

# PWM : MODULATION DE LARGEUR D'IMPULS

[PIERRE-YVES ROCHAT](#), EPFL

Document en cours de relecture, version du 2015/07/19

## VARIER L'INTENSITÉ D'UNE LED

Beaucoup d'enseignes à LED se contentent d'allumer et d'éteindre des LED ou des groupes de LED, mais il y en a beaucoup plus intéressants et variés en ayant la possibilité de changer leurs intensités lumineuses.

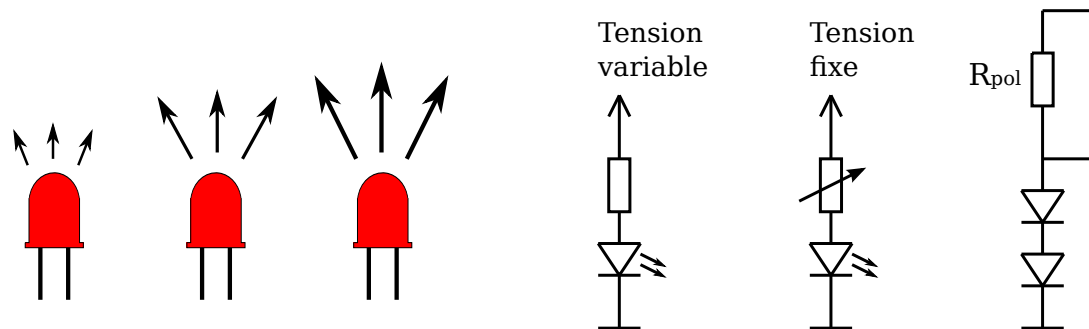


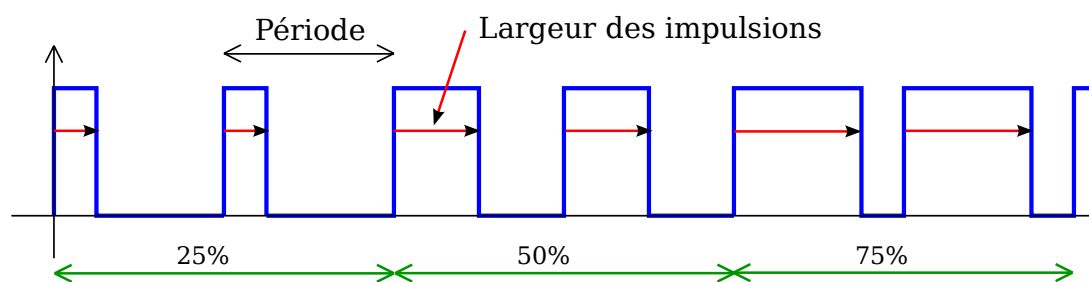
Figure : Varier l'intensité d'une LED

Comment faire varier l'intensité d'une LED ? On sait que c'est le courant qui la traverse qui produit la lumière. Faire varier le courant n'est généralement pas facile. On peut faire varier la tension qui alimente la LED, ou en faire varier le courant. Il est aussi possible de piloter le courant par un transistor, avec un montage adéquat.

Mais il existe une solution très différente et généralement beaucoup plus simple à implémenter. Soit une LED. Que se passe-t-il lorsqu'on augmente la fréquence du clignotement ? À partir d'une certaine fréquence, l'œil ne voit plus qu'une lumière continue. En augmentant encore la fréquence, il voit simplement la lumière de plus en plus faible que si elle était allumée en permanence.

## LE PWM

À l'idée de faire clignoter rapidement la LED, ajoutons l'idée de faire varier le temps pendant lequel la LED est allumée. Cette technique est appelée la Modulation de Largeur d'Impulsion (MLI) ou Pulse Width Modulation (PWM) en anglais. Regardons



*Figure : signal PWM*

Le signal a une période constante (donc une fréquence constante). La durée de la partie active du signal du tracé, 25% de la puissance disponible peut être envoyée à la charge, vu que le signal est à 1 pendant 1/4 de la période et à 0 le reste du temps. De même, la puissance passe à 50% au milieu du tracé et à 75% dans la dernière partie. On peut aussi choisir d'envoyer aucune puissance (0%) en laissant la sortie à 0, ou toute la puissance (100%) en laissant la sortie à 1.

Le rapport entre la durée de la partie active du signal et la durée de la période s'appelle le **rapport**

## FRÉQUENCE DU PWM

L'usage du PWM est très répandu, avec des applications dans beaucoup de domaines. La commande vent par PWM. Dans ce cas, c'est la nature inductive du moteur qui effectue l'intégration du signal, système.

