**CHAPITRE 6:**

**Les Timers / Counters**

Généralement, nous utilisons les **Timer/Counter** pour générer des retards, des formes d’onde, pour compter les événements ou pour la génération PWM, etc.

Dans AVR ATmega16 / ATmega32, il y a trois **Timer/Counter**:

* **Timer 0** : minuterie 8 bits
* **Timer1** : minuterie 16 bits
* **Timer2** : minuterie 8 bits

|  |  |  |
| --- | --- | --- |
| **Timer 0**   * 8-bit timer/counter * 10-bit clock prescaler * Functions:   -- Pulse width modulation  -- Frequency generation  -- Event counter  -- Output compare -- 2 ch  - Modes of operation:  -- Normal  -- Clear timer on compare match (CTC)  -- Fast PWM  -- Phase correct PWM | **Timer 1**   * 16-bit timer/counter * 10-bit clock prescaler * Functions:   -- Pulse width modulation  -- Frequency generation  -- Event counter  -- Output compare -- 2 ch  -- Input capture  - Modes of operation:  -- Normal  -- Clear timer on compare match (CTC)  -- Fast PWM  -- Phase correct PWM | **Timer 2**   * 8-bit timer/counter * 10-bit clock prescaler * Functions:   -- Pulse width modulation  -- Frequency generation  -- Event counter  -- Output compare -- 2 ch  - Modes of operation:  -- Normal  -- Clear timer on compare match (CTC)  -- Fast PWM  -- Phase correct PWM |

**Registres de base et drapeaux des timers**

**TCNTn: registre** **Timer/Counter**

* Il compte à chaque impulsion d'horloge.
* lors de la réinitialisation **TCNTn** = 0.
* Nous pouvons lire / écrire une valeur dans ce registre.

**TOVn: indicateur de débordement de Timer/Counter**

           Chaque **Timer/Counter** a un indicateur de dépassement. Lorsqu’il y a un débordement, cet indicateur est activé.

**TCCRn: Registre de contrôle du Timer/Counter**

           Ce registre est utilisé pour régler les modes de **Timer/Counter**.

**OCRn: registre de comparaison de sortie**

          La valeur de ce registre est comparée au contenu du registre TCNTn. Quand ils sont égaux, le drapeau OCFn sera activé.

