D18 : Décodeur d’accessoires DCC

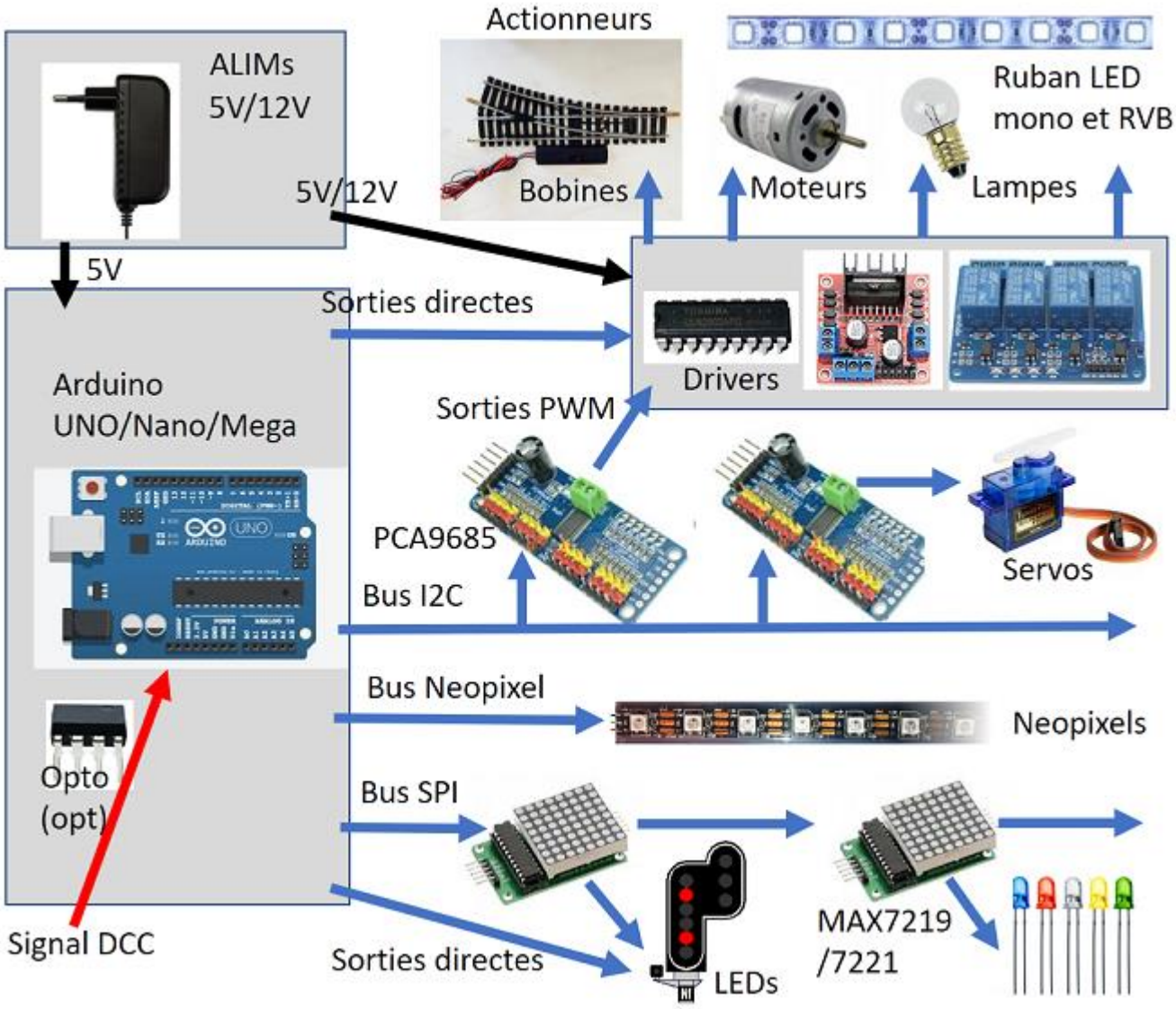


Table des matières :

(Ulysse Delmas-Begue documentation du 08 février 2018)

1. Introduction

** Présentation

Le décodeur D18 (D pour Décodeur et 18 pour 2018) est un décodeur d'accessoires DCC à réaliser soi-même.
Il est basé sur une carte Arduino (UNO, Nano ou Mega) qui grâce à son programme analyse le signal DCC en provenance de la centrale DCC et en extrait les ordres pour les décodeurs afin d'exécuter les actions demandées.

Il permet de commander économiquement tout ce que vous voulez sur votre réseau. (Hormis les locomotives gérées par la centrale)
Il est par exemple possible de piloter :
- des aiguillages de tout type (à bobines, à moteurs ou à servos)
- des LEDs
- des Néopixels (LED multicolores chainables)
- des lampes (avec control de la luminosité grâce à la PWM)
- des moteurs (avec control de la vitesse grâce à la PWM)

S'ajoute de nombreuses possibilités comme le clignotement des LEDs avec phase réglable, le déplacement lent des servos, des effets lumineux ...
Le décodeur est compatible avec n'importe quelle centrale et vous pouvez régler tout ce que vous voulez.

En utilisant une carte Arduino UNO ou Nano, vous pouvez piloter de base 16 sorties simples. Elles sont généralement utilisées en paires ce qui permet de piloter par exemple 8 aiguillages à bobines. Avec la carte Arduino Mega, vous disposez de 66 sorties simples. J'appelle sorties directes les sorties de l'Arduino.

Il est possible d'ajouter jusqu'à 4 modules MAX7219/21 pouvant chacun piloter 64 LEDs, soit 256 LEDs max. Le décodeur gère aussi le clignotement et la phase des LEDs. Cela sacrifie 3 sorties directes.

Il est possible de rajouter jusqu'à 6 modules PCA9685 rajoutant chacun 16 sorties soit 96 sorties max. Ces sorties peuvent fonctionner en mode « PWM » afin d'avoir une sortie variable pour par exemple faire varier l'intensité lumineuse d'une lampe ou la vitesse de rotation d'un moteur. Elles peuvent également fonctionner en mode « Servo » afin de contrôler des servo-moteurs. Cela sacrifie 2 sorties directes.

Il est aussi possible de commander jusqu'à 60 Néopixels qui sont des LEDs multicolores chainables grâce à 3 fils seulement. Cela sacrifie une sortie directe.

** Prix

Dans le commerce, un décodeur à 8 sorties (pouvant gérer 4 aiguillages à bobines par ex) coute dans les 50€. A noter, que l'on commence à en trouver des moins cher dans les 35€. A 50€, cela fait 12€ pour commander un aiguillage à bobines, moteur ou servo et 6€ pour contrôler une LED… Avec de tel prix, la note grimpe vite et il peut même arriver que le décodeur coute plus chère que le dispositif qu'il contrôle !

Pour réaliser un décodeur D18 à 16 sorties, il vous en coutera 5€ pour la carte Arduino, 1€ pour un optocoupleur, 1€ pour 2 ULN2803 pour obtenir des sorties 12V 500mA capable par ex de commander des aiguillages Jouef. Quelques euros de plus pour mettre les ULN sur une petite carte. Pour moins de 10€ vous obtenez l'équivalent de 2 décodeurs du marché à 50€ pièce soit 100€. Le prix par sortie tombe à 0€60. C'est donc 10 fois plus économique …

Un décodeur D18 à 66 sorties revient quant à lui à 20€ et remplace plus de 8 décodeurs à 50€, soit 400€. La sortie passe donc à 0€30. C'est 20 fois moins chère …

Si vous avez besoin de plus de sorties, un PCA9685 à 8€ avec 2ULN revient à 10€ et remplace 2 décodeurs soit 100€. C'est 10 fois moins cher ... Vous pouvez aussi choisir de réaliser un second décodeur.

Pour les servos, un module PCA à 8€ gérant 16 servos remplace avantageusement 4 décodeurs soit 200€. C'est donc 25 fois plus économique …

Pour les LEDs, un module MAX à 4€ gérant 64 LEDs remplace avantageusement 8 décodeurs soit 400€. C'est donc 100 fois plus économique …

Pour résumer :

```
Prix du commerce  >>>> Prix avec D18 et division par rapport au commerce

1 sortie    =  6€  >>>> 0€60   %10   (sur UNO/Nano)
1 aig(bob)  = 12€  >>>> 1€20   %10   (sur UNO/Nano)
1 aig(mot)  = 12€  >>>> 1€20   %10   (sur UNO/Nano)
1 sortie    =  6€  >>>> 0€30   %20   (sur Mega)
1 aig(bob)  = 12€  >>>> 0€60   %20   (sur Mega)
1 aig(mot)  = 12€  >>>> 0€60   %20   (sur Mega)
1 servo     = 12€  >>>> 0€50   %25   (via PCA9685)
1 led       =  6€  >>>> 0€06   %100  (via MAX7219/21)
1 neo       = NON  >>>> 0€           (sur UNO/Nano/Mega)
1 pwm       = NON  >>>> 0€50         (via PCA9685)
```

Le décodeur D18 n'est pas seulement économique, il est aussi très performant et adaptable.

** Réalisation :

Pour la réalisation, il n'y a pas besoin de compétences particulière puisque les cartes électroniques sont disponibles sur Amazon par exemple. Les cartes peuvent être reliés simplement grâce à des câbles « Dupont ». La partie électronique explique toutes ces connections.

Si vous avez besoin d'amplifier les sorties il faudra peut-être souder quelques composants et réaliser de petites cartes. Dans le monde Arduino il est courant de faire des « Shields » (boucliers) qui sont des petites cartes qui se branchent sur la carte. Les carte UNO et MEGA sont conçues en ce sens. Pour la carte Nano, c'est l'inverse, c'est la carte Nano qui s'insère sur une carte fille.

Pour réaliser des petites cartes électroniques, vous pouvez utiliser du « veroboard ». Ce sont des cartes percées au pas de 2.54mm qui disposent de pastilles ou bandes de cuivre. Si vous êtes vraiment motivés, vous pouvez réaliser un circuit imprimé. Si vous ne voulez vraiment rien souder, vous pouvez utiliser des supports de circuit à « wrapper » et brancher des câbles Dupont sur leurs longues pattes.

** Le programme de l'Arduino :

Pour que le décodeur fonctionne, il faudra programmer l'Arduino. Dans le langage Arduino, on dit qu'il faut téléverser (télécharger) le sketch (le programme) sur la cible (la carte Arduino) par l'USB.
Avant de téléverser le programme, vous devrez le configurer afin que le décodeur fasse exactement ce que vous voulez. La partie Logiciel explique cela.

2. Electronique

2.1. Les Arduinos

Comme le cœur du montage est une carte Arduino, cette section s'intéresse un peu à ces cartes.

Les cartes Arduino embarquent un microcontrôleur qui est un composant électronique contenant un véritable petit ordinateur. En effet, on y trouve un processeur, de la mémoire (pour le programme et les données) ainsi que des périphériques, comme des ports pour lire ou changer l'état des pattes (0 ou 5V), des temporisateurs, des dispositifs de communication, un convertisseur analogique numérique pour mesurer des tensions … Le microcontrôleur de la plupart des cartes Arduino est l'AT328P d'Atmel qui dispose de 32KB pour le programme et 2KB pour les données. Il exécute 16millions d'opération par seconde.

Sur la plupart des modèles, un circuit (AT32U4, FT232 ou son clone Chinois CHR340) permet de faire la conversion entre l'USB et le port série du microcontrôleur. Certains microcontrôleurs n'ont pas besoin de ce circuit car ils supportent l'USB directement comme l'Arduino Leonardo ou micro. Il est important de ne pas choisir ces modèles car le programme de l'Arduino est interrompu à chaque requête USB ce qui peut empêcher le décoder correctement le signal DCC à la dizaine de micro seconde près et le montage ne fonctionnera pas. AT32U4 et FT232 sont supportés de base par l'environnement Arduino. Par contre, pour le CHR340, il faudra installer manuellement le driver. Pour ma part, je l'ai trouvé ici : chr340/341: https://arduino-info.wikispaces.com/Nano-USB-Drivers

Pour en savoir plus, je vous invite à aller sur le site de la fondation Arduino https://www.arduino.cc/.
Je vous conseille de lire les excellents manuels d'utilisation qui pullulent sur Internet
(ex : https://www.tutorialspoint.com/arduino/arduino_tutorial.pdf)
Vous verrez que les Arduinos offrent d'énormes possibilités. Peut-être, les utiliserez-vous pour d'autres projets.

Installez l'environnement de développement (on parle d'IDE pour « Integrated Development Environment »), compilez le programme d'exemple « blink led » (led clignotante) qui fait clignoter la led « L » présente sur la carte.
Ne me demandez pas mon aide pour cette phase, tout est très bien expliqué sur Internet et rien ne vient de moi ;-)
Maintenant que vous êtes des pros d'Arduino, vous pourrez télécharger le programme des décodeurs après l'avoir configuré. Dans le jargon Arduino, on dit que l'on « téléverser » un « sketch »

Pour repérer les pattes, nous utiliserons les noms sérigraphiés sur la carte. Nous utiliserons les pattes suivantes :

- Le port USB pour alimenter la carte (soit via un ordinateur ou via un chargeur 5V)
- GND : La masse (à relier à toutes les masses du système)
- 5V : pour alimenter les circuits sous 5V qui ne consomment pas trop (fusible de 500mA sur l'USB)
- D2 à D13 ou D53 et A0 à A5 ou A7 ou A15 : les entrées et sorties pour nos signaux.

Les autres pattes ne seront pas utilisées et seront laissées en l'air.
Le D devant les entrées/sorties (E/S) signifie Digital (numérique, 0/1 = 0V/5V). Le D est souvent absent.
Le A devant les entrées/sorties (E/S) signifie Analogique. Ces pattes sont connectées au convertisseur analogique/numérique qui permet mesurer la tension présente sur la patte (0=0V … 1023=5V). Ces pattes peuvent aussi fonctionner en numérique. Sur Nano, A6 et A7 fonctionnent uniquement en entrées, nous ne les utiliserons donc pas. Sur Mega, toutes les pattes A0 à A15 peuvent être utilisées en sortie.

Vous remarquerez le tilde devant les pattes D3, D5, D6, D9, D10 et D11 sur UNO et Nano. Cela signifie qu'elles supportent la PWM (Pulse Width Modulation = Modulation de largeur d'impulsion) afin d'avoir des sorties variables. Nous n'utiliserons pas la PWM des Arduinos.
Ne pas utiliser les pattes :
- D0 et D1 qui sont utilisés pour la liaison série (qui est ensuite transmise via USB)
- D13 qui est connectée à une LED sur la board que nous ferons clignoter pour montrer que tout se passe bien.
- D2 qui supporte les interruptions sera utilisé pour capturer le signal DCC
Toutes les autres pattes D et A peuvent être utilisées pour les sorties du décodeur

La carte la plus connu est l'Arduino UNO que voici :

```
Flash : 32KB
RAM   : 2KB
Quartz: 16MHz
E/S   : 12 digitales + 6 analogiques
UART  : 1 connectée à l'USB
PWM   : 6 (3 timers)
```

Elle dispose de 20 lignes d'entrées/sorties pour y connecter notre électronique. Pour communiquer avec un PC, elle dispose d'un port USB qui sert aussi à l'alimenter en 5V. Si on n'utilise pas l'USB, il est possible de l'alimenter via un chargeur USB via le port USB ou via un bloc secteur via l'entrée Jack. Cette carte permet de réaliser un décodeur D18 à 16 sorties.

La carte Arduino Nano est beaucoup plus petite, moins chère et dispose de 2 entrées analogiques (pouvant mesurer une tension entre 0 et 5V) supplémentaires. Vous trouverez cette carte à moins de 5€ sur Amazon par exemple. La carte est souvent livrée avec les connecteurs à souder. Le connecteur à 2x 3pattes ne nous sert pas, il n'est pas nécessaire de le souder. Cette carte permet de réaliser un décodeur D18 à 16 sorties.

```
Flash : 32KB
RAM   : 2KB
Quartz: 16MHz
E/S   : 12 digitales,6+2 analogiques
UART  : 1 connectée à l'USB
PWM   : 6 (3 timers)
```

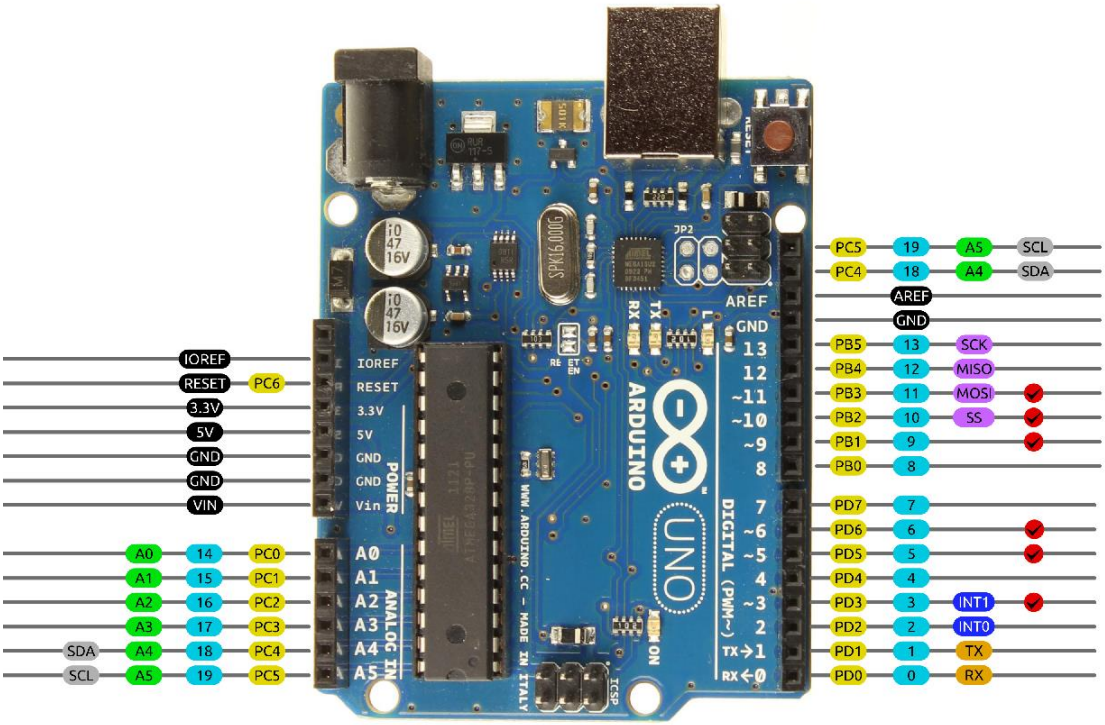
La « grosse » carte Arduino Mega (2560) propose quant à elle bien plus d'entrées/sorties. Le microcontrôleur utilisé est de la même famille que les cartes précédentes, mais il dispose de plus de ports, timers, liens séries, mémoire. Cette carte permet de réaliser un décodeur D18 à 66 sorties.

```
Flash : 256KB
RAM   : 8KB
Quartz: 16MHz
E/S   : 54 digitales + 16analogiques
UART  : 1 connectée à l'USB + 3
PWM   : 12 (5 timers)
```

