Free-DCC 2017 (FDCC2017)

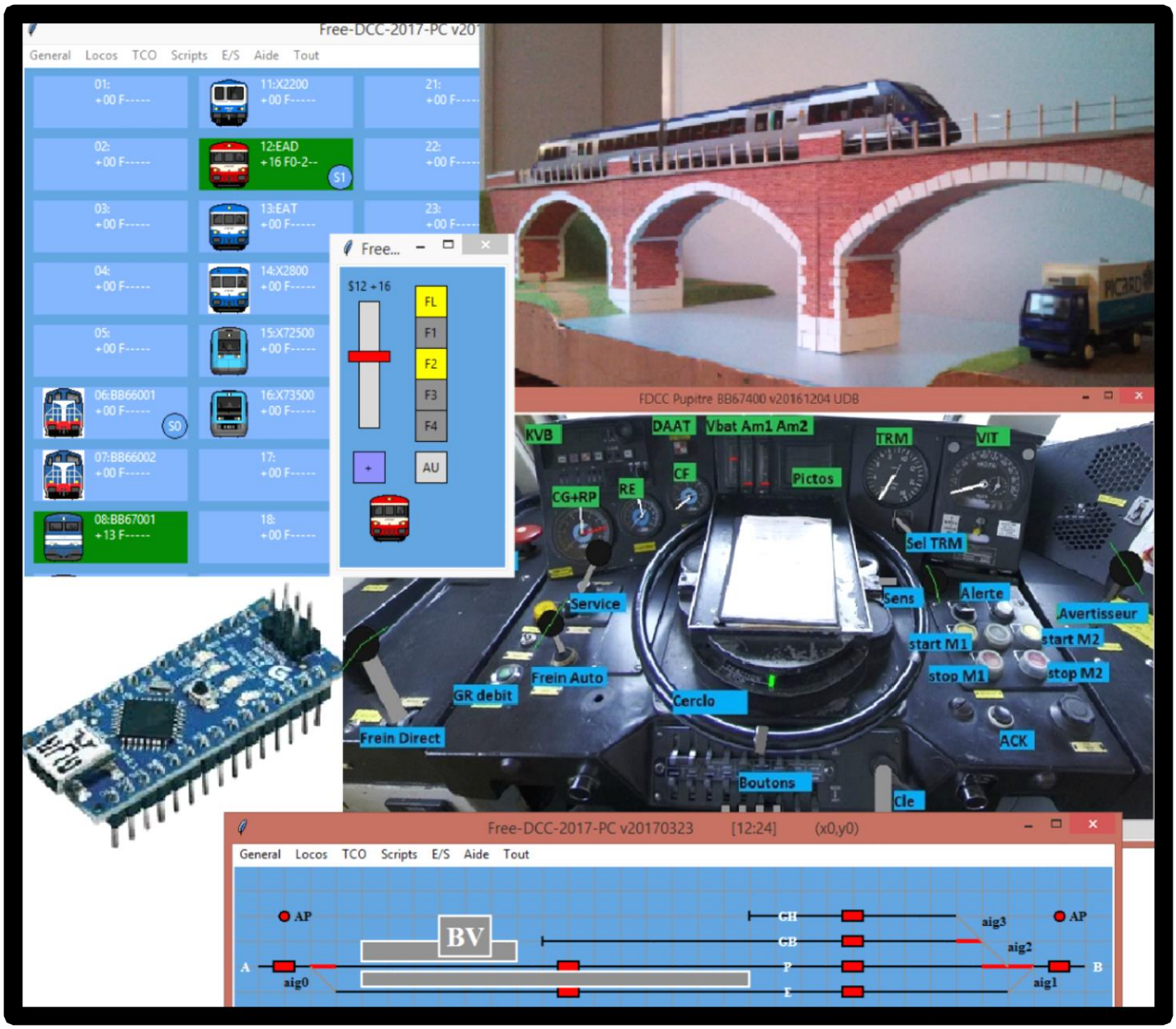


Table des matières :

1 - Introduction
2 - Architecture
3 - Centrales
4 - Electronique
5 - Utilisation en autonome
6 - Les Logiciels
7 - Conclusion

Annexe A - Le DCC
Annexe B - Electronique
Annexe C - Cas pratique

(Ulysse Delmas-Begue documentation du 26 juillet 2017)

1. Introduction

Free-DCC 2017 (FDCC 2017) est un système de commande de réseau ferroviaire miniature numérique basé sur la norme digitale DCC et/ou analogique. Le système se compose d'une centrale pour contrôler le réseau. La centrale peut être autonome pour commander les locomotives et gérer la signalisation, mais il est plus intéressant de la connecter à un ordinateur pour profiter de plus de fonctionnalités : TCO, itinéraires, cabines de conduite, automatisation … Ce système est « open hardware » et « open source », c'est-à-dire que les schémas électriques et le code des logiciels sont fournis. Cela vous permet de comprendre le fonctionnement, du système, le réparer, l'adapter à vos besoins, le faire évoluer …

La version 2017 profite de l'expérience acquise avec les précédents systèmes fdcc, fdcc2008 et fdcc2010. fdcc et fdcc 2008 ne sont plus utilisés. Pour les personnes ayant fdcc2010, il est possible de faire évoluer le système en remplaçant seulement la centrale (5€). Les améliorations sont simplification et modularité !

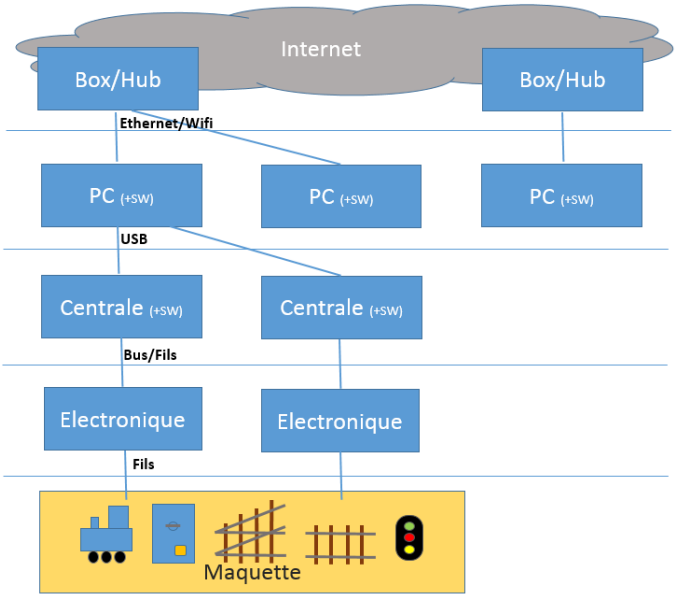
- Niveau simplification, la centrale repose maintenant sur la populaire carte Arduino NANO au lieu d'une solution personnelle à base de microcontrôleur Microchips. Une carte à microcontrôleur toute-faite, effraie moins les néophytes et l'environnement de développement permet de transférer facilement le logiciel dans la carte. L'Arduino se programment en C qui est plus facile à comprendre, à maintenir et à modifier que l'assembleur. Niveau ordinateur, le langage Python remplace le C afin d'être plus facilement modifiable (pas besoin de compiler) et pour tourner sur différents OS comme Windows et Linux. Avec Linux, l'ordinateur peut même être remplacé par une Raspberry PI à 20 euros.

- Niveau modularité, plusieurs centrales peuvent être utilisées ce qui permet de diminuer le câblage et de mieux s'adapter aux gros réseaux modulaires. Les centrales sont reliées par USB à un ordinateur sur lequel tourne le programme « fdcc_pc » qui permet d'une part de communiquer avec les cartes et d'autre part de communiquer avec les applications (TCO, cabines …) par TCP/IP. Les applications peuvent donc être non seulement sur l'ordinateur local, mais encore sur n'importe quel autre ordinateur relié en réseau avec cet ordinateur. Des téléphones ou tablettes Android reliées par Wifi peuvent être utilisés comme souris. Il est possible de prévoir des extensions futures comme des cabines ou TCO sur tablettes …

Ce système a été conçu pour être libre, le moins cher possible, facile à modifier, modulaire, simple à utiliser et performant.

2. Architecture

Architecture globale :

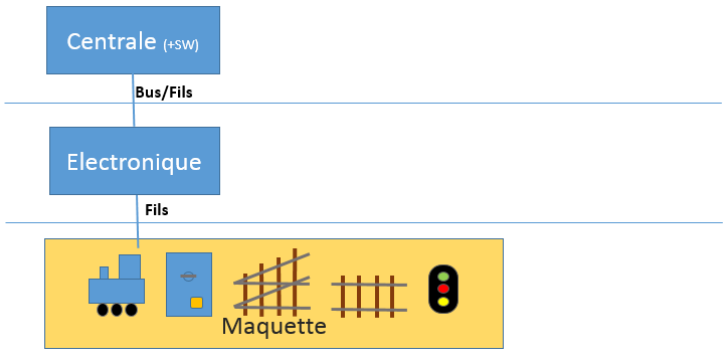


Le schéma précédent montre l'architecture globale du système. Le système étant fortement modulaire, vous pouvez utiliser uniquement les éléments dont vous avez besoin et en rajouter d'autres par la suite. Afin d'expliquer les différentes parties, nous partirons d'un système minimal pour aller vers un système utilisant tous les éléments.

La centrale repose sur une carte Arduino (Nano ou UNO). Ces cartes électroniques seront présentées plus tard. Ce choix a été fait pour la simplicité d'utilisation et de programmation. En effet la communauté Arduino propose gratuitement tout un environnement de développement qui vous permettra de programmer facilement la carte avec le dernier programme « fdcc_centrale.ino ». De plus la carte fonctionnera du premier coup et son prix est vraiment modique (souvent 5€).

La centrale génère les différents signaux pour piloter la maquette. Il faut rajouter un peu d'électronique entre la centrale est la maquette comme un booster pour amplifier le signal DCC …

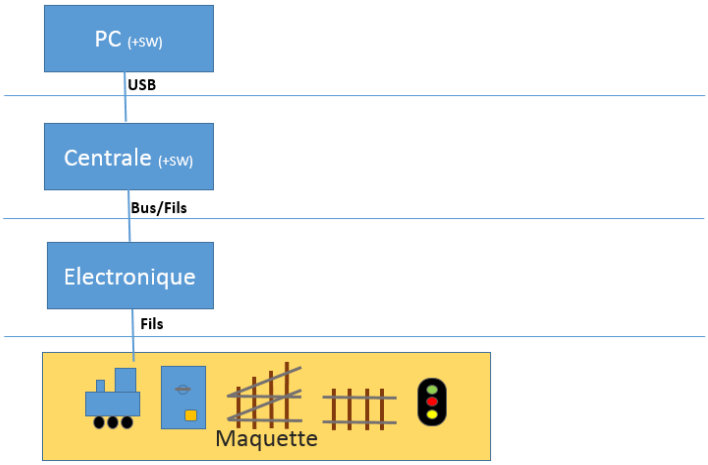
Architecture1 : Centrale autonome



Commençons par la configuration minimale avec la centrale en mode autonome, c'est-à-dire sans PC. Les souris (manettes) permettent de contrôler la marche des locomotives. Les aiguillages sont manuels ou contrôlés électriquement avec des boutons poussoirs sans passer par le système. Il n'y a pas de signalisation ni de TCO géré par le système.

*Pour les utilisateurs avancés, il est possible d'aller beaucoup plus loin en mode autonome. Le programme de la centrale appelle régulièrement une fonction « user_loop() » dans laquelle vous pouvez faire ce que vous voulez. Par exemple, gérer la signalisation, des itinéraires ou automatiser ce que vous voulez. Bien entendu, il faut connaitre le langage C et le fonctionnement du programme de la centrale. Je ne décris pas cette partie car les débutants pourront réaliser ces fonctions bien plus facilement sur le PC.

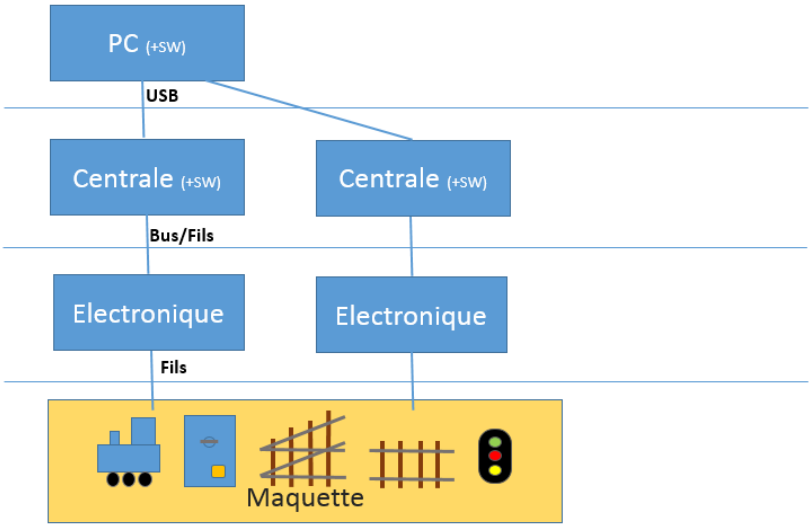
Architecture2 : Une centrale et 1 PC



En connectant un PC à la centrale, vous disposez de nouvelles fonctionnalités comme :

- Gestion et conduite des locomotives à partir du PC avec des souris virtuelles
- Conduite des locomotives à partir de reproductions de cabines réelles !
- Programmation des décodeurs
- TCO
- Itinéraires, Signalisation
- Suivi des trains sur les différents cantons, Conduite automatique en mode cantonnement
- Equations et Scripts pour automatiser le réseau (par exemple pour déplacer automatiquement les trains)
- …

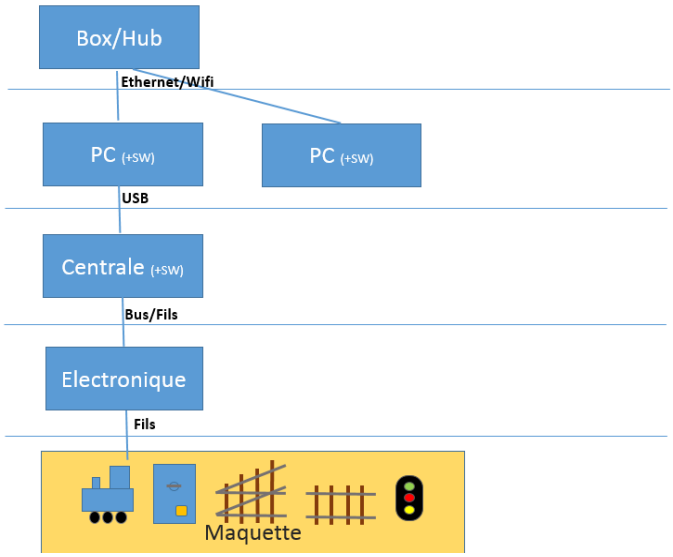
Architecture3 : Un PC et plusieurs centrales



Free-DCC offre la possibilité de connecter plusieurs centrales. Cela peut être intéressant pour réduire le câblage ou pour s'adapter aux réseaux modulaires. Comme vous le verrez par la suite free-dcc propose plusieurs centrales plus ou moins « grosses ». Avec une approche modulaire, il est par exemple possible d'utiliser plusieurs petites centrales à la place d'une grosse.

Le signal DCC étant commun à tout le réseau, il est de ce fait généré que par une seule centrale. Par contre chaque centrale gère ses aiguillages, détections, leds …
Le programme « fdcc_pc » tournant sur le PC est écrit en Python ce qui permet de l'exécuter sur n'importe quel OS. Par exemple Windows ou Linux. Dans le cas de Linux, il est même possible de remplacer le PC par une carte Raspberry PI à 20 euros !

Architecture 4 : Plusieurs PC et une centrale

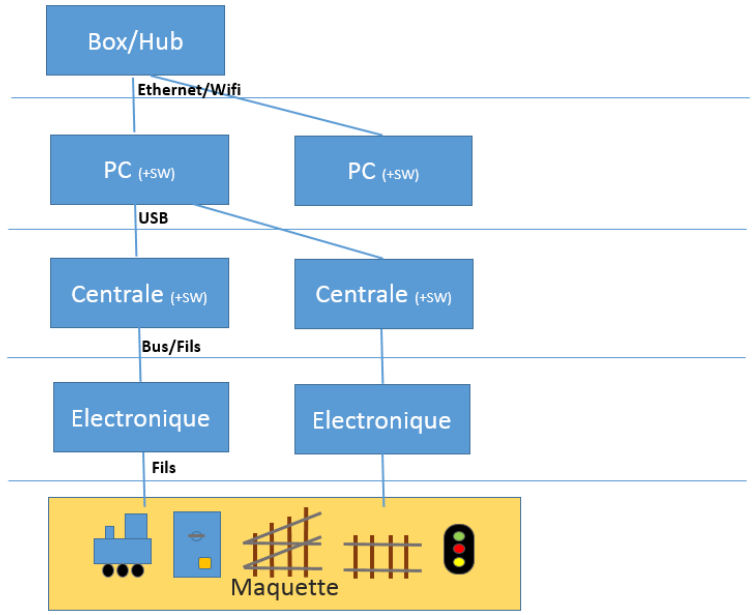


Il est également possible d'utiliser plusieurs ordinateurs connectés en réseau via des câbles Ethernet ou sans fils via wifi. Ces connections peuvent se faire via un Hub, un switch, un routeur wifi ou votre Box Internet.
Le programme « fdcc_pc » tourne impérativement sur le PC connecté à la centrale. Il accepte que d'autres programmes s'y connectent via TCP/IP. Ces programmes peuvent être sur le même PC ou sur un PC distant. Il devient alors possible par exemple d'avoir des TCO, souris virtuelles et cabines de conduites sur autant de PC que nécessaire.

Il n'y a pas que des PC qui peuvent se connecter à « fdcc_pc ». En effet une tablette Windows est aussi un PC ;-)
Les tablettes et téléphones Android peuvent aussi s'y connecter via wifi et servir de souris. Je développerai certainement dans le futur une souris wifi basée sur le circuit esp8266 pour ceux qui n'aiment pas le tactile. Il est aussi envisageable de proposer des TCOs sur les tablettes Android. Je ne parle pas d'iPhone/iPad car je n'en ai pas.

Bien entendu, il est nécessaire d'avoir quelques notions de réseau pour connecter plusieurs ordinateurs entre eux.
Il est possible de connecter le système à Internet via la box wifi pour par exemple piloter le réseau à distance, mais cela ne semble pas très utile.

Architecture5 : Plusieurs PC et plusieurs centrales



Il est possible de combiner les 2 configurations précédentes afin d'avoir plusieurs centrales et plusieurs PC.
Notez toutefois que toutes les centrales doivent être connectées au même PC.

---

3. Les centrales

La centrale est le cœur du système fdcc. Elle repose sur une carte électronique Ardiuno Nano.

Le microcontrôleur de cette carte avec son programme permet de :

- Générer le signal DCC pour contrôler les locomotives,
- Communiquer avec le PC via USB
- Prendre en compte les commandes des souris
- Lire les entrées (détection des locomotives, boutons poussoirs ILS …)
- Mettre à jour les sorties, pwm, aiguillages, leds et néopixel.

Free-DCC propose actuellement 2 centrales.

- La A qui est une mise à jour de l'ancienne centrale fdcc-2010. C'est une « grosse » centrale qui avec ses nombreux bus permettent de gérer de gros réseaux. Elle représente l'approche traditionnelle de ces dernières années.

- La B est une « petite » centrale qui permet de gérer un petit réseau ou un gros réseau (y compris modulaire) si on en utilise plusieurs. Cette nouvelle approche permet de simplifier le câblage et s'adapte bien aux réseaux modulaires dont la configuration change souvent.

Il est bien entendu possible de mixer les centrales. Leur nombre n'est pas limité. Il est envisageable de développer d'autres centrales si le besoin s'en fait sentir.

3.a. 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 ne permet pas de générer le signal DCC à la micro seconde près et le montage ne fonctionnera pas. AT32U4 et FT232 sont supportés de base par l'environnement Arduino. 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

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 fonctionne bien avec fdcc.

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 fonctionne bien avec fdcc.

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

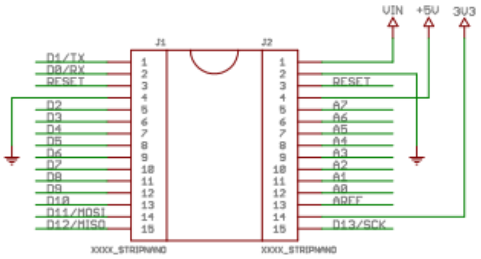
Pour information, il existe d'autres cartes Arduino, comme la « grosse » Arduino Mega qui propose bien plus d'entrées/sorties. Si certains veulent faire des centrales plus grosses, vous pouvez les utiliser. Il faudra et modifier un peu le programme afin de gérer les nouvelles entrées/sorties. Pas de panique il s'agit du même microcontrôleur mais avec plus de ports, timers, liens séries, mémoire. (Je n'ai pas encore testé cette carte avec fdcc)

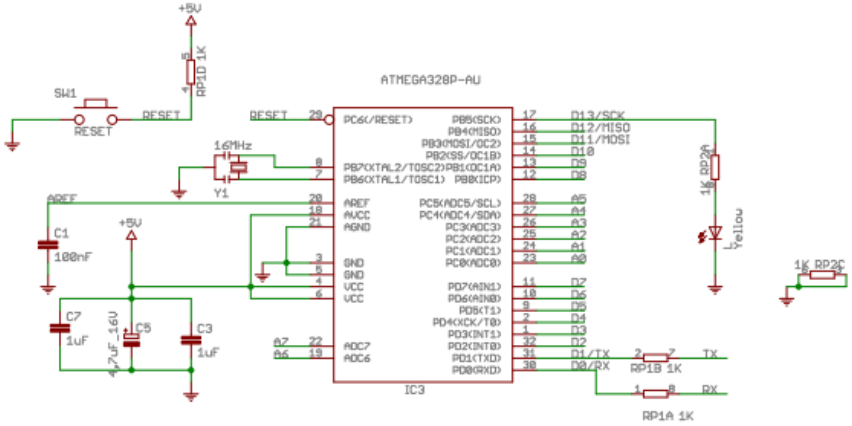

```
Flash : 256KB
RAM   : 8KB
Quartz: 16MHz
E/S   : 54 digitales + 16analogiques
UART  : 1 connectée à l'USB + 3
PWM   : 12 (5 timers)
```

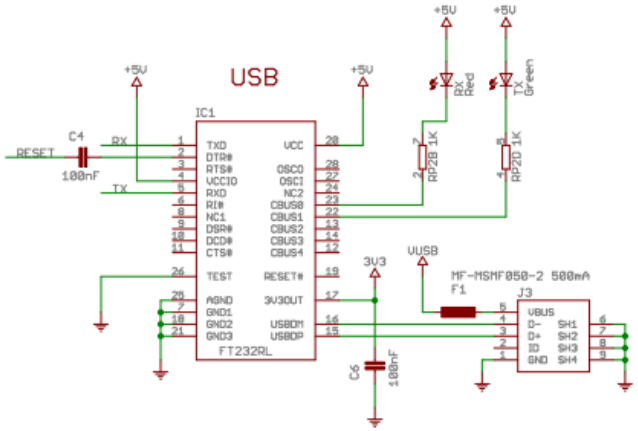
La figure suivante présente le schéma électronique de l'Arduino Nano

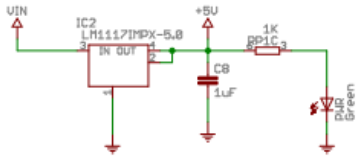
Based on design by Gravitech (gravitec.us)

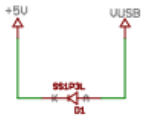
Arduino Nano

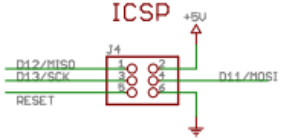


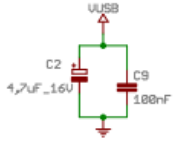






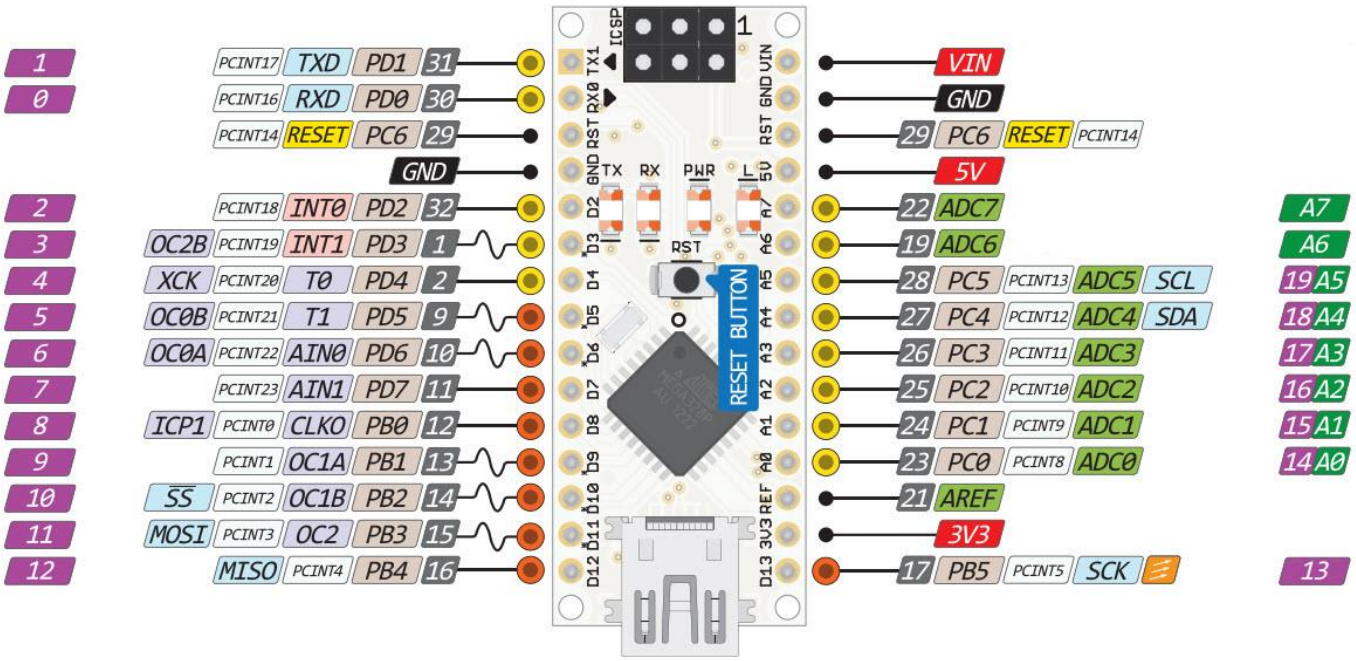






Author: E.Vita	
TITLE: Arduino Nano-Rev3.2	
Document Number:	REV: 3.2
Date: 30.12.2014 15:30:26	Sheet: 1/1

Le schéma suivant identifie les pattes :



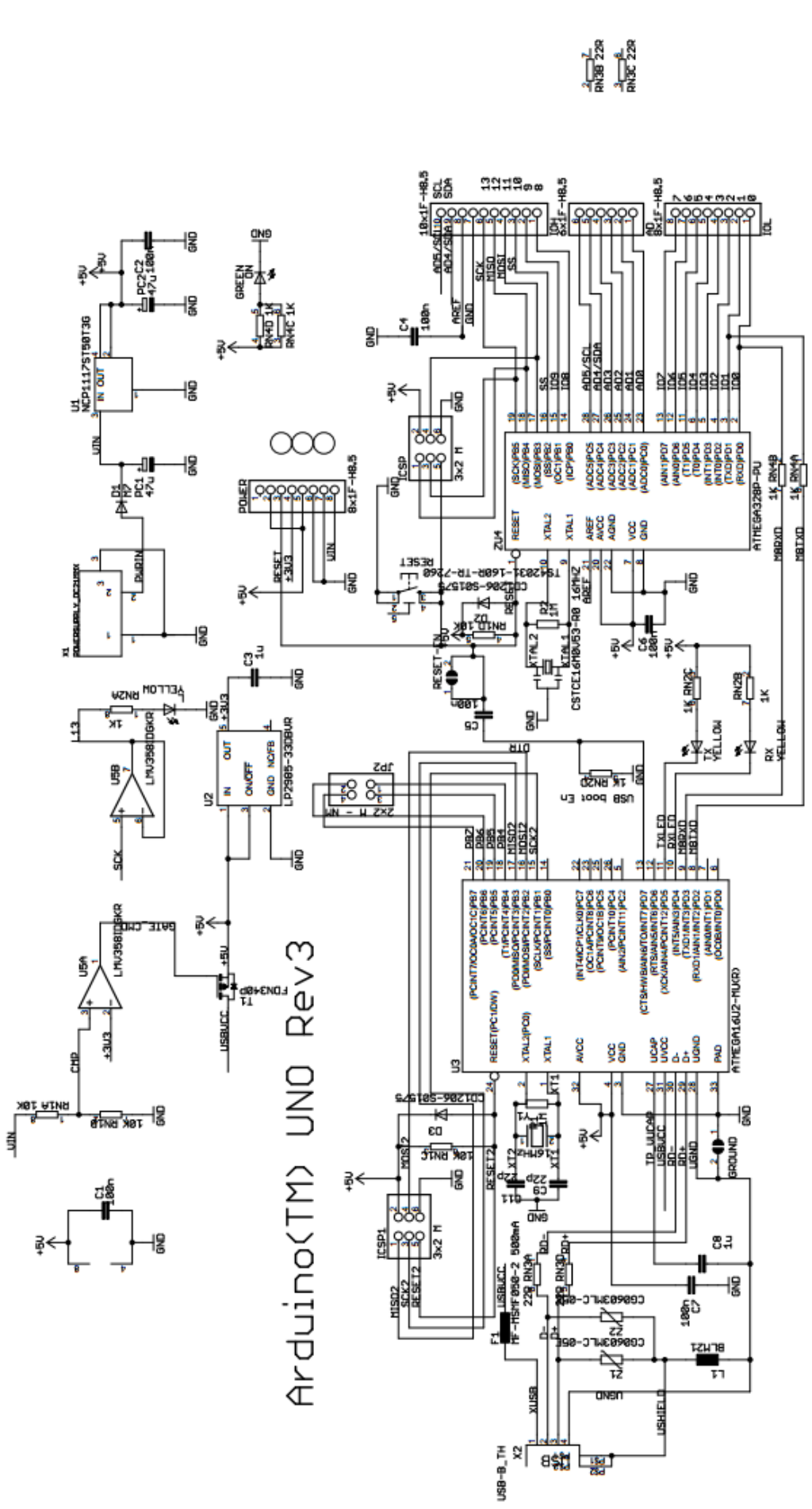
Pour repérer les pattes, nous utiliserons les noms sérigraphiés sur la carte. Nous utiliserons les pattes suivantes :

- 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 et A0 à A7 : les entrées et sorties pour nos signaux

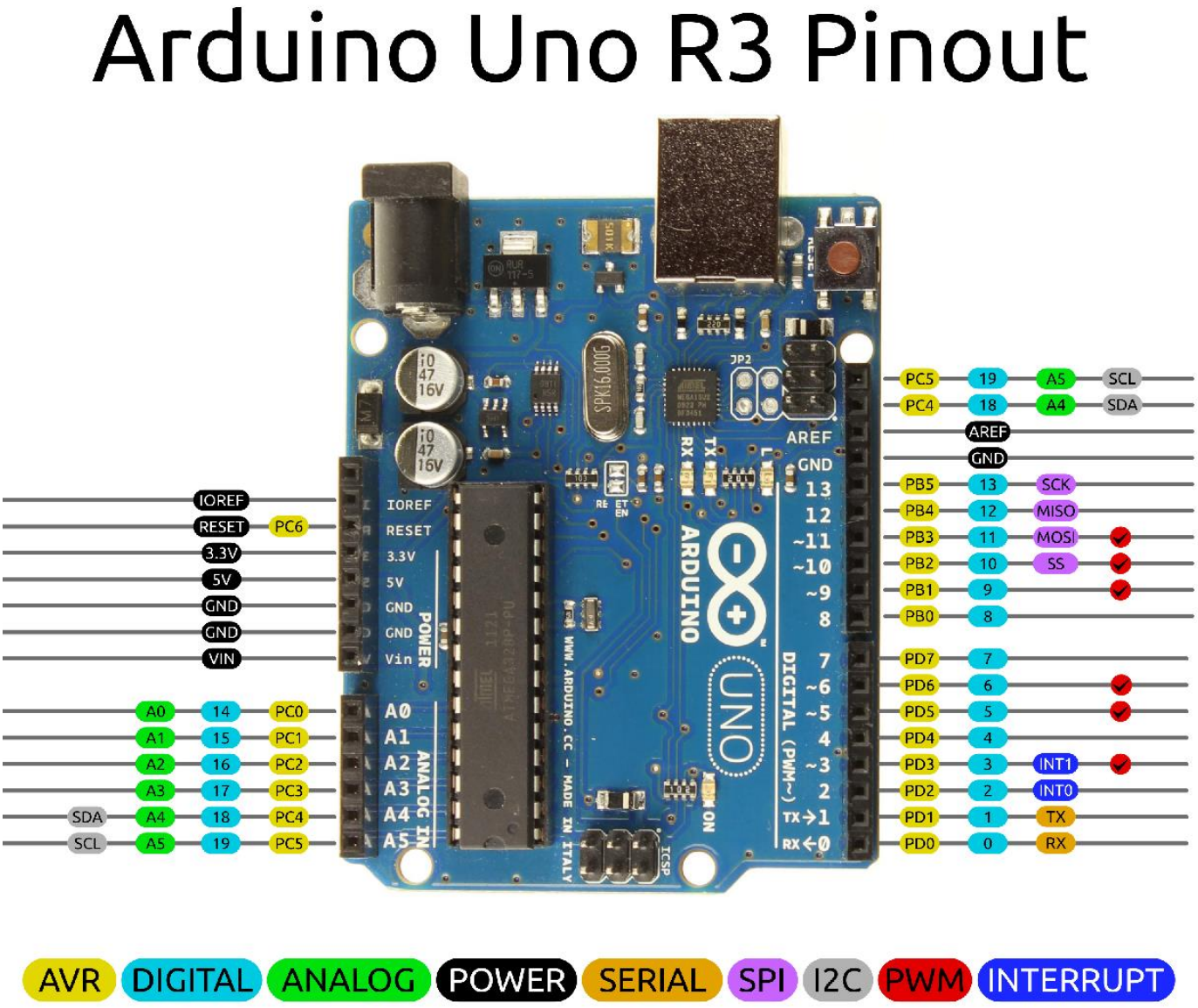
Les autres pattes ne seront pas utilisées et seront laissées en l'air.
Le D devant les entrées/sorties (E/S) signifie Digital (numérique, 0/1 = 0V/5V).
Le A devant les entrées/sorties (E/S) signifie Analogique. Ces pattes sont connectées au convertisseur analogique/numérique qui permet mesurer la tension présente sur la patte (0=0V … 1023=5V). Ces pattes peuvent aussi fonctionner en numérique. A6 et A7 fonctionnent uniquement en entrées

Vous remarquerez l'étoile devant les pattes D3, D5, D6, D9, D10 et D11. Cela signifie qu'elles supportent la PWM (Pulse Width Modulation = Modulation de largeur d'impulsion) afin d'avoir des sorties variables.

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



Le schéma suivant identifie les pattes :



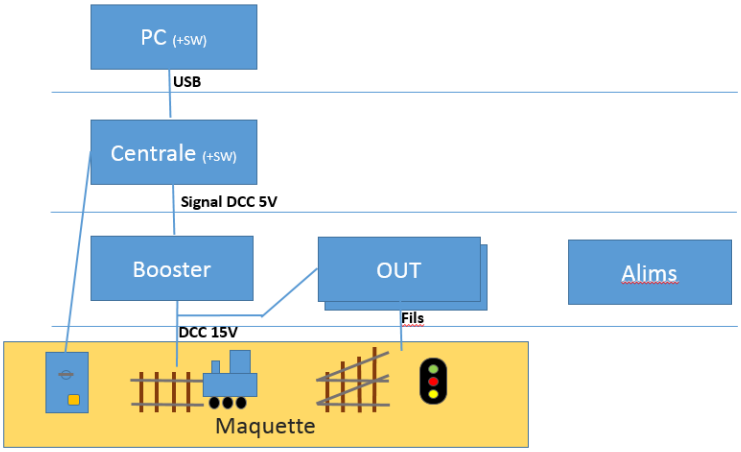
Pour en savoir plus, je vous invite à aller sur le site de la fondation Arduino https://www.arduino.cc/.
Je vous conseille de lire les excellents manuels d'utilisation qui pullulent sur Internet
(ex : https://www.tutorialspoint.com/arduino/arduino_tutorial.pdf)
Vous verrez que les Arduinos offrent d'énormes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

Installez l'environnement de développement, compilez le programme « 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 de la centrale de votre choix et commencer à utiliser fdcc. Dans le jargon Arduino, on dit que l'on « télé-verser » un « sketch »

3.b. Les différentes centrales

Cette partie présente les centrales A et B. Mais avant cela, nous allons présenter ce que dit la NMRA (Association des modélistes d'Amérique du Nord) qui a créé le standard DCC :

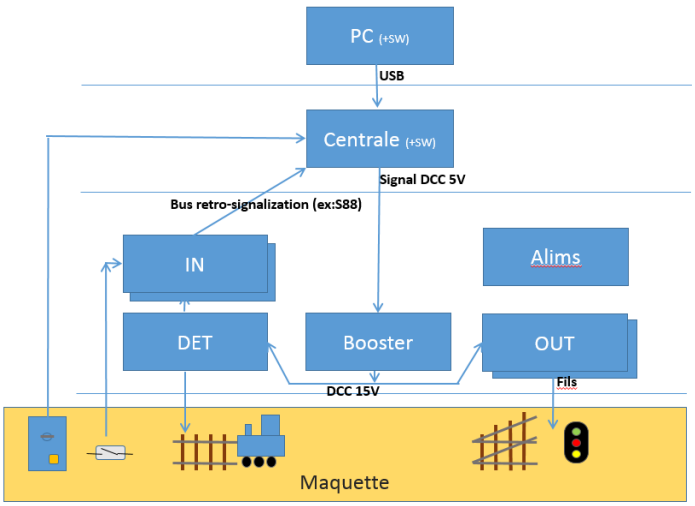
Centrale NMRA



Ce schéma montre ce qui avait été imaginé voici plus de 10 ans. Une centrale génère le signal DCC (0-5V) qui est amplifié par le booster (+15 / -15V, 3A max par exemple). Cela permet de commander les locomotives à l'aide de souris. Le signal DCC est également utilisé pour commander les « accessoires » que sont les aiguillages et feux. Hormis cela, le reste n'est pas normalisé : Souris (manettes), lien avec un ordinateur …

Ce système est uniquement dédié à la commande. Il est vite apparu qu'il était nécessaire d'ajouter des entrées au système par exemple pour détecter les trains.

Centrale NMRA + retro-signalisation



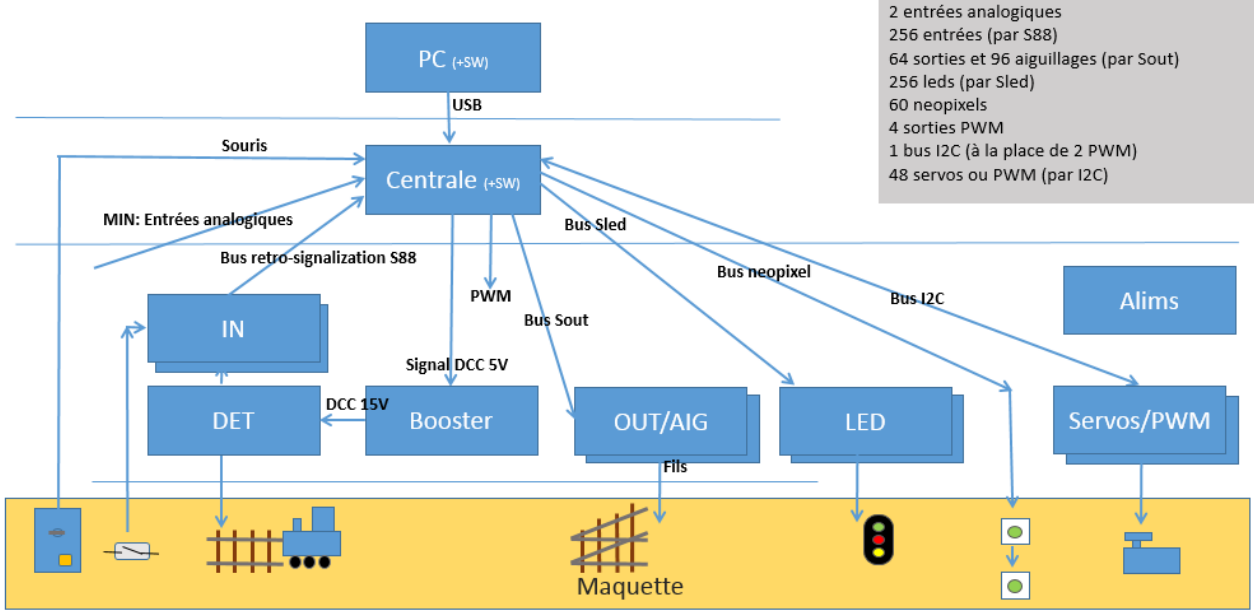
Malheureusement, rien n'a été normalisé pour les entrées et plusieurs bus de rétro-signalisation ont été utilisés par les constructeurs. Le plus courant était le bus S88 qui consiste à relier des modules de 8 à 16 entrées par un lien série synchrone.

Les modules d'entrées acceptent des interrupteurs, ILS (interrupteur à lame souples) qui sont des interrupteurs dans une ampoule de verre déclenchés par un aimant placé sous une locomotive par exemple. Souvent, il est nécessaire de détecter les trains par le courant, on intercale alors un détecteur de courant entre les voies et le module d'entrées.

On voit très bien que dans les centrales NMRA il est dommage d'utiliser le signal DCC pour les accessoires.
De nombreux fabricants proposent des centrales avec d'autres bus. LocoNet, CAN, I2C, XpressNet …

Centrale FDCC typeA (compatible FDCC2010)

Connection USB
50 locos // (28crans+5fonctions)
Programmation des décodeurs
Possibilité de fonctionner en analogique
10 souris
2 entrées analogiques
256 entrées (par S88)
64 sorties et 96 aiguillages (par Sout)
256 leds (par Sled)
60 neopixels
4 sorties PWM
1 bus I2C (à la place de 2 PWM)
48 servos ou PWM (par I2C)



La centrale free-DCC type A est typique d'une approche standard avec une « grosse » centrale pouvant gérer un grand réseau. Elle propose une multitude de bus pour y connecter une grande quantité de modules.

On retrouve la partie DCC, entrées (IN) et détecteurs (DET) de l'évolution des centrales NMRA

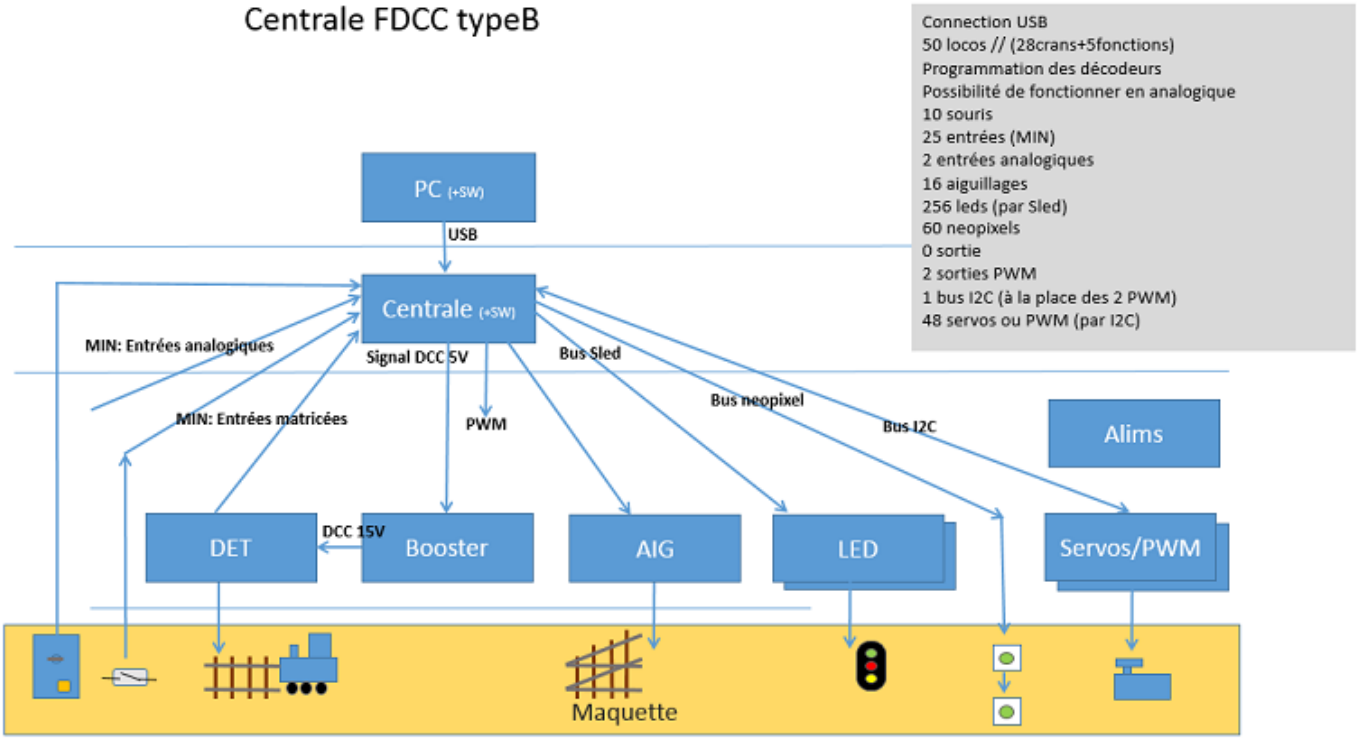
Par contre les accessoires n'utilisent pas le signal DCC mais 2 nouveaux bus : Le bus Sout (Serial Out) pour les sorites et aiguillages. Le bus Sled (Serial Led) pour les leds. Ces choix ont été faits pour être compatible avec les modules de la centrale fdcc-2010 que certaines personnes ont réalisé. Ils peuvent ainsi faire évoluer leur système sans grand frais.

J'ai rajouté la possibilité de gérer des LED néopixel. Ces leds sont 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 (16 millions 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 … Il est également possible de générer quelques effets spéciaux sur les leds normales.

La centrale propose aussi 4 sorties PWM qui peuvent être utilisées pour piloter des sorties et les faire varier. Il est aussi possible de les utiliser pour contrôler des locomotives en analogique (nous y reviendrons).

Il est possible de remplacer 2 sorties PWM par un bus I2C afin de rajouter des fonctionnalités. Il existe en effet un grand nombre de circuits I2C pour faire tout est n'importe quoi. Actuellement, la centrale supporte 3 circuits PCA8596. Chaque circuit dispose de 16 sorties pouvant être utilisées individuellement en mode PWM ou Servomoteur. Donc 48 sorties PWM/Servomoteur !

La centrale supporte 10 souris, 2 entrées analogiques et 256 entrées numériques via le bus S88.
La centrale peut être connectée à un PC en utilisant le bus USB.



La centrale free-DCC type B est une petite centrale permettant de gérer un petit réseau ou un grand réseau avec plusieurs centrales. Cette nouvelle approche est particulièrement bien adaptée aux réseaux modulaires. Dans tous les cas, une seule centrale est utilisée pour générer le signal DCC.

Par rapport à la centrale type A, on remarque que les bus S88 pour les entrées et Sout pour les sorties et aiguillages ont disparus. La place libérée permet piloter directement jusqu'à 16 aiguillages. Il n'y a plus que 2 sorties PWM. Les entrées se connectent directement sur les 25 entrées matricées.

Il est possible de remplacer les 2 sorties PWM par un bus I2C afin de rajouter des fonctionnalités.
Il existe un grand nombre de circuits I2C pour faire tout est n'importe quoi (MCP23017 pour rajouter des entrées avec mémorisation, PCF8574 pour de simples entrées/sorties, PCA8596 pour la pwm et les servos, de multiples références pour les leds, convertisseurs analogique/numérique et inversement …
De plus, il est possible de connecter plusieurs circuits du même type.
Actuellement, la centrale supporte 3 circuits PCA8596. Chaque circuit dispose de 16 sorties pouvant être utilisées individuellement en mode PWM ou Servo. Donc 48 sorties PWM/Servo !

4. Electronique

Cette partie décrit l'électronique de la centrale et des différents modules. La plupart des modules peuvent s'acheter par exemple sur Amazon pour quelques euros. Vous pouvez les relier par des fils ou alors réaliser une carte électronique pour être plus propre. (Je n'ai pas encore réalisé la carte électronique). Si vous n'avez pas de grandes notions en électronique, l'annexe A peut vous aider.

4.1. Les alimentations

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

- 5V continu pour l'Arduino, la plupart des circuits électronique, les leds, néopixel et certains relais
- 12V à 18V continu pour le DCC, aiguillages, certains relais et accessoires

Il faudra dimensionner la puissance des alimentations en fonction de la consommation maximale. Par exemple : 3A pour le DCC, 1A pour les aiguillages, 60mA par néopixel (max) …

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 pourrait être tenté d'alimenté la carte par son entrée Vin ou en par sa patte 5V, mais ce serait une mauvaise idée car si ces alimentations ne sont pas présente alors l'alimentation proviendrait de l'USB et déclencherai le fusible.

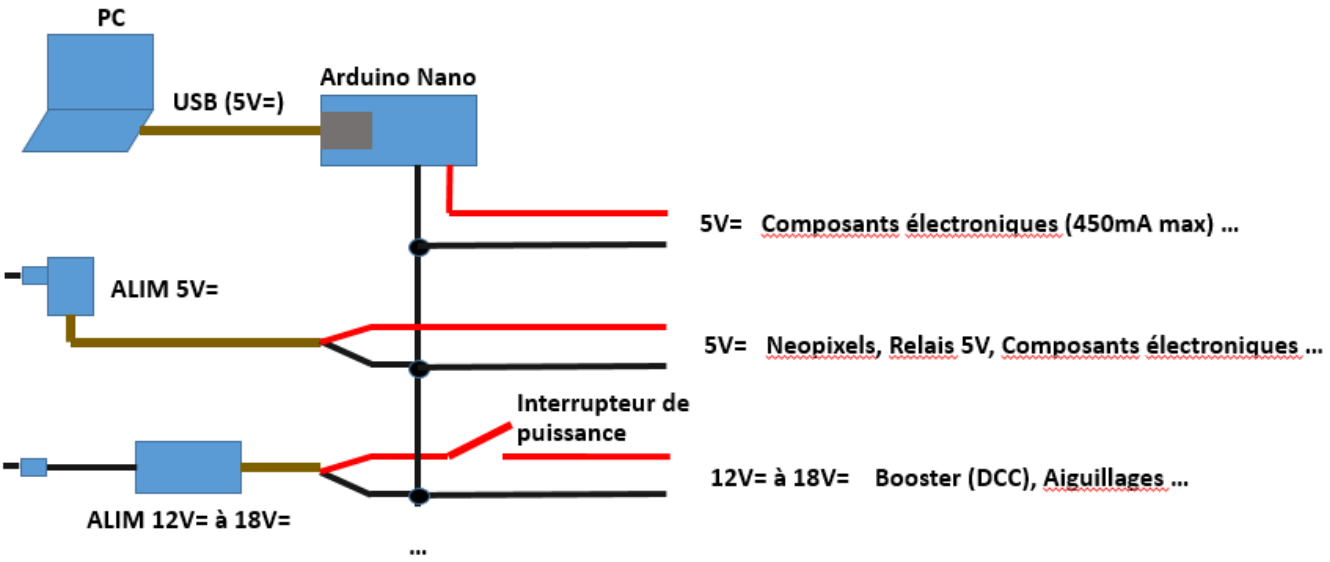
Exemple de quelques consommations :

- Train à vitesse max : 500mA (mais cela varie énormément en fonction du moteur, de l'éclairage …)
- 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 et de ne le fermer qu'une fois le système prêt. En effet, en l'absence de signal DCC à l'entrée du booster, les locomotives pourraient se croire en analogique et rouler à fond. Les bobines des aiguillages pourraient cramer si la commande est mauvaise.
Voici une séquence d'allumage :

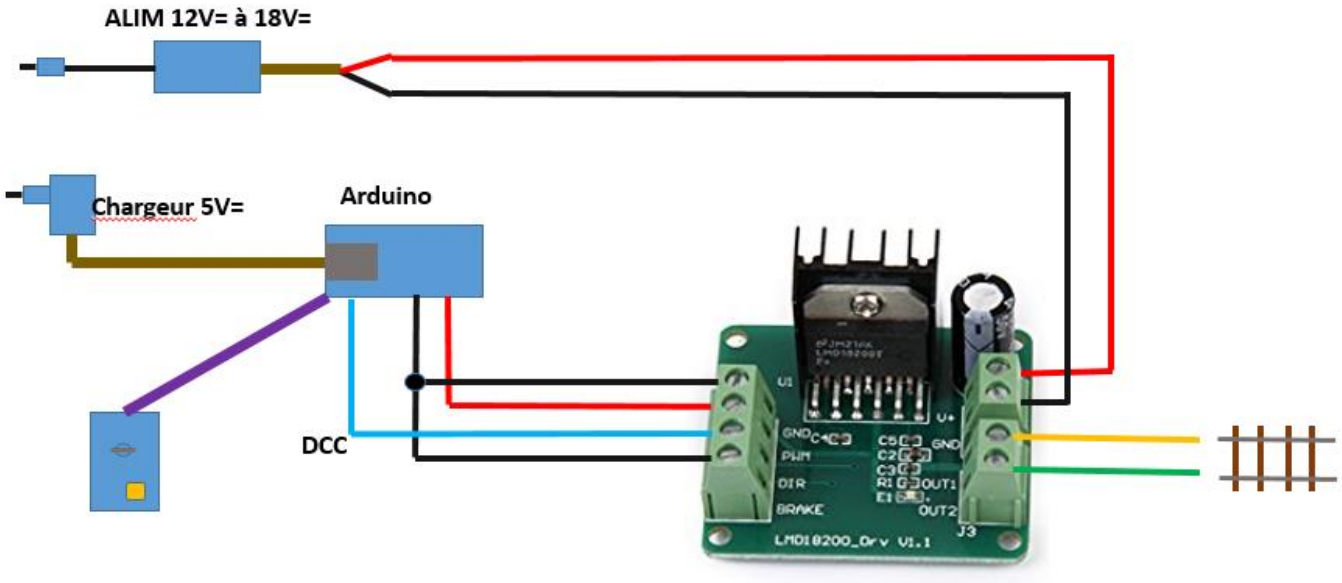
- Brancher l'alimentation 5V
- Brancher ou rebrancher l'Arduino (afin que le programme démarre et initialise correctement l'électronique)
- Brancher l'alimentation de puissance

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



4.2. Le Booster

Le booster est un pont en H qui transforme le faible signal DCC de commande 0-5V (quelque mA max) en un fort signal -15V/+15V (3A max). Il existe plusieurs composants aptes à cette mission. Le plus adapté est le LMD18200T. Je ne l'avais pas conseillé dans les précédentes réalisations du fait de son prix. Mais avec Amazon, on trouve des modules tout fait entre 5 et 10€. La photo suivante montre un tel booster connecté sur la centrale en autonome :



Il suffit de brancher l'alimentation de puissance entre V+ et GND
Connecter PWM au +5V pour activer le booster
Connecter OUT1 et OUT2 aux voies.
Connecter DIR à la sortie PWM de l'Arduino (A0) :

- Si A0=0V alors OUT1=0V et OUT2=V+ Donc OUT1-OUT2=-V+ (ex : -15V)
- Si A0=5V alors OUT1=V+ et OUT2=0V Donc OUT1-OUT2=+V+ (ex : +15V)

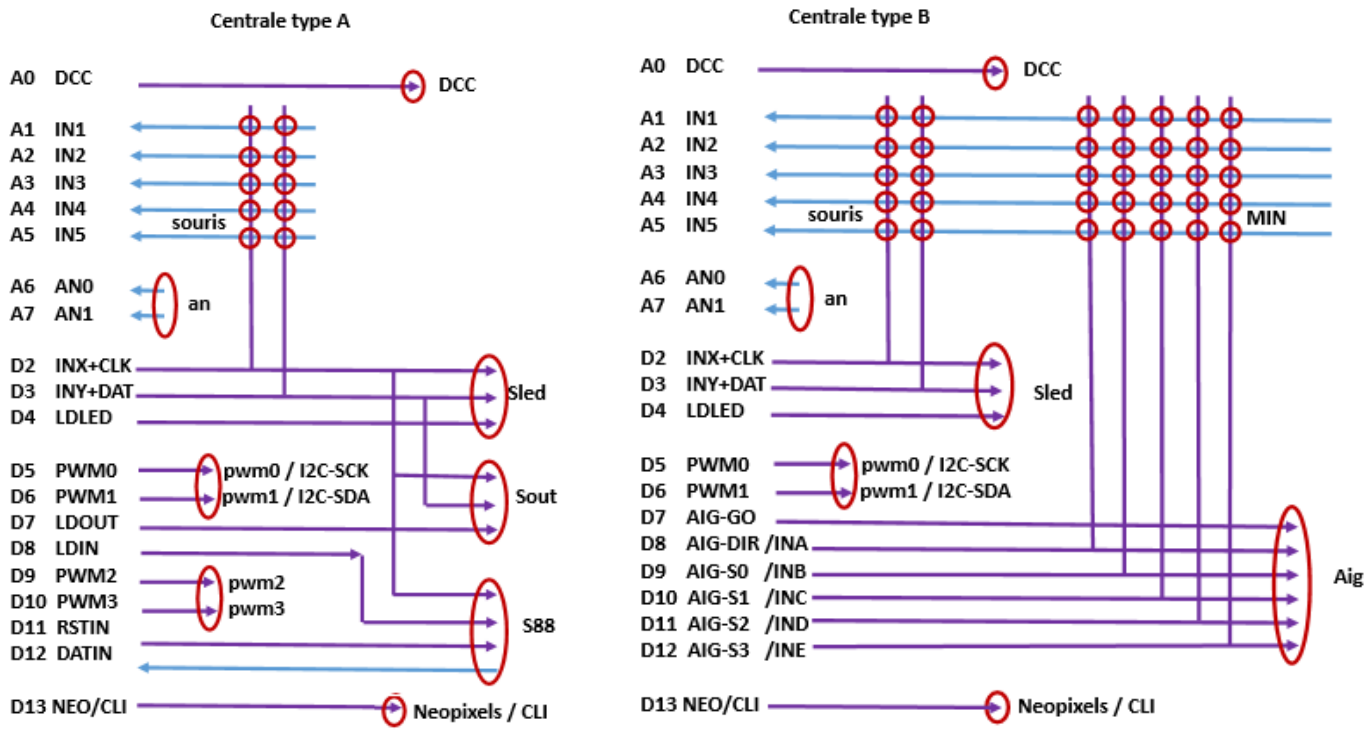
Connecter BRAKE à la masse (on n'a pas besoin de court-circuiter le pont).
Ne pas oublier de relier les masses des alimentations.
Pour info, les 2 connecteurs GND du module sont connectées.

