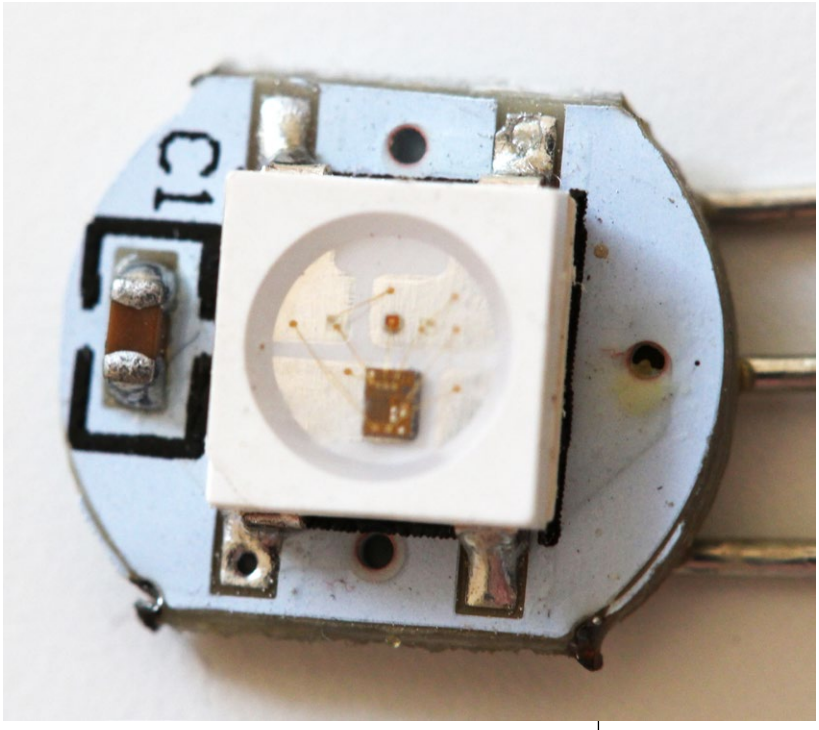
WS2812 : LA LED INTELLIGENTE

Denis Bodor

Elles sont belles, elles sont excitantes, elles sont captivantes... Et maintenant elles sont aussi intelligentes. Je parle bien entendu des leds qui illuminent nos vies et qui sont un facteur multiplicateur de l'intérêt pour n'importe quel projet. Mais qu'est-ce qu'une led intelligente après tout ? Imaginez simplement que vous puissiez à l'aide d'un seul signal ordonner à une led de prendre une couleur et une intensité donnée : voici la nouvelle génération de leds. Démonstration avec la WS2812.



La fameuse WS2812B telle que vendue sur circuit imprimé et accompagnée d'un condensateur. On voit ici clairement le circuit intégré dans le composant ainsi que les connexions internes aux leds rouges, vertes et bleues.

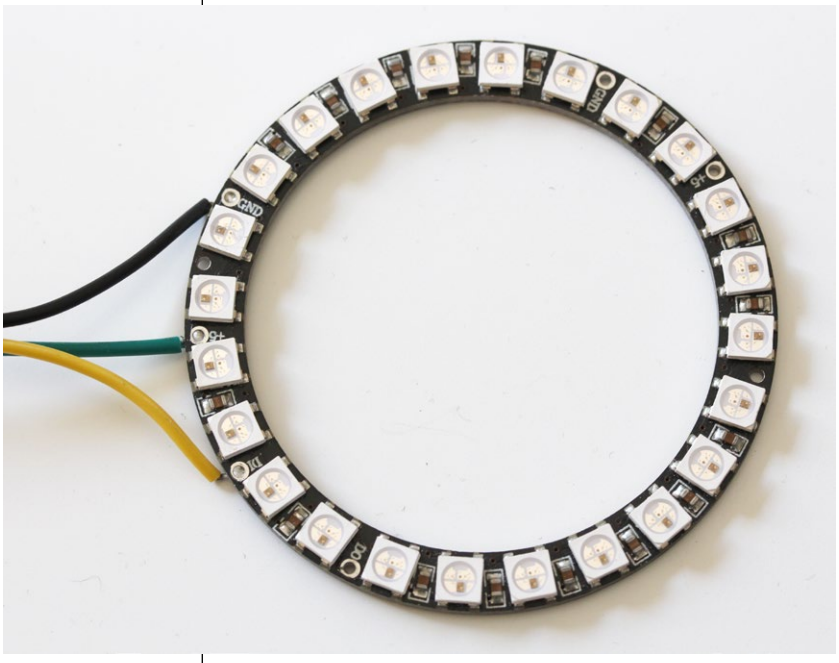
Les leds multicolores ne sont pas une nouveauté. D'abord bicolores, vertes et rouges et donc équipées de trois pattes, celles-ci permettaient déjà d'obtenir plusieurs couleurs. Vert, rouge et vert+rouge = jaune dans un premier temps puis, en jouant avec la PWM, toute une gamme de teintes entre vert et rouge. Il a fallu cependant attendre l'arrivée des leds bleues pour compléter la palette et pouvoir enfin obtenir toute la gamme de couleurs par synthèse additive. Ce procédé, consistant à combiner les lumières de plusieurs intensités de couleurs pour obtenir une teinte précise, est inverse de la synthèse soustractive qu'on utilise, par exemple, en peinture ou en impression. Ici, cyan, magenta et jaune sont mélangés et combinés pour obtenir une gamme complète. En imprimerie comme en peinture, on ajoute généralement le noir pour former un mélange CMJN (ou CMYK en anglais pour *Cyan Magenta Yellow Key*). Notez au passage que les éléments rouges dans ce magazine ne proviennent pas d'un pigment rouge, pas plus que le vert. C'est un mélange de magenta et de jaune, et un mélange de cyan et de jaune. Le noir est nécessaire, car très difficile à obtenir à partir des trois autres couleurs (arriver à un noir acceptable avec de la gouache par exemple peut vous occuper tout un week-end).

