

# Laboratoire 3 : Les temps. PWM

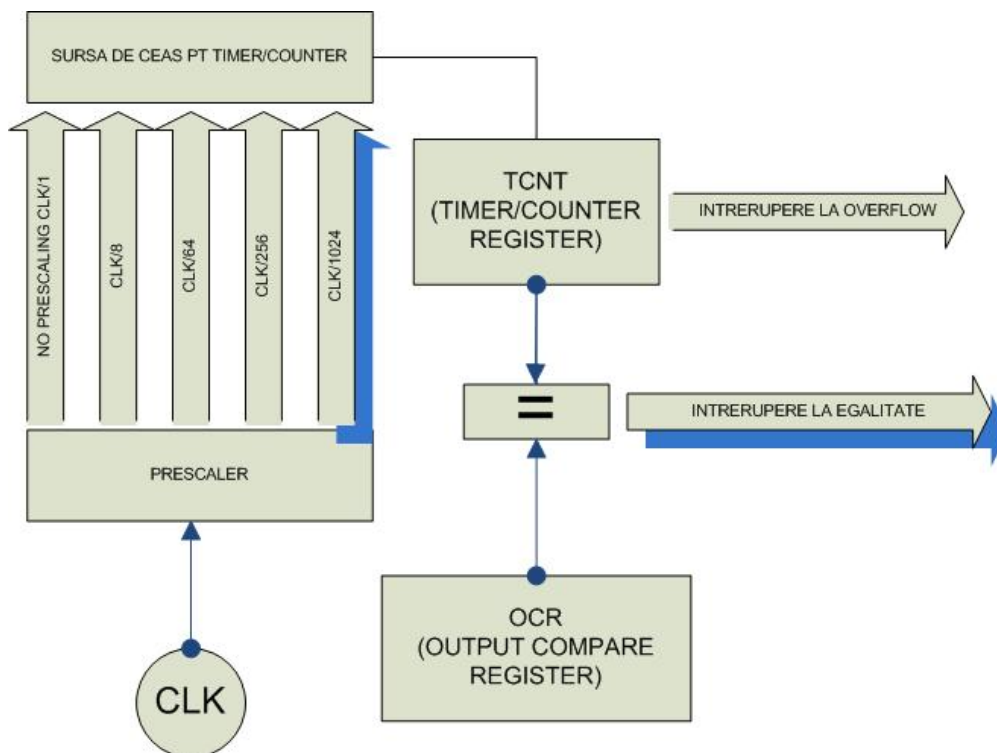
Ce laboratoire vise à vous familiariser avec le travail avec les interruptions matérielles générées par les minuteriers présentes dans le microcontrôleur Atmega328p. Nous utiliserons des minuteriers pour compter les intervalles de temps et générer des signaux périodiques. À l'aide de signaux PWM, nous contrôlerons l'intensité lumineuse d'une LED et la position d'un servomoteur.

## Timers

### 1.1. Le principe de fonctionnement d'un Timer

Le minuteur/compteur offre la possibilité de mesurer des intervalles de temps fixes et de générer des interruptions lorsque l'intervalle mesuré expire. Une minuterie, une fois initialisée, fonctionnera indépendamment de l'unité centrale (UCP). Cela permet de supprimer les boucles de retard du programme principal.

Les composants et le fonctionnement d'une minuterie pour ATmega328p peuvent être largement décrits par les trois unités de la figure ci-dessous :



1. **Timer Counter** (TCNT) - mesure en fait des intervalles de temps et est automatiquement incrémenté d'une fréquence donnée.
2. **Le prescaler** - a pour rôle de diviser la fréquence d'horloge en fonction des besoins de l'application.
3. A chaque incrément de TCNT il y a une **comparaison entre ce registre et une valeur stockée dans le registre OCRn**. Cette valeur peut être chargée par le programmeur en écrivant le registre OCRn. S'il y a égalité, une interruption est générée, sinon l'incrément continue.

Les temporisateurs sont pourvus de plusieurs canaux afin que différents comptages puissent être effectués en parallèle. L'ATmega328p est équipé de 3 minuteriers : deux 8 bits et une 16 bits.

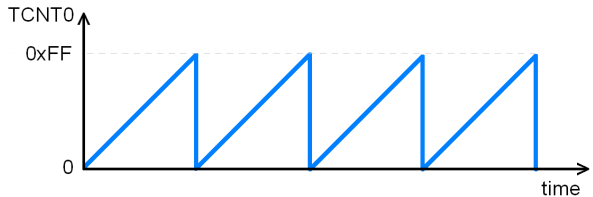
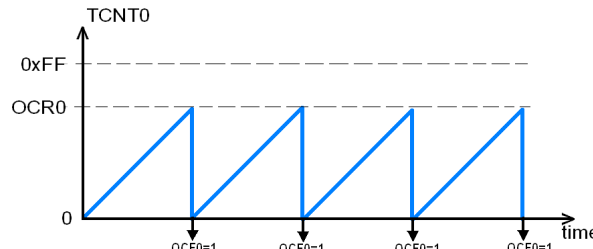
Les temporisateurs peuvent également fonctionner en modes PWM, de sorte qu'ils génèrent un signal de commande de tension variable (moyenne) sur une broche de sortie.

### 1.2. Modes de fonctionnement

Les temporisateurs offrent plusieurs modes de fonctionnement (avec de petites différences entre Timer/Counter0, Timer/Counter1 et Timer/Counter2), qui se différencient par :

- les valeurs jusqu'à laquelle le compteur est incrémenté
- comment il est compté (croissant uniquement, ou alternativement croissant/décroissant)
- l'heure à laquelle le compteur est réinitialisé

Le tableau suivant montre deux modes que nous utiliserons dans ce laboratoire.

