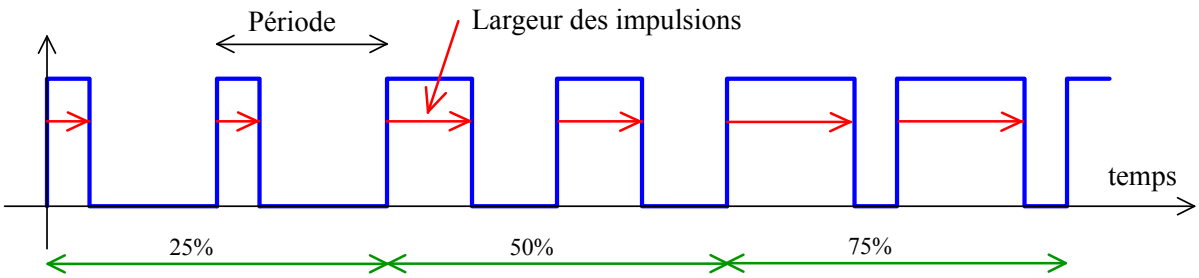
BCM : LA MODULATION CODÉE BINAIRE

Pierre-Yves Rochat, EPFL

rév 2016/05/14

PWM SUR UNE MATRICE

La technique classique pour faire varier l'intensité d'une LED est la modulation de largeur d'impulsion *PWM*. Le signal se présente de la manière suivante :



Signal PWM

Dans un afficheur matriciel, l'intensité de chaque LED doit pouvoir être indépendante. Avec la plupart des schémas utilisés pour un afficheur matriciel, il est nécessaire de renvoyer la valeur de toutes les LED pour changer l'état d'une seule LED. C'est par exemple le cas des afficheurs basés sur des registres série-parallèles.

Si la fréquence du PWM est F_{pwm} et le nombre de valeurs possibles de l'intensité est N_{intens}, la fréquence de rafraîchissement sera de $F_{raf} = F_{pwm} \times N_{intens}$

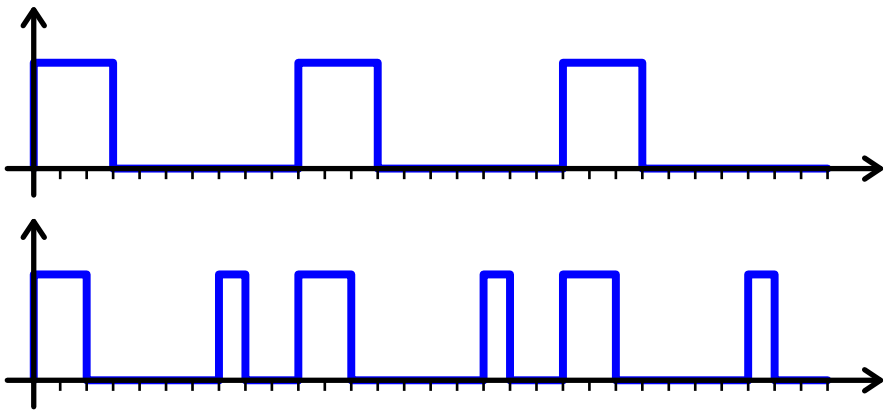
Par exemple, pour une fréquence du PWM de 100 Hz et une intensité sur 8 bits, produisant donc 256 intensités différentes, on devra rafraîchir la matrice à une fréquence de

100 × 256 = 25’600 Hz, ce qui correspond à une période de 39 µs. Si l’afficheur comporte un nombre important de LED, il n’est pas facile de parvenir à ce résultat !

Or la fréquence minimale d’une afficheur est donnée par les propriétés de l’œil humain. Diminuer la fréquence du PWM de dessous de 100 Hz dégradera la qualité visuelle de l’afficheur. Il n’y a pas de marge de manœuvre de ce côté.

Tolérance sur la forme

Par contre, la forme du signal qui parvient à une LED n’a pas beaucoup d’importance. Ce qui va déterminer l’intensité perçue par l’œil, c’est le rapport entre le temps pendant lequel la LED est allumée durant un cycle et le temps total du cycle. Les deux signaux de la figure ci-dessous donnent le même résultat :