Le plus amusant dans tout cela, et pour la synthèse additive en particulier, est le fait que ceci n'est possible qu'en raison de la façon dont nous, humain, percevons les couleurs. Le fait d'obtenir du jaune en combinant des lumières vertes et rouges n'est pas une propriété physique, mais un « défaut » dans le fonctionnement de notre rétine. Celle-ci, en effet, est composée de cônes qui sont des récepteurs réagissant à une longueur d'onde donnée. Une lumière jaune possède une longueur d'onde entre 573 nm et 584 nm (nanomètres). Plus courte c'est du vert, plus grande c'est de l'orange, puis du rouge. Lorsque ce rayonnement rencontre notre rétine, il excite plusieurs types de cônes de façon bien précise. Nous n'avons pas de cône pour le jaune, mais uniquement des types B (437 nm, bleu), des types V (533 nm, vert) et R (564 nm, rouge).

Et là, cela devient très amusant, car si on utilise judicieusement un mélange en bonne quantité de chaque lumière, on arrive à faire croire à la rétine que c'est effectivement une lumière de couleur équivalente qui lui arrive. Autrement dit, en ajustant les quantités de rouge, de vert et de bleu, on peut faire croire à l'œil qu'il voit du jaune, de l'orange, du rose ou encore du violet. Pour le rouge et le vert qui se combinent, on ne voit donc pas de jaune, mais l'œil réagit de la même manière : on a la sensation de voir du jaune.

Vous trouvez ça fou ? Pourtant ce n'est pas nouveau et surtout c'est très largement utilisé partout autour de nous. Votre téléviseur, votre écran de PC,

votre smartphone, le projecteur dans le salon... Tous ces appareils vous trompent, car aucun n'émet réellement un rayonnement entre 573 nm et 584 nm. Méfiez-vous ! Ils essayent tous de vous faire croire que vous voyez du jaune alors que ce n'est que du vert et du rouge !



Un clone chinois du NeoPixel Ring 24 d'AdaFruit Industries. Nous avons ici 24 leds WS2812B chaînées sur un circuit imprimé circulaire. Inutile de se creuser la tête bien longtemps pour trouver des idées de projets avec ce type de modules. L'anneau existe également en versions 12, 16 et 60 leds.

Quoi qu'il en soit, le fait de pouvoir obtenir une gamme complète de teintes à partir de ces trois couleurs n'est pas réservé aux appareils vendus dans le commerce. Les leds RVB (en anglais RGB pour *Red Green Blue*) sont composées de trois blocs de semi-conducteurs émettant une lumière d'une longueur d'onde différente lorsqu'un courant les traverse. La led en question comporte donc quatre pattes, une commune (cathode ou anode selon le type de led) et une par couleur. En utilisant la technique de modulation en largeur d'impulsion (PWM), on arrive à jouer sur l'intensité de chaque lumière en ajustant les rapports cycliques et donc à produire la couleur de son choix, même si le blanc est parfois difficile à obtenir en fonction du positionnement des blocs dans la led.

Le problème ici est bien entendu le fait de devoir gérer trois signaux PWM par led (ou le multiplexage) et donc de faire appel soit à un microcontrôleur soit à un circuit intégré dédié. Et voici qu'arrive la nouveauté et une très bonne idée : intégrer directement ce circuit de gestion dans la led elle-même et donc créer une led intelligente (*smart led*) qui reçoit des instructions et se charge de presque tout le travail.

1. WS2812B, UN PEU PLUS QU'UN UNIQUE MODÈLE

Parmi les différents modèles de smart leds existant, il en est un dont le nom revient souvent : WS2812. On retrouve cette dénomination dans presque toutes les descriptions de produits équivalents sur eBay par exemple et ce, quelle que soit la forme de l'objet : simples leds, leds sur support (*breakout board*), barrettes de leds, rubans de leds adressables, matrices, etc.

Mais en réalité, les choses sont un peu plus complexes. Avant tout, faisons un point sur quelle désignation correspond à quoi :

- WS2811 n'est pas une led, mais un circuit intégré de gestion de leds ;
- WS2801 est aussi une puce, mais contrairement à la WS2811, celle-ci utilise une liaison avec deux fils DIN et SCK, les données et le signal d'horloge ;
- WS2812 est une led RVB intégrant une puce proche de la WS2811, pilotable avec un seul signal (pas de CLK), mais à une fréquence de 800 KHz et non 400 Khz comme la WS2811 ;

- et enfin, la WS2812B qui est également une led avec une puce intégrée, mais qui est une évolution compatible de la WS2812 ajoutant diverses améliorations et protections. Vous l'avez compris, ce modèle est précisément celui généralement préconisé et traité ici.

Tous ces éléments sont fabriqués par la société WorldSemi basée à Guangdong en Chine et le site du constructeur en référence encore bien d'autres modèles. Ce a quoi il faut ajouter les incontournables leds et puces compatibles, dont celles de BaiCheng proposant un format de led différent (5mm et 8mm) sous le nom PL9823-F8, pilotables avec le même protocole que les WS2812 et WS2812B. Un autre exemple, plus « évolué » est la APA102©, mais utilisant un protocole différent puisqu'il s'agit de SPI (avec un signal d'horloge supplémentaire donc). Ce modèle sort du cadre de cet article, car se pilotant de manière totalement différente.