Le LMD18200T est capable de commuter 3A en continue (5A en pic). Ce qui correspond à 6 trains roulant à vitesse max, ce qui est déjà beaucoup. Le circuit est protégé contre les court-circuits et la surchauffe. Je ne conseille pas de monter au-delà de 3A avec un booster car dans la plupart des cas les circuits de détection de courant n'y résisteraient pas. Il faut bien entendu adapter les câbles au courant traversé.

Il est à noter que les transistors du pont ne sont pas parfaits et ont une résistance typique de 0.33Ω en conduction. Donc à 3A, chaque transistor accuse une chute de tension de 0.33*3=1V et comme il y en a 2, on se retrouve avec une chute de tension globale de 2V. La sortie n'est donc pas de 15V-0V=15V mais de 14V-1V=13V. De plus cela fait chauffer le pont 2V*3A=6W d'où le radiateur. Dans ce cas la puissance fournit est de 14V*3A=42W d'où un rendement de (42-6)/42=86%, ce qui n'est pas si mal surtout si on compare à de l'analogique. Une amélioration pourrait être d'utiliser la patte « Sense » (non reportée sur les connecteurs) afin de mesurer le courant.

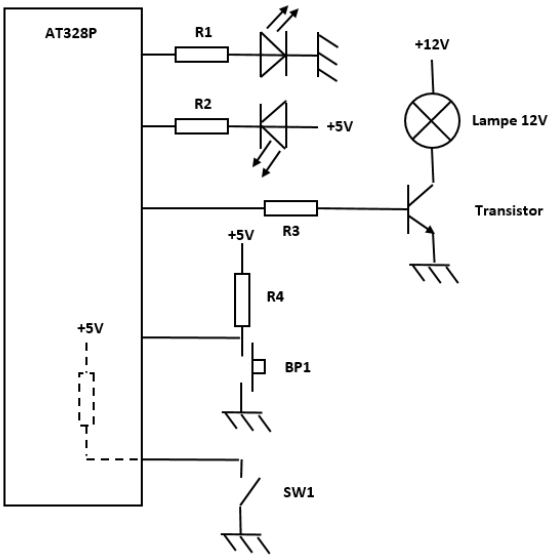
4.3. Les entrées/sorties

Les schémas suivants montrent l'assignation des différentes pattes des centrales :



Les entrées/sorties 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.

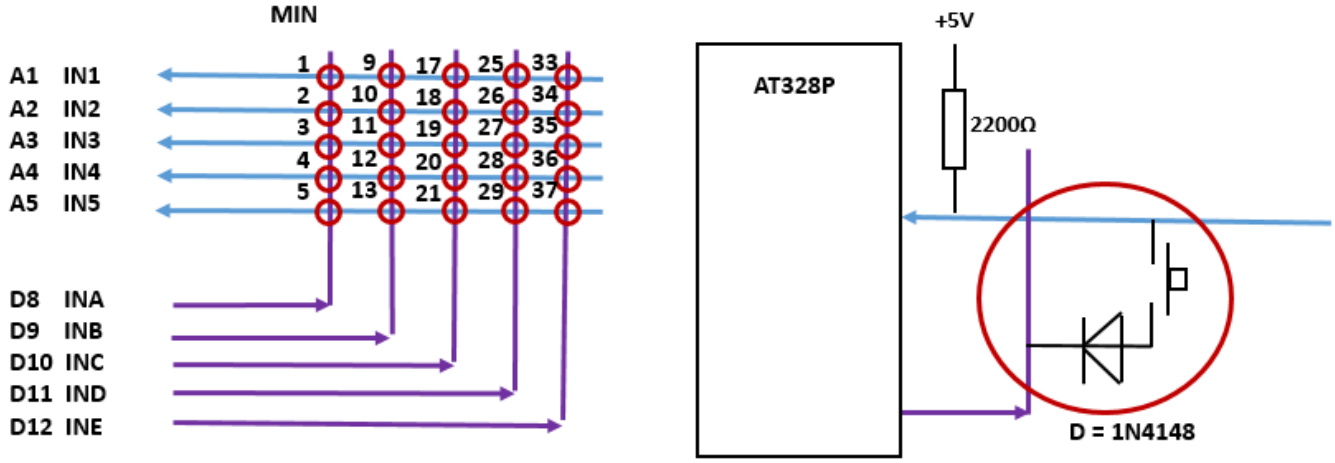


En entrée, le programme peut lire l'état de la patte dont le niveau dépend de ce que l'on y connecte. Par exemple si on y connecte un fil relié à la masse, le programme lira « 0 ». Inversement si on relie ce fil au 5V, le programme lira « 1 ». Du fait des parasites une patte non connectée pourra retourner des « 0 » ou des « 1 ». Afin d'avoir un niveau par défaut, on ajoute souvent une résistance entre la patte et le +5V (résistance de pull-up) ou entre la patte et la masse (résistance de pull-down). Ainsi on pourra détecter l'état inverse de l'état de repos imposé par la résistance.

Par exemple pour détecter l'état d'un interrupteur connecté entre la masse et la patte, on utilisera une pull-up. L'AT328P possède en interne des pull-up (entre 20KΩ et 50KΩ) qu'il est possible d'activer sur chaque patte. Ces valeurs sont un peu trop grandes pour éliminer les parasites sur de long fils. Nous rajouterons des pull-up de 2.2kΩ pour améliorer cela. Plus la résistance est petite, plus le courant est fort et plus il est difficile de perturber l'entrée. Avec 2.kΩ, I=5V/2200Ω=2.3mA (contre 0.10 à 0.25mA). Il ne faut pas non plus avoir un trop gros courant pour ne pas consommer pour rien. Dans le cas de notre matrice comme une colonne est connectée à 5 entrées, c'est un courant 5 fois plus fort (soit 5x2.3=11.5mA) qui est drainé.

4.4. Les entrées matricées (MIN)

La centrale type B gère directement une matrice de 25 entrées. Le choix d'utiliser une matrice permet d'avoir plus d'entrées avec le même nombre de broches. Avec 5 lignes et 5 colonnes donc 10 pattes, on dispose de 5*5=20 entrées. Les colonnes sont prélevées sur les aiguillages. De ce fait à chaque lecture des entrées, les aiguillages sont coupés pendant un bref instant de 5.75us. Cela ne les perturbe pas. La centrale A ne dispose pas de ces entrées.



Chaque ligne est connectée sur une entrée maintenue au niveau haut par une pull-up de 2.2kΩ. Chaque colonne est connectée à une sortie. Le programme met tour à tour ces sorties à la masse (à 0). Si on relie une ligne et une colonne par exemple avec un bouton poussoir, alors la ligne donc l'entrée passera à 0 lorsque la colonne sera à 0 si le bouton poussoir est pressé.

Il faudra rajouter une diode pour éviter les détections fantômes lorsque plusieurs boutons sont pressés. Sans diodes, si les boutons 1, 2 et 9 sont pressé alors 10 sera détecté car quand la colonne INB sera mise à la masse, alors la masse apparaitra sur la ligne IN0, mais aussi sur IN1 par l'intermédiaire des boutons 0 et 1. On utilisera une petite diode comme la 1N4148 qui revient à quelques centimes.

La numérotation des entrées peut sembler bizarre. Cela est dû à une simplification du code qui lit directement le port A0-A7, donc 8 bits pour chaque colonne. Seuls les bits 1 à 5 sont utilisés. 0 est pour le DCC et 6,7 pour an0 et 1.

Il est possible de relier toute sorte de capteur sur ces entrées : bouton poussoir, interrupteur, inverseur, ILS, transistor d'optocoupleur (nous verrons cela pour les capteurs de courant) …

500 fois par seconde, chaque entrée est lue 3 fois. Ce nombre important de lecture permet de ne pas rater les contacts fugitifs comme dans le cas d'un ILS. Les 3 lectures sont synchronisées avec le signal DCC afin de lire pendant les états hauts et bas du signal DCC car la majorité des capteurs de courant ne détectent le courant que dans un sens. Comme le signal DCC change toute les 58 ou 116us, 3 lectures espacées de 58us liront à coup sûr pendant les 2 états.

Notez, que si nécessaire, il est possible d'utiliser la patte A5 (IN5) pour générer le complément du signal DCC (DCC#). Cela peut être utile pour certains boosters. Dans ce cas vous disposerez de 5 entrées en moins. Une autre solution étant d'inverser le signal DCC avec un transistor.

4.5. Les souris

Les souris permettent de commander les locomotives à la main.

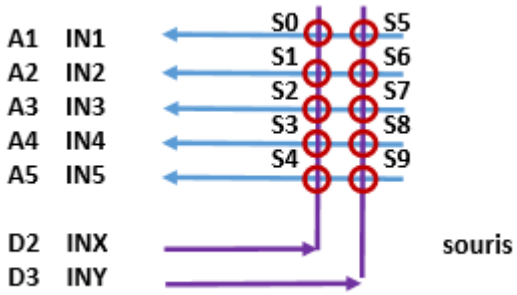
Beaucoup de souris utilisent un potentiomètre pour le sens et la vitesse. Je n'ai pas retenu ce choix car j'ai souvent été embêté par des potentiomètres qui produisent des sautes de tension lorsqu'on les tourne et se font perturber facilement. J'ai remplacé le potentiomètre par un interrupteur à 2 positions instables. On parle d'un inverseur MOM-OFF-MOM. Lorsque le programme détectera un appuie vers le haut, il augmentera la vitesse de la locomotive et la diminuera s'il est poussé vers le bas.

J'ai rajouté un bouton poussoir afin de changer de sens à chaque appuie. Ce bouton poussoir sert également à piloter les fonctions des décodeurs, provoquer l'arrêt d'urgence de l'ensemble du système et sélectionner une locomotive. Cela est possible grâce à des combinaisons d'appuis court (C), long (L env) ou très long (TL) sur ce bouton. La commande est prise en compte après un repos de 1s. Cela est nécessaire pour identifier la fin de la commande. Dans le cas de l'arrêt d'urgence, il n'y a pas d'attente.

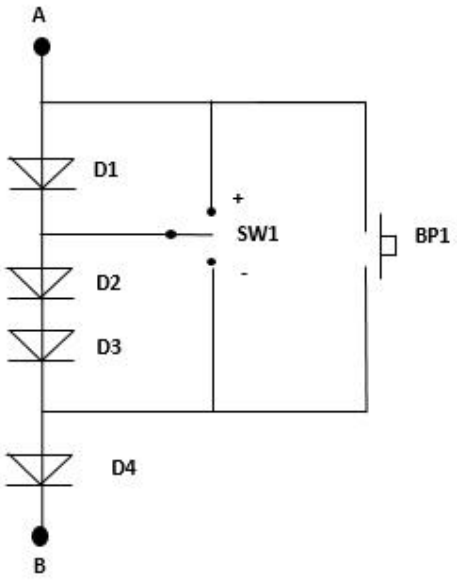
C = changement de sens
C+C = arrêt d'urgence
C+C+C = fin d'arrêt d'urgence
L = changement d'état de la fonction 0 (F0 est souvent appelée FL et gère souvent l'allumage des phares)
L+C = F1
L+C+C = F2
L+C+C+C = F3
L+C+C+C+C = F4
TL + L… + C… = sélection de la locomotive d'adresse (10*nombre de L + nombre de C). Une adresse hors 1-50 désactive la souris.

C < .5s (typique : 0.2s)
L > .5s et < 5s (typique : 1s)
TL > 5s (typique : 10s)

Je voulais absolument que la souris se connecte avec un câble à seulement 2 fils afin de pouvoir utiliser des connecteurs simples. Pour ma part j'utilise des fiches RCA. J'aurais pu opter pour la même solution que fdcc-2008 en faisant passer le courant dans un sens puis dans l'autre pour détecter la position des interrupteurs, mais cela aurait nécessité un peu plus d'électronique. J'ai réutilisé le principe du module MIN en rajoutant 2 colonnes INX et INY qui réutilisent les signaux CLK et DAT durant les intervalles ou S88, Sout, Sled ne sont pas utilisés. Les lignes restent les mêmes que le module MIN. Par chance les lignes du module MIN sont connectés au convertisseur analogique-numérique qui permet de mesurer des tensions. Il est possible d'utiliser jusqu'à 10 souris.



J'aurais pu utiliser des résistances faisant un pont diviseur avec les résistances de pull-up, mais j'ai préféré utiliser des diodes (1N4148) afin de ne pas avoir à utiliser plusieurs valeurs de résistances. La figure suivante présente le schéma électrique et une de mes souris.



La diode D4 est nécessaire pour le module MIN afin d'éviter les détections fantômes. Les autres diodes permettent d'obtenir une chute de tension en fonction de l'état des interrupteurs qui les court-circuitent. Une diode produit une chute de tension de 0.6 à 0.7V environ. Prenons 0.7V pour nos calculs. Si les boutons sont au repos alors aucune diode n'est court-circuitée et la tension est de 4*0.7=2.8V. Le point à étant connectée à une entrée MIN avec sa pull-up de 2.2k au 5v, et le point B à une sortie MIN mise à la masse lors de la lecture. L'entrée voit donc 2.8V et le programme lit aux alentours de 796/1024. Si la souris n'est pas connectée, l'entrée est à 5V par l'intermédiaire de la pull-up. Si On souhaite accélérer et que l'on appuie sur l'inverseur en position « + », alors on court-circuite une diode et l'entrée est donc à 3*0.7 = 2.1V. Dans le cas de la position « – », on court-circuite 2 diodes et l'entrée est donc à 2*0.7 = 1.4V. Si on presse maintenant le bouton poussoir, on court-circuite 3 diodes, et la tension de l'entrée est imposée par la diode restante, soit 1*0.7=0.7V. De cette façon le programme déduit l'état des boutons en mesurant la tension sur la broche d'entrée. Vous remarquerez que lorsque le bouton poussoir est pressé, la position de l'inverseur SW1 n'a aucun impacte. En clair, on ne peut pas connaître sa position mais ce n'est pas important.

La réalisation est facile car il suffit simplement de souder 4 diodes à 2 boutons. Pour le câble, je conseille d'utiliser un bout de câble RCA (Avec son connecteur male). De brancher un connecteur RCA femelle coté centrale. Le prix d'une souris, boitier compris ne devrait pas dépasser 5€.

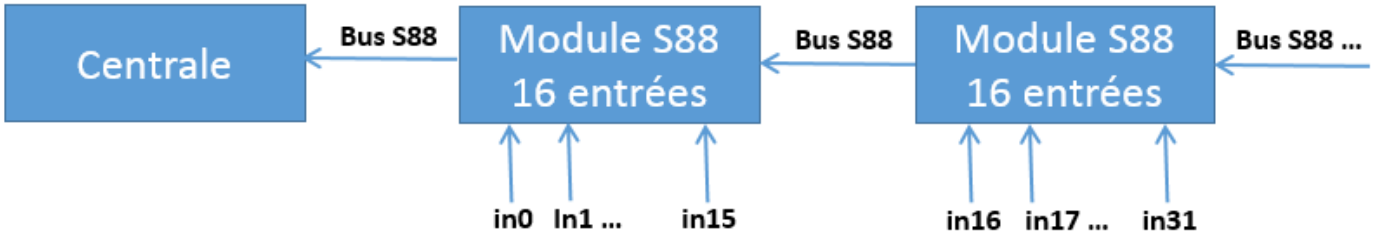
Sans modification du code de l'Arduino, les souris sont associées à aucune locomotive au démarrage. Il est ensuite possible de les sélectionner directement avec la souris ou en utilisant le logiciel PC. En modifiant quelques paramètres du code Arduino, il est possible d'associer une locomotive par défaut à chaque souris afin de ne pas avoir à les sélectionner. Si une locomotive est sélectionnée par plusieurs souris, alors toutes les souris contrôleront cette locomotive ! Idem si la locomotive est contrôlée par le PC, elle reste contrôlable par les souris.

4.6. Les entrées analogiques

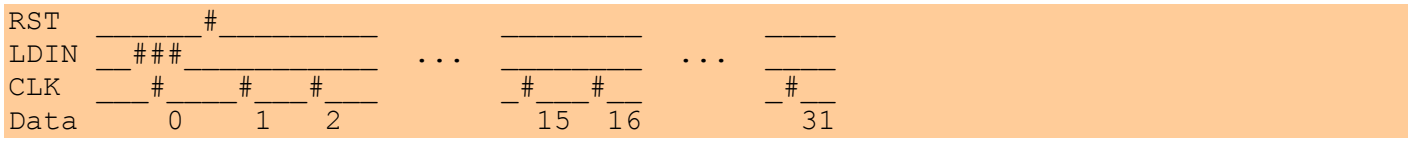
Les centrales disposent de 2 entrées analogiques AN0 et AN1 (sauf sur l'Arduino UNO). Elles peuvent être utilisées pour y brancher des capteurs analogiques. Il est par exemple possible de brancher une LDR pour mesurer la luminosité de la pièce … Une application intéressante serait de mesurer le courant du booster et d'afficher la courbe.

4.7. Le bus S88 (entrées) (centrale A uniquement)

Le bus de rétro-signalisation S88 permet de récupérer l'état des entrées des modules S88. Généralement, les modules S88 permettent de rajouter 8 ou 16 entrées au système. Fdcc supporte au maximum 256 entrées (par exemple 16 modules de 16). Vous pouvez trouver dans le commerce des modules S88 (assez chères) ou les réaliser vous-même. De nombreux montages sont disponibles sur Internet. Lors de la réalisation de free-dcc 2010, j'avais proposé un module à base de PIC. Bien que fonctionnant parfaitement, je ne conseille pas de le réaliser au néophyte car il faut programmer le PIC. Maintenant que vous maitriser les Arduino, je propose de réaliser un module avec un Arduino qui dispose de 16 entrées directes. (Je proposerai une version à 64 entrées matricées plus tard).



Le bus S88 fonctionne de la façon suivante : Une impulsion sur LDIN (Load IN = Charger les entrées), copie l'état des entrées dans un registre à décalage. L'entrée 0 (la première entrée du premier module (le plus proche de la centrale)) apparait sur la ligne DATA connectée sur l'entrée DATIN de la centrale. A Chaque impulsion sur la ligne CLK, le registre à décalage décale les entrées et la 1 remplace la 0 … Cela permet de récupérer toutes les entrées. Afin de ne pas rater une entrée transitoire, les modules intercalent une mémorisation entre l'entrée et le registre à décalage. Ainsi même si l'entrée est active brièvement, elle sera mémorisée. Le signal RST permet d'effacer cette mémoire. Il est généralement effectué juste après le chargement de manière à détecter immédiatement tout état actif. Il est dommage de ne pas avoir combiné LDIN et RST, mais telle est la spécification du bus S88.



Les entrées sont souvent inversées. Au repos, elles sont misent au niveau haut par des pull-up. Les capteurs (ex : interrupteurs) se connectent entre ces entrées et la masse. Ainsi lorsque l'entrée voit la masse la sortie est active. Un « 1 » sera mémorisé.

Réalisation d'un module S88 à 16 entrées avec un Arduino :

Vous trouverez dans l'archive, le « sketch » (programme) module_s88.ino à « téléverser » (télécharger) dans un Arduino UNO ou Nano. Avant de téléverser le programme, configurez-le en renseignant les lignes suivantes :

```
// Choisir la configuration (mettre seulement un 1 pour la configuration selectionne)
#define use_15_in 1   // 15 entrees directes  (la broche CLR est utilisee)
#define use_16_in 0   // 16 entrees directes  (la broche CLR n'est pas utilisee)
#define use_48_in 0   // 48 entrees matricees (la broche CLR est utilisee)      PAS ENCORE DISPO
#define use_64_in 0   // 64 entrees matricees (la broche CLR n'est pas utilisee)PAS ENCORE DISPO

// Choisir si les entrees doivent etre filtrees ou pas
#define use_filter 1
```

Les constantes **use_15/16/48/64_in** permettent de choisir le nombre d'entrées.
Pour l'instant, vous avez le choix entre :
- une configuration standard qui fournit 15 entrées et la broche RST
- une configuration à 16 entrées sans la broche RST (Le RST étant fait par la broche LDIN).

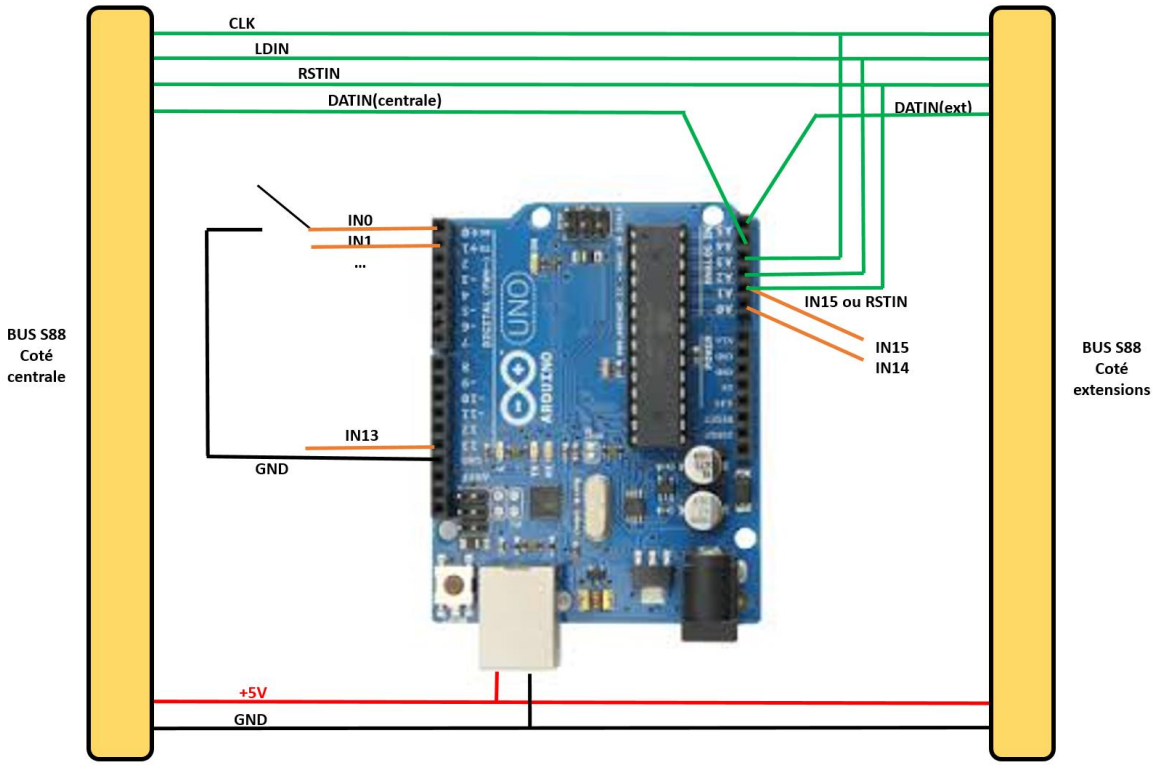
La constante **use_numeric_filter** permet d'activer un filtrage numérique des entrées afin d'éviter les « glitches ». Une entrée est déclarée active seulement lorsqu'elle a été vu 3 fois de suite au niveau bas.

Les entrées sont actives au niveau bas et les résistances de pull-up des entrées de l'Arduino sont activées. Ces pull-up sont assez grosses, n'hésitez pas à en rajouter des externes plus petites (ex : 2200Ω) si vous voyez des perturbations. Les fronts de plus de 30uS sont mémorisés.

Les connections sont les suivantes :

```
// Patte Arduino / Fonction S88

D0  = IN0
D1  = IN1
D2  = IN2
D3  = IN3
D4  = IN4
D5  = IN5
D6  = IN6
D7  = IN7
D8  = IN8
D9  = IN9
D10 = IN10
D11 = IN11
D12 = IN12
D13 = IN13
A0  = IN14

A1  = IN15 ou RSTIN

A2  = LDIN
A3  = CLK
A4  = DATcentrale
A5  = DAText
```

Le shéma suivant montre comment connecter les entrées et le bus S88 sur un Arduino UNO. Je vous laisse extrapoler pour un Arduino Nano. Pour l'alimentation, je conseil d'utiliser du 5V que vous fournirez par le port USB en utilisant un vieux cable USB que vous couperez.



Vous pouvez utiliser les connecteurs de votre choix pour chainer les modules. De plus en plus de monde utilise des connecteurs RJ45 (ceux des réseaux informatiques Ethernet) pour relier les modules par des cables réseaux. Dans ce cas, on parle de S88-N. Rien ne change par rapport au S88 à part l'utilisation des connecteur RJ45.

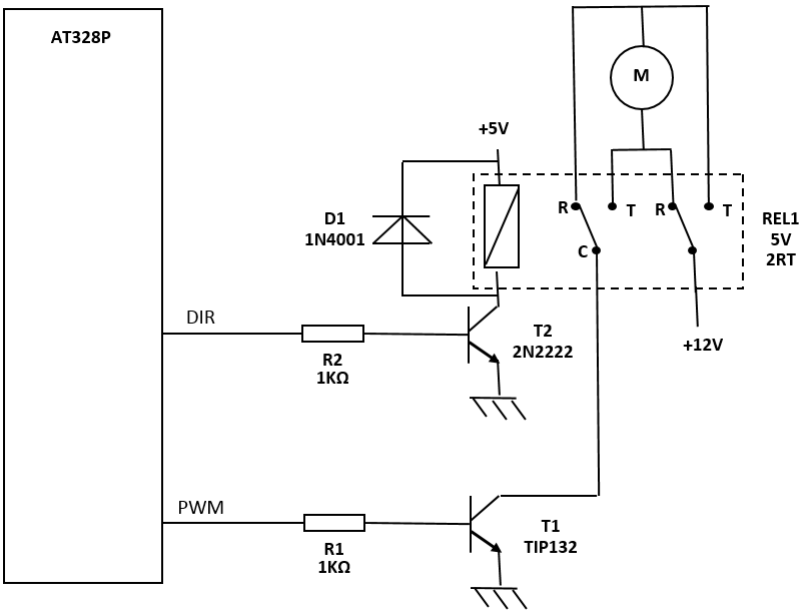
4.8. La PWM

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%
```

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. On pourra donc commander des locomotives analogiques. 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. Si on remplace le moteur par des voies, on pourra alors piloter une locomotive analogique.



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. Le moteur de la locomotive n'est pas alimenté en continu mais par

Dans le cas d'un réseau analogique, on ne pourra contrôler qu'une seule locomotive à la fois par circuit de puissance. Il faudra par exemple couper l'alimentation des autres avec des relais. Pour les plus ambitieux, il est possible de mixer les locomotives numériques et analogiques, en les isolants avec des relais, et en choisissant des détecteurs capables de détecter les 2 type de locomotives.

L'avantage de la PWM sur une tension continue est que la locomotive capte mieux le courant et autorise des ralentit spectaculaires car la tension maximale est toujours utilisée. De plus, 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 ! Les PWM de l'Arduino fonctionnent à 1000Hz et ceux des PCA9685 sont réglés à 50Hz

La centrale A possède 4 sorties PWM (PWM48-51). La B en possède 2 (PWM48-49). Il faut choisir entre les PWM18-49 et le bus I2C. Mais i on utilise ce dernier, on perd certes 2 PWM, mais on peut connecter jusqu'à 3 PCA9685 dont chacun dispose de 16 sorties que l'on peut utiliser unitairement en mode PWM ou servomoteur.

4.9. Les locomotives analogiques

Bien que free-DCC soit principalement conçu pour les locomotives numériques DCC, il est possible d'utiliser occasionnellement ou régulièrement des locomotives analogiques (donc sans décodeur). Il serait en effet dommage de ne pas pouvoir utiliser une ancienne locomotive occasionnellement sous prétexte qu'elle n'a pas de décodeur ! Ceux qui possèdent un réseau purement analogique pourraient aussi être tentés d'utiliser free-DCC uniquement en analogique pour les nombreuses fonctionnalités qu'il propose.

Afin de garder toutes les fonctions d'automatisation offerte aux locomotives numérique, la centrale peut commander des alimentations pour les locomotives analogiques en fonction des ordres s'adressant à une locomotive numérique ! Par exemple, on peut configurer la centrale pour qu'elle commande une alimentation à partir des ordres reçus pour la locomotive numérique d'adresse 50. Ainsi chaque fois qu'une souris physique ou virtuelle change la vitesse de la locomotive 50, l'alimentation analogique change en conséquence. Cela marche aussi pour tout ce qui touche à l'automatisation et vous pouvez déplacer une locomotive analogique comme une locomotive numérique !

Il est possible de réaliser une alimentation analogique avec 2 sorties PWM. Ces sorties sont disponibles sur l'Arduino (2 sur la centrale A et 4 sur la B) ou sur les modules I2C à base de PCA9685 (16 par modules). Une sortie est utilisée pour générer le signal PWM servant à faire varier la vitesse. Le second gère le sens. Il est dommage d'utiliser un signal PWM pour le sens qui n'a que 2 valeurs (0V=avant, 5V=arrière), mais cela simplifie la réalisation et laisse la possibilité d'utiliser dans le futur des amplificateurs qui nécessitent 2 sorties PWM (une par sens).

Se référer au chapitre précédent sur la PWM pour réaliser un amplificateur avec :
- 1 transistor pour la PWM et un relais inverseur pour le sens
- 1 pont en H comme celui utilisé par le booster

La centrale gère jusqu'à 5 alimentations analogiques. Vous remarquerez que le terme alimentation analogique est inexact car la tension de sortie n'est pas une tension variable mais un signal PWM qui vaut soit 0V

Si vous utilisez plusieurs locomotives analogiques simultanément sur une zone alimentée par la même alimentation analogique alors vous ne pourrez en utiliser qu'une à la fois et il vous faudra créer des zones de coupure pour isoler les autres. Vous pouvez utiliser des interrupteurs ou alors utiliser des relais contrôlés par free-DCC. Une manière de faire réel est de les contrôler par des commutateurs sur le TCO comme cela se fait en réalité pour couper le courant dans les caténaires, pour nous, ce sera couper le courant dans les rails !

Rien n'empêche de mixer les locomotives numériques et analogiques s'il est possible de les isoler avec des zones de coupure et de commuter les alimentations (le booster ou une alimentation analogique). Si un décodeur reçoit un signal autre que la PWM, certains ne font rien, d'autre arrêtent la loco, d'autre se comportent comme une locomotive analogique et d'autres encore sont configurables…

Notez que le booster est tout indiqué pour réaliser une alimentation analogique.
Vous pouvez ajouter un inverseur double sur les entrées de vitesse et sens ou utiliser un relais contrôlé par free-DCC pour sélectionner :
- En numérique (DIR=DCC, VIT=5V)
- En analogique (DIR=SENS, VIT=PWM)

Il serait possible dans le futur si le besoin s'en fait sentir de se passer de cet interrupteur en connectant une sortie PWM sur l'entrée VIT du booster. Il suffirait de la laisser à 5V lorsque le booster est utilisé en DCC. Bien entendu il faudrait générer le signal DCC ou de sens sur DIR. Le choix du mode pourrait alors se faire par logiciel. Mais ne compliquons pas pour le moment !

Bien entendu, l'arrêt d'urgence fonctionne aussi pour les locomotives analogiques en mettant le signal PWM qui commande la vitesse à 0%

En analogique certains détecteurs de courant conçu pour le DCC peuvent ne pas marcher pour la simple raison qu'en DCC le courant passe dans les 2 sens contrairement à l'analogique ou il ne passe que dans le sens donné par la direction. De plus il n'y a aucun courant à l'arrêt … Mais rien d'insurmontable, des montages existent !

Attention si vous utiliser l'alimentation du booster pour alimenter des locomotives analogiques. En effet cette tension est souvent choisie entre 15 et 18V. Les locomotives risquent de ne pas aimer. La configuration permet de régler ce problème en définissant la PWM max donc la tension moyenne maximale.

Pour la configuration des alimentations analogiques, il suffit de paramétrer les 5 fonctions suivantes :

```
// locomotives analogiques
// La fonction add_alim_analog(adr, pwm_vit, pwm_dir, pwm_cran1, pwm_cran28)
// permet de remplacer une loco numérique par une alimentaion analogique
// Pour le système, l'alimentation analogique obeit aux ordres de la loco numérique associée !
// adr        = adresse de la loco numérique associée (1-50).
//              Si adr==0 l'alimentation analogique n'est pas utilisée.
// pwm_vit    = numéro de la broche pwm gérant la vitesse (0-51)
// pwm_dir    = numéro de la sortie pwm gérant le sens    (0-51).
// pwm_cran1  = valeur de la pwm (0-255) pour le cran numérique  1 (la locomotive doit démarer)
// pwm_cran28 = valeur de la pwm (0-255) pour le cran numérique 28 (la locomotive doit démarer)
// vous pouvez définir jusqu'a 5 alimentations analogiques

#define CONFIG_ALIM_ANALOG \
add_alim_analog(50, 48, 49, 28, 170); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
```

Dans cet exemple, seule une alimentation analogique est définie. Elle est contrôlée par les ordres de la locomotive numérique d'adresse 50.

La sortie PWM48 est utilisée pour générer le signal PWM pour la vitesse tandis que la sortie PWM49 est utilisée pour contrôler le sens.

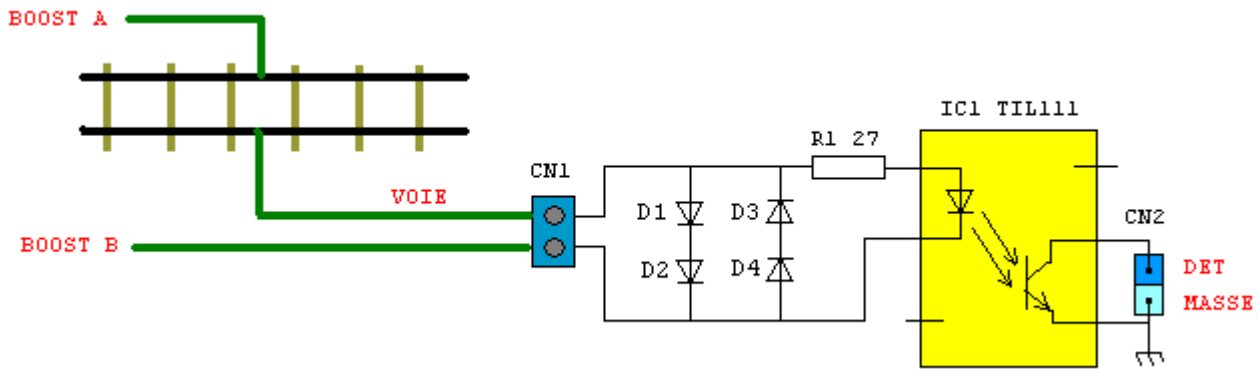
La valeur du signal PWM peut varier entre 0 (0%) et 255(100%). En numérique les ordres de vitesses sont sur 28 crans (en plus de l'arrêt). Les 2 dernières valeurs permettent de choisir :

- La valeur de la PWM pour le cran1 qui correspond au seuil de démarrage de la locomotive. Supposons que L'alimentation analogique soit alimentée en 18V, avec 28 on obtient une tension de (28/255)*18 = 2V. En jouant sur cette valeur, la locomotive démarrera dès le cran1.

- La valeur de la PWM pour le cran28 qui correspond à la tension moyenne maximale. Avec 18V et une valeur de 170, on obtient (170/255)*18=12V

4.10. Les détecteurs de courant

Les détecteurs de courant permettent de détecter l'occupation des voies par les trains en toute discrétion contrairement aux pédales de voies ou ILS. En fait ils détectent les décodeurs des locomotives, les moteurs ou les éclairages des voitures. Si vous souhaitez également détecter les wagons, il suffit de rajouter une résistance de 4.7 K sur les essieux ou graphiter ces derniers. Les détecteurs sont particulièrement bien adaptés pour la détection sur des sections de voies ou cantons contrairement aux pédales de voies ou ILS qui ont un rôle de détection plus ponctuelle. Il est également possible d'utiliser de courtes sections pour des détections ponctuelles. Avec un prix de revient de 2 euros par section de détection il ne faut pas vous en priver.



Le schéma précédent présente un détecteur pour une voie.
Le booster délivre alternativement +15V et -15V. Ceci se traduit lorsqu'une locomotive est présente par un courant passant par D1 et D2 dans un sens puis D3 et D4 dans l'autre. Dans le cas contraire aucun courant ne passe. Lorsque du courant passe dans D1et D2, ceci provoque une chute de tension de 1.2V au minimum aux bornes des diodes ce qui alimente la LED de l'optocoupleur par l'intermédiaire de la résistance R1 de faible valeur. La diode éclaire le phototransistor de l'optocoupleur ce qui provoque la mise à la masse de l'entrée de détection. Cette mise à la masse sera ensuite mémorisée et remontée à la centrale par un module S88. Bien entendu les diodes doivent supporter le courant maximal du booster soit 3A dans notre cas comme les BY255.

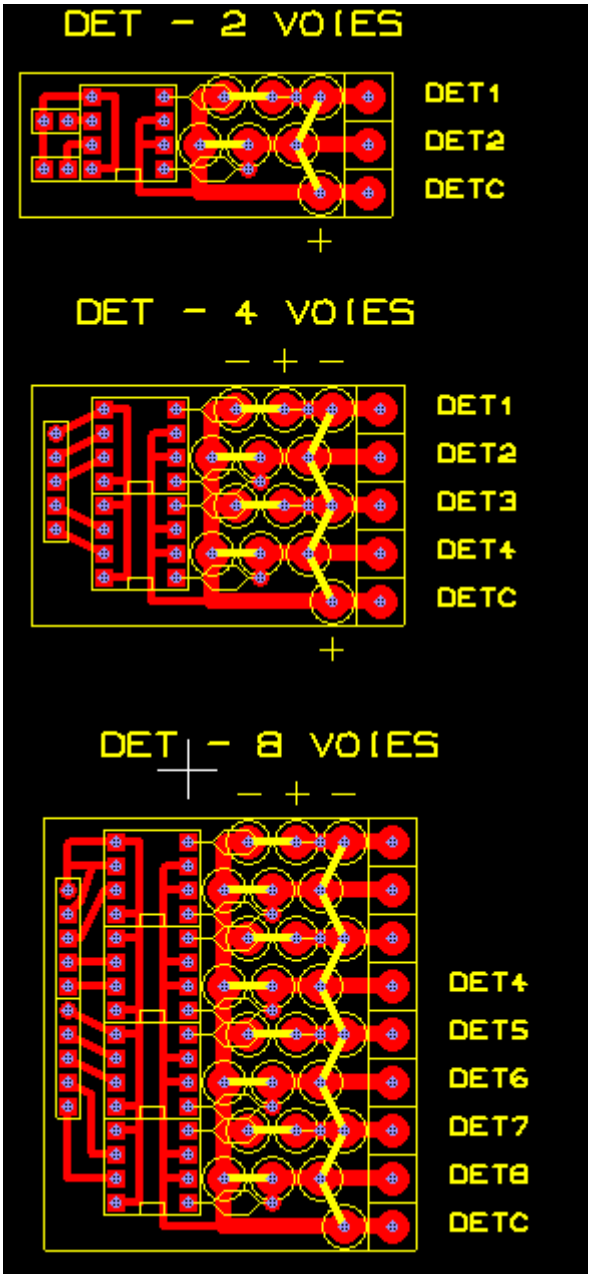
D2 peut bien entendu être supprimée, mais cela déséquilibre un peu le signal. Dans ce cas en tenir compte avec les décodeurs Gold de Lenz qui peuvent faire des actions particulières en cas de déséquilibre du signal.
En cas de regroupement de plusieurs détecteurs, il pourrait être tentant de mettre une D4 commune, mais ce serait une grave erreur source de détections parasites.

Vous pouvez extrapoler le schéma pour un détecteur à 4, 8 voies ou à un nombre quelconque de voie.
Au niveau de la réalisation, vous pouvez utiliser de la plaque à trou ou les typons suivants.
Avec les typons, les résistances sont à souder coté cuivre.

Dans le cas d'utilisation d'un module S88, on reliera DET à l'entrée correspondante et MASSE à la masse du système.
Dans le cas d'utilisation d'une entrée MIN, on connectera DET à une des entrées de la matrice (IN0-IN5) par l'intermédiaire d'une diode. La MASSE ne SERA PAS CONNECTEE à la masse du système, mais à une sortie de la matrice (INA-IND). Si vous constatez des détections fantômes surtout lors d'un court-circuit sur les voies, il faudra rajouter un petit condensateur sur les entrées S88. Pour les entrées MIN un filtrage numérique a été rajouté dans le programme de la centrale. Ce phénomène est dû à un couplage électromagnétique des fils/pistes avec les forts courants.

En analogique pure avec une alimentation qui délivre une tension continue variable ce montage ne fonctionnera pas lorsque la tension sera inférieure à 2V et le sens du courant inverse au sens de fonctionnement de la LED de l'optocoupleur. Il faudra alors utiliser un autre montage.

En analogique PWM, le montage tel quel ne fonctionnera pas lorsque les créneaux sont au niveau bas et lorsque le sens de la locomotive est inverse au sens fonctionnement de la LED de l'optocoupleur. Mais il est possible de le faire fonctionner en ajoutant une résistance entre la masse et BOOST-B, ainsi qu'une autre entre BOOST-A et le plus de l'alimentation de puissance. Ainsi lors du créneau à 0, la LED sera alimentée à travers les résistances. En inversion, ça ne marchera toujours pas, mais en utilisant une PWM à 99%, le 1% de créneau à 0 sera suffisant pour faire fonctionner le détecteur …


DET - 2 VOIES
DET1
DET2
DETC
DET - 4 VOIES
DET1
DET2
DET3
DET4
DETC
DET - 8 VOIES
DET4
DET5
DET6
DET7
DET8
DETC

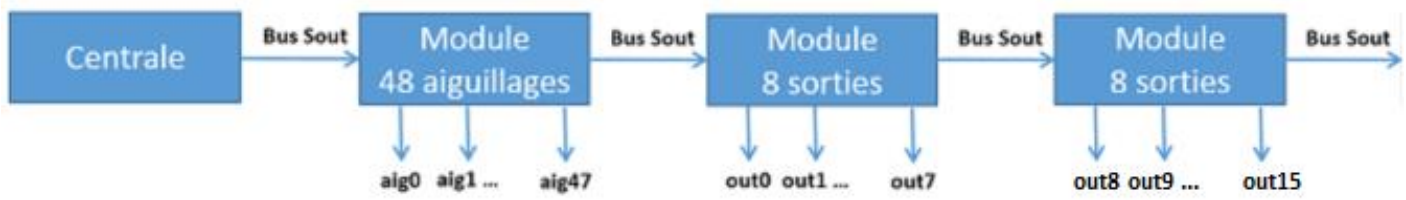
4.11. Le bus Sout (sorties et aiguillages) (centrale A uniquement)

Le bus Sout fonctionne de la manière inverse à celle du bus S88. La centrale commence par mettre la dernière sortie du dernier module sur DATA puis la décale par une impulsion sur CLK et ainsi de suite jusqu'à avoir décalé la première sortie du premier module. A ce moment-là, la centrale génère une impulsion sur la ligne LDOUT (Load Out = Chargement des sorties). A ce moment-là, les valeurs du registre à décalage passent sur les sorties.

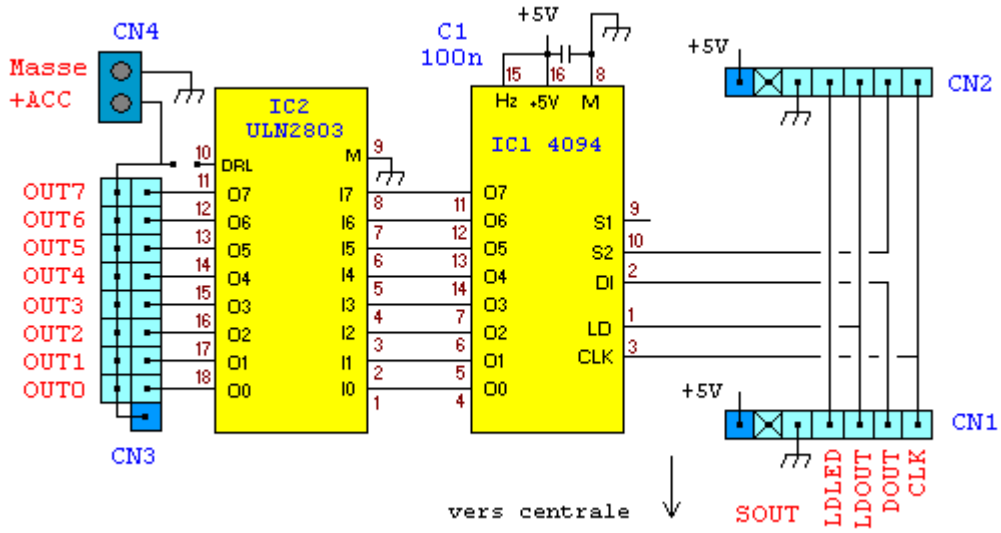
```
LDOUT ________________   _______#_
CLK    __#___#___#___   _#___#___
Data    63   62   61      01   00
```

Free dcc 2017 réutilise les modules à 8 sorties et 32 aiguillages de free-dcc 2010. Les modules d'aiguillages passent de 32 à 48 aiguillages par l'adjonction d'un 3eme L293D connectée aux 2 sorties restantes. Free dcc 2017 supporte 8 modules à 8 sorties (soit 64 sorties au total) et 2 modules de 48 aiguillages (soit 96 aiguillages au total). Free dcc 2010 imposait d'avoir le seul module 32 aiguillages au début de la chaine. Maintenant le placement est libre, mais il faudra l'indiquer dans le programme en modifiant la ligne suivante :