La figure suivante présente le schéma électronique de l'Arduino Nano

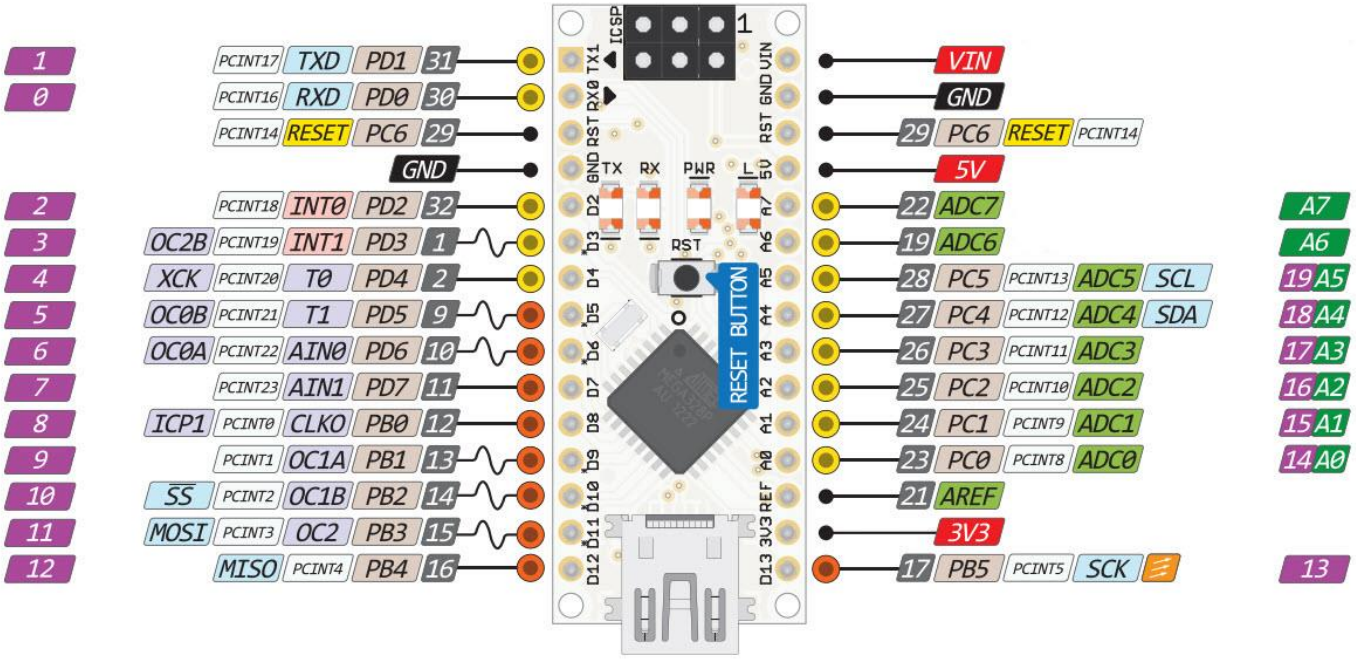
Le schéma suivant identifie les pattes de l'Arduino UNO :

Arduino Uno R3 Pinout

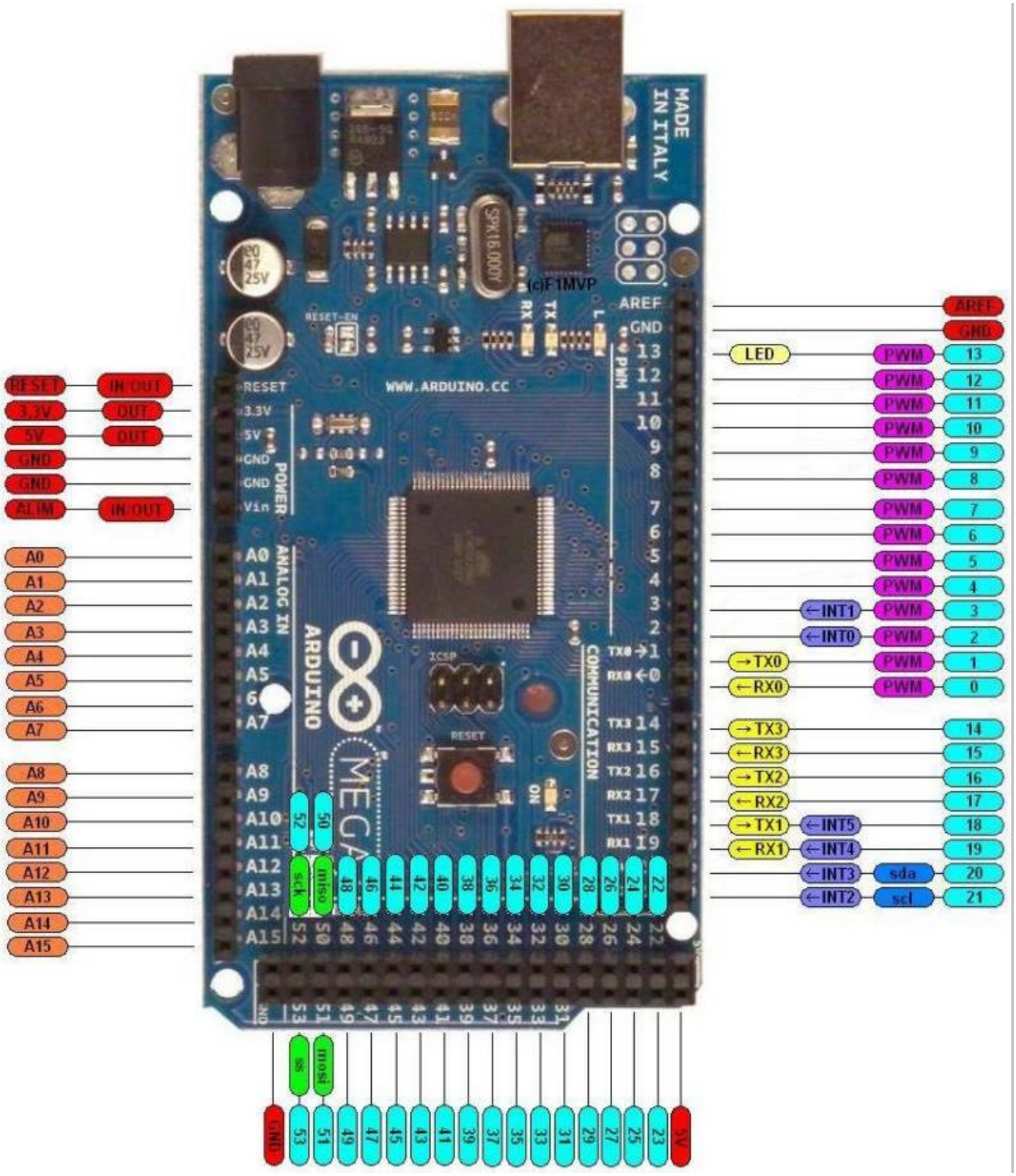


AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT

Le schéma suivant identifie les pattes de l'Arduino Nano:



Le schéma suivant identifie les pattes de l'Arduino Mega :



Comme ces cartes sont en « open hardware », les schémas sont disponibles gratuitement. Vous les trouverez sur Internet ou en annexes.

2.2. Les alimentations

Toute réalisation électronique a besoin d'être alimentée par différentes tensions. Dans notre cas, nous auront certainement besoin des tensions suivantes :

- 5V continu pour l'Arduino, la plupart des circuits électronique, les leds, néopixels, servos et certains relais
- 12V pour les aiguillages, certains relais et autres accessoires.

Il faudra dimensionner la puissance des alimentations en fonction de la consommation maximale.

Il est important de relier la masse des différentes alimentations sous peine de surprises ;-)

La carte Arduino, s'alimente en 5V par son port USB (via un PC ou via un chargeur en autonome). Le port USB de la carte dispose d'un fusible (à réarmement automatique) de 500mA. En d'autres termes, la carte ne pourra pas délivrer plus de 500mA (moins sa consommation modique) sur sa patte 5V. On peut être tenté d'alimenté la carte par son entrée Vin ou en par sa patte 5V mais cela peut poser des problèmes si ces alimentations sont éteintes et que l'on utilise l'USB.

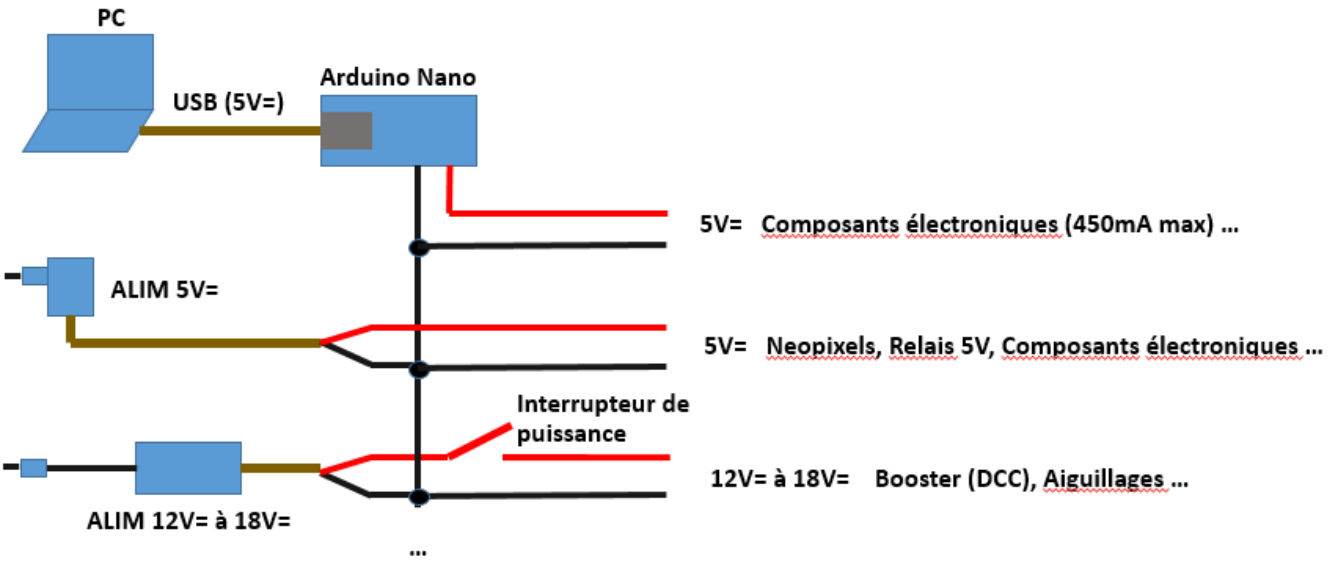
Utiliser un PC permet d'utiliser le port série de l'Arduino pour par exemple voir les paquets DCC décodés par le décodeur ou contrôler le décodeur autrement que par le DCC.

Exemple de quelques consommations :

- Aiguillage : cela dépend énormément. Ex : à bobines : 500mA à 2A, à moteur : 200mA …
- 7219 avec 64 leds : 330mA
- Néopixel (60mA par led lorsque les 3 composantes rouge/vert/bleu sont à 100%)

Il est de bonne pratique de prévoir un interrupteur sur l'alimentation de puissance.

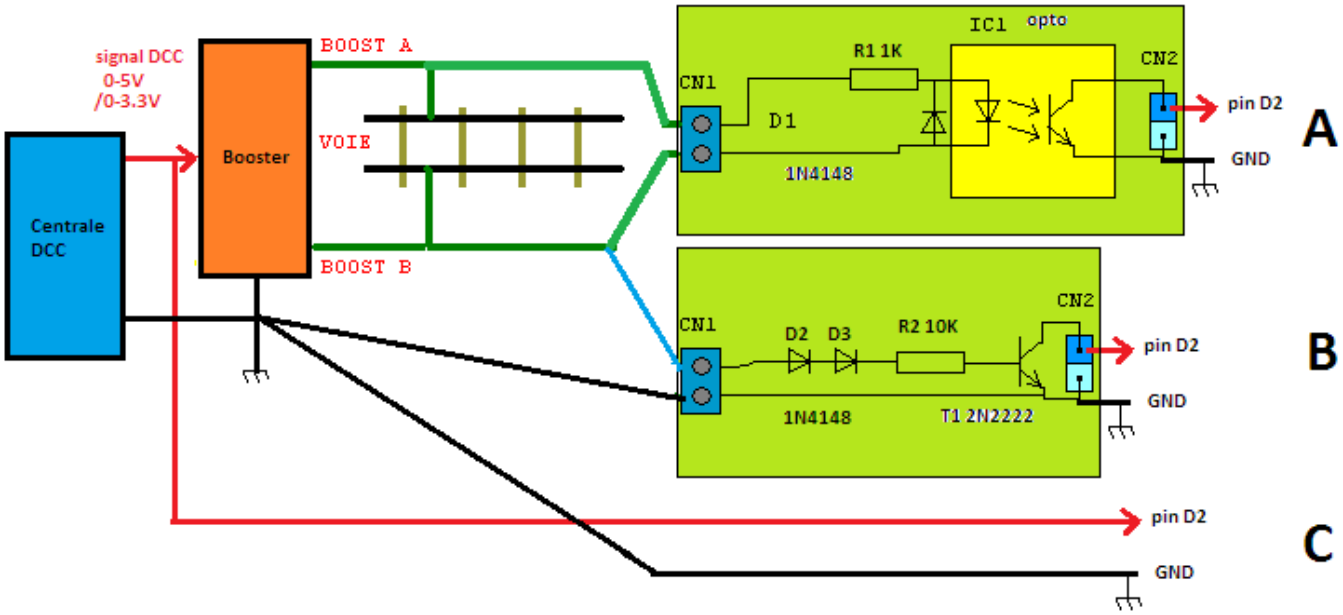
Exemple de raccordement (Alim 5V 1A, Alim 13.8V 3A) :



.

2.3. Capture du signal DCC

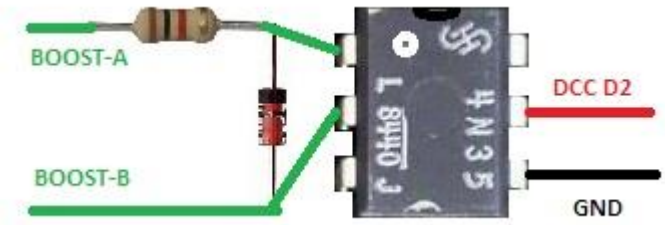
Pour décoder les trames DCC, le décodeur a besoin de capturer le signal DCC. La figure suivante montre quelques exemples de connexions :



Si vous avez accès au signal 0-5V ou 0-3.3V qui commande le booster, le cas C, est le plus simple puisqu'un fil suffit. Cette solution offre l'avantage de ne pas subir les problèmes du signal après le booster comme un court-circuit entre les rails.

Si vous avez accès à la masse du booster, vous pouvez utiliser le montage du cas B. Pas besoin de résistance de pull-up après le transistor, car je l'ai activé sur l'entrée D2 de l'Arduino. Les 2 diodes servent à augmenter la tension de commutation car suivant la consommation le booster n'est pas capable de commuter 0V exactement à l'état bas. Par exemple, dans le cas du LMD18200T, sa résistance interne est de 0.3Ω, ce qui fait 0.9V à 3A pour un niveau bas. Notez que vous pouvez aussi utiliser un pont diviseur bien calculé, mais je préfère cette solution qui ne dépend pas de la tension d'alimentation du booster.

Enfin, la solution A est la plus complexe et plus universelle. Pour l'optocoupleur, beaucoup de montages DCC utilisent les 6N137 ou 4N35. Vous pouvez utiliser d'autres modèles mais assurez-vous qu'ils fonctionnent au moins à 100kHz. Je conseille le 4N35 qui est moins chère et coute moins de 1€. La figure suivante montre le montage avec un 4N35, la résistance de 1kΩ et la diode 1N4148 pour protéger la LED de l'optocoupleur en inverse.



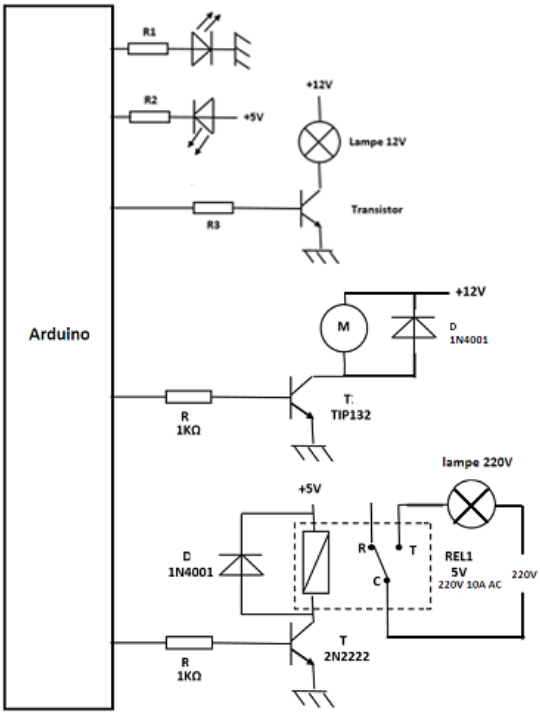
Pour vérifier le montage, vous pouvez par exemple relier l'Arduino à un ordinateur avec l'IDE Arduino et utiliser le moniteur série pour voir si le décodeur arrive à décoder les trames DCC. Exemple pour la réception d'un paquet de 3 octets qui demande au décodeur d'adresse 10 d'activer la sortie 5 :

```
#3 138 253 119 ACC pkt adr(1-511)=10 out(0-7)=5 val(0-1)=1
```

2.4. Les sorties

Les entrées/sorties D et A de l'Arduino sont des pattes sont capables de fonctionner en entrée ou en sortie.

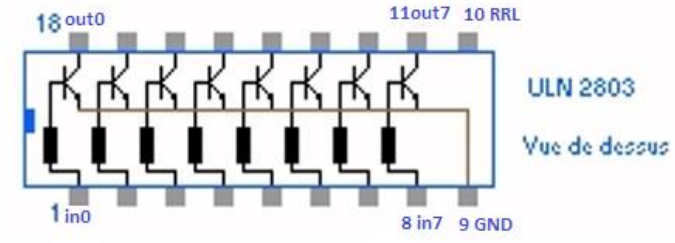
En sortie, le programme peut mettre soit un niveau haut = « 1 » = 5V ou un niveau bas = « 0 » = 0V. Les sorties peuvent fournir un courant max de 40mA. Par sécurité, on veillera à ne jamais dépasser 20mA par patte et 200mA pour l'ensemble des ports sous peine de détruire le circuit. 20mA suffit pour une LED, mais pour de plus gros consommateurs, il faudra amplifier ce courant en connectant un transistor que l'on utilisera en commutateur. On commute généralement la masse plutôt que l'alimentation positive. Notez que la première LED s'allumera avec 5V alors que la seconde s'allumera avec 0V.