Mode	Description	Compteur d'images	Caractéristiques
<b>Normal</b>	commence à partir de 0 compte jusqu'à 0xFFFF		la fréquence est fixée à certaines valeurs prédéfinies données par la fréquence d'horloge et le prédiviseur
<b>Timer d'effacement CTC lors de la comparaison</b>	commence à partir de 0 compte jusqu'à ce qu'un seuil soit atteint ( OCRnApour Timer 0/2, OCR1A ou ICR1 pour Timer 1)		la fréquence est variable, déterminée par la valeur de seuil, la fréquence d'horloge et le prédiviseur

Définitions qui apparaissent dans la fiche technique :

- **BOTTOM** : l'extrémité inférieure de la plage de comptage
- **TOP** : extrémité supérieure de la plage de comptage
- **MAX** : Limite de comptage supérieure, 255 (0xff) pour 8 bits, 65535 (0xffff) pour 16 bits. **TOP** peut être **MAX** dans certains modes de fonctionnement (ex : *Normal* ).

### 1.3. enregistrer

Minuteur	enregistrer	Rôle
<b>Timer0 8 bits</b>	TCNT0	Registre du compteur Timer 0 (celui qui compte)
	TCCR0A, TCCR0B	Registres de contrôle du temporisateur 0 (divers bits pour la configuration du temporisateur)
	OCR0A, OCR0B	Registres de seuil pour le temporisateur 0 (seuil du comptage auquel l'interruption peut être déclenchée, selon la configuration)
	TIMSK0, TIFR0	Registres de bits d'activation de l'interruption du temporisateur 0 / drapeaux d'activation (activation des interruptions)
<b>Timer1 16 bits</b>	TCNT1	Registre du compteur Timer 1 (identique à timer0, uniquement 16 bits)
	TCCR1A TCCR1B TCCR1C	Registres de contrôle de la minuterie 1 (identiques à la minuterie 0)
	OCR1A, OCR1B	Registres de seuil de 16 bits de la minuterie 1 (identiques à la minuterie 0)
	TIMSK1, TIFR1	(identique à timer0)
	ICR1	Registre utilisé pour conserver la valeur du compteur lorsqu'un événement externe se produit sur la broche ICP ou comme seuil pour l'un des modes CTC
<b>Timer2 8 bits</b>	mêmes registres que Timer0	La différence avec le Timer 0 est la possibilité d'utiliser un oscillateur externe séparé pour le Timer 2, sur les broches TOSC1 et TOSC2
	ASSR, GTCCR	Registres liés au fonctionnement asynchrone de ce temporisateur par rapport au reste du système

Les interruptions ne sont activées que si le bit **Iin** SREG est défini (les interruptions sont activées globalement)

## 1.4. Travailler avec la minuterie

### Réglage du mode de fonctionnement

Pour configurer un mode de fonctionnement, nous allons paramétrer :

- les bits WGMdu temporisateur respectif (qui peuvent être trouvés dans les registres TCCRnAde la fiche technique, dans les sections relatives aux temporisateurs, "Description du registre")
- seuil de comptage

Par exemple, pour régler le timer 0 pour qu'il compte jusqu'à 5, nous allons regarder dans la fiche technique au chapitre 14 ( *8-bit Timer/Counter0 with PWM* ) - section *Register Description* → *TCCR0A*

1. on retrouve le mode CTCcomme étant petit 0 1 0à petitWGM2. .0
2. nous constatons que le mode CTCcompte jusqu'àOCRA

En supposant que nous partons d'un registre complètement non initialisé (0 est la valeur par défaut pour la plupart des bits), nous avons le code suivant :

```
TCCR0A |= ( 1 << WGM01 ) ;  
OCR0A = 5 ;
```

### Réglage du prédiviseur

Pour régler le prédiviseur, les bits de type CS. . dans le registre TCCRnBdu temporisateur respectif seront modifiés.

Par exemple, pour régler la valeur du prédiviseur 256 pour le temporisateur 2, nous suivrons dans la fiche technique chapitre 17 ( *8-bit Timer/Counter2 with PWM* ) - section *Register Description* → *TCCR2B*

1. on retrouve le tableau des valeursCS. .
2. on constate que le prescaler 256 correspond bit 1 1 0à bitCS22 CS21 CS20

En supposant que nous partons d'un registre complètement non initialisé (0 est la valeur par défaut pour la plupart des bits), nous avons le code suivant :

```
TCCR2B |= ( 1 << CS22 ) | ( 1 << CS21 ) ;
```

### Configuration des interruptions

```
// exemple de configuration pour le Timer 1 en mode CTC, qui générera des interruptions avec la fréquence de 2Hz  
OCR1A = 31249 ; // compare le registre de correspondance 16 MHz/256/2 Hz - 1  
TCCR1B |= ( 1 << WGM12 ) ; // Mode CTC  
TCCR1B |= ( 1 << CS12 ) ; // 256 prédiviseur
```

### Nous écrivons la routine de gestion des interruptions

Les routines de gestion des interruptions doivent être référencées en mémoire à certaines adresses fixes (dans le vecteur d'interruption). Pour cela on utilise la macro ISR() qui reçoit en paramètre le type d'interruption en cours de traitement.