```
//du plus proche au plus loin de la centrale
unsigned char sout_config[] = { SOUT_AIG0_47, SOUT_OUT0_7, SOUT_OUT8_15, SOUT_OUT16_23,
SOUT_OUT24_31, SOUT_AIG48_95,  SOUT_OUT32_39, SOUT_OUT40_47, SOUT_OUT48_55, SOUT_OUT56_63,
SOUT_END };
```



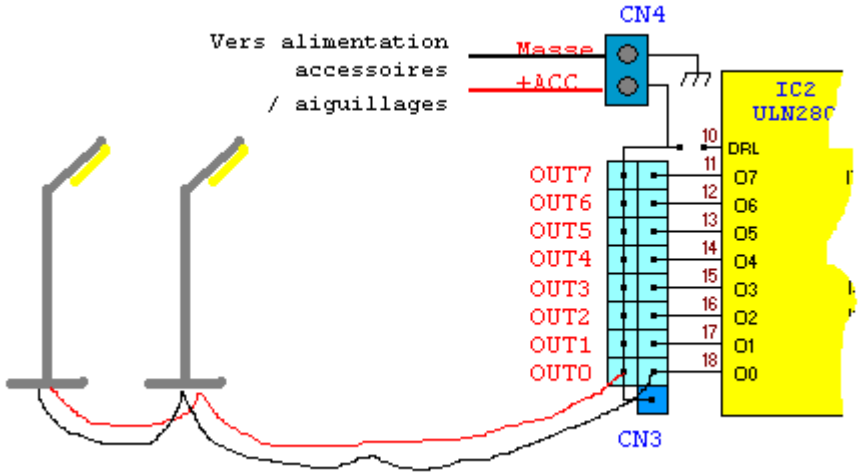
Le module 8 sorties (doc de fdcc2010)



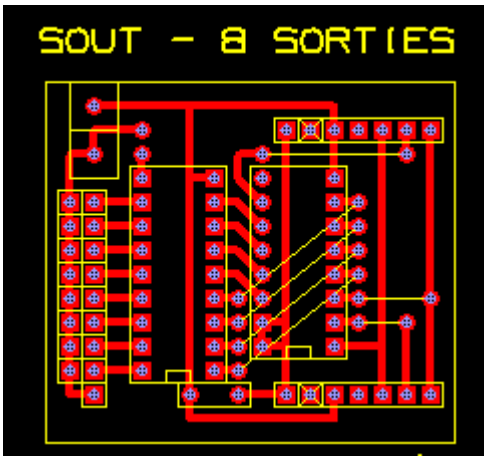
Ce module de sortie permet de contrôler 8 sorties de 500mA chacune. Il est composé du registre à décalage IC1 qui reçoit les données en provenance de la centrale par l'intermédiaire de sa broche DI au rythme de l'horloge CLK. Les données à destination des autres modules ressortent par la broche S2. Lorsque toutes les données sont correctement décalées alors la centrale les charge dans les registres de sorties grâce à la patte LD. A ce moment-là les nouvelles données de sortie sont disponibles en sortie d'IC1 et contrôlent le réseau de transistor Darlington qui active ou pas les sorties. Quelques connecteurs ainsi que le condensateur de découplage complètent le montage.

Pour la centrale, le numéro de la sortie est égal à celui indiqué à proximité du connecteur plus 8 fois le nombre de 4094 qui précédent le module. Par exemple s'il y a sur le bus un module d'aiguillage (qui possède 2 4094) puis un module de sortie (avec un 4094), et enfin notre module de sortie, alors le numéro de la sortie OUT 5 sera 3*8+5=29

Lorsqu'une sortie est activée, l'ULN283 la connecte à la masse alors que dans le cas contraire elle est en l'air. Il faut donc brancher la charge entre le + d'une alimentation et cette sortie (Le – de l'alimentation étant relié à la masse du montage). Dans la plupart des cas, les charges seront alimentées par l'alimentation accessoire de 15V également utilisée par les aiguillages. Dans ce cas, il suffit de relier cette alimentation sur CN4 et brancher les charges sur CN4. Si vous utilisez des charges inductives, penser à rajouter une diode de roue libre en parallèle afin d'éviter les pics de tensions qui ne manquent pas de se produire lorsque l'on coupe la tension à une bobine. Si toutes les charges sont alimentées par la même alimentation alors il est possible d'utiliser les DRL du circuit en connectant la patte 10 à l'alimentation. La figure suivante montre comment relier un ensemble de 2 lampadaires (qui s'allumeront en même temps) sur la sortie OUT0



Coté réalisation vous pouvez utilisez de la plaque pastillée percée ou le typon suivant :

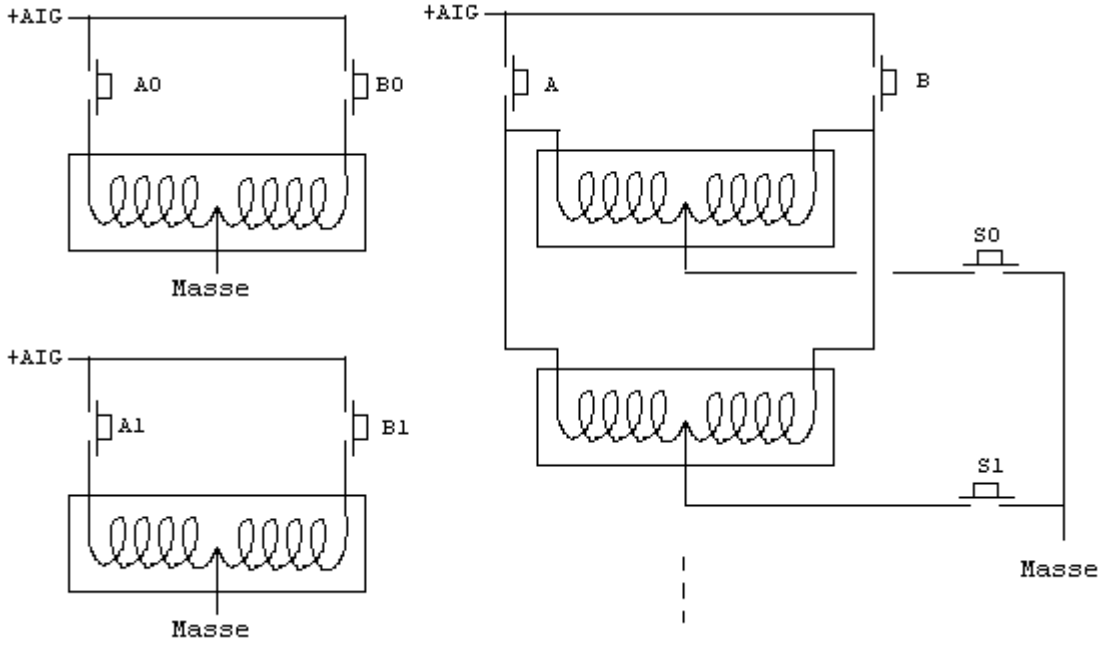


Le module 48 aiguillages (doc de fdcc2010 à 32 aiguillages)

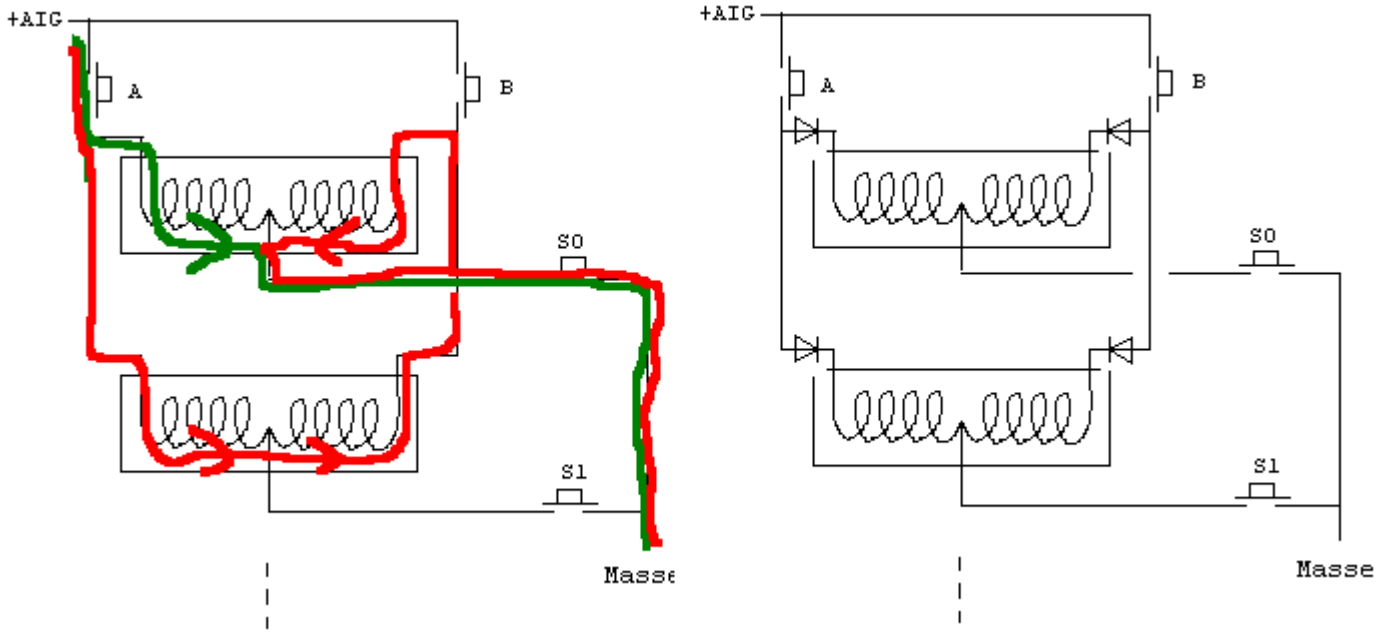
Ce module permet de commander jusqu'à 32 aiguillages. Il a été conçu pour des aiguillages à solénoïdes, mais en rajoutant quelques relais, il est également capable de commander des aiguillages à mouvement lents.

Avant de décrire le schéma revenons sur la commande des aiguillages.

Un aiguillage à solénoïde peut se commander simplement avec 2 interrupteurs ou plutôt boutons poussoirs si on tient à la vie des bobines. Dans ce mode chaque bouton contrôle une bobine ce qui positionne l'aiguillage en direct ou en dévié.



Le problème est qu'il nous faudrait 64 interrupteurs électroniques pour commander nos 32 aiguillages. Une solution intermédiaire serait d'utiliser 2 interrupteurs selon la position et un autre pour sélectionner chaque aiguillage. Ceci nous amènerait donc à 34 interrupteurs (2 avec le + et 32 avec la masse).



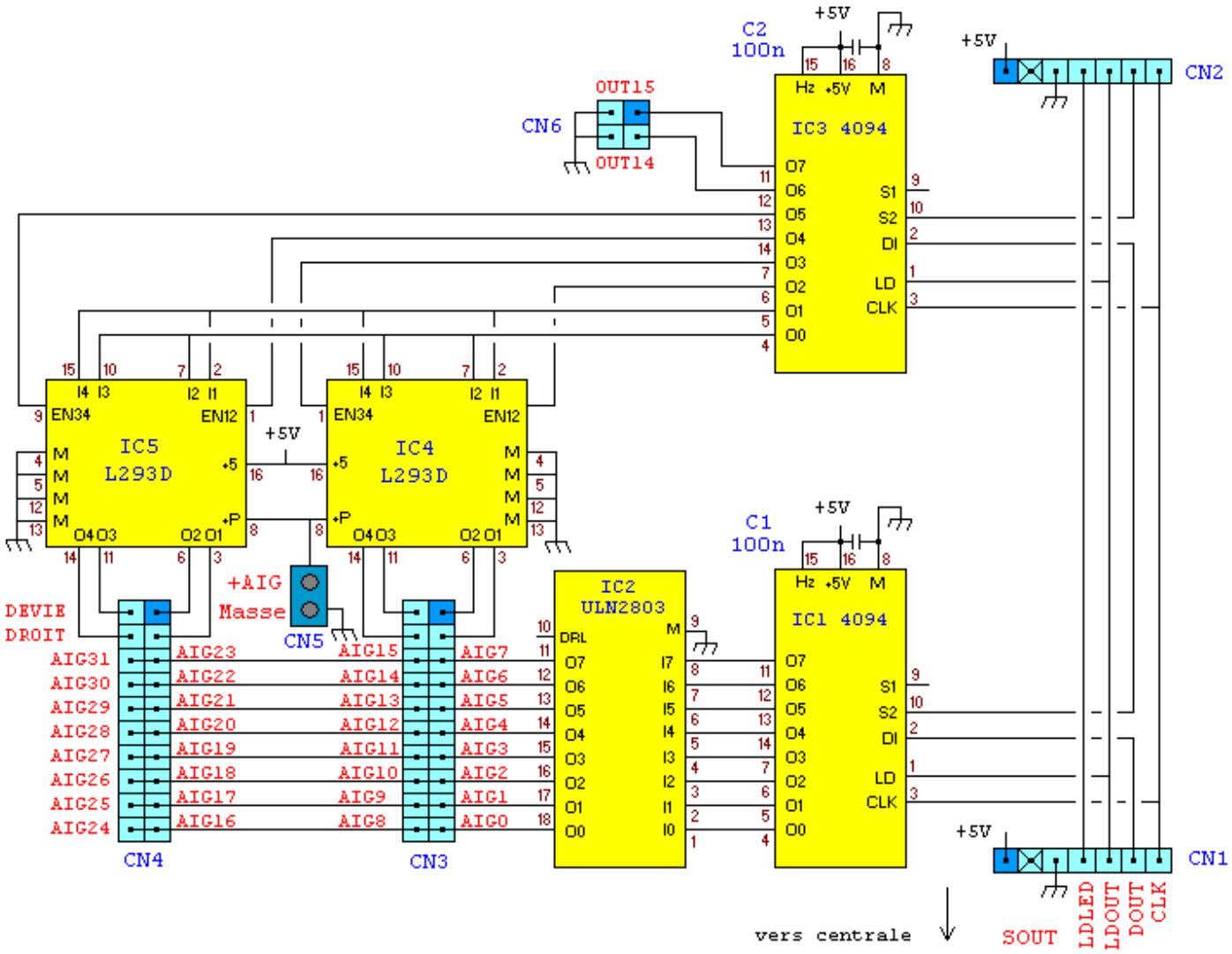
Il est à noter que ce système ne fonctionne pas correctement car le courant passe également par des interrupteurs qui ne sont pas sélectionnés. Pour y remédier, il suffit d'ajouter une diode sur chaque bobine.

Maintenant que nous avons une structure de commande matricée, nous allons réduire le nombre d'interrupteur en répartissant nos aiguillages en 4 groupes de 8. Avec cette astuce nous n'avons besoin plus que de 16 interrupteurs (8 avec le + et 8 avec la masse).

Pour les interrupteurs à la masse, nous utiliserons un ULN2803 qui en possède 8. Il autorisera la commande d'aiguillages de 500mA. Pour de plus grosse intensité, il suffit de superposer un 2eme ULN au premier. Il n'y a pas de risque de surchauffe vu le faible temps de commande.

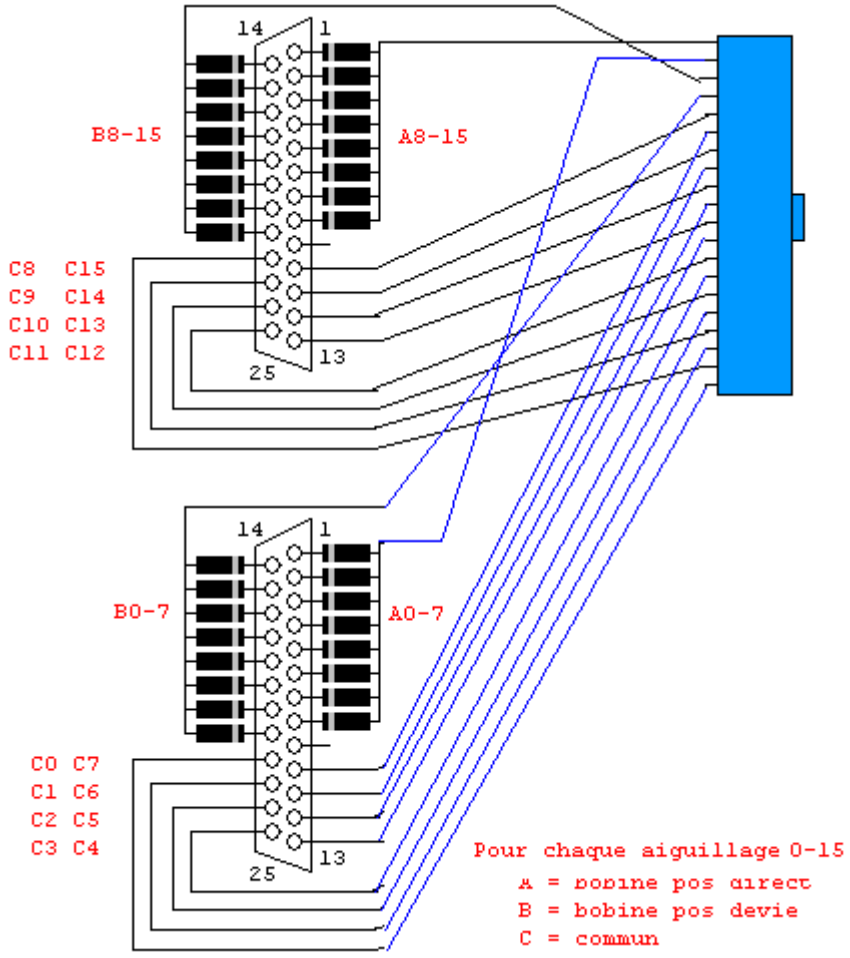
Pour les interrupteurs au plus, nous utiliserons des L293D. Chaque circuit dispose de 4 demi pont, qui peuvent chacun être utilisé pour commuter la sortie au + ou à la masse. Avec les diodes précédentes, nous utiliseront uniquement le +. Pour nos 4 groupes de 8 aiguillages, il nous faudra 8 interrupteurs et donc 2 L293D. Ce circuit est capable de commander des charges de 700 mA ce qui est suffisant pour la plupart des aiguillages.

Avec ces informations, l'étude du schéma est maintenant aisée. On retrouve 2 4094 pour extraire les données série du bus SOUT. IC1 commande les 8 interrupteurs à la masse d'IC2, tandis qu'IC3 commande les 8 interrupteurs au + d'IC4 et IC5. Au lieu de commander directement les interrupteurs au + avec 8 sortie, j'ai préféré contrôler la position des aiguillages avec les 2 première sortie puis ensuite activer le bon groupe d'aiguillage avec les entrées « enable » des L293D. Ceci permet de sauver 2 sorties qui peuvent être utilisées pour commander ce que vous voulez. Mais elles offrent surtout la possibilité d'une future extension à 48 aiguillages en ajoutant un L293D supplémentaire.



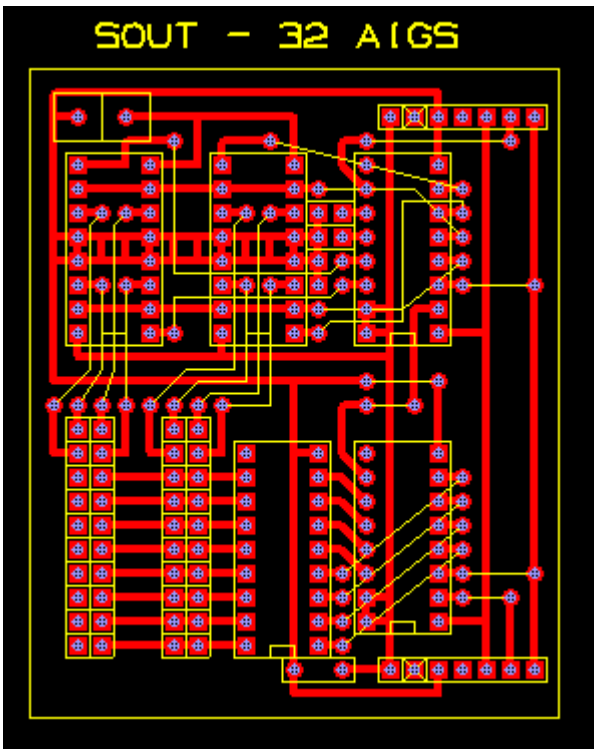
Le connecteur CN5 est destiné à recevoir l'alimentation 15V des aiguillages

Les connecteurs CN3 et CN4 permettent chacun de commander 16 aiguillages. Pour la connexion aux aiguillages, je conseille de fabriquer l'adaptateur suivant. Sertir un câble nappe sur un connecteur HE10 2*10 femelle, détacher les fils de la nappe et y connecter 2 connecteurs DB25 sur lesquels on soude les 32 diodes nécessaires aux 16 aiguillages. La figure suivante explicite le concept.

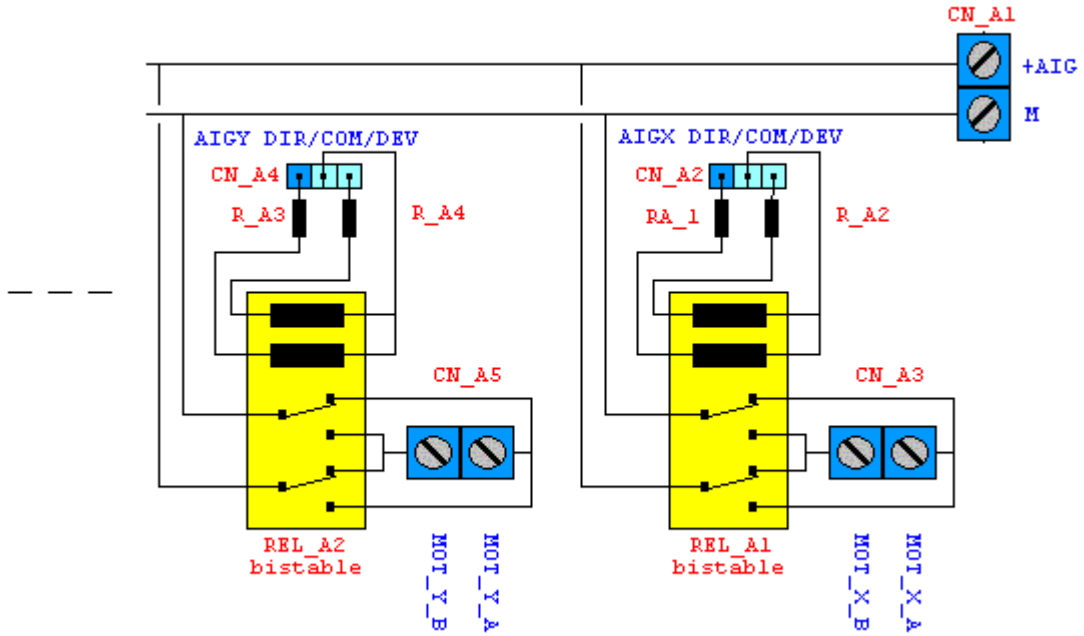


.

Coté réalisation vous pouvez utiliser de la plaque pastillée percée ou le typon suivant :



Pour les aiguillages à mouvement lents qui utilisent des moteurs à la place des solénoïdes, il est possible de les commander en interposant un relais bistable câblé en inverseur entre la carte et chaque aiguillage. En effet un relais bistable se comporte comme un aiguillage classique à solénoïde. Le câblage des contacts en inversion, inverse le courant ce qui fait tourner le moteur dans un sens ou dans l'autre. Pas besoin de gérer la durée car ces aiguillages ont des contacts pour couper l'alimentation du moteur en fin de course. Le schéma suivant montre la réalisation d'une carte pour 2 aiguillages à moteur. Pour un plus grand nombre, il suffit de multiplier l'exemple. N'oubliez pas les diodes entre la carte free-dcc et les relais. Les résistances (dont les valeurs sont à calculer) sont nécessaires si la tension des bobines est différente de celle les commandant.

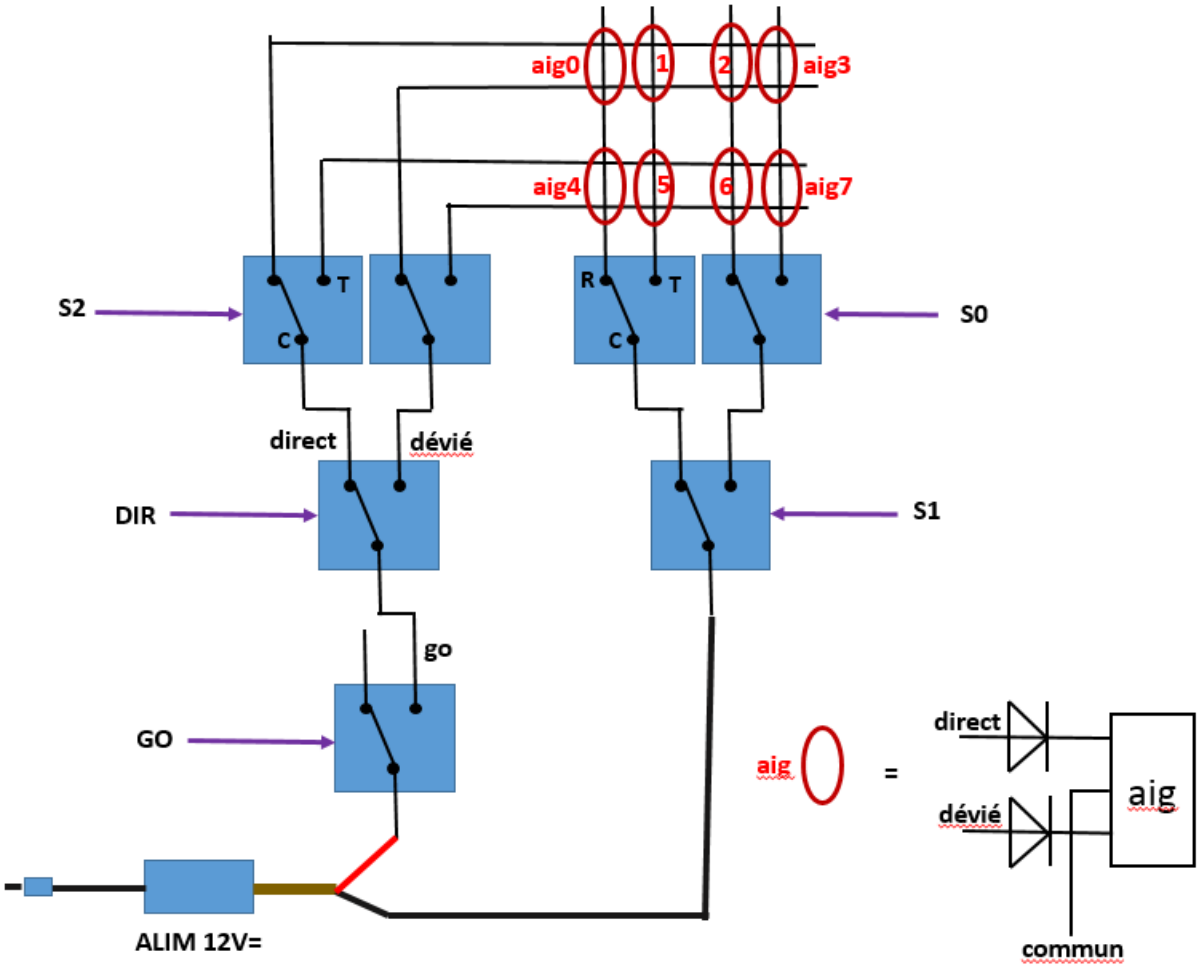


4.12. Les aiguillages (centrale B uniquement))

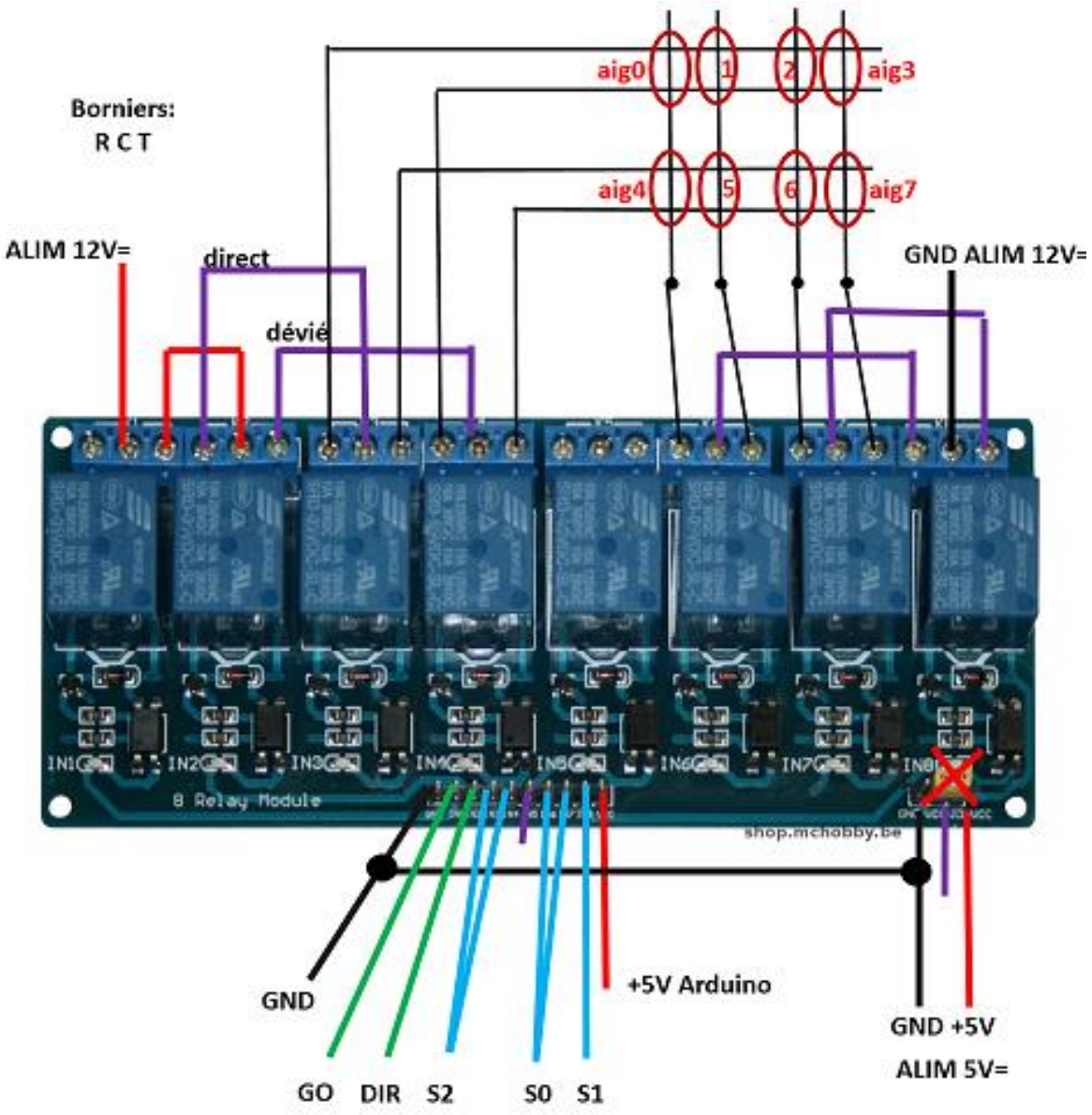
La centrale B permet de commander jusqu'à 16 aiguillages avec 6 pattes sans passer par les modules Sout.

AIG-GO s'active lorsqu'il faut mouvoir l'aiguillage. Sa durée est réglable parmi 125ms, 250ms, 500ms, 1s, 2s, 5s et 10s. AIG-DIR indique la position de l'aiguille (0=position directe, 1=position déviée). S0 à S3 sélectionnent l'aiguillage à déplacer comme suit : numéro de l'aiguillage = 8*S3 + 4*S2 + 2*S1 + S0. Par exemple pour S3=0, S2=0, S1=1, S0=0, ce sera l'aiguillage 2 qui sera sélectionné entre les aiguillages 0 et 15.

Vous pouvez par exemple utiliser quelques relais pour contrôler vos aiguillages. De tels modules sont disponibles par exemple sur Amazon pour quelques euros. Il s'agit souvent de relais à un seul RT d'où le besoin d'un grand nombre de relais. On peut réduire ce nombre en réalisant soit même son module avec des relais 2RT. La figure suivante montre une interface à relais pour 8 aiguillages avec 7 relais 1RT ou 3 1RT + 2 2 RT. Il faudra bien entendu rajouter un peu d'électronique pour commander les bobines des relais ou utiliser un module qui la prévoit. Pour 4 relais 1RT ou 2 2RT ou 1 4RT contrôlés par S3, on passe de 8 à 16 aiguillages. Bien entendu, il est possible d'utiliser de l'électronique de puissance sans relais comme les modules à Sout à 48 aiguillages. Se reporter à cette section pour plus d'information.



Exemple en utilisant un module à 8 relais de chez amazon (5€) :



Ce module est constitué de 8 relais 1RT dont les bobines s'alimentent en 5V. Les concepteurs ont été un peu paranoïaques car ils utilisent des optocoupleurs pour commander des transistors qui alimentent les bobines !

Le connecteur principal de 10 broches reçoit, le +5V, les 8 entrées et la masse.
Ce +5V est utilisé pour alimenter les leds témoins et les leds des optocoupleurs. On peut utiliser le +5V de l'Arduino.

Le connecteur auxiliaire de 3 pattes expose : la masse, le +5V, le +5V des bobines des relais.
Si on laisse le jumper, alors le +5V des bobines est relié au +5V du connecteur principal.
Je conseille de ne pas alimenter les bobines avec le +5V de l'Arduino car avec 50mA par bobine, avec les 8 bobines, on atteint 400mA. On enlèvera donc le jumper et on connectera le +5V d'une alimentation indépendante sur la patte JD-VCC (la plus droite). Bien entendu on reliera la masse de cette alimentation à l'Arduino

Attention, la commande est inversée. En effet, c'est en mettant les entrées à la masse que le relais colle. Une entrée à 5V ou en l'air ne fera pas coller le relais. Il faudra donc inverser les commandes en mettant à 1 l'option suivante :

```
// inversion la commande directe des aiguillages
#define use_inv_aig_direct_cmd 0 // 1 = pour les modules de relais qui collent à l'etat bas au lieu de l'etat haut
```

Vous utiliserez ensuite les borniers à vis pour y connecter les aiguillages.
Il est possible de commander 2 aiguillages supplémentaires avec le relais non utilisé que l'on commandera par S3.

La configuration par défaut de l'ensemble des aiguillages est définie par l'option suivante :

```
// Configuration par defaut des aiguillages:
#define AIG_DEFAULT_INIT 0x32
```

Il est ensuite possible de définir une configuration spécifique pour chaque aiguillage n'utilisant pas la configuration par défaut :

```
// Configuration par defaut des aiguillages:
#define AIG_DEFAULT_INIT 0x30

// Configuration specifique de chaque aiguillages:
#define AIG_SPECIFIC_INIT \
aig_data[0] = 0x30; \
aig_data[1] = 0x30; \
// La valeur permet de spécifier l'inversion de commande, la duree d'impulsion, la position
souhaitee a l'initialisation
// inv.pulsems.pos1.pos0.pos.cmd
// pulsems fait 3 bits: 0=aiguillage non utilise 1=125ms 2=250ms 3=500ms 4=1s 5=2s 6=5s 7=10s
// premier digit: code de la duree de l'impulsion (+8 si inversion de commande, mais attention
on peut passer en hexa)
// 2eme digit   : 0=initialisation directe / 1=initialisation deviee
// Ne pas oublier le \ pour les initialisations specifiques car il s'agit d'une liste
```

Comme explique dans le commentaire du code, ce nombre permet de spécifier :

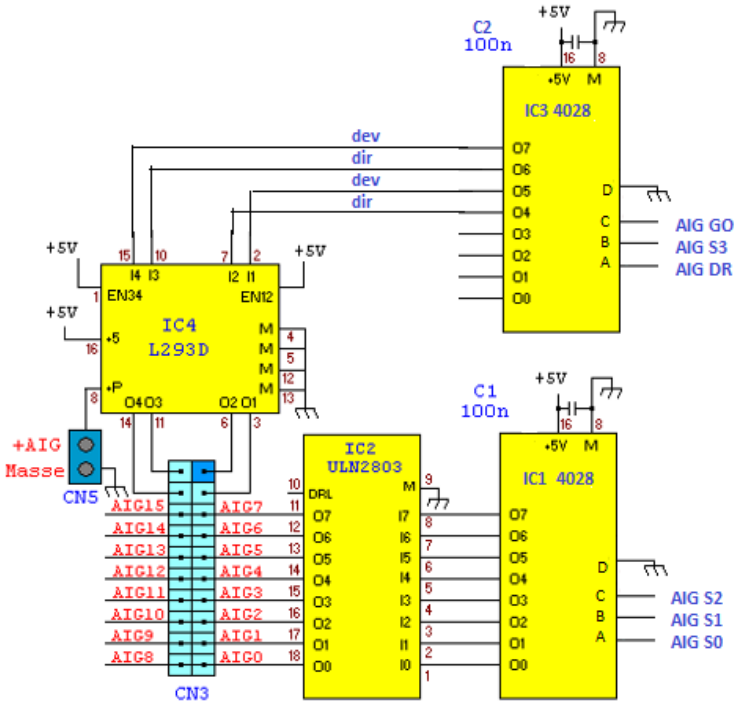
- La durée de l'impulsion de commande (125ms, 250ms, 500ms, 1s, 2s, 5s, 10s)
- L'inversion de la commande entre directe et dévié pour ceux qui branches les aiguillages à l'envers !
- La position souhaitée à l'initialisation

Attention, une impulsion trop longue peut détruire les bobines des aiguillages !

FreeDCC commande les aiguillages un par un. Ceci à l'énorme avantage de ne pas demander un courant trop important et de pouvoir matricer la commande des aiguillages afin de réduire l'électronique. Cette commande est bien entendu plus lente qu'une commande parallèle.

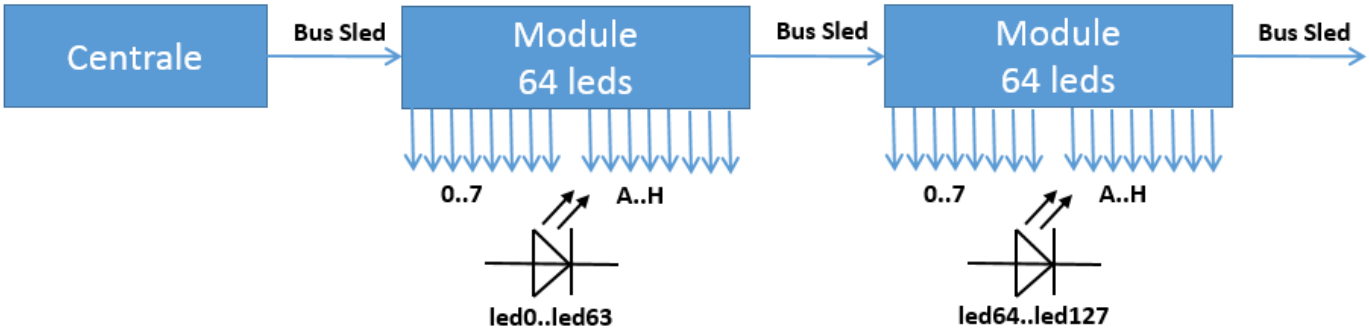
A la mise sous tension de l'Arduino, le programme initialise l'électronique afin de ne pas commander les bobines, puis attend 10s avant d'initialiser les aiguillages. Ceci afin de laisser le temps aux personnes qui ont un interrupteur sur l'alimentation de puissance de le fermer. Pour plus d'information, se reporter au chapitre sur les alims.

Il est également possible d'utiliser une solution sans relais comme avec la centrale A. Par exemple :



4.13. Les LEDs

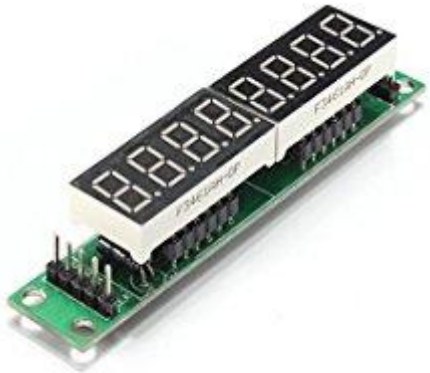
Le bus Sled permet de raccorder jusqu'à 4 modules de 64 leds. Chaque module utilise un driver de LED MAX7219 ou 7221 capable de gérer 64 leds.



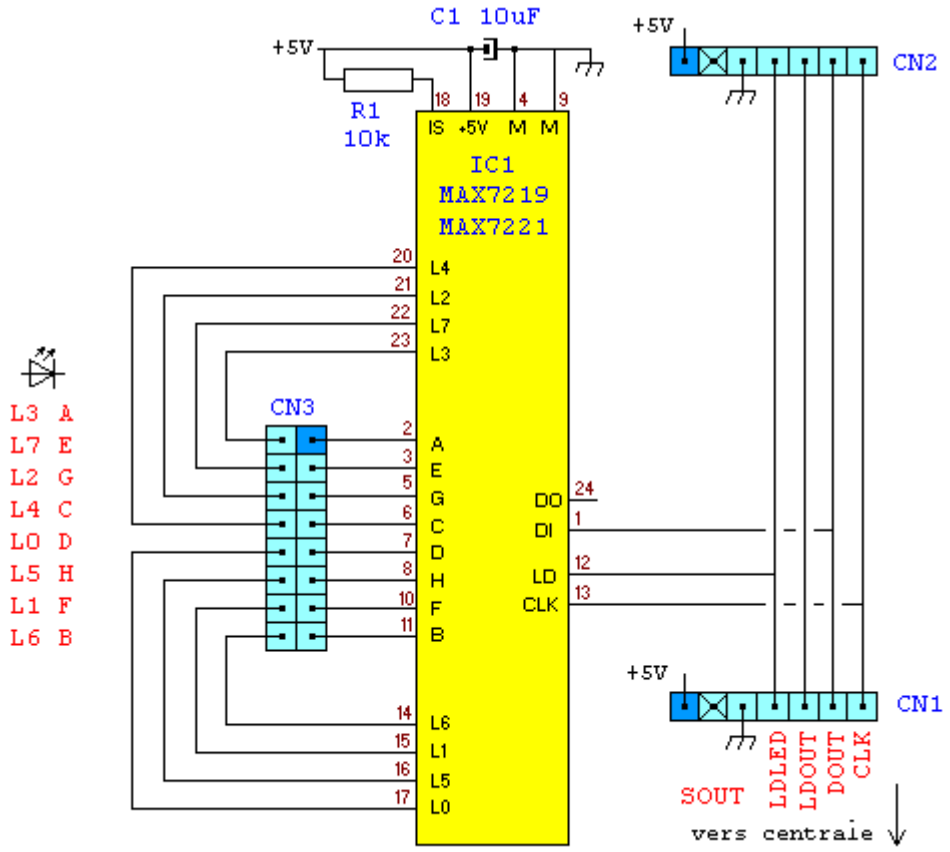
Un bus série de type SPI (ressemblant à Sout) permet de communiquer avec les 7219. Chaque échange commence par la mise à l'état bas du signal d'activation LDLED (Load Led = chargement des LED) puis d'un décalage de 4 mots de 16 bits (un pour chaque MAX7219), puis se termine par une validation de l'échange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

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

Vous pouvez réaliser des modules avec le circuit MAX7219 (fonctionne aussi avec le 7221) comme indiqué dans fdcc2010 ou alors acheter directement de tels modules sur Amazon par exemple pour 5€. 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 module 64 LEDs (doc de fdcc2010)



Ce module permet de contrôler matriciellement 64 LEDs.
Il suffit de les brancher sans aucune résistance à partir du connecteur 3 en mettant

- Les anodes (+) des LEDs sur les pattes L0 à L7
- Les cathodes (-) sur les pattes A-H

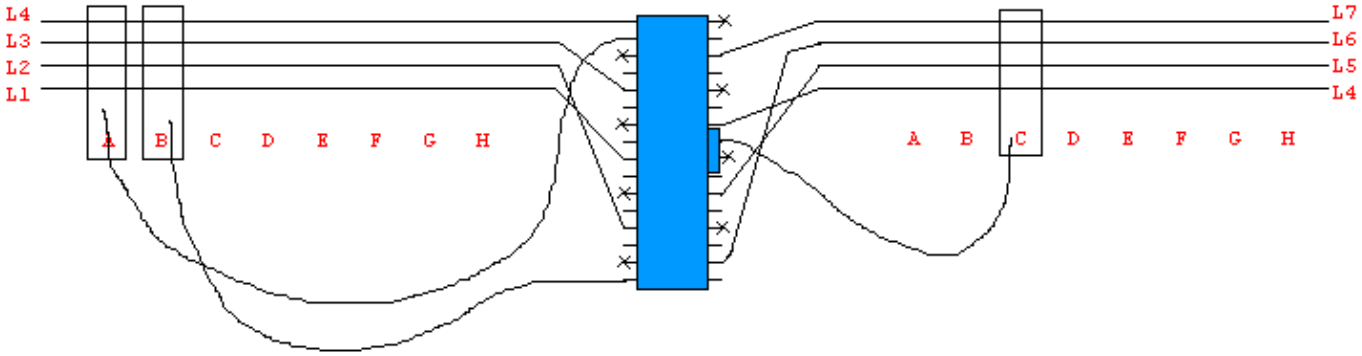
Le numéro de la led pour Free-DCC se calcule en additionnant le numéro de son anode et de sa cathode avec les poids suivants :

L0 = 0	L1 = 1	L2 = 2	L3 = 3	L4 = 4	L5 = 5	L6 = 6	L7 = 7
A = 0	B = 8	C = 16	D = 24	E = 32	F = 40	G = 48	H = 56

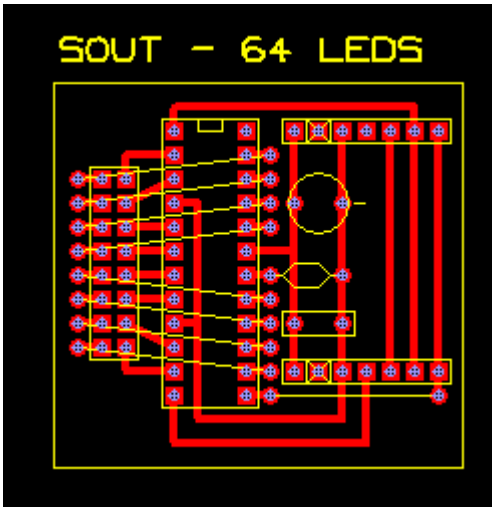
Par exemple une LED connectée entre E et L3 à comme numéro 32+3=35

Le schéma électrique se compose principalement du driver de LED IC1 qui peut être un MAX7219 ou 7221. Le circuit est alimenté en 5V par le bus SOUT et cette tension est filtrée par le condensateur C1 beaucoup plus gros que d'habitude car le driver perturbe beaucoup l'alimentation lorsqu'il passe d'un groupe de 8 LEDs à un autre. Le module s'interface à la centrale par le bus SOUT via le connecteur CN1. Le module suivant qui doit être le module aiguillage s'il est utilisé ou alors un module de sortie s'interface sur le connecteur CN2. Le connecteur CN3 permet quant à lui d'y connecter les LEDs. Il reçoit les commandes et données sur son entrée DI au rythme de l'horloge CLK, puis le transfert est validé avec la broche LD. La résistance R1 permet de fixer le courant maximal dans les diodes. Par la suite la centrale à la possibilité de diminuer ce courant. Ceci peut être utile pour voir la signalisation en plein jour et éviter qu'elle ne se transforme en lampadaire dans l'obscurité. Le courant traversant toutes les LED sera identique, si vous souhaiter le diminuer pour une LED particulière, il faudra lui ajouter une résistance.

Bien entendu, vous ne pouvez pas relier les 64 LEDs une à une sur le pauvre connecteur CN3 à 16 contacts. Pour ce faire vous pouvez réaliser une carte de répartition et brancher les leds une à une ou alors les souder directement sur des fils. Une solution simple est de sertir par le milieu un câble nappe à 16 conducteur dans un connecteur HE10 2*8 femelle, puis de désolidariser les fils et enfin créer 2 bus de 4 leds où vous brancherez les anodes (+) des LEDs avec pour les anodes 8 sélections possible par bus. Ce qui permet par exemple de connecter 16 signaux à 4 leds. La figure suivante explicite cette solution avec uniquement 3 signaux pour la lisibilité

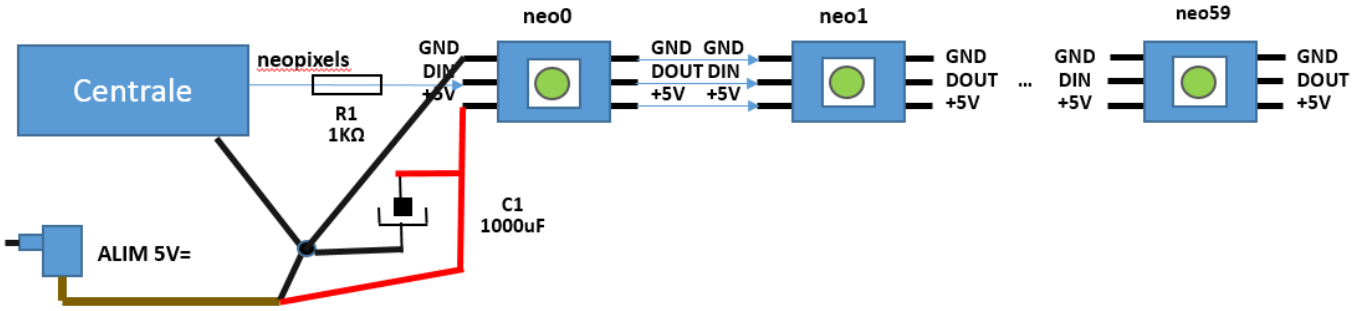


Coté réalisation vous pouvez utiliser de la plaque pastillée percée ou le typon suivant :



4.14. 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 …

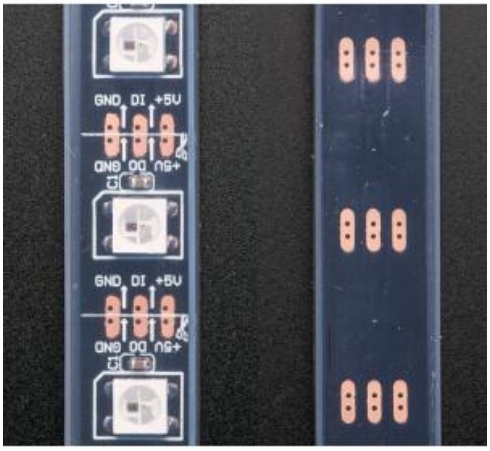


Chaque composante (rouge, vert ou bleue) consomme 20mA lorsqu'elle est à 100%. La consommation maximale est donc de 60mA par led. A ce niveau-là, ça éclaire vraiment beaucoup ! Vu la consommation, il est conseillé d'utiliser une alimentation 5V indépendante. Sa puissance (donc son ampérage sera choisi 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 où 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.

Si vous n'utilisez pas les néopixels, vous pouvez utiliser la fonction CLI à la place. La patte clignotera alors à la fréquence de 1Hz comme c'était le cas sur fdcc2010. Pour indiquer l'arrêt d'urgence, le clignotement est différent.



4.15. Les effets lumineux spéciaux FX

La centrale permet 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 ! La bonne nouvelle est que vous pouvez faire les effets que vous voulez ! La mauvaise est qu'il va falloir les programmer en C ! Pour vous aider, je donne quelques exemples ;-) 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 une sortie 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 le PC par les variables des effets spéciaux s0-63.

Dans le code, les fonctions d'exemple des effets spéciaux se trouvent après le commentaire :

```
// <<<<<<<<<<< Debut des fonctions utilisateurs des effets speciaux FX
```

Vous pouvez les utiliser, les adapter ou vous en inspirer pour vos nouvelles fonctions.

Par exemple, voici le code d'une fonction qui simule un feu de cheminé sur le néopixel 0 :

```
// Fonction simule en permanence un feu de cheminee sur le neopixel 0
void fx_neo0_cheminee(void)
{
    byte r = random(50,80);
    byte v = random(0,30);

    neo_rvb[0][NEO_R]=r;
    neo_rvb[0][NEO_V]=v;
    neo_rvb[0][NEO_B]=0;
}
```

Cette fonction est appelée toute les 125ms (8x par seconde) si vous mettez son nom dans la fonction suivante :

```
// Cette fonction est appelee toute les 125ms. Appellez vos fonctions ici
void fx_maj(void)
{
    fx_cpt++;

    fx_neo0_cheminee();
    //fx_neo1_tv();
    //fx_neo2_scintillement();
    //fx_neo3_neonkc();
    //fx_neo4_arc();
    //fx_led0_arc();
    //fx_led1_neonkc();
    //fx_pwm48_scintillement();
    //fx_pwm49_arc();
}
```

Pour simuler une flamme, j'utilise un rouge variable que j'obtiens avec la fonction random() de l'Arduino. Le néopixel supporte des niveaux entre 0 et 255. Avec 255, c'était vraiment trop brillant ! J'obtiens ici une valeur comprise entre 50 et 80. Comme une flamme n'est pas que rouge mais aussi jaune, j'ajoute un peu de vert pour faire du jaune ! Il n'y a que 3 couleurs de base et pour faire du jaune, il faut mélanger du rouge et du vert ! Je ne mets pas trop de vert pour rester dans le rouge-jaune. Une flamme verte ne serait pas très réaliste ! Ensuite, il me suffit d'affecter les valeurs calculées au néopixel 0.

Notez, que ce feu sera permanent sur le néopixel 0. Vous ne pourrez pas utiliser ce néopixel avec le logiciel PC car toute demande de modification du PC serait écrasée au mieux 125ms plus tard. Il est possible de rendre ce feu activable par une variable des effets spéciaux (s0-63). Pour cela, rajoutez la ligne suivante en première ligne de la fonction :