Dans le cas de la commande de diodes lumineuses, le temps d'allumage et d'extinction est très court, jusqu'à plusieurs dizaines de mégahertz. C'est l'œil humain qui ne voit pas le clignotement. L'œil a une fréquence maximale située vers 75 Hz. C'est bien lui qui effectue l'intégration du signal pour en percevoir une valeur moyenne.

Dans divers domaines de l'électronique, les fréquences des signaux PWM peuvent aller couramment jusqu'à 100 kHz. Mais plus la fréquence est élevée, plus les pertes électriques à l'instant des changements de direction peuvent conduire à une dissipation importante d'énergie dans les éléments de commutation.

L'œil a une fréquence limite de perception du clignotement. Par exemple, on sait qu'un tube fluorescent peut être commandé par du courant alternatif à 50 Hz, et que chaque période a une alternance positive et négative. Les cellules sensibles de notre œil (les cônes pour la vision en couleur et les bâtonnets pour la vision nocturne) n'ont pas la même limite de perception du clignotement.

Pour les enseignes et afficheurs à LED, on vise généralement des fréquences de l'ordre de 100 à 200 Hz. Il est donc relativement difficile d'envoyer à ces fréquences toutes les informations à l'ensemble des LED, qui peuvent être très nombreuses. Par exemple, les murs de LED capables d'afficher de la vidéo nécessitent des circuits électroniques complexes. Nous n'en discuterons pas dans ce cours.

## PROGRAMMATION D'UN PWM

Voici un programme très simple qui génère un signal PWM sur une sortie d'un microcontrôleur :

```

1  #define LedOn digitalWrite(P1_0, 1)
2  #define LedOff digitalWrite(P1_0, 0)
3
4  uint16_t pwmLed; // valeur du PWM, 0 à 100
5
6  void setup() { // Initialisations
7      pinMode(P1_0, OUTPUT); // LED en sortie
8      pwmLed = 25; // valeur du PWM.
9  }
10
11 void loop() { // Boucle infinie, durée 10ms => un cycle du PWM à 100 Hz
12     LedOn;
13     delayMicrosecond(100*pwmLed); // durée de l'impulsion
14     LedOff;
15     delayMicrosecond(100*(100-pwmLed)); // solde de la période
16 }
```

Le programme a été écrit de telle manière que les valeurs du PWM doivent être choisies entre 0 et 100, ce qui correspond à des *pour cents*. Les informaticiens choisissent plus souvent des valeurs dans une plage binaire (8 bits, voir l'exemple suivant).

La fréquence a été choisie ici à 100 Hz. En effet les deux délais de la boucle principale totalisent 100 ms.

Après les initialisations de la sortie et de la variable qui contient en permanence la valeur du PWM, le programme active et la partie inactive du signal. Les attentes sont obtenues par des délais exprimés en us.

Ce programme donne l'impression que la diode lumineuse est à demi intensité, malgré une commande de 128. La réponse de l'œil n'est pas linéaire, mais logarithmique. On remarque aussi que le PWM est visible en continu devant l'œil.

## GÉNÉRATION DE PLUSIEURS SIGNAUX PWM

Le principe du programme que nous venons de voir ne convient pas à la programmation de PWM pour plusieurs sorties. Une manière de programmer un PWM qui se prête à gérer plusieurs sorties :

```

1  uint8_t pwmLed; // valeur du PWM, 0 à 255 (8 bits)
2  uint8_t cptPwm; // compteur du PWM
3
4  void setup() { // Initialisations
5      pinMode(P1_0, OUTPUT); // LED en sortie
6      pwmLed = 64; // valeur du PWM. Elle est ici fixe, mais pourrait changer
7                  // à tout moment en complétant le programme.
8      cptPwm = 0; // compteur du PWM
9  }
10
11 void loop() { // Boucle infinie, durée 39us (256 * 39us = ~10ms)
12     if ((cptPwm==0) && (pwmLed>0)) LedOn; // seulement pour une valeur p
13     if (cptPwm==pwmLed) LedOff;
14
15     cptPwm++; // passe automatiquement de 255 à 0 (overflow)
16     delayMicroseconds(39);
17 }
```

Dans ce cas, la boucle principale dure seulement le temps qui correspond à une fraction de la période du PWM. L'usage de cette valeur, associée à un compteur de type uint8\_t (8 bits non signés), simplifie le retour à zéro, qui s'effectue au moment du dépassement de capacité (overflow).