Par exemple, pour les interruptions générées par le Timer 1 en mode CTC, le code ressemblerait à ceci :

```
// implémente la routine de gestion des interruptions TIMER1_COMPA  
ISR ( TIMER1_COMPA_vect ) {  
    // code d'interruption de Timer1  
}
```

### Nous activons l'interruption et testons le programme

```
// active l'interruption TIMER1_COMPA
TIMSK1 |= ( 1 << OCIE1A ) ;
// active les interruptions globales
sei ( ) ;
```

Pour un temporisateur déjà configuré, pour activer les interruptions, il suffit d'activer le bit correspondant dans TIMSKx

Par exemple, pour le seuil A du temporisateur 1 on écrira :

```
ISR ( TIMER1_COMPA_vect ) {
    // code d'interruption
}

void configure_timer1 ( ) {
    // exemple de configuration pour le Timer 1 en mode CTC
    // qui générera des interruptions avec la fréquence de 2Hz
    TCCR1A = 0 ;
    TCCR1B = 0 ;
    TCNT1 = 0 ;
    OCR1A = 31249 ;           // compare le registre de correspondance 16MHz/256/2Hz-1
    TCCR1B |= ( 1 << WGM12 ) ; // Mode CTC
    TCCR1B |= ( 1 << CS12 ) ;  // 256 prédiviseur
}

void init_timer1 ( ) {
    TIMSK1 |= ( 1 << OCIE1A ) ; // active l'interruption de comparaison de la minuterie
}

void setup ( ) {
    // désactiver les interruptions globales
    cli ( ) ;
    configure_timer1 ( ) ;
    init_timer1 ( ) ;
    // active les interruptions globales
    sei ( ) ;
}

void loop ( ) {
}
```

## 1.5. Calculatrice

En général, pour configurer un Timer, il faut choisir un prescaler et une limite de comptage, en fonction de la période souhaitée et de la fréquence de travail du processeur (ex : 16 MHz pour Arduino UNO) et du mode de fonctionnement choisi. Un exemple de calcul est présenté ci-dessous :

- calcul de la fréquence d'interruption en fonction de la fréquence d'horloge et du seuil de comptage :

$$f_{\text{int}} = f_{\text{clk}} / (\text{prescaler} * (\text{tc} + 1))$$

- calcul du seuil de décompte des temporisateurs pour obtenir la fréquence souhaitée :

$$\text{tc} = f_{\text{clk}} / (\text{prescaler} * f_{\text{int}}) - 1$$

Nous notons qu'une valeur pratique pour le prescaler (parmi celles disponibles, par exemple 8, 64, 256, 1024) et un seuil de comptage (0-255 pour les temporisateurs 8 bits, 0-65535 pour les temporisateurs 16 bits) doivent être choisis afin d'obtenir la fréquence exacte.

Il existe des calculatrices qui peuvent être utiles pour déterminer rapidement les valeurs des registres de configuration d'un Timer :

Calculate ATmega Timer/Counter and prescaler values directly on this page.

## Timer/Counter0 (8 bit) and Timer/Counter1 (16 bit)

Clocks	Prescalers	Registers
$f_{\text{clock}} =$ <input type="text" value="10 M"/> Hz $f_{\text{timer}} =$ <input type="text" value="1 k"/> Hz Factor <input type="text" value="10e3"/> <input type="button" value="Calculate"/>	Prescaler <input type="text" value="1"/> <input type="text" value="8"/> <input type="text" value="64"/> <input type="text" value="256"/> <input type="text" value="1024"/> Decimal <input type="text"/> Hexadec. <input type="text"/> Error [%] <input type="text"/> Green - Timer/Counter0. Blue - Timer/Counter1. Red - Cannot be used.	FOC0 WGM00 COM01 CO COM1A1 COM1A0 COM1B1 COM ICNC1 ICES1 - WG OCIE2 TOIE2 TICIE1 OC

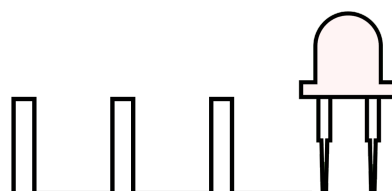
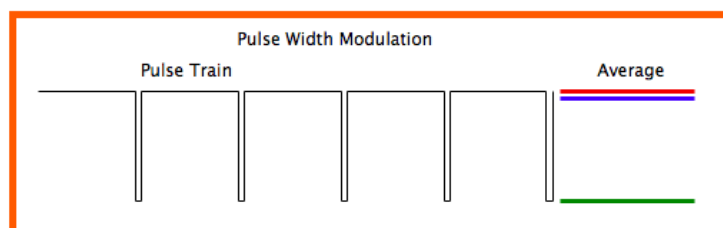
[home](#) | [nicki](#) | [emil](#) | [christian](#) | [the bug](#) | [ihk menu](#) | [www.et06.dk](#) • 2006-2010 | [realm](#)

Le calculateur indique la valeur correspondant à l'intervalle de comptage. Pour configurer la valeur du seuil du temporisateur, une unité est soustraite de cette valeur.

## 2. PWM (modulation de largeur d'impulsion)

PWM ( **Pulse Width Modulation** ) est une technique utilisée pour faire varier de manière contrôlée la tension appliquée à un appareil électronique . Avec seulement deux niveaux de tension disponibles (0V et 5V), on peut approximer d'autres valeurs en basculant très rapidement entre les deux niveaux.