```
if(is_fx(10)==0) return;
```

Si l'effet spécial 10 n'est pas actif alors la fonction retourne sans exécuter le reste du code donc sans mettre à jour le néopixel. Vous pouvez utiliser cette variable par script, un bouton sur un TCO ou la tester avec le menu E/S > FX.

A vous de réaliser maintenant de superbes effets spéciaux. Si vous n'y arrivez pas, je peux vous aider ;-) Un ajout pour s'exercer serait le flash d'un radar à des temps variables ! Ne transformez pas non plus votre réseau en foire !

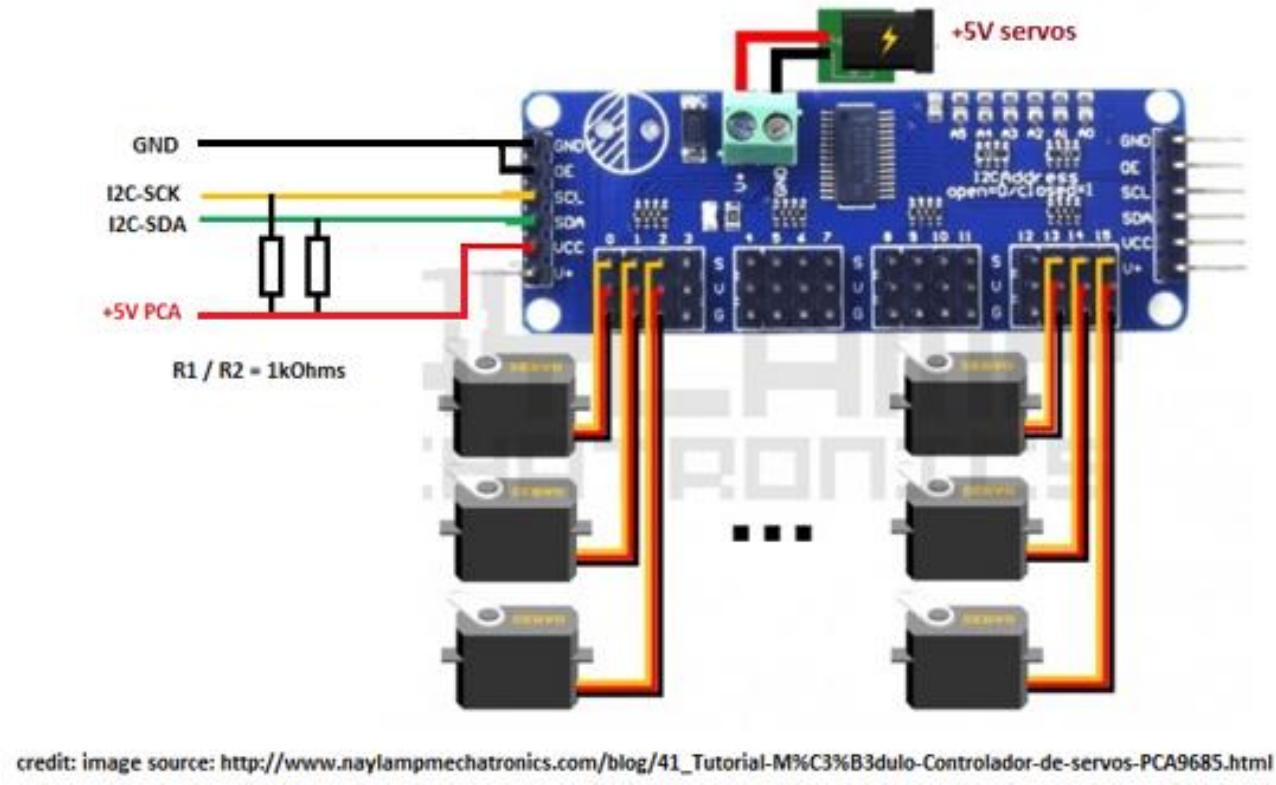
4.16. Le bus I2C

Le bus I2C permet de relier tous les circuits I2C existants à la centrale. 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. Afin d'étendre les capacités du système, il est possible de remplacer 2 sorties PWM par les 2 signaux du bus I2C: SDA et SCK. Vous noterez que ces 2 pattes ne sont pas connectées sur le périphérique I2C de l'AT328P. Qu'à cela ne tienne, il est facile d'émuler le protocole du bus I2C sur E/S. Ne pas oublier les 2 résistances de pull-up (entre 1 et 10kΩ).

Actuellement, les centrales supportent 3 PCA8596. Chaque circuit dispose de 16 sorties pouvant être utilisées individuellement en mode PWM ou Servo. Donc 48 sorties ! Free-DCC sera bientôt capable de gérer les aiguillages à servomoteurs.

Exemple de module 16 servos à moins de 10€ et servos à 2€ pièce sur Amazon :

Connexion à fdcc :



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

Ne pas utiliser le 5V de l'Arduino pour alimenter les servos.
Ne pas oublier les résistances de pull-up du bus I2C.
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)

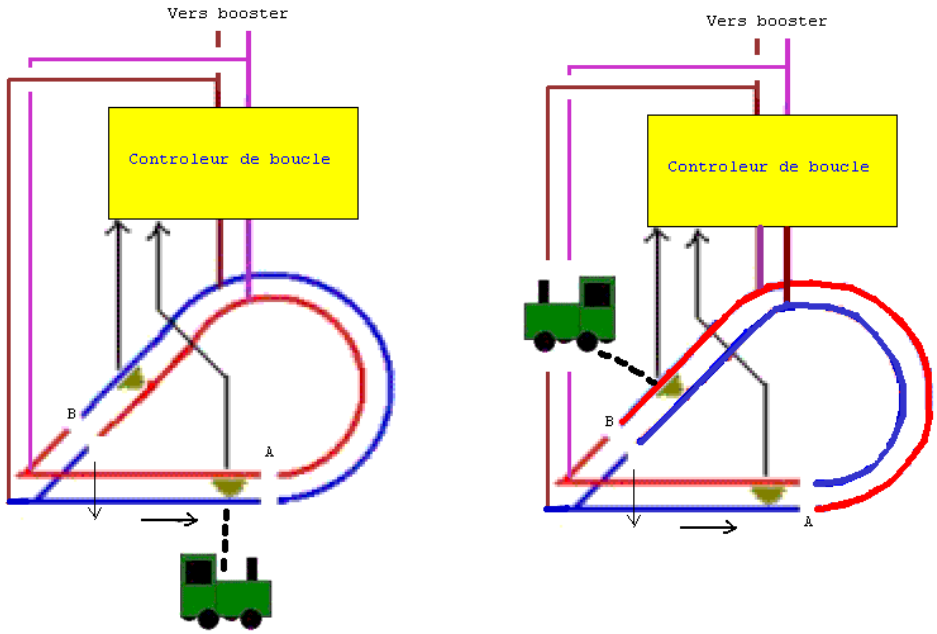
4.17. Les boucles de retournement

Il existe plusieurs manières de gérer les boucles de retournement :

- La plus simple étant de connecter en // de l'aiguillage d'entrée un relais bistable câblé en inverseur qui alimentera la boucle en fonction de la position de l'aiguillage. Il est également possible d'utiliser les inverseurs de positions de l'aiguillage s'il en est muni, mais vérifier qu'ils supportent le courant.
  Bien entendu il faudra piloter l'aiguillage pour que cela fonctionne
- En détectant un court-circuit lorsque le train franchira la frontière électrique. Je ne conseille pas cette m méthode
- En détectant le train à l'endroit où il faut inverser le courant et l'inverser. Cela peut se faire simplement en utilisant un relais bistable.

Les boucles de retournement (doc de fdcc2010)

Les boucles de retournement présent sur de nombreux réseaux souvent en coulisse contribuent au réalisme des circulations en retournant les rames. En 3 rails, un tel dispositif ne pose aucun problème car les plots centraux sont à relier à une sortie du booster et les rails à l'autre. Par contre, il n'en va pas de même en 2 rails. En effet à l'aide de la figure suivante en fonction de la polarité de la boucle, on provoque un court-circuit soit au point A, soit au point B. La solution est d'appliquer une polarité à la boucle compatible avec le passage d'une locomotive au point A lorsqu'elle actionne la pédale A (première figure) et de l'inverser lorsqu'elle actionne la pédale B afin d'autoriser son passage en B.



Bien entendu avec cette solution la boucle ne peut être parcourut que dans un sens. Pour un parcourt dans les 2 sens, il suffit d'utiliser 2 pédales positionnées de chaque côté des points de basculement. Si vous jugez les pédales pas assez fiables, vous pouvez toujours les remplacer par des capteurs de courant.

Le contrôleur de boucle se compose d'un simple relais 2RT qui permet d'inverser la polarité de la boucle. Ce relais est soit un modèle bistable qui change de position et la maintient suivant les impulsions venant des pédales ou un modèle standard qu'il faut associer à une mémoire. La mémorisation des modules S88 précédent peut être utilisée à cette fin.

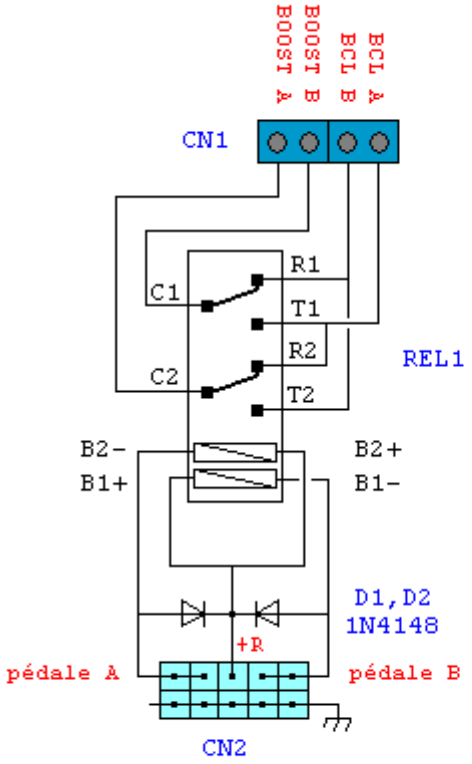
Le schéma suivant présente ces 2 cas.

Le premier utilise un relais bistable qui bascule les contacts :
- C et T lorsque la bobine 1 est activée par la ou les pédales B
- C et R lorsque la bobine 2 est activée par la ou les pédales A

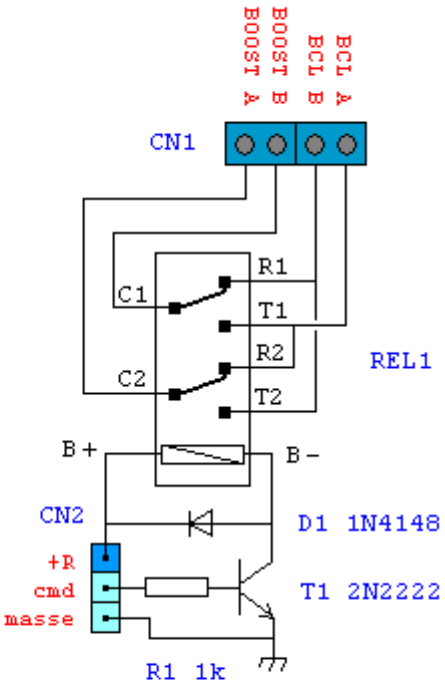
Les diodes éliminent la surtension qui apparaît lorsque l'on coupe l'alimentation d'une bobine.

Le second schéma met en œuvre un relais standard commandé par la broche cmd du connecteur CN2. Le transistor amplifie le signale de commande afin de piloter le relais

Dans les 2 cas la tension +R alimente les bobines des relais. Elle est à raccorder au +5V ou +15V en fonction du relais utilisé. Les contacts des relais devront supporter 3A, mais si vous n'en trouvez pas des modèle 1A peuvent faire l'affaire.

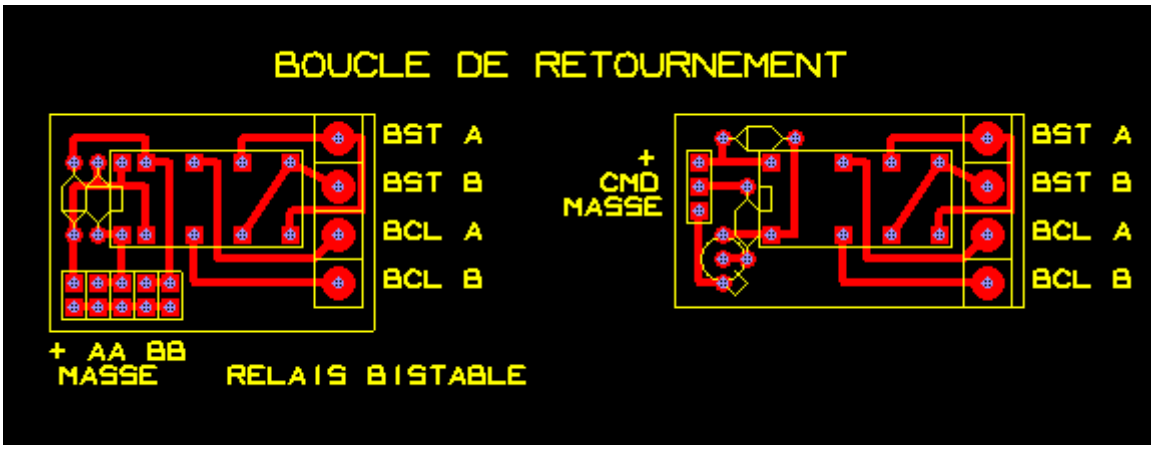


Relais bistable



Relais standard

La réalisation peut se faire sans plaque, sur une plaque à trou ou alors en utilisant un circuit imprimé en s'inspirant du typon suivant. Vu la différence de brochage entre les différents relais, il faudra peut-être modifier les typons.



Notez qu'il existe d'autres méthodes pour gérer les boucles de retournement. Pour ma part j'utilise maintenant la solution décrite dans la doc de Free-dcc-2008 afin de remplacer les pédales de voie ou ILS par des courtes sections de voies de détection.

4.18. Les servomoteurs

48 servos sont actuellement supportés via le bus I2C. Voir le chapitre sur le bus I2C. Vous pouvez les configurer en éditant le code des centrales afin les activer, régler la position de départ et la vitesse de rotation.

4.19. Les moteurs pas à pas

Non gérés actuellement

4.20. Les plaques tournantes

Non gérés actuellement

4.21. Prix

Pour donner une idée des prix, voici les prix du commerce :

Elément	Prix du commerce
centrale DCC (avec alim / booster /2 commandes fixes)	500€
Souris	50€
décodeur 8 sorties (4 aiguillages /ou 4 feux 2 leds /ou 2 feux 4leds)	50€
détecteur 8 cantons	50€
module de rétro-signalisation a 16 entrées sur bus S88	50€
décodeur de locomotive	30€
boucle de retournement	50€

Ce qui donne par exemple pour un réseau composé de

- 1 centrale = 500€ (incluant 2 souris, les alims et le booster)
- 16 cantons (2x détecteur 8 cantons + 1x rétro-signalisation de 16 entrées) = 2x50+50=150€
- 16 aiguillages (4x décodeurs 8 sorties (4 aiguillages)) = 4x50=200€
- 8 feux (8x4 leds) (4x décodeurs 8 sorties (2x4leds)) = 4x50=200€
- 1 boucle = 1x50=50
- 4 souris (donc 2 supplémentaires) = 2x50=100€
- 4 sorties (1x décodeur à 8 sorties) = 50€

Total = 1250€ pour la partie fixe

- 8 locomotives (8x décodeurs) = 8x30=240€

Total = 1590 €

Au vu de ce prix (qui comprend uniquement l'électronique), on peut se demander si le DCC vaut toujours la peine !

La norme DCC très bien adaptée aux locomotives, mais l'est beaucoup moins pour les accessoires. Car avec 50€ pour 8 sorties (permettant de piloter 4 aiguillages ou 8 leds de feux), la facture monte vite…

En comparaison, voici les prix avec free-dcc

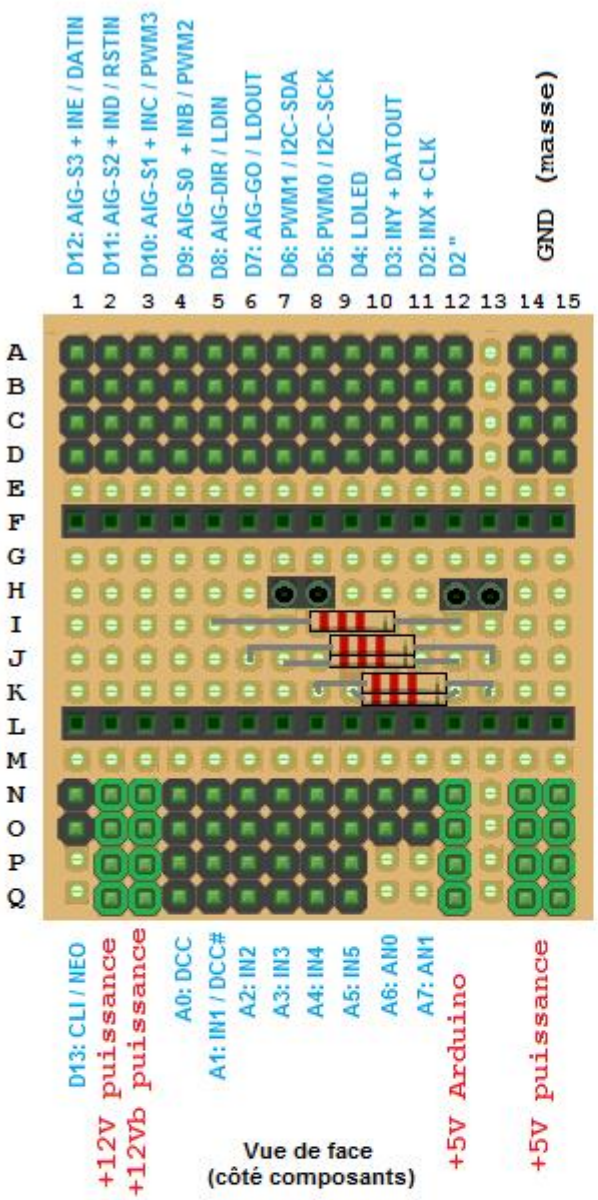
Elément	Prix
Centrale DCC (Arduino Nano) Alim Booster/Aig 3A (12V ou 13.5V ou 15V) L293D (pont en H du booster) Souris Alim 5V 1A (pour extension LEDs, Neopixels) centrale DCC (avec alim / booster /2 commandes fixes)	5€ 10€ 10€ 5€ 5€ 40€
Souris	5€
décodeur 8 sorties + relais (4 aiguillages /ou 4 feux 2 leds /ou 2 feux 4leds)	10€ (mais moins cher d'utiliser les modules led et aig)
détecteur 16 cantons	15€
module de rétro-signalisation a 16 entrées sur bus S88	5€
module 64leds	5€
module 48 aiguillages	15€
Module 8 relais (pour 10aigs centrale type B)	5€
décodeur de locomotive	30€ (du commerce)
boucle de retournement	5€

Pour le même réseau, la facture passe de 1250€ à 100€ pour la partie fixe. Avec les décodeurs, le prix de revient s'élève à 100€ + 240€ = 340€. Cette prise de conscience démontre qu'il est financièrement préférable de réaliser soi-même le système La division est encore plus spectaculaire pour les grands réseaux. Mais bien-entendu, il faudra un peu de temps pour réaliser le système

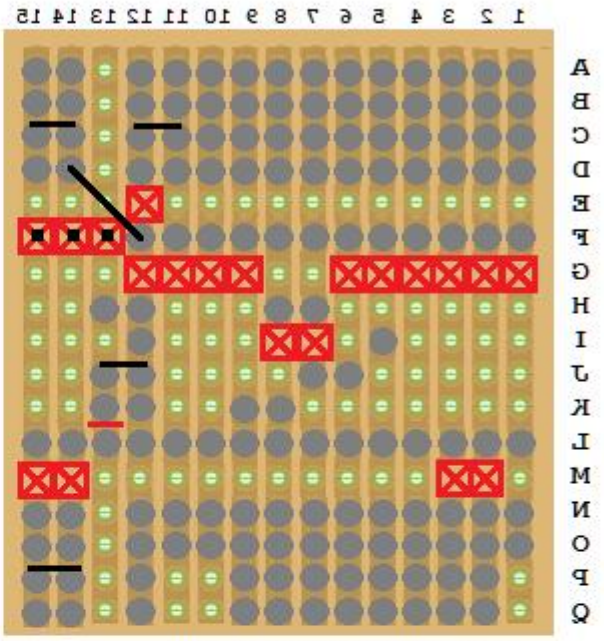
4.22. Réalisation

Niveau réalisation, il est possible de relier les modules par des fils ou alors de réaliser une carte électronique incluant les différents modules ou permettant de les enficher. La carte électronique peut être réalisée à partir d'une plaque à trous (Veroboard) ou en réalisant un vrai circuit imprimé. Je n'ai pas réalisé une carte à circuit imprimé pour le moment. Je conseille d'utiliser des connecteurs afin de pouvoir connecter/déconnecter facilement les différents modules. On mettra ensuite la réalisation dans un joli boitier pour faire pro-;-) Envoyez-moi les photos de vos réalisations !

Actuellement, j'utilise une petite carte réalisée avec une plaque à trous (avec des bandes plutôt que des pastilles). Cette petite carte comporte un connecteur pour y brancher un Arduino Nano ainsi que de multiples connecteurs pour y raccorder les autres modules. Elle est très pratique, surtout en phase de développement. L'image suivante, montre comment la réaliser. N'oubliez pas de couper les pistes comme indiqué. Quand la coupure est sur tout un trou, il est facile d'enlever la piste en tournant avec la main un foret pour le métal. Il faudra juste réaliser une coupure plus précise entre K13 et L13. Cette carte comporte juste des connecteurs (femelles pour y brancher l'Arduino nano et les résistances I2C et males pour connecter la carte aux autres modules via des fils femelle-femelle pour Arduino. Ne pas oublier les 5 « staps » (fils) afin de relier certaines pistes. Les résistances sont toutes des 2200Ω. Vu la place disponible certaines sont montées les unes sur les autres. Des connecteurs permettent de rajouter ou d'enlever les résistances I2C. Si vous êtes sûre d'utiliser l'I2C ou pas, les connecteurs des résistances ne sont pas obligatoires. Au niveau des alimentations, le +5V Arduino, provient de la carte Arduino, il peut servir à alimenter quelques circuits sous 5V mais sans jamais dépasser 400mA. Les alimentations de puissances (+5V puissance, 12V puissance et 12Vb puissances) ne sont pas reliées à l'Arduino mais les connecteurs permettent de les répartir facilement. Il est impératif de brancher les masses de toutes les alimentations ensembles !

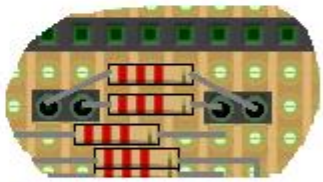


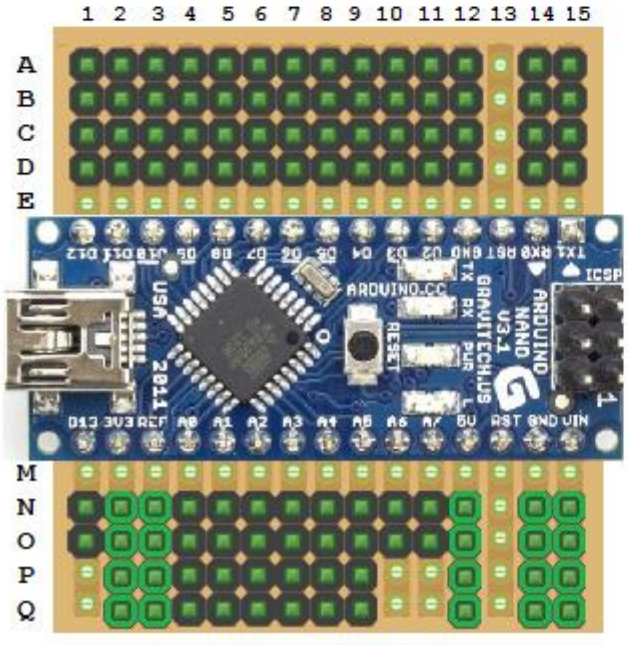
Vue de face
(côté composants)



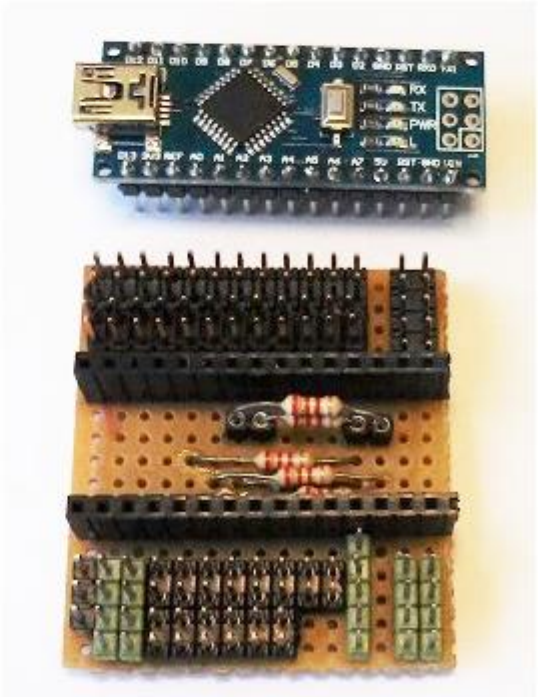
Vue de dessous (côté pistes)

Ajout des 2
résistances I2C

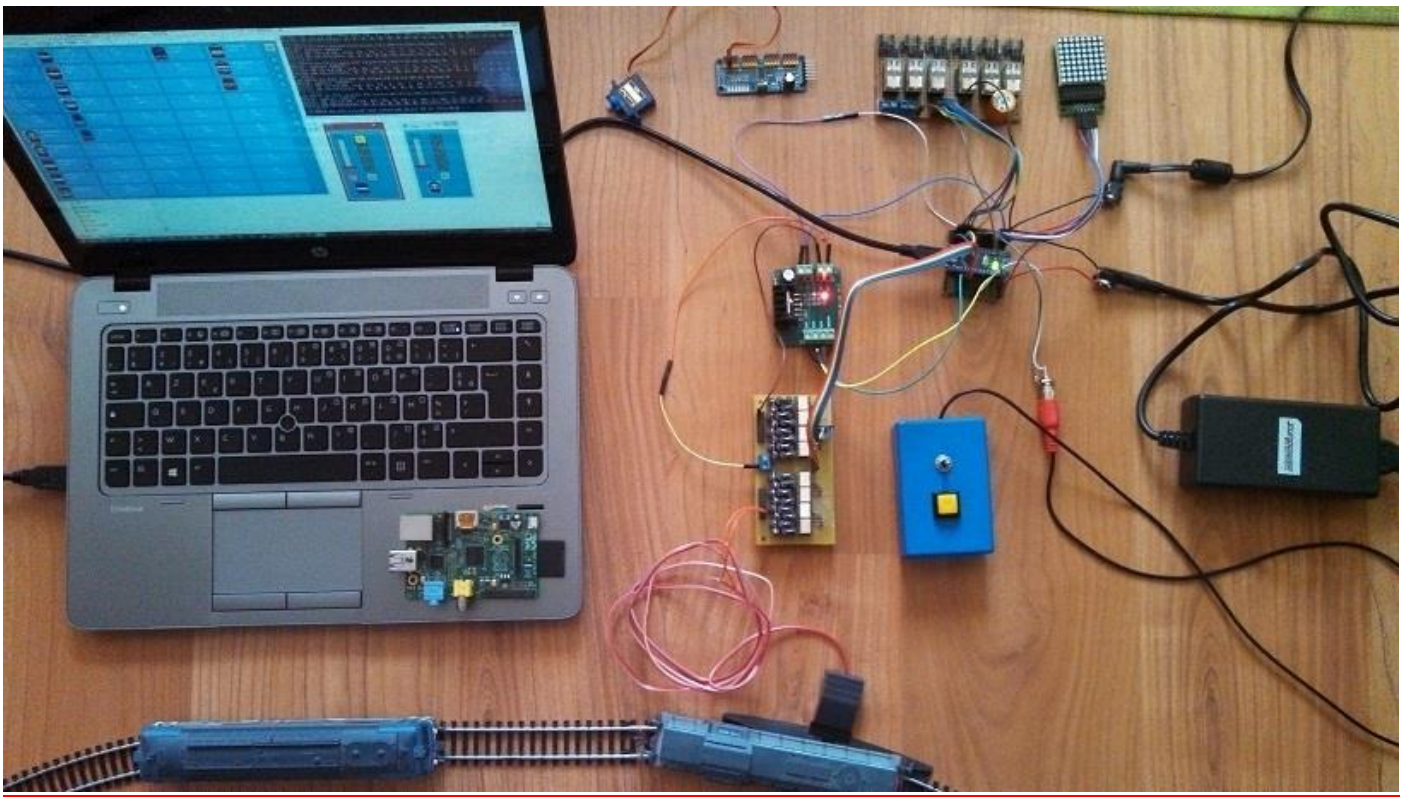




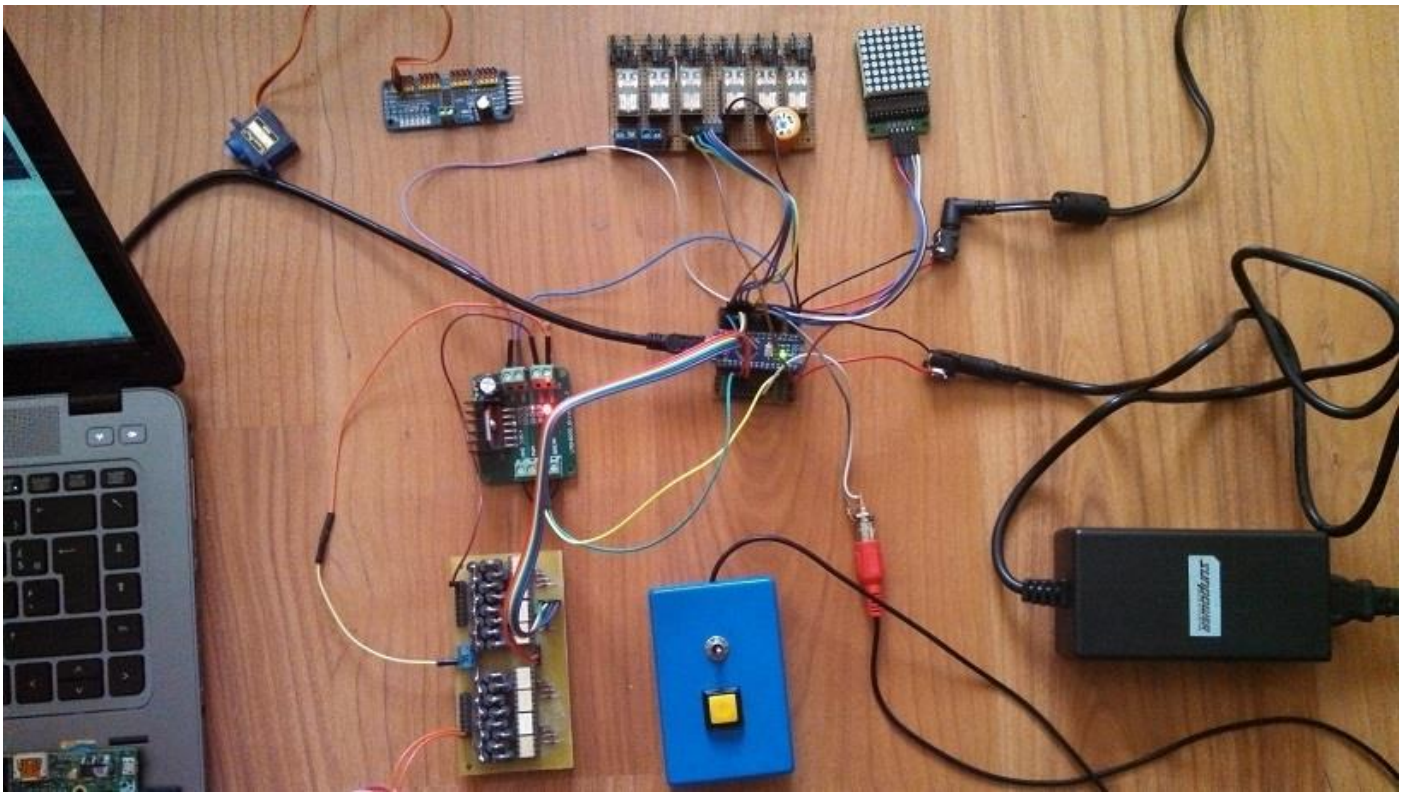
Avec la carte Arduino Nano



Quelques photos de cette petite carte en action :



Au centre, la carte Arduino Nano enfichée sur une carte à connecteurs reliée au booster (carte verte avec la led rouge), lui-même relié sur une carte de détection à 16 voies dont une est branchée sur le circuit d'essai. En haut, on trouve une carte pour servomoteurs (non branchée), une carte à relais pour 12 aiguillages à bobines, une carte pour 64 leds. La boite bleue est une souris. La carte Arduino est reliée au PC ou tourne le logiciel fdcc_pc pour la sélection des locomotives, les TCO et l'automatisation. La grosse alimentation 5V que l'on voit sur la photo alimente les relais et les leds. Une autre de 12V hors champ alimente les voies. L'Arduino est lui alimenté par le PC. La carte que l'on trouve sur le PC est une Raspberry PI à 25€ qui peut remplacer le PC car le logiciel PC en Python tourne sur tous les OS dont Windows, Linux... Si l'on souhaite juste conduire les locomotives, on peut se passer du PC.
Zoom sur l'électronique :

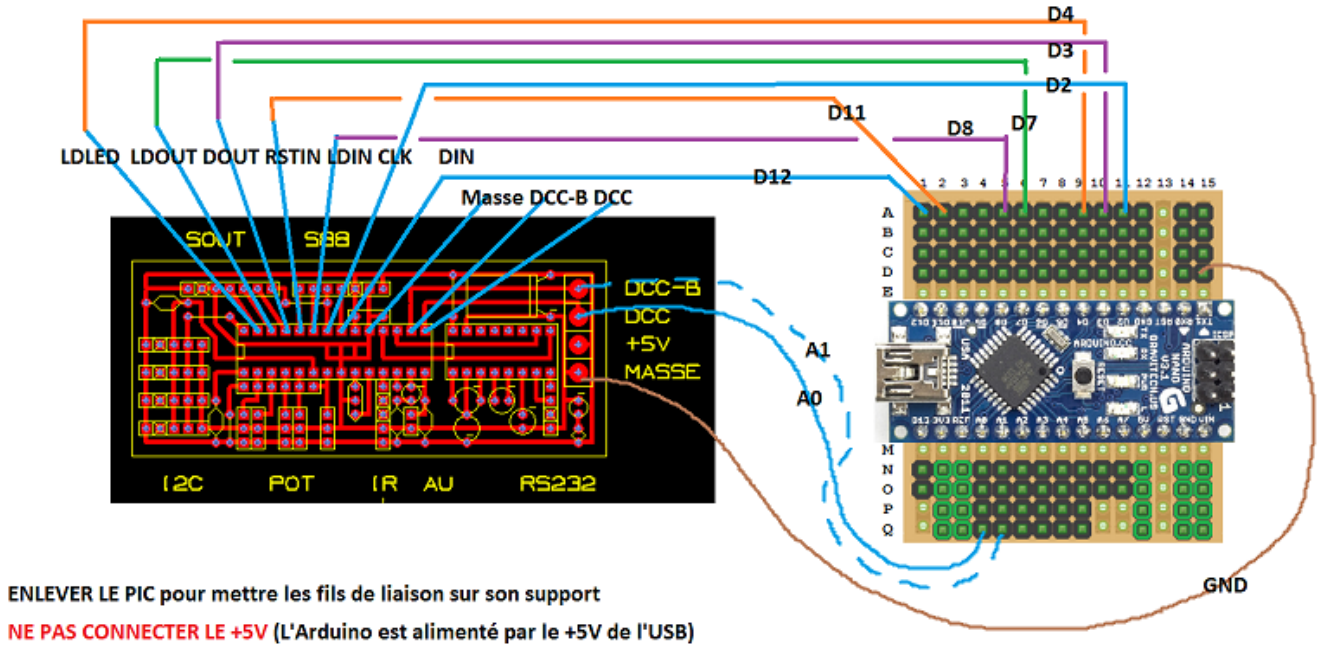


4.23. Passage de free-DCC 2010 à free-DCC 2017

Pour les utilisateurs de free-DCC2010, vous pouvez passer à free-DCC2017 en enlevant le PIC et en connectant des fils entre son support et la nouvelle carte Arduino. Vous pouvez aussi abandonner la carte de free-DCC 2010 et connecter directement les modules et booster à la nouvelle carte Arduino.

Cet « upgrade » à moins de 10€ (5€ pour l'Arduino nano), vous permet de bénéficier de toutes les nouveautés de free-DCC 2017 tout en conservant les modules des aiguillages, leds, sorties, entrées S88, détection, boucle de retournement, booster de free-DCC 2010. Les souris de free-DCC 2010 (Potentiomètres, I2C, consoles) ne sont plus utilisables. Il faudra réaliser les nouvelles souris (moins de 5euros pièce).

Le schéma ci-dessous montre comment faire :



ENLEVER LE PIC pour mettre les fils de liaison sur son support

NE PAS CONNECTER LE +5V (L'Arduino est alimenté par le +5V de l'USB)

MASSE, DCC, DCC-B sont aussi accessibles sur le support du PIC (à la place des borniers)

DCC-B est uniquement nécessaire pour les boosters à 2 entrées

Vous pouvez aussi abandonner la carte 2010 et relier directement les modules et le booster à l'Arduino

Remplacement du PIC de free-DCC 2010 par l'Arduino de free-DCC 2017

Penser à configurer correctement le logiciel de la centrale :

- Centrale de type A
- Ordre des modules de sortie et d'aiguillage. Sur free-DCC 2010, le module à 32 aiguillages est le plus proche de la centrale. (Vous pourrez même rajouter 16 aiguillages car sur free-DCC 2017, les modules supportent 48 aiguillages). Viennent ensuite les modules de sorties.
- DCC-B si vous utilisez un booster à 2 entrées (mais ceci réduit le nombre des souris de 10 à 8 max)

Le chapitre suivant explique comment configurer la centrale.

Notez que free-DCC 2010 n'est plus supporté ;-)

5. Configuration des centrales et Utilisation en autonome

5.1. Configuration des centrales

Après avoir réalisé l'électronique, il est temps de passer au logiciel des centrales. Les utilisateurs auront juste à le configurer. Les plus téméraires pourront le modifier pour l'adapter si nécessaire à leurs besoins. Ce logiciel génère tous les signaux pour piloter l'électronique dont le fameux signal DCC. Il offre un protocole de communication pour permettre à un ordinateur de se connecter et contrôler le réseau. Il peut également fonctionner en autonome sans ordinateur et permet de conduire les locomotives avec des souris. Les sources du logiciel de l'Arduino se trouve dans l'archive dans le répertoire fdcc2017/arduino/fdcc_2017_centrale /fdcc_2017_centrale.ino. Il est valable pour les 2 types de centrales (il suffit de choisir entre centrale A ou B dans le fichier). Afin de ne pas complexifier le code, tout le logiciel tient dans un fichier (certes de plus de 3000lignes). Je vous propose d'ouvrir le logiciel avec un éditeur de texte simple (comme notepad, pas word !) pour le configurer. Une fois sauvé, vous pourrez l'ouvrir avec l'environnement Arduino, le compiler (avec le bouton qui ressemble à un V), et le « téléverser » dans votre Arduino aussi simplement que le programme de led clignotante de l'initiation à Arduino. Je considère que vous êtes familiers avec l'environnement de développement Arduino et que vous avez réussi à faire clignoter la LED comme conseillé au chapitre des Arduinos. Le téléchargement n'est à faire qu'une seule fois. Le programme est mis dans la mémoire flash qui conserve le programme même lorsque la carte n'est pas alimentée. Dès le téléchargement terminé, le programme doit s'exécuter. Vous devez alors voire la LED clignoter (si use_CLI=1).

La partie suivante (au début du programme) vous permet de configurer la centrale (en rouge, ce que vous pouvez modifier):

```
/*-------------------------------------------------------------------------
0. Configuration de la centrale
-------------------------------------------------------------------------*/

// Type de centrale
#define use_centraleA 0    // 1 = centrale de type A (0 sinon)
#define use_centraleB 1    // 1 = centrale de type B (0 sinon)

// Numero de la centrale (si plusieurs centrales sont utilisees)
// normalement seulement la 0 genere le signal DCC
#define num_centrale 0

// DCC # (utilise DCC#)
#define use_DCCB 0         //0=PORT_A1->IN_4 / 1=PORT_A1->DCC#

// CLI (utilise CLI)
#define use_CLI  0         //0=NEO / 1=CLI

// adresse des locos des souris a l'initialisation (1-50  0=aucune locomotive selectionee)
#define MOUSE_0_ADR 10
#define MOUSE_1_ADR 0
#define MOUSE_2_ADR 0
#define MOUSE_3_ADR 0
#define MOUSE_4_ADR 0
#define MOUSE_5_ADR 0
#define MOUSE_6_ADR 0
#define MOUSE_7_ADR 0
#define MOUSE_8_ADR 0
#define MOUSE_9_ADR 0

// locomotives analogiques
// La fonction add_alim_analog(adr, pwm_vit, pwm_dir, pwm_cran1, pwm_cran28)
// permet de remplacer une loco numérique par une alimentaion analogique
// Pour le système, l'alimentation analogique obeit aux ordres de la loco numérique associée !
// adr        = adresse de la loco numérique associée (1-50). Si adr==0 l'alimentation
analogique n'est pas utilisée.
// pwm_vit    = numéro de la broche pwm gérant la vitesse (0-51)
// pwm_dir    = numéro de la sortie pwm gérant le sens    (0-51). Il est dommage d'utiliser une
sortie pwm pour le sens mais ca simplifie !
// pwm_cran1  = valeur de la pwm (0-255) pour le cran numérique  1 (la locomotive doit démarer)
// pwm_cran28 = valeur de la pwm (0-255) pour le cran numérique 28 (la locomotive doit démarer)
// vous pouvez définir jusqu'a 5 alimentations analogiques
#define CONFIG_ALIM_ANALOG \
```

```
add_alim_analog(50, 48, 49, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \

// Protocol de communication centrale <--> PC
// 1=utilisation du protocol (0=non). Seul un protocol peut être utilise !
#define use_UU        1  // 1 = protocol UU
#define use_6050      0  // 1 = protocol 6050 Marklin
#define use_DBG_mouse 0  // 1 = debug mouse
//vitesse de communication parmi:
#define RS232_SPEED 115200  // 1200, 2400, 4800, 9600, 19200, 115200bps

// constantes des modules sur le bus Sout (ne pas modifier)
#define SOUT_OUT0_7   0
#define SOUT_OUT8_15  1
#define SOUT_OUT16_23 2
#define SOUT_OUT24_31 3
#define SOUT_OUT32_39 4
#define SOUT_OUT40_47 5
#define SOUT_OUT48_55 6
#define SOUT_OUT56_63 7
#define SOUT_AIG0_47  8
#define SOUT_AIG48_95 9
#define SOUT_END     10
//definir les modules OUT et AIG sur le bus Sout du plus proche au plus lointain de la centrale
//(sans importance pour centrale type B)
unsigned char sout_config[] = { SOUT_AIG0_47, SOUT_OUT0_7, SOUT_OUT8_15, SOUT_OUT16_23,
SOUT_OUT24_31, SOUT_AIG48_95,  SOUT_OUT32_39, SOUT_OUT40_47, SOUT_OUT48_55, SOUT_OUT56_63,
SOUT_END };

// AIG
    //constantes aig (ne pas modifier)
    #define AIG_SOUT_DIRECT 0  //centraleA: aiguillage sur bus Sout, centraleB: sorties directes
    #define AIG_SRV         1  //centraleA ou B: servos
    // impulsion (AIG_PULSE_X desactive l'aiguillage).
    #define AIG_PULSE_X      0 // aiguillage non utilise
    #define AIG_PULSE_125ms  1
    #define AIG_PULSE_250ms  2
    #define AIG_PULSE_500ms  3
    #define AIG_PULSE_1s     4
    #define AIG_PULSE_2s     5
    #define AIG_PULSE_5s     6
    #define AIG_PULSE_10s    7
    // inversion
    #define AIG_NORMAL       0
    #define AIG_INV          1
    // position demandee a l'init
    #define AIG_DIR          0
    #define AIG_DEV          1
    // macro
    #define config_aig(num_aig,module,num_aig_du_module,inv,pulse,cmd)  aig_data[num_aig] =
(((module&2)<<14) + (num_aig_du_module<<8) + (inv<<7) + (pulse<<4) + ((module&1)<<3) + cmd)

// definir pour chacun des aiguillages 0-95, le module, le numero de l'aiguillage sur ce module,
l'invesion, la duree de l'impulsion ou la non utilisation, la commande a l'init
// tout aiguillage non decalre ne sera pas utilise
#define AIG_SPECIFIC_INIT \
    config_aig(  0, AIG_SOUT_DIRECT,  0, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  1, AIG_SOUT_DIRECT,  1, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  2, AIG_SOUT_DIRECT,  2, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  3, AIG_SOUT_DIRECT,  3, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  4, AIG_SOUT_DIRECT,  4, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  5, AIG_SOUT_DIRECT,  5, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  6, AIG_SOUT_DIRECT,  6, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  7, AIG_SOUT_DIRECT,  7, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  8, AIG_SOUT_DIRECT,  8, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  9, AIG_SOUT_DIRECT,  9, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 10, AIG_SOUT_DIRECT, 10, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 11, AIG_SOUT_DIRECT, 11, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 12, AIG_SOUT_DIRECT, 12, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 13, AIG_SOUT_DIRECT, 13, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 14, AIG_SOUT_DIRECT, 14, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 15, AIG_SOUT_DIRECT, 15, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
```

```
// Cas particulier des servos:
// - la duree de l'impulsion n'est pas valable pour les servos
//   (mais AIG_PULSE_X desactive quand meme l'aiguillage)
// - Utiliser io_srv_vit (pour la vitesse de deplacement des lames)
// - io_srv_pos0 pour la position directe
// - io_srv_pos1 pour la position deviee
// - io_srv_cmd  pour la position a l'init avant l'init des aiguillages
//   (normalmeent, on mettra la meme que pos0)
// - inv n'a pas d'impact

// inversion la commande des aiguillages directs pour la centrale B
// 1 = pour les modules de relais qui collent à l'etat bas au lieu de l'etat haut
#define use_inv_aig_direct_cmd 0

// I2C
#define use_i2c 0  // 0=PWM48&49 / 1=I2C

// PCA9685
// Les 3 PCA9685 sont maintenant automatiquement detectes
// - pca9685_0 adr0(A5=0 A4=0 A3=0 A2=0 A1=0 A0=0)
// - pca9685_1 adr1(A5=0 A4=0 A3=0 A2=0 A1=0 A0=1)
// - pca9685_2 adr2(A5=0 A4=0 A3=0 A2=0 A1=1 A0=0)

#define SRV 0
#define PWM 1
// choisir entre mode SRV ou PWM  (pca9685_0:0-15, pca9685_1:16-31, pca9685_2:32-47)
// numéro de la pwm ou du servo           0    1    2    3    4    5    6    7     ...
const byte pca9685_cfg[48]  PROGMEM = { SRV, SRV, SRV, SRV, SRV, SRV, PWM, PWM,    ...
// vitesse des servomoteurs (1 =env 1deg/sec) choisir entre 1 (tres lent) et 255 (tres rapide)
// position des servomoteurs a l'initialisation (le PC va surement la changer !)
//   ( multipliez par 10us -> 150=1500us=1.5ms=neutre)
// pos0 et pos1 si le servo est utilise par les aiguillages
// io_srv_aig a 1 indique que le servo est utilise pour un aiguillage
//   (Il ne repondra pas a un ordre de mise a jour srv du PC)
// io_srv_aig a 0 indique que le servo n'est pas utilise pour un aiguillage
//   (Il ne repondra a un ordre de mise a jour d'aiguillage
// numéro du servo                        0    1    2    3    4    5    6    7       ...
const PROGMEM byte io_srv_vit[48]  = {   1,   2,   3,   4,   5,  10,  20,  50,      ...
            byte io_srv_cmd[48]  = { 150, 150, 150, 150, 150, 150, 150, 150,      ...
const PROGMEM byte io_srv_pos0[48] = { 100, 100, 100, 100, 100, 100, 100, 100,      ...
const PROGMEM byte io_srv_pos1[48] = { 200, 200, 200, 200, 200, 200, 200, 200,      ...
const PROGMEM byte io_srv_aig[48]  = {   0,   0,   0,   0,   0,   0,   0,   0,      ...
```

Les lignes suivantes permettent de choisir la centrale (ici centrale B)

```
// Type de centrale
#define use_centraleA 0   // 1 = centrale de type A (0 sinon)
#define use_centraleB 1   // 1 = centrale de type B (0 sinon)
```

La configuration par défaut est de ne pas générer le signal DCC complémenté sur la patte A5. Si vous en avez besoin, vous pouvez remplacer le 0 en 1. Bien entendu vous ne pourrez plus utiliser les entrées MIN 10-14.

```
// DCC # (utilise DCC#)
#define use_DCCB 0                //0=PINC / 1=DCC#
```

Pour le signal DCC non inversé, il n'y a rein besoin de configurer car il est toujours présent sur A0.

Pour un premier essai, je vous conseille de laisser l'option suivante :

```
// CLI (utilise CLI)
#define use_CLI  1                 //0=NEO / 1=CLI
```

Cette ligne permet de faire clignoter la LED verte de la carte connectée à la patte 13, vous indiquant que le programme fonctionne. En cas d'arrêt d'urgence, la led clignote de façon non symétrique. Il est possible de mettre cette option à 0 pour y connecter des néopixels. La led de la carte clignotera alors brièvement à toutes les mises à jour des néopixels.

Les configurations suivantes permettent d'indiquer pour chacune des 10 souris (mouse en Anglais), la locomotive sélectionnée par défaut par cette souris au démarrage. Si 0, aucune locomotive n'est sélectionnée. Il est bien entendu possible de les sélectionner à postériori en utilisant la souris comme indiqué au chapitre souris.

```
// adresse des locos des souris a l'initialisation (1-50  0=aucune locomotive selectionee)
#define MOUSE_0_ADR 10
#define MOUSE_1_ADR 0
#define MOUSE_2_ADR 0
#define MOUSE_3_ADR 0
#define MOUSE_4_ADR 0
#define MOUSE_5_ADR 0
#define MOUSE_6_ADR 0
#define MOUSE_7_ADR 0
#define MOUSE_8_ADR 0
#define MOUSE_9_ADR 0
```

Par défaut, seule la souris 0 est initialisée avec la locomotive d'adresse 10.

Les lignes suivantes permettent de définir jusqu'à 5 alimentation analogique. Chaque alimentation analogique peut remplacer une locomotive numérique. Ainsi chaque fois que la locomotive numérique reçoit un ordre, l'alimentation analogique modifie sa tension moyenne en conséquence. Les paramètres sont décrits dans les commentaires :

```
// locomotives analogiques
// La fonction add_alim_analog(adr, pwm_vit, pwm_dir, pwm_cran1, pwm_cran28)
// permet de remplacer une loco numérique par une alimentaion analogique
// Pour le système, l'alimentation analogique obeit aux ordres de la loco numérique associée !
// adr        = adresse de la loco numérique associée (1-50). Si adr==0 l'alimentation
analogique n'est pas utilisée.
// pwm_vit    = numéro de la broche pwm gérant la vitesse (0-51)
// pwm_dir    = numéro de la sortie pwm gérant le sens    (0-51). Il est dommage d'utiliser une
sortie pwm pour le sens mais ca simplifie !
// pwm_cran1  = valeur de la pwm (0-255) pour le cran numérique  1 (la locomotive doit démarer)
// pwm_cran28 = valeur de la pwm (0-255) pour le cran numérique 28 (la locomotive doit démarer)
// vous pouvez définir jusqu'a 5 alimentations analogiques
#define CONFIG_ALIM_ANALOG \
add_alim_analog(50, 48, 49, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
add_alim_analog( 0,  0,  0, 25, 255); \
```

Les lignes suivantes permettent de choisir le protocole de communication avec un ordinateur ainsi que sa vitesse. Fdcc_pc.py utilise le protocole UU à 115200bits par seconde.

```
// Protocol de communication centrale <--> PC
// 1=utilisation du protocol (0=non). Seul un protocol peut être utilise !
#define use_UU        1  // 1 = protocol UU
#define use_6050      0  // 1 = protocol 6050 Marklin
#define use_DBG_mouse 0  // 1 = debug mouse
//vitesse de communication parmi:
#define RS232_SPEED 115200  // 1200, 2400, 4800, 9600, 19200, 115200bps
```

La ligne suivante indique à la centrale dans quel ordre, les modules du bus Sout ont été raccordés.
Dans l'exemple suivant, un module à 48 aiguillages (aig0-47) est connecté à la centrale, puis un module à 8 sorties (out0-7) est branché sur le module des aiguillages …

```
//definir les modules OUT et AIG sur le bus Sout du plus proche au plus lointain de la centrale
//(sans importance pour centrale type B)
unsigned char sout_config[] = { SOUT_AIG0_47, SOUT_OUT0_7, SOUT_OUT8_15, SOUT_OUT16_23,
SOUT_OUT24_31, SOUT_AIG48_95,  SOUT_OUT32_39, SOUT_OUT40_47, SOUT_OUT48_55, SOUT_OUT56_63,
SOUT_END };
```

Les lignes suivantes configurent les aiguillages.
Les centrales autorisent un maximum de 96 aiguillages.
Ces aiguillages sont à choisir parmi les aiguillages sur :
- un des 2 modules à 48 aiguillages sur bus Sout pour la centrale A
- les 16 aiguillages directs de la centrale B
- un des 48 servomoteurs

Le premier paramètre est le numéro de l'aiguillage tel qu'il est présenté au logiciel
Le second est le type de module (AIG_SOUT_DIRECT pour les aiguillages sur SOUT de la centrale A ou direct sur la B) (AIG_SRV si l'aiguillage utilise un servomoteur)
Le 3eme paramètre indique le numéro de l'aiguillage sur le module (0-95 pour les modules Sout (0-47 pour le premier et 48-95 pour le second), 0-15 pour les aiguillages directs, 0-47 pour les servomoteurs). Le numéro de l'aiguillage physique peut donc être différent de celui présenté au logiciel. Par exemple l'aiguillage 0 peut être en fait piloté par le servomoteur 15.
Le 4eme paramètre indique si la commande doit être inversée (suite à une erreur de câblage). Choisir entre AIG_NORMA et AIG_INV
Le 5eme paramètre indique si l'aiguillage est inutilisé si AIG_PULSE_X
Si l'aiguillage est utilisé, il faut spécifier la durée de l'impulsion de commande des bobines parmi 125ms, 250ms, 500ms, 1s, 2s, 5s, 10s. Pour les servomoteurs ce temps n'est pas utilisé, mais il ne faut pas mettre AIG_PULSE_X !
Le 6eme paramètre indique la position à mettre à l'init.

```
// definir pour chacun des aiguillages 0-95, le module, le numero de l'aiguillage sur ce module,
l'invesion, la duree de l'impulsion ou la non utilisation, la commande a l'init
// tout aiguillage non declare ne sera pas utilise
#define AIG_SPECIFIC_INIT \
    config_aig(  0, AIG_SOUT_DIRECT,  0, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  1, AIG_SOUT_DIRECT,  1, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  2, AIG_SOUT_DIRECT,  2, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  3, AIG_SOUT_DIRECT,  3, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  4, AIG_SOUT_DIRECT,  4, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  5, AIG_SOUT_DIRECT,  5, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  6, AIG_SOUT_DIRECT,  6, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  7, AIG_SOUT_DIRECT,  7, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  8, AIG_SOUT_DIRECT,  8, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig(  9, AIG_SOUT_DIRECT,  9, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 10, AIG_SOUT_DIRECT, 10, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 11, AIG_SOUT_DIRECT, 11, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 12, AIG_SOUT_DIRECT, 12, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 13, AIG_SOUT_DIRECT, 13, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 14, AIG_SOUT_DIRECT, 14, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
    config_aig( 15, AIG_SOUT_DIRECT, 15, AIG_NORMAL, AIG_PULSE_500ms, AIG_DIR); \
```

Pour les aiguillages directs, il est possible d'inverser la commande

```
// inversion la commande des aiguillages directs pour la centrale B
// 1 = pour les modules de relais qui collent à l'etat bas au lieu de l'etat haut
#define use_inv_aig_direct_cmd 0
```

Les aiguillages à servomoteurs ont quelques particularités rappelées si dessous :

```
// Cas particulier des servos:
// - la duree de l'impulsion n'est pas valable pour les servos
//   (mais AIG_PULSE_X desactive quand meme l'aiguillage)
// - Utiliser io_srv_vit (pour la vitesse de deplacement des lames)
// - io_srv_pos0 pour la position directe
// - io_srv_pos1 pour la position deviee
// - io_srv_cmd  pour la position a l'init avant l'init des aiguillages
//   (normalmeent, on mettra la meme que pos0)
// - inv n'a pas d'impact
```

La ligne suivante permet d'activer ou pas l'I2C. Si l'I2C est utilisé alors les sorties PWM49 et 50 ne peuvent plus être utilisées.

```
// I2C
#define use_i2c 1          //0=PWM49&50 / 1=I2C
```

Pour l'instant, il est possible de recorder jusqu'a 3 circuits PCA9685 sur le bus I2C. La détection de ces circuits est maintenant automatique et il n'y a plus besoin d'indiquer s'ils sont présents ou pas.

```
// PCA9685
// Les 3 PCA9685 sont maintenant automatiquement detectes
// - pca9685_0 adr0(A5=0 A4=0 A3=0 A2=0 A1=0 A0=0)
// - pca9685_1 adr1(A5=0 A4=0 A3=0 A2=0 A1=0 A0=1)
// - pca9685_2 adr2(A5=0 A4=0 A3=0 A2=0 A1=1 A0=0)
```

Chaque PCA9685 dispose de 16 sorties qui peuvent être utilisées en mode PWM ou servomoteur. Les lignes suivantes permettent d'indiquer le mode :

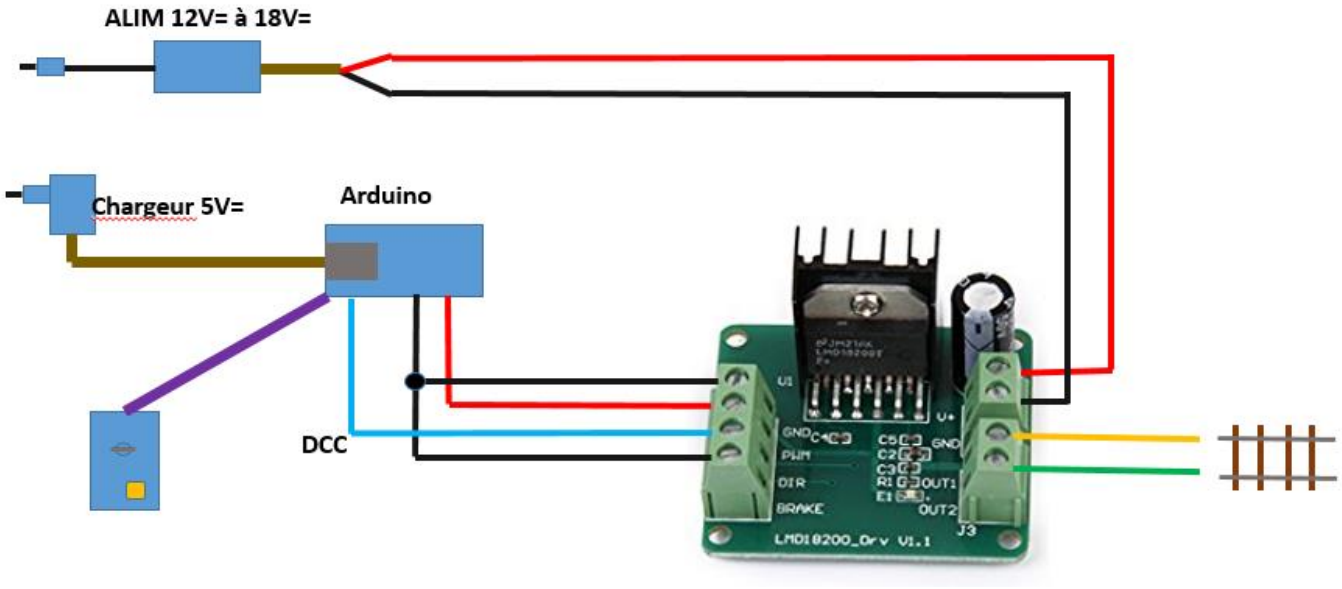
```
#define SRV 0
#define PWM 1
// choisir entre mode SRV ou PWM  (pca9685_0:0-15, pca9685_1:16-31, pca9685_2:32-47)
// numéro de la pwm ou du servo           0    1    2    3    4    5    6    7     ...
const byte pca9685_cfg[48]  PROGMEM = { SRV, SRV, SRV, SRV, SRV, SRV, PWM, PWM,    ...
```

Les lignes suivantes configurent les servomoteurs :

```
// vitesse des servomoteurs (1 =env 1deg/sec) choisir entre 1 (tres lent) et 255 (tres rapide)
// position des servomoteurs a l'initialisation (le PC va surement la changer !)
//   ( multipliez par 10us -> 150=1500us=1.5ms=neutre)
// pos0 et pos1 si le servo est utilise par les aiguillages
// io_srv_aig a 1 indique que le servo est utilise pour un aiguillage
//   (Il ne repondra pas a un ordre de mise a jour srv du PC)
// io_srv_aig a 0 indique que le servo n'est pas utilise pour un aiguillage
//   (Il ne repondra a un ordre de mise a jour d'aiguillage
// numéro du servo                        0    1    2    3    4    5    6    7       ...
const PROGMEM byte io_srv_vit[48]  = {   1,   2,   3,   4,   5,  10,  20,  50,      ...
             byte io_srv_cmd[48]  = { 150, 150, 150, 150, 150, 150, 150, 150,      ...
const PROGMEM byte io_srv_pos0[48] = { 100, 100, 100, 100, 100, 100, 100, 100,      ...
const PROGMEM byte io_srv_pos1[48] = { 200, 200, 200, 200, 200, 200, 200, 200,      ...
const PROGMEM byte io_srv_aig[48]  = {   0,   0,   0,   0,   0,   0,   0,   0,      ...
```

5.2. Utilisation en autonome

Après avoir réalisé l'électronique et configuré la centrale, Il est maintenant temps de jouer !



Au niveau électronique, nous pouvons réaliser le montage ci-dessus.
A la place de l'alimentation 5V, vous pouvez bien entendu utiliser un PC.
Avec ce simple montage vous pouvez conduire les locomotives avec des souris.
(Voire le chapitre souris pour l'utilisation des souris).

Les utilisateurs avancés pourront aussi utiliser toutes les entrées et sorties du système en autonome. Pour cela, il leur faudra mettre du code dans la fonction user_loop(). Il est par exemple possible de gérer la signalisation (leds) en fonction des entrées. Créer des itinéraires que l'on déclenche en appuyant sur un bouton … La fonction user_init() appelée en tout début du programme peut être utilisée pour les initialisations. Les utilisateurs « normaux » feront ces automatisations bien plus facilement avec un PC. En cas d'utilisation avec un PC, il faudra prendre garde que les 2 parties n'interfèrent pas. Par exemple, le PC et user_loop() commandant la même led. Exemple :

```
void user_loop(void)
{
    //ex: allumage led 5 si in 3
    if(io_in[3/8]&(1<<(3%8))) io_led[5/8]|=(1<<(5%8)); else io_led[5/8]&=~(1<<(5%8));

    //ex: feu d'un canton
    // led 10 = v  /  led 11 = r  /  led 12 = j
    // in0 = occupation canton  /  in1 = occupation canton suivant
     io_led[11/8]&=~(1<<(11%8)); // extinction de toutes les leds
     io_led[12/8]&=~(1<<(11%8));
     io_led[13/8]&=~(1<<(11%8));
     if(     io_in[0/8]&(1<<(0%8))) io_led[11/8]|=(1<<(11%8));  //canton occupe = rouge
     else if(io_in[1/8]&(1<<(1%8))) io_led[12/8]|=(1<<(12%8));  //canton libre et
canton suivant occupe = jaune
     else                           io_led[10/8]|=(1<<(10%8));  //canton libre et
canton suivant libre = vert
}
```

6. Les logiciels PC

6.1. Les protocoles de communication entre la centrale et le PC

Pour communiquer avec un PC, la centrale dispose de 2 protocoles de communication.

- Le protocole standard Marklin 6050 permet à de nombreux logiciels existants de piloter la centrale.

- Le protocole propriétaire que j'ai codé « UU » et qui permet d'utiliser toutes les fonctionnalités de la centrale. Mes Logiciels fonctionnent bien entendu avec ce protocole.

Le choix du protocole se fait dans le programme avec les lignes suivantes :

```
// Protocol de communication centrale <--> PC
// 1=utilisation du protocole (0=non). Seul un protocole peut être utilise !
#define use_6050 0              //protocol 6050 Marklin
#define use_UU   1              //protocol UU
//vitesse de communication parmi:
// 1200, 2400, 4800, 9600, 19200, 115200bps
#define RS232_SPEED 115200
```

Vous pouvez aussi choisir la vitesse de communication.

La description des protocoles vous permet de créer vous-même un logiciel capable de piloter les centrales.

Le protocole standard Marklin 6050 (aussi connu sous le nom P50)

Ce protocole utilisé depuis longtemps pour piloter la centrale Marklin 6050 qui reposée alors sur le protocole Motorola et non DCC est devenu au fil du temps un standard de fait et de nombreux logiciels l'utilisent. Il est très simple à utiliser et permet de:

- Contrôler 80 locomotives en sens / vitesse / fonctions F0-F4 (limitation à 50 avec fdcc)
- Contrôler les accessoires (aiguillages, sorties, leds)
- Lire l'état des entrées des entrées (modules S88 par exemple)
- Gérer l'arrêt d'urgence

Le tableau suivant résume ce protocole :

Titre	Codage 1er octet	2eme octet	Commentaires
Vitesse	000.F0.VVVV	ADR loco	VVVV=0-14=vitesse VVVV=15=inv de sens
Fonctions	010.0.F4.F3.F2.F1	ADR loco	ADR=1-50
Aig/Out/Led	001.000.R.V	Num Aig/Out	R=rouge/dévié/off V=Vert/droit/on
AU	011.0000.AU		AU=1=Arrêt d'urgence
Lecture N S88	100.NNNNN	-> 16 entrées	NNNNN=Nb de S88 à lire (0=mode sans eff) Nb=1-16
Lecture 1 S88	110.NNNNN	-> 16 entrées	NNNNN=Num du S88 à lire (0=mode avec eff) Num=1-16

Arrêt d'urgence :

Pour l'arrêt d'urgence il suffit d'envoyer #01100001 soit 97 en décimal.
Pour revenir en mode normal il suffit d'envoyer #01100000 soit 96 en décimal.
De nombreux logiciels commencent par envoyer 2 arrêts d'urgence pour se synchroniser, font leurs initialisations puis envoient 1 mode normal.

Vitesse/Sens/F0 :

Le premier octet indique la vitesse, l'état de la fonction 0 (souvent l'éclairage) et le sens.
La vitesse est la suivante :

0 = arrêt
1 = cran de vitesse 1/14
14 = cran de vitesse 14/14
Bien entendu, la centrale adapte ces crans au système DCC (4-31)

Pour le sens, il n'y a pas de bit consacré mais seulement un ordre d'inversion lorsque l'on envoie la commande 15. A l'initialisation de la centrale tous les sens sont en avant, ensuite il convient au logiciel sur PC de mémoriser le sens courant et de le modifier si besoin.
Pour activer la fonction 0, il suffit d'ajouter 16 à la commande
Par exemple la commande 26 = #00011010 indique F0=on / vitesse = cran 10/14

Le second octet indique le numéro de la locomotive et le protocole définit 80 locomotives (1-80). Ce nombre est limité à (1-50) avec Free-DCC 2017.

Le bit F0 permet de contrôler la fonction 0 qui est souvent associée à l'allumage des phares.

Fonctions F1-F4
Cette commande permet de jouer avec les fonctions spéciales 1 à 4.
Par exemple #01001011+#00000001 active les fonctions 4, 2, 1 et désactive la fonction 3 de la locomotive qui se trouve sur le canal 1.

Lecture des modules de feedback S88

Le standard permet de lire 31 modules de 16 entrées, Free-DCC limite se nombre à 16, ce qui fait tout de même 16*16=256 entrées. Le mode de lecture peut être choisit avec ou sans effacement. (#1000000=sans effacement, #11000000=avec effacement). Cela signifie que l'état de l'entrée est remis à 0 après lecture et non conservée. Sauf cas particulier, il convient de choisir le mode avec effacement pour lire la valeur courante. Le mode par défaut est avec effacement. Il peut être changé dans le code si nécessaire.

Ensuite à vous de choisir si vous voulez lire :
- un module en particulier avec #110NNNNN avec NNNNN = numéro du module (1-16)
- un nombre spécifié de module avec #100NNNNN avec NNNNN = nombre de module (1-16). 1 pour le premier module, 2 pour les 2 premiers modules …

En général les logiciels lisent tous les modules à la fois puis interagissent sur les locomotives, aiguillages et sorties en fonction de ces entrées.

Commande des aiguillages, sorites et LEDs

Le premier octet définit l'état d'une ou de 2 sorties complémentaires #001000RV puis le second indique le numéro de la sortie.

Avec de vrai décodeur d'accessoires DCC, les sorties sont souvent utilisées 2 à 2 comme suit :
- Pour commander un aiguillage, une bobine est placée sur la première sortie et la seconde sur la 2eme sortie. - Pour les feux, la led verte est placée sur la première tandis que la rouge est placée sur la seconde
R.V=0.1 est donc utilisé pour mettre l'aiguillage en position direct ou le feu au vert
R.V=1.0 est utilisé pour mettre l'aiguillage en position dévié ou le feu au rouge
R.V=0.0 est utilisé pour arrêter l'impulsion sur l'aiguillage ou éteindre les 2 LEDs. 0.0 est de moins en moins utilisé car les décodeurs d'accessoires permettent maintenant de définir la longueur des impulsions pour les aiguillages et il n'y a aucun intérêt à éteindre toutes les led d'un feu.
Cette façon de coupler 2 sorties n'a que peu d'intérêt avec Free-DCC hormis pour les feux, si vous utilisez uniquement des feux à 2 leds. Les accessoires sont le point faible de la norme DCC car il faut beaucoup de temps pour en changer un grand nombre, ce PB se retrouve sur ce protocole car il faut envoyer 2 octets pour changer une sortie alors qu'avec ces 2 octets (16 bits), on aurait pu en changer 16 !

Le tableau suivant indique les accessoires commandés avec Free-DCC en fonction de l'adresse

Type	Adresses	Nb	Fonctionnement (octet 1 & octet 2)
Aiguillages	0-95	96	00100001 & 0 -> aig0 direct pas besoin d'envoyer 00100000 pour 00100010 & 0 -> aig0 dévié stopper l'impulsion car automatique
Sorties	96-127	32	00100001 & 100 -> out4=1 00100010 & 100 -> out4=0
LEDs	128-255	128	00100001 & 128 -> led0=1 00100010 & 128 -> led0=0

Ce protocole est très simple à utiliser. Il constitue un point d'entrée très accessible pour les personnes qui veulent coder leur propre logiciel de contrôle. On trouve d'ailleurs sur Internet quelques exemples.

Vous noterez néanmoins que toutes les possibilités des centrales ne peuvent pas être utilisées. Limitation à 32 sorties et 256leds. Pas de PWM, entrées analogiques, gestion des souris ou néopixels.

Le protocole propriétaire UU

Ce protocole plus complet est vraiment spécifique à la centrale et aucun logiciel tiers ne l'implémente. Tous mes logiciels l'utilisent car il est le seul à pouvoir utiliser toutes les fonctionnalités de la centrale. Je ne le décri pas en détail car les débutant commenceront pas le 6050, mais voici toutes les commandes :

```
/* Commandes du protocol "UU" (toute autre commande recue est sans effet) */

#define RS232_CMD_PING     0x00   // PING      0x00 -> 0x83
#define RS232_CMD_VERS     0x01   // VERS      0x00 -> 16 octets d'information (0x84.........)