Voici comment modifier ce programme pour qu'il commande 8 LED, avec 8 valeurs différentes de PWM.

```

1  uint8_t pwmLed[8]; // valeurs des PWM, 0 à 255 (8 bits), pour 8 LED
2  uint8_t cptPwm; // compteur du PWM
3
4  void setup() ...
5
6  void LedOn (uint8_t n) ... procédure qui allume une des 8 LED
7  void LedOff (uint8_t n) ... procédure qui éteint une des 8 LED
8
9  void loop() { // Boucle infinie
10     for (uint8_t i=0; i<8; i++) {
11         if ((cptPwm==0) && (pwmLed[i]>0)) LedOn(i); // allume la LED conce
12         if (cptPwm==pwmLed[i]) LedOff(i); // éteint la LED correspondante
13     }
14
15     cptPwm++; // passe automatiquement de 255 à 0 (overflow)
```

```

16 | delayMicrosecond(39); // on pourrait diminuer cette valeur
17 | // pour tenir compte du temps d'exécution de la boucle for.
18 | }

```

## PROGRAMMATION D'UNE SÉQUENCE EN PWM

Dans des applications comme la gestion des moteurs d'un robot mobile, les valeurs du PWM sont calculées en fonction des valeurs des capteurs (capteurs de distance, caméras, etc.)

Dans le cas des enseignes et afficheurs à LED, les valeurs sont calculées en fonction du temps qui passe. Comment connaître le temps qui s'écoule ? Une des manières consiste simplement à compter les cycles du PWM à fréquence constante.

Prenons un exemple. Pour donner l'impression qu'un appareil est en mode "repos", une LED va avoir une intensité qui varie de la manière suivante :

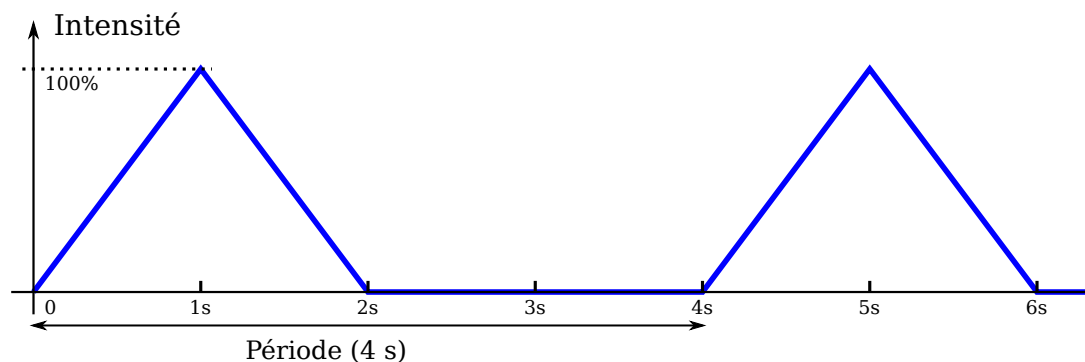


Figure : diagramme des temps de la LED suggérant le repos

La séquence est cyclique et dure 4 s. Il imite la respiration d'une personne qui dort. Durant la première seconde, elle augmente jusqu'au maximum. Durant la deuxième seconde, elle diminue jusqu'à zéro, valeur qui reste jusqu'à la fin de la période.

La courbe se divise en 3 parties. Il est facile d'exprimer l'intensité en fonction du temps dans chaque partie. On va donc compléter le programme :

```

1 | uint16_t pwmLed; // valeur du PWM, 0 à 255 (8 bits, dans une variable 16 bits)
2 | uint16_t cpt10ms = 0; // compteur des cycles, de 0 à 400 (par 10ms, total 4s)
3 | ...
4 | void loop() { // Boucle infinie, durée 39us (256 * 39us = ~10ms)
5 |     if (cptPwm==0) {
6 |         cpt10ms++;
7 |         if (cpt10ms<100) { //première seconde
8 |             pwmLed = cpt10ms * 256 / 100; // droite montante

```

```

9      } else if (cpt10ms<200) { // deuxième seconde
10      pwmLed = 256 - ((cpt10ms-100) * 256 / 100); // droite descendante
11      } else {
12      pwmLed = 0;
13      if (cpt10ms==400) cpt10ms = 0; // fin des 4 s
14      }
15  }
16
17  if ((cptPwm==0) && (pwmLed>0)) LedOn; // seulement si valeur non nulle
18  if (cptPwm==pwmLed) LedOff;
19
20  cptPwm++; // passe automatiquement de 255 à 0 (overflow)
21  delayMicroseconds(39);
22 }

```

Il est important de contrôler que les variables utilisées ne produisent pas de dépassement de capacité. La variable `pwmLed` a été choisie ici sur 16 bits. Pour la première droite, `cpt10ms` peut aller jusqu'à 999, nombre inférieur à 65'535 (qui est le nombre maximum que peut représenter un nombre entier de 16 bits) pour tous les cas.