**Configuration du Timer/Counter 0**



**Figure 6.1:** Timer 0

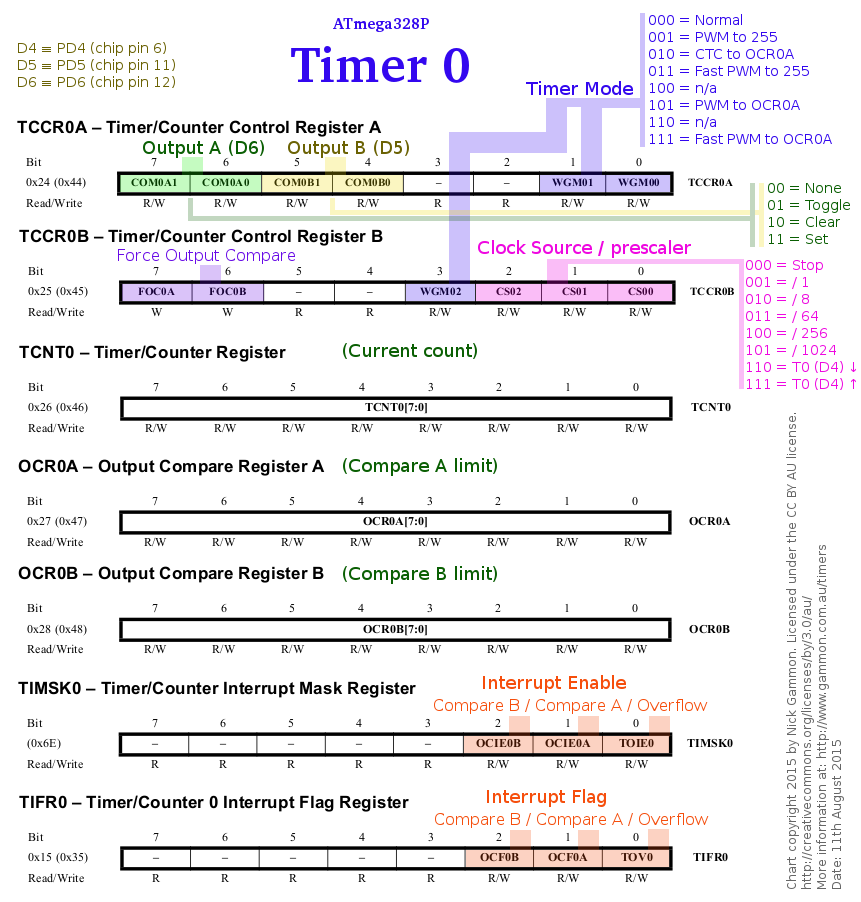
**Les registres du Timer/Counter 0**

1. **TCNT0:**    C'est un registre de 8 bits. Il compte à chaque impulsion d’horloge.

2 **. TCCR0: Registre de contrôle de Timer/Counter 0**

            Il s’agit d’un registre de 8 bits utilisé pour le mode de fonctionnement et la sélection de la source d’horloge.

   3. **TIFR: registre d'indicateurs d'interruption de Timer/Counter 0**



**Figure 6.2: Timer 0 registers.**

**Modes de fonctionnement en mode temporisateur :**

**Normal :** le registre TCNT0 est incremente de 1 a chaque evenement de comptage. Le registre ne

revient a 0 qu'apres un debordement (passage de 0xFF a 0x00).

**CTC (Clear Timer on Compare)** : le registre TCNT0 s'incremente a chaque evenement de

comptage mais est remis a 0 si TCNT0=OCR0A.

D'autres modes non detailles, notamment pour la gestion des PWM.

**Le choix du mode se fait via les bits WGM02:00 (bits TCR0A et TCR0B)**

**Clear Timer on Compare Match (CTC) Mode :**

En mode CTC (WGM02:0 = 2), le registre OCR0A regle la resolution. Le compteur TCTN0 est

remis a zero apres l'egalite (match) TCTN0=OCR0A. Le registre OCR0A definit la valeur

maximale pour le compteur, et donc sa resolution.

**Prescaler :** en fonction temporisateur, le registre de comptage TCNT0 est incremente en fonction des cycles horloge. L'increment peut etre a chaque cycle horloge (pas de prescaler) ou bien a une frequence moindre. Rappelons que le cycle horologe est de 1/16 micro-secondes.

**Remarque :** pour les Timers 1 et 2, les configurations sont similaires.

**Exemple :**

Pour le Timer 0, a chaque debordement de TCNT0 (passage de 0xFF a 0x00) une interruption

**TIMER0\_OVF\_vect** peut etre exploitee. Il faut donc activer ce mecanisme

**TIMSK2=0x01;** // IT Timer0 Over Flow Active

Fournir la routine d'IT :

**ISR(TIMER0\_OVF\_vect)**{ ... } //ISR de l'IT débordement Timer0

et configurer le Timer 0.

// ARDUINO UNO - IT Timer0 Overflow

**volatile unsigned char cpt=0; // compteur d'IT**

// Fonction de traitement IT Timer 0 OverFlow

**ISR(TIMER0\_OVF\_vect)**{

cpt++; if(cpt==61) {

PORTB ^=0x20;

cpt=0;

}}

void setup(){

//Configuration PORTB.5

DDRB |= 0x20; // PB5 en sortie

PORTB &= ~0x20; // PORTB.5 <-0

cli();

// Configuration Timer 0

**TCCR0A=0;** // Mode Normal (pas PWM)

**TCCR0B=0x07;** // Prescaler 1024 (Clock/1024)

**TIMSK0=0x01;** // IT Timer0 Over Flow Active

**sei();** // activation des IT (SREG.7=1)

}

void loop() { /\* aucun traitement\*/ }

**Principe :** apres la fonction setup(), le registre **TCNT0** (8bits) est incremente a chaque tick du signal periodique clock/1024. chaque debordement du registre TCNT0, declenche une interruption : "Timer0 Over Flow". Tous les 60 appels de cette fonction, la broche PB5 (LED) change d'etat.

