}$

a) Le mode Normal : $WGM1_{3:0}$=0000

Dans ce mode, le compteur s'incrémente (cf figure 3.2.3) jusqu'à la valeur TOP=0xFFFF et recommence à $BOTTOM$=0x0000, $tov1$ passe alors à 1. Ce mode ne sert pas à générer une PWM, mais il est plutôt utilisé pour générer une IT de débordement ou une it d'égalité avec un seuil défini dans le registre OCR_{1A}. Il peut aussi être utilisé pour dater des captures d'évènements.

Timer 1 en mode normal : IT de débordement possible sur les TOP

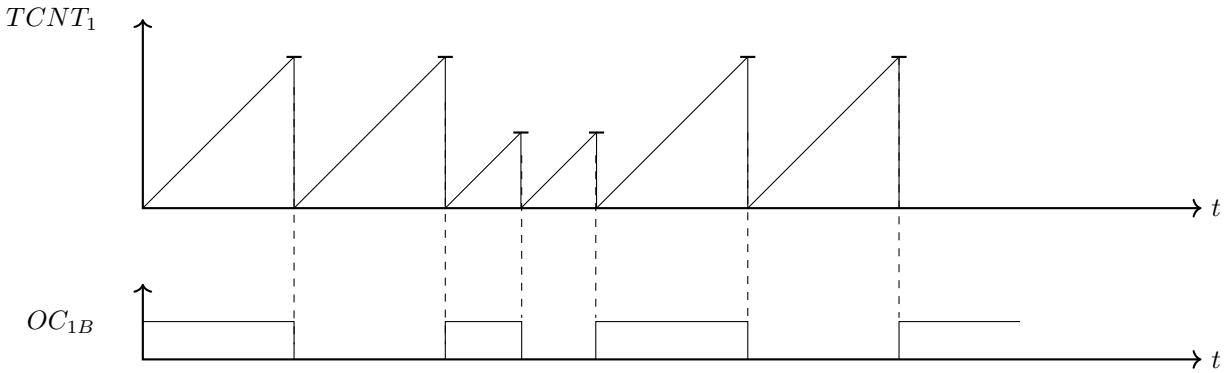


b) Le mode Clear Timer on Compare Match (CTC) $WGM1_{3:0}$ = 0100 ou 1100

Dans ce mode, OCR_{1A} (mode 4) ou ICR_1 (mode 12) peuvent définir TOP. Le compteur s'incrémente puis est raz quand il atteint TOP puis il recommence. Des its de débordement ou d'égalité sur seuil peuvent être mises en place.

Dans le cas ou $OC1A$ a été déclarée en sortie ($DDRB$) et que $COM1A1:0$ a été positionné à 10 alors on se place Cette patte va alors commuter à chaque fois que le compteur atteint TOP. (Voir la table Table 3.1 (hyperlien) pour le positionnement des bits com1x1:0)

Mode CTC: Commutation de OC_{1B} sur TOP



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

Cette *PWM* à simple pente permet de générer des *PWM* deux fois plus rapide. Ce mode peut être utile pour les applications de regulation de puissance, ou de conversion numérique-analogique. $TCNT_1$ compte de *BOTTOM* à *TOP* et recommence à partir de *BOTTOM* .

Comme l'indique la figure (Figure 3.2.3 (hyperlien)), en mode non inversé, lorsque $TCNT_1$ atteint le seuil OCR_{1A}, OC_{1A} est raz sur égalité de seuil et set à *BOTTOM*. Des its de débordement ou d'égalité sur seuil peuvent être mises en place. *TOP* peut être défini avec les valeurs 0x00FF, 0x01FF, or 0x03FF (mode 5, 6, ou 7) ou par ICR_1 (mode 14) ou bien par OCR_{1A} (mode 15). Utiliser ICR_1 pour définir le *TOP* permet d'utiliser OCR_{1A} comme registre de seuil et de générer une *PWM* sur $OC1x$. Positionner les bits $com1x1:0$ (voir la table Table 3.2) permet de définir un mode inversé (ou non). La *PWM* est obtenu par des set ou des raz de $OC1x$

NB : On peut générer un front étroit en mettant OCR_{1x} à *BOTTOM*. En mettant OCR_{1x} à *TOP* on a un niveau haut.

