RUBANS DE LED

Pierre-Yves Rochat, EPFL

rév 2016/07/17

Plusieurs sortes de rubans de LED

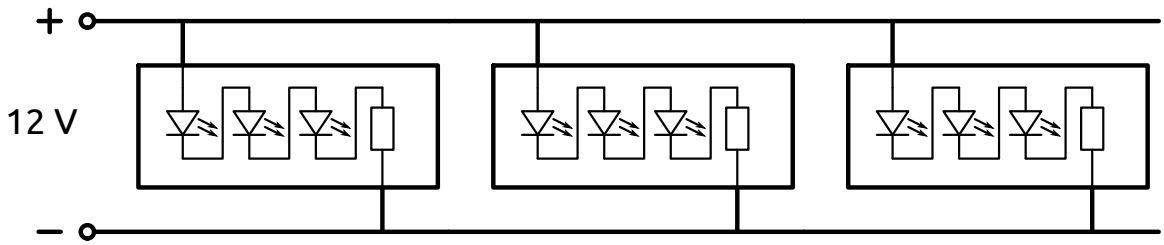
Les rubans de LED, en anglais *LED strips*, sont des LED disposées en ligne. Elles sont reliées entre elles par un circuit imprimé flexible, ou simplement par des fils. On trouve sur le marché plusieurs types de rubans de LED. On distingue principalement les rubans **uniformes** et les rubans **adressables**.

Les rubans uniformes peuvent être d'une seule couleur fixe. Dans ce cas, seule l'intensité peut être modifiée, pour toutes les LED du ruban en même temps, par une commande avec un signal PWM. Les rubans peuvent aussi être multicolores (RGB). Dans ce cas, tout le ruban peut changer de couleur en même temps. La commande se fait par un triple PWM, un pour chaque couleur.

Basés sur une technologie très différente, il existe aussi des rubans adressables (*Addressable LED Strips*). Chaque pixel du ruban peut alors prendre une couleur indépendante des autres pixels à un instant donné.

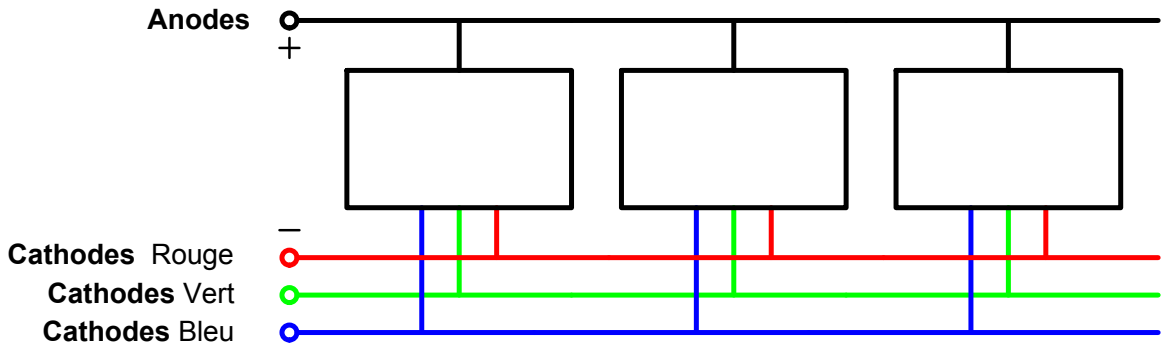
Les rubans uniformes

Les rubans uniformes ne contiennent que des LED avec leurs résistances de limitation. Généralement, l'alimentation est de 12 V, ce qui signifie que plusieurs LED sont mises en série :



Rubans uniformes

Pour des rubans RGB, le câblage est généralement à anode commune, mais pas toujours ! Quatre fils sont utilisés, un pour les anodes de toutes les LED et trois pour les cathodes des LED rouges, vertes et bleues :



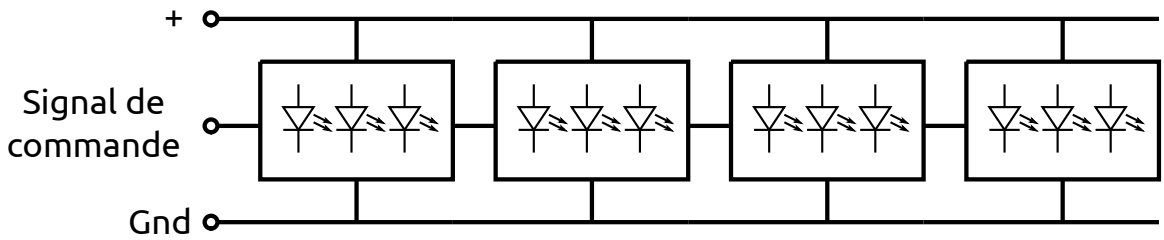
Câblage des rubans uniformes à anode commune

LES RUBANS ADRESSABLES

Plusieurs solutions sont techniquement possibles pour faire varier l'intensité de chaque pixel d'un ruban. Celles qui minimisent le nombre de fils utilisés sont les plus intéressantes.