Les WS2812B se présentent sous la forme d'un composant à souder en surface (SMD) de 5mm de côté (format communément appelé 5050 PLCC4). Elles sont généralement distribuées par des boutiques en ligne, des sites d'achats directs (type Alibaba) ou sur eBay. Il ne sera pas forcément nécessaire de vous diriger vers un vendeur en Asie et donc d'attendre un temps non négligeable pour la livraison. Certains distributeurs situés en Allemagne ou en Angleterre commercialisent des packs de 100 WS2812B pour moins de 30 euros, port gratuit (kt-elektronic). Un petit conseil au passage, la led WS2812B se différencie de la WS2812, plus fragile, par son nombre de connecteurs, 4 pour le B contre 6 pour l'ancienne WS2812. Vérifiez bien le descriptif du produit et les photos, les erreurs et tromperies sur la marchandise ne sont pas courantes, mais le risque est réel.

Cependant, à moins de savoir précisément ce que vous faites, je vous recommande de tourner votre attention non pas vers les composants bruts (*bulk*), mais vers leurs déclinaisons montées sur un petit circuit imprimé ou moulées dans un ruban sécable.

Les WS2812B sont des composants fragiles et à traiter avec soin, et le modèle WS2812 est encore plus sensible, une simple inversion de polarité suffit à le détruire. Ainsi, à la différence des composants vendus seuls, les versions montées sont accompagnées d'un condensateur de découplage et d'une résistance de protection qui sont plus que recommandés (en plus d'utiliser le support comme dissipateur thermique). Tout cela en restant de taille réduite et dans un prix acceptable. J'en ai trouvé sur eBay aux tarifs de 200 pour 70 euros, 100 pour 40 euros, 30 pour 15 euros, 10 pour 5 euros... auxquels s'ajoute une dizaine d'euros de port (depuis l'Angleterre, vendeur *transnova*).

Notez que les WS2812 et WS2812B sont souvent désignés sous le nom « NeoPixel » qui est utilisé et popularisé principalement par le site AdaFruit proposant

Gros plan sur une partie du NeoPixel Ring. On distingue un des condensateurs de découplage à proximité d'une des leds. Il faudra cependant toujours ajouter un autre condensateur entre la masse et Vcc au niveau de l'alimentation. Précisons qu'il est hors de question d'alimenter 24 WS2812B avec une carte Arduino.

ses propres déclinaisons sous forme de rubans, d'anneaux (12, 16, 24, 60 leds), de shields Arduino, etc., s'accompagnant d'une bibliothèque NeoPixel et d'un guide en ligne. Les tarifs cependant sont loin d'égaler les offres des sites d'enchères, mais le niveau de qualité de la fabrication n'est pas le même. L'alignement des WS2812B sur un anneau de 24 leds à 15 euros provenant d'eBay, par exemple, est assez aléatoire par rapport aux produits similaires chez AdaFruit.

2. COMMENT COMMUNIQUER AVEC UN SEUL FIL ?

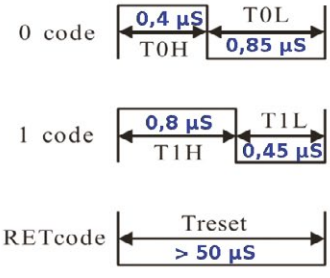
Les WS2812B et donc les circuits, rubans et barrettes qu'elles équipent disposent d'uniquement quatre connecteurs : l'alimentation en +5V, la masse, les données en entrée (DIN) et les données en sortie (DOUT). Il n'y a donc pas de signal d'horloge permettant de valider l'arrivée d'un bit. En prenant le cas d'une seule led, un connecteur suffit donc à transmettre au circuit intégré les informations sur l'intensité lumineuse des trois leds, rouge, verte et bleu, placées dans le composant.

Comme pour les registres à décalage et d'autres circuits logiques similaires, il est possible de chaîner les WS2812B en reliant le DOUT du premier composant au DIN du suivant, etc. Ainsi, en envoyant la bonne quantité de bits, il est possible de définir la couleur et l'intensité de tous les WS2812B de la chaîne. La quantité de composants qu'il est ainsi possible de chaîner n'est pas spécifiée dans la documentation, très concise et vaguement anglaise. Ce n'est pas un problème logique, mais physique qui limite la quantité, par la distance même que doit parcourir l'information. Ce à quoi s'ajoutent, bien entendu, les problèmes d'alimentation puisque chaque composant comprend trois leds de forte puissance. Il est très facile de dépasser le watt, voire la dizaine de watts (~100 leds, ce qui semble beaucoup, mais finalement n'est qu'un carré de 10×10). Enfin, il faut prendre en compte les paramètres temporels. Comme l'envoi de l'information est basé sur la durée des impulsions, le temps nécessaire pour envoyer les 24 bits à chaque led est fixe. Entre 1,2 µS et 1,3 µS par bit et donc une trentaine de microsecondes par led, ce à quoi il faut ajouter le temps de calcul nécessaire côté Arduino. Si vous comptez générer une animation sur *n* leds, ces délais impacteront la fréquence de rafraîchissement, car il faut que les informations aient le temps d'arriver à la dernière led avant de commencer un nouveau cycle.

La documentation du composant détaille comment une telle communication sur une broche peut être réalisée. Celle-ci parle de NZR qu'on pourrait interpréter comme étant une faute de frappe

Extrait amélioré de la documentation de Worldsemi à propos de leur WS2812B. On comprend ici clairement qu'envoyer 1 bit prend 1,25 µS et chaque led utilise 24 bits, ce qui donne un temps de 30 µS par composant.