– : Egalité de comparaison de $TCNT_1$ avec OCR_{1B}

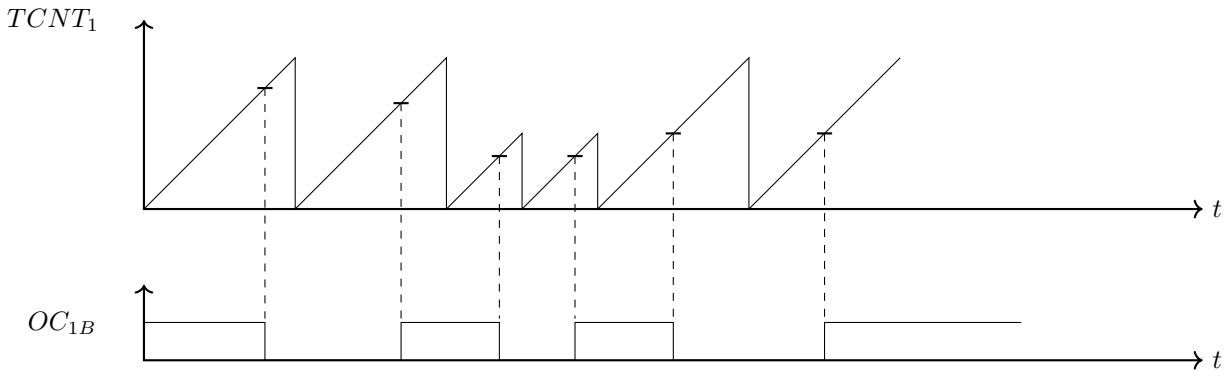


Figure 3.6: Chronogramme du mode Fast PWM

Fréquence Dans ce mode la fréquence est définie par : $f_{OC_{1A}PWM} = \frac{f_{clk_{I/O}}}{N.(1+TOP)}$

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

c) 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'a *BOTTOM* et recommence indéfiniment. *TOP* est défini par les valeurs 0x00FF, 0x01FF, ou 0x03FF (mode 1, 2, or 3), ou bien par ICR_1 en mode 10, ou encore par OCR_{1A} en mode 11.

Instant de commutation de $OC1_x$ Les instants de commutations correspondent aux moment ou le registre OCR_{1x} est égal à $TCNT_1$. On aura le même principe pour le mode suivant (Phase et fréquence correcte). On voit ces instants de commutation dans la figure **??** ou les événements se produisent lors des montées et des descentes :

- En mode non inversé, $OC1_x$ est **raz** sur l'égalité de $TCNT_1$ avec OCR_{1x} en comptant, et **mis à un** sur l'égalité de $TCNT_1$ avec OCR_{1x} en décomptant.
- En mode inversé, $OC1_x$ est **mis à un** sur l'égalité de $TCNT_1$ avec OCR_{1x} en comptant, et **raz** sur l'égalité de $TCNT_1$ avec OCR_{1x} en décomptant.

(cliquer sur le lien Table 3.3 pour le positionnement des modes inversés ou non inversés)

Remarque 3.2.3 *La différence avec le mode "Phase et fréquence correcte" se situe dans la mise à jour de OCR_{1x} . En phase correcte OCR_{1x} et TOP sont mis à jour sur TOP. Cela signifie que si OCR_{1x} est modifié pendant une pente, sa nouvelle valeur n'est prise en compte que sur le prochain TOP.*

En application de cette remarque, on peut voir sur la figure 3.2.3 sur le quatrième triangle on a changé OCR_{1x} et il est pris en en, compte après le top sur la pente descendante.

Ce mode à double pente et son caractère symétrique (MLI centrée) le rend plus utilisé dans les applications de commande de moteurs. Pour des valeur de TOP statiques les modes phase correcte et phase et fréquences correctes sont presque identiques. Cependant, si l'on veux changer TOP alors il vaut mieux utiliser le mode phase et fréquence correcte. En mode phase correcte il y a des asymétries dues, à la remarque précédente, qui apparaissent lors du changement de TOP si la nouvelle valeur est inférieure à l'ancienne.

TOV_1 est positionné sur débordement du compteur et peut être utilisé pour declencher une it. Une autre it est celle de comparaison sur égalité de seuil.