La tension reçue par un appareil représente la moyenne de ces impulsions et est le rapport de la durée correspondant à la valeur ON sur la durée totale d'un cycle ON-OFF. Ce rapport est appelé **rapport cyclique** . Ainsi, les circuits analogiques peuvent être contrôlés à partir du domaine numérique. En gros, cela signifie qu'une LED opérée de cette manière pourra s'allumer/éteindre progressivement, et dans le cas d'un moteur elle tournera plus ou moins vite.

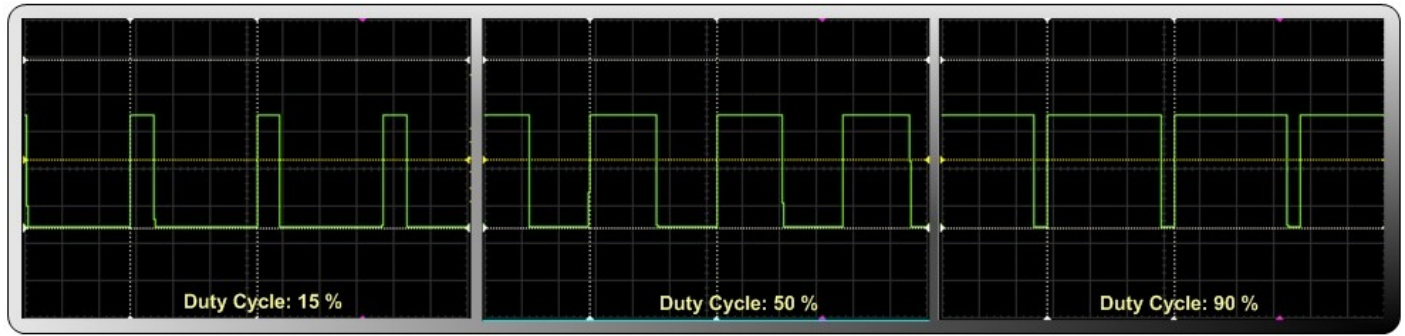


### 2.1. Le principe de fonctionnement

Le facteur de remplissage est exprimé en pourcentage et représente le pourcentage de la période d'un signal au niveau ON. Dans la figure ci-dessous, les signaux PWM avec différents facteurs de remplissage peuvent être vus. Ainsi, on peut très facilement déduire la formule pour obtenir la valeur du facteur de remplissage (D) :

$$D [\%] = \frac{t_{on}}{t_{on} + t_{off}} \cdot 100 = \frac{pulse\_width}{periode} \cdot 100$$

Ainsi, la tension moyenne parvenant au dispositif est donnée par la relation :  $D \cdot V_{cc}$ .



La modulation utilise la variation du facteur de remplissage d'un signal d'onde carrée pour générer une sortie de tension analogique. En considérant une forme d'onde rectangulaire  $f(t)$  avec une valeur minimale  $y_{min}=0$  et une valeur maximale  $y_{max}$  et le facteur de remplissage  $D$  (comme sur la figure), la valeur moyenne de la forme d'onde est donnée par la relation :

$$\bar{y} = D \cdot y_{max}$$

De nombreux circuits numériques peuvent générer des signaux PWM. La plupart des microcontrôleurs offrent cette possibilité, qu'ils mettent en œuvre à l'aide d'un compteur incrémenté périodiquement (connecté directement ou indirectement à une unité d'horloge) et qui est réinitialisé à la fin de chaque période PWM. Lorsque la valeur du compteur est supérieure à la valeur de référence, la sortie PWM (sortie) passe de l'état HIGH à l'état LOW (ou inversement).

Le signal PWM peut également être généré en logiciel par plusieurs méthodes : bit-banging, timer avec ISR. Pratiquement n'importe quelle méthode par laquelle le logiciel peut générer la séquence :

- Définir la broche haute
- Attendez  $T_{on}$
- Définir la broche basse
- Attendre  $T_{off}$

Cependant, dans de nombreuses situations, une fréquence de l'ordre du kHz ou des dizaines de kHz du signal PWM est souhaitée (par exemple, commande de moteur CC ou CC sans balais), et une telle méthode n'est pas la plus efficace, nécessitant l'intervention du processeur pour faire face aux interruptions fréquentes. .

## 2.2. Travailler avec PWM

Travailler avec PWM implique d'initialiser une minuterie et de configurer la sortie sur les broches. Chaque timer a deux broches sur lesquelles il peut générer un tel signal en sortie (les deux canaux) : Timer0 a OC0A, OC0B Timer1 a OC1A et OC1B etc.

Atmega328p dispose de 6 canaux PWM répartis comme suit :

- **Timer0 - OCR0A, OCR0B** respectivement **Timer2 - OCR2A, OCR2B** - un total de 4 canaux 8 bits
- **Timer1 - OCR1A, OCR1B** total 2 canaux 16 bits

Le contrôleur dispose de 6 canaux PWM s'ils sont configurés dans des modes qui ont respectivement les valeurs TOP 0xFF et 0xFFFF. Alternativement, nous pouvons configurer chaque temporisateur (à la fois 8 bits et 16 bits) de manière à avoir comme TOP une valeur spécifiée dans l'un des registres OCRnA ou OCRnB. Une minuterie configurée comme celle-ci n'a qu'un seul canal qui génère une sortie comme on pourrait s'y attendre. Et le canal correspondant au registre contenant la valeur TOP générera un signal, mais pas de la manière souhaitée.

Sur l'Atmega328p il y a 3 types de PWM qui peuvent être générés :

- PWM rapide
- PWM correct de phase
- PWM correct de phase et de fréquence

Nous traiterons principalement du mode Fast PWM qui est utilisé pour la plupart des applications, moins celles où un contrôle précis est nécessaire (ex : moteurs BLDC, audio).