**Quelle fréquence de clignotement ?**

**Clock =** signal periodique de 16MegaHz (16000000 de cycles par seconde)

**Prescaler** = 1024 -> la frequence d'increment de TCNT0 = (16/1024) MegaHz

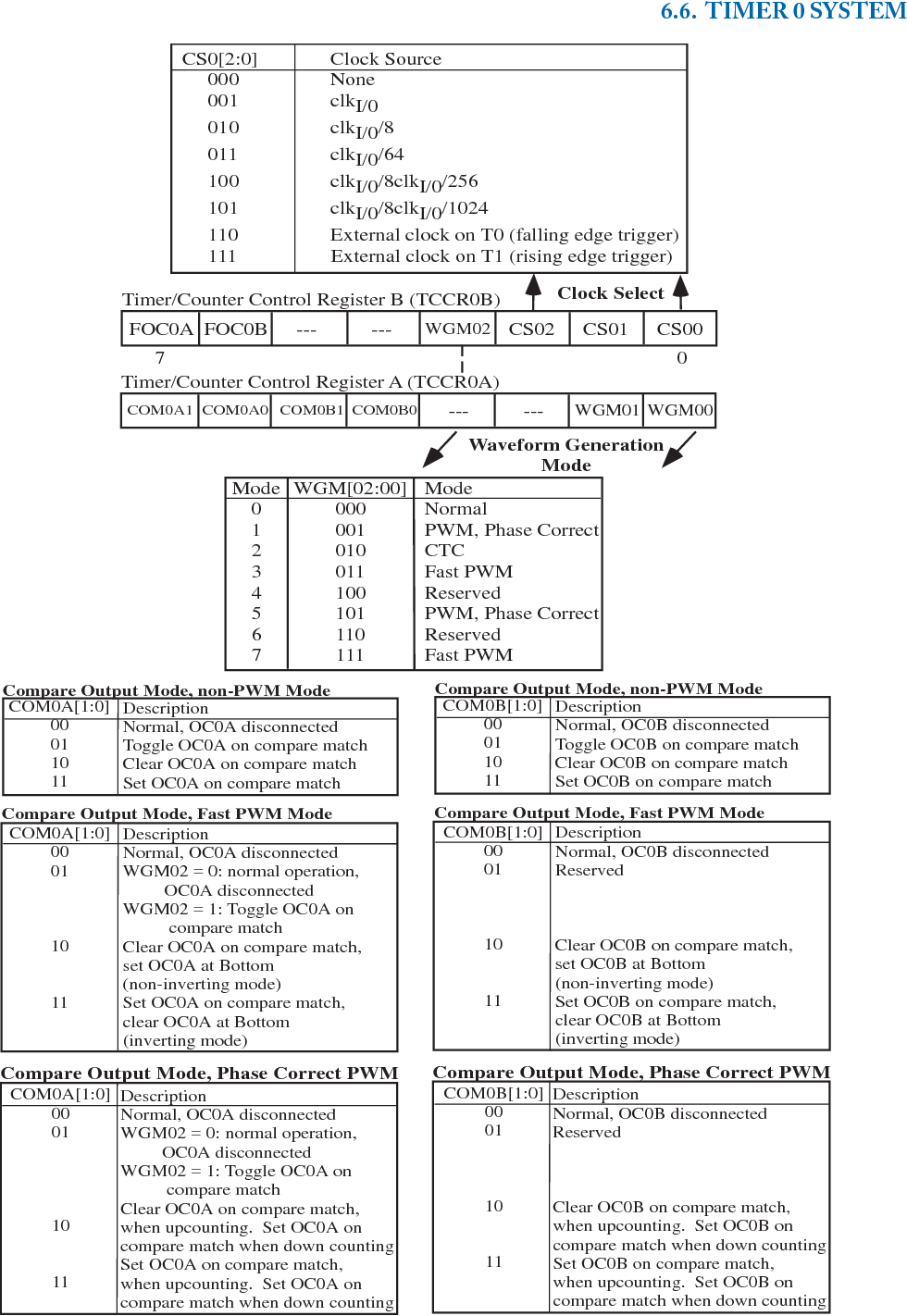
**Fréquence de débordement :** TCNT0 ne deborde que tous les 256 increments