Même intensité pour deux signaux différents

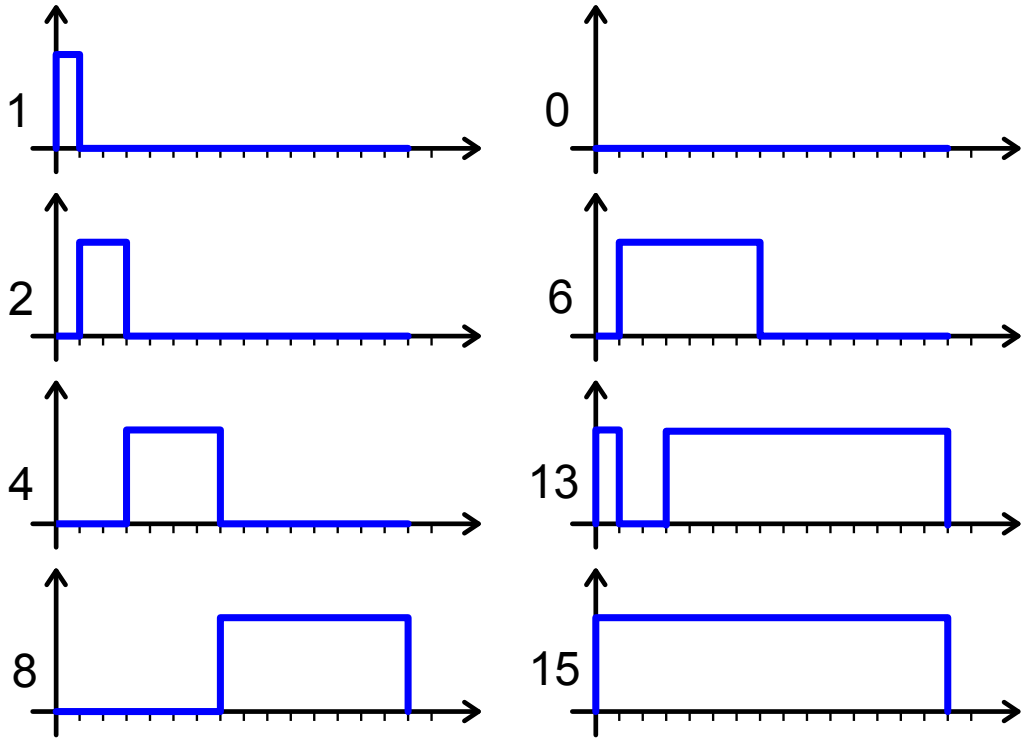
Principe du BCM

Comment produire simplement un signal dont le total du temps actif est un nombre de fois le temps unitaire ? On peut utiliser les poids binaires du nombre. C’est l’idée de la **Modulation Codée Binaire** *Binary Coded Modulation* ou BCM.

Il s’agit de découper la période du signal en tranches dont les durées sont les poids binaires. Penons par exemple un signal dont l’intensité est donnée par un nombre de 4 bits, produisant 16 intensités possibles. Les tranches du BCM auront les valeur 1, 2, 4 et 8, divisant ainsi la période du signal en 15 tranches de durée égale : 1 + 2+ 4 + 8 = 15 = 2^4 - 1.

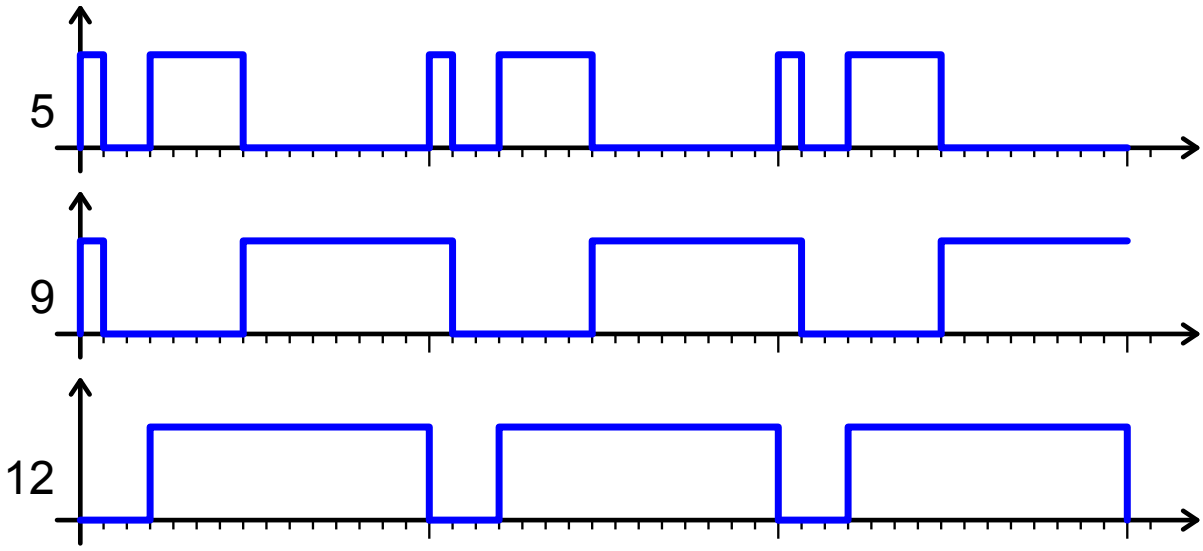
Pour chaque valeur d'intensité, on active la LED chaque fois que le poids binaire concerné est à 1 dans le nombre.