– : Égalité de comparaison avec de TOP avec OCR_{1B}

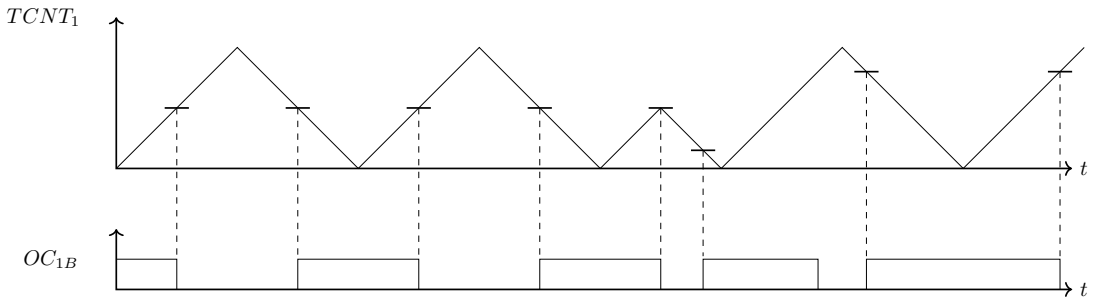


Figure 3.7: Chronogramme du mode phase correcte

Fréquence Dans ce mode la fréquence est définie par : $f_{OC_{1A}PCPWM} = \frac{f_{clk_{I/O}}}{2.N.(TOP+1)}$

ou 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. (consulter la table Table 3.3 pour le positionnement des modes inversés ou non inversés)

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

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 jout 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 minimum plus basse comparée à celle en simple pente. Cependant son caractère très symétrique en fait le mode le plus utilisé en terme de contrôle moteur.

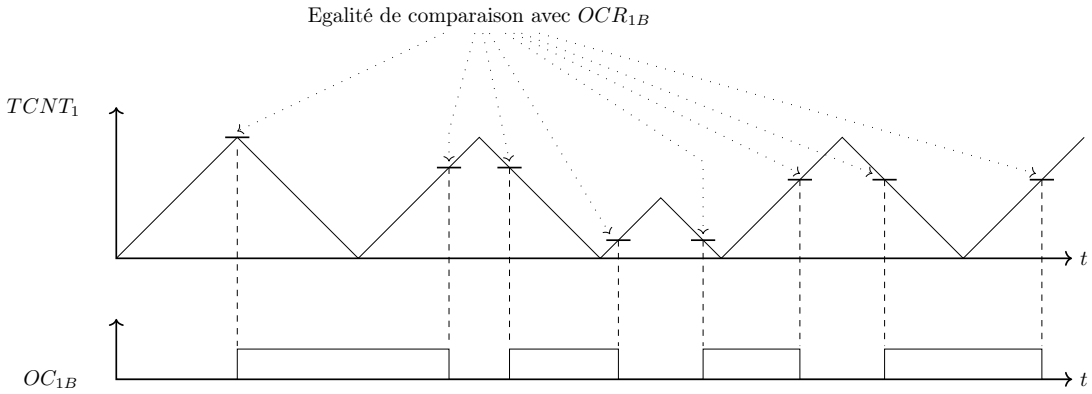


Figure 3.8: Mode Phase et fréquence correcte

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 $PORTB$ que si il a été programmé en sortie via 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 : $f_{OC_{1A}PFCPWM} = \frac{f_{clk_{I/O}}}{2.N.(1+TOP)}$

ou 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}

$COM1x1$	$COM1x0$	*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

Table 3.1: Mode normal et CTC

$COM1x1$	$COM1x0$	*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** $OC1x$ sur égalité **mis à un** sur $BOTTOM$
1	1	**mis à un** sur égalité **raz** sur $BOTTOM$

Table 3.2: Mode Fast PWM

$COM1A_1$ $COM1B_1$	$COM1A_0$ $COM1B_0$	*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

Table 3.3: Mode phase correcte et phase et fréquence correcte

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'échantillonage. 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 précédement, 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_{10}

$ICNC_1$ **: 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 $TCNT_1, OCR1_x, ICR_1$

$TCNT_1$ 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 continument 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 $OCF1_A \in TIFR$ 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 $OCF1_B \in TIFR$ 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_1est automatiquement **raz** par le sous-programme d'*interruption* est exécuté. Sinon ICF_1 peut être **raz** en y écrivant un un.

$OCF1_A$**: "Timer/Counter1, Output Compare B Match Flag"** :

$OCF1_A$ est un drapeau (flag) qui reçoit un quand l'égalité de OCR_{1A} avec $TCNT_1$ se produit. $OCF1_A$ est automatiquement **raz** par le sous-programme d'*interruption* est exécuté. Sinon $OCF1_A$ peut être **raz** en y écrivant un un.

$OCF1_B$**: "Timer/Counter1, Output Compare B Match Flag"** :

Ce bit est un drapeau (flag) qui reçoit un quand l'égalité de $OCR1_B$ avec $TCNT_1$ se produit. $OCF1_B$ est automatiquement **raz** par le sous-programme d'*interruption* est exécuté. Sinon$OCF1_B$ peut être **raz** en y écrivant un un.