Il faut forcément deux fils pour alimenter les LED, le *Gnd* et le *Vcc*. Est-ce possible d'utiliser un seul fil pour apporter à chaque pixel l'information qui le concerne ?

Une solution très souvent utilisée présente l'architecture suivante :

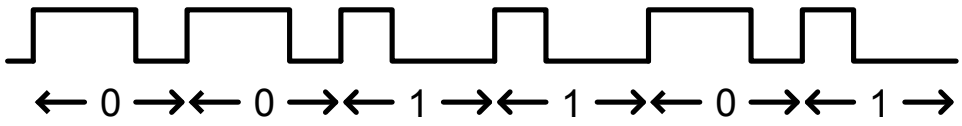


Rubans adressables à 3 fils

Alors que les deux fils d'alimentation relient tous les modules de pixels, le troisième fil relie la sortie d'un module vers l'entrée d'un autre.

Le protocole des WS28xx

Les données vont être alors transmises en série. Mais il devient nécessaire de regrouper sur un seul fil les données et l'horloge. Le principe est celui d'une horloge asymétrique :



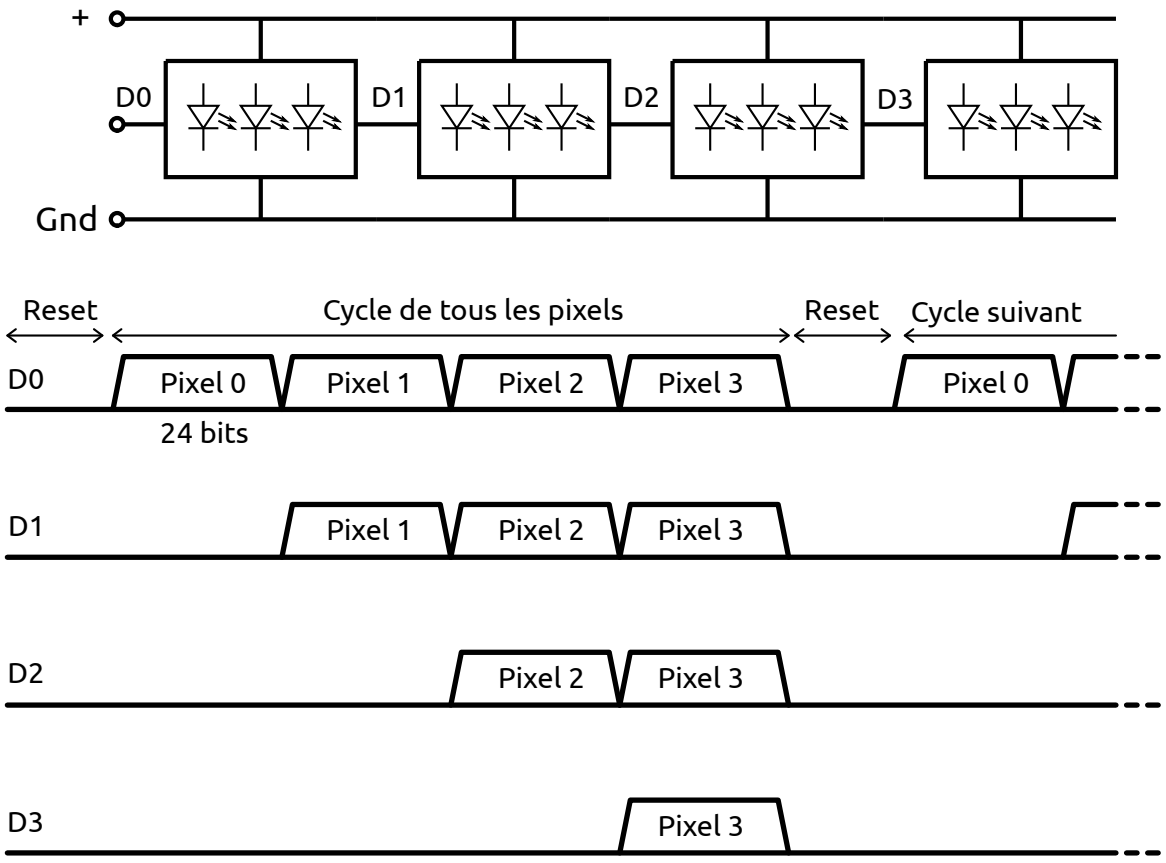
Horloge asymétrique

Chaque bit est transmis par un cycle d'horloge. Lorsque la durée de la partie haute du signal est plus longue que la durée de la partie basse, le bit transmis est un *0*. À l'inverse, un *1* est transmis lorsque la partie basse du signal est plus longue que sa partie haute.

Le fabricant chinois WorldSemi propose une famille de circuits utilisant ce principe. Le WS2811 pilote les trois LED : une rouge, une verte et une bleu. Souvent, les trois LED sont encapsulées dans le même boîtier. On parle alors d'une LED RGB (*Red, Green, Blue*). Chaque LED est commandée par un signal PWM de 8 bits. Il est donc nécessaire d'envoyer 24 bits pour chaque pixel.

Pour synchroniser le début de l'envoi d'une nouvelle série de valeurs à tous les pixels du ruban, une attente d'au moins 50 µs est nécessaire. Chaque circuit est alors prêt à recevoir 24 bits. L'astuce utilisée est alors la suivante : chaque circuit ne transmet à sa sortie les informations qui se présentent à son entrée qu'après avoir enregistré les 24 premiers bits qui succèdent au *reset*.

La figure suivante explique ce principe, illustré pour un ruban de 4 LED :



Transmission des bits aux registres successifs

Signaux rapides