## PWM

Le comptage se fait uniquement sur le front montant du signal d'horloge. En mode Fast PWM, le changement du facteur de remplissage est effectué instantanément, au lieu de cela le signal n'est pas centré (il est déphasé, pratiquement un "glitch" apparaît lorsque le facteur de remplissage change de manière significative). Il existe plusieurs modes Fast PWM proposés par le microcontrôleur. Par exemple, pour la minuterie 1, nous avons :

- PWM 8 bits rapide, avec la valeur TOP = 0x00FF.
- PWM rapide sur 9 bits, avec la valeur TOP = 0x01FF.
- PWM rapide sur 10 bits, avec la valeur TOP = 0x03FF.
- PWM rapide avec valeur TOP en **ICR**
- PWM rapide avec la valeur de TOP dans **OCRnA**

La fréquence du signal PWM dépend du prédiviseur et de la fréquence de l'oscillateur. À partir de la section 15.9.3 (page 102), la formule de calcul de la fréquence en mode Fast PWM 8 bits est :

$$F_{OCnX} = \frac{F_{clk}}{N \cdot (TOP + 1)} = \frac{F_{clk}}{N \cdot 256}$$

Dans ce laboratoire, nous n'utiliserons que le mode Fast PWM. Pour les variantes Fast PWM proposées par les autres timers (0 et 2) et pour les autres modes PWM, voir la fiche technique.

Par exemple, supposons que Timer1 soit réglé sur le mode de fonctionnement Fast PWM (le mode de fonctionnement ne doit pas nécessairement contenir le mot « PWM » pour être utilisé pour générer un signal). Fast PWM se caractérise par une fréquence fixe et un seuil défini par le programmeur, qui peut être modifié pendant l'exécution.

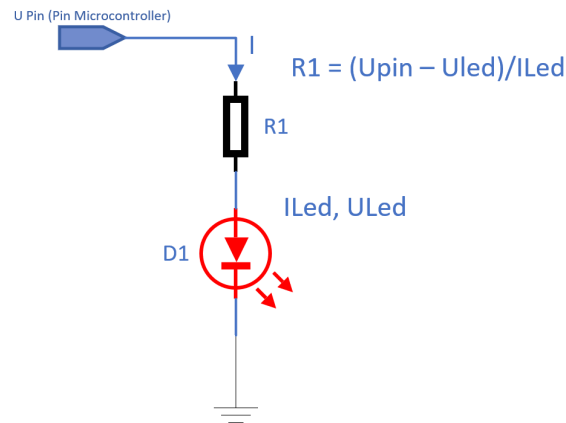
- Le mode 1 0bit COM1A1 COM1A0 laissera le signal sur la broche OC1A à 1 pendant le comptage (jusqu'à ce que le seuil soit atteint) et mettra le signal à 0 à partir du moment où le seuil est atteint jusqu'à la fin d'un cycle (comptage complet jusqu'au TOP).
- Pour obtenir un facteur de remplissage de x%, nous allons définir ici  $OCR1A = x \cdot TOP / 100$  (pour Fast PWM, TOP peut être 0xFF, 0x1FF ou 0x3FF, selon la configuration choisie)

Exemple d'initialisation de Timer1 en mode Fast PWM 8 bits non inverseur, avec Prescaler à la valeur 1024 :

```
CCR1A = 0 ;
TCCR1B = 0 ;
TCNT1 = 0 ;
// Sortie PB1 - OC1A est PB1
DDRB |= ( 1 << PB1 ) ;
//pour le mode Fast PWM 8 bits, les bits WGMn0 et WGMn2 ont la valeur 1
TCCR1A |= ( 1 << WGM10 ) ;
//TCCR1A ne contient que les bits WGM10 et WGM11, WGM12 et WGM13 se trouvent dans TCCR1B
TCCR1B |= ( 1 << WGM12 ) ;
//pour le mode non inverseur, COM1A1 = 1 et COM1A0 = 0
TCCR1A |= ( 1 << COM1A1 ) ;
//pour le Prescaler 1024 on écrit 1 sur CS12 et CS10
TCCR1B |= ( 1 << CS12 ) | ( 1 << CS10 ) ;
//Le seuil auquel le signal est décalé pour atteindre un facteur de remplissage de 0,5
//Puisque dans ce mode TOP est 0xFF, OCR1A sera 50 * 255 / 100 = 127
OCR1A = 127 ; sei( ) ;
```

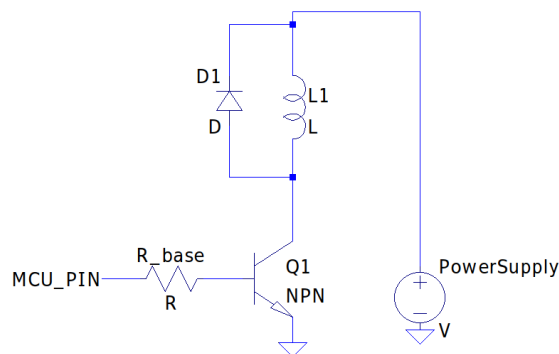
## 3. Applications avec PWM dans Arduino

Pour contrôler une LED à l'aide d'un signal PWM, elle peut être connectée comme on/off, à l'aide d'une résistance de limitation de courant. Si nous avons une LED de puissance (par exemple, des LED automobiles, des LED d'éclairage), un pilote spécialisé est requis.



