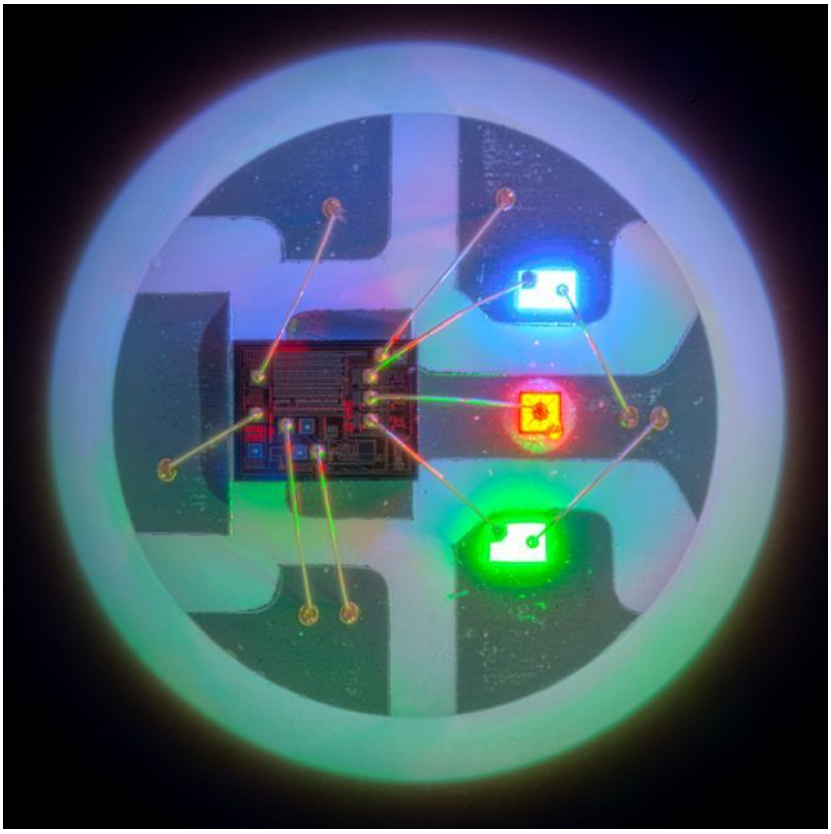
# Les LED WS2812B

Ces LED sont très différentes des LED tricolores classiques. Une LED tricolore classique intègre 3 LED, une rouge, une verte et une bleue, et possède donc 4 broches. Il existe deux organisations pour les 4 broches. Dans la première les LED sont reliées entre elles par la cathode (1 broche) et les 3 autres broches correspondent aux anodes des 3 LED. Dans la seconde, les LED sont reliées entre elles par l'anode et les 3 autres broches correspondent aux cathodes des 3 LED. On préfère généralement les LED à cathode commune car le point commun est relié à la masse.

## Caractéristiques des LED WS2812B

Les WS2812B sont des composants en boîtier 5050, boîtier plat de 5mm de côté et destiné à être soudé en surface [1]. Elles intègrent évidemment 3 LED mais aussi un circuit intégré logique destiné à piloter les LED. Ce circuit intégré gère chacune des LED via une PWM avec une résolution de 8 bits, soit 256 niveau par couleur, pour un total de 16 millions de couleur possibles. Sur le cliché ci-dessous, on peut voir l'implantation des LED et du circuit de pilotage dans le composant.