#define RS232_CMD_INIT     0x78   // INIT      0x78 + 0.0.CLRIN.AIG.PHA.CLI.LED.OUT
#define RS232_CMD_CFG      0x79   // CFG       0x79 + action(2=aig_off, 3=aig_on)
#define RS232_CMD_AU0      0x70   // AU#       0x70
#define RS232_CMD_AU1      0x71   // AU        0x71
#define RS232_CMD_AUGET    0x72   // AUGET     0x72 -> AU

#define RS232_CMD_LOCO     0x10   // LOCO      0x10 + ADR(1-50)  + VIT(xxS[0-28])  + FCT
(xxxF4F3F2F1F0)
#define RS232_CMD_LOCOIN   0x11   // LOCO-IN   0x11 + ADR(1-50) -> VIT(xxS[0-28]) -> FCT
(xxxF4F3F2F1F0)
#define RS232_CMD_LOCOPGM  0x12   // LOCOPGM   0x12 + 0x55 + 0xAA ++ CV(1-1024) + DAT(0-255)
#define RS232_CMD_LOCOMODE 0x13   // LOCOMODE  0x13 + mode (0=DCC, 1=ANALOGIQE, 2=0, 3=1)

#define RS232_CMD_MOUSE    0x60   // MOUSE     0x60 + #mou(0-9) + adr    (ne fait plus: + vit +
fct)
#define RS232_CMD_MOUSEIN  0x61   // MOUSE-IN  0x61 + #mou(0-9) -> adr
#define RS232_CMD_MOUSEIN2 0x62   // MOUSE-IN2 0x62 + #mou(0-9) -> adr -> vit -> fct

#define RS232_CMD_AIGDIR   0x20   // AIG-DIR  0x20 + #aig(0-95)
#define RS232_CMD_AIGDEV   0x21   // AIG-DEV  0x21 + #aig(0-95)
#define RS232_CMD_AIGPOS   0x22   // AIG-POS  0x21 + #aig -> T1.N6.N5.N4.N3.N2.N1.N0
                                  //                    + inv.pulsems(3bits).T0.pos1.pos0.cmd
#define RS232_CMD_OUT0     0x30   // OUT-OFF  0x30 + #out(0-63)
#define RS232_CMD_OUT1     0x31   // OUT-ON   0x31 + #out(0-63)
#define RS232_CMD_PWM      0x32   // PWM      0x32 + #out(0-3) + val

#define RS232_CMD_LEDOFF   0x40   // LED-OFF  0x40 + #led(0-255)
#define RS232_CMD_LEDON    0x41   // LED-ON   0x41 + #led(0-255)
#define RS232_CMD_LEDCLI0  0x42   // LED-CLI0 0x42 + #led(0-255)
#define RS232_CMD_LEDCLI1  0x43   // LED-CLI1 0x43 + #led(0-255)
#define RS232_CMD_LEDPHA0  0x44   // LED-PHA0 0x44 + #led(0-255)
#define RS232_CMD_LEDPHA1  0x45   // LED-PHA1 0x45 + #led(0-255)
#define RS232_CMD_NEO      0x48   // NEO      0x48 + #neo(0-59) + R + V + B

#define RS232_CMD_IN       0x50   // IN       0x50 -> in7..0 -> in15..8 -> ... -> in 255..248
#define RS232_CMD_INMEM    0x51   // IN-MEM   0x51 "
#define RS232_CMD_INMCLR   0x53   // IN-MCLR  0x53 "
#define RS232_CMD_AN       0x58   // AN       0x58 + #an(0-19) -> val

#define RS232_CMD_BYTE     0x49   // BYTE     0x49 + #byte + val
#define RS232_CMD_BIT      0x4a   // BIT      0x4a + #bit  + val
```

PS : Pour être sûr d'avoir la version la plus à jour, regardez directement dans le programme Arduino des centrales.

6.2. Installation (de Python et pyserial)

Installation de Python

Python est un interpréteur qui lit le fichier du programme (par exemple fdcc_pc.py) et l'exécute. Pour l'installer, téléchargez Python en version 3, par exemple 3.5.2 sur https://www.python.org/
Attention, free-DCC ne fonctionne pas (encore) en Python2 !
Apres l'installation, quand vous cliquerez sur un fichier avec l'extension .py il sera exécuté par Python. Vous pouvez ouvrir les programmes Pythons avec un éditeur de texte basic (Notepad étant le plus simple) pour voire le code source et le modifier.

Le choix de programmer en Python a été fait pour 3 raisons :

- Le programme est interprété et il n'y a donc pas besoin de le compiler. Vous pouvez, ouvrir le fichier, modifier le programme, le sauver. La prochaine fois que vous l'exécuterez, votre modification sera prise en compte. Cela est aussi très pratique pour la configuration : On peut mettre la configuration dans le code, plutôt que de créer de multiples fichiers de configuration.
- L'interpréteur Python est disponible pour tous les OS. Vous pourrez donc utiliser ces logiciels sous Windows, Linux, (Je n'ai pas testé sur Mac) … Le cas de Linux est très intéressant, car outre le PC, il est possible de faire tourner les logiciels sur des petites cartes comme la Raspberry PI.
- C'est un langage simple et puissant.

Attention sous Windows, si vous avez Python2 et Python3, vérifiez que les variables d'environnement pointent bien vers Python3. Exemple, dans la variable système, PATH, il devrait y avoir : …C:\Python35\Scripts\;C:\Python35\...
En principe, c'est le dernier Python installé qui gagne !

Installation de pyserial

Pour communiquer avec la carte Arduino le logiciel fdcc_pc.py a besoin de la librairie de gestion du port série « pyserial ». Malheureusement cette librairie n'est pas installée par défaut. Pour l'installer, rendez-vous sur: https://pypi.python.org/pypi/pyserial. Si cela marche, vous devriez voire un nouveau port série (qui en fait passe par l'USB) dans le gestionnaire de périphérique (sous Windows) ou un nouveau fichier de périphérique du type /dev/ACM0 sous Linux.
Pensez à mettre à jour le numéro du port série virtuel dans le logiciel PC fdcc_pc en gardant les \ :
LIST_SERIAL_PORTS = ["\\\\.\\COM6"]

Si vous n'avez pas l'habitude d'installer des librairies Python, cela peut vite devenir fastidieux avec les outils qui marchent plus ou moins bien. Je vous conseille de télécharger l'archive, comme par exemple : pyserial-3.2.1.tar.gz. La décompresser ou vous voulez, puis exécuter setup.py :

- Sous Windows, en tapant dans une fenêtre DOS : python setup.py install
- Sous Linux : sudo python3 setup.py install

Pour vous aider, j'ai rajouté pyserial dans l'archive de free-DCC, je l'ai repackagé en pyserial-3.2.1.zip (car le tar posait des problèmes à certains).

J'ai aussi ajouté un script install.bat qui tape la commande automatiquement pour les utilisateurs de Windows lorsqu'on l'exécute.

Sous Windows, certains ont eu les problèmes suivants :

- répertoire de python protégé en écriture
- variables d'environnements qui ne pointent pas vers Python3 (si vous avez aussi Python2 installé)
- erreur avec l'outil miniterm (dans ce cas mettre un # devant script à la fin du fichier setup.py)
  #scripts=['serial/tools/miniterm.py']

6.3. Les différents logiciels

Je livre free-dcc les logiciels suivants :

- fdcc_pc.py : C'est le logiciel principal. C'est lui qui est connecté aux centrales. Il permet de tout faire. Gestion des locomotives (sélection, programmation, conduite avec des souris virtuelles), TCO, automatisation (équations, scripts) et tester les entrées/sorties. De plus il permet à d'autres logiciels de s'y connecter.

- fdcc_regu.py : Ce logiciel est une souris virtuelles qui peut se connecter par le réseau à fdcc_pc.py. On peut donc conduire une locomotive à partir d'un autre PC. Bien entendu il est possible d'avoir plusieurs logiciels.

- cc6500.py / bb67400.py / y8000.py / ead.py / etg.py / rtg.py : Ces logiciels sont des cabines virtuelles qui peuvent se connecter par le réseau à fdcc_pc.py. On peut donc conduire une locomotive à partir d'un autre PC à partir d'une véritable cabine reproduite. Il est même possible de connecter ce logiciel à une carte Arduino pour y brancher des manettes afin de reproduire les commandes de l'engin réel !

- fdcc_tco_net.py : Le logiciel fdc_pc.py permet d'afficher 10 TCO. Le logiciel fdcc_tco_net.py se connecte à fdcc_pc.py par le réseau informatique (Ethernet, wifi) et affiche un de ses TCO. On peut ainsi piloter un réseau à partir d'un autre PC. Il est possible d'utiliser plusieurs instances de ce logiciel afin d'avoir plusieurs TCO.

Notez que des logiciels tiers peuvent aussi être utilisés s'ils communiquent avec la centrale par le protocole P50.

J'ai eu beaucoup de demandes me demandant si la centrale pouvait être utilisée avec CDM et RocRail. Ces 2 logiciels utilisent le protocole P50x qui est différent du P50. Donc actuellement la réponse est non. J'essaierai dans le futur de supporter le protocole P50x …

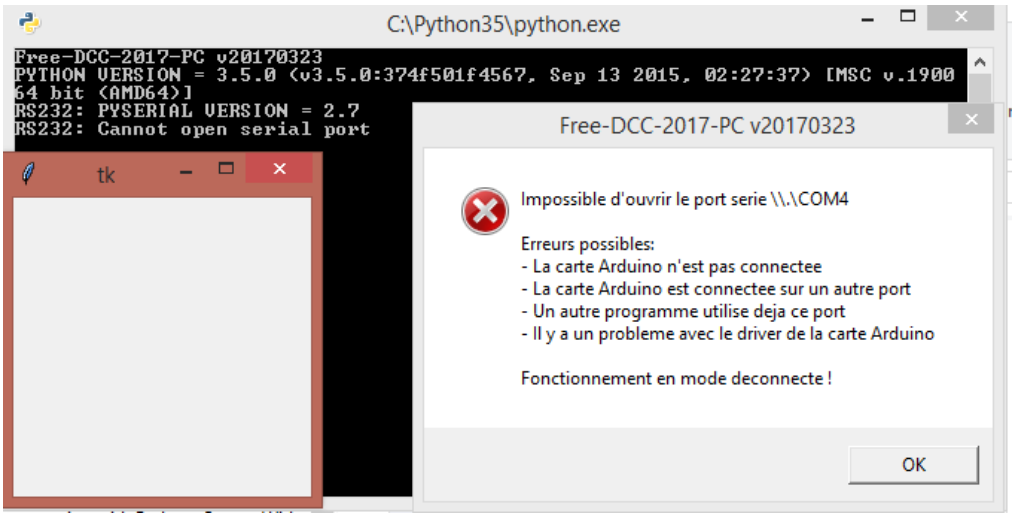
6.4. Le logiciel fdcc pc

Le logiciel fdcc_pc.py est le logiciel principal. C'est lui qui est connecté aux centrales. . Il permet de tout faire. Gestion des locomotives (sélection, programmation, conduite avec des souris virtuelles), TCO, automatisation (équations, scripts) et tester les entrées/sorties. De plus il permet à d'autres logiciels de s'y connecter.

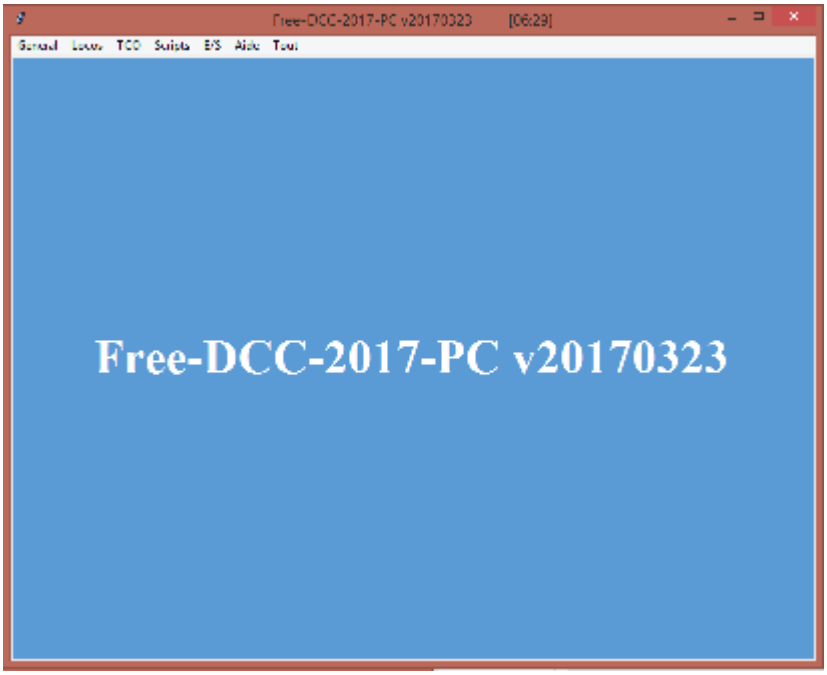
A. Ouverture/Fermeture :

Une fois python et pyserial installés, Téléchargez et décompressez l'archive ou vous voulez. Cliquez ensuite sur fdcc_pc.py pour lancer le logiciel.
Vous devriez voir les fenêtres suivantes :

- Une fenêtre noire avec du texte ou s'affiche du debug.
- L'interface graphique tk (un peu vide pour le moment)
- Une magnifique erreur indiquant un problème de communication avec la carte.



L'erreur est normale si la carte n'est pas branchée ou si pyserial n'est pas installé ou si le numéro du port série est mauvais. Pas de panique, le logiciel peut fonctionner en mode déconnecté. Appuyez juste sur OK.



Bravo ! Le logiciel fonctionne. Pour fermer le programme, appuyez sur la croix ou utiliser le menu Fichier->Sortir. (Il y a un bug à la fermeture actuellement et il vous faudra aussi fermer la fenêtre de debug). Voilà, vous savez lancer et arrêter le logiciel !!

Pour communiquer avec votre carte Arduino, il faut indiquer à fdcc_pc le numéro du port série virtuel ou est branchée la carte.

Editez fdcc_pc.py avec un éditeur de texte comme Notepad, chercher la section « user config variables » et renseigner le numéro de votre port série.

```
#-------------------------------------------------------------------------------------
# U S E R   C O N F I G   V A R I A B L E S
#-------------------------------------------------------------------------------------

# Liste des ports series sur lesquels sont connectes les centrales
# Pensez à renseigner le numero de la carte dans les centrales
# Avec 1 port  serie , le logiciel s'attend a avoir la carte 0
# Avec 2 ports series, le logiciel s'attend a avoir les cartes 0 et 1 ...
# Ex sous Windows:
# - LIST_SERIAL_PORTS = [ "\\\\.\\COM6" ]
# - LIST_SERIAL_PORTS = [ "\\\\.\\COM6", "\\\\.\\COM4" ]
# Ex sous Linux:
# - LIST_SERIAL_PORTS = [ "/dev/ACM0" ]
# - LIST_SERIAL_PORTS = [ "/dev/ACM0", "/dev/ACM1" ]
LIST_SERIAL_PORTS = [ "\\\\.\\COM6" ]

# Numero du port TCP-IP pour y connecter les cabines et TCO distants
fdcc_server_port = 1234
```

Pour trouver ce numéro, vous pouvez aller dans le gestionnaire dans le gestionnaire de périphérique (sous Windows). Une autre méthode est d'utiliser l'environnement Arduino (Outils -> Port)
Par exemple, si votre port est le COM4, remplacer le COM6 par défaut par COM4. (Garder les \\\\.\\)

Sous Linux, il faudra utiliser les ACM comme « /dev/ACM0 » ou les USB comme « /dev/USB0 »

Si vous utilisez plusieurs centrales, renseignez les différents ports séries. L'ordre importe peu puisque fdcc_pc interroge les cartes pour récupérer le numéro de la carte définit par la ligne suivante dans le logiciel de la centrale :
`#define num_centrale 0`
Si vous utilisez 2 centrales, le logiciel s'attend à trouver la 0 et la 1.
Si vous utilisez 3 centrales, le logiciel s'attend à trouver la 0, la 1 et la 2 …
Si ce n'est pas le cas, il indiquera une erreur et fonctionnera en mode déconnecté.

Dans fdcc_pc, pour accéder aux variables des centrales, il faut ajouter 1000 fois le numéro de la centrale.
Par exemple, la 1ere entrée de la centrale 0 est i0. Pour la centrale 1, ce sera i1000.
Bien entendu, les variables qui ne dépendant pas des centrales comme les itinéraires ne suivent pas cette règle.

Avec fdcc_pc, seule la centrale 0 contrôle les locomotives, souris et arrêt d'urgence. Ce qui est assez logique car le signal DCC est commun à l'ensemble du réseau.

Pour que la communication fonctionne, fdcc_pc et les centrales doivent avoir des versions compatibles. Les archives zip du site contiennent bien évidement des versions compatibles.

La section de config permet également de configurer le port réseau sur lequel peuvent se connecter les cabines et TCO distants.

B. Les locomotives

Le menu « Locos -> Sélection » affiche la fenêtre de sélection des locomotives DCC. Chaque rectangle représente une des 50 locomotives supportées par fdcc. Dans chaque rectangle sont affichés l'adresse de la locomotive ainsi que son sens, sa vitesse et ses fonctions spéciales F0-F4. Cliquer sur un rectangle, ouvre le régulateur virtuel associé à sa locomotive afin de conduire cette dernière. Toute locomotive en mouvement est signalée par son rectangle peint en vert.

Pour plus de commodité, il est possible d'associer une image et un nom à chaque locomotive. Pour cela, ouvrez le fichier config_loco.txt et indiquez le nom de la locomotive et son image. Les images sont dans le répertoire img. Une bonne taille est 40 sur 48 pixels. J'ai utilisé des images en provenance de http://www.trainfrontview.net/ . J'en ai modifié certaines lorsque la locomotive n'existait pas sur ce site. Le logiciel est livré avec l'exemple ci-dessus. Le listing ci-dessous montre les premières lignes du fichier :

```
#locos
6
BB66001
img/bb66000.gif
7
BB66002
img/bb66000.gif
```

L'arrêt d'urgence de toutes les locomotives du réseau peut être obtenu et annulé avec les boutons AU ou via le menu « locos -> arrêt d'urgence » ou « locos -> fin d'arrêt d'urgence »

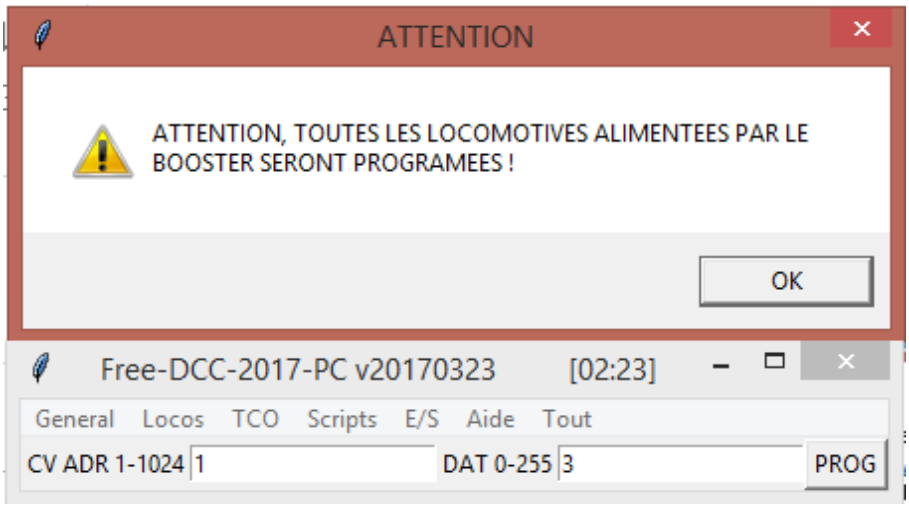
Les ronds S0 à S9 représentent les souris physiques. Il est possible de les faire glisser pour sélectionner ou désélectionner une locomotive. Ils bougent automatiquement lors d'une sélection par les souris.

Toutes les informations de la fenêtre de sélection et des régulateurs virtuels sont aussi mises à jour chaque fois qu'il y a un changement demandé par une cabine de conduite, une souris physique ou encore par le script d'automatisation. Il n'y a aucun verrouillage entre les modes si bien qu'il est par exemple possible de changer la vitesse d'une locomotive utilisée par le script, ce qui peut être bien comme catastrophique ;-)

La commande des locomotives analogique n'a pas encore été implémentée.

Pour utiliser les cabines de conduite, se reporter au chapitre adéquat.

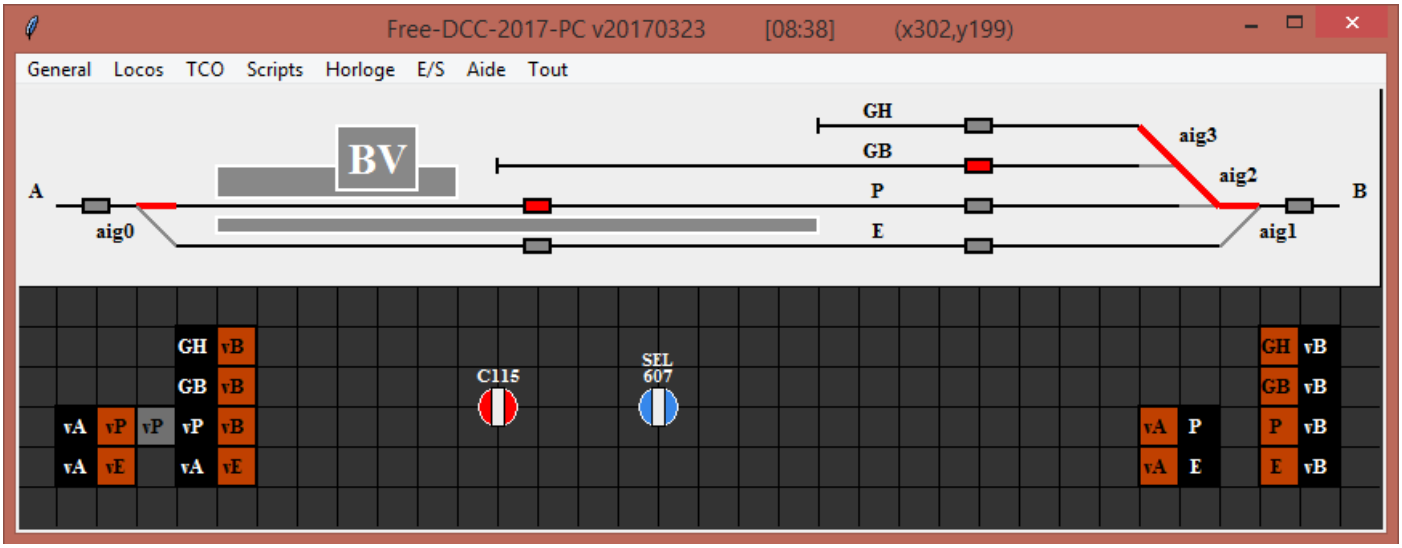
Enfin, il est possible de programmer les CV (paramètres) des décodeurs des locomotives avec le menu « locos -> prog CV ». Un message vous indique en premier de faire attention car toutes les locomotives alimentées par le booster seront programmées ! Vous pouvez par exemple retirer toutes les autres locomotives du réseau ou alors avec un interrupteur alimenter seulement une portion de voie pour programmer la locomotive. Après ce message d'alerte, une fenêtre permet d'entrer le numéro du CV (1-1024) et sa valeur (0-255). Par exemple pour affecter l'adresse 3 à la locomotive, il faut utiliser le CV1 : CV-ADR=1 CV-DAT=3. Après appuie sur le bouton PROG, le CV devrait être programmé. Pour acquitter la programmation, vous devriez voire la locomotive bouger très brièvement. Certains décodeurs restent sur les anciennes valeurs tant que l'alimentation perdure. Dans ce cas, en fin de programmation, enlevez puis remettez la locomotive sur les rails !



C. Les TCO

Les TCO (pour Tableau de Control Optique) permettent de voir l'état du réseau et de le commander. Afin de ne pas avoir un TCO spaghetti très peu réaliste qui représente l'ensemble du réseau, le logiciel permet d'afficher jusqu'à 10 TCO. Mettez-en au moins un par gare pour faire vrai.

La capture suivante présente le TCO d'une petite gare. On y trouve, les voyants d'occupation, la position des aiguillages (qui se modifie si on clique dessus), 13 boutons pour les itinéraires (12 TP et 1 DA ce qui n'est pas très réaliste !). Un essai de 2 sélecteurs. Il manque encore la signalisation (représentation des carrés), un commutateur de fermeture par carré…



Sur les TCO, vous pouvez représenter le réseau avec des lignes/rectangles, ajouter des indicateurs et des commandes.

Il est très difficile de satisfaire les utilisateurs car les symboles sont toujours trop grands, trop petits, de la mauvaise couleur … Pour pallier ce problème, tout est configurable dans cette nouvelle version et il n'y a plus de symboles prédéfinis ! Ce sera un peu plus difficile pour vous à mettre au point, mais au moins, vous aurez ce que vous voulez !

Le fichier config_tco.txt contient vos TCO. Il « suffit » d'y ajouter des instructions pour créer vos lignes, rectangles, ovales et indiquer leurs couleurs en fonction des variables du système. Vous pouvez même indiquer les zones à cliquer qui modifient les variables.

Pour les positions, nous utiliserons les coordonnées (x,y) des pixels de la fenêtre du TCO. Pour vous aider, elles sont affichées dans la barre des titres des TCO à chaque click. (ex : x=304, y=195).

Pour les couleurs, nous utiliseront le format rouge-vert-bleu hexa. 00=0% sur la couleur sélectionnée, FF=100%. . Il est ainsi possible d'utiliser 16 millions de couleur. Mettre un # devant est une pratique courante. Par exemple :
Noir: #000000
Blanc: #FFFFFF
rouge : #FF0000
vert : #00FF00
bleu : #0000FF

Les instructions sont les suivantes :

```
# instructions des TCO
#
# #       commentaire (# seulement sur la première colonne)
# tco     tco_num
# grid    dx,dy,gdx,gdy color_grid color_fond
# txt     x,y "txt" color font_size
# txtc    x,y "txt" color font_size
```

```
# txt2    "txt" color font_size
# txtc2   "txt" color font_size
# line    x1,y1,x2,y2 color size
# rect    x1,y1,x2,y2 color_border color size
# rectc   xc,yc,dx,dy color_border color size
# ovalc   xc,yc,dx,dy color_border color size
# color   [!]var color              s'applique au dernier line,oval,rect
# color2  [!]var color color0
# size    [!]var size               " (size 0 = invisible)
# size2   [!]var size size0         " (size 0 = invisible)
# action  x1,y1,x2,y2 var[+,-,^]    +:mise a 1 / -: mise a 0 / ^:changement d'etat
# actionc xc,yc,dx,dy var[+,-,^]    !:changement color/size si la variable est a 0
# action2 var[+,-,^]    +:mise a 1 / -: mise a 0 / ^:changement d'etat
```

sur la première colonne est utilisé pour signaler que la ligne est un commentaire. Il est interdit de mettre un commentaire en dehors d'une ligne avec un # sur la première colonne.
Vous pouvez mettre des lignes vide.

tco suivit du numéro du tco (0-9) indique à quel tco se réfèrent les prochaines instructions
ex : tco 0 pour le 1er TCO

grid permet de dimensionner la fenêtre (dx,dy), la taille de la grille (gdx, gdy) ainsi que la couleur de la grille et celle du fond.
Si vous ne souhaitez pas utiliser la grille, mettre 0 dans gdx.
Ex de la capture :
tco de ma gare A
tco 0
grid 850,275,25,25 #888888 #5B9BD5

txt permet de placer un texte. Il débutera à droite du point (x,y). Ne pas oublier les guillemets autour du texte. Pour l'instant, les espaces sont interdits. Ne pas oublier la taille de la fonte.
Ex : txt 52,162 "iti0" #000000 10

txtc permet de placer un texte centré sur le point (x,y).
Ex : txtc 52,162 "iti0" #000000 10

txt2 permet de placer un texte à droite de la dernière forme créée parmi (rect,rectc,ovalc)
Ex : txt2 "iti0" #000000 10

txtc2 permet de placer un texte au centre de la dernière forme créée parmi (rect,rectc,ovalc)
Ex : txtc2 "iti0" #000000 10

line permet de placer une ligne. (x1,y1) est le point de départ et (x2,y2) le point d'arrivé. Size permet de spécifier l'épaisseur de la ligne.
Ex : line 100,100,725,100 #000000 2

rect trace un rectangle. (x1,y1) est le point en haut à gauche et (x2,y2) le point en bas à droite. Size permet de spécifier l'épaisseur de la ligne de contour et color_border sa couleur. Color étant la couleur de remplissage.
Ex : rect 200,90,250,50 #FFFFFF #888888 2

rectc est une 2eme façon de créer un rectangle en spécifiant son centre (xc,yc), sa demi longueur (dx) et sa demi hauteru (dy). On a donc x1=xc-dx, y1=yc-dy, x2=xc+dx, y2=yc+dy. Cela est très pratique pour créer les voyants et interupteurs.
Ex : rectc 600,50,10,5   #000000 #FF0000 2

ovalc crée un ovale à la place d'un rectangle. Si dx == dy alors l'oval est un rond.
Ex : ovalc 800,50,5,5 #000000 #FF0000 2

color permet de modifier la couleur de la dernière forme créée (line,rect,rectc,ovalc) en fonction de l'état d'une variable. Par exemple pour avoir un voyant rouge lorsqu'il y a une détection sur l'entrée in1 et gris dans le cas contraire :

```
# voyants d'occupation
rectc 600,50,10,5      #000000 #FF0000  2
color i1 #FF0000
color !i1 #888888
```

Vous pouvez aussi par exemple choisir de colorer la ligne du secteur …
J'ai utilisé cela pour la position des aiguillages
! indique que la variable doit être à 0 (les variables sont décrites plus loin)

color2 est une optimisation de color. Elle permet de modifier la couleur de la dernière forme créée (line,rect,rectc,ovalc) en fonction de l'état d'une variable. Si l'état est ok alors la première couleur est utilisée, sinon la seconde. L'exemple précédent peut ainsi être optimisé:

```
# voyants d'occupation
rectc 600,50,10,5      #000000 #FF0000  2
color2 i1 #FF0000 #888888
```

size permet de modifier l'épaisseur de la dernière forme créée (line,rect,rectc,ovalc). C'est surtout utile pour une ligne. Vous pouvez par exemple changer la taille d'une ligne pour représenter une position d'aiguillage active, ou un itineraire ou une occupation. Si size est à 0, alors la dernière forme n'est pas affichée …

```
#aig3 dir
line 700,75,725,75    #FF0000 4
color d3 #ff0000
size  d3 4
color !d3 #888888
size  !d3 2
#aig3 dev
line 700,50,725,75    #888888 2
color e3 #ff0000
size  e3 4
color !e3 #888888
size  !e3 2
```

size2 est une optimisation de size. Elle permet de modifier l'épaisseur la dernière forme créée (line,rect,rectc,ovalc) en fonction de l'état d'une variable. Si l'état est ok alors la première taille est utilisée, sinon la seconde. L'exemple précédent peut ainsi être optimisé:

```
#aig3 dir
line 700,75,725,75    #FF0000 4
color2 d3 #ff0000 #888888
size2  d3 4 2
#aig3 dev
line 700,50,725,75    #888888 2
color2 e3 #ff0000 #888888
size2  e3 4 2
```

action/actionc permettent de changer l'état d'une variable lorsque l'on clique sur une zone définit par le rectangle (x1,y1,x2,y2) ou (xc,yc,dx,dy).

Il est possible de mettre la variable à 1 avec +, à 0 avec – ou l'inverser avec ^.

```
# comande des aiguillages
action 75,100,100,125  a0^  # aig 0
action 750,100,775,125 a1^  # aig 1
action 725,75,750,100  a2^  # aig 2
action 700,50,725,750  a3^  # aig 3
```

action2 est une optimisation de action/actionc. Elle permet de changer l'état d'une variable lorsque la dernière forme parmi (rect,rectc,ovalc) est cliquée. Cela évite de rappeler les coordonnées. Exemple pour un interrupteur :

```
#inter led 23
rect 100,100,125,125 #000000 #FF0000  2
color2 l23 #FF0000 #888888
action2 l23^
```

Exemple de réalisation du commutateur C115. (qui agit que sur l21 pour l'instant) (Essayer de comprendre !):

```
#selecteur
txtc 300,180 "C115" #FFFFFF 8
ovalc 300,200,12,12 #FFFFFF #88aa88 1
actionc2 l21^
color2 !l21 #700707 #ff0000
rectc 300,200,12,4  #000000 #EEEEEE 1
size2 !l21 1 0
rectc 300,200,4,12  #000000 #EEEEEE 1
size2 l21 1 0
```

Les instructions demandent un peu de pratique, mais vous pouvez tout faire !

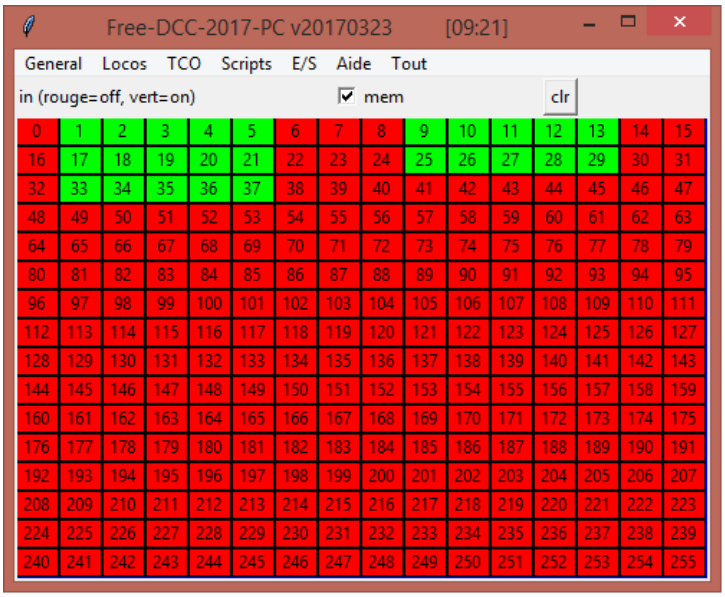
Les variables utilisables sont les suivantes :

```
# variables  : descritpion
#
#  i <0-255> : entree
#  y <0-255> : etat parecedent de l'entree
#  o <0-63>  : sortie
#  l <0-255> : led
#  c <0-255> : clignotement de led
#  p <0-255> : phase de led
#  a <0-95 > : commande aiguillage
#  d <0-95 > : aiguillage en position directe
#  e <0-95 > : aiguillage en position dévié
#  t <0-999> : itineraire cmd(ecriture) et ok(lecture)
#  v <0-999> : variable
```

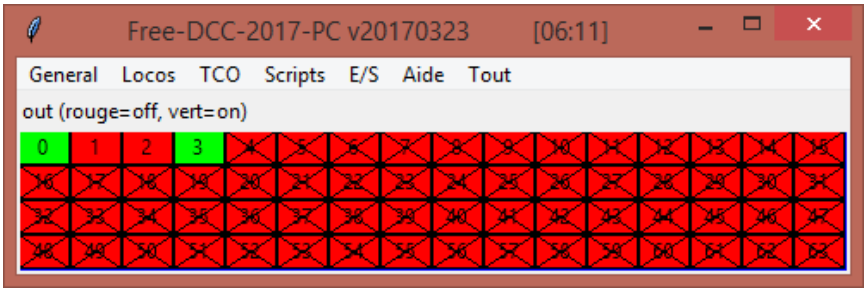
D. Test des entrées/sorties

Le menu E/S permet de visualiser les entrées, sorties, leds et position des aiguillages. Il permet également de tester les sorties, leds et aiguillages.

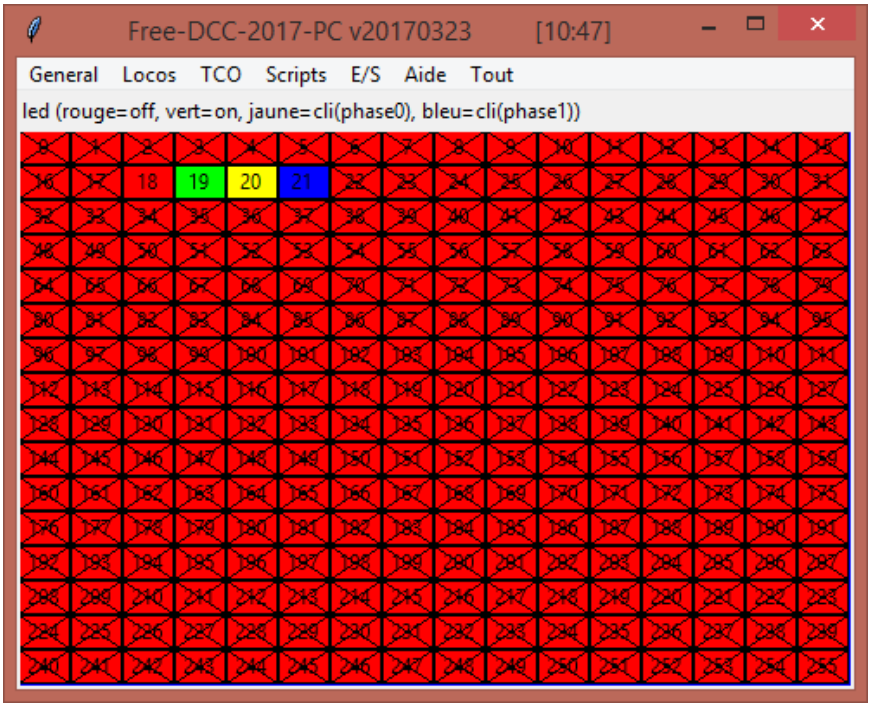
Pour les entrées, la couleur verte, indique que l'entrée est active. La couleur rouge indique une entrée inactive. L'écran des entrées permet de mémoriser les entrées actives en cochant la case « mem ». La mémorisation s'efface ensuite avec le bouton CLR. Cette fonctionnalité est utile si vous voulez tester des entrées à l'autre bout du réseau ou très fugitives. La capture suivante montre les 25 entrées de la centrale dans l'état actif :



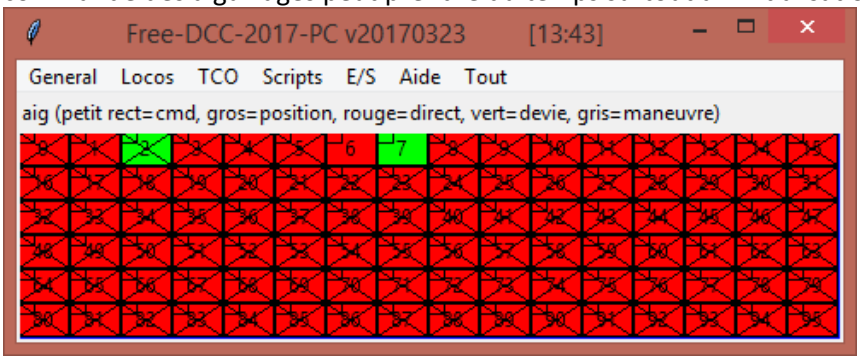
L'écran des sorties permet de voir l'état actuel des sorties. Une croix indique que la sortie est gérée par le système. Pour la tester, vous devez d'abord la mettre en manuel avec le bouton droit de la souris ce qui enlève la croix. Ensuite, vous pouvez la mettre à 0 ou 1 en cliquant avec la souris. Dans l'exemple suivant j'ai mis les entrées 0 à 3 en manuel et mise la 0 et 3 à 1. Les sorties 4 à 63 sont gérés par le système et sont ici toutes à 0.



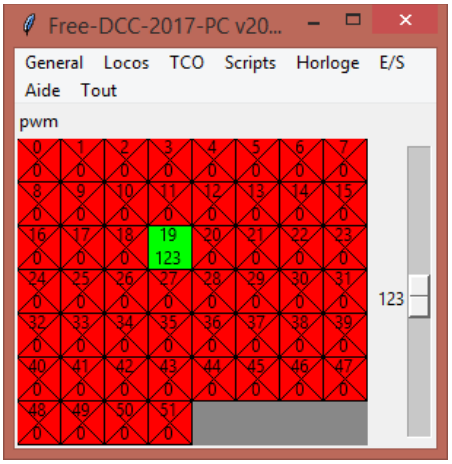
L'écran des leds permet de tester les leds. Rouge=OFF, Vert=ON, Jaune=clignotement, Bleu=clignotement inversé.



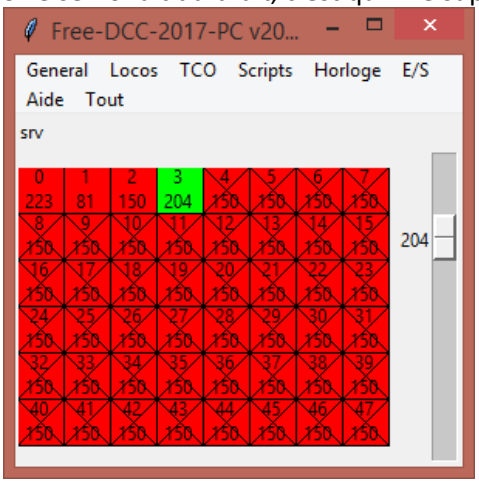
L'écran des aiguillages permet de jouer avec les aiguillages. Le petit carré représente la commande et le grand la position. ROUGE=position direct, VERT=position déviée, gris=position inconnu (ex en cours de manœuvre). La commande des aiguillages peut prendre du temps surtout à l'initialisation.



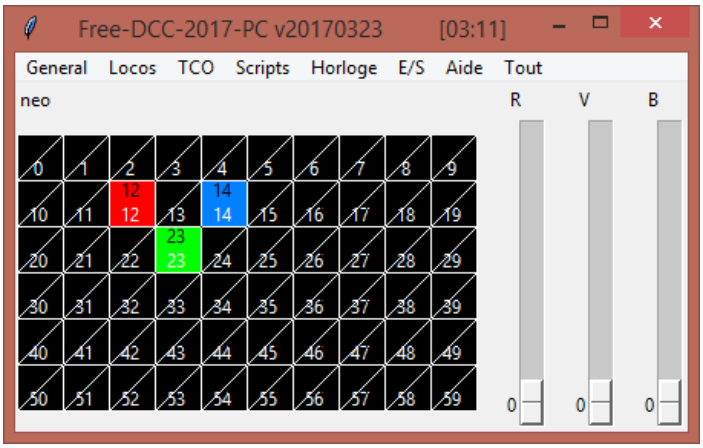
L'écran PWM permet de tester les sorties PWM.
PWM48 et 49 sont disponibles sur l'Arduino des centrales lorsque le bus I2C n'est pas utilisé
PWM50 et 51 sont disponibles sur l'Arduino de la centrale A uniquement
PWM0-47 sont disponibles sur les modules I2C à bas de PCA8596. Il faut éditer le code de la centrale pour choisir entre les modes PWM ou Servomoteur pour chaque sortie des PCA8596.
Utilisez le bouton droit de la souris pour enlever la croix d'une sortie PWM afin de la gérer manuellement.
Sélectionnez une des sorties (elle devient verte). Faites-la varier avec le potentiomètre.
0 = 0%
255 = 100%.

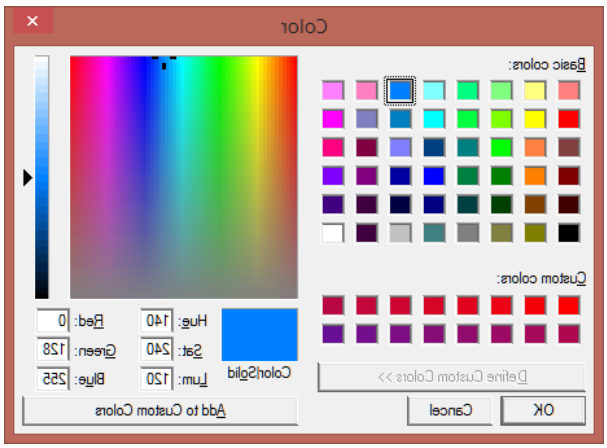


L’écran SRV permet de tester les Servomoteurs
SRV0-47 sont disponibles sur les modules I2C à bas de PCA8596. Il faut éditer le code de la centrale pour choisir entre les modes PWM ou Servomoteur pour chaque sortie des PCA8596. La vitesse de rotation du servo est aussi à configurer dans le code de la centrale.
Utilisez le bouton droit de la souris pour enlever la croix d’une sortie SRV afin de la gérer manuellement.
Sélectionnez une des sorties (elle devient verte). Faites-la varier avec le potentiomètre.
50 = impulsions de 500us = 0.5ms (servo à fond d’un côté)
150 = 1500us = 1.5ms (servo au milieu)
255 = 2550us =2.5ms (servo à fond de l’autre côté)
Si le servo fait du bruit, c’est qu’il ne supporte pas la position demandée. Il faut alors revenir vers le point médian.

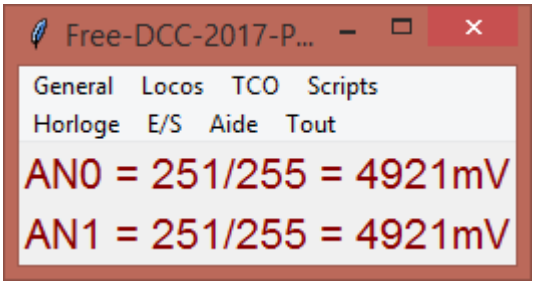


L’écran NEO permet de tester les néopixels.
Utilisez le bouton droit de la souris pour enlever la croix d’une led afin de la gérer manuellement.
Cliquez sur une des led pour ouvrir la fenêtre de sélection des couleurs.
Choisissez votre couleur
Fermer la fenêtre
Le néopixel est mis à jour !
Les potentiomètres RVB ne sont pas encore gérés.





L’écran ANA permet d’afficher la valeur des entrées analogiques ana0 et ana1



E. Les équations

Le fichier config_eq.txt permet de définir des équations. C'est-à-dire, calculer l'état des variables en fonction de l'état d'autres variables. Vous pouvez par exemple l'utiliser pour la signalisation. Les équations sont exécutées à chaque boucle du programme, c'est-à-dire toute les 250ms.

```
#
# fichier des équations
#
# operations
#  l <var>  / ln <var> : lecture       acc = var          / acc = ~var
#  a <var>  / an <var> : ET            acc = acc ET   var / acc = acc ET   ~var
#  o <var>  / on <var> : OU            acc = acc OU   var / acc = acc OU   ~var
#  x <var>  / xn <var> : OU EXCLUSIF   acc = acc OUEX var / acc = acc OUEX ~var
#  = <var>  / =n <var> : affectation   var = acc          / var = ~acc
#  =s <var> / =c <var> : mise a 1      si(acc==1) var=1   / mise a 0 si(acc==1) var=0
#
# variables binaires (0/1): (L=Lecture seule)
#  i<0-255> L: etat courant de l'entree
#  y<0-255> L: ancien  etat de l'entree
#  j<0-99>  L: etat d'occupation du canton
#  o<0-63>   : sortie
#  l<0-255>  : led
#  c<0-255>  : clignotement de led
#  p<0-255>  : phase de led
#  a<0-95 >  : commande aiguillage
#  d<0-95 > L: position directe aiguillage
#  e<0-95 > L: position evitement aiguillage
#  t<0-999>  : itineraire cmd
#  k<0-999> L: itineraire ok
#  w<0-999> L: itineraire voyant
#  u<0-999> L: itineraire ouverture
#  r<0-95 >  : reservation d'aiguillage
#  z<0-199> L: reservation de zone
#  v<0-999> L: variable
#
# variables entières (0-255):
#  ana<0-1  > L: entrée analogique (0=0v, 255=5V)
#  pwm<0-51 >  : sortie pwm (0=0%, 255=100%)
#  srv<0-47 >  : servomoteur (50=0.5ms, 150=1.5ms, 250=2.5ms)
#  var<0-999>  : variable (les variables var et v ne sont pas communes)
# - pour un test, remplacer <var> par <var255>==<val> (ou <>, >=, <=, <, >)
# - pour une affectation, remplacer <var> par <var255>=val
#    (l'affectation est effectuée si acc==1)
#
# variables des couleurs (#000000-#FFFFFF):
#  neo<0-59>  : led neopixel
#
# ex:
# l i1 a ana1>=200 = o3 = srv4=28 =n srv4=0
#

l i1 = l1
ln i1 a i2 = l0
ln i1 an i2 = l2
```

l (pour Load) charge la valeur de la variable dans l'accumulateur (acc).

ln (pour Load Not) charge l'accumulateur (acc) avec la valeur inverse de la variable

a (pour AND) effectue un ET logique entre acc et la variable.
Si la variable et ACC sont à 1 alors ACC est mis à 1. (ou 0 dans le cas contraire).

o (pour OR) effectue un OU logique entre acc et la variable.
Si la variable ou ACC (ou les 2) sont à 1 alors ACC est mis à 1. (ou 0 dans le cas contraire).

x (pour XOR) effectue un OU exclusif entre acc et la variable.
Si seulement une des 2 valeurs est à 1 alors ACC est mis à 1. (ou 0 dans le cas contraire.)

an, on, xn effectuent un ET,OU,XOR avec l'inverse de la variable spécifiée.

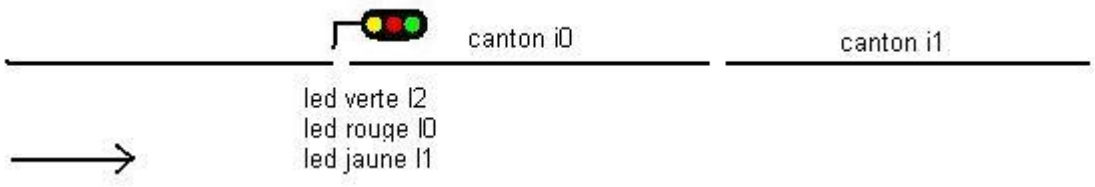
= met la valeur de ACC dans la variable

=S met la variable à 1 si ACC==1

=C met la variable à 0 si ACC==1

Exemple1 : Feu de block automatique lumineux.

Il s'agit dans cet exemple de piloter les 3 leds (verte, rouge et jaune) d'un feu de block. Il doit donc être rouge quand le canton précédent est occupé, jaune lorsqu'il est libre mais que le suivant est occupé et vert lorsque les 2 sont libres. Les trains sont détectés sur les différents cantons grâce à des détecteurs de courants reliés aux entrées in_0 à in_2. Bien entendu, les essieux des wagons doivent être résistifs afin d'être détectés.




```
l i0 = l0
ln i0 a i1 = l1
ln i0 an i1 = l2
```

Le feu est géré simplement à l'aide de ces 3 équations.
La première équation allume la led rouge si le canton de in_1 est occupé …
Le module signalisation peut maintenant aussi être utilisé pour gérer la signalisation parfois très complexe.

Exemple2: Réalisation d'un TCO avec des interrupteurs et leds.

Il s'agit de positionner l'aiguillage 5 en fonction de l'interrupteur connecté en in_10 et d'allumer la led 11 si un train est présent sur le canton détecté par un capteur de courant branché sur i11.

```
l i10 = a5
l i11 = l11
```

Attention, avec une telle commande d'aiguillage, vous ne pourrez plus commander l'aiguillage par le tco virtuel ou par le script car l'interrupteur mettra la commande de l'aiguillage à jour à chaque boucle du programme. Une solution est d'utiliser 2 boutons poussoir ou de ne faire l'action que si l'interrupteur change d'état.

```
ln y0 a i0 =s a5
l y0 an i0 =c a5
l i11 = l11
```

Il est également possible d'utiliser les variables ana, pwm, srv et var qui ne sont plus limitées à 2 valeurs (0 ou 1) mais à 256 valeurs (0-255).
Pour continuer à fonctionner avec le mécanisme L/A/O/X (n)/=/=s/=c/=n, il faut tester ces variables avec une valeur. Ainsi le résultat du test vaut 1 ou 0.
Ex : l **ana0>=200** = o1
(On ne laissera pas de blanc dans l'expression)
Pour les affectations, si acc vaut 1 alors l'affectation spécifiée est réalisée.
Ex : l i1 = **srv28=150** =n **srv28=250**
(On ne laissera pas de blanc dans l'expression)

Dans le cas des neopixels, les valeurs varient entre #000000 et #FFFFFF
Ex : l i1 = **neo3=#00FF00** =n **neo3=#FF0000**
(On ne laissera pas de blanc dans l'expression)
Le neopixel 3 s'allume au vert si i1 et actif. Il s'allume au rouge dans le cas contraire

F. L'horloge de modéliste

Cette horloge s'affiche en permanence dans la barre des titres. Le menu « horloge » permet de l'arrêter, la redémarrer et la remettre à l'heure de démarrage. Par défaut, l'horloge démarre à 6h00 et s'incrémente d'une minute toute les secondes réelles. Si vous voulez changer ces paramètres, vous pouvez modifier le code suivant :

```
#-------------------------------------------------------------------------------------------
# Horo
#-------------------------------------------------------------------------------------------

horo_on = 1 #activation de l'horloge
horo_j_dep = 0  #jour   0-6  au demarrage
horo_h_dep = 6  #heure  0-23 au demarrage
horo_m_dep = 0  #minute 0-59 au demarrage
# 4  = 1min /   sec      -> 1h / min
# 40 = 1min / 10sec      -> 1h / 10min
horo_tic_by_m = 4
```

Le jour n'est pas utilisé pour l'instant.

G. Les itinéraires

Un itinéraire est un ensemble d'aiguillages dans une position donnée. Par exemple l'aiguillage 0 en position dévié et les aiguillages 2 et 5 en position directe. Afin de coller un minimum à la réalité je me suis inspiré du site de Daniel Roverch : http://roverch.net/modelisme/Modelisme.html. Merci à l'auteur pour le partage !

Le logiciel propose les 2 types d'itinéraires suivant :

- L'itinéraire à « destruction automatique » (DA) : L'aiguilleur l'active et le train le désactive automatiquement lorsqu'il est passé par cet itinéraire.
- L'itinéraire à « tracé permanent » (TP) : L'aiguilleur doit l'activer et le désactiver manuellement. Ce mode est bien utile lorsque plusieurs trains doivent emprunter le même itinéraire.

Apparemment dans la réalité, la plupart des itinéraires sont en DA et certains sont doublés en TP.

Le fonctionnement du logiciel s'écarte un peu de la réalité pour s'adapter à la plupart des modélistes qui négligent souvent la détection en utilisant un nombre réduit de circuits de voie et en n'équipant pas leurs wagons d'essieux résistifs. En somme seule la locomotive est détecté et en peu d'endroits. Ce fonctionnement simpliste est aussi compatible avec l'utilisation des ILS. La première simplification est que ces itinéraires ne sont pas souples mais rigides, c'est-à-dire que les ressources de l'itinéraire ne sont pas libérées au fur et à mesure de l'avancement du train mais seulement lors de la fin de l'itinéraire. Le control d'approche n'est pas implémenté à la destruction des itinéraires. Si vous n'avez pas de détection, le mode DA ne peut bien entendu pas être utilisé.

Les itinéraires réservent les aiguillages. Un itinéraire utilisant les mêmes aiguillages devra donc attendre. En plus des aiguillages, il est aussi possible de réserver des zones z<0-199>. Cela est très utile pour éviter les cas de nez à nez et afficher les itinéraires sur le TCO.

Pour gérer signalisation, vous pourrez ensuite utiliser :

- Les équations pour commander les LEDs ou servomoteurs à partir des variables des itinéraires. Par exemple la variable w peut être utilisée pour fermer le carré protégeant l'itinéraire. D'autres variables comme les entrées ou cantons peuvent être utilisées pour gérer les autres états du feu.

- Ou récemment, plus simplement en utilisant le module de signalisation.

Les itinéraires peuvent aussi être utilisé en mode script. Voici comment utiliser les utiliser :

```
#en TP:
set t10+
wait k10
...
set t10-

#en DA
set t20+
wait k20
...
wait !t20
```

La configuration des itinéraires est décrite dans le fichier iti_config.txt :

```
# fichier des itineraire
#
# syntaxe: iti<0-999> tp <a0-a127><-,/> <z0-z199>
#          iti<0-999> da <a0-a127><-,/> <z0-z199> prox:<i0-255> in:<i0-255> fin:<i0-255> <temps_sec>
#
# ex: iti1 tp a1- a0/ a2- z2 z5
#     iti8 da a1- a4- a5- z3 z5 prox:i23 deb:i25 fin:i27 tlib:2s
#
# tp = "trace permanent"
#     il faut l'activer et le desactiver manuellement
#     si activation (var t<0-999> à 1):
#         - memorisation de la commande
#         - mise au clignotement de la variable "voyant" w<0-999>
#         - attente tant qu'un aiguillage ou zone de l'itineraire est reserve par un autre itineraire
#         - reservation de de tous les aiguillages et de toutes les zones de l'itineraire
#         - commande de positionement des aiguillages
#         - attente de positionement correcte des aiguillages
#         - mise a 1 de la variable "ouverture" u<0-999>
#         - mise a 1 de la variable "ok" k<0-999>
#         - mise a 1 de la variable "voyant"
#     si desactivation (var t<0-999> à 0):
#         - mise a 0 de la variable "ok"
#         - mise a 0 de la variable "ouverture"
#         - de-reservation des aiguillages et zones
#         - mise a 0 de la variable "voyant"
#
# da = "destruction automatique"  :
#     il faut l'activer manuellement,
#     il se desactive automatiquement lorsque le train arrive sur l'entree de fin
#     (Il peut aussi etre desactive manuellement a tout moment)
#     si activation (var t<0-999> à 1):
#         - memorisation de la commande
#         - mise au clignotement de la variable "voyant" w<0-999>
#         - attente tant qu'un aiguillage ou zone de l'itineraire est reserve par un autre itineraire.
#           attente egalement des entrees de debut et fin a 0.
#         - reservation de de tous les aiguillages et de toutes les zones de l'itineraire
#         - commande de positionement des aiguillages
#         - attente de positionement correcte des aiguillages
#         - attente de l'entree de proximite (si elle est presente)
#         - mise a 1 de la variable "ouverture" u<0-999>
#         - mise a 1 de la variable "ok"
#         - mise a 1 de la variable "voyant"
#         - attente de detection du train sur l'entree de debut (i25 dans l'exemple)
#         - mise a 0 de la variable "ouverture"
#         - attente de detection du train sur l'entree de fin (i27 dans l'exemple)
#         - attente durant temps_sec
#         - destruction de l'itineraire
#         - mise a 0 de la variable "ok"
#         - de-reservation des aiguillages et zones
#         - mise a 0 de la variable "voyant"
#         - mise a 0 de la variable d'activation t<0-999>
#     si desactivation (var t<0-999> à 0):
#         - destruction de l'itineraire
#         - mise a 0 de la variable "ok"
#         - de-reservation des aiguillages et zones
#         - mise a 0 de la variable "voyant"
#
# notes:
#     aiguillage en position directe  : -
#     aiguillage en position evitement: /
#     Les zones peuvent etre utilisees pour eviter les itineraires nez a nez,
#     les itineraires en cisaillements sans aiguillages.
#     Elles peuvent aussi être utilisées pour représenter les itineraires sur les TCO
#
# adaptation aux modelisme ou certains circuits de voie peuvent être absent:
#     Seul le premier vehicule a besoin d'etre detecte. Si vous utilisez le refoulement, une destruction
#     manuelle sera surement necessaire. Dans le cas de materiel reversible, il vaut mieux equiper la
#     remorque reversible d'une detection.
#     Si la detection de fin est placee trop tot, la tempo tlib peut etre utilisee pour laisser le temps
#     convoi degager la zone des aiguillages (bien entendu ceci est une entorse a la securite !)
#     Si il n'y a pas de detection de fin, alors la destruction devra se faire manuellement.
#         Dans ce cas quel est l'avantage du DA par rapport au TP ?
#         Reponse: C'est que le carre se ferme des l'entree, le train suivant devra donc attendre
#             l'etablissement d'un itineraire
#         Il est aussi possible d'utiliser la meme detection que celle d'entree et une grosse tempo
#             (ici aussi ceci est une entorse a la securite !)
#     L'entree prox est rarement utilisee, elle permet l'enclenchement de proximite, l'itineraire s'ouvre
#          seulement lorsque cette entree passe a 1. Cela permet d'economiser les signaux RAL et RRAL.
#
# non traites:
#    Passage de DA à TP et vice-versa
```

```
#    Enclenchement d'approche pour liberer les itineraires ouverts


iti0    tp a0- z0
iti1    tp a0/ z1
iti2    tp a3/ a2/ a1-
iti3    tp a3- a2/ a1-
iti4    tp a2- a1-
iti5    tp a1/

iti10   tp a0-
iti11   tp a0/
iti12   tp a3/ a2/ a1-
iti13   tp a3- a2/ a1-
iti14   tp a2- a1-  z0
iti15   tp a1/      z1

iti20   da a0- prox:v2 deb:v0 fin:v1 tlib:2s
```

Les itinéraires sont utilisables avec les variables suivantes :

t<0-999> : Cette variable permet d'activer un itinéraire. En mode TP, il faut la mettre à 1 pour activer l'itinéraire et à 0 pour le désactiver. En mode DA, il faut la mettre à 1 et le système la remet à 0 automatiquement en fin d'itinéraire. On peut utiliser des boutons sur le TCO pour contrôler cette variable. Il est aussi possible de l'utiliser en script. 0-999 est le numéro de l'itinéraire.

k<0-999> : Cette variable (en lecture seule) indique que l'itinéraire est formé et actif.

u<0-999> : Cette variable (en lecture seule) indique que le carré protégeant l'itinéraire peut être ouvert. En TP, elle est identique à la variable k. En DA, elle se met à 1 lorsque l'itinéraire est formé et remise à 0 lorsque le train est détecté dans la zone de début d'itinéraire « deb ».

w<0-999> : Cette variable (en lecture seule) peut être utilisée comme voyant d'un bouton de commande. Elle clignote lorsque l'itinéraire est enregistré et pas encore formé (t=1, k=0). Elle est allumée lorsque l'itinéraire est actif (k=1). Dans tous les autres cas (k=0), elle est éteinte.

En DA, les détections suivantes peuvent être utilisées :

deb : indique l'entrée de détection qui repère un train qui s'engage sur l'itinéraire. Cette variable est obligatoire en DA. Si vous ne pouvez pas faire cette détection, alors, il faudra utiliser le mode TP. Lorsqu'un train est détecté sur cette zone, alors le carré de protection doit se fermer afin que l'itinéraire ne puisse pas être réutilisé.

fin : indique l'entrée de détection qui repère l'arrivée du train en fin d'itinéraire. Cette indication est optionnelle, mais dans ce cas la destruction ne sera plus automatique. Cela limite l'intérêt pour le DA, mais il y a quand même un intérêt car le carré s'est refermé après l'entrée et les trains suivant devront attendre l'établissement d'un nouvel itinéraire. Dans le cas d'une zone de détection de fin mal placée qui détecterait les trains trop tôt, il est possible de rajouter une temporisation comme **t:10s** afin d'attendre que le train dégage la zone des aiguillages. Bien entendu ceci est une entorse à la sécurité !

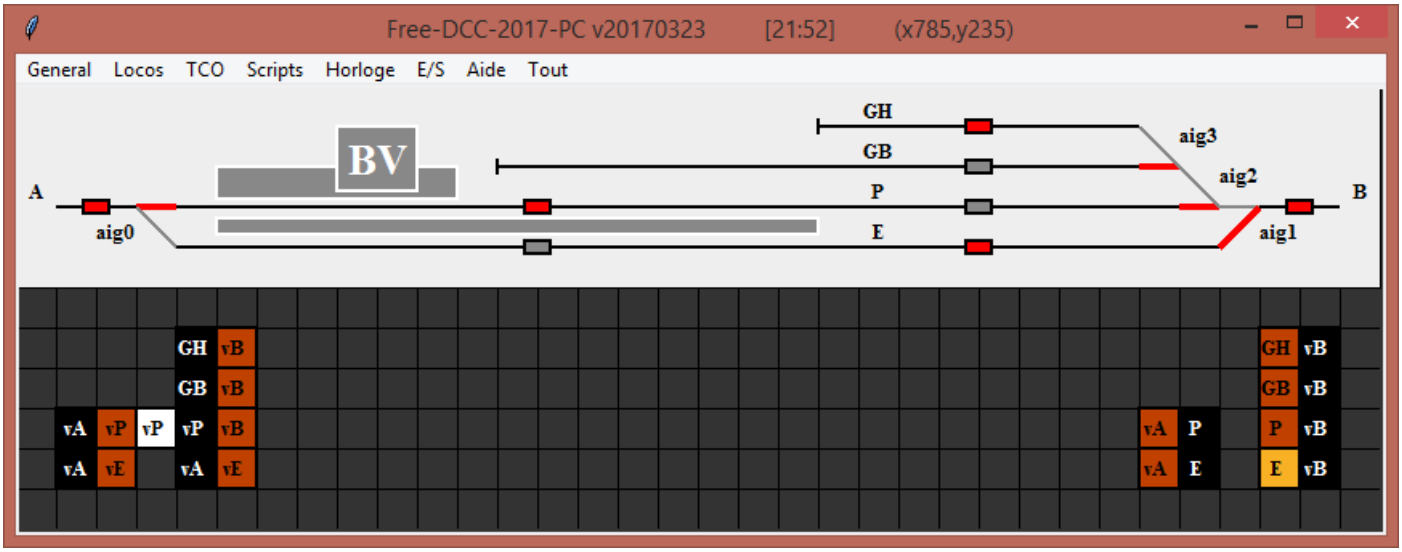
prox : indique l'entrée de détection utilisée pour l'enclenchement de proximité. Cette technique rarement utilisée permet de détecter le train juste avant le carré protégeant l'itinéraire et de n'ouvrir ce dernier qu'à ce moment-là. Cela permet d'économiser les signaux de ralentissement et rappel de ralentissement précédent normalement les aiguillages car le train arrivera normalement au pas devant le carré fermé.

Si une détection n'est pas spécifiée, alors elle n'est pas utilisée. Seule deb est obligatoire. Le système vérifie que deb et fin (si utilisée) sont libres avant d'activer l'itinéraire.

L'exemple suivant, montre comment réaliser des boutons/voyants utilisant les variables pour le TCO :