TOV_1**: "Timer/Counter1, Overflow Flag"** :

Dans les modes normal et CTC modes, TOV_1 est mis à un sur un dépassement de capacité TOV_1 est automatiquement **raz** par le sous-programme d'*interruption* est exécuté. Sinon TOV_1 peut être **raz** en y écrivant un un.

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

v	WGM_{13}	WGM_{12}	WGM_{11}	WGM_{10}	*Mode* des Timers	*TOP*	Mise à jour *de* $OCR1_x$	Mise à un *de* TOV_1
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*	$OCR1_A$	*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	ICR_1	*BOTTOM*	*BOTTOM*
9	1	0	0	1	PWM Phase, freq. correctes	$OCR1_A$	*BOTTOM*	*BOTTOM*
10	1	0	1	0	PWM Phase Correcte	ICR_1	*TOP*	*BOTTOM*
11	1	0	1	1	PWM Phase Correcte	$OCR1_A$	*TOP*	*BOTTOM*
12	1	1	0	0	*CTC*	ICR_1	*immediate*	*MAX*
13	1	1	0	1	Réservé	–	–	–
14	1	1	1	0	Fast *PWM*	ICR_1	*BOTTOM*	*TOP*
15	1	1	1	1	Fast *PWM*	$OCR1_A$	*BOTTOM*	*TOP*

3.2.6 Exemples d'utilisation du timer 1

-1- *MLI* centrée de 66% sur la patte $PB_2(OC1_B)$.

```
int main (void) {
  int i;
  DDRB = 0x04;               //PB2 en sortie

  TCCR1A= (1<<COM1B1)+(1<<WGM10); //Mli centree
  TCCR1B= (1<<WGM13)+(1<<CS10); // Prediv N=1
  OCR1A= 532;    // Fh = 15khz,  Th = 66 mus
  OCR1B= 361;    // Ton/Thash = 0.66
  while(1) {}
}
```