## CONVERTISSEUR DAC SIMPLE

Bien que le signal PWM soit un signal binaire (il est soit à 0 soit à 1 à un instant donné), il est porteur d'information. On peut en effet faire l'intégrale du signal pour s'en rendre compte. Il faut pondérer le 1 logique comme la valeur -1, comme le montre la figure suivante :

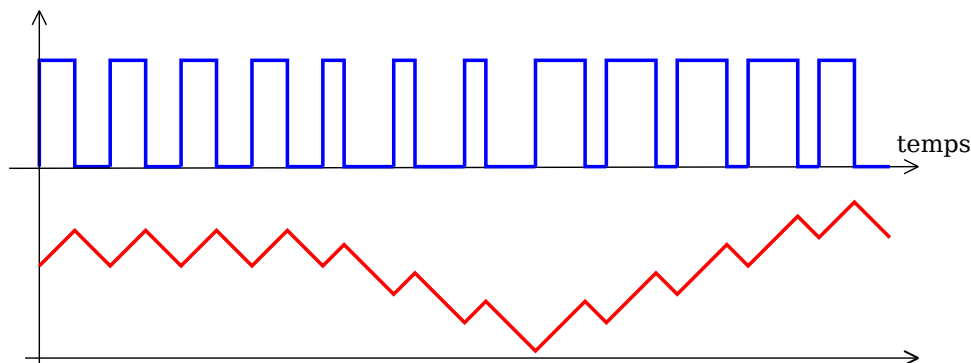


Figure : signal PWM intégré

Il est donc possible de réaliser un convertisseur numérique-analogique (Digital to Analog Converter). Ce convertisseur doit être suivi par un filtre passe-bas adapté à la fréquence du PWM. Ce filtre peut être un simple filtre RC.

## CIRCUITS SPÉCIALISÉS

Comment générer un signal en PWM avec des circuits logiques ? Il faut utiliser un compteur, piloté par une horloge fixe. Un détecteur de 0 permet de signaler le début du cycle. Un comparateur, connecté à un registre, permet de détecter la fin de l'impulsion. Une bascule *Set-Reset* génère le signal.

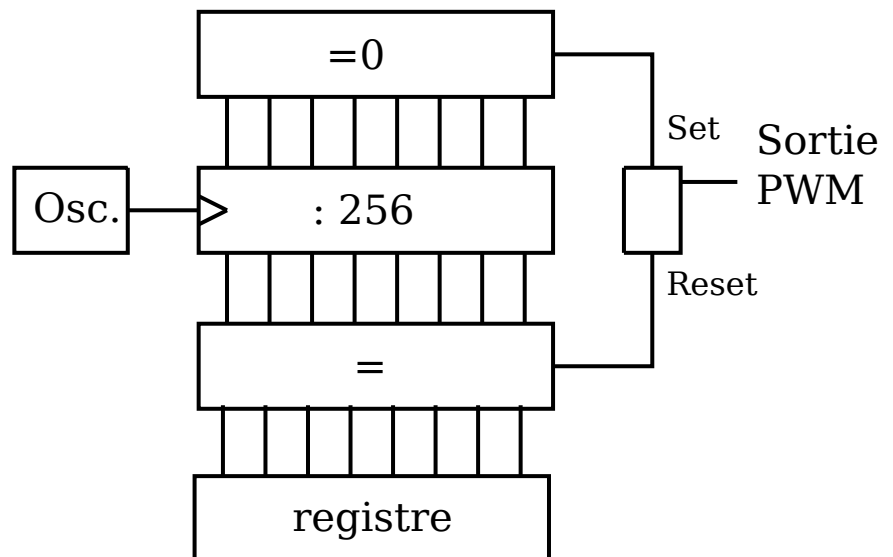


Figure : génération d'un PWM avec un système logique

Les microcontrôleurs contiennent des circuits logiques supplémentaires ressemblant beaucoup à des *Timers*. Complétés de circuits logiques dédiés, ils peuvent générer du PWM sans autre programmation. Nous en parlerons plus tard dans ce cours.