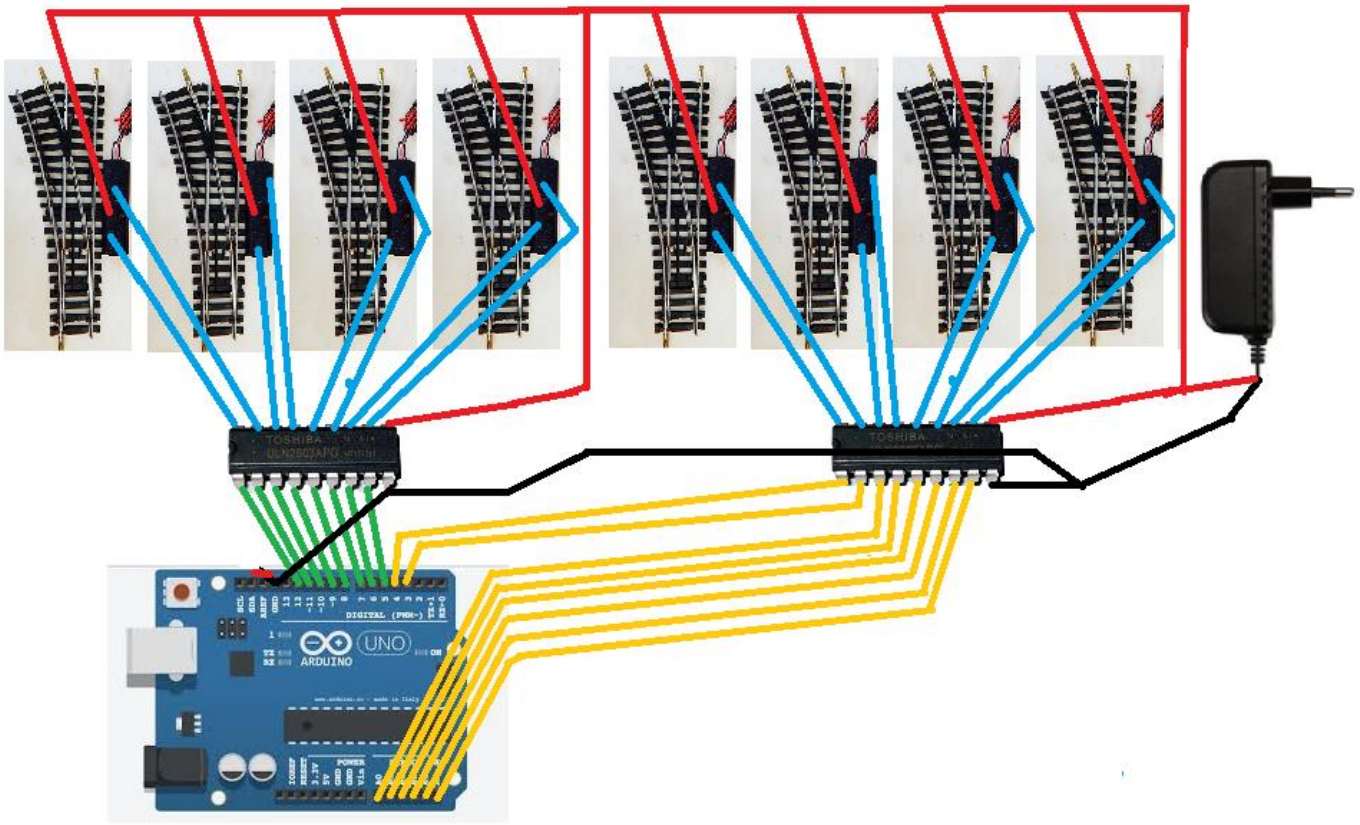
Il existe des milliers de transistors, on pourra par exemple choisir un 2N2222 associée à une résistance de 1KΩ pour commuter 100mA ou un TIP132 pour commuter 10A ! Si la charge est inductive, on rajoutera une diode en inverse afin d'éliminer la surtension suite à la coupure du courant dans la bobine. On pourra par exemple utiliser une 1N4001 pour une charge de 1A max. Notez qu'un transistor ne peut pas commuter des tensions alternatives car les transistors fonctionnent que dans un seul sens, utilisez un relais dans ce cas.

Pour les charges plus importantes ou nécessitant une isolation, vous pouvez utiliser un relais électromagnétique

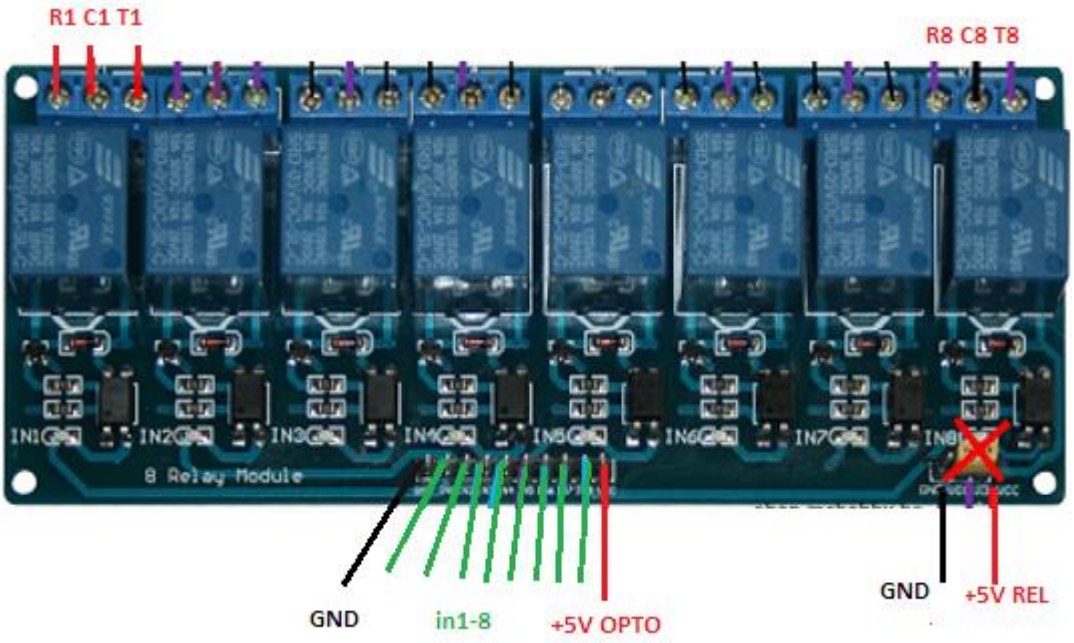
Notez l'existence d'un circuit particulièrement intéressant, l'ULN2803. Ce réseau de transistors Darlington permet de commander à partir d'un faible courant disponible en sortie d'un circuit intégré des charges jusqu'à 500mA. Lorsque l'on met à l'état haut une des entrées, alors la sortie correspondante est reliée à la masse grâce à un transistor de puissance. Lorsqu'il n'y a pas de commande ou qu'elle est à l'état bas, alors le transistor ne conduit pas et la sortie est en l'air. Il faut donc voir chaque sortie comme un interrupteur relié à la masse qui se commande par l'entrée correspondante. Ne connectez pas une charge qui consomme plus de 500mA. Si toutes les charges utilisent la même tension, vous pouvez connecter la broche 10 à l'alimentation positive, en effet de cette broche part une DRL sur chaque sortie. Ce circuit coute moins de 1€. La figure suivante détaille un ULN2803 :



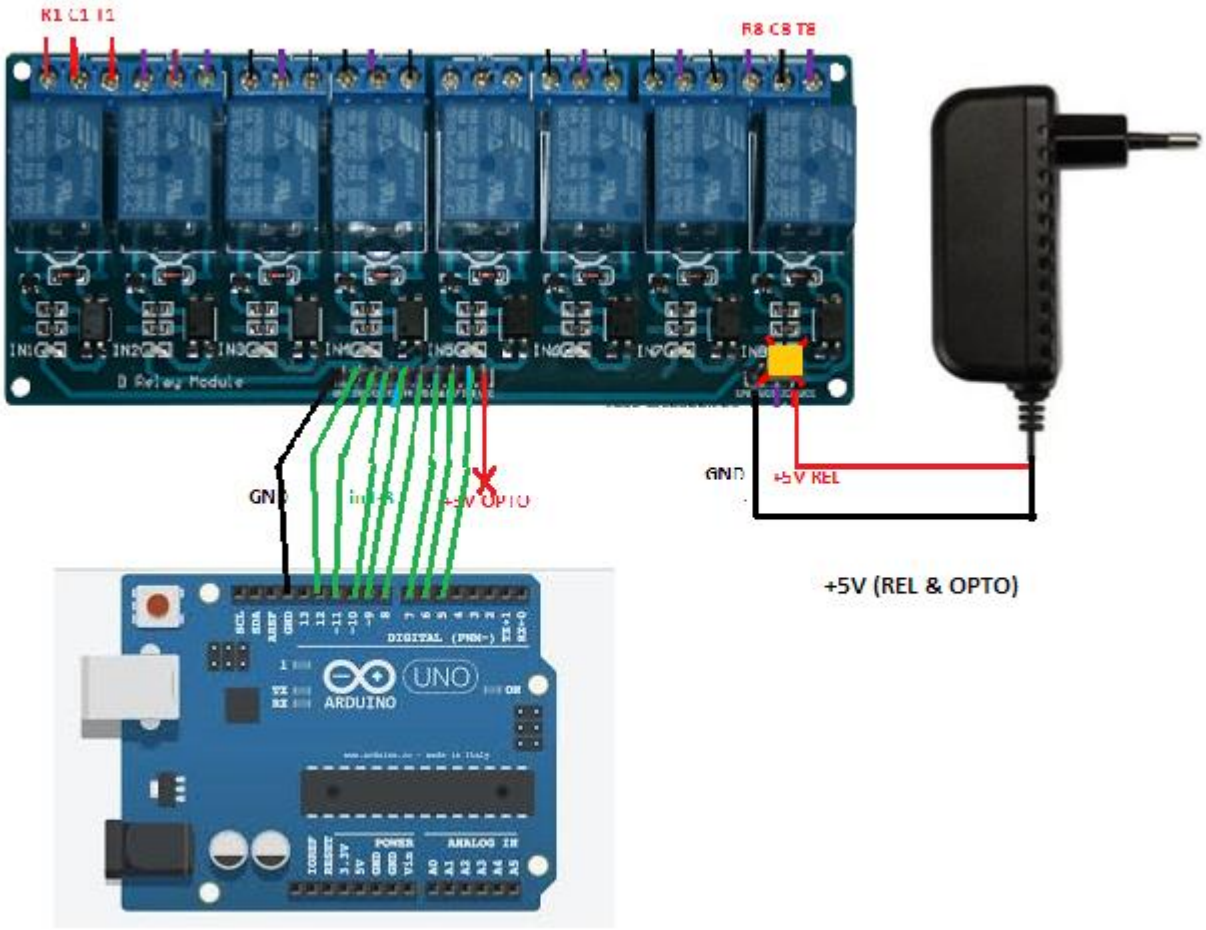
Vous pouvez par exemple commander des aiguillages à bobines qui consomment moins de 500mA comme les vieux aiguillages Jouef. Pour les aiguillages plus gourmands, il faudra utiliser des transistors plus costauds ou des relais. Bien entendu comme les aiguillages Jouef n'ont pas d'interrupteurs de fin de course il faudra utiliser des impulsions afin de ne pas cramer les bobines. Lorsque vous commander de tel aiguillages, essayez toujours en premier avec des LEDs afin de vérifier que la commande par impulsions fonctionne.



A notez, vous pouvez trouver des modules de 1, 2, 4 ou 8 relais sur Amazon au prix de moins de 1€ le relais. Chaque canal comporte un optocoupleur qui commande un transistor qui commande un relais. C'est un peu bête car les masses sont communes et donc les optocoupleurs ne servent à rien … Attention à la commande, les relais collent lorsque l'on met la masse sur les entrées in. Cela éclaire les leds des optocoupleurs. Il faut relier le +5V des leds des optocoupleurs à un 5V. Les relais 5V sont alimentés par le +5V REL. Ne pas utiliser le +5V de l'Arduino pour les relais, cela pourrait le perturber lors des commutations. Pour désactiver une sortie, il faudra laisser l'entrée correspondante en l'air ou alors mettre 5V !



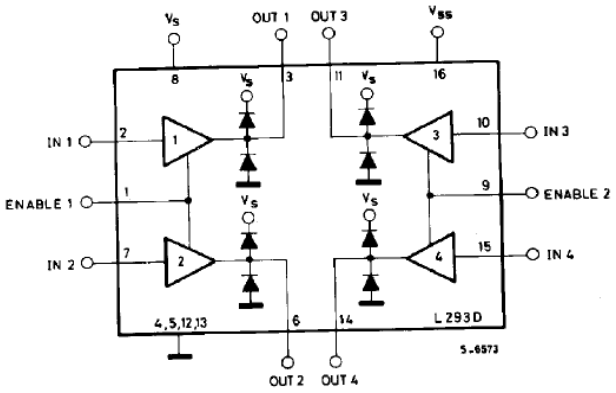
Le +5V des optocoupleurs pourra être pris sur l'Arduino mais dans ce cas il faudra enlever le jumper. Sinon optos et relais peuvent être alimentés par le même 5V. Les masses sont communes sur les 2 connecteurs. Nous avons utilisé cette dernière solution pour le montage suivant :



Les relais peuvent être utilisés pour commander les aiguillages mais vous pouvez aussi les utiliser en complément si le moteur de l'aiguillage ne comporte pas d'interrupteur et qu'il faut polariser son cœur métallique.

Pour les aiguillages à moteur, il faut un dispositif capable d'inverser le sens de rotation du moteur. Vous pouvez utiliser un relais 2RT en inverseur. Vous pouvez aussi utiliser une alimentation alternative et utiliser un relais 1RT associé à 2 diodes qui laissent passer les alternances positives ou négatives selon la position du relais.

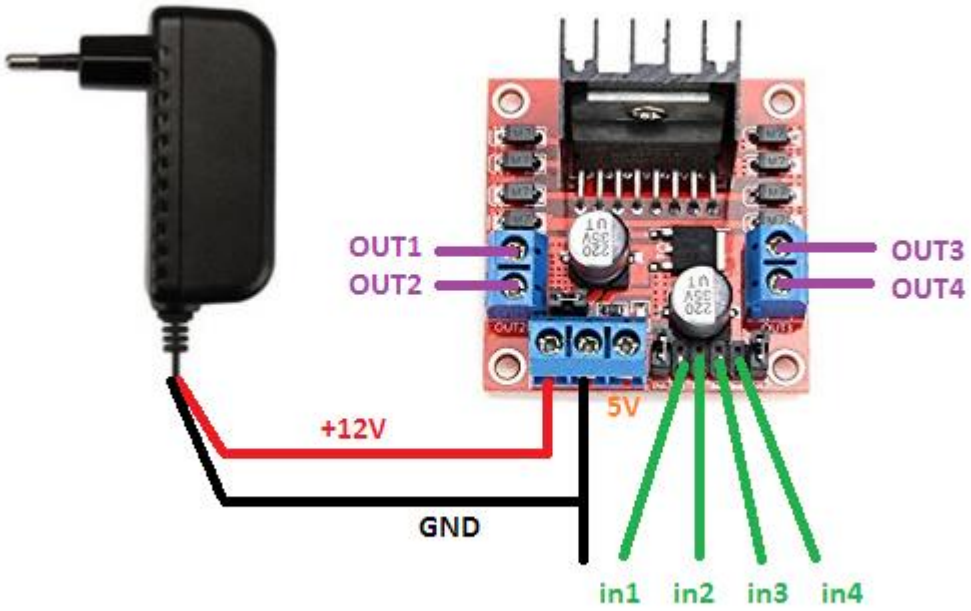
Notez l'existence du circuit L293D qui est un pont en H intéressant pour commander les aiguillages à moteur (lent ou non). Nous l'utiliserons comme de simples interrupteurs commandés. Les amplificateurs 1 à 4 mettent Vs ou la masse sur les sorties OUT1 à OUT4 suivant les niveaux présents sur les broches IN1 à IN2. Enable 1 et 2 permettent d'activer respectivement les amplificateurs 1,2 et 3,4 lorsqu'ils sont mis à 5V. La partie logique du circuit s'alimente par Vss que l'on mettra à 5V. La partie puissance est alimentée par Vs (patte 8) que l'on mettra par exemple à 12V. Les sorties supportent au maximum 700mA ce qui est largement suffisant pour les moteurs des aiguillages qui consomment fort peu comparés aux aiguillages à bobines. Ce composant est également protégé contre les courts-circuits.



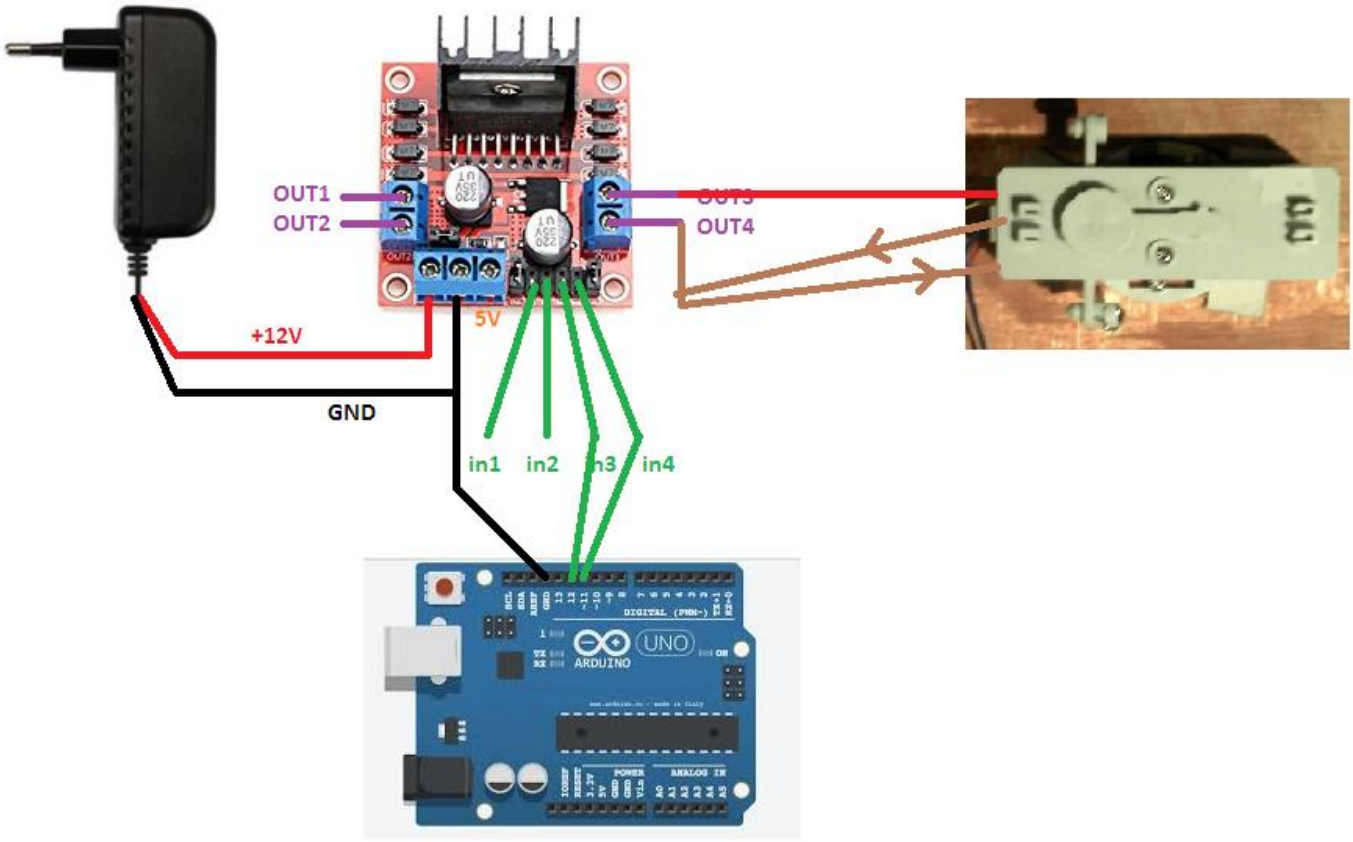
Pour mettre l'aiguillage en directe on mettra une entrée à la masse et l'autre à 5V et inversement pour la position complémentaire. On peut laisser l'alimentation en permanence car tous les aiguillages à moteur possèdent des contacts de fin de course qui coupe l'alimentation du moteur une fois la position souhaitée atteinte.