-2- Interruption de débordement du timer 1:

```
ISR (TIMER1_OVF_vect) {
  static byte Compteur;
  if (Compteur++ == 50) {// commuter led 1/50
    Compteur=0;
    PortB^=1;  // clignotement PB0
  }
}

int main() {
  DDRB=0x01; //PB0 sortie

  TCCR1A = 0;          // Mode normal
  TCCR1B = 1<<CS10;    //Prediv N=1
  TIMSK = 1<<TOIE1;    //It debordement
  sei();
  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

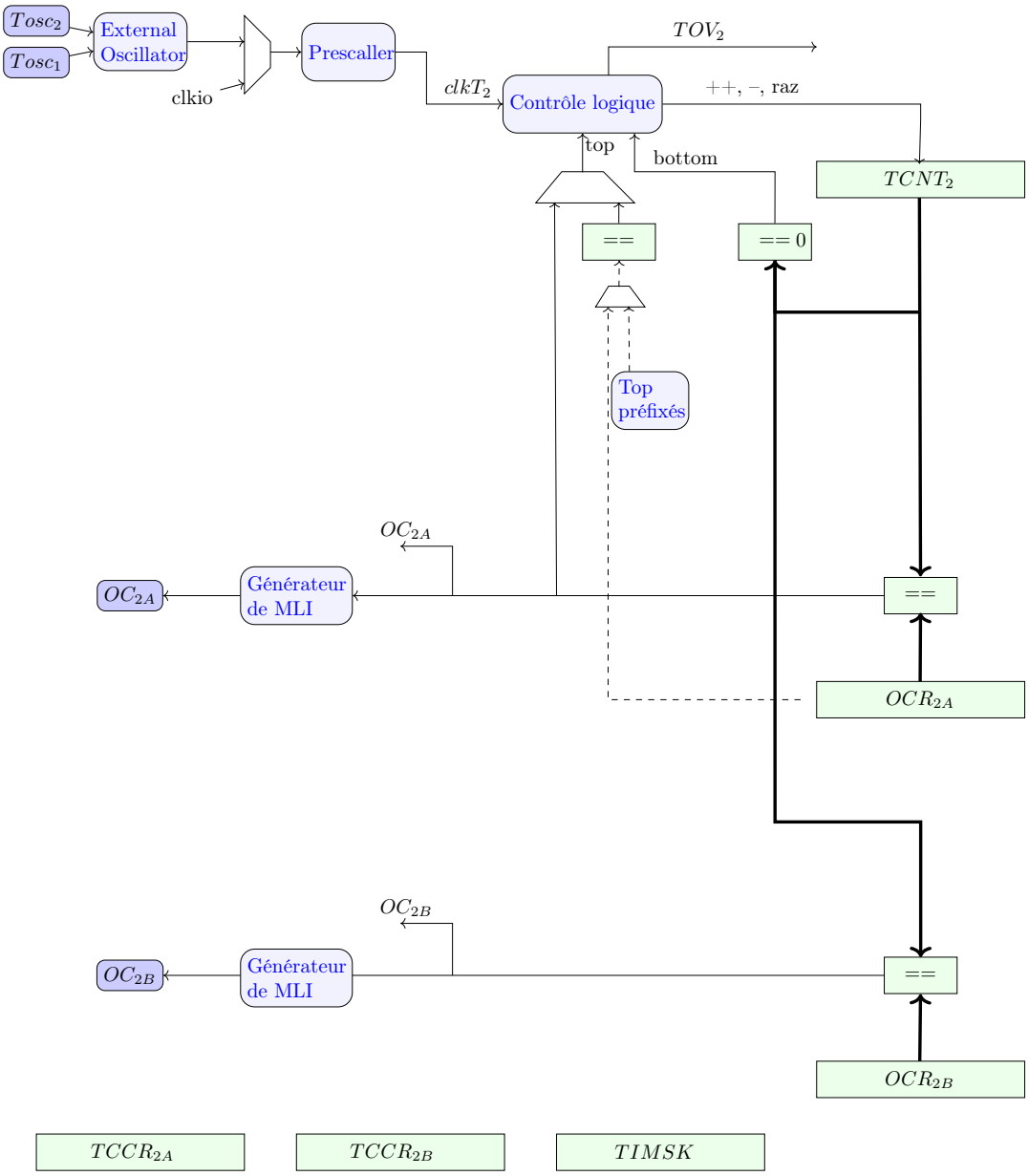


Figure 3.9: Aperçu du timer 2

Le timer 2 est un compteur 8 bits aussi, $TCNT_2$, OCR_2 sont des registres de 8 bits. 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'incrémentation. OCR_2 est comparé à tout moment à $TCNT_2$ et le test de cette égalité peut être utilisé à tout moment pour générer des formes

de type PWM ou pour faire varier une fréquence sur les pattes OC_{2A} OC_{2B}. 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

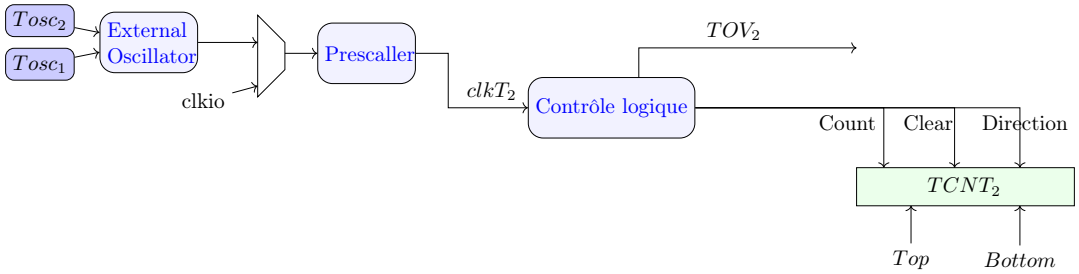


Figure 3.10: Controle du comptage du timer 2

- Comptage : incrémentation ou dé-crémentation ou remise à zéro ;
- $clkT2$ 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 ;
- $WGM2_1$ $WGM2_1 \in TCCR_2$ permettent de déterminer la séquence de comptage.

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

Les Mli sont obtenues en comparant à tout moment les registres *ocr2* et *tcnt2* :

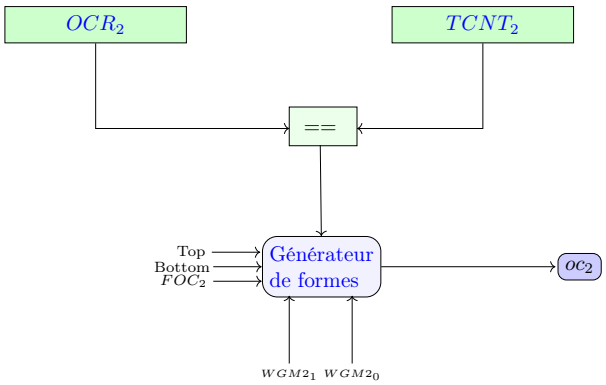


Figure 3.11: Génération de formes

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 ou $TCNT_2$ 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 $OC2$ (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