c'est-a-dire a la frequence de 16/(1024\*256) MegaHZ ≈ 61 Hz

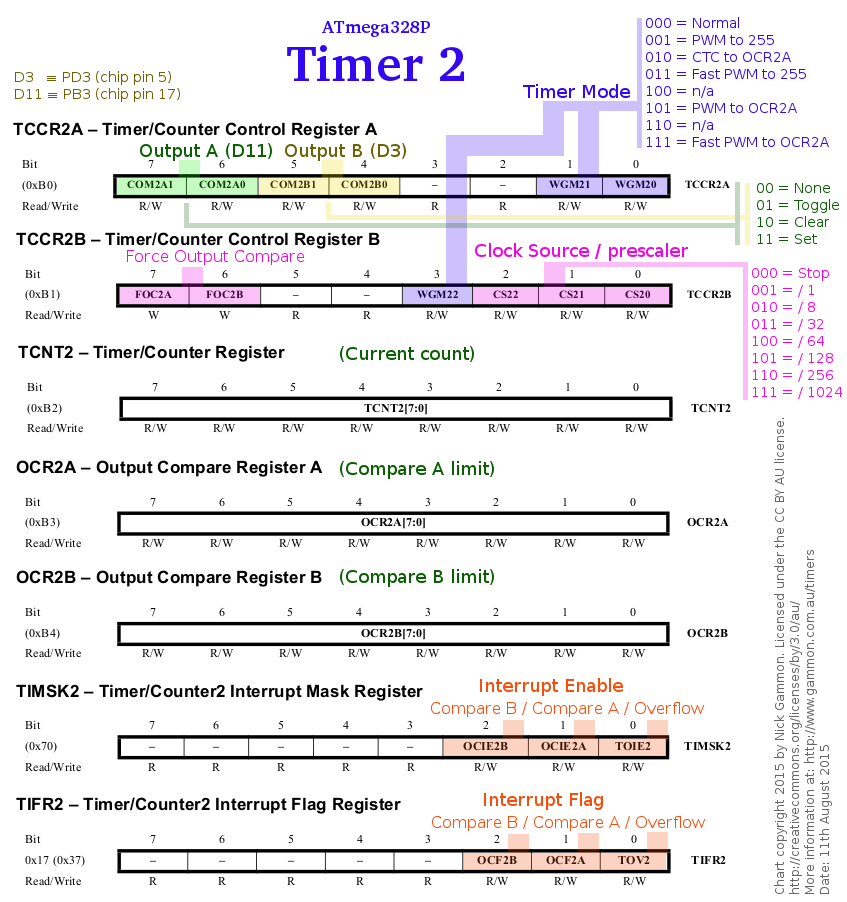
Il y a 1 interruption Timer0 Over Flow tous les 1/61 secondes.