Voici une période du signal, à gauche pour des intensités correspondant aux poids binaires, à droite pour d'autres valeur composées d'un nombre différent de bits :



Principe du BCM

Voici encore les signaux pour trois autres valeurs, représentés sur plusieurs périodes :

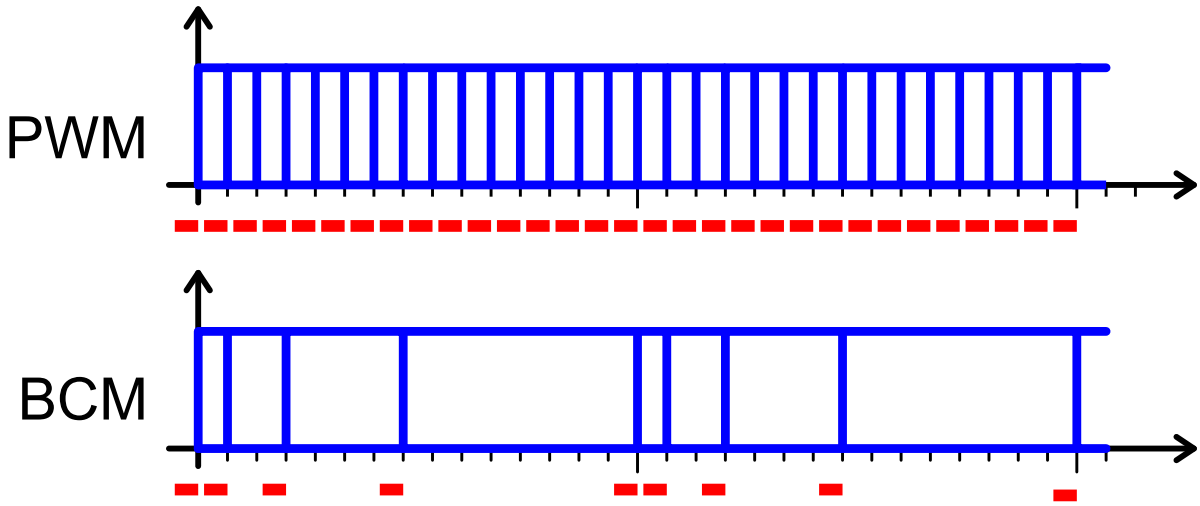


Exemples de signaux BCM sur 3 périodes

Avantages et limites du BCM

La production de signaux BCM est surtout intéressante lorsque le nombre de LED à commander est important. Au lieu d'avoir 2^b rafraîchissements à faire à chaque cycle, il suffit de b rafraîchissements (b est le nombre de bits de l'intensité).

Mais on remarquera que le temps minimum entre deux rafraîchissements n'a pas changé par rapport au PWM. Ce temps correspond à la durée du bit de poids faible. Il reste le point critique pour l'affichage sur une matrice de LED. Par contre, la durée du bit de poids fort est égal à la moitié de la période. Souvent, on pourra utiliser cette durée pour calculer les valeurs des intensités de chaque LES en vue du prochain cycle.



Chargement des valeurs pour chaque bit

Programmation de signaux BCM

Voici un exemple de programme qui pilote 8 LED avec chacune une intensité de 8 bits :

```
#define BITS_BCM 8

void Attente(uint16_t duree) {
  volatile uint16_t i;
  for (i=0; i<(duree*64); i++) {
  }
}
uint8_t Intens[8] = {0, 0, 0, 0, 0, 0, 128, 0};
uint8_t n, b;
uint8_t t = 0;

int main() {
  WDTCTL = WDTPW+WDTHOLD; // stoppe le WatchDog
  BCSCTL1 = CALBC1_16MHZ; DCOCTL = CALDCO_16MHZ;
  P1DIR = 0xFF; // toutes les broches de P1 en sortie

  while (1) { // Boucle infinie :
    for (n=0; n<BITS_BCM; n++) { // pour une période du BCM
      for (b=0; b<8; b++) { // pour chaque bit de sortie
        if (Intens[b] & (1<<n)) P1OUT|=(1<<b); else P1OUT&=~(1<<b);
      }
      Attente(1<<n);
    }
    // ...calcul des prochaines valeurs des intensités
  }
}
```

La première boucle for correspond à une période des signaux. Elle contient une attente dont la durée progresse selon les puissances de 2 : `Attente (1<<n);`

Il faut noter que cette manière de programmer la durée de chaque bits est imprécise, vu qu'elle ne tient pas compte du temps d'exécution des autres instructions du programme. On peut observer un certaine irrégularité de la progression des intensités, par exemple au passage de 127 à 128. Une programmation en utilisant des Timer et des interruptions résoudra ce problème.