Le schéma suivant (3.3.2) exprime que, si on a configuré le registre DDR en sortie (porte avant la pin $oc2$), alors la patte $oc2$ pourra recevoir le signal du bloc générateur de formes (ou bien $PORT$) :

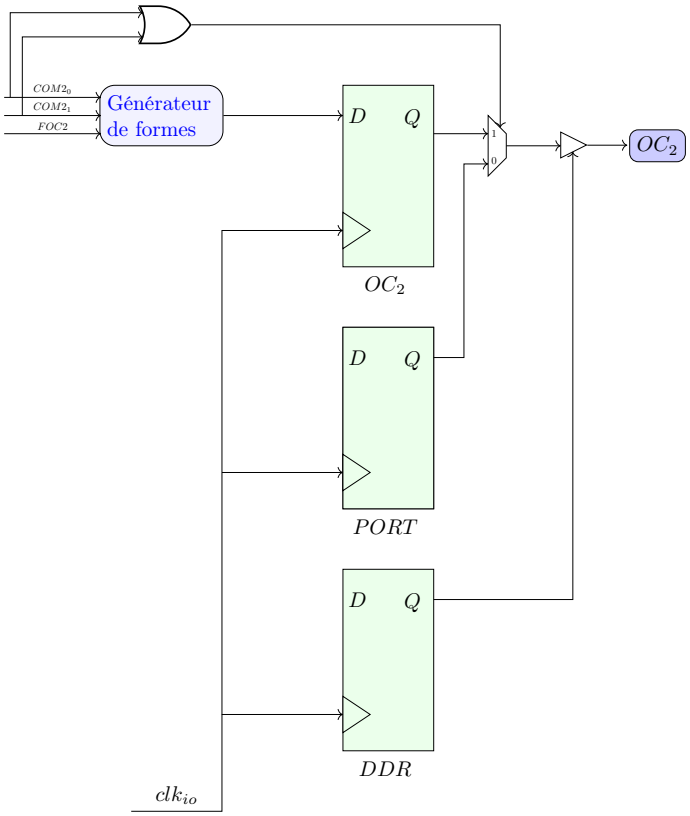


Figure 3.12: Unité de comparaison

- $COM2_{1:0}$ contrôle 2 fonctions :
  - La sortie de comparaison du générateur de formes
  - Ce que reçoit OC_2 dans le cas ou 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'a $0XFF$ puis recommence depuis 0. TOV_2 passe à un quand $TCNT_2$ retombe à zero. TOV_2 sert alors de $9^{\text{ième}}$ 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 le top et donc la résolution. $TCNT_2$ est **raz** quand $TCNT2$ devient égal à OCR_2 qui définit la valeur TOP (cf figure 3.3.3).

Mode CTC: Commutation de OC_2 sur TOP

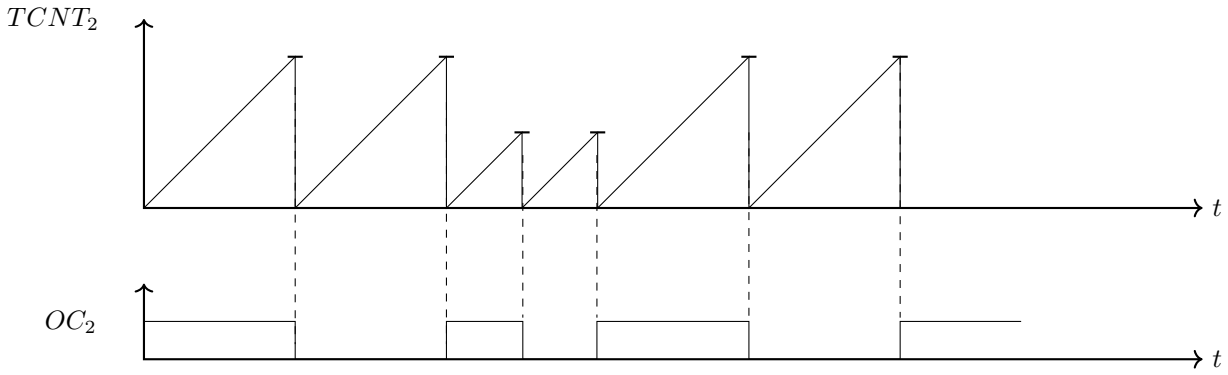


Figure 3.13: le timer 2 : Mode Clear timer on Compare

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 $TCNT_2$.

Pour ce mode, la fréquence s'obtient par la formule : $f_{OC_2} = \frac{f_{clk_{I/O}}}{2.N.(1+OCR_2)}$

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 MAXn 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 $BOTTOM$ Pour ce mode, la fréquence s'obtient par la formule : $f_{OC_2PWM} = \frac{f_{clk_{I/O}}}{N.(1+OCR_2)}$

– : Egalité de comparaison de $TCNT_2$ avec OCR_2

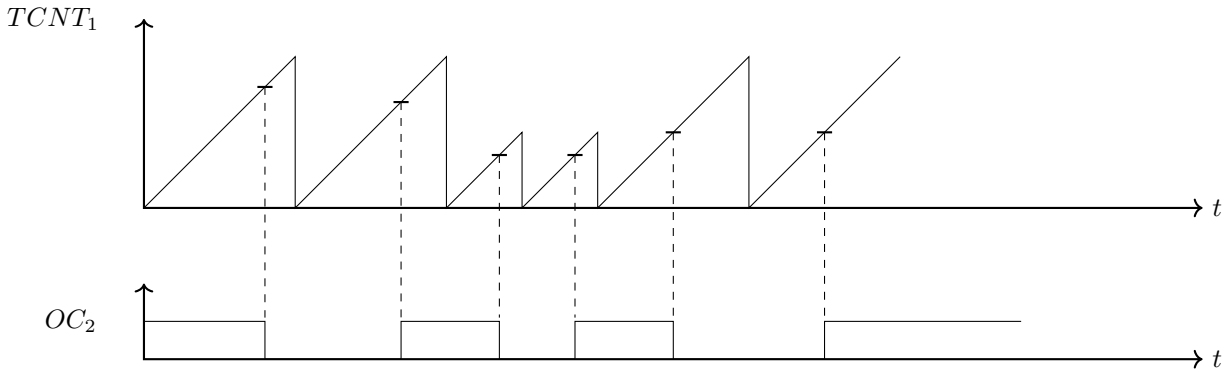


Figure 3.14: le timer 2 : Mode PWM rapide