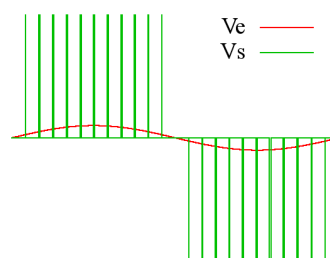
Pour un moteur auquel un signal PWM de rapport cyclique de 0% est appliqué, sa vitesse de rotation sera égale à 0 RPM. Un facteur de remplissage de 100 % entraînera un maximum. Pour de tels cas où nous avons une charge inductive et des courants élevés (relais, moteurs, inducteurs, électrovannes, etc.), il est nécessaire d'utiliser un pilote (élément de commutation/transistor, pilote de moteur, etc.)

Pour de tels cas où nous avons une charge inductive et des courants élevés, une diode flyback ( Wikipedia Flyback diode [[https://en.wikipedia.org/wiki/Flyback\\_diode](https://en.wikipedia.org/wiki/Flyback_diode)] ) est également nécessaire. Le signal PWM généré par le microcontrôleur contrôlera le pilote (par exemple, la base du transistor à travers une résistance) et la diode flyback prendra en charge les pointes de tension générées par le moteur/la charge inductive lorsque le transistor est fermé (ce qui peut sinon conduire à la destruction du transistor).



Une autre utilisation des signaux PWM est de générer un signal analogique avec des applications dans l'amplification audio (classe D), les chargeurs de batterie, etc. En changeant périodiquement le facteur de remplissage, des signaux se rapprochant d'un signal analogique (par exemple une sinusoïde) peuvent être obtenus.

Génération d'un signal sinusoïdal à l'aide de PWM :



Les signaux PWM peuvent également avoir d'autres utilisations, telles que l'envoi de commandes à un actionneur. Un tel exemple est le servomoteur, qui reçoit des commandes déterminées par le rapport cyclique du signal PWM pour contrôler la position. Servocommande [[https://en.wikipedia.org/wiki/Servo\\_control](https://en.wikipedia.org/wiki/Servo_control)]



Ensuite, nous utiliserons la bibliothèque Arduino pour réaliser certaines applications des signaux PWM : contrôler une LED RVB et contrôler la position d'un servo de loisir. Alternativement, pour ceux qui souhaitent approfondir la programmation au niveau des registres, les applications peuvent être réalisées à l'aide de temporisateurs en mode Fast PWM et/ou CTC au lieu des bibliothèques Arduino ( `analogWrite` , `Servo` ) .

## AnalogWrite

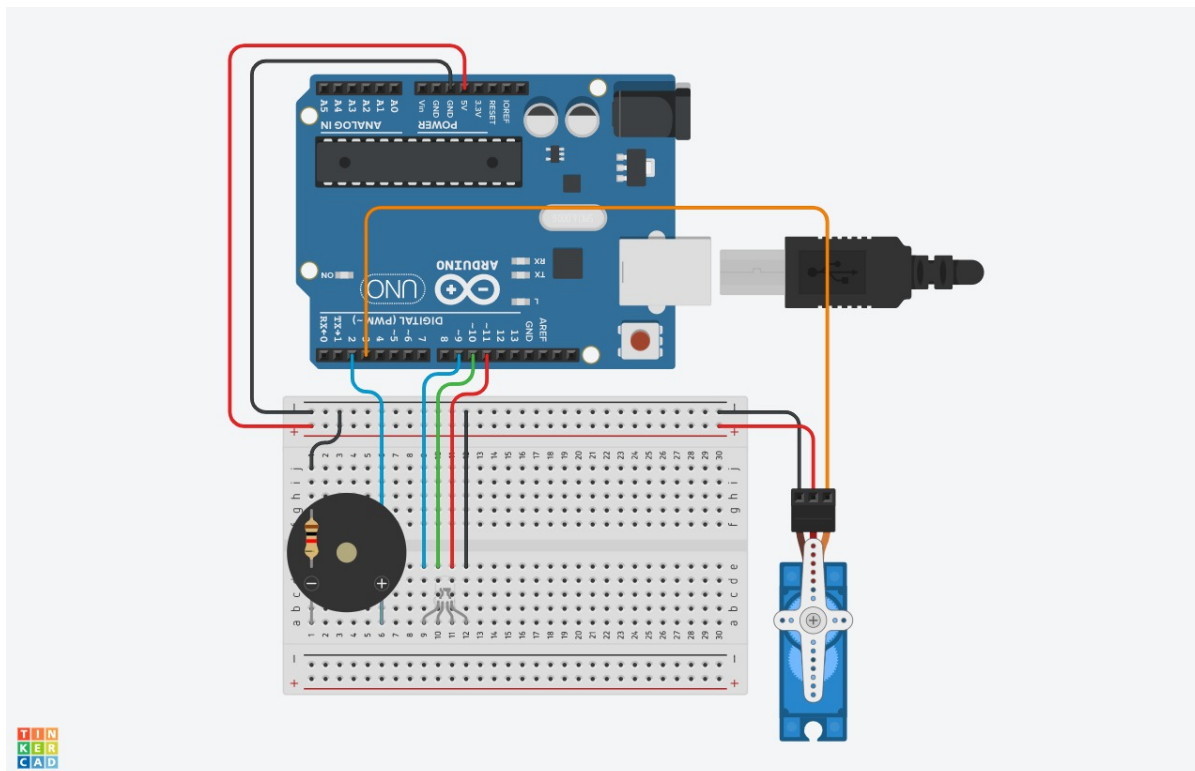
La fonction `analogWrite` d'Arduino configure en fait une minuterie en mode **PWM rapide 8 bits** et peut générer un signal PWM (uniquement) sur les broches associées à l'une des minuteries.