Data transfer time(TH+TL=1.25µs±600ns)



T0H	0 code ,high voltage time	0.4us
T1H	1 code ,high voltage time	0.8us
T0L	0 code , low voltage time	0.85us
T1L	1 code ,low voltage time	0.45us
RES	low voltage time	Above 50µs

sur l'acronyme NRZ comme *Non-Return to Zero* (un type de codage utilisé avec RS232 par exemple), mais en réalité, ceci se rapproche plus d'un encodage basé sur une sorte de PWM. Un bit 0 est 0,4 µS (micro seconde) à l'état haut, suivi de 0,85 µS à l'état bas. Un bit 1 est encodé par 0,8 µS à l'état haut et 0,45 µS à l'état bas. Un signal de plus de 50 µS à l'état bas est considéré comme un reset. Ces différents états sont nommés respectivement T0H, T0L, T1H, T0L et Treset.

Pourquoi vous raconter tout cela puisque, vous vous en doutez, nous ne comptons pas piloter directement les WS2812B, mais utiliser une bibliothèque ? C'est simple, la communication repose intégralement sur des signaux d'une durée donnée (+/- 150 nanosecondes) qui doit impérativement être respectée. Ceci signifie entre autres choses que tout ce qui peut perturber la communication et impacter ces délais provoquera des problèmes, à commencer par la longueur des fils. Un environnement riche en nuisance électromagnétique (tubes fluorescents), une mauvaise alimentation, l'absence de condensateur de découplage... sont autant de choses qui risquent non seulement de perturber le fonctionnement de ces leds, mais peuvent aussi les détruire.

Comme le décrit admirablement Fabien Batteix sur son blog Skyduino (coucou au passage), il est de bon ton de prendre soin des WS2812B. Comme des lemmings, ces composants ne semblent en effet vouloir qu'une chose : mourir (concernant les lemmings c'est absolument faux, ils ne sont pas suicidaires, mais juste complètement stupides lorsqu'ils sont en masse (comme les humains)).

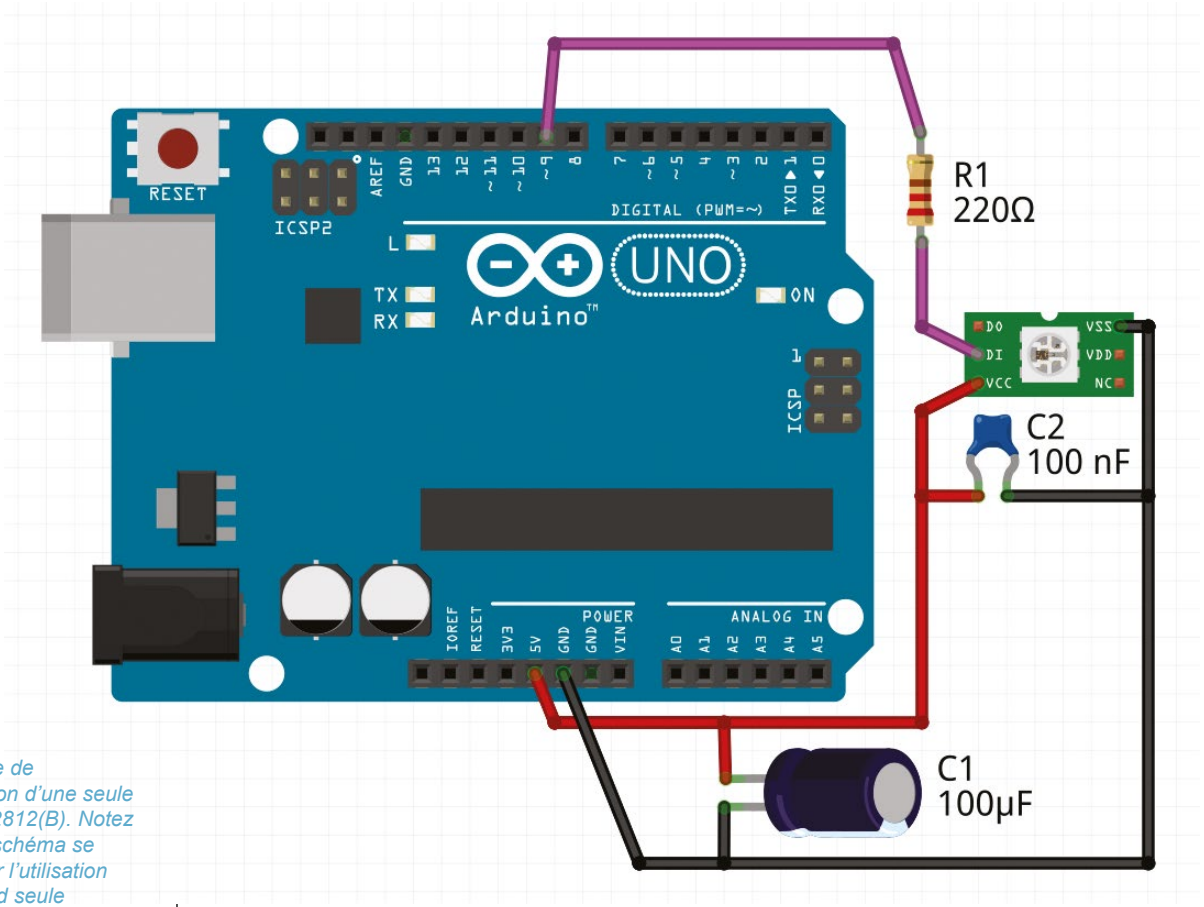
Il faut impérativement donc prendre en compte les faits suivants. Les WS2812B :

- sont hypersensibles à l'électricité statique ;
- ne supportent pas l'humidité ;
- doivent être alimentées en 5V précisément ;
- doivent être équipées d'une manière ou d'une autre d'un système de dissipation thermique ;
- ne supportent pas les alimentations non stables (il faut des condensateurs partout, et par led) ;
- ne supportent pas une inversion de polarité (ce qui est réglé avec le modèle B en principe) ;
- ne doivent pas recevoir de courant sur DIN si la led n'est pas alimentée ;
- etc.