```
#boutons d'itineraires

#TP: A>P
rect 25,200,50,225 #000000 #000000  2
txtc  37,212 "vA" #FFFFFF 10
rect 50,200,75,225 #000000 #FFFFFF  2
txtc  62,212 "vP" #000000 10
color !w0 #C04000
color w0  #F8B025
action 50,200,75,225 t0^

#DA: A>P
rect 75,200,100,225 #000000 #FFFFFF  2
txtc  85,212 "vP" #000000 10
color !w20 #707070
color w20  #FFFFFF
action 75,200,100,225 t20^
```

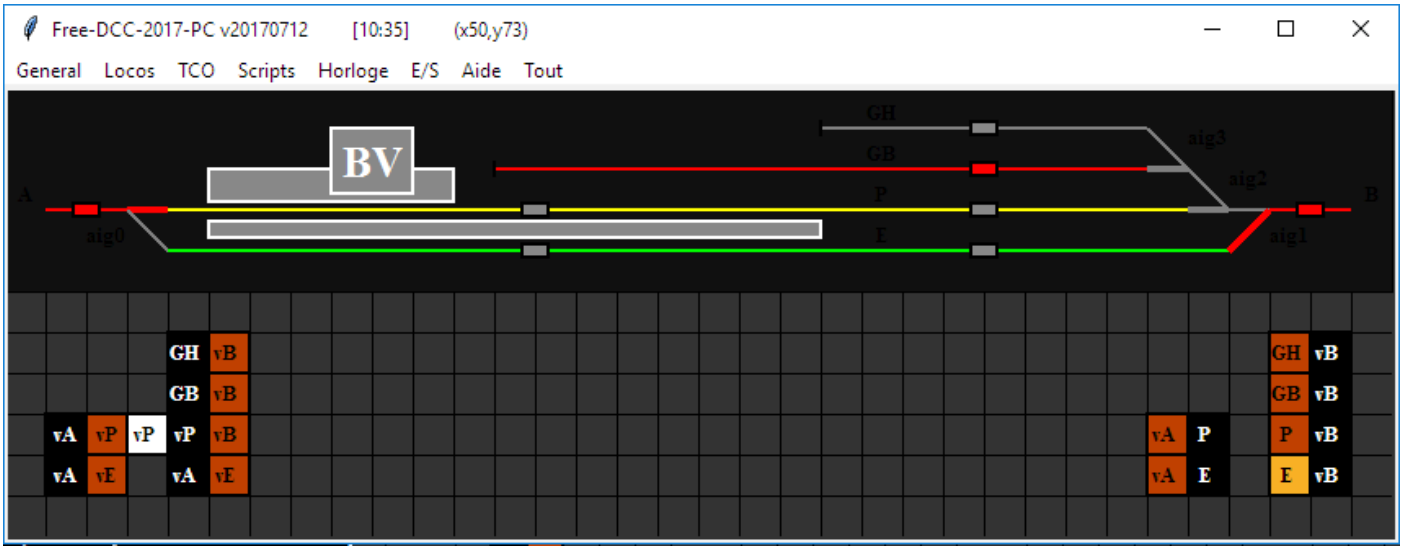
Ces quelques lignes permettent de réaliser les boutons TP et DA vA>vP. La capture suivante montre le résultat. On voit qu'il y a 12 itinéraires TP et un DA. (Ce qui est peu réaliste !). Le DA vA>vP et le TP vB>E sont actifs. Ce TCO est disponible en exemple dans l'archive. Il s'agit du TCO0.



Certains m'ont demandé s'ils pouvaient faire des TCO comme Daniel Roverch en changeant la couleur des voies :

- Gris par défaut.
- Rouge si un train est présent
- Jaune pour un itinéraire en DA
- Vert pour un itinéraire en TP

La réponse est oui comme le montre le TCO 3 de l'archive. Vous pouvez analyser le fichier config_tco.txt pour voire en détail comment cela est réalisé. J'ai conservé les rectangles de détection pour pouvoir simuler la présence des trains lorsque l'on clique dessus (en effet, en mode déconnecté, on peut modifier les entrées ce qui est bien pratique pour tester). J'ai aussi conservé la possibilité de commander les aiguillages lorsque l'on clique dessus comme pour le TCO0. Bien entendu cela ne marche que lorsque aucun itinéraire ne les utilise ! La position de chaque aiguille est matérialisée par un gros trait.



Pour nous simplifier la vie, nous utiliserons les zones. Nous en utiliserons une par segment de tco. On indiquera pour chaque itinéraire toutes les zones qu'il utilise. Nous utiliserons ensuite ces zones plus les détecteurs pour changer la couleur. J'ai choisi les zones suivantes

```
#        z12 z8
#        z13 z7 z6
# z4 z3 z0      z5 z9  z11
#    z2 z1         z10
```

Pour commencer, intéressons-nous uniquement aux TP. Nous colorerons uniquement les itinéraires en vert

Dans les zones d'aiguillage nous utiliserons l'algorithmes suivants :

- Si occupation (détecteur i6), peindre en rouge, sinon en vert
- Si pas de zone, peindre en gris

Par exemple pour le segment horizontal du 1er aiguillage associé à la zone z3 cela adonne :

```
line 75,75,100,75   #888888 2  #aig0 dir
color2 i6 #FF0000 #00FF00
color !z3 #808080
```

Hors zone d'aiguillage, nous utiliserons l'algorithme suivant :

- Si itinéraire peindre en vert, sinon peindre en gris
- Si occupation (détecteur i4 ou i2), peindre en rouge

Par exemple pour la voie entre le BV et le quai central associé à la zone z0

```
line 100,75,725,75  #000000 2
color2 z0 #00FF00 #808080
color  i4 #FF0000
color  i2 #FF0000
```

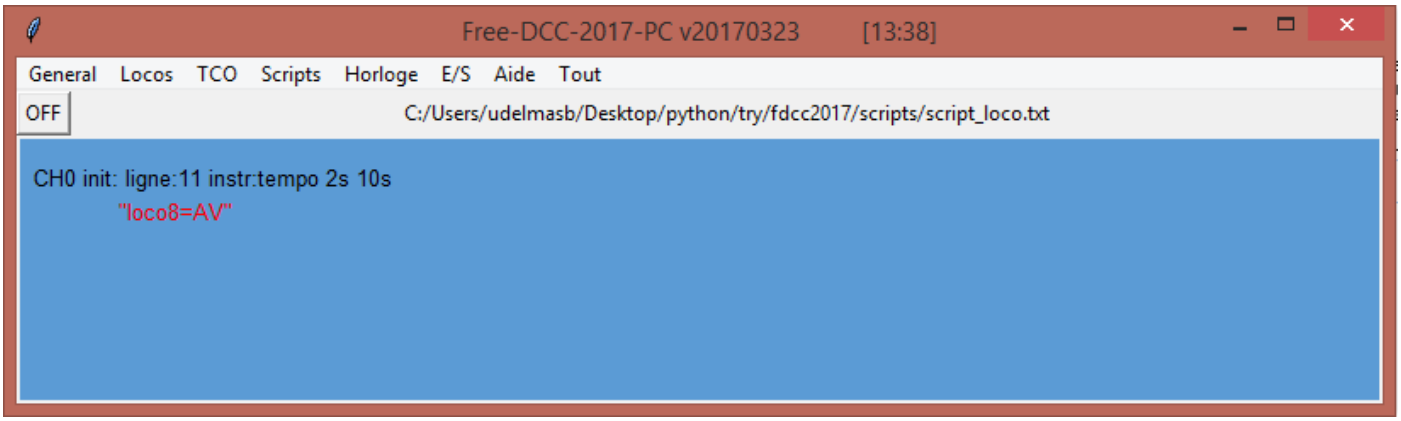
Pour les DA, le même principe s'applique sauf que nous les peindrons en jaune.

Tout se complique si un segment peut à la fois être sur un DA ou un TP car il faut pouvoir le peindre en jaune ou en vert. Dans ce cas je conseille d'utiliser la même zone pour les 2 types d'itinéraire et en ajouter une autre uniquement pour un des 2 types. Ainsi la zone commune pourra être utilisée pour peindre en gris lorsqu'elle n'est pas active et la zone spécifique pour différencier le vert du jaune ! Exemple précédent si j'ajoute z20 à z3 et z21 à z0, j'obtiens :

```
line 75,75,100,75   #888888 2   #aig0 dir
color2 z20 #FFFF00 #00FF00
color  i6  #FF0000
color  !z3 #808080

line 100,75,725,75  #000000 2
color2 z0  #00FF00 #808080
color  z21 #FFFF00
color  i4  #FF0000
color  i2  #FF0000
```

H. Les scripts

Les scripts permettent d'automatiser des séquences de jeu. Il est possible d'automatiser absolument tout. Le menu « Scripts -> Scripts » permet de charger un script pour l'exécuter. Un script est un fichier texte qui contient des programmes qui contiennent des instructions. L'écran des scripts indique ou en est le script. Un script peut avoir de multiples programmes s'exécutant en parallèle des autres dans un des canaux CH0-CH9. Cela permet de paralléliser les séquences. Par exemple vous pouvez définir des programmes pour déplacer les trains, un programme pour lancer ces déplacements en fonction de l'heure et un autre pour gérer l'éclairage des bâtiments. Bien entendu, il est possible de synchroniser les différents programmes avec les variables v. Le bouton OFF permet d'arrêter le script.



Voici la documentation complète que nous allons éclaircir plus loin avec quelques exemples.

```
# ----------------------------------------------------------------------------------------------------------------
# Syntaxe:
# ----------------------------------------------------------------------------------------------------------------
#
# ----commentaires-------------------------------------------------------------------------------------------------
#
# # commentaire
#
# ----programmes--------------------------------------------------------------------------------------------------
#
# <label>:                              // un label est un repere ds le programme, on peut y sauter
# instructions
#
#    init:                              // au demerrage, le programme commence a s'executer a partir du label
#                                       // init:
#    loco 2
#    bcl:                               // ce label est utilise par la boucle
#    vit +10
#    tempo 1s
#    vit +0
#    tempo 1s
#    goto bcl                           // on saute au label bcl:
#    ...
#
#    Le programme commence a s'executer automatiquement a partir du label init lors du chargement d'un script
#
# run <label>                           // cette instruction demande d'executer le programme indique sans
#                                       // attendre sa fin
#                                       // pour attendre sa fin avant de continuer, faire suivre de
#                                       // wait_end_run
#
#    run depl22:                        // lance le programme depl22 puis continue en //
#
# wait_fin <label>                      // cette instruction attend la fin du programme indique
#
#    wait_fin depl22                    // attend que le programme de debut depl22 se termine
#
# end                                   // fin du programme
#
# ----loco--------------------------------------------------------------------------------------------------------
#
# loco <adr loco>
#
#    loco 23              // selectionne la loco d'adresse 23 (si deja utilisee, attend qu'elle soit libre)
#    loco 0               // 0 libere la loco
#
```

```
# vit <+/-><0-31>
#
#    vit +31              // vitesse avant max (cran 31/31) pour la locomotive selectionnee
#    vit -0               // arret (cran 0/31), sens arriere
#
# fct f<0-4><+/->
#
#    fct f0+ f4-          // active la fonction 0 de la locomotive selectionnee (souvent les phares),
#                         // desactive la fonction 4
#
# ----aiguillages-----------------------------------------------------------------------------------------
#
# aig a<0-127><-,/>
#
#     aig a3/ a23- a2-    // positione a2 et a23 en direct et a3 en devie
#
# les reservations ne sont pas encore gerees
#
# ----itineraires-----------------------------------------------------------------------------------------
#
# les itineraires s'utilisent comme des es avec la variable t (de iTineraire)
# seul les itineraires directs sont geres, il n'y a pas de reservation, d'ou risque de pb
#
# set t<0-999><+/->
# wait [!]t<0-999>
#
# set t31+  // demande l'etablissement de l'itineraire 31
# wait t31  // attend l'itineraire 31
#
# ----es-variables----------------------------------------------------------------------------------------
#
# wait [!]i<0-255> [!]d<0-127> [!]e<0-127> [!]v<0-999>  // attend que toutes les conditions soient vraies
#      ana<0-1><==...><0-255> var<0-999><==...><0-255>
#
#     wait i1 !i2 v1      // attend i1 actif, i2 inactif, v1 active
#     wait an0>=127 a var3<>22 a i1
#
# set o<0-63><+/-> l<0-255><+/-> c<0-255><+/-> p<0-255><+/-> v<0-999><+/->  // positione les variables
#     pwm<0-51>=<0-255>  srv<0-47>=<50-255> neo<0-59>=#RRVVBB
#
#     set o2+ l3- l4+ l5+ c4+ c5+ p4- p5+ // active out2, eteint led3, fait clignoter led4 et 5 en alternance
#     set pwm3=45 neo28=#FF0000 srv28=124 o5+   //met pwm3 à 45(45/255=18%), le servo28 à 124 (1.24ms),
#                                               //le neopixel 8 au rouge à fond, active la sortie 5
#
# ----temps-----------------------------------------------------------------------------------------------
#
# tempo <temps><t,s> ...  // attend pendant une duree
#
#     tempo 1s            // attend une seconde
#     tempo 2t            // attend 2 tics (un tic=250ms, 2=500ms ...)
#     tempo 1s 10s        // attend aleatoirement entre 1 et 10 secondes
#
# waithm <hh:mm>          // attend l'heure specifiee (sur l'horloge de modelisme)
#
#     waithm 22:30        // attend 22h30
#
# sethm <hh:mm>           // regle l'heure de l"horloge de modelisme
#
#     sethm 06:00         // regle l'heure a 6h
#
# ----sauts-----------------------------------------------------------------------------------------------
#
# goto <label:>           // saute au label indique (ne pas oublie les :)
#
#     la_boucle:
#         ...
#         goto la_boucle:
#
# goto <label:> <pourcentage%>  // saute au label indique aleatoirement suivant le pourcentage sinon continue
#
#     la_boucle:
#         ...
#         goto la_boucle: 80%
#         ...
#         goto la_boucle:
#
# ----communication-utilisateur---------------------------------------------------------------------------
#
# msg "message"           // affiche un message
#
#    msg "deplacement_1"  // affiche ce message ds le champ message du programme dans la fenetre des scripts
#                         // il ne doit pas y avoir de blanc ds le message
#                         // la chaine ne doit jamais etre vide
#
```

```
# nomsg                  // efface le message
#
# sound "fichier.wav"    // joue un son (le script continue sans attendre la fin du son)
#
#    sound "sound/annonce_depart_25.wav"
```

Exemple1 : Ce script allume la led 20 pendant 10 secondes puis la 21 pendant 1.25 secondes :

```
init:
set l20+
tempo 10s
set l20- l21+
tempo 5t
set l21-
end
```

Tout script contient au moins un programme qui commence au label init. Ce programme contient des instructions. Il se termine comme tous les programmes par l'instruction end. Les instructions sont alors exécutées par le canal 0.
Les instructions set permettent de changer l'état des variables des leds 20 et 21 avec + pour mise à 1 donc allumage et ' pour 0 et donc extinction.
Les attentes sont réalisées avec les instructions tempo. 10s pour 10 secondes et 5t pour 5 tics, soit 1.25s car un tic vaut 250ms.

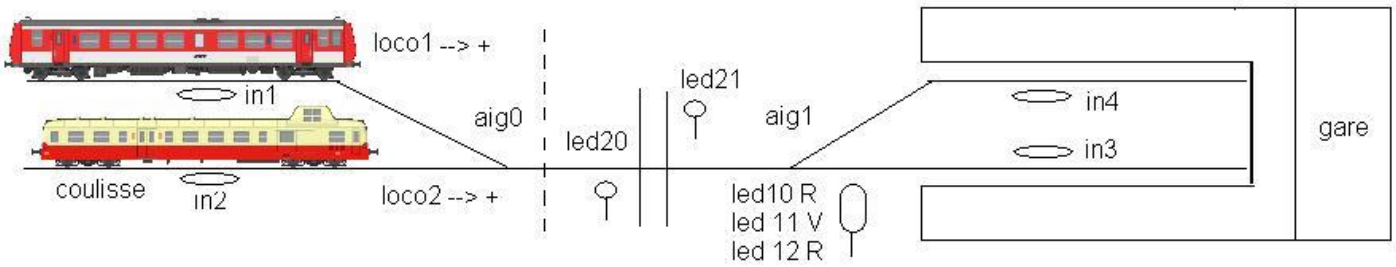
Exemple2 : Ce script déplace la loco d'adresse 8 en avant puis en arrière avec des temps variables.

```
init:
loco 8
fct f0+
tempo 1s
msg "loco8=AV"
vit +10
tempo 2s 10s
msg "loco8=STOP"
vit +0
tempo 2s 10s
msg "loco8=AR"
vit -10
tempo 2s 10s
msg "loco8=STOP"
vit +0
tempo 2s 10s
fct f0-
loco 0
nomsg
end
```

Ici aussi, nous n'utilisons qu'un seul programme. Le programme qui commence au label init.
La loco 8 est sélectionnée au début du script et désélectionnée à la fin
Après la sélection, les instructions vit et fct permettent de changer le sens la vitesse et les fonctions spéciales de la loco sélectionnée.
L'instruction tempo possède maintenant 2 temps. Le temps utilisé sera choisi au hasard entre ces 2 valeurs. Cela ajoute une petite part de hasard qui rend plus amusante l'automatisation car on ne sait pas combien de temps vont durer les attentes.
L'instruction msg permet d'afficher un message dans la fenêtre des scripts. Celui-ci s'affichera pour le programme qui commence à init : et qui est exécuté par le canal 0.
nomsg efface ce message

Exemples 3 :

Pour aller un peu plus loin, nous allons maintenant utiliser le petit réseau suivant qui dispose d'une coulisse à 2 voies, une gare terminus également à 2 voies. 2 autorails desservent cette gare. Ils sont repérés par des ILS. Un passage à niveau et un feu sont également présents.



Exemple 3.1 :

Dans cet exemple le Picasso (loco2) va desservir la gare puis revenir en coulisse

```
init:
aig a0- a1-
loco 2
vit +10
wait i3
vit +0
tempo 20s
vit -8
tempo 2s
vit -10
wait i2
vit +0
loco 0
end
```

Exemple 3.2 :

Un peu plus complexe, l'autorail dessert 2 fois la gare.
Nous allons utiliser les 3 programmes suivants :

- loco2_c_gare : pour déplacer le picasso de la coulisse vers la gare
- loco2_gare_c : pour déplacer le picasso de la gare vers la coulisse
- init : qui enchainera les déplacements

```
#sequencement
init:
run loco2_c_gare
wait_fin loco2_c_gare
tempo 20s
run loco2_gare_c
wait_fin loco2_gare_c
tempos 30s 120s
run loco2_c_gare
wait_fin loco2_c_gare
tempo 20s
run loco2_gare_c
wait_fin loco2_gare_c
end

#picasso coulisse 2 > gare 3
loco2_c_gare:
loco 2
aig a0- a1-
vit +10
```

```
wait i3
vit +0
loco 0
end

#picasso gare 3 > coulisse 2
loco2_gare_c:
loco 2
aig a0- a1-
vit -8
tempo 2s
vit -10
wait i2
vit +0
loco 0
end
```

Exemple 3.3 :

Rajoutons l'horloge de modélisme et des annonces sonores stockées dans le répertoire « sound ».

```
#sequencement
init:
sethm 05:55

waithm 6:00
run loco2_c_gare
wait_fin loco2_c_gare

waithm 6:15
run loco2_gare_c
wait_fin loco2_gare_c

waithm 7:00
run loco2_c_gare
wait_fin loco2_c_gare

waithm 7:15
run loco2_gare_c
wait_fin loco2_gare_c

end


#picasso coulisse 2 > gare 3
loco2_c_gare:
loco 2
aig a0- a1-
vit +10
sound "sound/arrive_picasso.wav"
wait i3
vit +0
loco 0
end

#picasso gare 3 > coulisse 2
loco2_gare_c:
sound "sound/depart_picasso.wav"
tempo 2s
loco 2
aig a0- a1-
vit -8
tempo 2s
vit -10
wait i2
vit +0
```

```
loco 0
end
```

Exemple 3.4

Rajoutons la gestion des leds du feu et du passage à niveau. Et répétons le scenario en boucle.

```
#sequencement
init:
set c20+ c21+ p21+ l20- l21-
set l10+ l11- l12+
bcl:
sethm 5:55
waithm 6:00
run loco2_c_gare
wait_fin loco2_c_gare
waithm 6:15
run loco2_gare_c
wait_fin loco2_gare_c
waithm 7:00
run loco2_c_gare
wait_fin loco2_c_gare
waithm 7:15
run loco2_gare_c
wait_fin loco2_gare_c
waithm 8:00
goto bcl:
end

#picasso coulisse 2 > gare 3
loco2_c_gare:
loco 2
aig a0- a1-
set l20+ l21+
vit +10
wait i3
vit +0
set l20- l21-
loco 0
end

#picasso gare 3 > coulisse 2
loco2_gare_c:
loco 2
aig a0- a1-
set l10- l11+ l12- l20+ l21+
vit -8
tempo 1s
set l10+ l11- l12+
tempo 1s
vit -10
wait i2
vit +0
set l20- l21-
loco 0
end
```

Bien entendu la signalisation gagne à être gérée par les équations en fonction de l'occupation des voies. Mais certaines fois il est préférable de la simuler comme ici.

Exemple 3.5 :

Ajoutons une desserte avec l'autorail X2200. Afin de faciliter la compréhension, la signalisation a été enlevée.

```
#sequencement
prog 0
sethm 5:55
waithm 6:00
run_wait_prog 1
run_wait_prog 3
waithm 6:15
run_wait_prog 2
run_wait_prog4
waithm 7:00
run_wait_prog 1
waithm 7:15
run_wait_prog 2

#picasso coulisse 2 > gare 3
loco2_c_gare:
loco 2
aig a0- a1-
vit +10
wait i3
vit +0
loco 0
end

#picasso gare 3 > coulisse 2
loco2_gare_c:
loco 2
aig a0- a1-
vit -10
wait i2
vit +0
loco 0
end

#X2200 coulisse 1 > gare 4
loco1_c_gare:
loco 1
aig a0/ a1/
vit +10
wait i4
vit +0
loco 0
end

#X2200 gare 4 > coulisse 1
loco1_gare_c:
loco 1
aig a0/ a1/
vit -10
wait i1
vit -0
loco 0
end
```

Ces quelques exemples sont volontairement simples afin de comprendre rapidement le fonctionnement du système, maintenant à vous de jouer en imaginant des scénarios aussi complexes et réalistes que possible. La détection des trains peut également être effectuée par des détecteurs de courant ou des pédales de voies. Les scripts ne sont pas limités aux déplacements des trains, ils peuvent par exemple aussi gérer des animations lumineuses …

I. Le suivi des trains

Motivations :

Le suivi des trains est une fonction du logiciel qui permet de suivre automatiquement les trains sur les cantons du réseau. Le logiciel va essayer de déduire sur quel canton se trouve quel train en fonction des détections.
Vous n'êtes pas obligé d'utiliser cette fonctionnalité et vous pouvez déjà faire beaucoup avec tout ce qui précède.
Les petits réseaux ou ceux n'utilisant pas le cantonnement s'en passeront aisément.
Ce dispositif nécessite bien entendu le cantonnement du réseau ou il ne doit y avoir théoriquement qu'un seul train au maximum par canton.

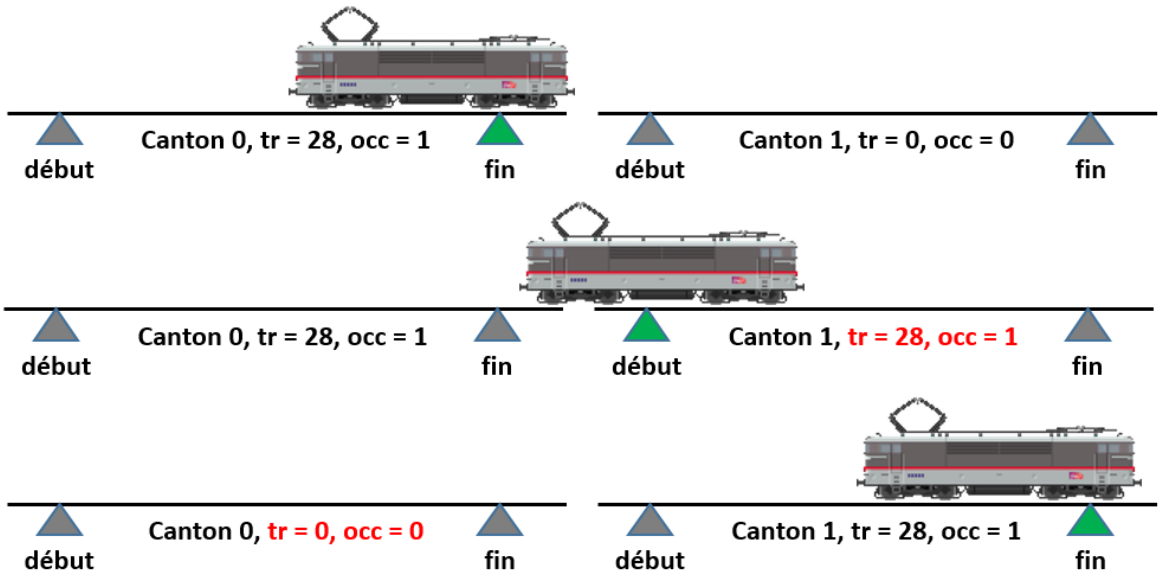
Le suivi des trains permet :

- L'Implémentation d'une signalisation fonctionnelle pour ceux qui n'ont pas d'essieux résistifs sur leurs wagons.
- Le suivit des trains sur le TCO
- La conduite automatique des trains en mode cantonnement comme le faisaient de nombreux modélistes en analogique.

J'avoue avoir créé ce dispositif car Je n'étais pas très enclin à mettre des essieux résistifs sur tous mes wagons, les puristes m'en excuseront. De nombreux modélistes étant aussi dans ce cas, il fallait faire quelque chose car sans cela, impossible d'avoir une signalisation fonctionnelle sans parler de la sécurité ! Par exemple un canton (ou une zone) avec des wagons non résistifs est déclarée libre avec un joli feu au vert … Il est à noter que Le système fonctionne sans problème avec les essieux résistifs, il évitera même le clignotement de la signalisation suite à des problèmes de captage de courant.

Fonctionnement :

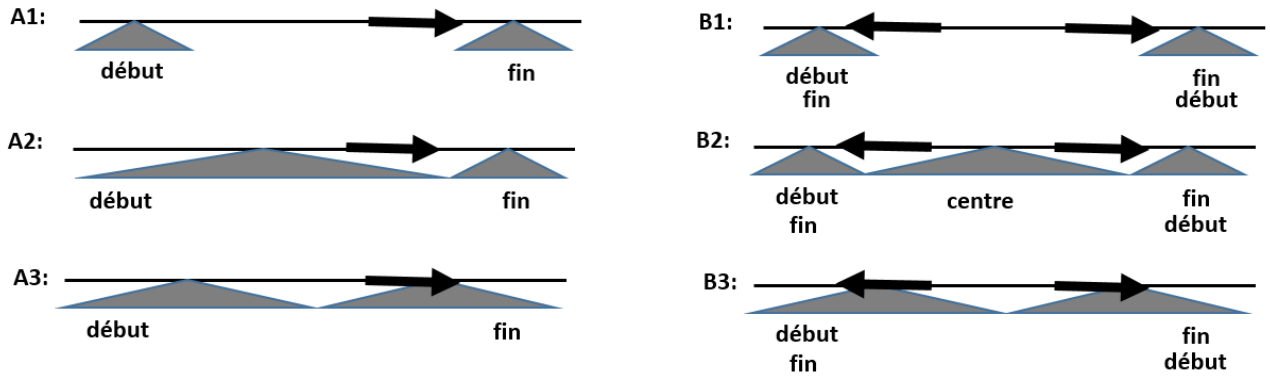
Chaque canton possède un capteur d'entrée (ILS, pédale de voie ou capteur de courant) et un capteur de fin.
Lorsque le capteur d'entrée est actif, le canton est déclaré occupé et le logiciel recherche la locomotive qui y est entrée en fonction de la position des aiguillages et du fichier config_cantons.txt qui donne le lien entre les cantons en fonction de la position des aiguillages.
Lorsque le capteur de fin est actif, tout autre canton avec la locomotive détectée est libéré.
Les variable d'occupation des cantons j<0-99> sont mise à jour et peuvent être utilisées par la signalisation.
Ce fonctionnement n'est pas 100% conforme à la réalité car le canton précédent n'est pas libéré immédiatement lorsque le train le quitte mais lorsqu'il arrive en fin du canton courant, mais ce n'est pas bien grave.



En fonctionnement automatique, Il est conseillé que le capteur de fin détecte la loco au moins 30cm avant la fin du canton. Le capteur de début doit quant à lui détecter la locomotive au plus tôt. Si le canton est parcourable dans les

2 sens, alors le capteur de fin peut être utilisé comme capteur de début et inversement. Le premier véhicule doit pouvoir être détecté, modifier en conséquence vos rames réversibles et autorails (aimant, essieux résistif).
Les cantons peuvent être séparés par des aiguillages ou zones d'aiguillages. A vous de voir si vous ajouter une détection sur ces aiguillages ou utilisez celles des cantons

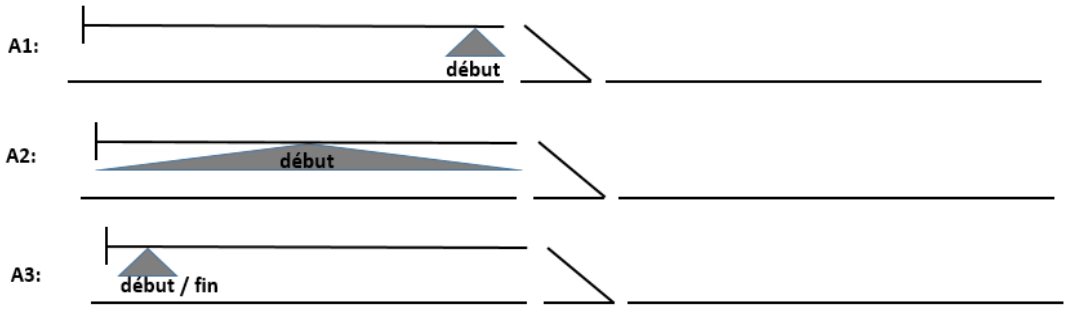
Si des capteurs de courant sont utilisés, vous pouvez les positionner comme suit :



Le cas A1 et B1 sont équivalent aux solutions à ILS ou pédale de voie. Par contre ils imposent de tronçonner chaque canton en 3 parties. Le cas A2 réduit le tronçonnement à 2 parties. Les cas A2 et B2 peuvent être utilisés par les puristes qui utilisent des essieux résistifs sur tous leurs véhicules. Le capteur centre n'est pas utile pour le cantonnement. Les cas A3 et B3 permettent de ne tronçonner le canton qu'en 2 parties mais risquent de détecter un train long avant qu'il n'ait libéré le canton précédent. Comme on le verra par la suite, une temporisation peut être utilisée avant de déclencher la vraie fin. Sauf besoin particulier, utilisez A1, A2 et B1.

Voies de garage :

Il est possible d'utiliser 2 détecteurs pour les voies de garage comme précédemment, mais il est également possible d'utiliser un seul détecteur comme l'indique la figure suivante :



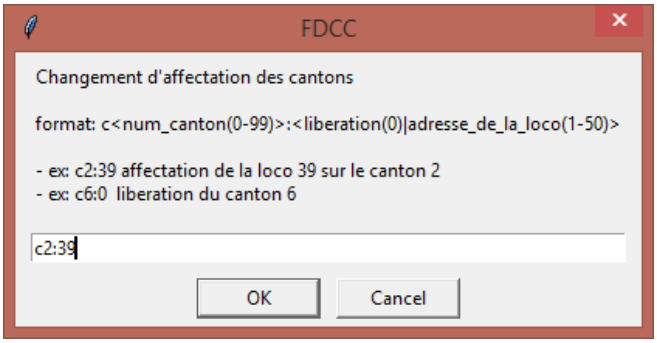
Dans les 2 premiers cas, il n'y a pas de capteur de fin. A l'activation du capteur d'entrée, une temporisation sera lancée et simulera une fin après un temps écoulé. Il est bien entendu impératif que le train en entier soit sur la voie de garage avant libération du canton précédent !

Dans le dernier cas que l'on emploiera surtout en coulisse, le capteur de début et de fin est le même. Lorsqu'il est actif, le canton est aussi tôt occupé et le canton précédent libéré. Dans ce cas le canton n'est pas mis à occupé lors de l'entrée car le capteur est à la fin, mais ce n'est pas très grave pour une voie de garage.

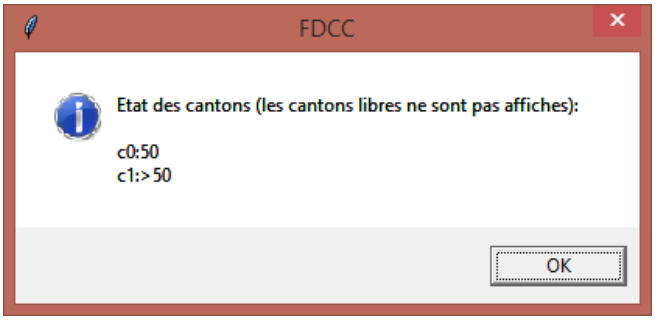
Menus :

Le menu « Locos > Affecter un canton » permet d'affecter une locomotive sur un canton ou de libérer un canton.

Il est parfaitement envisageable d'avoir toute une zone non couverte par le cantonnement comme les voies d'un dépôt. Dans ce cas ce menu devra être utilisé pour enlever chaque locomotive du canton donnant accès au dépôt lorsqu'elle y rentre. Inversement, il faudra affecter au canton en sortie du dépôt la locomotive qui y sort.



Le menu « Loco > Etat des cantons » permet de voir l'état des cantons. (Dans capture suivante, la loco 50 entre sur le canton c1 mais n'a pas encore libéré le canton c0)



Si une erreur survient. Par exemple le logiciel n'arrive pas à trouver la locomotive qui entre sur un canton, alors ce canton est déclaré en erreur et occupé. Il faudra alors utiliser le menu précédent pour régler le problème.

Fichier de configuration

Le fichier de configuration indique pour chaque canton les capteurs d'entrée et de fin, la temporisation et donne le lien entre les cantons en fonction de la position des aiguillages afin que le logiciel déduise le déplacement des trains. L'entête du fichier explique sa syntaxe :

```
# config_cantons.txt:
#
#   Ce fichier decrit les cantons afin de suivre les trains et gerer l'occupation des cantons
#
#   Syntax:
#   c<0-99>:     canton destination (avec :)
#   c<0-99>      canton source (sans :)
#   a<0-_95><-/>  position d'un aiguillage
#   i<0-_255>    detecteurs
#                 - le permier indique le detecteur d'entree
#                 - le second  indique le detecteur de fin
#                 s'il n'y en a qu'un seul, il sert de début et de fin
#   t<0-59>s     tempo qui retarde la fin apres l'activation du detecteur de fin s'il existe
#                ou qui simule une fin apres l'activation du capteur d'entrée
#   l<1-50>      assigner la loco au demarrage

# exemple d'une boucle a 3 cantons a un seul sens
# c0: c2 i10 i0 l50
# c1: c0 i11 i1
# c2: c1 i12 i2

c0: c2 i0 l50
c1: c0 i11 i1 t10s
c2: c1 i12 i2
```

L'exemple commenté gère une boucle à 3 cantons mono-sens avec 2 capteurs par canton. L'exemple non commenté aussi mais avec un seul capteur de fin pour le canton 0 et une temporisation pour le canton 1. Vous pouvez tester cet exemple dans le tco 4. Bien entendu pour votre propre réseau, il faudra enlever cet exemple.

Affichage des trains sur le TCO

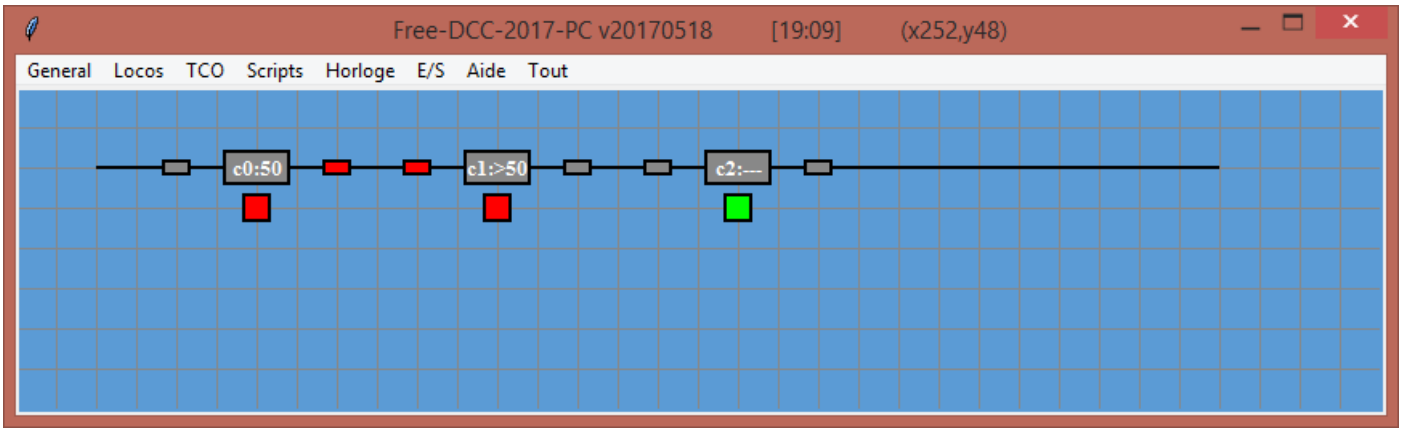
La commande suivant peut être utilisée pour afficher l'adresse de la loco présente sur un canton.

```
# cantontxtc2 canton color font_size
```

Il est conseillé d'afficher cette ligne sur un rectangle pour simuler un afficheur.
Les instructions suivantes affichent les capteurs i12 et i2 du canton 2, le numéro de la loco et un rectangle suivant l'occupation. Les actions permettent de changer l'etat des capteurs en mode déconnecté pour tester.

```
rectc 400,50,8,4     #000000 #FF0000  2
color2 i12 #FF0000 #888888
action2 i12^
rectc 500,50,8,4     #000000 #FF0000  2
color2 i2 #FF0000 #888888
action2 i2^
rectc 450,50,20,10  #000000 #888888 2
cantontxtc2 2 #FFFFFF 10
rectc 450,75,8,8     #000000 #FF0000  2
color2 j2 #FF0000 #00FF00
```

Voici un exemple d'affichage sur le TCO. Les lignes ci-dessus s'occupent du canton 0 :



J. La signalisation

Introduction :

La signalisation peut être gérée avec les équations. Ce mode s'approche de la réalité car les relais utilisés dans la réalité peuvent être décrit par des équations logiques. Néanmoins, avec de nombreux itinéraires, les équations deviennent vite un peu trop complexes. Ce module de signalisation a été rajouté pour gérer la signalisation plus simplement. Il de configure en utilisant le fichier config_signalisation.txt.

Chaque nouveau signal se définit par s<0-199>: (ne pas oublier les :)

Pour le système, chaque signal doit être dans un des états suivants :

Etat	
C = Carre	Franchissement interdit (aucun itinéraire ou aiguillages mal positionnes ou commutateur de fermeture ou BMVU occupé)
S = Sémaphore	Canton suivant occupé
A = Avertissement	Le signal suivant est C ou S
L = voie Libre	Le signal suivant n'est pas C ou S
M = manœuvre	Manoeuvre autorisée

Pour que le système définisse dans quel état sont les feux, il faut indiquer pour chaque « chemin » possible :
- l'itinéraire ou la position des aiguillages,
- les détecteurs ou le canton
- le signal suivant ou le BMVU

Pour un heurtoir, remplacer le signal suivant par stop
Si le signal, ne gère pas l'avertissement, ex S de BAPR, omettre le signal suivant
S'il n'y a aucun itinéraire pour entrer sur le canton protégé par le feu, mettre itix

Il faudra ensuite définir pour chaque état C/S/A/L/M, l'état de toutes les LEDs en signalisation lumineuse ou/et la position des servomoteurs en signalisation mécanique.
Si vous souhaitez juste spécifier les états actifs des LEDs, alors rajouter l'état D (défaut) en premier pour éteindre toutes les LEDs avant d'activer uniquement les actives dans les autres états. Les signaux peuvent être réels (implantés sur le réseau) ou virtuels (non présents). Les signaux peuvent servir à la conduite automatique des trains en mode cantonnement qui franchiront les signaux en L ou A et s'arrêteront devant tout autre état.

Format du fichier :

```
# fichier de configuration de la signalisation
#
# Il est possible de gerer la signalisation avec les equations mais cela peut vite devenir
# fastidieux. Ce fichier permet de gerer des signalisations complexes facilement
#
# Chaque nouveau signal se definit par s<0-199>:  (ne pas oublier les :)
# Pour le systeme, chaque signal doit être dans un des etats suivants:
# - C = Carre         = franchissement interdit (aucun itineraire ou aiguillages mal positiones
#                                                ou commutateur de fermeture ou bmvu occupe)
# - S = Semaphore     = canton suivant occupe(indique par les entrees i ou l'etat des cantons j)
# - A = Avertissement = le signal suivant est C ou S
# - L = voie Libre    = le signal suivant n'est pas C ou S
# - M = Manoeuvre     = un itinéraire de manœuvre est actif
#
# Il faut donc indiquer pour chaque "chemin", l'itineraire ou la position des aiguillages,
# les detecteurs ou le canton et le signal suivant ou le bmvu
# Pour un heurtoir, remplacer le signal suivant par stop
# Si le signal, ne gere pas l'avertissement, ex S de BAPR, omettre le signal suivant
# S'il n'y a pas d'itineraire, mettre itix dans tous les cas (meme avec des aiguillages ou bmvu)
# Il est possible de rajouter des aiguillage apres un iti<0-999> ou un itix
# Pour un itineraire de manœuvre, rajouter m, il n'y aura donc que les valeurs C(pour le Cv) ou
M
#
# Il faudra definir pour chaque etat C/S/A/L/M, l'etat de toutes les leds en signalisation
# lumineuse ou la position des servomoteurs en signalisation mecanique.
# Si vous souhaitez juste specifier les etats actifs des leds, alors rajouter l'etat D (defaut)
# en premier pour eteindre toutes les leds avant d'activer les actives dans les autres etats.
# Les signaux peuvent être reels (implantes sur le reseau) ou virtuels (non presents)
# Les signaux peuvent servir a la conduite automatique des trains.
#
# Pour un signal qui accepte le v200 (vert clignotant pour reduire a 160km/h), rajouter v200
# l'etat LCLI est alors ajoute
#
# Si vous souhaitez utilsier le rappel de ralentissement RR et RRCLI pour les aiguilles en
pointes
# rajouter rr ou rrcli dans chaque chemin
# Si vous souhaiter avertir un RR ou RRCLI par un avertissement, ajouter ra dans le chemin
# Si vous souhaiter avertir un RR ou RRCLI par un ralentissement R ou RCLI, ajouter r dans le
chemin
# Les etats suivant apparaissent RR0,RR,RRCLI,R,RCLI (et peuvent etre en // de l'etat A)
# Il faudra definir les leds a allumer et la position des servos pour ces etats
#
# Vous pouvez rajouter un indicateur de direction en rajoutant dir<0-4>, 0-4 indique le nb de
lampes a activer
#
# syntax:
# s<0-199>:
# iti<0-999>/itix a<0-95*>[-,/] i<0-255*>/j<0-99> s<0-199>/stop m rr/rrcli ra/r dir<0-4> v200
/ ?? bmvu<0-9>[a,b]
# D/C/S/A/L/LCLI: l/c<0-255*>[+/-] srv<0-47*>=<50,255>
# R0/RR/RRCLI/R/RCLI: "
# D0/D1/D2/D3/D4:     "
# * = ajouter 0 pr la board 0, 1000 pr la board 1 ...
#
# ex:
# # C101
# s1:
# iti1 j1 s11
# iti2 j2 s12
# C: l0+ l1+ l2- l3-
# S: l0- l1+ l2- l3-
# A: l0- l1- l2+ l3-
# L: l0- l1- l2- l3+
#
# meme ex:
# # C101
# s1:
# iti1 j1 s11
# iti2 j2 s12
# D: l0- l1- l2- l3-
# C: l0+ l1+
# S: l1+
# A: l2+
# L: l3+
```

La réalité :

Avant de mettre des signaux n'importe où, il est bon de s'intéresser un peu à la réalité. Les lignes suivantes expliquent ce que j'ai compris de la signalisation SNCF. Désolés, je ne me suis pas intéressé à la signalisation internationale. N'hésitez pas à me corriger, n'étant pas du métier, je n'ai surement pas tout compris. J'ai classé la signalisation en 3 grandes familles qui sont :

- La voie unique
- La double voie
- Les gares et bifurcations

Pour chaque famille, il y a ensuite plusieurs sous familles. Je me suis intéressé aux sous familles suivantes :

Voie Unique (VU) :

- VUTR (Voie Unique Traffic Restreint)
- VUSS (Voie Unique Signalisation Simplifiée)
- VUSO (Voie Unique Signalisation Ordinaire)
- BMVU (Block Manuel de Voie Unique)
- BAPR-VU (Block Automatique à Permissivité restreinte de Voie Unique)

Double voie (DV)

- BMU (Block Manuel Unifié)
- BAL (Block Automatique Lumineux)
- BAPR (Block Automatique Lumineux à Permissivité restreinte)

Gares et bifurcations

- Les itinéraires
- Ralentissement et Rappel de Ralentissement avant les aiguilles en pointes

Il existe d'autres signalisations mais je ne les ai pas retenus.

Pour plus de détails, vous trouverez de nombreux sites sur Internet très bien fait.
Par exemple :
- http://geillon.pagesperso-orange.fr/trains/signaux/
- http://e.bournez.free.fr/signalisation.pdf
- http://www.roverch.net/modelisme/Modelisme.html

Les sections suivantes expliquent chaque famille et sous famille de signalisation et montre comment les gérer avec free-dcc pc.

La Double Voie (DV) :

Nous commencerons par la double voie, car c'est la plus facile à comprendre. Etablir une double voie coute bien plus chère qu'une voie unique. On l'établit quand on a besoin de débit et on la dote donc presque toujours d'une signalisation performante. Bien que le BMU (Block Manuel Unifié) soit encore utilisé, fdcc_pc ne le propose pas car il serait fastidieux pour le modéliste de valider le passage de chaque train d'un canton à l'autre. Néanmoins on pourra le simuler automatiquement avec le BAPR.

Le BAL :

Le BAL ou Block Automatique Lumineux très bien connu des modélistes est le plus simple à comprendre. C'est aussi le plus performant en termes de débit.

Si le canton protégé est occupé alors le feu est rouge (sémaphore). Il peut être franchit après arrêt en maintenant la marche à vue (30km/h) sur le canton. Il peut être préférable d'attendre le feu orange pour parcourir le canton plus

vite. Dans une rampe ou pour une gare non desservit, il est possible d'avoir un rouge clignotant autorisant le franchissement à 30km/h sans arrêt. Pour détecter l'occupation du canton sur votre réseau, il est possible d'utiliser un ou plusieurs capteurs de courant reliées à des entrées (i0-255). Cela fonctionne bien comme en réalité si tous les essieux sont résistifs. Plusieurs capteurs sont supportés car on découpe souvent les cantons en 2 afin d'avoir une zone d'arrêt pour arrêter les trains en fonctionnement automatique. Il est également possible d'utiliser du matériel avec des essieux non résistifs en utilisant les cantons de fdcc_pc. En effet les cantons détectent le premier véhicules grâce moteur de la locomotive qui consomme ou avec un essieu résistif d'une remorque en cas de réversibilité ou à l'aide d'un ILS ou d'une pédale de voie. Ensuite un algorithme permet d'occuper et libérer les cantons (en plus de suivre les trains). Les variables d'occupations des cantons sont les variables j0-99

Si le canton suivant est occupé alors le feu du canton protégé est orange (on ne dit jamais jaune à la SNCF !). Pour plus de simplicité, on utilisera l'état du feu qui protège le canton suivant au lieu d'utiliser les capteurs de détection comme dans le cas du feu rouge. Cela nous permettra de gérer les itinéraires simplement. Si le feu suivant est rouge alors notre feu sera orange (s'il n'est pas déjà rouge bien entendu). Il faudra donc juste indiquer le numéro du feu suivant.

Dans les cas contraires, le feu est vert.

Voici les lignes à rajouter pour gérer le signal s0 qui protège le canton0 :


```
# S18.5
s0:
itix j0 s1
S: l0- l1+ l2-
A: l0- l1- l2+
L: l0+ l1- l2-
```

ou

```
# S18.5
s0:
itix j0 s1
D: l0- l1- l2-
S: l1+
A: l2+
L: l0+
```

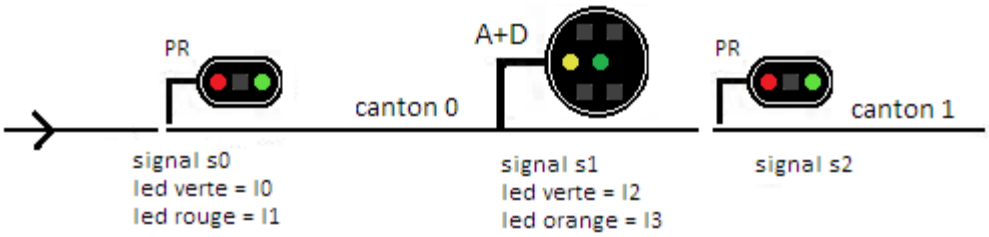
Comme il n'y a aucun itinéraire entre les 2 cantons, on indique « itix ».
Si l'on souhaite un rouge clignotant, on peut rajouter c1+
J'ai choisi d'utiliser la variable d'occupation j0 du canton 0 afin de pouvoir utiliser du matériel sans essieux résistif. Comme indiqué au chapitre sur les cantons cela ne correspond pas exactement à la réalité mais c'est très acceptable.
Pour parfaire la réalisation, on ajoutera une pancarte « F » blanc pour franchissable sur le signal plus un numéro noir pour le repérer comme 18.5km pour indique le PK (point kilométrique)
Notez que le canton suivant n'est pas forcément un canton de BAL.

FDCC-PC gère aussi le vert clignotant pour les voies aptes à 200km/h afin de réduire la vitesse à 160 km/h. Pour cela, il suffit de rajouter v200 dans le « chemin ». Cela crée le nouvel état LCLI.
Ex si on équipe le feu s0 du vert clignotant :

```
# S18.5
s0:
itix v200 j0 s1
S:    l0- l1+ l2-
A:    l0- l1- l2+
L:    l0+ l1- l2- c0-
LCLI: l0+ l1- l2- c0+
```

Le BAPR :

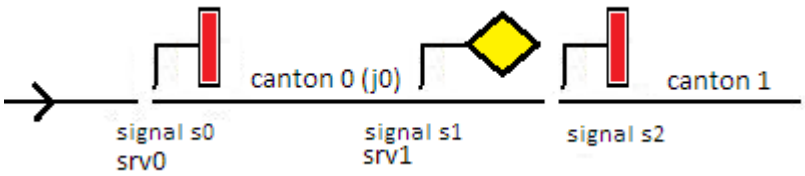
Le BAPR ou Block Automatique à Permissivité Restreinte est utilisé sur les lignes à double voie à faible débit où il n'est pas nécessaire d'avoir des cantons courts. Un canton de BAL mesure 1500m environ alors qu'un canton de BAPR peut faire jusqu'à 20km. Un canton de BAPR est protégé par un sémaphore (rouge ou vert) non franchissable reconnaissable à sa pancarte PR. Avant la sortie du canton un signal d'avertissement à cible ronde indique si le feu du canton suivant est rouge. Voici les lignes à rajouter pour gérer le signal s0 qui protège le canton0 :




```
# PR18.5
s0:
itix j0
S: l0- l1+
L: l0+ l1-

# A30.5
s1:
itix s2
A: l2- l3+
L: l2+ l3-
```

L'état du signal s0 ne dépend pas du canton suivant, le signal s2 n'est donc pas spécifié sur sa ligne de commande, il ne sera donc jamais dans l'état A. Inversement le signal s1 dépend entièrement du signal s2, comme il n'y a pas de détecteur d'occupation dans sa ligne de commande, il ne présentera jamais l'état S.
Notez que le canton suivant n'est pas forcément un canton de BAPR.

Il est possible d'émuler le BMU avec le BAPR en remplaçant les LEDs par des servomoteurs.




```
# PR18.5
s0:
itix j0
S: srv0=100
L: srv0=200

# A30.5
s1:
itix s2
A: srv1=100
L: srv1=200
```

La Voie Unique (VU) :

La voie unique est plus compliquée que la double voie car il ne s'agit plus d'assurer uniquement l'espacement des trains mais aussi d'éviter qu'ils ne rentrent en collision frontale. La signalisation est bien plus variée qu'en double voie. On trouve fréquemment :

- VUTR (Voie Unique Traffic Restreint)
- VUSS (Voie Unique Signalisation Simplifiée)
- VUSO (Voie Unique Signalisation Ordinaire)
- BMVU (Block Manuel de Voie Unique)
- BAPR-VU (Block Automatique à Permissivité restreinte de Voie Unique)

La VUTR est la plus simple car il n'y a pas du tout de signalisation ! Juste une pancarte VUTR à l'entrée de la voie, mais il ne peut y avoir qu'un seul train et encore de marchandise dans toute la zone.

Ensuite vient la VUSS qui utilise des pancartes fixes comme signalisation puis la VUSO avec de la signalisation gérée à la main. Inutile de préciser qu'en VUSS les gares sont d'arrêt générale, tous les trains s'y arrêtes et ne poursuive leur route que sur autorisation du chef de gare.

En VUSS et VUSO utilisent le cantonnement téléphonique pour s'envoyer des trains.
Il s'agit simplement d'échange oraux standardisés.
- « GareA, annonce l'envoie le train numéro 1234 à GareB »
- « Gare B, prend note de l'envoie du train 1234»
- « Gare B, rend la voie libre après réception du train 1234 arrivé complet »
- « Gare A, prend note de la réception du train 1234 »

Afin de renforcer la sécurité, le système CAPI (Cantonnement assisté par informatique) peut être installé et remplace les échanges oraux par des échanges informatique.
- Gare A tape : A 1234 (annonce). Le train ne peut être envoyé que si le système est d'accord.
- Gare B acquitte avec : B 1234
- Gare B libère avec : L 1234
- Gare A acquitte avec : M 1234

Bien que facile à implémenter, je n'ai pas retenu ce système car il rendrait fastidieux le jeu. Il faut juste savoir qu'il existe et qu'il a sans doute évité de nombreux accidents. La CAPI peut être couple au DAAT (Dispositif d'Arrêt Automatique des Trains) qui arrête tout train sortant non autorise (grâce au crocodile).

Le BMVU est un système avec des appareils dédiés qui permet d'envoyer des trains en toute sécurité. Il contrôle le sémaphore de chaque gare protègent le canton. Il peut être couplé au DAAT.

Enfin le BAPR-VU fonctionne comme les itinéraires. Il faut établir un itinéraire pour que le carré d'une des voies de sortie présente la voie libre. Les enclenchements protégeant la voie unique tant que le train ne l'a pas libéré.

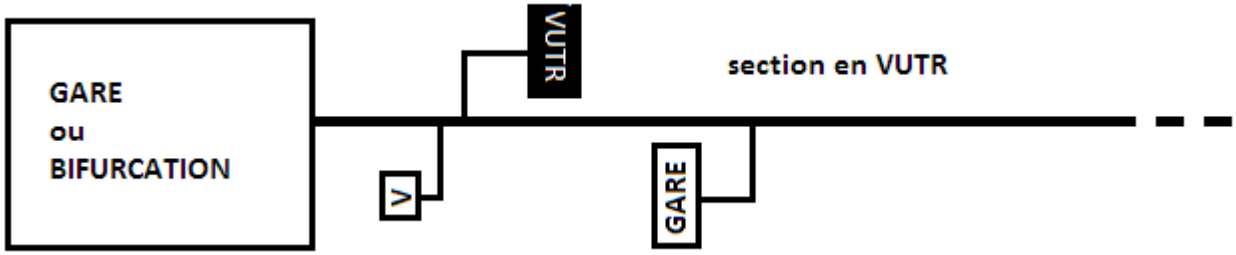
Il convient également de protéger la gare dans le cas d'une manœuvre sur la voie unique. Le signal disque assure alors cette protection et commande au train d'être en mesure de s'arrêter avant la pancarte LM (Limite de manouvre) que le train en manœuvre ne doit pas dépasser. Le disque est aubiné (il se ferme tout seul au passage d'un train) ce qui assure sa protection si un autre train est envoyé. Il convient donc de le réouvrir pour recevoir un train.

Lorsque l'arrêt n'est pas obligatoire, il convient d'ajouter un avertissement commandé par le signal de sortie de la gare afin d'arrêter le train si nécessaire.

Enfin il peut être nécessaire d'indiquer la position des aiguilles avec une pancarte mobile, un avertissement, un ralentissement plus rappel de ralentissement …

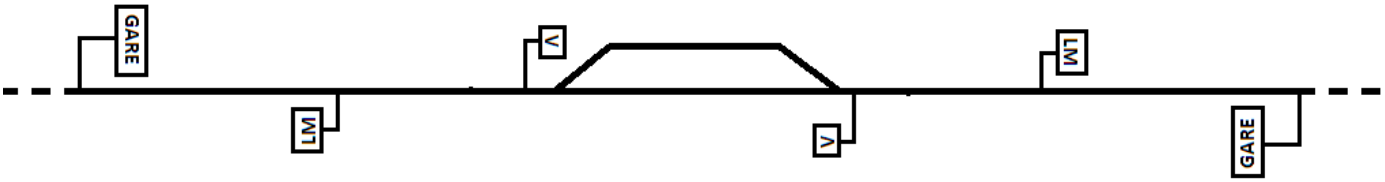
La VUTR :

L'entrée dans la VUTR est simplement indiquée par une pancarte VUTR. La sortie dépend de la gare ou de la bifurcation. Cela peut aller de la signalisation simplifiée à un carré annoncé par un avertissement. Un carré violet doit également être suffisant si la vitesse est inférieure à 30km/h. Bien entendu, il n'y a rien à faire pour supporter la VUTR avec fdcc. Naturellement les trains en cantonnement automatique ne pourront pas emprunter ce type de voie. Vous pouvez néanmoins automatiser avec le mode script mais sans aucune garantie en termes de sécurité !



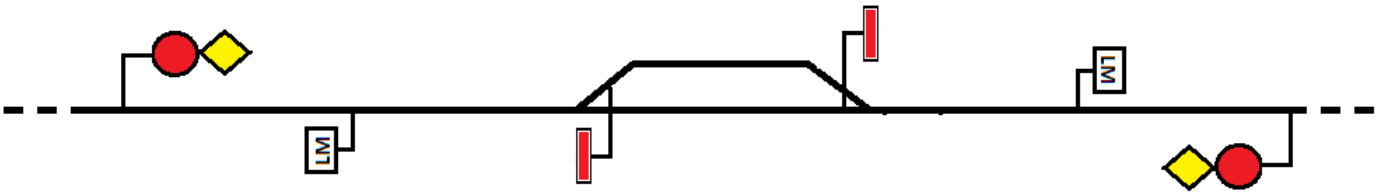
La VUSS :

En VUSS, l'entrée en gare est indiquée par la pancarte GARE. Il s'agit bien de l'indication gare et non pas du nom de la gare ! Le conducteur doit aussi tôt se mettre en marche a vue (30km/h) pour s'arrêter si un train manœuvre après la pancarte LM, pour franchir les aiguilles à vitesse raisonnable, pour ne pas entrer en collision avec un autre train. Un disque peut être ajouté pour encore plus protéger les manœuvres. Une fois en gare, le train doit s'arrêter et ne repartira pas sans l'autorisation du chef de gare. Par surcroit de sécurité le chef de gare dispose un SAM (Signal d'Arrêt à Main) qui est un carré portatif pour interdire les départs. Comme indiqué plus avant la VUSS est complétée du cantonnement téléphonique et quelque fois du CAPI et du DAAT. Ici aussi, il n'y a rien à faire pour supporter la VUTR avec fdcc. Naturellement les trains en cantonnement automatique ne pourront pas emprunter ce type de voie. Vous pouvez néanmoins automatiser avec le mode script mais là encore sans aucune garantie en termes de sécurité !



La VUSO :

En signalisation ordinaire des signaux mobiles ou lumineux sont ajoutés. Ils sont gérés manuellement sans dispositif de protection hormis les enclenchements entre les signaux et les aiguilles. Certains sont aubinés, c'est-à-dire qu'ils se ferment avec le passage des trains. Les gares peuvent être franchit sans arrêt. Dans ce cas, il faut ajouter un avertissement commandé par le sémaphore ou carré de sortie. Comme indiqué plus avant la VUSO est complétée du cantonnement téléphonique et quelque fois du CAPI et du DAAT. Une pancarte mobile de limitation de vitesse (ex : 40) peut être ajouté en cas de franchissement en dévié d'une aiguille. Je ne sais pas si l'avertissement est utilisé en cas d'une aiguille en dédiée. La signalisation mécanique peut être remplacée par de la signalisation lumineuse.



Le sémaphore ou la carrée sera commandé de manière manuelle. Nous utiliserons la variable du commutateur de fermeture du signal f<0-199>. Normalement cette variable permet de forcer un carré au rouge. On pourra ajouter sur un TCO un commutateur de fermeture ou pourquoi pas un levier commandant ce signal. L'avertissement correspondant sera associé à ce signal. Bien que l'on puisse faire tout ceci avec des équations, on peut aussi utiliser le fichier de signalisation comme suit :

```
# S101
s0:
itix i1
C: srv0=100
L: srv0=200

# A103
s1:
itix s0
A: srv1=100
L: srv1=200
```

ou

```
# S101 + A103
s0:
itix
C: srv0=100 srv1=100
L: srv0=200 srv1=200
```

Il est possible de rajouter l'aubinage pour fermer le commutateur de fermeture avec les équations. Ex :

```
l i1 =s f0
```

Lorsque l'entrée sera active, le commutateur de fermeture sera force sur la fermeture. Bien entendu c'est plus complexe en réalité car l'aubinage ne vient pas commander la commande !