Il est également possible de commander des aiguillages à bobine comme des aiguillages à moteurs avec l'aide de 2 diodes, chacune alimentant une bobine suivant la polarité de la tension. Si le moteur n'est pas équipé d'un interrupteur de fin de course, alors il faudra générer des impulsions sous peine de griller les bobines. L'intérêt est qu'il n'y a besoin que 2 fils au lieu de 3 pour commander l'aiguillage

Si vous avez besoin de plus de courant, vous pouvez utiliser le L298 qui supporte jusqu'à 2A et fonctionne quasiment comme le L293D (à part qu'il faut mettre les broches d'activations « enable 1 & 2 » à la masse pour activer les sorties) . Attention, le L298 n'incorporent pas les DRL. Vous pouvez trouver des modules tout fait sur Amazon au prix de 4€. Des jumpers activent les sorties par défaut. Le module incorpore les DRL ainsi qu'un petit régulateur qui délivre du 5V, pratique si vous en avez besoin ailleurs.



L'exemple suivant commande un moteur Conrad avec ce module. Le module est surdimensionné avec son courant maximale de 2A alors que le moteur ne demande que 100mA mais je n'ai pas trouvé de modules à base de L293D. Pour mettre l'aiguillage en position directe, il faudra par exemple mettre in3 à 1 et in 4 à 0. Cela mettre 12V sur OUT3 et OUT4 à la masse et le moteur tournera dans un sens et sera coupé en fin de course grâce à l'interrupteur de fin de course relié à une diode sur l'un des 2 fils marron. Pour la position déviée, il faudra faire l'inverse, c'est-à-dire in3 à 0 et in4 à 1.



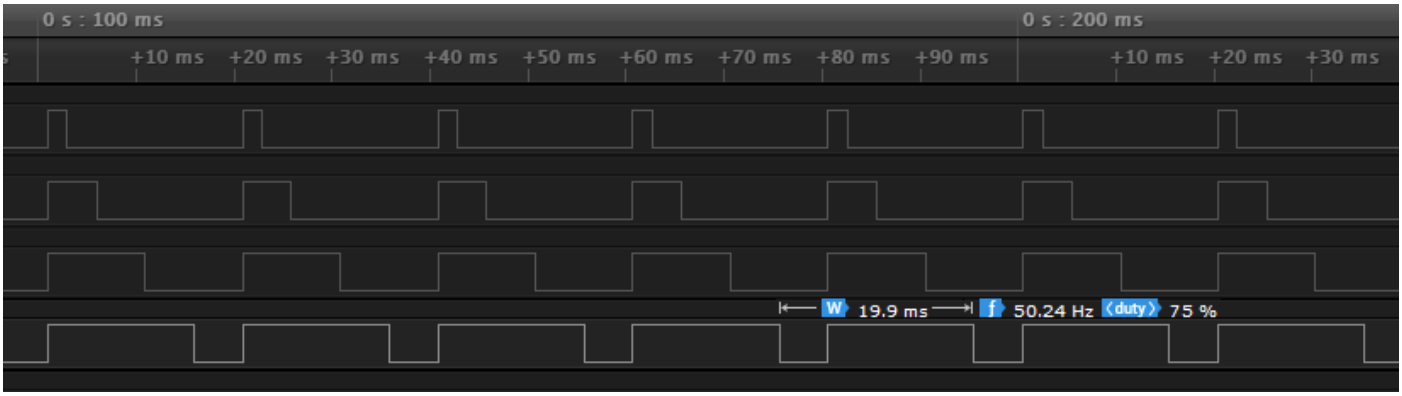
Comme déjà mentioné, vous pouvez aussi commander les moteurs avec des relais.

2.5. La PWM

La PWM pour modulation de largeur d'impulsion (Pulse Width Modulation) est une technique pour rendre une sortie « réglable ». Ce n'est pas une sortie ou la tension est réglable comme dans le cas d'une alimentation continue, mais une sortie qui génère un signal rectangulaire qui soit vaut la tension V+, soit 0V. En faisant varier la durée des impulsions entre 0 et 100%, on fait varier la tension moyenne entre 0 et V+.

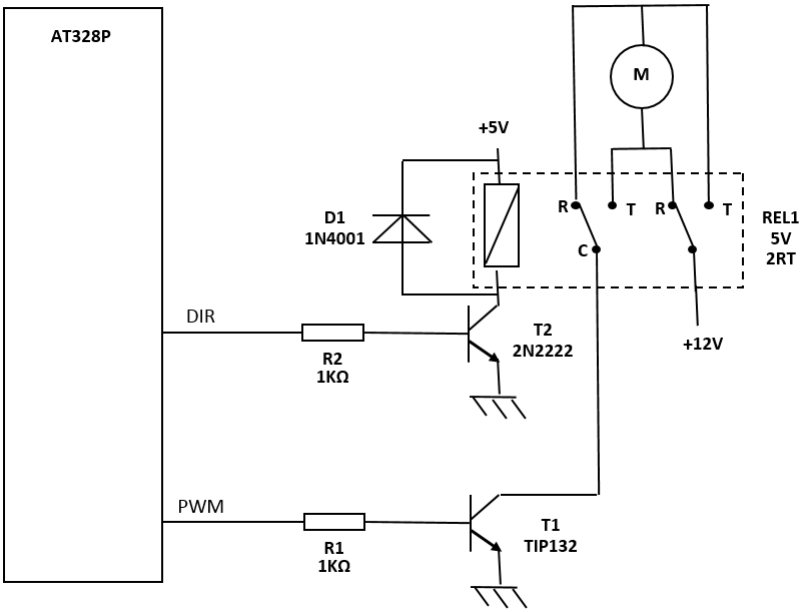
```
-------------------- 0%        (-=0V   #=V+)
#---------#--------- 10%
#####-----#####----- 50%
#########-#########- 90%
#################### 100%
```

Le chronogramme suivant a été capturé avec un analyseur logique sur 4 sorties PWM du PCA9685 réglées à 10%, 25%, 50% et 75%on utilise. La fréquence est de 50Hz car elle est globale à tout le PCA9685 et les servos ont besoin de cette fréquence.



Si on utilise la PWM pour contrôler une lampe par l'intermédiaire d'un transistor, alors on pourra faire varier l'intensité de cette lampe. Rien n'empêche d'utiliser une sortie PWM en sortie standard (0%=éteint et 100% allumé).

La PWM peut aussi être utilisée pour faire tourner un moteur plus ou moins vite. Il faudra rajouter une autre sortie pour le sens. Le schéma suivant montre une telle réalisation. La sortie DIR, contrôle le relais REL1 qui permet d'inverser le sens de rotation. La sortie PWM commande le « gros » transistor T1 qui amplifie le signal PWM est permet de régler la vitesse de rotation du moteur.

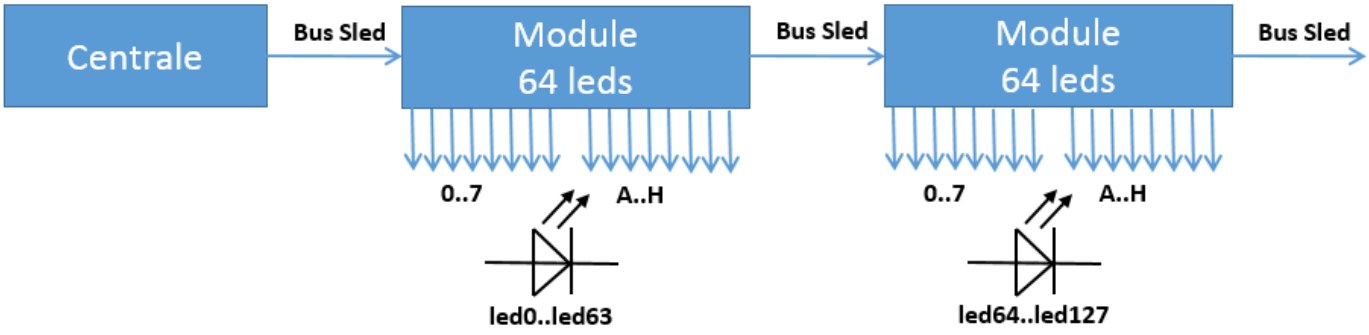


Il est possible de remplacer le montage précédent par un pont en H comme celui utilisé par le booster.
En effet, le booster peut également être utilisé pour contrôler un moteur en connectant une sortie PWM de l'Arduino à l'entrée PWM du booster afin de régler la vitesse. La boche DIR étant utilisée pour le sens de marche

L'avantage de la PWM sur une tension continue est que l'on ne dissipe quasiment pas de puissance dans le pont. L'inconvénient de cette technique est le bruit généré dans les moteurs par la fréquence de découpage qui s'entend ! La fréquence de la PWM des PCA9685 est réglée à 50Hz afin de pouvoir contrôler des servos. Les arduino UNO et Nano disposent de 6PWM tandis que le Mega en possède 12 mais afin de simplifier la compréhension, nous ne les utiliserons pas pour le moment.

2.6. Les LEDs

Le bus SPI (ou Sled) permet de raccorder jusqu'à 4 modules de 64 leds. Chaque module utilise un driver de LED MAX7219 ou 7221 capable de gérer 64 leds.

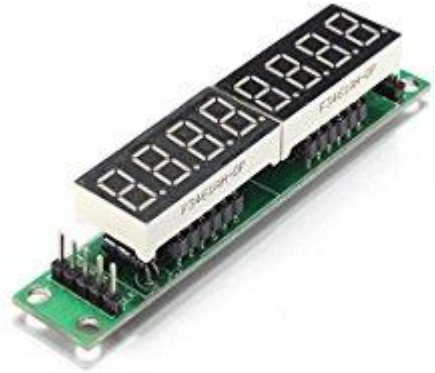


Un bus série de type SPI permet de communiquer avec les 7219. Chaque échange commence par la mise à l'état bas du signal d'activation LDLED (Load Led = chargement des LED) puis d'un décalage de 4 mots de 16 bits (un pour chaque MAX7219), puis se termine par une validation de l'échange par la mise à 1 de LDLED.

```
LDLED #####_______________  _____________###
CLK       _______#____#___  _#____#___  _#______
Data          3-15 3-14   3-00 2-15   0-00
```

Il est possible de configurer la luminosité de l'ensemble de LED de chaque 7219.

Vous pouvez réaliser des modules avec le circuit MAX7219 (fonctionne aussi avec le 7221) ou alors les acheter directement sur Amazon par exemple pour 4€. Il existe des modules avec une matrice de 64 LEDs (8*8). Il existe aussi des modules avec un afficheur comprenant 8 afficheurs à 7 segments. Il suffit d'enlever la matrice ou l'afficheur pour accéder au connecteur afin d'y connecter vos LEDs. La matrice ou l'afficheur, pouvant servir à tester votre montage.

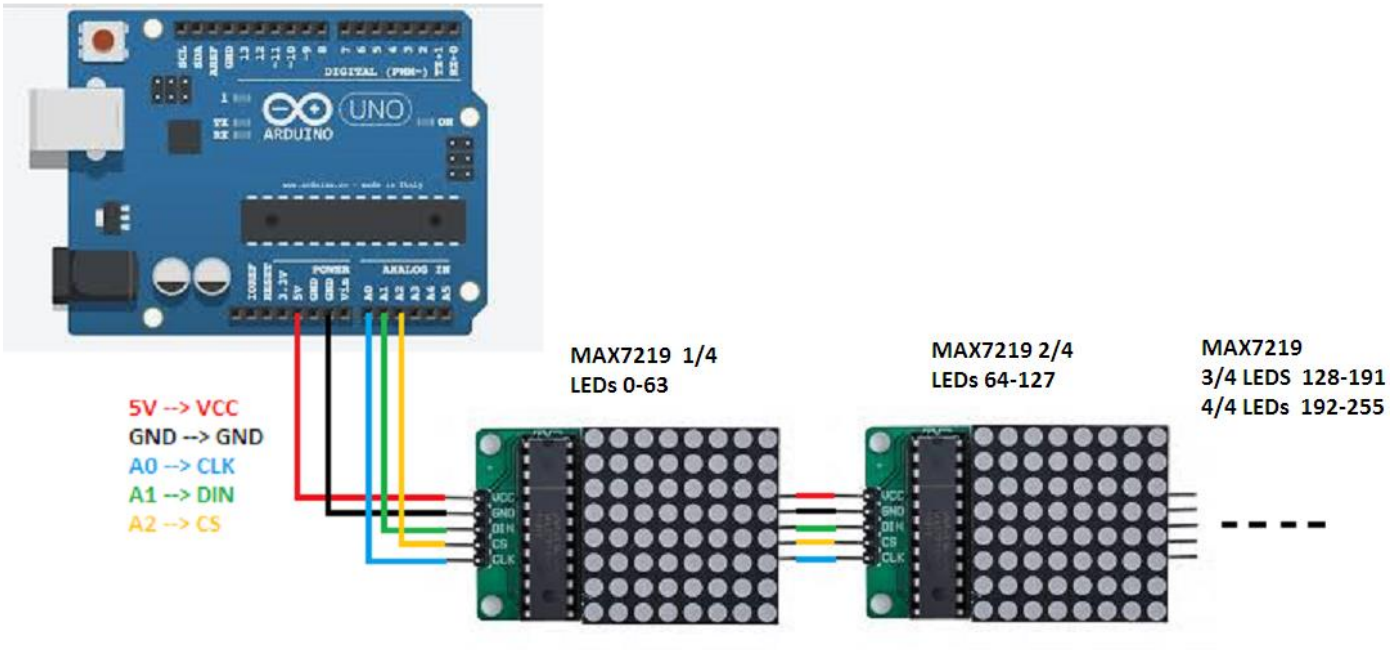


Le circuit commande les LEDs en courant, il n'y a donc pas besoin de rajouter une résistance en série avec les LEDs. L'intensité globale des LEDs peut être réglée par la programme. Ceci peut être utile pour voir la signalisation en plein jour et éviter qu'elle ne se transforme en lampadaire dans l'obscurité. Comme les LEDs sont commandés an matrice, vous disposez de 8 pattes pour les cathodes et 8 pour les anodes. Vous pouvez donc commander aussi bien des signaux à anodes ou cathodes communes.

Le programme permet aussi de faire clignoter les LEDs que vous voulez à 1Hz et d'inverser la phase si nécessaire.

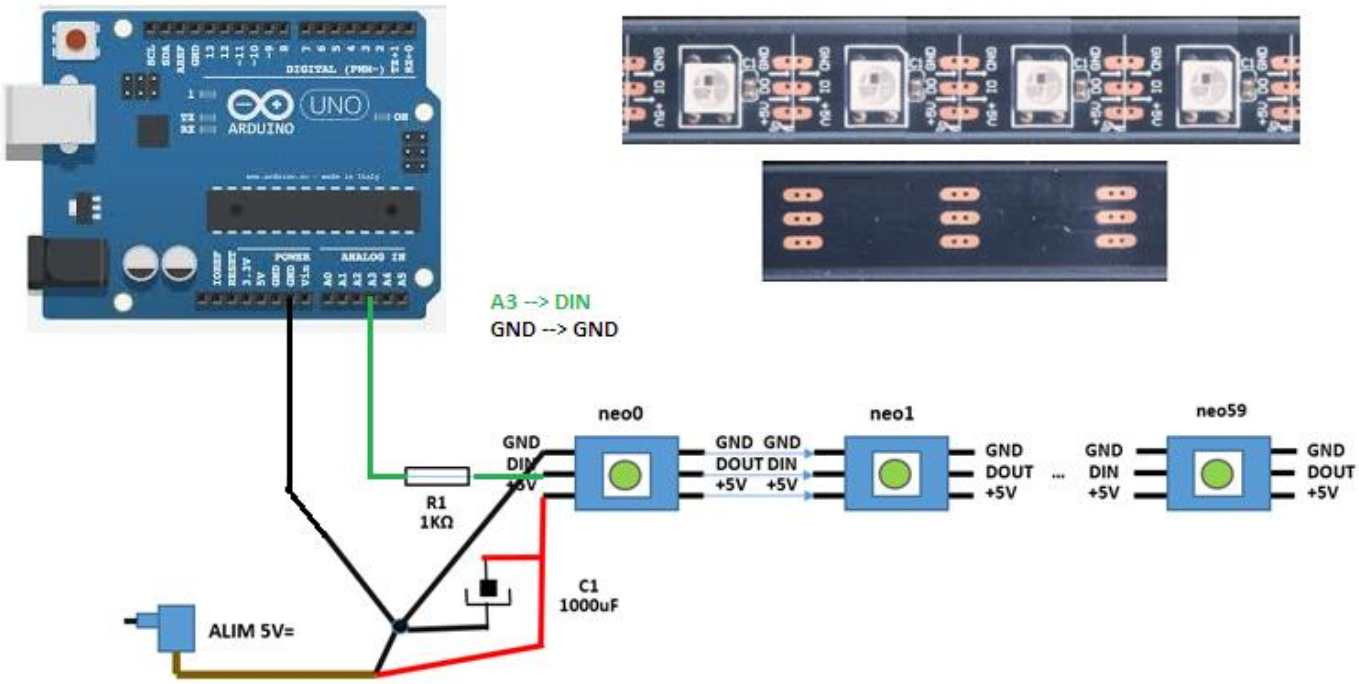
Si toutes les LEDs sont utilisées alors la consommation sera de 330mA. Si cela excède les capacités de l'Arduino, utiliser le 5V d'une autre alimentation. Penser à relier sa masse à celle de l'Arduino

Le schéma suivant montre comment connecter des modules MAX7219 à l'Arduino UNO.