`analogWrite(pin_arduino, value_0_255)` Par exemple, sur l'Arduino/Atmega328p nous avons les broches suivantes qui peuvent générer un signal PWM en utilisant la fonction `analogWrite` :

Broche Arduino	Broche Atmega328p	Sortie minuterie	Fréquence PWM (par défaut)
5	PD5	OC0B (Timer0)	980Hz
6	PD6	OC0A (Timer0)	980Hz
9	PB1	OC1A (Minuterie1)	490Hz
10	PB2	OC1B (Minuterie1)	490Hz
11	PB3	OC2A (Minuterie2)	490Hz
3	PD3	OC2B (Minuterie2)	490Hz

Nous devons peut-être modifier la fréquence du signal PWM, auquel cas nous devons configurer explicitement le Timer (par exemple, prescaler, TOP) à l'aide de registres. `SecretsOfArduinoPWM` [<https://www.arduino.cc/en/pmwiki.php?n=Tutorial/SecretsOfArduinoPWM>]

## 4. Exercices



### 4.1. Tâche 0

À l'aide de la minuterie 1 sur l'Arduino et des opérations d'enregistrement, affichez un message de votre choix sur l'interface série toutes les 5 secondes.

La minuterie de l'Arduino ne peut pas chronométrer de si longues périodes. Vous devrez afficher le message après un certain nombre de périodes, d'une durée choisie par vous. Ex : période de 0,5s (2Hz), on affiche le message à chaque deuxième entrée dans la fonction associée à l'interruption temporisée ; période de 0,1 s (10 Hz), nous affichons le message toutes les 10 entrées.

Choix des paramètres de minuterie pour les paramètres de registre :

```
OCR1A = 31249 ; // compare le registre de correspondance 16MHz/256/2Hz-1 => 2Hz
TCCR1B |= ( 1 << WGM12 ) ; // Mode CTC
TCCR1B |= ( 1 << CS12 ) ; // 256 prédiviseur
TIMSK1 |= ( 1 << OCIE1A ) ; // active l'interruption de comparaison de la minuterie
```

## 4.2. Tache 1

En utilisant le code de l'exercice précédent, toutes les 5 secondes vous alternerez entre deux états de lecture d'une chanson sur le buzzer dans le montage :

- vitesse normale et octave normale
- double vitesse (note Durée/2) et octave supérieur

L'élévation d'une octave est obtenue en doublant la fréquence des notes. Pour jouer une note, vous utiliserez la fonction `tone()` [<https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/>] . Suivez les exemples sur ce référentiel `arduino-songs` [<https://github.com/robsoncoutho/arduino-songs>] et choisissez une chanson.

Faites jouer la chanson dans la fonction `loop()` afin que vous puissiez utiliser les interruptions générées par la minuterie et jouer la chanson en continu.

## 4.3. Tâche 2

À l'aide de la minuterie 0, faites défiler la LED RVB entre plusieurs couleurs et déplacez le servo de 10° toutes les x secondes (vous choisissez une période appropriée pour suivre les changements). N'oubliez pas la suggestion de la première tâche (vous devrez compter un nombre de périodes beaucoup plus court que celui souhaité).

Faites attention aux limites angulaires du servomoteur. Leur dépassement peut endommager le servomoteur. Vérifiez que l'angle ne dépasse pas la plage [0°, 180°] (ou [0, pi] pour les amateurs de trigonométrie 😊).

Dans cette tâche, vous allez afficher différentes couleurs à l'aide de PWM et contrôler un servomoteur :

- Avec la fonction `analogWrite` [<https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/>] , vous définirez le facteur de remplissage PWM pour chaque couleur
- Exemple de commande d'un servomoteur qui oscille dans la plage [0°, 180°]

```
#include <Servo.h>

Servo monservo ; // crée un objet servo pour contrôler un servo
// douze objets servo peuvent être créés sur la plupart des cartes

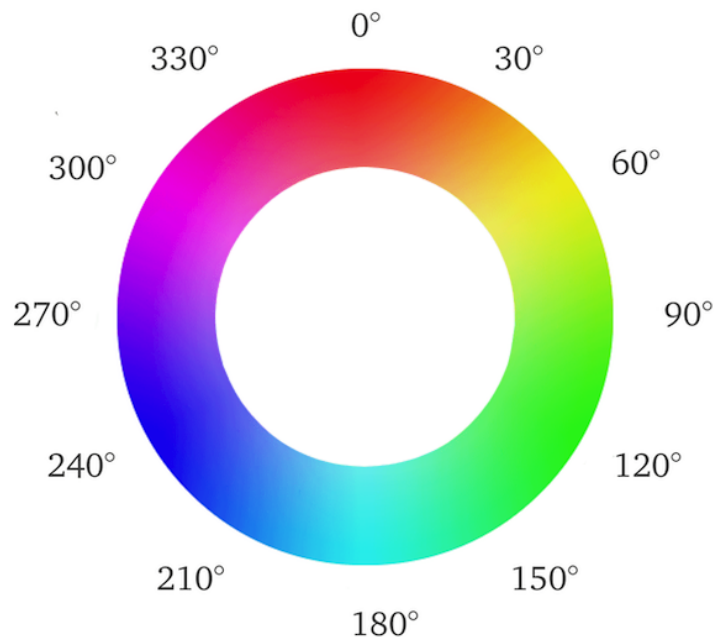
entier pos = 0 ; // variable pour stocker la position du servo

void setup() {
  monservo.attach(3); // attache le servo sur la broche 3 à l'objet servo
  // led de test
  DDRD |= (1 << PD7);
  PORTD &= ~(1 << PD7);
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // va de 0 degrés à 180 degrés
    // par pas de 1 degré
    myservo.write(pos); // indique au servo d'aller à la position dans la variable 'pos'
    delay(15); // attend 15ms que le servo atteigne la position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // passe de 180 degrés à 0 degrés
    myservo.write(pos); // indique au servo d'aller à la position dans la variable 'pos'
    delay(15); // attend 15ms que le servo atteigne la position
  }
}
```