Les documents techniques de WorldSemi semblent indiquer des valeurs différentes pour le *timing* des signaux. Pour le WS2811S, le temps haut du *0* et le temps bas du *1* doivent être de 0.5 µs, le temps bas du *0* et le temps haut du *1* doivent être de 2 µs. Une tolérance pour ces deux valeurs est donnée à 150 ns. Le temps de la pause du *reset* doit être supérieur à 50 µs.

Ces contraintes temporelles rendent difficile sa programmation avec un AVR ou un MSP430, dont les processeurs ont fréquences de l'ordre de 16 MHz (125 ns par cycle). Des solutions ont toutefois été trouvées, soit en programmant en assembleur, soit encore en utilisant de manière astucieuse le circuit de communication série.

Avec un processeur ARM fonctionnant avec une horloge de fréquence plus élevée, c'est plus facile de respecter ces exigences temporelles. Nous allons ici montrer un programme écrit pour un STM32 de STmicro, testé sur une carte Nucleo. Voici la partie avec les définitions :

```
// Signal One Wire pour WS2811
#define PORT_WS2811 GPIOA
#define BIT_WS2811 10
#define WS28On (PORT_WS2811->ODR|=(1<<BIT_WS2811))
#define WS28Off (PORT_WS2811->ODR&=~(1<<BIT_WS2811))
```

La mise à *1* et à *0* du signal se fait avec des *bit set*, que le compilateur va transcrire dans l'instruction assembleur correspondante, dont l'exécution est très rapide. La production des signaux pour le *0* et pour le *1* se font par l'exécution successive de ces instructions :

```
#define Un WS28On;WS28On;WS28On;WS28On;WS28On;WS28On;WS28On;WS28Off;
#define Zero WS28On;WS28Off;WS28Off;WS28Off;WS28Off;WS28Off;WS28Off;WS28Off

#define UnCourt WS28On;WS28On;WS28On;WS28On;WS28On;WS28On;WS28Off;
#define ZeroCourt WS28On;WS28Off;WS28Off;WS28Off;WS28Off;WS28Off;WS28Off;
```

Le temps de ces successions de *bit set* et de *bit clear*, additionné au temps de l'instruction de test de la boucle d'envoi des bits, va correspondre aux spécifications du fabricant. Leur nombre a été déterminé expérimentalement, avec un oscilloscope. Les versions *`UnCourt`* et *`ZeroCourt`* sont volontairement plus courtes, pour pouvoir compléter leur durée par une instruction.