En utilisant des leds montées sur platine ou en ruban, vous réglez potentiellement une partie du problème puisque ces circuits intègrent un condensateur entre l'alimentation et la masse ainsi qu'une résistance sur DIN pour limiter le courant. Les rubans sont souvent moulés dans une résine souple qui met les leds à l'abri de l'humidité, mais ne règle pas le problème de dissipation thermique. Les platines elles, servent de dissipateur thermique (dans une certaine mesure), mais ne règlent pas le problème lié à l'humidité. Selon votre projet, il faudra donc peser le pour et le contre de chaque solution et surtout respecter les besoins des composants en toute occasion. Tester un ruban de 5m avec 150 leds avec une intensité maximum pour chaque couleur et le tout enroulé dans son dévidoir est une très, très, très mauvaise idée !

3. PASSONS À LA PRATIQUE

La connexion d'une led ou une série de leds à une carte Arduino (ou autre) est relativement simple. N'importe quelle sortie numérique peut être utilisée pour relier la carte à DI. Il n'est pas nécessaire que celle-ci dispose de fonctionnalités « analogiques » (PWM et contrôlées par *timer*). Le seul élément important concerne les recommandations que nous avons déjà précisées : gardez les fils les plus courts possible, ajouter une résistance de protection et assurez-vous de filtrer correctement l'alimentation avec des condensateurs de découplage.



Exemple de connexion d'une seule led WS2812(B). Notez que ce schéma se base sur l'utilisation d'une led seule montée sur platine. De ce fait, une résistance de protection est ajoutée sur DI, un gros condensateur (100 µF) stabilise la ligne d'alimentation et un plus petit (100 nF), plus près de la led, ajoute une protection contre les perturbations électromagnétiques (condensateur de découplage). R1 et C2 peuvent se trouver intégrés directement sur le circuit, le ruban ou la platine selon le produit acheté.

Connecter plusieurs leds en série se fait selon le même principe, il suffit de relier le DO d'une led au DI de la suivante. Gardez toutefois à l'esprit que plus vous avez de leds, plus il y aura besoin de courant. Connecté en USB, un montage dans sa totalité ne pourra pas utiliser plus de 500mA. De la même manière, le régulateur de tension monté sur les cartes Arduino possède également ses limites dépendantes de la tension en entrée, de la dissipation thermique et du courant circulant. Pour des montages utilisant une dizaine de leds WS2812B ou plus, il est recommandé (sinon nécessaire) d'utiliser une alimentation adaptée et de qualité, exactement comme pour le pilotage de leds de forte puissance (cf article sur l'émission infrarouge dans *Hackable n°3*).

Côté logiciel, il existe plusieurs bibliothèques en mesure de gérer une ou plusieurs leds WS2812B ou compatibles. La plus complète et la plus utilisée est sans doute celle d'AdaFruit Industries disponible sur GitHub (**https://github.com/adafruit/Adafruit_NeoPixel**) sous le nom *Adafruit NeoPixel* (référence aux produits correspondants chez ce distributeur).

L'un des avantages de cette bibliothèque, bien que provenant d'un vendeur de composants, est le fait qu'il est possible de l'utiliser avec plusieurs composants.

En effet, si nous prenons l'exemple des leds WS2812B et PL9823, bien que relativement similaires, les deux composants possèdent une différence notable. La WS2812B attend une suite de bits sous la forme **GGGGGGGGRRRRRRRRBBBBBBBB** avec **G**, **R** et **B** correspondant respectivement à l'intensité du vert, du rouge et du bleu, soit trois paquets de 8 bits GRB. La PL9823 en revanche utilise le format **RRRRRRRRGGGGGGGGBBBBBBBB** ou, en d'autres termes, trois paquets de 8 bits RGB. Utiliser une PL8323 avec une bibliothèque ne gérant que la WS2812B fonctionnera, mais les couleurs seront fausses.

L'autre différence pouvant exister entre les composants (leds ou circuits de pilotage seuls) concerne les délais utilisés pour représenter les bits. Certains composants (WS2812B, WS2812, PL9823) utilisent un *timing* permettant d'atteindre 800 Kb/s, alors que d'autres comme le WS2811 (qui est un contrôleur sans led) utilisent une vitesse de 400 Kb/s.

La bibliothèque Adafruit NeoPixel est heureusement en mesure de gérer ces différentes combinaisons (et même le format BRG) et ce, pour la totalité des cartes Arduino utilisant un AVR (plus l'Arduino Due et quelques cartes Teensy de PJRC). Il ne vous sera pas possible, en revanche, d'utiliser la bibliothèque avec une plateforme comme une MSP Launchpad de TI, car la bibliothèque, évitant soigneusement d'utiliser les *timers*, repose sur des instructions en assembleur *inline* (de l'assembleur embarqué dans le code C/C++). Bien évidemment, qui dit « assembleur » dit « jeu d'instructions » et donc implicitement une dépendance complète à une famille précise de microcontrôleurs (pour les développeurs : sans surprise, la bibliothèque est bourrée de `ifdef`). Il existe cependant des portages/adaptations de la bibliothèque disponibles pour TI MSP430, STM32, cartes compatibles mbed ou encore Microchip PIC.