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'a $BOTTOM$;

inversé/non inversé En mode non inversé, en comptant $OC2F$ est **raz** sur égalité de $TCNT_2$ et OCR_2 , tandis que en décomptant, OC_2a ou OC_2b sont **mis à un** sur sur égalité de $TCNT_2$ et OCR_2 . On a un mode ici symétrique : "phase correcte". Pour ce mode, la fréquence s'obtient par la formule : $f_{OC_2} = \frac{f_{clk_{I/O}}}{2.N.(1+OCR_2)}$

– : Égalité de comparaison avec TOP avec OCR_2

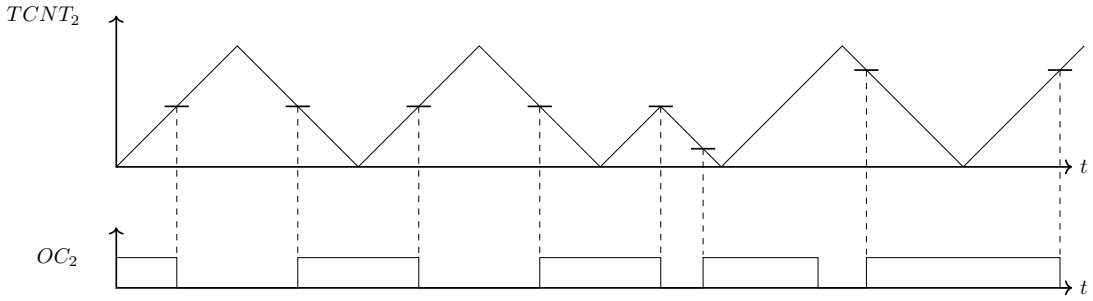


Figure 3.15: Génération de formes

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 êtr **raz** quand $TCCR_2$ est mis à jour en mode PWM . Quand FOC_2← 1 une comparaison immédiate est forcée selon la valeur des bits de $COM2_{1:0}$ OC_2 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 $BOTTOM$
1	1	OC_2 est **mise à un** sur égalité, **raz** sur $BOTTOM$

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 $TCCR2U_B \leftarrow 1$. Un niveau bads 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 $TCN2U_B$, $OCR2U_B$ ou $TCR2U_B$ 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 utliserait 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 $TOV_2 \leftarrow 1 (\in TIRF)$

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 TOV_2 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 en langage *C*

4.1 Structure de programme

Un programme C sur microcontrôleur est composé de trois parties. Les entêtes, les fonctions nécessaires et le programme principal (main) qui contiendra la boucle infinie.