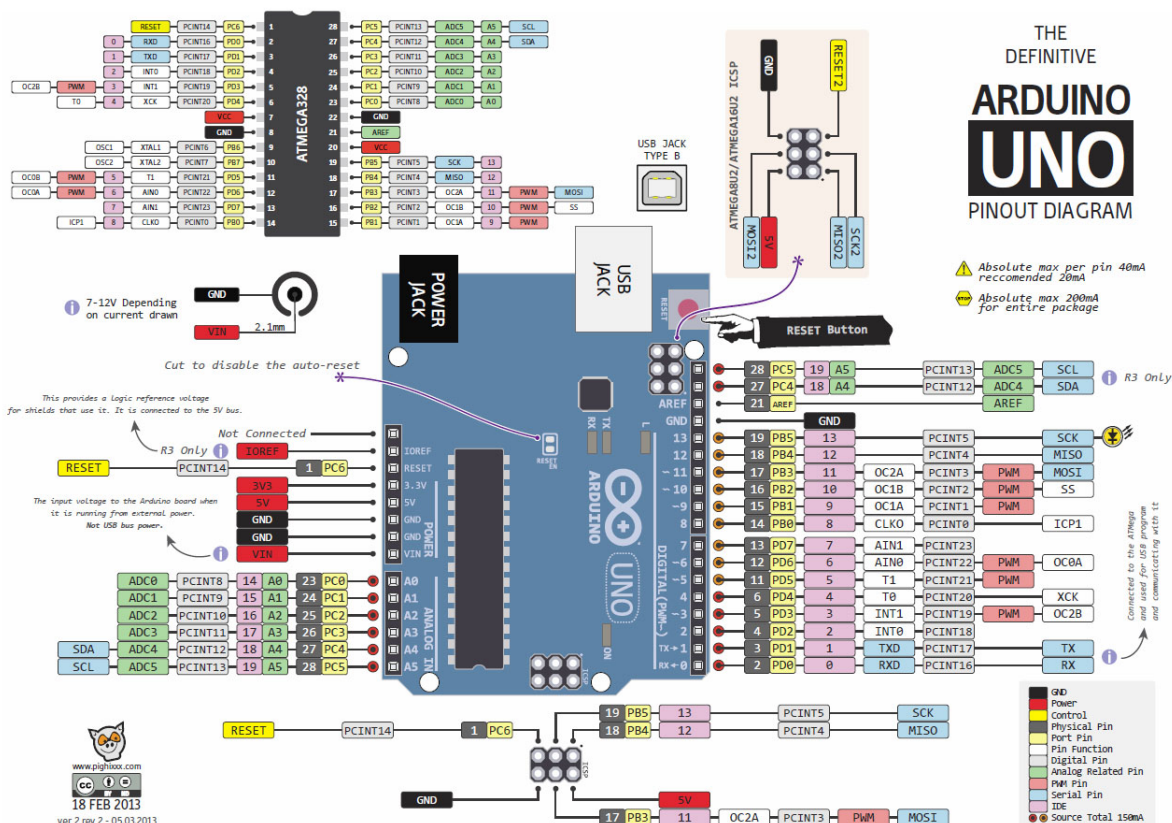
Bonus : modifiez le programme à l'aide de la fonction `setLedColorHSV()`

- La fonction `setLedColorHSV` permet de changer la couleur en utilisant la représentation alternative HSV (Hue Saturation Value), ce qui facilite ensuite la modification de la couleur, de la saturation et de l'intensité lumineuse. `setLedColorHSV`
- Réglez les valeurs de saturation (s) et d'intensité (v) sur 1 et changez la couleur (h) sur 0-360
- Pour suivre la correspondance entre les 2 représentations (RGB et HSV) il existe des sélecteurs de couleurs comme le sélecteur de couleurs en ligne [<https://colorpicker.me/#3237b9>]



## 5. Ressources

- Fiche technique Atmega 328p
- Brochage Arduino UNO



Responsable:

- Alexandre Predescu [mailto:alexandru.predescu@upb.ro]
- Ene de Drago [mailto:enedragos99@gmail.com]
- Narcis Caroi [mailto:narciscaroi24@gmail.com]

## 6. Liens utiles

---

- AVR Libc - interruption.h [[http://www.nongnu.org/avr-libc/user-manual/group\\_\\_avr\\_\\_interrupts.html](http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html)]
- anti-rebond [<https://www.arduino.cc/en/Tutorial/BuiltInExamples/Debounce>]
- SecretsOfArduinoPWM [<https://www.arduino.cc/en/pmwiki.php?n=Tutorial/SecretsOfArduinoPWM>]
- Servocommande [[https://en.wikipedia.org/wiki/Servo\\_control](https://en.wikipedia.org/wiki/Servo_control)]

pm/lab/lab3-2023.txt · Dernière modification : 2023/03/27 11:26 par narcis\_florin.caroi