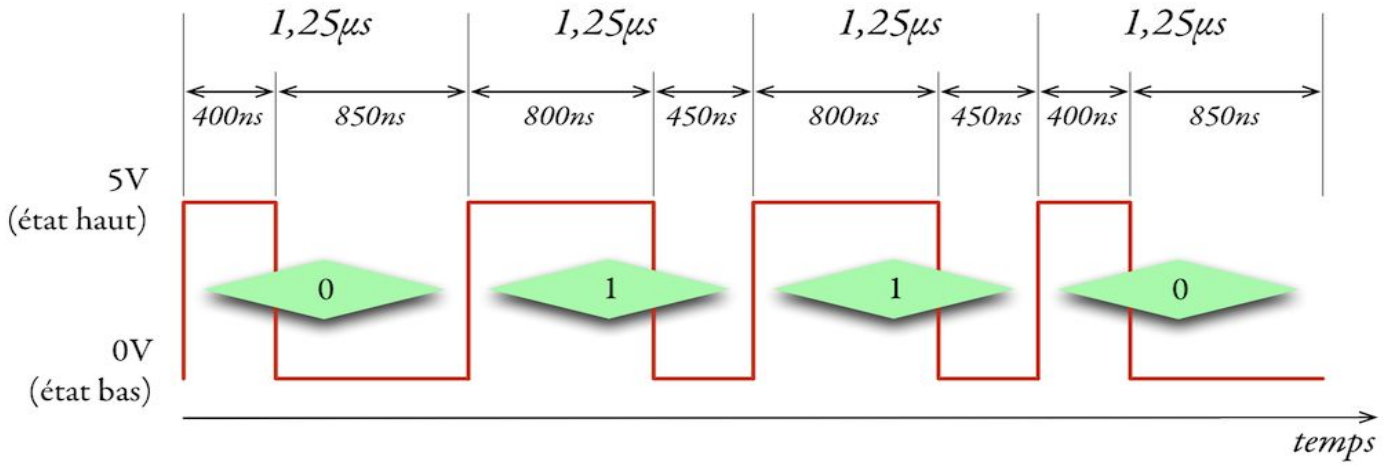
**LED WS2812B en gros plan**
À gauche le circuit de gestion des LED et de la communication.

Le circuit de pilotage gère également la communication. En effet, ces LED sont autonomes et disposent de deux broches de communication permettant de leur envoyer une valeur de PWM pour chacune des 3 couleurs, soit 24 bits de données. La première broche de communication est l'entrée DI (Data In) et la seconde DO (Data Out). Les LED sont conçues pour être chaînées, le DO de la LED amont est connecté au DI de la LED aval.

# Le mode opératoire

À l'état de repos, c'est à dire si aucune information ne doit être envoyée à la LED, le signal DI est maintenu à l'état bas. Le passage de DI à l'état haut indique à la LED le début d'une transmission d'une série de bits. Les bits sont transmis à une fréquence de 800kHz, soit 1,25µs par bit. Il n'y a pas de signal d'horloge séparé, la LED se base sur le signal transportant les données pour se synchroniser.

Un bit à 0 est codé en laissant DI à l'état haut pendant 400ns puis à l'état bas pendant 850ns. Un bit à 1 est codé en laissant DI à l'état haut pendant 800ns et à l'état bas pendant 450ns. Dans le chronogramme example ci-dessous, 4 bits sont transmis.




**Exemple de transmission de 4 bits de données, 0110**

La tolérance sur les durées est de 150ns.

Le micro-contrôleur n'est connecté qu'à la première LED d'une chaîne. Le mode opératoire consiste à envoyer à cette première LED autant de séries de 24 bits qu'ils y a de LED dans la chaîne. Chaque LED prend pour elle les 24 premiers bits qu'elle reçoit et transmet les suivants à la LED suivante de la chaîne. De fil en aiguille chaque LED reçoit les 24 bits qui lui sont destinés.

Après la transmission, DI est maintenu à l'état bas pendant 50µs afin de réinitialiser le mécanisme de transmission de l'information de LED en LED. De cette manière, à la prochaine transmission, l'échantillonnage des 24 premiers bits reçus sera de nouveau effectué par chaque LED de la chaîne.

24 bits demandent une temps de transmission de 1,25µs x 24, soit 30µs. L'envoi des données à une chaîne de 100 LED demande donc 3ms.

# La mise en œuvre

Adafruit a publié plusieurs recommandations pour la mise en œuvre de ces LED. On retiendra le fait de monter sur l'alimentation un condensateur de 1000µF pour absorber les pics de tension à la mise en route et l'ajout éventuel d'une résistance d'environ 500Ω en série entre le micro-contrôleur et la LED de tête lorsque le fil est long pour empêcher le signal de se réfléchir et de perturber la transmission.

Une résistance de 10kΩ entre DI et GND garantit également que la LED de tête ne reçoit pas un signal incohérent alors que la broche du micro-contrôleur n'a pas encore été programmée en sortie.

# La bibliothèque NeoPixel d'Adafruit

Ecrire le programme de rafraichissement d'une chaîne de LED est un exercice assez difficile. En effet, il est nécessaire de garantir les caractéristiques temporelles du signal de manière précise. Cette garantie ne peut être atteinte qu'en connaissant la durée de chaque instruction employée dans le programme de rafraîchissement et disqualifie donc le langage C. L'emploi de l'assembleur est de fait obligatoire.