3.1 Premier exemple : la base

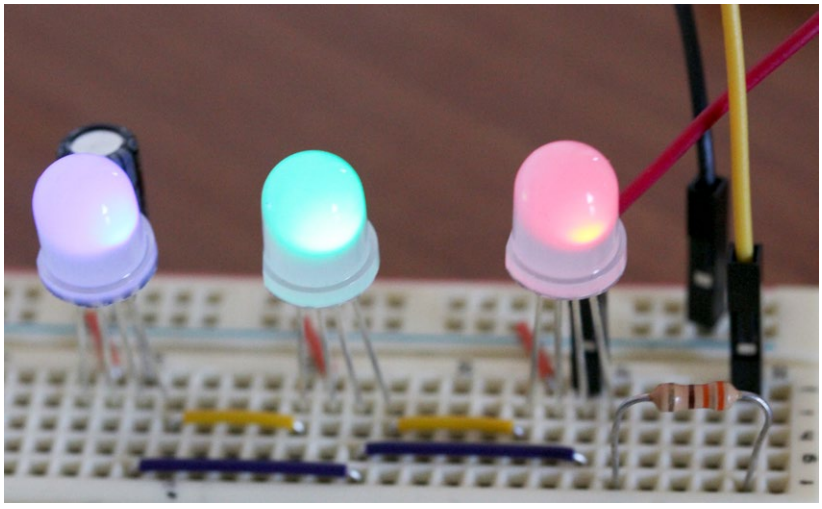
Le plus simple des exemples est le « helloworld » de l'électronique : allumer une led. Ceci sera également la base de l'utilisation de la bibliothèque :

```
#include <Adafruit_NeoPixel.h>

// création de "pixel" représentant
// notre chaîne de leds
// (ici, une chaîne de 1 led)
Adafruit_NeoPixel pixels =
  Adafruit_NeoPixel(1, 9, NEO_GRB + NEO_KHZ800);

void setup() {
  // mise en route
  pixels.begin();
}

void loop() {
  // led en position 0 :
  // 50 % de rouge, pas de vert, pas de bleu
  pixels.setPixelColor(0, pixels.Color(128,0,0));
  // envoi
  pixels.show();
}
```



Pour mon expérimentation, j'ai préféré utiliser des leds PL9823 dont le pilotage est quasi-identique à celui des WS2811B. Une led, même intelligente, qui a la forme d'une led c'est tout de même plus sexy, surtout en 8 mm.

La chaîne de leds (ici une seule led) est représentée par la variable `pixel` de type `Adafruit_NeoPixel`. Celle-ci est initialisée grâce à la fonction éponyme en précisant, en argument, le nombre de leds chaînées, le port utilisé et le type. `NEO_GRB` correspond à un encodage vert/rouge/bleu adapté au modèle WS2812B. Pour des PL9823, on utilisera `NEO_RGB`. Il n'est pas possible, bien entendu, de mélanger WS2812B et PL9823 sur une même chaîne, mais il est parfaitement possible de créer autant de variables `Adafruit_NeoPixel` que nous avons de chaînes sur des ports différents. `NEO_KHZ800` précise la vitesse utilisée (800Khz) et une autre option possible est `NEO_KHZ400` (pour les WS2811+leds par exemple).

La fonction `setup()` se contente d'initialiser la chaîne avec `begin()`. Ceci a pour effet de préparer les attributs privés et publics de la classe, mais réinitialise également toutes les leds. En effet, lors de la mise sous tension, l'état des leds et donc la couleur et l'intensité lumineuse sont des éléments indéterminés (même si mes PL9823 semblent systématiquement s'allumer en bleu). Ainsi, passé `setup()`, toutes leds doivent être éteintes.

Notre fonction `loop()` se contente d'envoyer les données à notre unique led via la méthode `setPixelColor()`. Celle-ci prend en argument la position de la led ciblée (ici `0`, la première et la seule) et les données formatées spécifiquement par la méthode `Color()`. Cette méthode prend en argument les valeurs de rouge, de vert et de bleu. Ceci peut paraître complexe, mais présente l'avantage d'avoir une seule fonction de définition de couleur, au format RVB/RGB, indépendamment de l'encodage utilisé réellement par le composant.

Mais ce n'est pas tout, `setPixelColor()` ne provoque pas le changement de couleur sur le ou les composants. Cette méthode se contente de changer une représentation interne des données. Il faut donc « pousser » ces données sur les leds. C'est le travail de la méthode `show()`.

Sur cette base, vous pouvez déjà explorer les différentes possibilités et, par la même occasion, découvrir l'aspect ludique de la synthèse additive. Vous remarquerez aussi sans doute les limites physiques du procédé. `pixels.Color(32,32,32)` ne vous donnera pas un blanc de faible intensité, mais une espèce de mélange douteux entre le violet clair et le rose. Ceci provient à la fois de la conception même des leds, mais également de la sensibilité de l'œil humain à certaines longueurs d'onde et donc certaines couleurs (le vert/jaune à 555nm est extrêmement bien perçu alors que le rouge ou le bleu nécessitent une plus grande intensité). Un bon exercice consiste à comparer les couleurs HTML format `#RRVVBB` avec les mêmes valeurs appliquées à une led (en commençant par le jaune).