2.7. Les Néopixels

Les néopixels sont des LEDs vraiment pratiques car on peut les chainer. (3 fils entre chaque LED). J'ai limité leur nombre à 60 car il faut pas mal de ressources pour les gérer. En plus d'être chainables elles sont multicolores (16millions de couleurs). Vous pourrez donc réaliser des animations uniques ! Malheureusement elles sont trop grosses pour rentrer dans les feux mais elles feront merveilles par exemple pour éclairer les bâtiments. De plus le système permet de générer des effets : (soudure à l'arc, néon qui scintille, néon cassé, feu de cheminé, TV …). Changer les couleurs peut être aussi utile, ex : blanc lorsqu'un bâtiment publiques est en service, vert très léger lorsqu'il est éteint et éclairé par les dispositifs de sortie de secours …



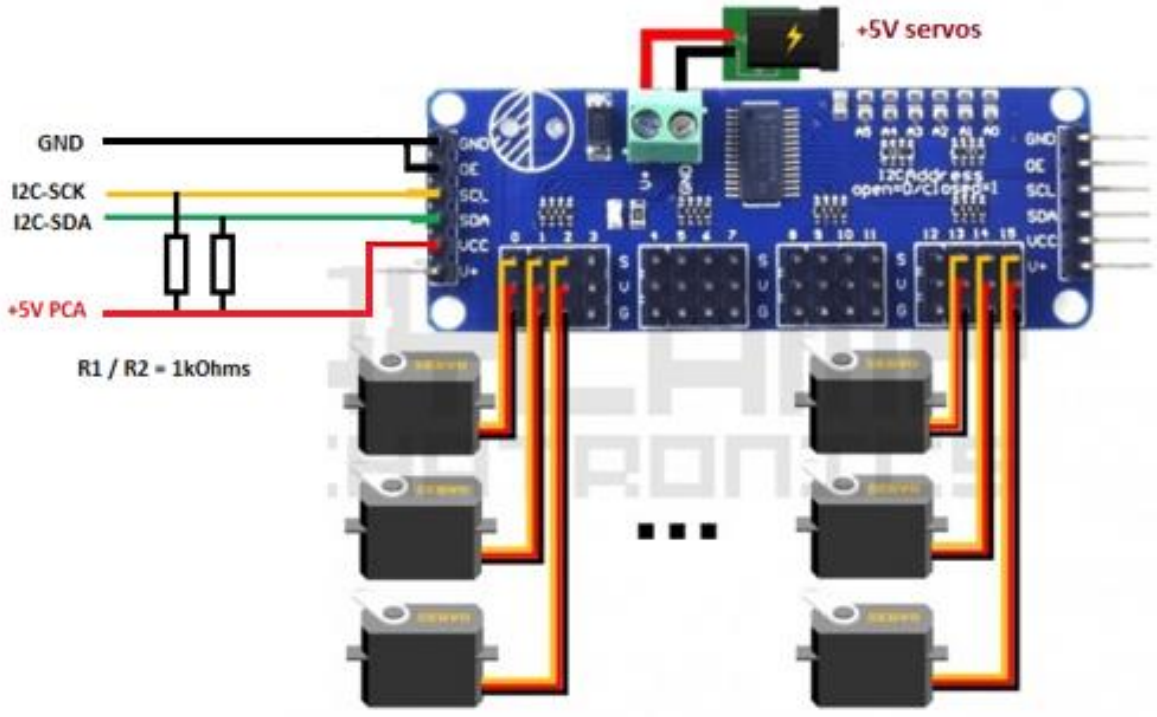
Chaque composante (rouge, vert ou bleue) consomme 20mA lorsqu'elle est à 100%. La consommation maximale est donc de 60mA par LED. A ce niveau-là, ça éclaire vraiment beaucoup ! Vu la consommation, il est conseillé d'utiliser une alimentation 5V indépendante. Sa puissance (donc son ampérage sera choisi en fonction de votre configuration. Par exemple si vous prévoyez d'allumer 60 LEDs à 100%, cela demandera 60 * 60mA = 3.6A ! Bien entendu, si vous avez besoin de 10 LEDs à 10% : 10 * 6mA = 60mA, vous pourrez vous passer d'une alimentation externe. Il est conseillé de rajouter un condensateur pour absorber les pics et rajouter une résistance pour éviter de détruire les vieux modèles. Les nouveaux WS2812 (à 4 pattes) ou WS2813 étant plus robustes. A puissance max, le néopixel peut chauffer, vérifier que cela est compatible avec votre maquette ou réduisez la puissance à un niveau acceptable !

A l'heure ou ces lignes sont écrites, il existe 4 types de néopixels :
- WS2811 qui est un circuit sur lequel il faut brancher une LED
- WS2812 ancien (à 6 pattes)
- WS2812 nouveau (à 4 pattes)
- WS2813 (à 6 pattes)

Tous fonctionneront, mais les mieux sont les WS2812 nouveau et WS2813 car ils sont plus robustes. Ils sont protégés contre les décharges et l'inversion de polarité. Les WS2813 possèdent un dispositif qui permet de sauter un pixel défectueux, mais la connexion et plus élaborée. Nous n'utiliserons pas cette fonctionnalité car nous voulons que tous nos pixels fonctionnent ! Les néopixels sont disponibles sous plusieurs formes. Pour ma part, je préfère les rubans de 30 ou 60 LEDs. En effet ces derniers sont économiques (0.5€ / LED) et ils peuvent être coupés.

2.8. Les PCA9685 sur bus I2C

Le bus I2C permet de relier tous les circuits I2C existants au décodeur. Cela est intéressant, car il existe un grand nombre de circuits I2C dans diverses domaines (entrées/sortie, led, pwm, servos, can, cna …). De plus, on peut en mettre plusieurs du même type. Les 2 signaux du bus I2C sont SDA et SCK. Actuellement, le décodeur supporte 6 PCA8596. Chaque circuit dispose de 16 sorties pouvant être utilisées individuellement en mode PWM ou Servo. Donc 96 sorties au total ! Vous trouverez sur Amazon des modules PCA9685 à 8€ ainsi que des micro-servos 9G à 2€ pièce afin de contrôler tout ce que vous voulez sur votre réseau



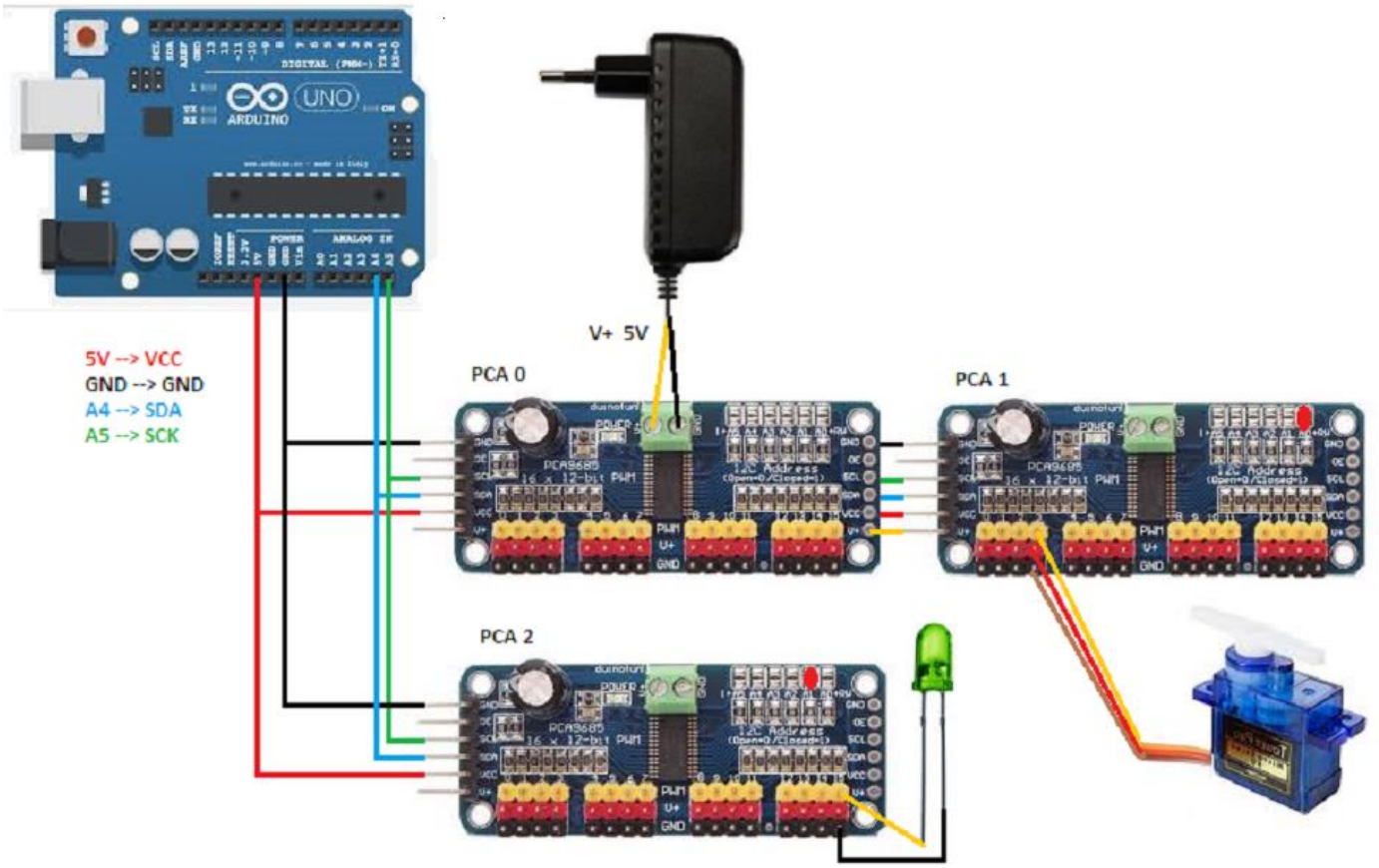
credit: image source: http://www.naylampmechatronics.com/blog/41_Tutorial-M%C3%B3dulo-Controlador-de-servos-PCA9685.html

Les résistances de pull-up du bus I2C du schéma ne sont pas obligatoire car chaque module contient déjà des pull-up de 10KΩ. Ajoutez les si vous constatez des problèmes suite à l'éloignement trop important du module par rapport au décodeur. Je génère moi-même les signaux I2C au lieu d'utiliser l'I2C de l'Arduino afin de réduire sa vitesse et augmenter la distance. Un bus I2C de 10m devrait fonctionner.

Pour utiliser plusieurs modules, il suffit de les brancher en // ou les chainer. Ne pas oublier de changer les adresses. (Ajoutez un point de soudure sur A0 pour le 2eme module, et point de soudure sur A1 pour le 3eme module, un point sur A0 et A1 pour le 4eme, A2 pour le 5eme, A2 et A0 pour le sixième)

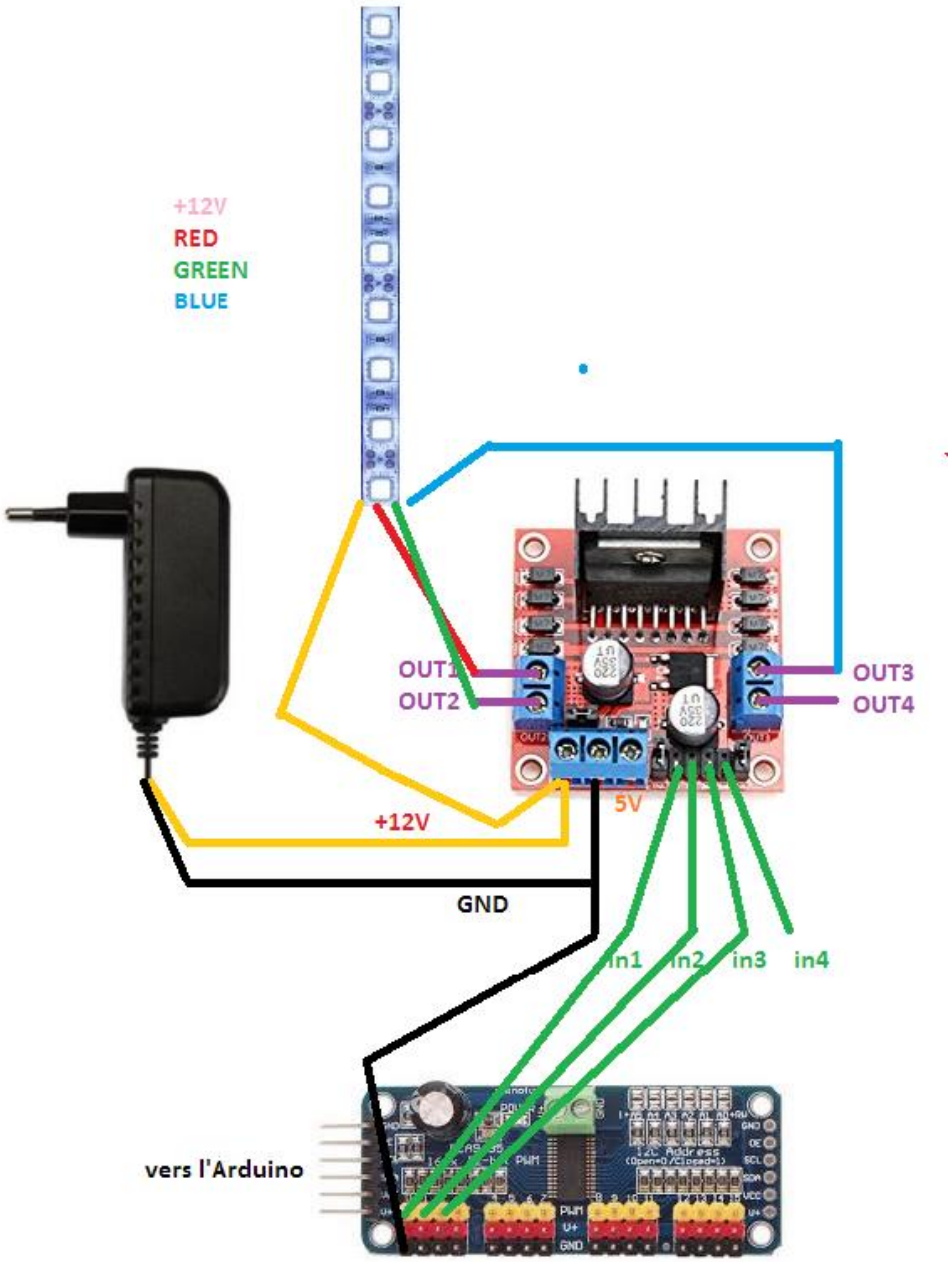
Pour utiliser des sorties au lieu des servos, utilisez les pattes jaunes. Ces sorties ne sont pas des sorties simples mais des sorties PWM (voire le chapitre PWM). Vous pouvez néanmoins les commander comme des sorties normales en mettant la PWM à 0% pour un état bas et à 100% pour un état haut.
Une résistance de 220Ω est en série avec chaque sortie, vous pouvez donc vous passer de résistances si vous contrôler des LEDs.

Ne pas utiliser le 5V de l'Arduino pour alimenter les servos car ils perturberaient ce dernier. L'alimentation +5V des servos se connecte sur le bornier à vis. La masse est commune avec les masses (GND) des connecteurs latéraux. Ce +5V nommé V+ est également disponible sur les connecteurs latéraux. Sur le schéma suivant ce 5V passe ainsi entre les cartes PCA0 et 1. Il est également disponible sur la broche centrale des 16 connecteurs des servos.
Le circuit est quant à lui alimenté par la broche VCC des connecteurs latéraux. C'est lui qui alimente aussi les broches PWM. Si vous utiliser 6 modules pour commander 96 leds cela tirera 96*15mA = 1.4A sur le 5V de l'Arduino ce qui est trop et il vous faudra donc alimenter la patte VCC par une nouvelle alimentation 5V en gardant une masse commune avec l'Arduino



Comme déjà expliqué, si vous commander plus que de LEDs (ou des servos), vous devrez utiliser des transistors ou relais pour commander des charges plus puissantes.

La figure suivante montre comment connecter un ruban de LEDs RVB pour éclairer votre réseau. Je l'ai testé avec un bandeau de 5m de 24W sous 12V. Attention, ce bandeau n'a rien à voir avec un bandeau néopixels. Toutes les LEDs rouges sont connectées entre elles et s'éclairent en même temps. Idem pour les LEDs vertes et bleues. Sur mon bandeau, les LEDs sont à anode commune, nous relierons donc le Vcc du bandeau au +12V. Les 3 pattes restantes R(Red=rouge), G(Green=vert), B(Blue=bleu) doivent être commutées à la masse pour que les LEDs s'allument. 24W = 8W / couleur = 8/12=660mA par couleur. C'est trop pour un ULN2803, limite pour un L293D surtout qu'en générale le bleu consomme plus que les autres couleurs. J'ai alors utilisé un L298 qui se pilote comme un L293D mais propose 2A max par canal. Il faudra inverser la commande de la PWM par logiciel car 0% allumera les LEDs aux max … Avec un tel bandeau vous pourrez éclairer votre réseau sous plusieurs couleurs et vous essayer aux effets spéciaux comme la foudre … Notez qu'un seul bandeau de 24W pour 5m n'est pas suffisant pour illuminer correctement un réseau.
Vous pouvez en utiliser d'autres ou utiliser une autre source pour le soleil comme un néon ou des lampes contrôlées en PWM ou pas …



2.9. Multiplexage

Pour économiser des sorties et électronique de puissance, il est possible de multiplexer !

Vous pouvez le faire si vous commander vos aiguillages un par un et lentement.

Par exemple, vous pouvez utiliser une sortie pour choisir la direction, une autre pour activer le mouvement, et une seule pour valider un aiguillage. Ainsi au lieu de commander 8 aiguillages avec 16 sorties, vous pouvez en commander 14. Il est également possible de multiplexer le choix de l'aiguillage. Ex avec 6 sorties, on peut commander 16 aiguillages. Attention, il faut utiliser des diodes pour ne pas commander les aiguilles non utilises. Le multiplexage complexifie cependant la compréhension, à vous de voir si ça vaut le coup ! Pour bien faire il faudrait aussi modifier le programme du décodeur afin qu'il mémorise les commandes et déplace les aiguillages un à un par la suite.

3. Le programme

Pour que votre décodeur D18 fonctionne, vous devez téléverser le programme d18.ino

Mais avant cela il faudra le configurer pour l'adapter exactement à vos besoins car par défaut il propose 16 sorties fixes utilisées 2 à 2 en paires aux adresse 10 et 11.

Avant cela, il est indispensable de comprendre les trames DCC pour les accessoires en théorie et comment les centrales les génère en réalité.

3.a. Les trames DCC pour la commande des accessoires

Ce chapitre décrit les trames DCC qui commande les décodeurs d'accessoires.

La norme DCC définit les 3 décodeurs suivants :

- « **Multifunction decoder** » : Décodeur de locomotive pour gérer un moteur et jusqu'à 28 = 1 fonctions auxiliaires. Ce décodeur est aussi connu sous le nom de « Mobile Decoder » car il bouge avec le loco en opposition aux « Stationary decoders » qui ne bougent pas. Il existe aussi des décodeurs de fonction qui sont en fait des décodeurs multifonctions mais sans la partie moteur. D18 ne gère pas ces décodeurs.

- « **Basic Accessory Decoder** » pour les accessoires (aiguillages, feux …)

- « **Extended Accesorry Decoder »** pour les accessoires un peu plus complexes.

D18 sait décoder les trames et extraire les commandes pour les 2 derniers décodeurs.