```
# S101 + A103
s0:
C: srv0=100 srv1=100
L: srv0=200 srv1=200
```

Enfin, on gérera le disque comme le sémaphore ou le carré.

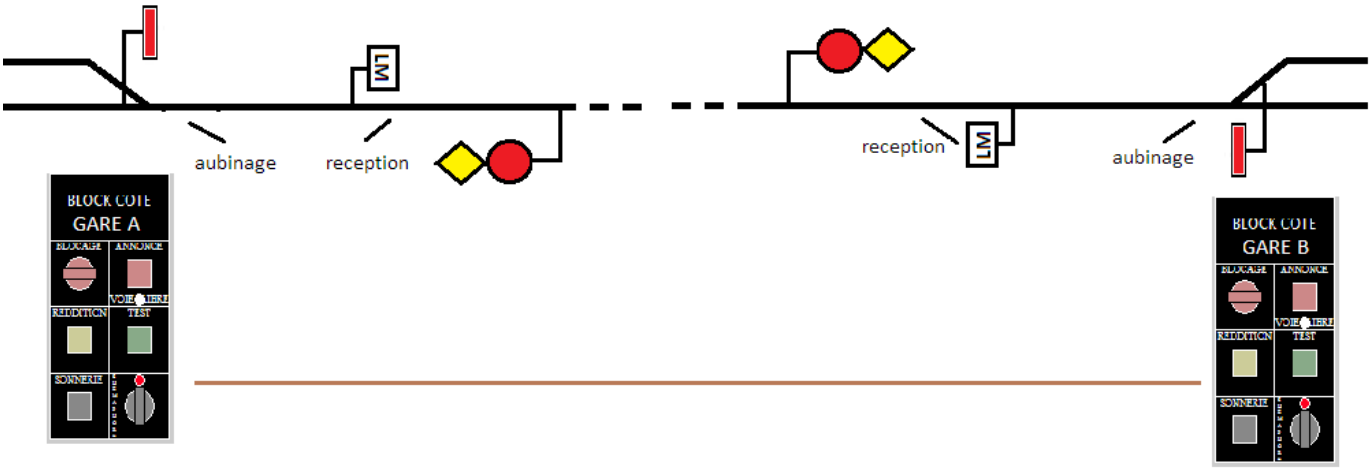
Un train en cantonnement automatique pourra emprunter ce type de cantonnement avec quelques restrictions. Comme le signal de sortie s'adresse aux 2 voies, il faudra prendre garde qu'il n'y est qu'un seul train ou alors créer des signaux virtuels se référant au signal réel mais en ajoutant après itix la position de l'aiguillage de sortie. Le respect du disque ne sera pas assuré. Il vaut mieux utiliser des trains en cantonnement automatique avec le BMVU ou le BAPR-VU.

Il y a également quelques spécificités selon que la gare est de voie de gauche ou directe.

Le BMVU : **PROCHAINEMENT DISPONBLE**

Visuellement les signaux sont les mêmes qu'en VUSO, mais un dispositif composé d'un boitier dans chaque gare est ajouté afin d'assurer la sécurité en contrôlant le sémaphore ou carré de sortie. Les boitiers communiquent entre eux

Le boitier peut être assez gros pour ses premières versions ou miniaturisé ce qui est plus intéressant pour le modélisme. Fdcc_pc permet de créer de tel boitiers.



Le fonctionnement est le suivant :

- Si GARE A veut envoyer un train, alors le chef de gare appuie sur le bouton test. S'il s'allume au vert (au bout de quelques secondes), le système indique que rien ne s'y oppose. Le voyant s'éteint après 20 secondes et si rien n'a été fait, il faut réappuyer sur test pour une nouvelle demande.
- A peut ouvrir le sémaphore avec le bouton rotatif du bas. Le bouton s'allume en blanc lorsqu'il est en position ouverture mais le sémaphore ne s'ouvre que si le bouton test est encore vert. La LED rouge qui indique la fermeture du sémaphore s'éteint. Le voyant blanc de voie libre s'éteint.
- Lorsque le train part, il actionne la pédale d'aubinage du sémaphore qui se ferme. Ce qui allume la LED rouge.
- Gare A appuie sur le bouton Annonce qui devient rouge. Cela allume le bouton reddition de B en orange et le bouton sonnerie en blanc en plus de faire sonner la sonnerie (qui est non simulée avec fdcc)
- B appuie sur le bouton de la sonnerie pour l'éteindre
- Lorsque le train arrive vers la gare B, il actionne la pédale de réception qui est une sécurité pour interdire B de rendre voie libre prématurément sans avoir reçu le train (bien que le train ne soit pas encore en gare). La pédale est placée en dehors de la zone de manœuvre.
- Lorsque le train est en gare, B appuie sur le bouton de reddition qui s'éteint. Au même moment le bouton d'annonce de A s'éteint, le bouton sonnerie et la sonnerie s'active fugitivement. (A n'a rien à faire)

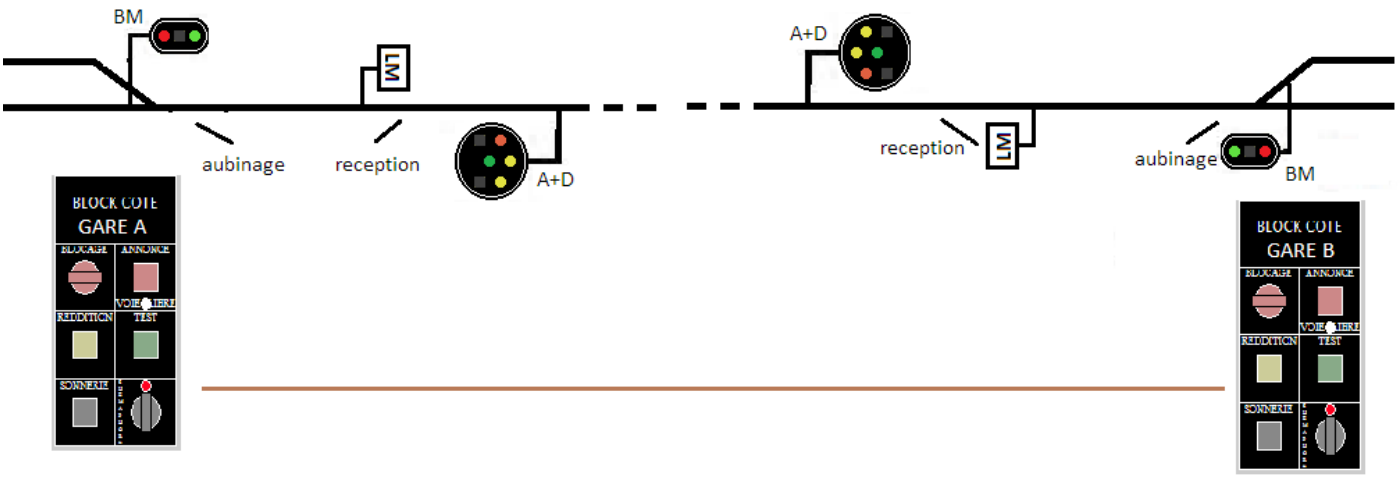
Le chapitre BMVU donne plus d'information sur le BMVU. Fdcc-PC permet de créer des BMVU qui communiquent avec le reste du système par des variables. Il est ainsi possible d'ajouter des boutons et voyant sur les TCOs. Dans le fichier de signalisation, il suffit d'indiquer le numéro du BMVU de 0-9 et sa position gare A ou B. Si le BMVU est actif alors l'état C du signal est appliqué. L dans le cas contraire. (On utilise l'état C bien qu'il s'agisse souvent d'un sémaphore)

```
# S101 + A103
s0:
itix bmvu0a
C: srv0=100 srv1=100
L: srv0=200 srv1=200
```

Le boitier dispose également d'un commutateur de blocage qui empêche le test de passer au vert. Utile par exemple lorsqu'il y a des travaux. Les manœuvres se font sémaphore fermé (sur autorisation du chef de gare et sous protection du disque).

La signalisation lumineuse peut remplacer la signalisation mécanique. Le DAAT peut être ajouté.

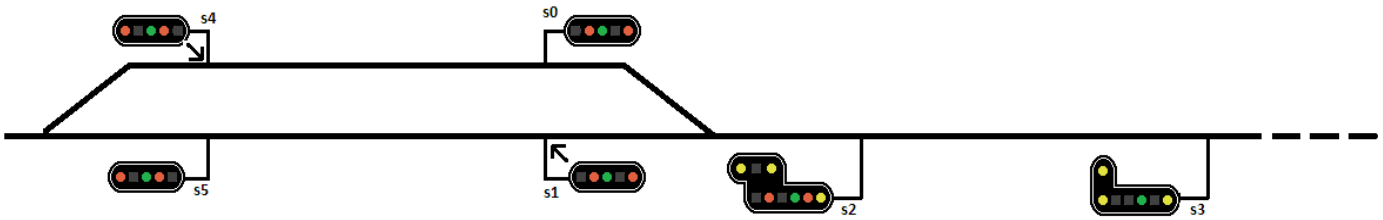
On ajoutera une pancarte BM sur le sémaphore et une pancarte A ou D sur le disque. (En mode disque les lampes rouge et orange du coté sont allumées)



Les trains en cantonnement automatiques peuvent emprunter le BMVU avec quelques restrictions. Comme le signal de sortie s'adresse aux 2 voies, il faudra prendre garde qu'il n'y est qu'un seul train ou alors créer des signaux virtuels se référant au signal réel mais en ajoutant après itix la position de l'aiguillage de sortie. Le respect du disque n'est pas assuré.

Le BAPR-VU :

Le BAPR-VU s'utilise comme les itinéraires. On prendra garde d'utiliser une variable de zone pour éviter les itinéraires nez à nez. La signalisation est assez lourde car il faut de nombreux signaux.



Chaque aiguillage est protégé par 3 carrés commandés par 4 itinéraires DA auxquels on peut ajouter des TP. Il y a donc 8 itinéraires par gare minimum. Coté voie unique la pointe de l'aiguillage est protégé par un carré avec un rappel de ralentissement, signalé en amont par un signal d'avertissement avec indication de ralentissement.

```
# C101
s0:
iti1 j0
C: l1+ l2- l3+
S: l1+ l2- l3-
L: l1- l2+ l3-
# C103
S1:
iti2 j0
...
# C102 (+RR)
S2:
iti4 j1 s4
iti5 j2 s5
C: l10+ l11- l12+ l13-
S: l10- l11- l12+ l13-
A: l10- l11- l12- l13+
L: l10- l11+ l12- l13-
# A104 (+RR)
S3:
itix s2
...
```

Ps : Crédits pour les images des jolis signaux : https://thuthuboutick.fr/bal/signaux1s.htm

Les gares ou bifurcations :

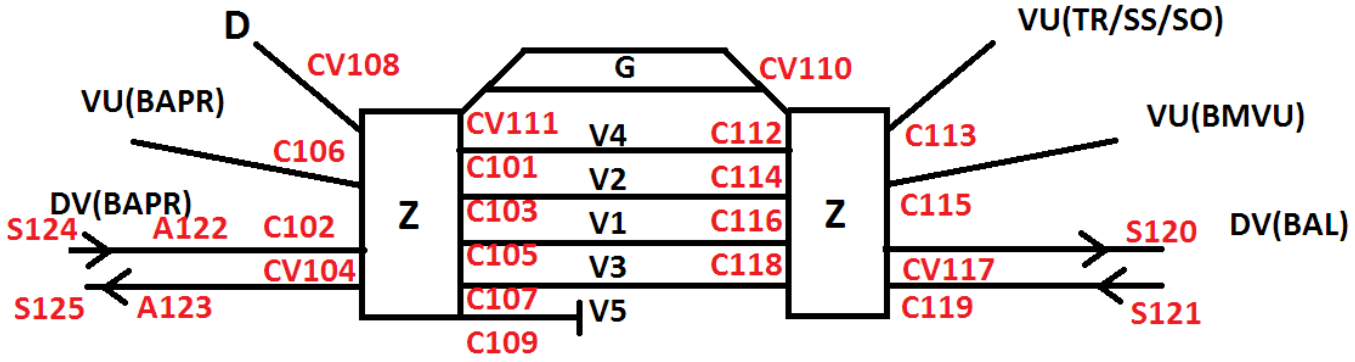
Introduction :

Pour les gares ou bifurcations, tout se complique car il faut protéger les différents itinéraires. La solution retenue est étonnement simple, en effet pour chaque signal, il suffit d'ajouter une ligne par itinéraire en spécifiant l'itinéraire ou la position des aiguillages. Si aucune ligne ne possède un itinéraire actif ou les aiguillages dans la bonne position, l'état du signal sera mis à C. Dans le cas contraire, si le canton est occupé, il sera mis à S, à A si le signal suivant est au rouge ou s'il s'agit d'un heurtoir (en indiquant stop à la place du signal suivant). Et L dans le cas contraire.
Les carrés violets pourront également être utilisés, il suffit de rajouter la lettre m dans les « chemins ». En carré violet il n'y a pas à indiquer ni détecteur d'occupation ni de signal suivant. L'état C allumera la lampe violette et V pour la blanche à moins que l'on préfère utiliser un servomoteur pour commander une pancarte de carré violet.
Une gare peut recevoir des lignes doubles et/ou uniques avec des itinéraires pour desservir les différents quais. Il est également possible de gérer les signaux de ralentissement et rappel de ralentissement. Si vous utiliser des itinéraires, merci de lire ou relire le chapitre sur les itinéraires.

Position des signaux et découpage électrique :

Les carrées servant à protéger les itinéraires formés par les aiguillages, il suffit de repérer les zones d'aiguillages (ainsi que les aiguillages isolés qui forment aussi une zone d'aiguillage) et de les protéger par des carrés. Chaque entrée d'une zone d'aiguillage sera protégée par un carré ou carré violet. Entre ces zones s'étendent les cantons où l'on peut trouver des signaux d'espacement s'ils sont suffisamment longs. En cas d'utilisation de capteurs de courant, le découpage électrique suivra les zones d'aiguillages. Dans une même zone d'aiguillage, on isolera les itinéraires pouvant fonctionner en parallèle. Pour les cantons on les découpera en 2 ou 3 zones comme indiqué au chapitre sur les cantons. Pour les très petites zones d'aiguillages, il est possible de se passer de détection.

La figure suivante montre la protection d'une gare de taille moyenne :



On y trouve 2 grosses zones d'aiguillages dont chaque entrée est protégée par un carré ou un carré violet. Le dépôt D, la zone de garage G et les rebroussements sur voies principales utilisent des carrés violets ce qui réduit la signalisation car la vitesse est limitée à 30km/h donc pas besoin d'avertissements ni de ralentissements. Les autres itinéraires parcourus par des circulations commerciales sont protégés par des carrés. Ces derniers sont annoncés par des avertissements inclus dans les signaux les précédents. Les carrées externes disposent normalement du rappel de ralentissement. Les carrés internes à la gare les remplacent opportunément par des pancartes de limitation de vitesse associés à des pancartes Z pour exécution immédiate.

Sur les voies de services (D, G), la signalisation est simplifiée et se limite souvent à des carrés violets

Fichier de signalisation

Il est maintenant temps de remplir le fichier de signalisation. Je ne vais pas décrire tous les signaux mais juste le carré C102 et le carré violet CV108. Pour le C102 disons que l'itinéraire 1 permet d'accéder à la voie 1, le 2 à la 2, …, le 5 à la 5 et le 0 aux voies de garages. Pour le CV108, rajoutons 10 à chacun de ces itinéraires. En gardant le même nom des signaux cela donnerait le fichier suivant :

```
# C102
s102:
iti0 m
iti4 j4 s112
iti2 j2 s114
iti1 j1 s116
iti5 j3 s118
iti3 j5 stop
#    R    B    R    V    J
D: l20- l21- l22- l23- l24-
C: l20+      l22+
S:           l22+
A:                     l24+
L:                l23+
M:      l21+

# CV108
s108:
iti10 m
iti14 m
iti12 m
iti11 m
iti13 m
iti15 m
M: l10- l11+
C: l10+ l11-
```

Vous voyez que ce n'est pas bien compliqué !

Il faut juste mentionner le cas particulier de l'accès à un canton BMVU pour lequel il faut combiner le BMVU à un itinéraire. Partons par exemple de la voie 1 pour aller sur la VU BMVU. Le fichier serait le suivant :

```
# C116
s116:
iti31 bmvu1a
...
```

Cet exemple ne tient pas compte des ralentissements et rappels de ralentissement pour les aiguillages en pointes qui sont d écrit dans le chapitre suivant.

Rappels de ralentissement et ralentissement avant les aiguillages :

Avant de prendre un aiguillage en pointe en position déviée, il faut souvent ralentir. Il y a 2 techniques :

- La plus simple consiste à ralentir quel que soit la position de l'aiguillage. Certes ce n'est pas optimal pour la vitesse mais c'est nettement moins cher en signalisation. En effet quelques pancartes fixes suffissent. Comme les pancartes GARE et CHEVRON en VUSS (limitation a 40km/h), ou les pancartes de limitation de vitesse (30,40…) conjuguées avec les pancartes Z (Zero = ralentir) ou R (Reprise)

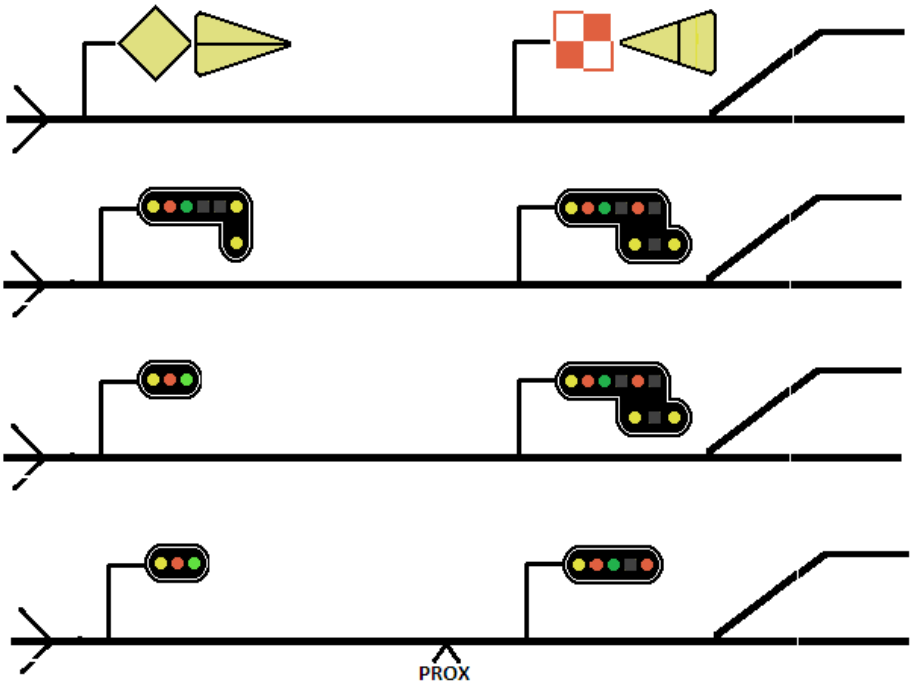
- Soit on optimise et on ne ralentit que lorsque l'aiguille est en déviée. La signalisation est plus complexe car elle doit changer en fonction de la position de l'aiguillage ou des aiguillages. On utilise alors :
    - Des panneaux de limitation de vitesse mobile (ex : 40 mobile en VUSO)
    - De la signalisation mécanique
    - De la signalisation lumineuse

Pour les pancartes fixes, il n'y a rien à faire dans FDCC.

Pour la pancarte de vitesse mobile, il faut simplement la lier à l'aiguillage.
On pourra par exemple utiliser une équation :

```
# PIV mobile 40 lie a l'aiguillage 2 et controle par le servo 5
ln e2 = srv5=100
l  e2 = srv5=200
```

En signalisation mécanique ou lumineuse, on placera un signal de rappel de ralentissement RR juste avant l'aiguillage. Ce signal sera normalement averti à distance par un signal de ralentissement R. Il est possible de remplacer le R par un A.



En signalisation mécanique, la limitation est de 30km/h.
En signalisation mécanique, elle est de 30km/h si les feux sont fixes ou de 60km/h si les feux clignotent.

Pour être complet, reste encore, l'enclenchement de proximité. Le carré protégeant l'aiguillage reste fermé même une fois l'itinéraire formé, et ne s'ouvre que lorsque le train est à proximité du carré. Le train aura donc cassé sa vitesse en vue de s'arrêter au carré et franchira donc l'aiguillage à vitesse réduite. FDCC-PC supporte les enclenchements de proximité. Au niveau de la signalisation, rien ne change par rapport aux autres carrés protégeant des itinéraires.

Il y a un cas encore plus compliqué ou le même panneau peut afficher un RR pour l'aiguillage protégé et un R pour un autre aiguillage après le canton protégé. Si RR et R doivent s'afficher alors R est remplacé par A. Cela afin de réduire le nombre de lampes allumées sur un feu. De même si R ou RR sont affiché alors L n'est pas affiché.

Aux états C,S,A,L,LCLI,M, fdcc-pc ajoute un autre ensemble d'état composé de R0,RR,R,RRCLI et RCLI

* Pour avoir RR ou RRCLI, il faudra spécifier rr ou rrcli dans la ligne de chaque itinéraire devant afficher cette information. Ensuite, il faudra renseigner l'état des LEDs ou servos dans les nouveaux états RR0 (aucun R/RR), RR ou RRCLI. L'état A peut être actif en // des états RR,R,RRCLI et RCLI.

```
# C102
s102:
iti0 M
iti4 rr    j4 s112
iti2 rrcli j2 s114
iti1    j1 s116
iti5 rrcli j3 s118
iti3 rr    j5 stop
#    R    B    R    V    J  J(RR) J(RR)
D: l20- l21- l22- l23- l24- l25-  l26- c25- c26-
C: l20+      l22+
S:           l22+
A:                   l24+
L:               l23+
M:      l21+
RR:                      l25+ l26+
RRCLI:                   l25+ l26+ c25+ c26+
```

Dans cet l'exemple :

- L'itinéraire 1 ne rencontrant aucune aiguille peut être franchi sans ralentissement. Il n'y a donc pas de rr ou rrcli.
- L'itinéraire 4 est un itinéraire de manœuvre annoncé par un feu blanc M, donc déjà limité à 30km/h. il n'y a donc pas besoin de rajouter rr ou ni rrcli.
- Les itinéraires 2 et 5 sont déviés avec des aiguillages autorisant 60km/h en dévié (on rajoute donc rrcli)
- Les itinéraires 3 et 4 sont déviés avec des aiguillages autorisant 30km/h en dévié (on rajoute donc rr)

On utilise les leds 25 et 26 pour signaler le RR. On les allume uniquement dans les états RR et RRCLI et on les fait clignoter uniquement dans l'état RRCLI. On aurait aussi pu faire ceci pour le même résultat :

```
...
R0:    l25- l26- c25- c26-
RR:    l25+ l26+ c25- c26-
RRCLI: l25+ l26+ c25+ c26+
```

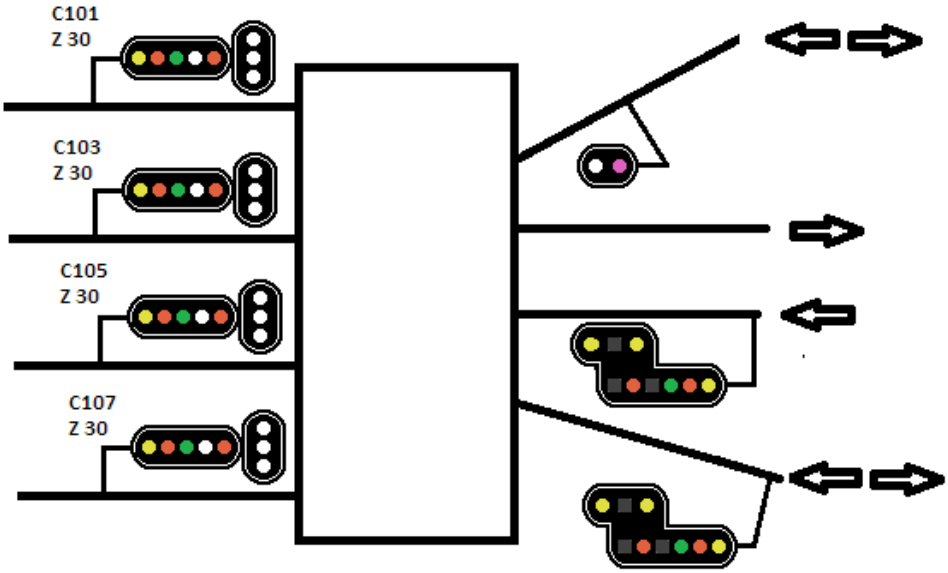
* Pour avoir R ou RCLI, il faudra rajouter « r » sur chaque ligne des itinéraires. Si l'on souhaite utiliser uniquement l'avertissement pour annoncer le RR, on remplacera le « r » par « ra ». Si l'on ne souhaite pas annoncer le RR, on n'ajoute ni « r », ni « r »a.

Dans l'exemple suivant, un R est rajouté sur le S20.3 qui précède le C102.

```
# S20.3
s44:
itix r j23 s102
#   R   J    V    J(RR) J(RR)
D: l50- l51- l52- l53-  l54-
S: l50+
A:      l51+
L:           l52+
R:                l53+  l54+
```

Indicateur de direction

Dans les bifurcations, la direction active est indiquée par un indicateur de direction géographique. Il comporte horizontalement autant de lampes blanches que de directions. Si le signal supportant l'indicateur n'est pas ouvert (L ou M), alors toutes les lampes sont éteintes. Si la direction géographiquement la plus à gauche est activée alors la lampe la plus à gauche s'allume. Pour la direction immédiatement à droite la lampe suivante s'allume en complément …



Pour utiliser un indicateur de direction, il suffit de rajouter dir<0-4> dans les différents « chemin » et de renseigner ensuite les états D0 à D4.

Ex :

```
# C101
s0:
iti10 m dir1
iti11 j23 s8 dir2
iti12 j24 s5 dir3
#    R    R     J     V     B
D: l50- l51- l52- l53- l54-
C: l50+ l51+
S:       l50+
A:             l52+
L:                   l53+
M:                         l54+
#    D1 D2 D3
D0: l60- l61- l62-
D1: l60+ l61- l62-
D2: l60+ l61+ l62-
D3: l60+ l61+ l62+
```

K. Conduite Automatique des trains en mode Cantonnement (CAC).

fdcc_pc permet La conduite automatique des trains en mode cantonnement automatique. Lorsque ce mode est activé sur une locomotive, elle va passer de canton en canton en respectant la signalisation. Cela n'est possible que si les cantons et la signalisation sont correctement configurés.

Le fonctionnement est simple : le module des cantons arrive à suivre les trains et les trains en mode cantonnement automatique obéissent au signal de sortie du canton. Si le signal présente :
- L, LCLI, A (sans RR ou RRCLI), R, RCLI, alors le train roulera à vitesse maximale sur tout le canton.
- RR ou RRCLI alors le train roulera à la vitesse de ralenti sur tout le canton.
- C, S, M, alors le train roulera au ralenti et s'arrêtera en fin de canton.

On peut comparer ce mode au mode block des anciens réseaux analogiques.

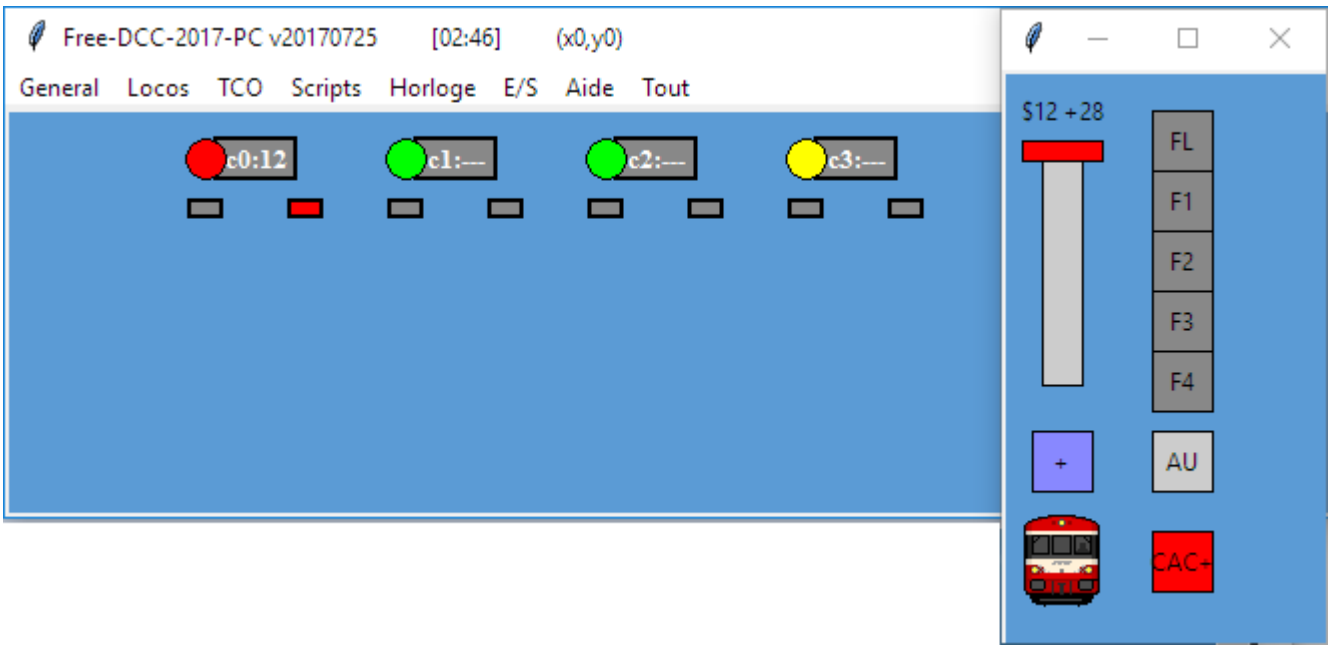
Tout cela est simple mais il faut associer les feux et les cantons et savoir vers quel côté du canton le train se dirige.
Pour les cantons, nous choisirons un côté « + » et un côté « - »
Nous allons rajouter quelques instructions dans le fichier des cantons :

- Au canton destination « c<0-99>: », nous ajouterons le coté final « + » ou « - » ou se dirigera le train qui rentre par ce chemin. Ce qui donne : « c<0-99>+/-: »
- Nous ajouterons les signaux du coté + et – avec par ex : « c<0-99>+: s28 »

Ex pour une boucle parcourable dans un seul sens composée de 4 cantons :

```
c0+: c3 i0 i10 s1 l12
c1+: c0 i1 i11 s2
c2+: c1 i2 i12 s3
c3+: c2 i3 i13 s0
```

Le TCO 6 propose une simulation avec les 4 cantons précédents, les détecteurs de début/fin et le feu associé :



Pour activer le mode CAC pour une locomotive :

- Ouvrir le régulateur virtuel de la locomotive.
- Amener la locomotive à son point de départ
- S'assurer qu'elle soit associée au canton, sinon l'associer avec le menu « loco > Affecter un canton »
- Mettre sa commande sens (+/-) de manière à partir dans bon sens
- Indiquer si la loco se diriger du côté + ou - du canton en cliquant droit sur le bouton CAC pour changer +/-
- Activer la conduite automatique en activant le bouton CAC qui devient rouge.

À tout moment, ce mode peut être désactivé en désactivant le bouton CAC.
En fonctionnement, il n'est pas possible de modifier manuellement la vitesse de la locomotive, par contre, il est possible de jouer avec les fonctions spéciales.

Dans l'exemple précédent, la signalisation est mise en mode BAL :

```
s0:
itix j0 s1
D: l0- l1- l2-
S: l0+
A: l1+
L: l2+
s1:
itix j1 s2
D: l10- l11- l12-
S: l10+
A: l11+
L: l12+
s2:
itix j2 s3
D: l20- l21- l22-
S: l20+
A: l21+
L: l22+
s3:
itix j3 s0
D: l30- l31- l32-
S: l30+
A: l31+
L: l32+
```

Vous pouvez ouvrir le fichier config_tco.txt pour voir le code du TCO6

Les vitesses à utiliser sont définis en début du fichier fdcc_pc.py :

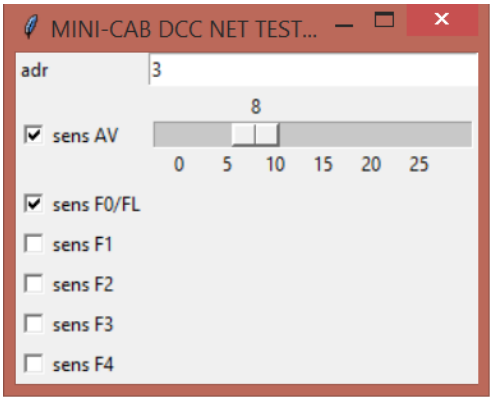
```
# vitesses a utiliser pour la conduite automatique en mode cantonnement (crans 0-28)
CAC_MAX  = 28
CAC_RAL  = 8
CAC_STOP = 0
```

La conduite automatique en mode cantonnement peut aussi être utilisé en mode script avec les instructions suivantes :

```
# ----conduite-automatique-en-mode-cantonnement-------------------------------------
#
# cac_init <loco> <canton>
#    ex: cac_init 12 3
#    demande au systeme d'affecter la loco indiquee sur le canton specifie
#    si la loco est 0 alors le canton est libere
#
# cac_start <loco> <+/->
#    ex: cac_on 12 +
#    demarre la conduite automatique de la loco specifiee en mode cac dans le sens spécifie
#
# cac_stop <loco>
#    ex: cac_stop 12
#    arrete la conduite automatique de la loco specifiee
#    met egalement sa vitesse a 0
#
# cac_wait <loco> <canton>
#    ex: cac_wait 12 8
#    attend que la loco soit completement sur le canton spécifié
#
# ex
#    loco 12
#    vit +0
#    set t28+
#    cac_init 12 3
#    cac_start 12 +
#    cac_wait 12 8
#    cac_stop 12
```

6.5. Le logiciel fdcc regu

Le logiciel fdcc_regu.py est un régulateur virtuel pouvant être utilisé pour conduire les locomotives via la centrale fdcc. Pour cela, il se connecte à fdcc_pc.py par le réseau. Bien entendu, il est possible d'utiliser plusieurs de ces régulateurs. Ces régulateurs, peuvent être sur le même PC que le logiciel fdcc_pc.py ou sur des PC distants connectés par Ethernet ou wifi.



Il suffit de régler les paramètres suivants au début du fichier

```
use_network   = 1
use_local     = 1
server_ip_adr = "10.20.30.40"
server_port   = 1234

dcc_adr = 3
sens = 1
cran_0_28 = 0
fct_43210 = 0
```

Mettre use_network à 1 pour activer le mode réseau.
Indiquer le numéro du port TCP sur lequel le programme fdcc_pc attend les connexions (1234 par défaut).
Mettre use_local à 1 si la cabine est sur le même PC que fdcc_pc.py, sinon renseigner l'adresse IP du PC ou se trouve dfcc_pc.py

Vous pouvez ensuite initialiser les paramètres dcc.

Pour voir si cela fonctionne, vous pouvez regarder le sens, la vitesse et les fonctions spéciales, évoluer dans la fenêtre de sélection des locomotives de fdcc_pc ou dans un régulateur virtuel de fdcc_pc. Le régulateur réseau n'est pas mis à jour si le sens, la vitesse ou les fonctions spéciales de la locomotive sont modifiés par un régulateur, une souris physique ou un script.

6.6. Les logiciels des cabines virtuelles

Les cabines du projet mini-cab peuvent être utilisées pour conduire les locomotives via la centrale fdcc. Pour cela, les cabines mini-cab se connectent à fdcc_pc.py par le réseau. Les cabines peuvent être sur le même PC que le logiciel fdcc_pc.py ou sur des PC distants connectés par Ethernet ou wifi.
Il suffit de régler les paramètres suivants au début des fichiers des cabines

```
use_serial     = 0
SZ_SERIALPORT = "\\\\.\\COM4"

use_network    = 1
use_local      = 1
server_ip_adr = "10.20.30.40"
server_port    = 1234
```

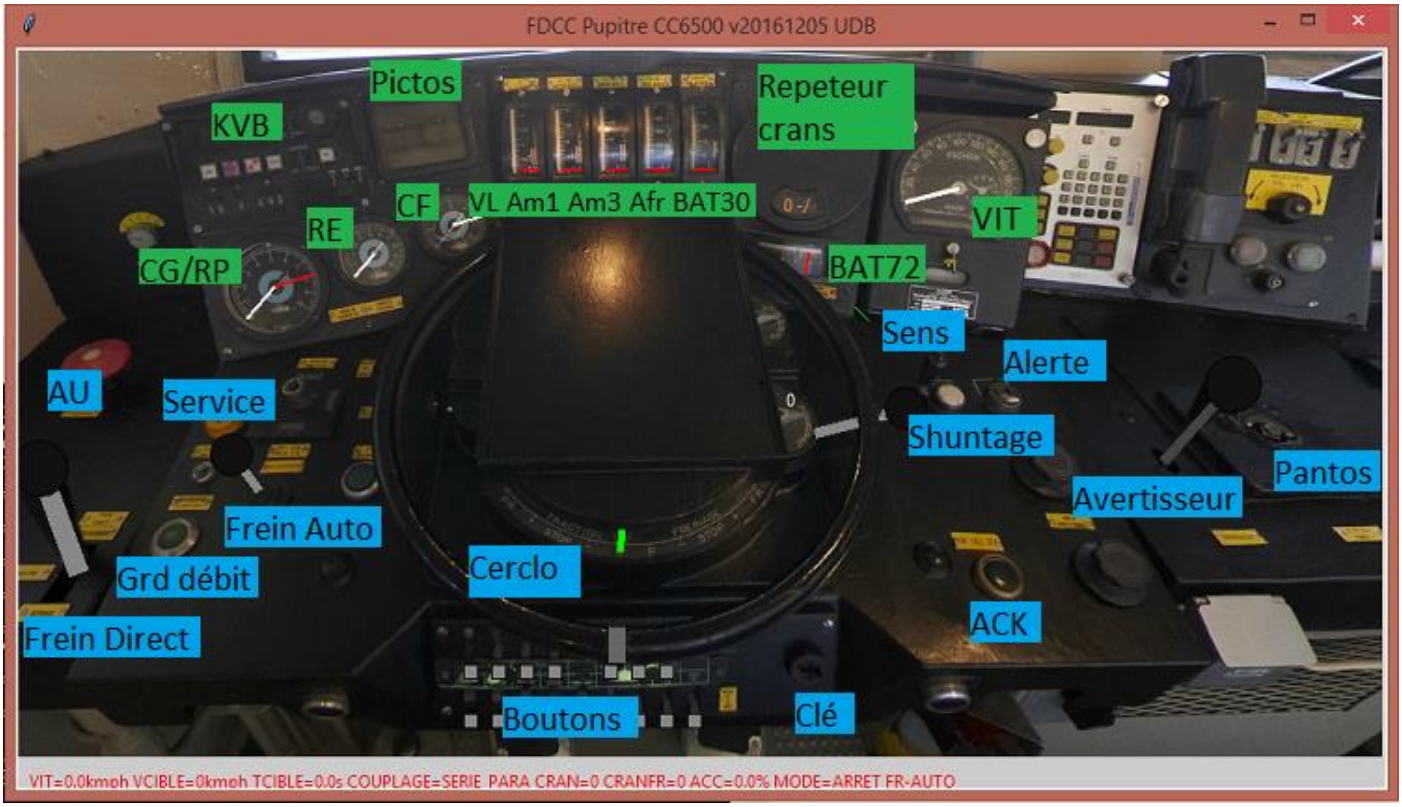
Les paramètres serial sont utilisés si vous voulez utiliser une carte Arduino pour y connecter des manettes afin de conduire avec plus de réalisme. Dans le cas contraire, désactivez cette fonctionnalité avec use_serial = 0. Dans ce cas le numéro de la liaison série n'a aucune importance.

Viennent ensuite les paramètres réseau.
Mettre use_network à 1 pour activer le mode réseau des cabines.
Indiquer le numéro du port TCP sur lequel le programme fdcc_pc attend les connexions (1234 par défaut).
Mettre use_local à 1 si la cabine est sur le même PC que fdcc_pc.py, sinon renseigner l'adresse IP du PC ou se trouve dfcc_pc.py

Il vous faudra aussi régler les paramètres dcc suivants :

```
# parametres dcc pour commander une locomotive via carte Arduino
dcc_adr        = 8    # adresse de la locomotive
dcc_vit_max    = 140  # vitesse correspondante a pwm_max
dcc_max        = 28   # valeur du cran DCC 0-28 a la vitesse max
dcc_min        = 1    # valeur du cran DCC 0-28 pour demarrer la loco
dcc_inv_sens   = 0    # inversion du sens (0=ne pas inverser, 1=inverser)
```

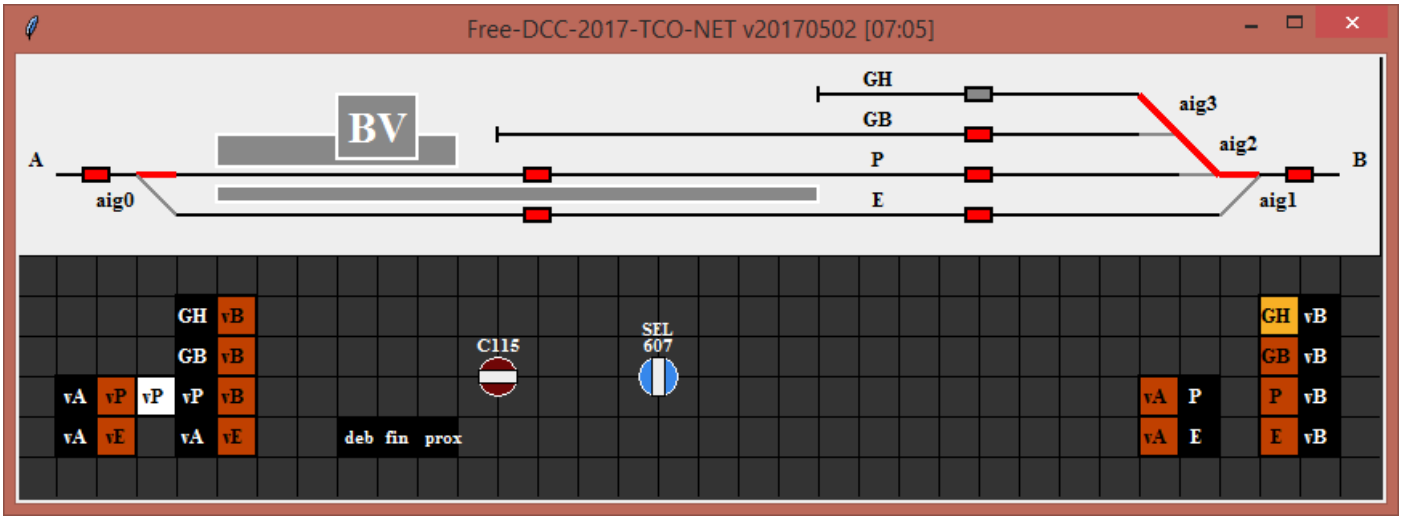
Reportez-vous au manuel de mini-cab pour apprendre à configurer et utiliser ces cabines.
Actuellement, les cabines suivantes sont disponibles : Y8000 / BB67400 / CC6500 / EAD / ETG / RTG
La capture suivante montre la cabine d'une CC6500 :



6.7. Le logiciel fdcc_tco_net

Ce logiciel permet d'afficher à distance un des 10 TCO de fdcc_pc.py.
Il se connecte à fdcc_pc.py par le réseau informatique, demande le TCO spécifié et l'affiche. Il demande ensuite toute les 250ms les mises à jour et envoie les clicks de la souris.
Bien entendu le logiciel peut aussi tourner sur le même PC que fdcc_pc.py.
Comme le TCO est définit dans le fichier config_tco.txt chargé par fdcc_pc.py et envoyé à fdcc_tco_net.py, il n'y a pas à dupliquer les définitions des TCO.
Le même TCO peut être utilisé simultanément. Toute modification dans l'un se répercute automatiquement dans les autres.
En utilisant les logiciel fdcc_regu (ou les cabines) et fdcc_tco_net, vous pouvez entièrement piloter votre réseau de n'importe quel PC.

La capture suivante montre le logiciel en action (on peut faire tout ce que l'on fait avec les TCO de fdcc_pc.py) :



Pour configurer le logiciel, éditez-le, et modifier les variables suivantes :

```
use_local      = 1
server_ip_adr = "10.20.30.40"
server_port    = 1234
tco_num        = -1
```

`server_ip_adr` indique l'adresse IP du PC ou se trouve fdcc_pc.
- Si les 2 programmes se trouvent sur le même PC :
  Vous pouvez mettre `use_local` à 1.
  Dans ce cas l'adresse IP spécifiée n'est pas utilisée. Le logiciel va la trouver tout seul
  (Vous pouvez aussi spécifier l'adresse IP et utiliser use_local à 0)
- Si les 2 programmes se trouvent sur des PC différent et reliés par le réseau,
  Vous devez mettre `use_local` à 1 et spécifier l'adresse IP

`server_port` indique le port qu'a ouvert fdcc_pc pour les TCO (1234 par defaut)

`tco_num` indique le numéro du TCO que vous voulez utiliser (0-9)
Si vous mettez -1, le programme vous demande quel TCO vous voulez utiliser

Il faut bien entendu lancer fdcc_pc avant de lancer fdcc_tco_net

6.8. Les souris Android avec le logiciel rfo_souris.bas

Il est désormais possible d'utiliser des téléphones et tablettes Android comme souris Wifi afin de contrôler vos locomotives. J'aurai pu réaliser une application Android pour réaliser ces souris, mais l'environnement de développement Android est lourd et la programmation en Java en aurait rebuté plus d'un. Afin de garder les choses simples et vous permettre de modifier facilement ces souris, j'ai choisi d'utiliser l'application RFO basic qui est un interpréteur basic très intéressant pour les appareils Android. Il faudra donc installer l'application RFO basic sur vos appareils Android. Ensuite, il suffira de copier le programme basic « rfo_souris.bas » disponible dans l'archive. Je vous conseille de le copier dans le répertoire rfo-basic/code de vos appareils Android. Lancez ensuite l'application RFO basic, Faire « Load » pour charger le programme « rfo_souris.bas » qui s'affiche alors dans l'éditeur, Profitez-en pour configurer la souris (adresse IP du PC de fdcc-pc, port TCP de fdcc-pc, adresse de la loco), Faire enfin « Run » pour exécuter le programme. La souris virtuelle essaye alors de se connecter au programme fdcc-pc et s'affiche sur l'écran de l'appareil Android si la connexion réussie ! Bien entendu, vous devez avoir connecté votre appareil Android au wifi. Vous pouvez utiliser la box internet de votre habitation ou alors utiliser un point d'accès wifi dédié. Pour ma part j'utilise un routeur wifi TPlink TL-WR481N (18 euros sur Amazon). Le PC de fdcc-pc doit bien entendu être connecté sur ce point d'accès, soit en wifi, soit par le réseau Ethernet. Sur les PC Windows, il faudra certainement désactiver le firewall de l'antivirus pour que les requêtes des souris ne soient pas rejetées.

La figure suivante présente l'architecture du système :

```
Centrale <-USB-> PC avec  <--WIFI-->  Point  <--WIFI--> Tablette/Téléphone
Arduino          FDCC-PC     ou       d'accès           Android avec RFO
                           Ethernet    WIFI              basic et rfo_souris
```

La configuration est à spécifier dans les premières lignes du programme « rfo_souris.bas »

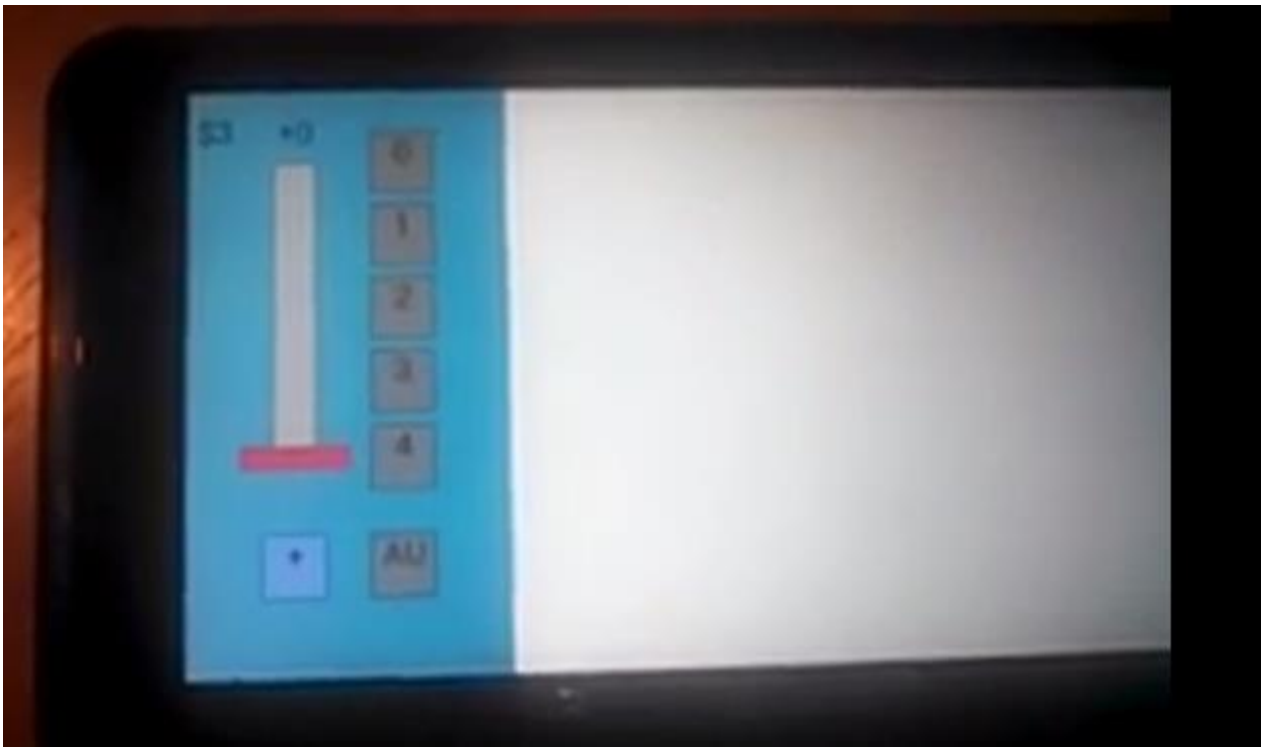
```
! Adresse du serveur FDCC-PC
! ex: "192.168.0.100"
! ex: "" (Dans ce cas le programme demande a l'utilisateur d'indiquer l'adresse IP)
ip$ = "192.168.0.100"

! Port du serveur FDCC-PC
! ex: 1234 (port par defaut de FDCC-PC)
! ex: 0 (Dans ce cas le programme demande a l'utilisateur d'indiquer le port TCP)
port = 1234

! Adresse de la locomotive
! ex: 3
! ex: 0 (Dans ce cas le programme demande a l'utilisateur d'indiquer l'adresse DCC)
dcc_adr = 3
```

Vous pouvez récupérer l'adresse IP et le port TCP de fdcc-pc en regardant dans la console de fdcc-pc. En effet, les lignes suivantes s'affichent :