Il faut 61 Interruptions pour que la LED change d'etat

**La LED change d'état (0->1 1->0) à intervalles de environ 1 second**



**Figure 6.3**



**Figure 6.4 : Timer 2 registers.**

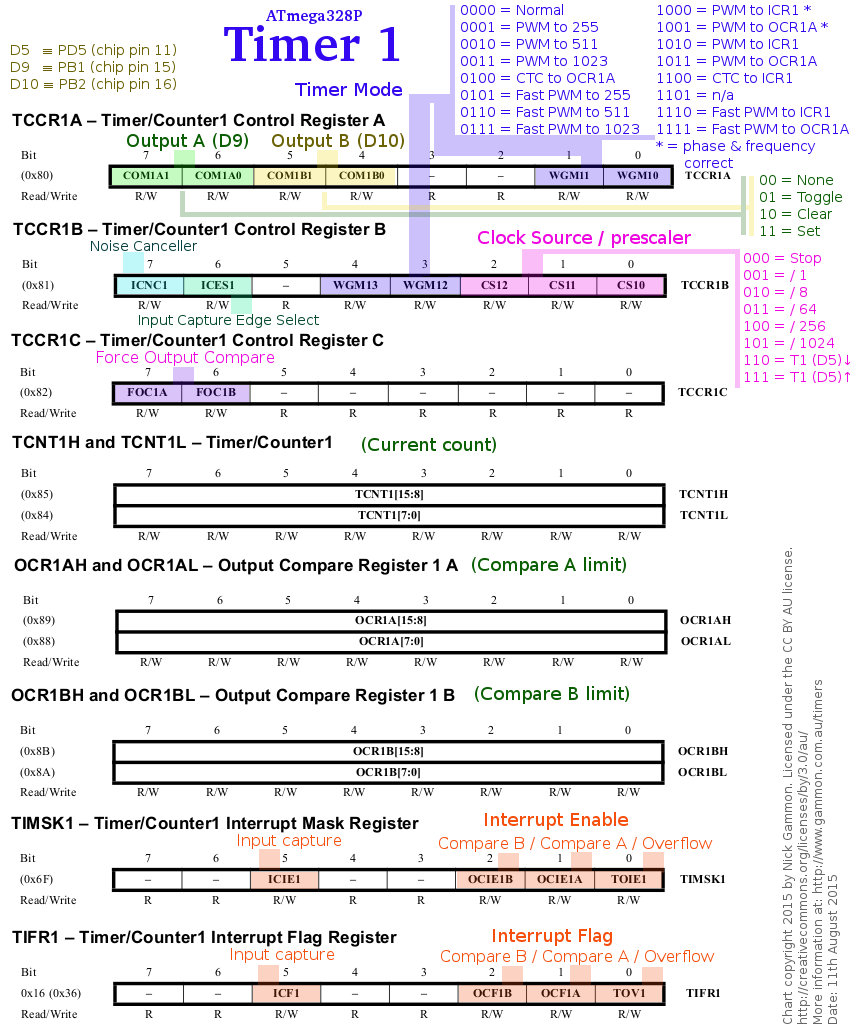
**Timer/Counter 1 (comptage 16 bits)**

Le registre de comptage TCNT1, ainsi que les registres de comparaison OCR1A et OCR1B, sont

Cette fois-ci sur 16 bits.



**Figure 6.5 : Timer 1 .**



**Figure 6.6 : Timer 1 registers.**

**PWM dans AVR ATmega16 / ATmega32**

**Introduction :**

**La modulation de largeur d'impulsion (PWM)** est une technique qui permet de faire varier la largeur d'une impulsion tout en maintenant la fréquence constante.

Pourquoi avons-nous besoin de faire ça ? Prenons un exemple de contrôle de la vitesse du moteur à courant continu, plus la largeur d'impulsion est grande, plus la vitesse. Il existe également des applications telles que le contrôle de l'intensité lumineuse par PWM.

Une période d'impulsion comprend un cycle **ON** (5V) et un cycle **OFF** (0V). La fraction pour laquelle le signal est activé sur une période est appelée **cycle de travail**.