Les « Basic Accessory Decoders ».

Ces décodeurs sont les plus courants.

La trame DCC qui commande les accessoires basiques est la suivante :
1111111111 0 10AAAAAA 0 1AAACDDD 0 EEEEEEEE 1
Les bits A codent l'adresse du décodeur (0-511)
Les bits D indiquent la sortie (0-7)
Le bit C indique l'état à appliquer à la sortie (0-1)

Il est possible d'avoir 510 de ces décodeurs (adresses 1 à 510). L'adresse 0 n'est pas utilisée et la 511 est une adresse de diffusion afin d'envoyer le même ordre à tous les décodeurs. Ces décodeurs ne sont pas dans le même espace d'adressage que les décodeurs des locomotives, avoir un décodeur d'adresse 1 pour une loco et 1 pour un accessoire ne pose pas de problème. Idem entre les 2 types de décodeurs d'accessoires.

Chaque décodeur possède 8 sorties (numérotées de 0 à 7 sur A18, mais 1 à 8 sur certaines centrales)
Les sorties sont souvent gérées par paires souvent numérotées de 1 à 4. Les sorties de la paire sont souvent nommées rouge et verte. L'activation d'une sortie d'une paire désactive l'autre sortie. Cela convient bien pour des feux à 2 "aspects" (rouge ou vert) ou aiguillages. Il est aussi possible de générer des impulsions plutôt que d'avoir des sorties fixes, ce qui est essentiel pour les aiguillages à bobines sans contacts de fin de course. Pour les paires il n'y a pas besoin d'envoyer de commande de remise à 0.

Toutes les centrales arrivent normalement à gérer correctement les paires. Par contre, elles ont plus de mal avec les sorties individuelles car il faut gérer la remise à 0. Si c'est le cas de votre centrale, vous pouvez utiliser la commande

qui utilisée pour une paire mais gérer qu'une sortie. Par exemple désactiver la sortie si la commande demande l'activation de la première sortie de la paire et l'activer pour une demande d'activation de la 2eme sortie.

Pour des signaux « multi aspects » (à plus de 2 états) comme un signal de canton (vert/rouge/jaune) certains décodeurs proposent des « paires » à 3 sorties. Par exemple l'activation de la sortie 0 allume la led verte et éteint toutes les autres, l'activation de la sortie 0, allume la rouge et éteint toutes les autres, l'activation de la sortie 2, allume la jaune et éteint toutes les autres …

Dans un boitier, il peut y avoir plusieurs de ces décodeurs. Par exemple de base la A18 qui gère 16 sorties peut être utilisé comme 2 décodeurs basiques de 8 sorties chacun. Le décodeur peut également utiliser plus d'adresses si vous rajouter d'autres fonctions que le on/off.

D18 peut gérer tous les cas facilement.
La fonction **notify_bas_acc_dec(adr_1_510, out_0_7, val_0_1)** est appelée à chaque réception d'un paquet DCC pour un « Basic accessory decoder ». A vous ensuite de rajouter du code pour faire exactement ce que vous voulez. Vous pouvez réagir à n'importe quel ordre car vous avez accès aux 3 paramètres. A18 est donc générique et vous l'adaptez a votre centrale, vous pouvez ensuite faire ce que vous voulez

Vous pouvez relier votre D18 à un ordinateur via l'USB et utiliser le moniteur série de l'IDE Arduino pour voire toutes les trames que la centrale génère. Cela vous permettra de configurer le décodeur en conséquence.
Exemple d'affichage à la réception d'un paquet :
#3 138 253 119 ACC pkt adr(1-511)=10 out(0-7)=5 val(0-1)=1
Vous verrez aussi des paquets non décodés lorsqu'il ne s'agit pas de paquets pour les accessoires :
#3 255 0 255

Les « Extended Accesorry Decoder »

Ces décodeurs sont les moins connus.

La trame DCC qui commande les accesoires étendus est la suivante :
1111111111 0 10AAAAAA 0 0AAA0AA1 0 000CCCCC 0 EEEEEEEE 1
Les bits A codent l'adresse du décodeur (1-2044)
Les bit C indiquent la valeur de la commande (0-31)

Il est possible d'avoir 2044 de ces décodeurs (adresses 1 à 2044). Chaque décodeur peut gérer 1 appareil qui reçoit une commande entre 0 et 31. A l'origine, cela a été surtout fait pour gérer les feux "multi aspects". Ex commande 0 pour carre, 1 pour sémaphore, 2 pour avertissement, 3 pour voie libre … Cela peut aussi servir à piloter la position d'un servo ou la vitesse d'un moteur. Ce décodeur est aussi connu sous le nom de décodeur de sortie ou « Output Decoder »

La fonction notify_ext_dec(adr_1_2044, val_0_31) est appelée à chaque réception d'un paquet DCC pour un « Extended Accessory Decoder ». A vous ensuite de rajouter du code pour faire exactement ce que vous voulez.

Beaucoup de centrales, ne gèrent tout simplement pas ce mode et vous devrez vous débrouiller avec des commandes pour les décodeurs basiques.

3.b. Configuration du décodeur D18

Vous pouvez utiliser les fonctions suivantes pour contrôler votre électronique :

** Pour les sorties directes de la carte Arduino

```
user_out(out#<3-12,14-53,A0-A15>, val<0_1>)
```

** Pour les LEDs sur les MAX

```
user_led(led#<0-255>, val<0-1>)          //allumage/extinction
user_led_cli(led#<0-255>, val<0-1>)      //mode clignotant
user_led_pha(led#<0-255>, val<0-1>)      //inversion de la phase de clignotement
```

** Pour les sorties des PCA

```
user_pwm_0_100(out#<0-96>, <0-100%>)
```

Pour faire comme des sorties normales, utilisez 0% ou 100%

```
user_servo_500_2500(out#<0-96>, <500-2500>)
```

La durée des impulsions est spécifiée en microsecondes. 1500us=1.5ms=position médiane du servo

```
user_servo_speed(out#<0-96>, <vitesse>)
```

vitesse de rotation du servo (vitesse = variation des impulsions en us par pas de 128ms) (0=vitesse max)

** Pour les néopixels

```
user_neo(neo#<0-59>, r<0-255>, v<0-255>, b<0-255>)
```

** Pour lancer des tempos

```
user_tempo_start(num_tempo#<0-79>, duration_ms<0-60000>)
```

Cette fonction peut être utilisée pour réaliser des sorties impulsionnelles.
Cette fonction lance une tempo de la durée indiquée en ms (max 60s). 0 désactive la tempo.
En fin de tempo, la fonction `notify_tempo_end(pulse<0-79>)` est appelée.

Vous pouvez mettre votre code et appeler les fonctions précédentes à partir des fonctions suivantes :
(Si vous avez déjà programmer des Arduino, vous pouvez mettre du code ailleurs et faire tout ce que vous voulez)

```
void user_init(void)
{
}

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
}

void user_notify_ext_acc_dec(int adr_1_2044, byte val_0_31)
{
}

void user_notify_tempo_end(byte num_tempo)
{
}

void user_125ms(void)
{
}
```

Suivant que vous voulez utiliser ou non des modules MAX, des modules PCA ou des Néopixels vous devrez mettre à 1 ou 0 les constantes suivantes :

```
#define USER_USE_MAX  1   // 1 to use MAX7219/21 (out A0, A1 & A2 no more available)
#define USER_USE_PCA  1   // 1 to use PCA9685 (out A3 & A4 no more available)
#define USER_USE_NEO  1   // 1 to use Néopixels (out A5 no more available)
```

Pour ceux qui n'ont jamais programmé en langage C, vous devrez vous en sortir avec ces quelques règles :

- Utiliser // pour un commentaire (tout ce qui suit jusqu'en fin de ligne est un commentaire)
- L'instruction if exécute ce qui est entre {} si le test entre () est vraie.
- Pour tester si une variable vaut une value, il faut utiliser == (2x=) et non 1 seul = qui est l'affectation.
- Ne pas oublier le ; après chaque appel de fonction
- Si vous utiliser une constante souvent, vous pouvez la définir avec #define, ex #define AIG_3_DIRCET 5

3.c. Exemples

** Gestion d'une paire de sorties fixes sur l'Arduino

Pour ce premier exemple, on veut commander la paire D12 et D11 en sorties fixes. Ces sorties répondront aux commandes pour les sorties 0 et 1 du décodeur d'accessoire basique d'adresse 10. D12 sera la sortie rouge activée par la commande de sortie 0 et D11 la sortie verte.

```
void user_init(void)
{
    user_out(12,0); user_out(11,0);  //pas necessaire
}

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==10)
    {
        if(out_0_7==0) { user_out(12,val_0_1); user_out(11,0); }
        if(out_0_7==1) { user_out(11,val_0_1); user_out(12,0); }
    }
}

void user_notify_ext_acc_dec(int adr_1_2044, byte val_0_31)
{

}

void user_notify_tempo_end(byte num_tempo)
{

}

void user_125ms()
{

}
```

A l'initialisation, la fonction user_init() est appelée et vous pouvez mettre du code d'initialisation. Ici je mets les sorties à 0, mais ce n'est pas la peine car elles sont à 0 par défaut.

Lorsque le décodeur reçoit un paquet d'accessoire basique, la fonction user_notify_bas_acc_dec() est appelée. Si l'adresse du décodeur est 10 on fait la suite sinon on ne fait rien, car le paquet est par exemple pour un autre décodeur. Si la sortie 0 est sélectionnée alors on met à 1 la sortie D12. Et comme on fonctionne par paire, on met à 0 l'autre sortie de la paire à savoir D11. Maintenant si la sortie est la 1 alors on met D11 à 1 et D12 à 0. On ne met pas le D dans le numéro de sorties, par contre, on met le A …

Vous pouvez faire de même pour les autres sorties. Comme il n'y a que 8 sorties par adresse, il faudra utiliser 2 adresses. Par défaut, le programme du décodeur est livré avec le code suivant :

```
void user_init(void)
{
}

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==10)
    {
        if(out_0_7==0) { user_out(12,val_0_1); user_out(11,0); }
        if(out_0_7==1) { user_out(11,val_0_1); user_out(12,0); }
        if(out_0_7==2) { user_out(10,val_0_1); user_out( 9,0); }
        if(out_0_7==3) { user_out( 9,val_0_1); user_out(10,0); }
        if(out_0_7==4) { user_out( 8,val_0_1); user_out( 7,0); }
        if(out_0_7==5) { user_out( 7,val_0_1); user_out( 8,0); }
        if(out_0_7==6) { user_out( 6,val_0_1); user_out( 5,0); }
        if(out_0_7==7) { user_out( 5,val_0_1); user_out( 6,0); }
    }
    if(adr_1_510==11)
    {
        if(out_0_7==0) { user_out( 4,val_0_1); user_out( 3,0); }
        if(out_0_7==1) { user_out( 3,val_0_1); user_out( 4,0); }
        if(out_0_7==2) { user_out(A0,val_0_1); user_out(A1,0); }
        if(out_0_7==3) { user_out(A1,val_0_1); user_out(A0,0); }
        if(out_0_7==4) { user_out(A2,val_0_1); user_out(A3,0); }
        if(out_0_7==5) { user_out(A3,val_0_1); user_out(A2,0); }
        if(out_0_7==6) { user_out(A4,val_0_1); user_out(A5,0); }
        if(out_0_7==7) { user_out(A5,val_0_1); user_out(A4,0); }
    }
}

void user_notify_ext_acc_dec(int adr_1_2044, byte val_0_31)
{
}

void user_notify_tempo_end(byte num_tempo)
{
}

void user_125ms()
{
}
```

**** Gestion d'une paire de sorties impulsionnelles de 500ms sur l'Arduino**

```
void user_init(void)
{

}

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==10)
    {
        if(out_0_7==0) { user_out(12,val_0_1); user_out(11,0); user_tempo_start(0,500); }
        if(out_0_7==1) { user_out(11,val_0_1); user_out(12,0); user_tempo_start(0,500); }
    }
}

void user_notify_ext_acc_dec(int adr_1_2044, byte val_0_31)
{

}

void user_notify_tempo_end(byte num_tempo)
{
    if(num_tempo==0) { user_out(12,0); user_out(13,0); }
}

void user_125ms()
{

}
```

Le programme est identique au précédent, sauf que l'on lance la temporisation 0 pour 500ms. Vous avez 80 temporisateurs à votre disposition numérotées de 0 à 79. Après 500ms, la fonction user_notify_trmpo_end() est appelée. On teste s'il s'agit de la tempo 0, si oui, on met à 0 les 2 sorties de la paire. On a donc transformé nos sorties fixes en sorties impulsionnelles. Si vous gérer plusieurs paires, utilisez un temporisateur par paire. La résolution des temporisations est de 125ms. Il vaut mieux donc mettre des multiples de 125ms. Si vous mettez d'autres valeurs, il y aura un arrondi inférieur. Ne pas mettre entre 1 et 124 car cela ne générera aucune impulsion. 125 à 249 générera une impulsion de 125ms … 0 annulera une tempo en cours.

Avant de brancher un aiguillage à bobines sans contact de fin de course, il est conseillé d'essayer avec des LEDs et voire si on a bien des impulsions, car si vous vous trompez et alimentez une bobine en permanence, elle va finir par cramer …

**** Gestion d'une paire de sorties fixes sur le second PCA**

```
void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==12)
    {
        if(out_0_7==0) { user_pwm_0_100(16,100*val_0_1);  user_pwm_0_100(17,0);}
        if(out_0_7==1) { user_pwm_0_100(16,0); user_pwm_0_100(17,100*val_0_1); }
    }
}
```

A partir de maintenant, je ne mets plus les fonctions non utilisées.

Il y a 16 sorties par PCA. Elles sont numérotées :

- 0 à 15 pour le premier PCA (celui d'adresse I2C 0)
- 16 à 31 pour le second PCA (celui d'adresse I2C 1)
- 32 à 47 pour le PCA suivant (celui d'adresse I2C 2)
- 48 à 63 pour le PCA suivant (celui d'adresse I2C 3)
- 64 à 79 pour le PCA suivant (celui d'adresse I2C 4)
- 80 à 95 pour le PCA suivant (celui d'adresse I2C 5)

On utilisera les 2 premières sorties du second PCA. Pour faire des sorties 0/1 sur le PCA, on utilisera les sorties PWM en 0% ou 100% uniquement. Pour des sorties impulsionnelles, on rajoutera une tempo.

**** Gestion de sorties individuelles**

```
void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==10)
    {
        if(out_0_7==0) { user_out(12,val_0_1); }
        if(out_0_7==1) { user_out(11,val_0_1); }
        if(out_0_7==2) { user_out(10,val_0_1); }
        if(out_0_7==3) { user_out(9,val_0_1); }
        if(out_0_7==4) { user_out(8,val_0_1); }
        if(out_0_7==5) { user_out(7,val_0_1); }
        if(out_0_7==6) { user_out(6,val_0_1); }
        if(out_0_7==7) { user_out(5,val_0_1); }
    }
}
```

On ne gère plus les sorties par paire, mais individuellement. La centrale enverra donc un message de mise à 1 pour activer la sortie et un autre de mise à 0 pour la désactiver. Cela est différent du mode de fonctionnement par paire ou seul des messages de mise à 1 sont envoyés car active rune sortie désactive l'autre. Quelques rares centrales peuvent envoyer des mises à 0 en mode paire pour faire des sorties impulsionnelles, mais il est plus sûr de laisser au décodeur la tache de réaliser les impulsions.

Toutes les centrales ne sont pas capables de gérer, les sorties, individuellement. Dans ce cas, vous pouvez utiliser les commandes de la paire pour gérer une seule sortie. Bien entendu comme une commande consomme 2 sorties de commandes pour une sortie réelle, il faudra doubler les adresses.

```
void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==10)
    {
        if(out_0_7==0) { user_out(12,0); }
        if(out_0_7==1) { user_out(12,1); }
        if(out_0_7==2) { user_out(11,0); }
        if(out_0_7==3) { user_out(11,1); }
        if(out_0_7==4) { user_out(10,0); }
        if(out_0_7==5) { user_out(10,1); }
        if(out_0_7==6) { user_out(9,0); }
        if(out_0_7==7) { user_out(9,1); }
    }
    if(adr_1_510==11)
    {
        if(out_0_7==0) { user_out(8,0); }
        if(out_0_7==1) { user_out(8,1); }
        if(out_0_7==2) { user_out(7,0); }
        if(out_0_7==3) { user_out(7,1); }
        if(out_0_7==4) { user_out(6,0); }
        if(out_0_7==5) { user_out(6,1); }
        if(out_0_7==6) { user_out(5,0); }
        if(out_0_7==7) { user_out(5,1); }
    }
}
```

**** Gestion d'un aiguillage à servo**

```
void user_init(void)
{
    user_servo_500_2500(2,1000); // aiguillage direct avec une commande de 100x10=1000us=1ms
    user_servo_speed(2,40);      // reglage de la vitesse a 40us toutes les 125ms
                                 // donc 2000-1000=1000us en 3.125s
}

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==10)
    {
        if(out_0_7==0) { if(val_0_1) { user_servo_500_2500 (2,1000); } }
        if(out_0_7==1) { if(val_0_1) { user_servo_500_2500 (2,2000); } }
    }
}
```

Un servo est normalement dans sa position médiane pour une impulsion de 1.5ms (1500us).
Il est normalement à -90° pour 1ms et +90° pour 2ms.
Il est souvent possible de le faire tourner plus avec 0.5ms et 2.5ms, mais attention à ne pas le faire forcer en butée. (On entend un zzzzzz alors que le servo ne tourne plus).
Dans notre exemple, nous utilisons 1ms pour la position directe et 2ms pour la position déviée.
Par défaut, la vitesse de rotation est la vitesse max du servo. Pour avoir un mouvement lent, on fixe la vitesse avec la fonction user_servo_speed() en spécifiant la vitesse de variation de l'impulsion. Avec 40us toutes les 125ms, pour faire varier l'impulsion de 1000us à 2000us, il faut donc : (2000-1000)/(40/0.125)=3.125s
On utilise la sortie 2 du PCA d'adresse 0.
L'aiguillage est mis en position directe à l'initialisation.
Pour être compatible avec toutes les centrales, on le gère en mode paire.