```
//----------------ENTETE
#define M1 0x1F
#define maxDelai 262

//----------------FONCTIONS
void sleep(int dizaines){
  int i=0;
  for(i=0;i<10*dizaines;i++) _delay_ms(10);
}
//----------------MAIN
int main(void)
{
   DDRD = 0b1111011;
      do {
        PORTD ^=0x0F;
        sleep(10);
      } while(1);
}
```

4.1.1 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
#define CommuteLed PORTB ^=0x01  // Macro qui commute PB0
```

4.1.2 Main

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

```
void fonction1(){...}
int fonction2(){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);
}
```

4.1.3 Fonctions

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

```
int carre(int d){return d*d;}

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

void entier(int nbtours, int sens){
int i,j;
int Tentier[4]={1,2,4,8};
  for(j=0;j<nbtours*12;j++) {
    for(i=3;i>=0;i--){

  sleep(2);
  PORTB=Tentier[i];
    }
  }
}
```

4.1.4 Déclaration de variable globales : attribut volatile

Pour les variables globales, on aura parfois besoin d'utiliser l'attribut volatile qui est une directive qui indique au compilateur de déclarer une variable dans la RAM et non dans la zone de registres. On utilisera cet attribut quand une variable dans deux cas :

- Quand la variable peut être, à la fois, modifiée dans un programme d'interruption et à la fois lue dans le programme principal. On évite alors des problèmes de mise en cache et de synchronisation entre l'interruption et la boucle principale qui font que la variable a peut être une valeur qui ne sera pas mise à jour.
- Détection d'un changement d'état : Si vous utilisez une interruption pour détecter un changement d'état sur une broche (par exemple, un bouton-poussoir), vous pouvez déclarer la variable de détection comme "volatile" pour vous assurer qu'elle sera mise à jour instantanément, elle ne sera pas mise en cache.
- Attention, cet attribut dégrade par contre le temps d'accès à la variable.

4.1.5 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 veux modifier un registre
// Si l'on veux 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);
```

4.2 Les interruptions

4.2.1 Mise en oeuvre

Une interruption se met en oeuvre par :

- L'autorisation de la source d'IT : Par exemple pour le timer $TOIE_i$ pour le timer i, ou le $ACIE$ pour l'ADC.
- l'autorisation de toutes les ITs : sei();
- la déclaration de la routine d'interruption : Directive ISR.

La directive ISR associe une source d'interruption à une routine :

```
ISR(TIMER0_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/IO
   sei(); // Toutes les ITs sont possibles
   TIMSK |= (1<<TOIE0); // Validation de l'It de debordement
   do {} while(1);
   return(1);
}
```

4.2.2 Description des sources d'Interruptions de l'$ATMEGA_8$

4.2.3 Convertisseur Analogique Numérique

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

4.2.4 Mémoire EEPROM

Nom de l'interruption	Description
EE_RDY_vect	EEPROM Ready
EE_READY_vect	EEPROM Ready
EXT_INT0_vect	External Interrupt Request 0

4.2.5 Interruptions externes

Nom de l'interruption	Description
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

4.2.6 Interruptions Diverses

Nom de l'interruption	Description
LCD_vect	LCD Start of Frame
LOWLEVEL_IO_PINS_vect	Low-level Input on Port B
OVRIT_vect CAN	timer Overrun
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

4.2.7 Timer 0

Nom de l'interruption	Description
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

4.2.8 Timer 1

Nom de l'interruption	Description
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

4.2.9 Timer 2

Nom de l'interruption	Description
TIMER2_COMPA_vect	Timer/Counter2 Compare Match A
TIMER2_COMPB_vect	Timer/Counter2 Compare Match A
TIMER2_COMP_vect	Timer/Counter2 Compare Match
TIMER2_OVF_vect	Timer/Counter2 Overflow

4.2.10 TWI

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

4.2.11 UART

Nom de l'interruption	Description
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

4.2.12 USART

Nom de l'interruption	Description
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

4.2.13 USI

Nom de l'interruption	Description
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

4.2.14 Watchdog

Nom de l'interruption	Description
WATCHDOG_vect	Watchdog Time-out
WDT_OVERFLOW_vect	Watchdog timer Overflow
WDT_vect	Watchdog Timeout Interrupt