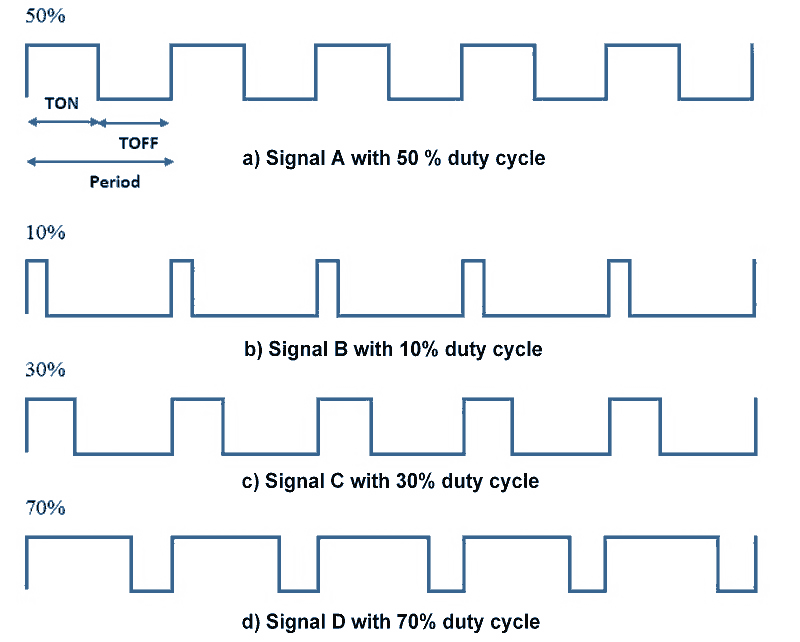
Cycle de service (en%) = \ frac {T_O_N} {Période totale} * 100

Par exemple, considérons une impulsion avec une période de 10 ms qui reste activée (haute) pendant 2 ms. Le cycle de service de cette impulsion sera

D = 2ms / 10ms = 20%

Grâce à la technique PWM, nous pouvons contrôler la puissance fournie à la charge en utilisant le signal ON-OFF.

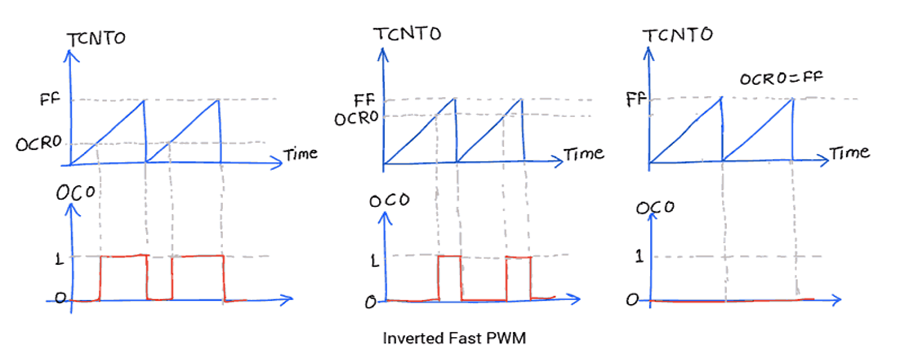
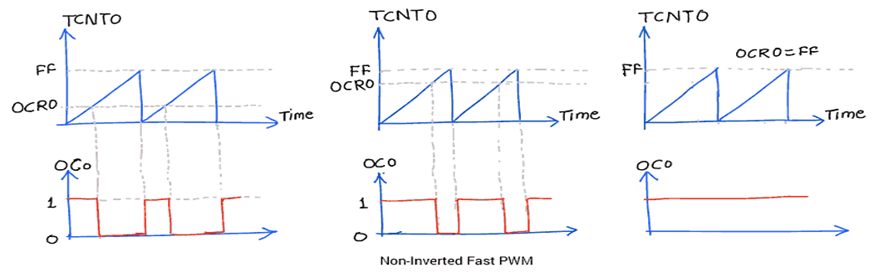
Les signaux modulés en largeur d'impulsion avec différents rapports cycliques sont indiqués ci-dessous