La bibliothèque NeoPixel d'Adafruit tombe donc à point nommé pour m'épargner cette tâche. Elle offre une classe C++ représentant la chaîne de LED dans la mémoire de l'Arduino, une série de méthode pour manipuler les couleurs et bien sûr une méthode pour rafraîchir la chaîne.

Chaque LED occupe 3 octets en mémoire SRAM. On peut donc piloter une ou plusieurs chaînes avec un Arduino Uno et de manière générale tous les modèles équipé d'un ATMega328 [3] à condition que le nombre total de LED ne dépasse pas 600 [4].

Une fois la bibliothèque installée à l'emplacement adéquat, elle est utilisable dans un croquis.

```
#include <Adafruit_NeoPixel.h>
```

Il faut ensuite instancier un objet de type `Adafruit_NeoPixel`. Comme ceci.

```
Adafruit_NeoPixel leds = Adafruit_NeoPixel(94, PIN, NEO_GRB + NEO_KHZ800);
```

94, le premier argument, est le nombre de LED dans la chaîne. Ici 94 correspond au nombre de LED dans la chaîne de la gare de Messingrohrstadt. PIN est le numéro de broche d'entrée/sortie numérique où le DIN de la première LED de la chaîne est connecté. Comme d'habitude, on évitera les broches 0 et 1 qui servent à téléverser le programme sur l'Arduino et incidemment à établir une liaison série avec la console de l'IDE Arduino à des fins de debug. On évitera également la broche 13 sur laquelle est câblée une LED présente sur la carte Arduino et dont la charge pourrait perturber la transmission.

NEO_GRB + NEO_KHZ800 permet de spécifier le modèle de LED employé. NEO_GRB indique l'ordre des couleurs dans les 24 bits transmis à chaque LED. Pour les WS2812B, l'ordre est vert, rouge, bleu. NEO_KHZ800 fixe la fréquence de transmission soit 800kHz.

### La méthode `begin()`

Une fois l'objet `leds` instancié, on peut appeler ses méthodes. La première chose à faire est d'activer la chaîne de LED en invoquant la méthode `begin()` qui a pour effet de programmer la broche en sortie et de la mettre à l'état bas. L'appel à `begin()` sera fait dans `setup()`.

### Les méthodes `setPixelColor(...)`

La seconde méthode importante est `setPixelColor(...)`. Elle permet de spécifier la couleur d'une LED de la chaîne. Deux versions, où la couleur est indiquée de manière différente, sont disponibles. Quelque soit la version, le premier argument est le numéro de la LED concernée, de 0 au nombre de LED moins 1. Dans la première version, les composantes rouge, vert et bleu sont données séparément et dans cet ordre. Par exemple :

```
leds.setPixelColor(10, 100, 150, 200);
```

réglera la couleur de la LED 10 à 100 (sur 255) pour le rouge, 150 pour le vert et 200 pour le bleu.

Dans la seconde version, un seul argument, un entier 32 bits qui regroupe les 3 composantes. La méthode `Color(...)` permet de construire un tel entier à partir des composantes et peut être employé directement dans `setPixelColor(...)`

```
leds.setPixelColor(10, leds.Color(100, 150, 200));
```

À quoi cela avance-t-il me direz vous ? À stocker des couleurs prédéfinies dans des entiers 32 bits au lieu de les disperser dans 3 octets chacune.

### La méthode `setBrightness(...)`

La méthode `setBrightness(...)` permet de changer la luminosité des LED. Les composantes de chaque LED sont ajustées en mémoire. Comme les calculs sont faits en nombre entiers, une augmentation de luminosité ne donne pas un résultat exact. Notamment diminuer la luminosité et la remettre au niveau initial ne redonnera pas la même couleur. Il est donc nécessaire, après un `setBrightness(...)` de mettre à jour la couleur de la totalité des LED de la chaîne via `setPixelColor(...)`.

La luminosité programmée est appliquée lors du `setPixelColor(...)`. `setBrightness(...)` prend pour argument une valeur de 0 à 255. 0 correspond à l'extinction des LED et 255 à la luminosité maximum qui est aussi la valeur par défaut à l'initialisation.

```
leds.setBrightness(127); // luminosité moitié
```

### La méthode `show()`

Surnommée aussi « méthode que l'on oublie systématiquement d'appeler », `show()` opère un rafraichissement de la chaîne de LED en envoyant les couleurs stockées en mémoire. Elle doit donc suivre les modification de couleur ou de luminosité effectuées par les fonctions vues précédemment.

```
leds.show();
```