Si l'on a un aiguillage à cœur métallique et que l'on n'a pas d'inverseur de polarité, on peut par exemple utiliser une sortie pour piloter un relais en conséquence. Comme on utilise la sortie 2, on peut par exemple utiliser le sortie 3 pour cela

```
... { user_servo_500_2500 (2,1000); user_pwm_0_100(3,0); }
... { user_servo_500_2500 (2,2000); user_pwm_0_100(3,100); }
```

**** Gestion d'un signal multi aspects**

Dans le jargon DCC, on appelle signal multi-aspect, un signal qui peut prendre plus de 2 indications. C'est le cas par exemple d'un carré qui peut présenter 4 aspects (C,S,A,VL). Nous utiliserons le même mécanisme que pour les paires mais appliqué à 4 sorties. Nous utiliserons des LEDs connectées à un MAX

```
#define C101_R2 8
#define C101_V  9
#define C101_R 10
#define C101_J 11

void user_init(void)
{
    user_led(C101_R2,1); //carre par defaut
    user_led(C101_R,1);
}

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==44 && val_0_1)
    {
        If(out_0_7>=0 && out_0_7<=3)
        {
             user_led(C101_R2,0); user_led(C101_V,0); user_led(C101_R,0); user_led(C101_J,0);
             if(out_0_7==0) { user_led(C101_R2,1); user_led(C101_R,1); }  //carre
             if(out_0_7==1) { user_led(C101_R,1); }  //semaphore
             if(out_0_7==2) { user_led(C101_J,1); }  //avertissement
             if(out_0_7==3) { user_led(C101_V,1); }  //voie libre
        }
    }
}
```

Afin de faciliter la compréhension, nous utilisons quelques constantes pour identifier les LEDs
Dans les tests vous remarquerez le &&, cela signifie que le test est vrai si les 2 expressions sont vraies.
Une expression sans test comme val_0_1 est équivalente à val_0_1<>0

Lorsque le décodeur 44 reçoit une commande de mise à 1 pour la sortie 0, 1, 2 ou 3, il commence par éteindre les 4 LEDs puis active les leds suivant la commande de sortie. Par exemple l'activation de la sortie 0, allume les 2 LEDs rouges du carré, la 1 la LED rouge du sémaphore, la 2 la LED jaune de l'avertissement et la 3 la LED verte de voie libre.

Si vous voulez commander un second carré à la même adresse, vous pouvez par exemple commander les sorties 4 à 7 de l'adresse 44. Notez qu'il n'y a aucun rapport entre le numéro de la sortie qui est dans le paquet et le numéro de la LED commandée.

**** Gestion d'un signal multi aspects avec un décodeur étendu**

Les décodeurs étendus ont justement été créés à l'origine pour contrôler des signaux multi aspects. Les adresses des décodeurs basiques et étendus ne sont pas dans le même espace, ce qui veut dire que vous pouvez avoir un décodeur basique à l'adresse 10 pour faire quelque chose et un décodeur étendu à l'adresse 10 pour faire autre chose. Chaque décodeur étendu reçoit une valeur entre 0 et 31 pour son adresse.

```
#define C101_R2 8
#define C101_V  9
#define C101_R 10
#define C101_J 11

void user_init(void)
{
    user_led(C101_R2,1); user_led(C101_R,1); //carre par defaut
}

void user_notify_ext_acc_dec(int adr_1_2044, int val_0_31)
{
    if(adr_1_510==49)
```

```
    {
         user_led(C101_R2,0); user_led(C101_V,0); user_led(C101_R,0); user_led(C101_J,0);
         if(val_0_31==0) { user_led(C101_R2,1); user_led(C101_R,1); }  //carre
         if(val_0_31==1) { user_led(C101_R,1); }  //semaphore
         if(val_0_31==2) { user_led(C101_J,1); }  //avertissement
         if(val_0_31==3) { user_led(C101_V,1); }  //voie libre
        }
    }
}
```

La valeur 0 affiche le carré, 1 le sémaphore, 2 l'avertissement, 3 la voie libre. Tout autre valeur (4-31) éteint la cible. La norme DCC conseille de mettre les signaux dans l'état bloqué pour la valeur 0 mais vous pouvez faire ce que vous voulez

**** Gestion d'un signal mécanique**

Notre signal mécanique sera un carré composé de :
- 1 servo pour faire tourner la cible
- 3 leds (2 rouges pour le carre et une verte voie libre)

```
#define C4_SERVO   5
#define C4_LED_R1 32
#define C4_LED_R2 33
#define C4_LED_V  34

void user_init(void)
{
    user_servo_500_2500(C4_SERVO, 1500); //carre ferme
    //user_servo_speed(C4_SERVO,10);     //reglage de la vitesse non supporte actuellement
    user_led(C4_LED_R1,1);
    user_led(C4_LED_R2,1);
    user_led(C4_LED_V,0);
}

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==16 && val_0_1)
    {
        if(out_0_7==3)  // ordre de fermeture
        {
            user_servo_500_2500(C4_SERVO, 1500);
            user_led(C4_LED_R1,1);
            user_led(C4_LED_R2,1);
            user_led(C4_LED_V,0);
        }
        if(out_0_7==4) // ordre d'ouverture
        {
            user_servo_500_2500(C4_SERVO, 1000);
            user_led(C4_LED_R1,0);
            user_led(C4_LED_R2,0);
            user_led(C4_LED_V,1)
        }
    }
}
```

**** Gestion d'itinéraires**

Les aiguillages peuvent être pilotés un par un, mais il est également possible d'en commander plusieurs à la fois pour former un itinéraire. On peut par exemple utiliser un décodeur étendu et commander les aiguillages en fonction du numéro de l'itinéraire contenu dans la valeur. Il est aussi possible d'utiliser un décodeur basique et d'activer un itinéraire suivant le numéro de sortie envoyé. Dans cet exemple pour compliquer un peu nous utiliserons des aiguillages à moteurs lents commandés chacun par un relais. La bobine de ces relais sont 5V mais ils sont alimentés en 12V, nous utiliserons donc des sorties pwm à 42% pour réduire la tension moyenne de 12V a 5V)

Voici le plan des voies :

```
    /---------------- iti0
--AIG0--AIG1--------- iti1
          \---AIG2--- iti2
                \---- iti3
```

```
#define AIG0 0
#define AIG1 1
#define AIG2 2
#define AIG3 3

void user_notify_ext_acc_dec(int adr_1_2044, byte val_0_31)
{
    if(adr_1_2044==45)
    {
        if(val_0_31==0) { user_pwm(AIG0,42); }  //iti0
        if(val_0_31==1) { user_pwm(AIG0,0); user_pwm(AIG1,0); }  //iti1
        if(val_0_31==2) { user_pwm(AIG0,0); user_pwm(AIG1,42); user_pwm(AIG2,0);  }  //iti2
        if(val_0_31==3) { user_pwm(AIG0,0); user_pwm(AIG1,42); user_pwm(AIG2,42); }  //iti3
    }

}
```

Cet gestion des itinéraires est plutôt basique, une bonne centrale ou un logiciel de control sur PC gérera certainement de manière plus complexe et réaliste les itinéraires en incluant la signalisation, l'interdiction d'établir des itinéraires incompatibles, la mémorisation … Dans ce cas le décodeur sera juste utilisé pour commander les aiguillages un à 1.

Il est possible de mixer différents types d'aiguillages comme des aiguillages à bobines, moteurs ou servos.

**** Gestion d'un passage à niveau**

Dans cette exemple le passage à niveau sera animé par :
- 2 servos branchés sur la même sortie (18) du PCA (3eme sortie du 2eme PCA)
- 4 leds (2 de chaque côté de la route qui clignotent de façon alternative)
- 1 sortie qui commande une sonnerie

```
#define PN_SERVO 18
#define PN_DRING 0
#define PN_LED_AV_DROITE 10
#define PN_LED_AV_GAUCHE 11
#define PN_LED_AR_DROITE 12
#define PN_LED_AR_GAUCHE 13

void user_init(void)
{
    user_servo_500_2500(PN_SERVO, 1500);//barrières fermées
    user_servo_speed(PN_SERVO, 30);     //vitesse 30us/125ms 1500-1000=500us 500/(30/0.125)=2.1s

    user_led_cli(PN_LED_AV_DROITE, 1);
    user_led_cli(PN_LED_AV_GAUCHE, 1); user_led_phase(PN_LED_AV_GAUCHE, 1);
    user_led_cli(PN_LED_AR_DROITE, 1); userled_phase(PN_LED_AR_DROIT, 1);
    user_led_cli(PN_LED_AR_GAUCHE, 1);
    user_led(PN_LED_AV_DROITE, 1);
    user_led(PN_LED_AV_GAUCHE, 1);
    user_led(PN_LED_AR_DROITE, 1);
    user_led(PN_LED_AR_GAUCHE, 1);
}

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510 == 15 && val_0_1)
    {
        if(out_0_7 == 0)  // ordre de fermeture
        {
            user_led(PN_LED_AV_DROITE, 1);
            user_led(PN_LED_AV_GAUCHE, 1);
            user_led(PN_LED_AR_DROITE, 1);
```

```
            user_led(PN_LED_AR_GAUCHE, 1);
            user_out(PN_DRING, 1);
            user_tempo_start(10, 2000);
        }
        if(out_0_7==1) // ordre d'ouverture
        {
            user_servo_500_2500(PN_SERVO, 1000);
            user_tempo_start(12, 2000);
        }
    }
}

void user_notify_tempo_end(byte tempo_num)
{
    if(tempo_num == 10)
    {
        user_servo_500_2500(PN_SERVO, 1500);
        user_tempo_start(11, 2000);
    }
    if(tempo_num == 11)
    {
        out(PN_DRING, 0);
    }
    if(tempo_num == 12)
    {
            user_led(PN_LED_AV_DROITE, 0);
            user_led(PN_LED_AV_GAUCHE, 0);
            user_led(PN_LED_AR_DROITE, 0);
            user_led(PN_LED_AR_GAUCHE, 0);
    }
}
```

Cet exemple gère le passage à niveau de manière réaliste grâce aux temporisations. Ainsi à la fermeture, les leds clignotent et la sonnerie sonne, après 2 secondes, les barrières se baissent puis 2 secondes plus tard la sonnerie se coupe. A l'ouverture, les barrières s'ouvrent puis 2 secondes plus tard les leds s'éteignent.

**** ex de commande d'un bandeau RGB**

Nous allons maintenant commander un bandeau RVB afin d'éclairer le réseau sous différentes couleurs.
Le commun du bandeau est alimenté en 12V, les leds R(rouges) sont commandées par la sortie pwm0 par l'intermédiaire d'un amplificateur, les leds G sur pwm1 et B sur pwm2. Comme on le voie si on met les sorties à 0 alors les leds s'allument. Il faudra donc inverser les commandes. Comme nous utiliserons la pwm, nous inverserons les commandes par 100-pwm.

```
void bandeau_rvb(byte r, byte v, byte b)
{
    user_pwm_0_100(0, 100-r);
    user_pwm_0_100(0, 100-v);
    user_pwm_0_100(0, 100-b);
}


void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==63 && val_0_1)
    {
        if(out_0_7==0) { bandeau_rvb(  0,  0,  0); } //eteint
        if(out_0_7==1) { bandeau_rvb( 25, 25, 25); } //jour 25%
        if(out_0_7==2) { bandeau_rvb( 50, 50, 50); } //jour 50%
        if(out_0_7==3) { bandeau_rvb( 75, 75, 75); } //jour 75%
        if(out_0_7==4) { bandeau_rvb(100,100,100); } //jour 100%
        if(out_0_7==5) { bandeau_rvb( 25, 12,  0); } //couche soleil rouge/jaune
        if(out_0_7==6) { bandeau_rvb(  0, 25,  0); } //aube verte
        if(out_0_7==7) { bandeau_rvb(  0,  0, 25); } //aube bleue
    }
}
```

Pour nous simplifier la vie, nous avons créé une fonction pour mettre la couleur souhaitée sur le bandeau.
Suivant la sortie activée dans la commande, l'éclairage sera différent. Byte signifie une variable 8 bits positives dons de valeurs extrêmes 0-255. Si vous avez besoin de plus vous pouvez utiliser int par exemple.

**** contrôle d'un servo et d'une sortie pwm**

Dans les exemples précédents, nous commandions les servos suivant 2 positions. Dans cet exemple nous commendons un servo avec un décodeur étendu afin d'avoir 32 positions. Nous faisons de même pour une sortie pwm.

```
void user_notify_ext_acc_dec(int adr_1_2044, byte val_0_31)
{
    int val2;

    if(adr_1_2044 == 45)
    {
        val2 = map(val, 0, 31, 1000, 2000);
        user_servo_500_2500(13, val2);
    }

    if(adr_1_2044 == 46)
    {
        val2 = map(val, 0, 31, 0, 100);
        user_pwm_0_100(14, val2);
    }
}
```

Dans cet exemple, nous utilisons une variable intermédiaire que nous déclarons en int.
La fonction map est très pratique car elle permet de convertir une valeur d'une plage sur une autre plage.
Dans le cas du servo nous voulons une impulsion de 1000us pour une commande de 0 et 2000us pour une commande de 31. Et une impulsion entre 1ms et 2ms pour les valeurs intermédiaires.

**** Eclairage d'un bâtiment avec un néopixel**

Pour éclairer l'intérieur de la gare, nous allons utiliser un néopixel. Nous définissons 4 états :

- Eclairage éteint
- Eclairage blanc
- Eclairage jaune (obtenu en mélangeant du rouge et du vert)
- Très faible lumière verte pour simuler les indicateurs d'évacuation.

Nous réduisons l'éclairage car au maximum (R=255, V=255, B=255), un néopixel éclaire trop
Nous utilisons le 3eme néopixel du bandeau, son numéro est donc le 2.

```
void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==33 && val_0_1)
    {
        if(out_0_7 == 0) { user_neo(2,  0,  0,  0);  }  //eteint
        if(out_0_7 == 1) { user_neo(2,128,128,128);  }  //blanc 50%
        if(out_0_7 == 2) { user_neo(2,128,128,  0);  }  //jaune 50%
        if(out_0_7 == 3) { user_neo(2,  0,  1,  0);  }  //vert tres faible
    }
}
```

**** Animations routières**

Nous allons maintenant faire quelques animations routières :

- Un feu avec un passage piéton
- Un bandeau de virage dangereux à 3 leds
- Un flash de radar

Pour cela nous mettrons notre code dans la fonction user_125ms qui est appelée toute les 125ms, soit 8 fois par seconde. Nous utiliserons également quelques compteurs. C'est compteurs seront déclarés en dehors des fonctions car ils doivent être conservés entre les appels des fonctions. Ce ne sont pas de variables temporaires et internes à la fonction. On parle de variables globales.

Le flash du radar pourra être activé par une commande. A la mise sous tension il sera actif. Nous mémoriserons sont état dans la variable radar_on.

Chronogrammes :

```
Radar : ---F------              un flash de 250ms toutes les  10s
Virage: 123-                    allumages des LEDs a tour de role
Feu   : V..........JR....V.
Pieton: R............V..R..
```

```
#define LED_RADAR 20
#define LED1_VIRAGE 21
#define LED2_VIRAGE 22
#define LED3_VIRAGE 23
#define FEU_R 24
#define FEU_J 25
#define FEU_V 26
#define PIETON_R 27
#define PIETON_V 28

byte radar_on = 1;
int cpt_virage = 0;
int cpt_radar = 0;
int cpt_feu = 0;
int cpt = 0;

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==4 && val_0_1)
    {
        if(out_0_7==0) { radar_on = 0;  }  //eteint
        if(out_0_7==1) { radar_on = 1;  }  //flash
    }
}

void virage(void)
{

    // incrementation du compteur cpt_virage et remise a 0 apres 4 secondes
    cpt_virage ++;
    if(cpt_virage == 4) { cpt_virage = 0; }

    // gestion des LEDs suivant le compteur cpt_virage (1s LED1, 1s LED2, 1s LED3, 1s rien
    if(cpt_virage == 0){ user_led(LED1_VIRAGE,1); user_led(LED1_VIRAGE,0); user_led(LED1_VIRAGE,0); }
    if(cpt_virage == 1){ user_led(LED1_VIRAGE,0); user_led(LED1_VIRAGE,1); user_led(LED1_VIRAGE,0); }
    if(cpt_virage == 2){ user_led(LED1_VIRAGE,0); user_led(LED1_VIRAGE,0); user_led(LED1_VIRAGE,1); }
    if(cpt_virage == 3){ user_led(LED1_VIRAGE,0); user_led(LED1_VIRAGE,0); user_led(LED1_VIRAGE,0); }
}

void feu(void)
{

    // incrementation du compteur cpt_feu et remise a 0 apres 30 secondes
    cpt_feu ++;

    // gestion des LEDs suivant le compteur cpt_feu
    if(cpt_feu == 20){ user_led(FEU_V,0); user_led(FEU_J,1); }
    if(cpt_feu == 21){ user_led(FEU_J,0); user_led(FEU_R,1); }
    if(cpt_feu == 22){ user_led(PIETON_R,0); user_led(PIETON_V,1); }
    if(cpt_feu == 27){ user_led(PIETON_R,1); user_led(PIETON_V,0); }
    if(cpt_feu == 28){ user_led(FEU_R,0); user_led(FEU_V,1);  cpt_feu = 0;}
}

void user_125ms(void)
{
    //radar
    cpt_radar = cpt_radar + 1;
    if(cpt_radar == 80) { cpt_radar = 0; }   //80*125ms=10sec
    if(cpt_radar == 20 && radar_on) { user_led(LED_RADAR,1); }
    if(cpt_radar == 22) { user_led(LED_RADAR,0); }   //flash dure 2x125=250ms

    //appelle des fonctions virage et feu toutes les secondes
    cpt++;
    if(cpt == 8) { cpt=0; virage(); feu(); }
}
```

cpt++; est identique à cpt=cpt+1;

** Effets spéciaux lumineux pour simuler la soudure à l'arc / un néon cassé / un feu de cheminé / une télévision

Il est possible de générer des effets spéciaux lumineux comme la soudure à l'arc, un néon qui scintille, un néon cassé, un feu de cheminé, une TV … Ceci permet de rendre votre réseau plus vivant ! Pour les effets basiques vous pouvez utiliser une led (soudure à l'arc, néon cassé). Pour des effets un peu plus poussés, vous pouvez utiliser la PWM car elle permettra de choisir entre plusieurs niveaux d'intensité au lieu d'avoir juste les états allumé ou éteint. Vous pouvez ainsi rajouter le scintillement d'un néon. Enfin avec les néopixels tout devient possible car la couleur et l'intensité peuvent varier. Vous pourrez ainsi rajouter des feux de cheminé ou des télévisions !

Ces effets peuvent être permanent ou alors être activable par une commande accessoire DCC. L'exemple du radar précédent montre comment piloter un effet par la centrale. Pour plus de réalisme et surtout moins d'ennuie, nous utiliserons la fonction random() qui signifie aléatoire en Anglais. Cette fonction retourne une valeur entre les valeurs minimale et maximale passées en paramètre.

Par exemple pour rendre notre radar précédent moins ennuyeux, on peut utiliser le code suivant qui ne flashe non plus toutes les 10s mais dans un temps compris entre 20s et 1minutes. C'est-à-dire entre 20*8=160 et 60*8=480 fois 125ms. La valeur dépasse 255, donc nous utiliserons des variables int et non byte.

```
byte radar_on = 1;
int cpt_radar = 10;

void user_125ms(void)
{
    //radar
    cpt_radar--;
    if(cpt_radar==2 && radar_on) { user_led(LED_RADAR,1); }
    if(cpt_radar==0) { cpt_radar = random(20*8,60*8) ; user_led(LED_RADAR,0); }
}
```

** Feu de cheminé avec un Néopixel

```
void fx_neo0_cheminee(void)
{
    byte r = random(50, 80);
    byte v = random(0, 30);

    user_neo(0,r,v,0);
}

void user_125ms(void)
{
   fx_neo0_cheminee();
}
```

Pour faire propre on crée une fonction que l'on appelle à partir de user_125ms().
La flamme est simulée par une variation de rouge entre les niveaux 50/255 et 80/255.
Une plus faible variation de vert entre 0/255 et 30/255 ajoute une composante jaune variable par mélange avec le rouge. (rouge + vert = jaune). Bien entendu, vous pouvez expérimenter d'autres algorithmes.
Vous pouvez bien entendu simuler plusieurs cheminées.