Un tableau va contenir les valeurs des intensités des trois LED de chaque pixel. Les 24 bits vont être mis dans un entier de 32 bits, pour optimiser l'accès au tableau :

```
// Contenu du ruban :
#define LgRuban 50
uint32_t Ruban[LgRuban];
```

Le programme principal initialise les périphériques par l'intermédiaire de la librairie HAL (*Hardware Abstraction Layer*). Des valeurs sont mises dans la variable *`Rubans`* pour avoir des couleurs différentes sur les LED.

La boucle principale du programme envoie, pour chaque pixels, les 24 bits. Lors du 23e bit, le pointeur est incrémenté. L'usage des valeurs courtes de l'envoi des bits est utilisée pour compenser le temps de l'incrémentation. Lors du 24e bit, les versions courtes sont aussi utilisées, cette fois pour tenir compte du saut pour le retour au début de la boucle.

```
int main(void) { // Programme principal
  HAL_Init(); // Initialisation de la librairie Hardware Level
  SystemClock_Config(); // Configure l'horloge système
  MX_GPIO_Init(); // Initialise les périphériques
  PORT_WS2811->MODER |= (0b01 << (BIT_WS2811*2)); // broche en sortie

  uint32_t i;
```

```
volatile uint16_t j;
uint32_t v;
uint32_t idx;
uint32_t *pt; // pointeur dans le tableau

// Initialisation fixe des couleurs
for (idx=0; idx<LgRuban; idx++) {
  Ruban[idx] = 1<<idx;
}

while (1) { // boucle principale
  pt = Ruban;
  __ASM volatile ("cpsid i"); // interrupt OFF

  for (idx=0; idx<LgRuban; idx++) {
    v = *pt;
    if (v & (1<<23)) {Un;} else {Zero;}
    if (v & (1<<22)) {Un;} else {Zero;}
    if (v & (1<<21)) {Un;} else {Zero;}
    if (v & (1<<20)) {Un;} else {Zero;}
    if (v & (1<<19)) {Un;} else {Zero;}
    if (v & (1<<18)) {Un;} else {Zero;}
    if (v & (1<<17)) {Un;} else {Zero;}
    if (v & (1<<16)) {Un;} else {Zero;}
    if (v & (1<<15)) {Un;} else {Zero;}
    if (v & (1<<14)) {Un;} else {Zero;}
    if (v & (1<<13)) {Un;} else {Zero;}
    if (v & (1<<12)) {Un;} else {Zero;}
    if (v & (1<<11)) {Un;} else {Zero;}
    if (v & (1<<10)) {Un;} else {Zero;}
    if (v & (1<<9)) {Un;} else {Zero;}
    if (v & (1<<8)) {Un;} else {Zero;}
    if (v & (1<<7)) {Un;} else {Zero;}
    if (v & (1<<6)) {Un;} else {Zero;}
    if (v & (1<<5)) {Un;} else {Zero;}
    if (v & (1<<4)) {Un;} else {Zero;}
    if (v & (1<<3)) {Un;} else {Zero;}
    if (v & (1<<2)) {Un;} else {Zero;}
    if (v & (1<<1)) {UnCourt;} else {ZeroCourt;}
    pt++;
    if (v & (1<<0)) {UnCourt;} else {ZeroCourt;}
  }

  __ASM volatile ("cpsie i"); // interrupt ON

  for (j=0; j<500; j++) {
```

```
    }
  }
}
```

Toute interruption qui pourrait se produire durant cette boucle perturberait les temps de l'envoi des bits. Une instruction va donc désactiver les interruptions au début du cycle de l'envoi des pixels à tout le ruban, puis les activer à nouveau dans la période qui correspond au *reset* des WS2811.

Il est possible de créer des animations sur les LED. Dans l'exemple suivant, une variable *temps* va compter le temps qui s'écoule et comptant les cycles de rafraîchissement du ruban. En fonction du temps, les couleurs des LED peuvent être modifiées :

```
    temps++; // comptage du temps

    // Clignotement des LED 0 et 30 :
    if (temps==500) {
        Ruban[30] = Ruban[0] = 0xFFFFFF;
    }
    if (temps==1000) {
        temps=0; Ruban[30] = Ruban[0] = 0;
    }
    // Changement progressif de la couleur de la LED 47 :
    Ruban[47]++;
```

La génération de tels signaux pourrait aussi se faire avec la technique du DMA (*Direct Memory Access*).