```
NET: host="udelmasb-MOBL1" IP=192.168.0.100 port=1234
NET: Attente
```

Sur mon réseau local (non relié à Internet), le routeur Wifi attribue l'adresse 192.168.0.100 au premier connecté, puis incrémente cette adresse. 192.168.0.101 au second … Je connecte toujours le PC de fdcc-pc en premier afin d'avoir la même adresse IP. Il est également possible de configurer le point d'accès pour associé une adresse IP à une adresse MAC afin qu'un appareil est toujours la même adresse IP.

La photo suivante montre une souris :



Vous pouvez :

- Changer de locomotive en cliquant dans le coin supérieur gauche
- Modifier la vitesse avec le potentiomètre (crans 0 à 28)
- Changer le sens (bouton +=avant/-=arrière)
- Changer les fonctions spéciales F0 à F4. (F0 est souvent attribuée aux phares)
- Déclencher et arrêter l'arrêt d'urgence.

7. Conclusion

Vous savez maintenant tout sur le système freeDCC 2017. 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. Notez que le système n'est pas encore terminé et sera amené à évoluer.

Bonne réalisation et bon jeu !

Ulysse.

Contacts :
Mail : ulysse.delmas@laposte.net
Site Web : http://udelmas.e-monsite.com/
Logiciels : Disponibles dans l'archive sur le site web (et bientôt dans git hub)
Source des logiciels : Disponibles dans l'archive 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/
- Python: https://www.python.org/
- pySerial: https://pypi.python.org/pypi/pyserial

Limitations actuelles:
- Limitations des logiciels (PC):
  - Les UM ne sont pas gérées
  - Les plaques tournantes ne sont pas gérées
  - Le BMVU n'est pas encore supporté
  - Les logiciels fonctionnent uniquement en Python3 (et pas en Python2)

Annexe A : Le DCC

Afin de vous permettre d'en savoir plus sur le DCC, cette annexe s'étend sur le protocole DCC et décrit les décodeurs.

A.1. Le protocole

Les bases

Je vous conseille de télécharger la norme DCC sur le site de la NMRA http://www.nmra.com
Je sais, elle est en Anglais, mais les figures parlent d'elles-mêmes … Elle décrit les éléments suivants :

** Un **bit 1** est composé d'une tension dans un sens durant 52 à 64 us (microseconde) (Il y a un million de us dans une seconde) puis d'une tension inverse (donc dans l'autre sens durant le même temps). Cette répétition simplifie les décodeurs car les locomotives peuvent être placées dans les 2 sens.

** Un **bit 0** est composé d'une tension dans un sens durant 90 à 10000 us. Puis d'une tension inverse durant 90 à 10000 us. Ces 2 parties n'ont pas besoin de durer le même temps pour un 0, alors que c'est impératif pour un 1.

** Une **trame** est composée de bits
- Un minimum de 10 bits a 1 pour indiquer le début d'une trame
- Un 0 de séparation
- Une adresse pour sélectionner un décodeur (8 bits)
- Un 0 de séparation
- Une commande (1x8bits ou 2x8bits …) (un 0 est ajouter entre chaque groupe de 8 bits)
- Un 0 de séparation
- Une vérification (8bits)
- Un 1 final

Par exemple avec un octet de commande (1 octet = 8 bits) :
1111111111 0 AAAAAAAA 0 CCCCCCCC 0 EEEEEEEE 1

Par exemple avec deux octets de commandes :
1111111111 0 AAAAAAAA 0 CCCCCCCC 0 CCCCCCCC 0 EEEEEEEE 1
Les A, C et E peuvent être à 0 ou 1.
Le nombre d'octets de commande dépend de certains bits C du premier octet.

Les bits de vérifications se calcul en faisant un OU EXCLUSIF entre chaque bit A et C. En clair ils valent 1 s'il y a un nombre impair de bits a 1.
Par exemple avec 1 octet de commande
Si AAAAAAAA = 00001111 et CCCCCCCC = 01000111
Alors EEEEEEEE = 01001000
C'est grâce à ce mécanisme que le décodeur sait ou non si la trame transmise est correcte. Si oui, il effectue l'action demandée. Sinon, il ne fait rien. Essayer de modifier 1 bit A ou C, et vous verrez que le E calculé ne colle pas.
On voit également que grâce aux 0 entre les octets, il n'est pas possible d'avoir plus de 8 bits à 1 hors du préambule. Donc la détection du début de trame est aisée.

Les 0 et 1 sont biens pour des ordres a 2 positions (ex : phares activés ou pas), mais on a besoin de transmettre des nombres comme l'adresse du décodeur ou encore la vitesse. Pour ce faire on utilise le principe suivant :

Avec 1 bit il est possible de coder 2 valeurs (0 ou 1)
Avec 2 bits il est possible de coder 4 valeurs (00=0, 01=1, 10=2, 11=3)
Avec 3 bits il est possible de coder 8 valeurs (000=0, 001=1, 010=2, 011=3, 100=4, …111=7)
Avec 4 bits il est possible de coder 16 valeurs (0 à 15)
Avec 5 bits il est possible de coder 32 valeurs (0 à 31)
Avec 6 bits il est possible de coder 64 valeurs (0 à 63)
Avec 7 bits il est possible de coder 128 valeurs (0 à 127)

Avec 8 bits il est possible de coder 256valeurs (0 à 255)
…

Pour cela on donne des poids aux bits. Le plus a droite à le poids le plus faible 1 (sa contribution est de 0 lorsqu'il est à 0 et de 1 lorsqu'il est à 1). Ce bit est connu sous le nom de LSB (Less Significant Bit=Bit le moins significatif). Le bit immédiatement à gauche du LSB a un poids de 2, le précédent de 4, le précédent de 8 … jusqu'au MSB (Most significant Bit= Bit le plus significatif car il a le plus gros poids).
Par exemple 10011011 = 1*128+0*64+0*32+1*16+1*8+0*4+1*2+1*1=155

Les trames sont transmises en permanence car à chaque perte de contact électrique, les décodeurs doivent recevoir les ordres au plus vite afin que l'effet de la perte de contact passe inaperçu. Par exemple lorsque la locomotive passe sur un rail encrassé, il n'y a plus de contact électrique, donc le décodeur n'est plus alimenté et perd donc ses ordres. L'inertie permet de dépasser la saleté, ce qui réalimente le décodeur, mais celui-ci ne possède plus l'ordre de vitesse qui doit donc arriver au plus vite pour passer inaperçu. Je considère qu'il est nécessaire d'envoyer un ordre sens-vitesse environ 4 fois par seconde. De toute façon, en dessous, la réactivité est plutôt mauvaise.

Comme les 0 durent plus longtemps que les 1, une trame à 1 octet de commande dure le plus longtemps si : 1111111111 0 00000000 0 00000000 0 00000000 1 = 11*1 et 27*0. Dans mon cas la centrale utilise 2*58us pour les 1 et 2*120us pour les 0 et 10 bits de préambule, donc une trame a 1 octet dure au maximum : 11*2*58+27*2*120=7756us. Ce qui fait donc 1000000us/7756us=128 trames max par seconde et donc 32 locos max si on transmet 4 fois par seconde. Ma centrale limite ce nombre a 16 car elle ne transmet pas que l'ordre sens-vitesse mais aussi des ordres pour les fonctions spéciales. Vous remarquerez qu'il aurait été préférable de complémenter les bits d'erreur mais les créateurs de la norme n'ont pas vu cette astuce qui aurait permis de gagner du temps.

La norme prévoyait à l'origine 128 adresses pour les locomotives, mais vous constatez déjà que ce nombre est irréalisable, sans parler de la consommation électrique. 128 décodeurs à l'arrêt mettraient déjà à genou le booster. (ex : 20mA par exemple * 128 = 2.56A à comparer aux booster de 3A couramment utilises …). La norme à quelques incohérences comme celle-là. Alors que dire de l'extension a 1024 adresses …

Les adresses sont codées sur 8 bits donc (0 à 255) et j'indique que la norme en prévoit 128 ! Ce n'est pas une erreur car les autres adresses sont attribuées à d'autres décodeurs comme les décodeurs fixes d'accessoires utilisés notamment pour la commande des aiguillages. Je n'utilise pas ce type de décodeur car le protocole est assez mal fait, en effet il faut envoyer une trame pour changer l'état d'une sortie, ce qui se révèle vite ingérable.

A noter que l'adresse 0 permet d'envoyer un ordre à tous les décodeurs. Ceci peut être utile pour un arrêt d'urgence généralisé (quoi qu'une coupure de courant fasse la même chose). La norme ne prévoit donc non pas 128 mais 127 locomotives.

A propos d'adresse spéciale la 255 (les 8 bits à 1) n'est destinée à aucun décodeur. On dit que c'est une trame de bourrage. Pour ma part, je les utilise s'il y a moins de 16 locomotives.

Vous en savez maintenant t assez pour regarder dans le détail les différentes trames

La trame de sens-vitesse

La norme actuelle permet d'utiliser des décodeurs à 14 ou 28 crans de vitesse. Il est en outre possible d'utiliser en complément une commande à 127 pas sur certains décodeurs. Si le décodeur supporte plus d'un mode alors il est possible de sélectionner le mode à utiliser par programmation.

La première norme prévoyait 14 vitesses et la trame se pressentait de la façon suivante :
1111111111 0 0AAAAAAA 0 01DFSSSS 0 EEEEEEEE 1
Les bits de l'octet de commande se définissent comme suit :
D : Direction (0 arrière, 1 avant)
F : Allumage ou extinction des feux (phares)
S : Pas de Vitesse à atteindre

0000 = 0 = arrêt
0001 = 1 = arrêt immédiat
0010 = 2 = pas de vitesse 1
…
1111 = 15 = pas de vitesse 14
Les pas de vitesse sont définis individuellement par programmation ou alors calculés à partir de la vitesse min et max. La norme prévoit la possibilité d'utiliser des profils pour l'accélération et la décélération. Ainsi lorsqu'un cran de vitesse est demande, la locomotive accélère jusqu'à l'atteindre. Idem pour la décélération. On comprend ainsi le stop immédiat, sorte d'arrêt d'urgence qui n'utilise pas le profil de décélération mais arrête immédiatement la locomotive.

L'évolution de la norme a permis de rajouter un nouveau mode possédant 28 crans en utilisant le bit des phares pour un cran intermédiaire entre les vitesses. La commande des phares passant alors dans le groupe 1 qui contient aussi 4 sorties auxiliaires. Ainsi l'octet de commande prit cette forme : 0-1-D-S0-S4-S3-S2-S1
00000 ou 10000 = arrêt
00001 ou 10001 = arrêt immédiat
00010 = cran 1
10010 = cran 2
…
11111 = cran 28

Enfin il est possible d'utiliser 127 crans sur les derniers décodeurs. Ces crans ne sont pas configurables individuellement mais utilisent la vitesse min pour le 1 et max pour 127. La trame comporte 2 octets de commande.
1111111111 0 0AAAAAAA 0 00111111 0 DSSSSSSS 0 EEEEEEEE 1
SSSSSSS = 0000000 = 0 = arrêt
SSSSSSS = 0000001 = 1 = cran 1 = vitesse min
…
SSSSSSS = 0000000 = 0 = cran 127 = vitesse max
Cette trame est différente de la trame habituelle donc il n'y a pas besoin d'indiquer au décodeur d'utiliser ce mode à condition qu'il le supporte. Il est donc possible d'utiliser ce mode en complément des 2 autres. Mais je pense que ce mode n'a pas grand intérêt car 28 pas configurables suffisent amplement ! D'ailleurs peu de centrales utilisent ce mode.

Les trames des fonctions spéciales (group 1 et 2)

Grâce au digital, il est possible d'activer des fonctions spéciales à utilisait pour ce que bon vous semble comme allumer les phares individuellement, la cabine, les ventilateurs… Il existe même un locotracteur du commerce qui permet d'atteler et dételer a volonté. Enfin les nouveaux décodeurs sonores utilisent aussi ces fonctions pour simuler le démarrage du moteur, les avertisseurs … Il existe aussi des décodeurs de fonctions (sans moteur) qui proposent uniquement des sorties spéciales. Ils sont destinés à être placés dans les voitures ou remorques pour par exemple gérer l'éclairage …

La commande group 1 contrôle 4 fonctions spéciales F1 à F4, plus la commande des feux en mode 28 vitesses. La commande d'allumage des phares appelée FL ou F0 permet d'allumer ou pas les phares. Le décodeur activant 2 sorties suivant le sens. Mais rien ne vous empêche d'utiliser les autres sorties pour contrôler les feux individuellement…
Voici la trame du groupe 1
1111111111 0 0AAAAAAA 0 100-FL-F4-F3-F2-F1 0 EEEEEEEE 1

Le groupe 2 propose quant à lui 4 sorties supplémentaires F5 à F8
1111111111 0 0AAAAAAA 0 1011-F8-F7-F6-F5 0 EEEEEEEE 1

Enfin notons une extension future qui permettra de gérer 13 sorties :
11111111 0 0AAAAAAA 0 110FFFFF 0 FFFFFFFF 0 EEEEEEEE 1

La trame de programmation

Afin de correspondre exactement aux souhaits des utilisateurs, les décodeurs sont configurables. Il suffit de programmer ce que l'on appelle des variables de configuration dites CV. Chaque variable est constituée d'un octet. Le CV le plus important est le 1 qui contient l'adresse du décodeur.

Les CV sont numérotés de 1 à 1024, mais rassurer vous, vous n'aurez pas 1024 paramètres à définir car seul un petit nombre est utilisé en réalité. Les fabricants de décodeurs se doivent d'implémenter les indispensable, ne sont pas obligé d'implémenter les optionnels et peuvent même en définir de nouveaux. C'est pourquoi, il est conseillé de bien lire la fiche technique du décodeur avant de le programmer. Bien entendu, les CV sont sauvés dans une mémoire EEPROM ou FLASH qui à la bonne idée de garder son contenu même lorsque le décodeur n'est pas alimenté.

Voici les principaux CV :

CV	Nom	Description
1	Adresse	1-127
2	Point de démarrage	0-255 (pour que la loco démarre au cran 1)
5	Vitesse max	0-255
3	Inertie à l'accélération	0-255
4	Inertie à la décélération	0-255
29	Configuration	

La norme n'a vraiment pas fait dans la simplicité quant à la programmation des CV, en effet on ne retrouve pas moins de 5 méthodes pour programmer les décodeurs, ce qui est complètement idiot car cela complique les décodeurs, la compatibilité en programmation centrale/décodeur et surtout ce n'était pas justifié !

Nous allons tout de suite abandonner la programmation POM « Programmation On Main track » (programmation sur voie principale) car elle ne permet pas de modifier l'adresse de la locomotive, ce qui vous en conviendrez est très embattant !

Nous nous concentrerons donc sur les programmations POL « Programmation On Learning track » (programmation sur voie d'apprentissage / de programmation). Ces méthodes sont aussi appelées « service mode ». Parmi ces méthodes, la programmation par page est la plus utilisée, c'est donc celle-ci que nous utiliserons. Les méthodes POL ont l'inconvénient de programmer toutes les locomotives présentes sur le réseau ! Pour pallier ce PB, les centrales disposent souvent d'une sortie à connecter sur une voie de programmation où vous programmerez vos décodeurs. Une autre solution est d'utiliser un interrupteur qui permet d'isoler le réseau à l'exception d'une voie dévolue à la programmation lors des opérations de programmation mais gare à l'oublie !

La trame de programmation est la suivante :
1111111111 0 011110RRR 0 DDDDDDDD 0 EEEEEEEE 1
Les bits D forment l'octet à programmer tandis que les bits R indiquent le registre.

Comme vous avez bien suivit vous allez me dire que 3 bits autorisent 8 registres au maximum (R0 à R7) et non 1024. L'astuce vient du registre R5 qui contient le numéro de page. Il suffit donc de choisir sa page avec R5 (256 choix, 00000000=0 pour la page 1, 1 pour la page 2, 255 pour la page 256) puis de programmer les registres R0 à R3. Ainsi pour programmer le CV1 il faut sélectionner la première page (mettre 000000000 dans R5), puis écrire dans R0
CV1 P1/R0 (P1=0 dans R5)
CV2 P1/R1
CV3 P1/R2
CV4 P1/R3
CV5 P2/R0 (P2=1 dans R5)
...

Afin de ne pas programmer le décodeur par erreur les protections suivantes ont été rajoutées :
- Le préambule passe maintenant à 10 bits au minimum
- La même trame de programmation doit être reçue identiquement 5 fois au minimum
- Des trames de reset doivent être judicieusement interposées à des endroits précis

Voici par exemple la succession de trames utilisées par Free-DCC

```
unsigned char dcc_prog_seq[] = { \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
\
DCC_PROG_FRAME_PAGE, \
DCC_PROG_FRAME_PAGE, \
DCC_PROG_FRAME_PAGE, \
DCC_PROG_FRAME_PAGE, \
DCC_PROG_FRAME_PAGE, \
DCC_PROG_FRAME_PAGE, \
\
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
\
DCC_PROG_FRAME_DAT, \
DCC_PROG_FRAME_DAT, \
DCC_PROG_FRAME_DAT, \
DCC_PROG_FRAME_DAT, \
DCC_PROG_FRAME_DAT, \
DCC_PROG_FRAME_DAT, \
\
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
DCC_PROG_FRAME_RESET, \
};
```

Pour indiquer que la programmation s'est bien déroulée, le décodeur doit augmenter sa consommation de courant, par exemple en activant brièvement le moteur. Cette surconsommation peut alors être détectée par la centrale pour vérification, mais ce n'est pas obligation pour la centrale.

La Gestion des sorties

Maintenant que vous savez tout sur la commande des locomotives, voyons comment la norme à décider de piloter les accessoires que sont les aiguillages, la signalisation …

Afin de contrôler des sorties comme des aiguillages ou feux, la norme prévoit de commander par le signal DCC des décodeurs d'accessoires de 8 sorties.

La trame de commande est la suivante :
11111111 0 10AAAAAA 0 1AAADSSS 0 EEEEEEEE 1
AAAAAA-AAA est l'adresse du module (512 modules de 8 sorties)
D est la valeur 0/1 à appliquer à la sortie SSS
(000=première sortie … 001=seconde sortie … 111=8éme sortie)

Ces sorties sont souvent couplées en 4 ensembles de 2 sorties, en effet un aiguillage qui possède 2 bobines a besoin de 2 sorties, de même qu'un feu vert/rouge Lorsque les sorties sont couplées
000=aiguillage droit ou feu vert
001=aiguillage dévié ou feu rouge
En mode couplé, le fait d'activer une sortie désactive la sortie complémentaire.
Il est fréquent de couper une section de voie lorsque le signal est rouge afin d'arrêter les trains (violemment) devant le signal.

Les décodeurs d'accessoires disposent également de CV afin de coupler les sorties, définir des temps d'activation. Ces temps sont utiles pour envoyer des impulsions précises aux aiguillages. Dans ces conditions, il n'est pas utile d'envoyer un ordre de mise à 0 de la sortie car elle revient à 0 après l'impulsion. Ces CV ne se programment pas avec les mêmes trames de programmations que celle des décodeurs de locomotive.

La gestion des accessoires n'est pas géniale car elle nécessite l'envoie d'une trame chaque fois que l'on veut changer une sortie ! Ce qui surcharge la bande passante du DCC. De plus comme les accessoires ne sont pas mobiles et donc il n'est pas vraiment justifié de les commander par le signal DCC.

PS : Pour ces raisons je n'utiliserai pas ce type de commande dans ma réalisation, étant donné qu'il faudra utiliser un bus pour les entrées, je mettrai également les sorties sur un bus. Je ne détaille donc pas d'avantage cette partie de la norme.

La Gestion des entrées

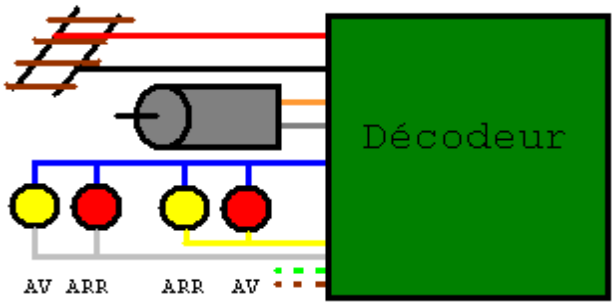
Nous savons maintenant piloter un réseau (locomotives, aiguillages, feux), mais comment connaître son état (occupation des cantons, passage des trains sur des ILS ou pédale de voie …) ?

Ce chapitre va être court car la norme n'a absolument rien prévu ! Conséquence chaque constructeur de centrale propose sa propre solution. La plus populaire est l'utilisation de modules de rétro-signalisations connectés sur un bus S88. Free-DCC utilisait ce bus, mais par soucis de simplicité, les entrées se connectent directement à la centrale pour free-dcc-2008.

A.2. Les décodeurs

Présentation des décodeurs

Vu de l'extérieur, un décodeur est une platine électronique de laquelle sorte des fils. Parfois ces fils vont sur une prise plus ou moins normalisée qui peut même être présente dans les locomotives récentes. Sinon, il faudra souder directement ces fils aux bons endroits. La couleur des fils est en principe normalisée, mais il vaut mieux lire la notice.



Pour les installations les plus simples il suffit de relier 2 fils (rouge et noir) aux palpeurs de courant et 2 autres (orange ou gris) au moteur. Les autres servants aux phares et fonctions spéciales. Les fils de captages peuvent être connecté au pif bien que la norme indique le contraire. Par contre les fils du moteur influent sur le sens de rotation du moteur. Il est à noter qu'en cas d'erreur, ce sens peut être inverse grâce aux CV. Donc pas d'inquiétude.

Pour les phares et fonctions spéciales un fil (bleu) fournit du 12V. Les autres fils des phares et fonctions spéciales sont reliés à des transistors qui mettent ou non à la masse le fil correspondant. Ainsi lorsque la fonction est activée, le fil est mis a la masse (sinon, il est en l'air) et la lampe entre 12V et la masse s'éclaire. Normalement le fil blanc utilisé pour les phares dans le sens avant, est activé lorsque FL=1 et sens=avant. Idem pour le jaune mais en marche arrière. Les fils des autres couleurs servent aux fonctions spéciales. Sur certains décodeurs, il est possible de redéfinir les sorties phares en fonction spéciales.

Le choix d'un décodeur se fait suivant les caractéristiques suivantes
- norme (ici DCC)
- connecteur
- taille
- prix
- courant pouvant être délivré au moteur / courant pouvant être utilise pour chaque fonction spéciale.
- existence des sorties phares / nombre de sorties spéciales (en plus des sorties phares)

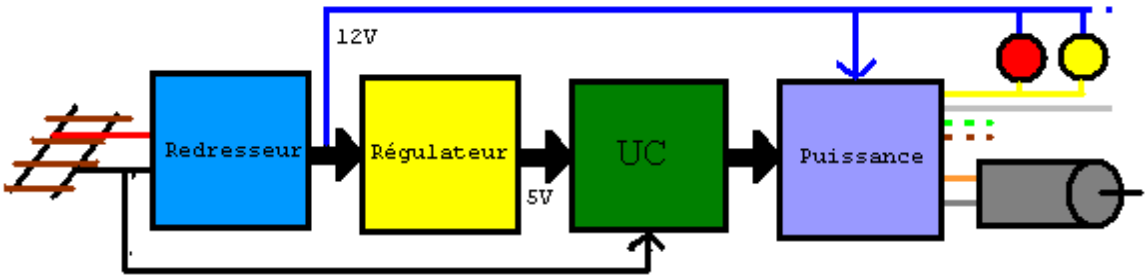
Il existe dans le commerce toute sorte de décodeur allant de 15 à 120 euros. Les plus simples gèrent uniquement le moteur, Les plus complexes gèrent en plus de multiples sorties spéciales ainsi que le son.

Le choix du connecteur est important, voici quelques connecteurs courants :

- Connecteur à 6 broches pour le N : Il suffit juste de l'enficher dans le connecteur de la locomotive
- Connecteur à 8 broches pour le HO : C'est le cas le plus courant. Ce connecteur est relié au décodeur par des fils. Il convient de le brancher dans le connecteur 8 broches de la locomotive s'il existe.
- Connecteur à 21 broches pour le HO : A la différence du connecteur à 8 broches, le connecteur est directement soudé au décodeur (comme pour le N). Il suffit juste de clipser le décodeur sur le connecteur 21 broches de la locomotive. Les matériels récents utilisent de plus en plus ce type de connecteur, car il n'y a pas de fils à la différence du connecteur 8 broches.

Outre les décodeurs de locomotive, il existe également :
- Des décodeurs d'accessoires pour les aiguillages, la signalisation …
- Des décodeurs de fonctions qui sont des décodeurs de locomotive sans la gestion du moteur. Ils peuvent par exemple être utilisés pour contrôler l'éclairage des voitures, les feux de fin de convoie …

Structure des décodeurs



Un décodeur est constitué des éléments suivants :

- Un redresseur qui transforme le + ou – 15V de la voie en une tension continue d'un peu plus de +12V compte tenu des chutes de tension dans les diodes. Ainsi, quel que soit la polarité de la voie, le décodeur est alimenté en 12V.

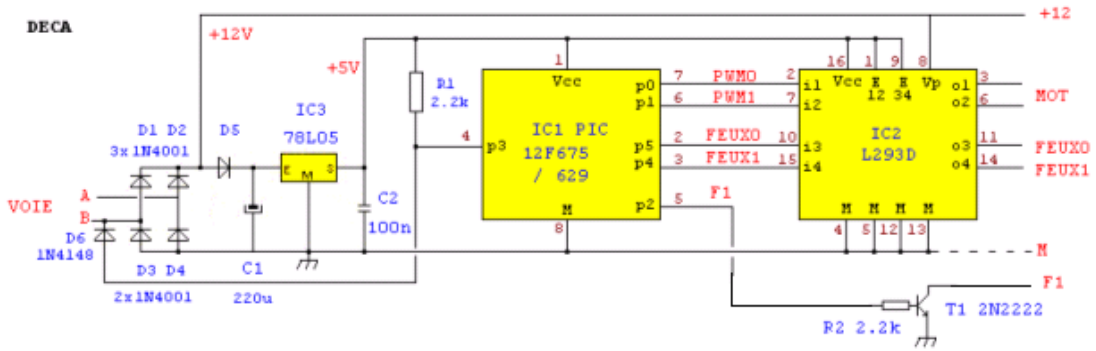
- Un régulateur qui transforme le +12V en +5V bien propre, ce qui est nécessaire pour alimenter certains circuits électroniques comme le microcontrôleur (UC). Un condensateur peut être rajouté pour garder un peu d'énergie pendant les microcoupures et alimenter le UC.

- Le microcontrôleur commande la partie puissance en fonction des trames qu'il reçoit. Ces trames sont formées de bits récupérés grâce au fil avant le redresseur qui permet de connaître la polarité relative du signal. Le UC est un composant qui contient un petit processeur, de la mémoire et des entrées/sorties. Il exécute un programme qui permet de donner vie au décodeur.

- La partie puissance amplifie les signaux du UC pour piloter le moteur, les phares et les fonctions spéciales. Le moteur est piloté par un composant qui s'appelle pont en H et qui permet de mette ou pas le 12V aux bornes du moteur et ceci dans un sens ou l'autre.

Il est à noter que le moteur est piloté en PWM pour éviter les échauffements. En PWM, on génère régulièrement des impulsions plus ou moins longues. Le nombre d'impulsion par seconde s'appelle fréquence et s'exprime en hertz (hz). Par exemple à 50Hz, donc 50 impulsions par secondes, la durée de l'impulsion peut varier de 0 à 1/50=0.02s=20ms. La PWM permet d'obtenir des ralentis très lents car le couple instantané est toujours au maximum. Bien entendu, jusqu'à 20kHz, les fréquences s'entendent. Certains prétendent qu'il est préférable d'alimenter les moteurs a fer doux à plus de 20kHz pour ne pas les détruire, mais ceci est plus une légende qu'autre chose. Le léger bruit des moteurs n'est pas vraiment gênant. Quelques décodeurs permettent d'ajuster cette fréquence grâce à un CV.

La figure suivante présente le schéma électrique d'un décodeur personnel, mais vu le temps de réalisation et la taille vaut mieux opter pour des modèles du commerce.



Annexe B : Electronique

Cette annexe devrait permettre à ceux qui ne sont pas très familier avec l'électronique, de comprendre la réalisation. Une première partie rappelle quelques notions d'électronique. Une seconde présente les alimentations. Enfin, une dernière, décrit composant utilisés.

B.1. Notions d'électronique

Afin de comprendre les explications électriques sur la réalisation, il est nécessaire d'avoir un minimum de connaissances en électronique. Et comme un modélisme ferroviaire n'est pas forcément un électronicien je vais rappeler quelques notions et essayer de faire sentir quelques règles. Si vous avez déjà des notions, vous pouvez sauter ce chapitre. Je n'utiliserai pas forcement des définitions rigoureuses, le but étant de faire comprendre et sentir en un minimum de temps.

Tension, Courant, Puissance, Energie :

L'électricité traite des charges électriques, mais malheureusement ces charges sont invisibles c'est pourquoi il est intéressant de faire l'analogie avec l'hydraulique qui est l'étude des liquides ou des gouttes d'eau dans notre cas. Il était une fois une charge et une goutte d'eau … Pour déplacer de l'eau par exemple pour faire tourner la roue d'un moulin, il faut une différence de hauteur. Et bien c'est pareil en électricité, il faut une différence de potentiel encore appelé voltage dont l'unité est le volt V.

Une autre grandeur importante en hydraulique est le débit qui est la quantité de liquide qui passe à un endroit en un temps donné. Ce débit s'exprime par exemple en mètre cube par seconde ou litre par minute … Ce débit, encore appelé courant peut facilement être transposé dans le domaine électrique, il représente alors le nombre de charge qui passent dans une section dans un temps donné. Son unité est l'ampère A. En électronique, on parle souvent de mA car l'ampère est une grosse unité. Il faut 1000mA pour faire un ampère. 1mA=0.001A ; 1A=1000mA.

La puissance est liée à la hauteur et au débit, en effet plus quelque chose tombe de haut et plus il en tombe, plus cela fait mal. Transposé à l'électricité, on en déduit la formule de la puissance : P=UI. La puissance s'exprime en Watt W. Par exemple une locomotive qui consomme 500mA sous 12V consomme P=UI=12*0.500=6W. La puissance peut aussi s'exprimer en chevaux avec la règle de conversion suivante (1kw = environ 1.3ch)

Tant qu'on y est, parlons de l'énergie. L'énergie permet de faire quelque chose. On voit de suite qu'elle est liée à la puissance consommée et au temps d'utilisation. D'où : E=Pt. Elle s'exprime en Watt Heure Wh. Attention pas Watt par heure mais Watt heure. Par exemple, une batterie de voiture de 12V chargée possède une énergie de 600Wh. Elle pourrait donc alimenter un moteur consommant 1A sous 12V pendant : E=Pt ; t=E/P ; t=E/(UI) ; t=600/(12*1)=50 heures.

Une petite application pratique au fils.
Plus la tension est grande et plus il faut d'isolant. Pour vous en convaincre, comparez l'épaisseur des isolateurs des supports de caténaire 1500V et 25000V.
Plus l'intensité est grande et plus le diamètre doit être important. Regardez la lourde caténaire 1500V et la légère caténaire 25000V.
Calculons le courant qui y passe lorsqu'une machine type BB22200 de 4400kW (5600ch) circule à pleine puissance P=UI ; I=P/U :
- sous 1500V : I=4400000/1500=2933 A
- sous 25000V : I=4400000/25000= 176A.

Résistance :

Une résistance est un composant à 2 pattes qui lie le courant et la tension avec la loi suivante : U=RI. R est sa résistance qui s'exprime en ohm et son symbole est oméga Ω. Une résistance transforme la puissance en chaleur. A notre niveau, nous verrons les moteurs et lampes comme des résistances mais transformant respectivement l'énergie en mouvement mécanique et lumière. En analogie électrique, une résistance peut se voir comme un tuyau,

plus le tuyau est gros (et donc sa résistance au passage de l'eau faible) et plus le débit est important pour une différence de hauteur donnée.

Par exemple une lampe qui consomme 100mA sous 12V à une résistance interne de U=RI ; R=U/I=12/0.100=120 Ohms.

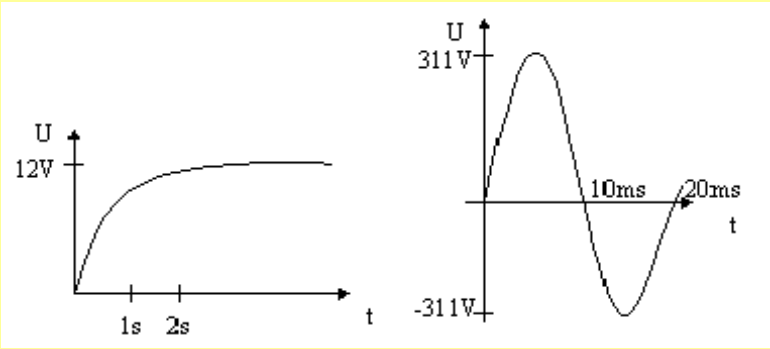
Sachant que le corps humain mouillé est assimilable à une résistance d'environ 1000Ohms et qu'un courant de 30mA le traversant est mortel, il vaut mieux ne pas s'exposer à plus de U=RI=1000*0.03=30V. (48V en alternatif). Donc pas de problème en modélisme ferroviaire ou les faibles tensions comme 12V sont inoffensives.

Maintenant calculons, le courant dans un fils de 0.1Ohms que l'on utilise pour court-circuiter une batterie de 12V. U=RI ; I=U/R=12/0.1=120A Le fil crame et souvent la voiture qui va avec aussi d'où l'intérêt des fusibles.

Calculons la résistance à mettre en série avec une led qui doit être alimentée en 2V sous 10mA si on dispose d'une alimentation de 12V. La résistance doit donc avoir 12-2=10V a ses bornes et est également traversée par 10mA, d'où R=U/I=10/0.010=1000Ohms. La puissance dissipée dans la résistance est de P=UI=10*0.010=0.1W donc une résistance standard d'un quart de watt 0.250 suffit.

Diagrammes temporels :

Un diagramme temporel est la représentation d'une grandeur en fonction du temps. Le diagramme suivant montre l'évolution de la tension aux bornes d'un condensateur lorsqu'on le charge à travers une résistance et un signal alternatif comme celui que l'on voit sur le secteur.



On voit par exemple que ce condensateur est chargé à 12V au bout de 2s. La tension du secteur passe a 0V, puis à 311V au bout de 5ms puis à 0V puis à –311V puis à 0V et le cycle recommence. Ce type de signal est périodique car il se reproduit à l'infinie. Cette forme de signal en sinusoïde est représentative du courant alternatif. La sinusoïde se reproduit toute les 20ms, on parle de période de T=20ms et donc de fréquence F=1/T=1/0.020=50Hz.1 hertz Hz correspond à une fois par seconde. On retrouve bien le 50Hz du réseau EDF. Et le alors, ou est le 220V? Et bien c'est la tension en continue qui produirait le même effet. Du point de vue énergétique un radiateur alimenté en alternatif chaufferait de la même manière que s'il était alimenté en continue avec 220V.

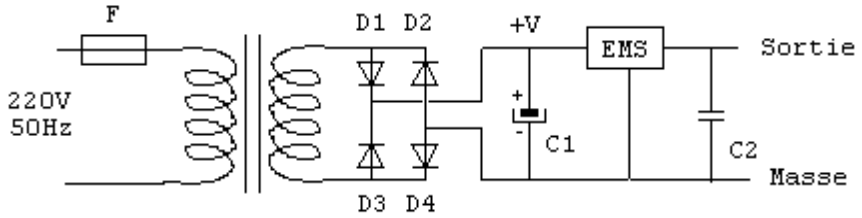
Etats logiques :

En électronique numérique, on préfère travailler avec des états, par exemple en alimentant les circuits en 5V on parle d'état bas ou 0 sous 2.5V et d'état haut ou 1 au-dessus de 2.5V. On parle de logique booléenne ou binaire car il n'y a que 2 états.

B.2. Les alimentations

Tout montage électronique à besoin d'une alimentation pour fonctionner, nous allons voir dans cette partie comment alimenter l'électronique et le réseau. Avant ceci regardons ce qu'il y a dans une alimentation linéaire. (Il existe un autre type d'alimentation : celles à découpage mais elles sortent du cadre de cette documentation).

Eléments d'une alimentation linéaire



Une alimentation ou alim comprend les éléments suivants :

- Fusible : Pour éviter à l'alimentation de prendre feux en cas de problème

- Transformateur : Pour abaisser la tension par exemple de 220V à 15V. Il est possible de s'arrêter ici pour une tension alternative. Lors de l'achat d'un transformateur, il faut spécifier, le type, le nombre d'enroulement du secondaire, la tension primaire, la tension secondaire et la puissance. Le type est la technologie par exemple normal, torique … Un enroulement secondaire est suffisant dans notre cas, mais il peut être intéressant d'en avoir plusieurs pour différentes tensions ou pour utiliser des montages à point milieu. On choisira 220V pour le primaire et une tension secondaire plus grande que la tension de sortie en tenant compte des pertes dans les diodes et le régulateur. Par exemple 15V est un minimum pour une tension de sortie de 12V. Attention, cette tension efficace n'est pas la tension instantanée maximale qui vaut Vmax=1.4Veff soit 21V pour 15Veff. Il faudra en tenir compte pour le choix des composants. Enfin la puissance efficace du transformateur est donnée en VA. Pour faire simple disons que cela correspond à la puissance disponible au secondaire. Si nous voulons 2A, il faudra donc prendre dans notre exemple un transfo de P=UI=2*15=30W soit environ 30VA

- Pont de diode : Pour redresser le courant. Pensez à spécifier la tension inverse et le courant max.

- Condensateur de filtrage : Pour s'approcher du continu. Le condensateur se charge pendant les impulsions et restitue une partie de son énergie lors des trous. Il est possible de s'arrêter ici pour une tension continue non régulée. On prend généralement un condensateur chimique capable de supporter la tension maximale du transformateur et de capacité de 1000uF par ampère.

- Régulateur : Permet de réguler la tension à une valeur fixée. Le régulateur peut être fixe ou ajustable. Sur la plupart des régulateurs, la tension d'entrée devra être plus grande d'au moins 2V que la tension de sortie afin que le régulateur régule convenablement. Le régulateur est souvent protégé contre les courts circuits. (Ce n'est pas le fusible qui grille mais le régulateur qui coupe la sortie). C'est un composant qui chauffe beaucoup car il doit convertir en chaleur la différence de tension entre l'entrée et la sortie multipliée par le courant. C'est pourquoi, on les monte souvent sur des radiateurs.

- Condensateur de découplage pour éliminer les composantes hautes fréquences. Un petit condensateur plastique de 100nF suffit.

B.3. Les composants électroniques utilisés

Cette partie décrit les composants électroniques utilisés. Cela commence avec les composants de base puis se poursuit avec la description des circuits intégrés. Pour plus de détail sur un composant particulier, chercher sa fiche technique ou datasheet en Anglais sur Internet. Beaucoup de fiches sont en Anglais mais l'Anglais technique se comprend facilement et les nombreuses figures sont d'une aide précieuse.

Pour vous procurer les composants vous pouvez :

- Aller au magasin d'électronique de votre ville
- Commander à distance par Internet (sites de revendeurs électroniques ou Amazon)
- Demander des échantillons gratuits sur les sites Web des constructeurs
- Faire de la récupération

4.2.A. Les Composants de base

- **La résistance** déjà présentée précédemment

- **La diode** est un composant à 2 pattes qui ne laisse passer le courant que dans un seul sens. La tension à ses bornes est de 0.6V lorsque le courant passe. L'entrée du courant s'appelle l'anode et la sortie la cathode. La cathode est repérée par un trait.

- **Le pont de diode** est un ensemble de 4 diodes qui permet de redresser un signal, c'est à dire de transformer les tensions négatives en positives (donc en sortie, le courant circule toujours dans le même sens).

- **La LED ou DEL** en Français pour diode électroluminescente. C'est une diode qui s'allume lorsque le courant la traverse. La tension de service est de 2V environ et le courant de 10mA mais cela dépend des modèles. La plus grande patte est le + (anode) et le chanfrein le – (cathode).

- **Le condensateur** est un réservoir d'énergie. Il est utilisé pour stocker de l'énergie temporairement dans les alimentations, faire des oscillateurs, éliminer les parasites. Il est recommandé d'en mettre un de 100nF sur chaque circuit intégré actif. La capacité d'un condensateur s'exprime en Farad F. Cette valeur est énorme c'est pourquoi on parle souvent de microfarad µF ou nano Farad nF (1F=1000000µF=1000000000nF). Les gros condensateurs sont ronds et chimiques et il faut les brancher à l'endroit sous peine d'explosion. (Une bande de – indique la patte -, l'autre patte étant le +). Les petits condensateurs que nous utiliserons seront dit plastiques et ne sont pas polarisés. Enfin lors de l'achat il faut préciser la tension maximale de fonctionnement du condensateur. Par exemple : chimique 1000µF 16V, plastique 100nF 63V…

- **Le transistor** est un composant à 3 pattes qui permet d'amplifier un courant qui passe entre la base et l'émetteur. Le transistor est capable de faire passer un courant maximal entre le collecteur et l'émetteur égale au courant de base (ou de commande) multiplié par le gain du transistor.

- **Le quartz** est un composant qui permet de produire des vibrations très précises.

- **L'oscillateur céramique** : Un quartz un peu moins précis.

- **Le relais électromagnétique** est un ensemble d'interrupteurs qui sont commandés par une bobine. Lorsque la bobine est alimentée, par effet magnétique des interrupteurs se ferment et d'autres s'ouvrent. Cela permet de commander de grosses charges, en isolant la source et la commande.

4.2.B. Les circuits intégrés du système

Les circuits intégrés sont des composants avec de nombreuses pattes reliées à une puce en silicium. Cette puce contient de nombreux composants qui sont souvent des transistors. Généralement ces circuits intégrés ou CI s'alimente avec une tension régulée. Dans notre cas, ce sera du 5V. Les pattes sont numérotées dans le sens des aiguilles d'une montre de 1 jusqu'au nombre de broches. La patte une est souvent indiquée par un petit point. Quand on lit l'inscription horizontalement, elle se trouve en bas à gauche. La figure suivante explicite le principe de numérotation sur le légendaire NE555 qui comporte 8 pattes.

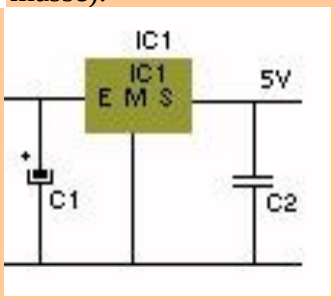
```
  8 7 6 5
 _#_#_#_#_
|          |
 | NE555    |
|o_________|
  # # # #
  1 2 3 4
```

== Les régulateurs == (7805 / 7809 / 7812 / 7815 / LM317T)

Le régulateur est un circuit qui permet de fournir en sortie une tension définit quelle que soit la tension d'entrée. Dans la plupart des cas la tension d'entrée doit tout de même être supérieure de 2V par rapport à la tension de sortie. Il existe des régulateurs fixes qui fournissent une tension définit ou des régulateurs réglables dont on peut choisir la tension de sortie avec un potentiomètre. Ces régulateurs sont dits linéaires et consomment la différence de tension sous forme thermique. Pour cette raison, il convient de les monter sur des radiateurs.

Les lignes suivantes présentent un régulateur fixe 7805 (05 pour 5V) et le régulateur réglable LM317T.

La figure suivante montre comment obtenir du 5V régulé à partir d'une tension continue d'au moins 7V en entrée. On peut prendre C1=1000uF en 16V, C2=100nF, IC1=7805. Attention à ne pas brancher le condensateur de filtrage C1 à l'envers car sinon il explose et dégage une puanteur tenace. Si vous avez peur de brancher le montage à l'envers alors rajoutez une diode en entrée. Pensez à mettre le 7805 sur un radiateur pour qu'il puisse dissiper. Notez que la languette de refroidissement du 7805 (la partie ou il y a le trou qui sert à fixer le 7805 sur un radiateur est reliée électriquement à la patte du milieu, c'est à dire la masse).

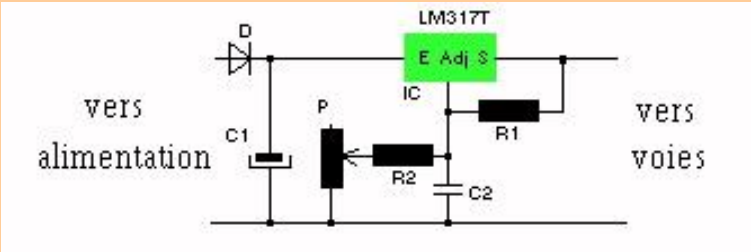


Le 7805 à la forme suivante :

```
|-----|   E = Entrée continue non régulée (doit être >= 7V)
|  O  |
|-----|   M = Masse
|7805 |
|-----|   S = Sortie régulée à 5V pour le 7805
 | | |                         9V pour le 7809
 | | |                        12V pour le 7812 …
 E M S    Imax=1.5A
```

Le courant maximal de sortie est de 1.5A. Le 7805 est protégé contre les courts circuits et contre la surchauffe. Si on note vin la tension d'entrée, Vout la tension de sortie et Is le courant de sortie, alors la dissipation thermique du 7805 en Watt est de P=(Vin-Vout)*Is.

La figure suivante montre l'utilisation d'un régulateur réglable LM317T



Le LM317T à la forme suivante :

```
|-----|   A = Ajustement
|  O  |
|-----|   S = Sortie régulée
|LM317|
|-----|   E = Entrée continue non régulée (doit être >= sortie+2V)
 | | |
 | | |    Imax=1.5A
 A S E
```

Voutmax=1.25+1.25*((R2+P)/R1)
Voutmin=1.25+1.25*(R2/R1)

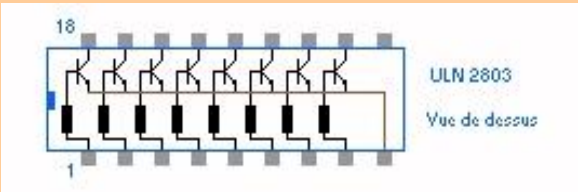
P est un potentiomètre, c'est à dire une résistance de valeur P entre les 2 pattes extrêmes avec une 3eme patte relié au curseur. Ainsi entre une patte et le curseur, la résistance peut être réglée entre 0 et P Ohms. Il existe des potentiomètres linéaires ou rotatifs.

Il est recommandé de choisir R1=220 ohms

Pour plus d'intensité, il est possible d'utiliser des LM317K (3A) ou LM350 (3A), mais les boîtiers sont plus difficiles a utiliser et les régulateurs bien plus chers. Une autre solution est de mettre des LM317T en parallèle. Mais dans ce cas, il faut équilibrer les sorties avec une résistance de 0.2ohm de puissance suffisante.

== Les réseaux de transistors Darlington == (ULN2803)

Ces réseaux de transistors Darlington permettent 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 à 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 'entée correspondante. La figure suivante détaille un ULN2803 :



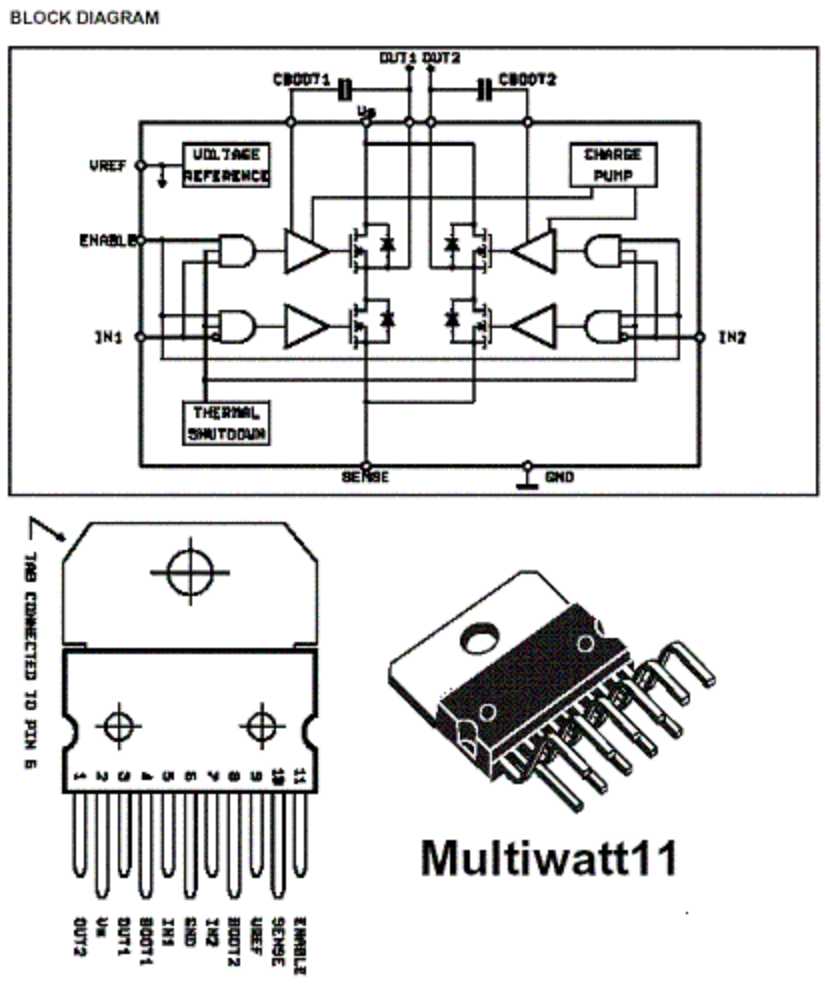
Vous pouvez ainsi connecter des moteurs, lampes, petits aiguillages, relais entre le + d'une alimentation et une des sorties de l'ULN. Ne connectez pas une charge qui consomme plus de 500mA. Utilisez un relais ou un transistor plus puissant dans ce cas. L'ULN ne peut pas commuter des tensions alternatives car les transistors fonctionnent que dans un seul sens, utilisez un relais dans ce cas. Enfin pensez à mettre une diode de roue libre DRL aux bornes de chaque charge inductive (qui contient une bobine comme un relais ou un moteur ou un aiguillage). Une DRL est une diode montée en inverse qui va éliminer la surtension énorme qui se crée lorsque l'on coupe l'alimentation d'une charge inductive. L'anode de la diode (+) est branchée sur la sortie et la cathode (-) sur l'alimentation.

== Les ponts en H == (L6203 / LMD18200D / L293D)

Les ponts en H sont constitués de 4 transistors de puissance. T1, T2, T3 et T4. Ils sont principalement destinés à commander des moteurs. En effet lorsque T1 et T4 conduisent simultanément alors le courant traverse la moteur MOT dans le sens de A vers B et donc le moteur tourne dans un sens. Lorsque T3 et T2 conduisent, le phénomène inverse opère.

```
+--------------+--- +Puissance
|              |
T1             T3
|              |
+---A MOT B----+
|              |
T2             T4
|              |
+--------------+--- masse
```

Le L6203 :



Le L6203 est un pont en H 3A pouvant être utilisé dans un booster. Il est un peu plus simple à comprendre que le LMD18200T, c'est pourquoi je le présente mais je recommande d'utiliser le LMD18200T.

IN1 contrôle le premier demi-pont (T1/T2), tandis qu'IN2 se charge du second (T3/T4). Ceci uniquement lorsque l'entrée d'activation générale est à l'état haut que nous pouvons connecter au +5V. Le circuit s'alimente entre sa

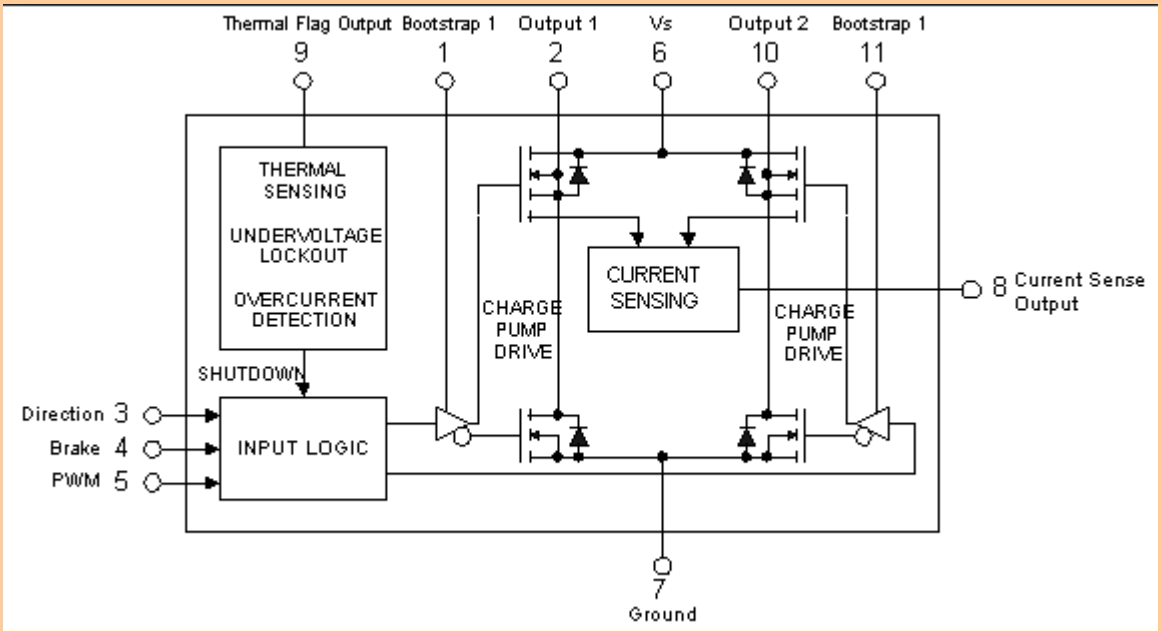
broche Vs et la masse. Afin de pouvoir mesurer le courant il est possible de mettre une résistance entre la sortie SENSE et la masse.

Pour la logique le circuit dispose de son propre régulateur, mais il faut lui rajouter une petite capacité de 220nF sur la patte Vref. Les sorties sont disponibles sur les pattes OUT1 et 2. Il est conseillé de relier les sorties aux pattes BOOST1 et 2 par l'intermédiaire de condensateur de 15nF afin d'augmenter la vitesse de commutation des transistors.

Les transistors sont des MOS de puissance, c'est-à-dire qu'ils se comportent comme une énorme résistance lorsqu'ils ne conduisent pas et comme une très faible résistance lorsqu'ils conduisent. La résistance à l'état passant est de 0.3ohm ce qui provoque une chute de tension de 2*3*0.3=1.8V à 3A. La tension de sortie variera donc légèrement en fonction de la charge du booster. Ces résistances sont des aubaines pour mettre des boosters en parallèle car elles équilibrent les charges.

Il convient de monter le circuit sur radiateur, car en fonctionnement normal (hors court-circuit), il peut dissiper jusqu'à P=U*I=1.8*3=5.4W. Signalons que le circuit dispose d'une protection contre la surchauffe ce qui l'empêche de partir en fumée en cas de problèmes. Comme le circuit ne possède pas de protection de courant, il est préférable de le précéder d'un régulateur. Cette solution permettra également de régler la tension de sortie aux alentours de 15V en HO et 12V en N

Le LMD18200T :

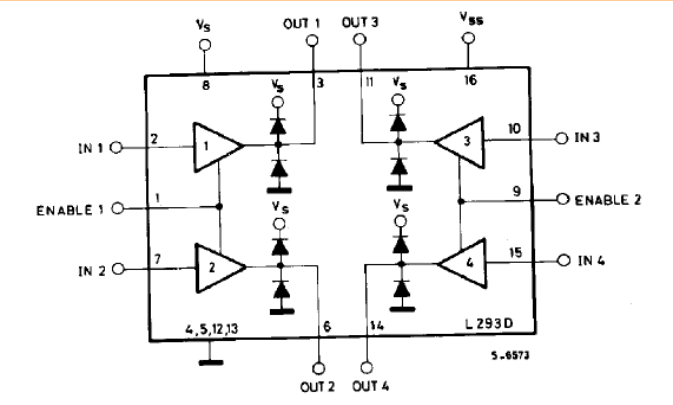


La commande de ce circuit est un peu différente : L'entrée direction sélectionne soit T1/T4, soit T2/T3. Mais ces transistors ne s'activent que lorsque l'entrée PWM est à 1. L'entrée « brake » (frein) permet de court-circuiter le moteur (par exemple avec T2/T4 pour le freiner plus rapidement qu'en le laissant en roue libre.

Dans le cas du DCC, ce composant est plus facile à utiliser car il suffit d'injecter le signal sur DIR et mettre PWM à 1 et « brake » à 0. Alors que dans le cas du L6203, il faut injecter le signal DCC sur IN1 et son complément sur IN2. De plus ce circuit dispose d'une protection de courant et d'une sortie pouvant être utilisée pour mesurer le courant.

Le L293D :

Pour appliquer aux aiguillages la tension accessoires, nous utiliserons les demi ponts (ex : TI/T2 ou T3/T4) comme de simples interrupteurs avec le plus (avec une diode en sortie pour ne pas être embêté par la masse). Ceci simplifiera énormément le montage car contrairement aux interrupteurs avec la masse, il est difficile de trouver des équivalents à l'ULN2830 pour les alimentations. Le composant retenu est le L273D de 700mA qui possède 2 ponts soit 4 demi-ponts.



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. La partie logique du circuit s'alimente par Vss que l'on mettra à 5V pour être compatible avec les autres CI de la carte. Toutes les entrées (IN1 à 4 et enable 1 et 2) doivent donc avoir des tensions entre 0 et 5V. La partie puissance est alimentée par Vs (patte 8). C'est sur cette patte qu'il faudra mettre le + de l'alimentation accessoires.

Enfin pour terminer la description de ce double pont en H, il est à signaler que les 4 broches centrales permettent d'évacuer la chaleur du composant, il faudra donc veiller à faire un gros pâté de soudure sous ces pattes. Mais vu le faible temps de manœuvre de s aiguillages, il ne devrait pas chauffer.

De toute façon, il ne risque pas de partir en fumée car il est protégé contre la surchauffe (Les sorties sont alors désactivées le temps qu'il refroidisse). Ce composant est également protégé contre les court-circuits.

== Le driver de LEDs == (MAX7219)

Les MAX7219 ou 7221 permettent de piloter 64 LEDs.

```
DO SG SDPSE SC +5 I  SG SB SF SA CLK
24 23 22 21 20 19 18 17 16 15 14 13
 #  #  #  #  #  #  #  #  #  #  #  #
#####################################
 #       MAX7219 / MAX7221          #
#####################################
 #  #  #  #  #  #  #  #  #  #  #  #
01 02 03 04 05 06 07 08 09 10 11 12
DI L0 L4 MM L6 L2 L3 L7 MM L5 M1 LD(CS\)
```

+led: SEGA,B,C,D,E,F,G,DP
-led: L0,1,2,3,4,5,6,7
données prisent en compte sur front montant de CLK (D15 a D0)
- validation par pulse positive LD (7219)
- mettre CS\ a 0 durant transmission (7221)
Réglage de l'intensité par registre et R entre `+5 et I (ex 10k)`

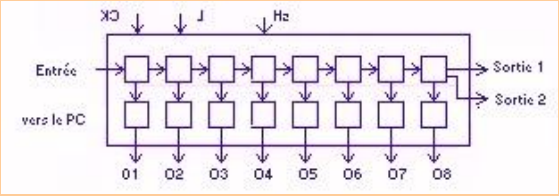
Les MAX7219 ou 7221 permettent de piloter 64 LEDs (matricées). C'est-à-dire qu'il faut mettre les LEDs entre un segment (A-G+DP) et une ligne (L0-7). La communication se fait par une liaison série synchrone SPI comme dans le cas du 4094 grâce aux pattes DI / CLK / LD. Il est possible de chaîner d'autres circuits grâce à la sortie DO mais nous n'utiliserons pas cette possibilité car 64 LEDs, cela fait déjà beaucoup. Le réglage de l'intensité lumineuse maximale se fait par une résistance à mettre entre +5V. Il est ensuite possible de diminuer cette intensité avec des commandes. Il est impératif de monter un condensateur chimique de 10uF sur les broches d'alimentation pour que le circuit fonctionne correctement.

== Le registre à décalage de sortie == (4094)
(non utilisé actuellement)

Les registres à décalage nous permettent d'obtenir autant de sortie que l'on veut en utilisant seulement 3 sorties du UC. Pour commander les sorties et aiguillages nous utiliserons des registres 4094 qui permet de contrôler 8 sorties. Pour les entrées, sont pendant est le 4021.

Voici son brochage :

Et le diagramme fonctionnel :



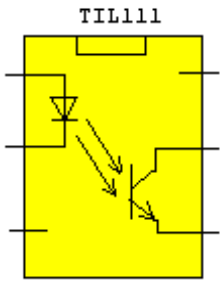
A chaque front positif (passage du niveau bas au niveau haut) d'horloge sur CK ; la donnée présente en entrée est mise dans la première bascule (carré en haut à gauche) alors que l'ancienne donnée de la 1ere va en 2 … Les données se décalent donc. Lorsque la patte L pour Load (chargement) est à l'état haut alors les données des bascules de décalages sont copiées dans les bascules de sorties qui se retrouvent sur les pattes O1 à O8. Au niveau bas, les sorties sont mémorisées et indépendantes de l'état des bascules de décalages. Hz permet de d'activer les sorties lorsqu'il est à l'état haut, nous le mettrons donc à l'état haut. La patte S2 permet de chaîner les registres à décalage en la connectant à l'entrée du registre suivant. Nous n'utiliserons pas S1. Nous utiliserons donc les 4094 comme suit :

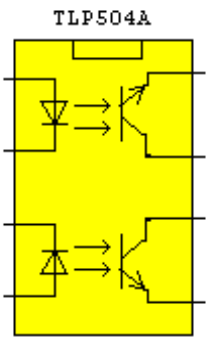
- Décalage des données présentées sur l'entrée avec des impulsions sur l'horloge
- Transferts des données sur les sorties avec une impulsion sur la patte Load.

Pour le registre des entrées, nous laisserons Load à l'état haut afin d'économiser une patte car ce n'est pas gênant de voire les données se déplacer en sortie alors que c'est inacceptable pour les 2 autres groupes de registres.

== Les optocoupleurs == (TIL111 / TLP504A)

Un optocoupleur se compose d'une LED qui lorsqu'elle est alimentée, éclaire un phototransistor qui devient passant. Cela permet d'isoler électriquement les 2 parties de l'optocoupleur et donc du montage associé. Dans notre cas, ils seront utilisés dans ce montage pour réaliser des capteurs de courant. On trouve des optocoupleurs simples ou multiples.





Annexe C : Cas pratique avec mon réseau

Cette partie présente la mise en œuvre de Free-DCC dans un cas concret. Je prends bien évidement comme exemple mon réseau.

TBD: rajouter