3.2 Jouons un peu

Une led c'est bien amusant, mais puisqu'on peut en ajouter sans que cela ne nous coûte des ports supplémentaires, pourquoi se priver ? Voici donc une petite déclinaison à 3 leds sur la base précédente, mais ajoutant un peu d'animation aléatoire :

```
#include <Adafruit_NeoPixel.h>

// trois leds sur la chaîne
Adafruit_NeoPixel pixels =
  Adafruit_NeoPixel(3, 9, NEO_GRB + NEO_KHZ800);

void setup() {
  pixels.begin();
  // initialisation du générateur de nombres
  randomSeed(analogRead(0));
}

void loop() {
  // première led
  pixels.setPixelColor(0, pixels.Color(
    random(255),random(255),random(255)));
  // seconde led
  pixels.setPixelColor(1, pixels.Color(
    random(255),random(255),random(255)));
  // troisième led
```

```
  pixels.setPixelColor(2, pixels.Color(
    random(255),random(255),random(255)));
  // envoi
  pixels.show();
  // pause
  delay(1000);
}
```

Cette fois, nous laissons le hasard décider des couleurs. Après avoir initialisé le générateur de nombres pseudo-aléatoire à partir d'une valeur lue sur la broche **A0** (non connectée), nous pouvons utiliser la fonction **random()** en précisant en argument la valeur maximale (moins 1) que nous tolérons. Notre fonction **loop()** se charge de décider seule des valeurs de rouge, de vert et de bleu pour les leds de 0 à 2.

Notez le fonctionnement global et l'intérêt de disposer de la méthode **show()**. Nous préparons nos données et ensuite seulement mettons à jour la couleur des leds en un seul mouvement.

Vous trouvez que c'est un peu éprouvant pour votre rétine ? Pas de problème, ajoutez simplement une ligne **pixels.setBrightness(32)**à la fin de la fonction **setup()** et la luminosité globale (théorique) en sera divisée par 8 (256/32). Cette méthode prend en argument une valeur entre 0 et 255 permettant de régler l'intensité lumineuse pour l'ensemble des leds. La bibliothèque se charge, en interne, d'adapter les valeurs effectives RVB que vous spécifiez en fonction de la luminosité choisie.

3.3 Vers une gestion plus pratique des couleurs

À ce stade, nous sommes en mesure d'ordonner à chaque led de prendre la couleur de notre choix et ce, avec l'intensité voulue. Quelques couleurs sont relativement faciles à obtenir avec 255 en guise de valeur :

- rouge ;
- vert ;
- bleu ;
- jaune/orange : rouge + vert ;
- cyan ou bleu verdâtre : vert + bleu ;
- magenta/rose : rouge + bleu ;
- blanc rosâtre : rouge + vert + bleu.

Pour le reste des couleurs intermédiaires, c'est un peu plus délicat, car il faut jouer sur l'intensité de chaque composante RVB. Pour de l'orange par exemple, 255 de rouge et 128 de vert, fonctionneront assez bien. Il est donc possible de se créer une palette, peut-être sous la forme d'un tableau stocké en flash, et ainsi piocher les valeurs pour chaque nuance souhaitée. Reste cependant à construire cette palette et cela, soyons clairs, relève en grande partie du tâtonnement. On pourrait utiliser des potentiomètres reliés aux entrées analogiques de l'Arduino pour régler manuellement les quantités R, V et B. En ajoutant un bouton provoquant l'affichage des trois valeurs sur le moniteur série, on disposera d'une solution pour créer plus facilement une palette.

Mais il existe une meilleure solution et pour en mesurer tout l'intérêt, imposons-nous un cahier des charges simple : piloter une led de manière à ce qu'elle parcoure de manière fluide toutes les couleurs de l'arc-en-ciel, dans l'ordre (si déjà). Avec une led qui peut afficher quelques 16 millions de couleurs (256 x 256 x 256) et donc 16 millions de combinaisons de valeurs RVB sur 24 bits en tout, la palette risque d'être un peu difficile à ajuster...

La bonne solution consiste à ne pas utiliser le modèle RVB, mais TSV pour Teinte Saturation et Valeur (HSV en anglais). C'est un modèle très pratique qu'on retrouve systématiquement dans les logiciels de création graphique par exemple. Comme pour RVB, TSV fait correspondre une valeur par élément contrôlé :

- Teinte : une valeur sur 360 correspondant à un angle dans le cercle des couleurs ou cercle chromatique avec le rouge à 0° (ou 360°), le vert à 120° et le bleu à 250°. Il suffit alors de choisir un angle pour sélectionner une teinte. Et comme c'est un cercle, on peut tourner autour à loisir ;

- Saturation : est exprimée en proportion, généralement sur cent ou 256, et fixe l'intensité de la couleur. Le terme « désaturation » est souvent utilisé pour décrire une image fade/délavée ou comme technique pour passer une image en niveaux de gris ;
- Valeur : c'est la brillance d'une couleur, également exprimée en proportion. Pour comprendre l'intérêt de la V en TSV, il suffit de se rappeler que plus V est petit, plus la couleur est sombre. Avec la valeur à zéro, on obtient du noir, quelles que soient la teinte et la saturation.

Ce qui nous intéresse ici est bien entendu surtout T. En disposant d'un algorithme de conversion TSV vers RVB, nous pouvons alors pousser la saturation au maximum ainsi que la valeur et parcourir le cercle de couleurs de 0 à 359.