** TV avec un Néopixel

```
void fx_neo1_tv(void)
{
    user_neo(1, 0, random(10, 80), random(10, 80));
}
```

Dans cet exemple nous faisons varier la composante bleu et la composante verte pour simuler un poste de télévision

**** Scintillement permanent d'un néon avec un Néopixel**

```
void fx_neo2_scintillement(void)
{
    byte w = random(21, 29);

    user_neo(2, w, w, w);
}
```

Le néon est blanc, mais vous pouvez changer sa couleur en ajoutant un offset aux différentes couleurs. Vous pouvez aussi réduire le scintillement en rapprochant les valeurs du random ou l'augmenter en les écartant. Bien entendu la luminosité du néon est aussi réglable. Le néon suivant éclairera plus fort, scintillera moins et tirera un peu plus vers le jaune.

```
void fx_neo2_scintillement(void)
{
    byte w = random(40, 42);

    user_neo(2, w+12, w+10, w);
}
```

**** Néon cassé avec un Néopixel**

Nous rajoutons ici un effet cassé au néon scintillant en éteignant le Néopixel de temps en temps.

```
int cpt_neon = 10 ;

void fx_neo2_scintillement_et_cassé(void)
{
    byte w = random(40, 42);

    user_neo(2, w+12, w+10, w);

    // extinction du neon pour les valeurs du decompteur 6,4,3 et 0
    // pour genere une coupure de 375ms suivit d'une autre de 125ms
    // Une extinction se produit aleatoirement toutes les 5 a 10 secondes
    cpt_neon--;
    if(cpt_neon<=6 && cpt_neon>=4 || cpt_neon==0) { user_neo(2, 0, 0, 0); }
    if(cpt_neon==0) { cpt_neon = random(5*8,10*8) ; }
}
```

**** Tous les effets précédents sur un Néopixel commandés par DCC**

Nous allons commander tous les effets précédents par le DCC sur le néopixel 11. Si vous mettez ce néopixel dans une maison, vous pourrez alors choisir tous ces effets spéciaux. Nous stockerons le numéro de la sortie à activer pour le décodeur d'adresse 4 dans la variable fx_neo11. Cela nous fait 8 effets possibles. Nous n'en utiliserons que 6 avec l'état éteint. Cette variable sera ensuite testée toutes les 125ms afin d'appeler la mise à jour du bon effet spécial.

```
byte fx_neo_11 = 0;

void user_notify_bas_acc_dec(int adr_1_510, byte out_0_7, byte val_0_1)
{
    if(adr_1_510==4 && val_0_1) { fx_neo11 = out_0_7; }
}

int cpt_neon = 10 ;

void fx_neo11_scintillement_et_cassé(void)
{
    byte w = random(40, 42);

    user_neo(11, w+3, w+2, w);

    // extinction du neon pour les valeurs du decompteur 6,4,3 et 0
    // pour genere une coupure de 375ms suivit d'une autre de 125ms
    // Une extinction se produit aleatoirement toutes les 5 a 10 secondes
    cpt_neon--;
```

```
    if(cpt_neon<=6 && cpt_neon>=4 || cpt_neon==0) { user_neo(11, 0, 0, 0); }
    if(cpt_neon==0) { cpt_neon = random(5*8,10*8) ; }
}

void fx_neo11_scintillement(void)
{
    byte w = random(38, 42);

    user_neo(11, w+2, w+2, w);
}

void fx_neo11_cheminee(void)
{
    byte r = random(50, 80);
    byte v = random(0, 30);

    user_neo(11,r,v,0);
}

void fx_neo11_tv(void)
{
    user_neo(11, 0, random(10, 80), random(10, 80));
}

void user_125ms(void)
{
    if(fx_neo11==0) user_neo(11, 0, 0, 0); //eteient
    if(fx_neo11==1) user_neo(11,80,80,80); //allumé
    if(fx_neo11==2) fx_neo11_scintillement();
    if(fx_neo11==3) fx_neo11_scintillement_et_casse();
    if(fx_neo11==4) fx_neo11_tv();
    if(fx_neo11==5) fx_neo11_cheminee();
}
```

**** Scintillement permanent d'un néon avec la pwm**

```
void fx_pwm48_scintillement(void)
{
    user_pwm(48, random(21, 29));
}
```

En s'aidant de l'exemple du néopixel, ou pourra aussi simuler le cas du néon cassé. On utilisera une LED blanche.

**** Soudure à l'arc avec la pwm**

```
int cpt_arc = 10;

void fx_pwm49_arc(void)
{
    byte flash;

    cpt_arc--;

    flash = 0;
    if(cpt_arc == 0) { flash=1; cpt_arc = random(15*8,30*8); }
    if(cpt_arc>=3 && cpt_arc<=6) flash=1;
    if(cpt_arc>=9 && cpt_arc<=14) flash=1;
    if(cpt_arc == 17) flash=1;
    if(cpt_arc == 19) flash=1;

    if(flash == 0) { user_pwm(49, 0); }
    if(flash) { user_pwm(49, random(40,100); }
}
```

Cette fonction génère toutes les 15 à 30 une séquence de 5 flashs pour simuler la soudure à l'arc.
Lorsque le décompteur cpt_arc atteint 0, il est mis à une valeur comprise entre 15*8 et 30*8 secondes.
Comme cette fonction est appelée toute les 125ms et qu'elle décrémente le compteur, il fait des séquences comprises entre 15 et 30 secondes. Sa valeur est testée pour savoir s'il faut flasher ou pas. Ainsi la valeur 19 déclenche un flash de 250ms, la 17 un autre de 250ms, entre 14 et 9 un flash de 1.5s, entre 6 et 3 un flash de 1sec et un dernier flash à la valeur 0. Si le flash est autorisé alors sa valeur scintille entre 40% et 100% de la pwm.
Vous pouvez utiliser une LED bleue.

4. Conclusion

Vous savez maintenant tout sur le décodeur D18. J'espère que ce document aura été suffisant claire pour vous avoir envie de vous lancer dans cette réalisation. N'hésitez pas à me faire part de vos remarques. Partagez vos réalisations et astuces, ainsi que les modifications électroniques et logicielles afin de contribuer à l'évolution du système.

Bonne réalisation et bon jeu !

Ulysse.

Contacts Mail : ulysse.delmas@laposte.net

Site Web : http://udelmas.e-monsite.com/

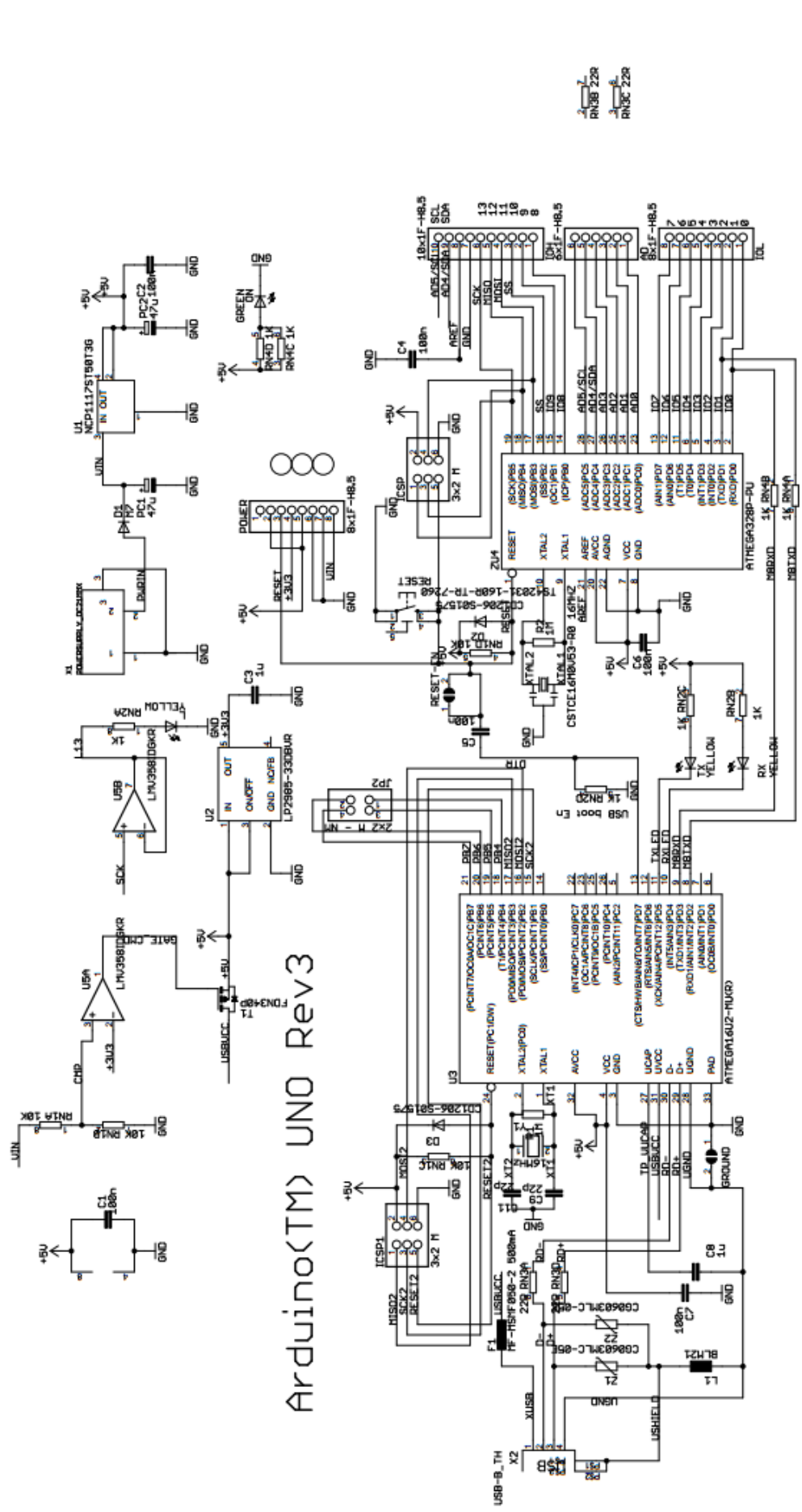
Programme (source) : Disponibles sur le site web (et bientôt dans git hub)

Quelques Liens :

- DCC NMRA: http://www.nmra.org/dcc-working-group
- Arduino: https://www.arduino.cc/
- Un bon article sur les trames DCC: http://trainminiature.discutforum.com/t12784-dcc-comment-ca-marche-place-a-la-technique
- Un bon article sur la réalisation d'un décodeur DCC sans Arduino : http://www.bahn-in-haan.de/mobatron/documents/wdecn_tn_en.pdf

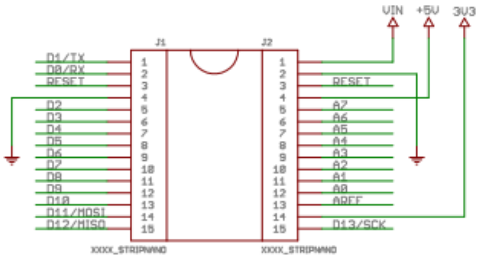
La petite histoire :

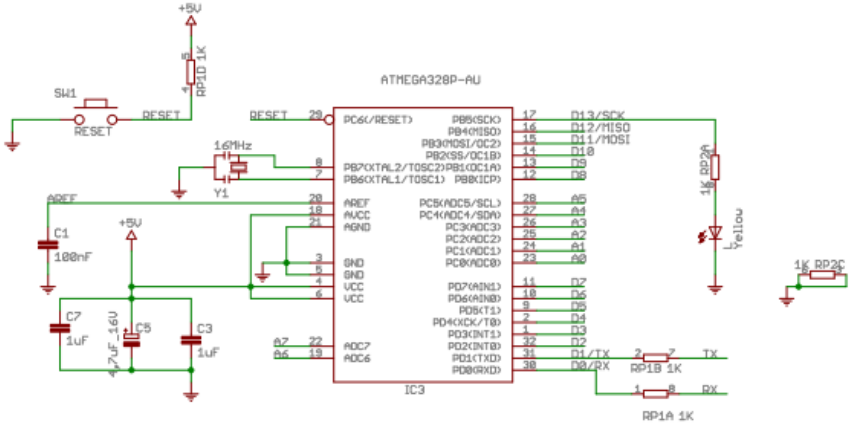
Vous pouvez vous demander pourquoi ai-je réalisé ce décodeur. En fait je voulais que ma centrale D17 supporte les décodeurs d'accessoires. Comme je n'en n'avais pas j'ai décidé d'en réaliser un pour la tester. A ma grande surprise, je n'en ai pas trouvé de simples sur Internet. J'ai alors utilisé la librairie DccNmra pour en coder un. Malheureusement après 2h de dev et debug, ça ne marchait toujours pas (surement de ma faute). J'ai donc décidé de faire un code tout seul. Mais le debug dans le code de DccNmra m'a néanmoins beaucoup aidé. Après 2h de codage et 2h de debug mon décodeur de 16 sorties fonctionnait. Ensuite, j'ai décidé de partager ce travail. Afin d'être plus complet, j'ai ajouté les MAX, PCA et Néopixels. J'espère que ce décodeur vous apportera satisfaction si vous décidez de le réaliser. Si vous souhaitez me remercier, envoyez-moi quelques photos de vos réalisations 😉

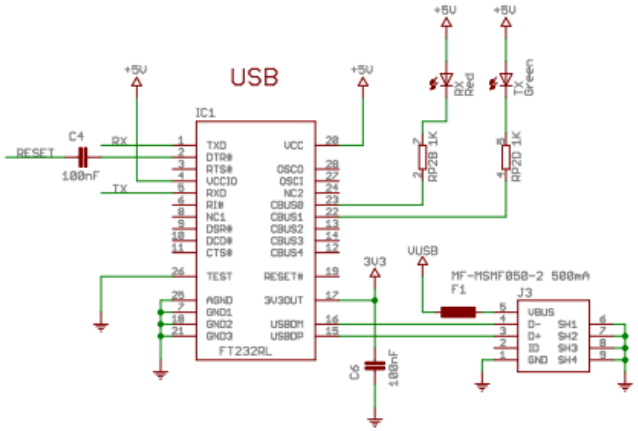


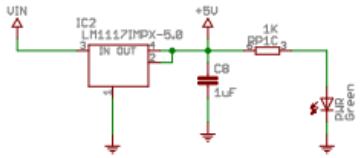
Based on design by Gravitech (gravitec.us)

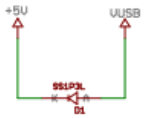
Arduino Nano

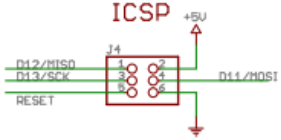


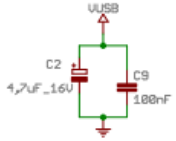






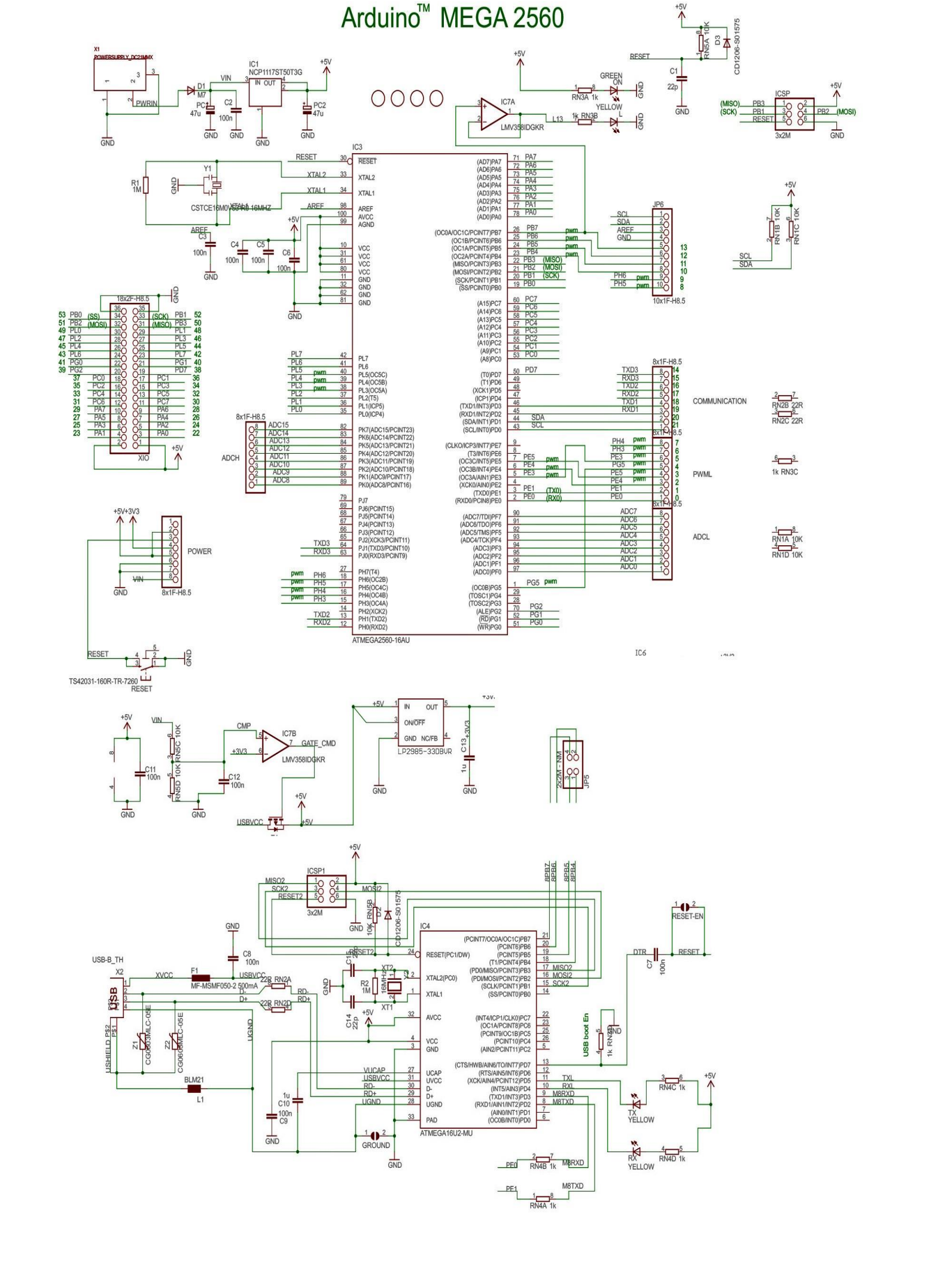






Author: E.Vita	
TITLE: Arduino Nano-Rev3.2	
Document Number:	REV: 3.2
Date: 30.12.2014 15:30:26	Sheet: 1/1

Arduino™ MEGA 2560



Carte PCA 9685