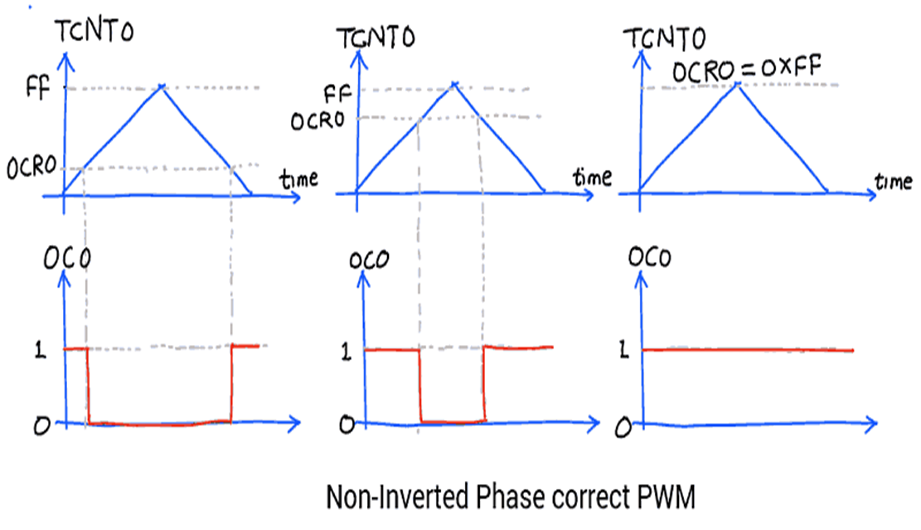
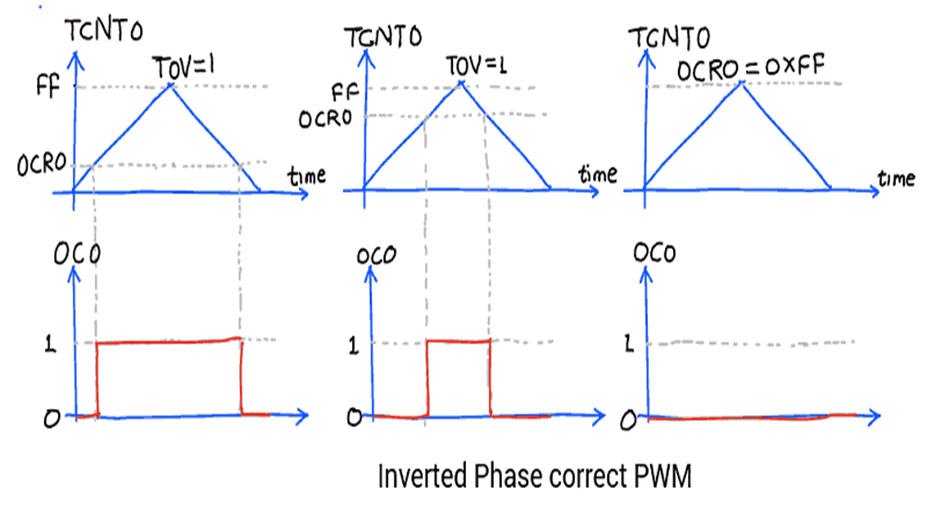
**Formes d'onde du cycle de service PWM :**

ATmega a une unité PWM intégrée. Comme nous le savons, ATmega dispose de 3 temporisations T0, T1 et T2 pouvant être utilisées pour la génération de PWM. Il existe principalement deux modes dans PWM.

1. **PWM rapide**



1. **Phase correcte PWM**



Nous devons configurer le registre du timer pour générer des PWM. La sortie PWM sera générée sur la broche de comparaison de sortie du programmateur (OCx) correspondante.

La sortie PWM est en quelque sorte proche de la sortie analogique. Nous pouvons l'utiliser comme sortie analogique pour générer une onde sinusoïdale, des signaux audio, etc.

Référencées :

1/ Fiche technique ATmega16 et ATmega32

2/ www.arduino.cc

3/ B. Cottenceau : Carte ARDUINO UNO Microcontrôleur ATMega328 - 2016-2017.

4/ http://www.electronicwings.com