Il existe des centaines d'implémentations de l'algorithme de conversion dans tous les langages possibles et imaginables. En voici une en C parfaitement adaptée à un code pour Arduino (que des entiers, pas de **float** et donc une économie notable de flash) :

```
void HSVtoRGB(int  *r, int *g, int
*b, int h, int s, int v) {
  int c;
  long l, m, n;

  // saturation zéro, pas de couleur
  if(s == 0) {
    *r = *g = *b = v;
    return;
  }

  // chroma
  c = ((h%60)*255)/60;
  h /= 60;

  // intermédiaire
  l = (v*(256-s))/256;
  m = (v*(256-(s*c)/256))/256;
  n = (v*(256-(s*(256-c))/256))/256;

  // choix dominante
  switch(h) {
    case 0:
      *r = v; *g = n; *b = l;
      break;
    case 1:
      *r = m; *g = v; *b = l;
      break;
    case 2:
      *r = l; *g = v; *b = n;
      break;
    case 3:
      *r = l; *g = m; *b = v;
      break;
    case 4:
      *r = n; *g = l; *b = v;
      break;
    default:
      *r = v; *g = l; *b = m;
      break;
  }
}
```

Notez que toutes les couleurs décrites par le modèle TSV ne peuvent être représentées en RVB, mais c'est un point de détail ici puisque nous travaillons de toute façon avec une led qui physiquement fonctionne en RVB.

Cette fonction prend en argument des pointeurs sur les variables contenant les valeurs de rouge, de vert et de bleu, ainsi que les valeurs TSV. Sans compliquer l'article plus que nécessaire en déviant de notre objectif, limitons-nous à son utilisation. Après avoir ajouté cette déclaration de fonction dans notre croquis, nous pouvons l'utiliser. Notre **loop()** devient alors :

```
void loop() {
  // valeur RVB
  int r, v, b;
  // on tourne dans le cercle de couleurs
  for (int i = 0; i < 360; i++) {
    // TSV = position, 100% S et 100% V
    HSVtoRGB(&r, &v, &b, i, 255, 255);
    // choix de la couleur de la première led
    pixels.setPixelColor(0, pixels.Color(r, v, b));
    // affichage
    pixels.show();
    // courte pause pour animation
    delay(15);
  }
}
```

Notre boucle `for` va incrémenter `i` de 0 à 359. Nous utilisons cette variable avec notre fonction toute neuve. Notez le `&` précédant les noms de variables `r`, `v` et `b`. Nous spécifions ici les adresses (pointeurs) auxquelles se trouvent ces variables et non leur contenu. Ainsi la fonction `HSVtoRGB()` pourra les utiliser pour leur affecter une valeur. Après appel à `HSVtoRGB()`, les trois variables contiennent les valeurs de rouge, vert et bleu que nous pouvons directement utiliser avec la bibliothèque NeoPixel.

Le résultat sera exactement celui attendu. Nous définissons successivement 360 nouvelles couleurs en tournant d'un degré à chaque fois dans le cercle. Arrivés à 359, on sort de la boucle `for` et on repart pour un tour de `loop` sans le moindre problème puisque 360 et 0 correspondent exactement à la même couleur/position. Notre led va donc, en à peu près 6 secondes, faire en douceur le tour des 360 couleurs.

CONCLUSION

Arrêtons là les expérimentations, car cet article, sinon, pourrait bien finir par occuper l'ensemble du magazine (voir tout ou partie du numéro suivant). L'algorithme proposé pour la conversion TSV vers RGB n'est qu'un exemple, il existe même quelques bibliothèques incluant cette fonctionnalité. Cependant, quelle que soit la solution que vous adopterez, le fait de travailler en TSV facilite énormément la mise en œuvre de ce type de leds dans un projet réel. Voici quelques exemples :

- En n'utilisant que les positions entre 120 et 0 dans le cercle de couleurs, nous obtenons un dégradé entre le vert et le rouge en passant par le jaune et l'orange : idéal pour indiquer un niveau de gravité, d'importance ou de température. En trichant un peu, on peut utiliser 128 positions et nous faciliter la tâche pour directement utiliser des valeurs sur 8, 9 ou 10 bits avec une simple division.
- En enregistrant une palette de couleurs basée sur la teinte, non seulement on économise de la mémoire, mais il devient possible de créer des transitions douces. Ainsi, plutôt que de passer brutalement du cyan au rouge on peut, à l'aide d'une simple boucle, glisser vers la couleur cible. Écrire une fonction dédiée pouvant utiliser n'importe quelle couleur cible (et source) ne demande que peu de travail.
- En utilisant plusieurs leds chaînées, il devient simple de créer un « effet de persistance » avec n'importe quelle couleur. La première led verra sa valeur poussée au maximum puis proportionnellement, on diminuera cette valeur pour les leds suivantes tout en conservant la même teinte et la même saturation. Résultat, un magnifique dégradé d'intensité.
- Il devient bien plus simple de contrôler la couleur d'une led avec une entrée analogique. Un seul potentiomètre connecté à la carte Arduino fixera l'angle (teinte) à utiliser et donc la couleur. Il sera possible d'en ajouter un second pour la saturation et ainsi accéder à des lumières plus « pastel » et au blanc.

Une prise en charge ainsi rendue simple par le duo led intelligente + TSV permet d'apporter beaucoup d'informations dans une « interface utilisateur ». Il deviendra même possible dans certains cas de remplacer purement et simplement un afficheur LCD. Ces composants sont, à mon sens, une véritable aubaine malgré leurs fragilités.

Allez, une dernière idée de projet pour la route : vous avez toujours voulu un réveil simulateur d'aube, mais sans ces horribles et pénibles sons d'oiseaux guillerets ? Une carte Arduino, une grosse alimentation, une RTC DS1307 et quelques rubans de WS2812B et voici votre chambre à coucher devenue elle-même le réveil de vos rêves... **DB**