D17 : Centrale DCC / WIFI

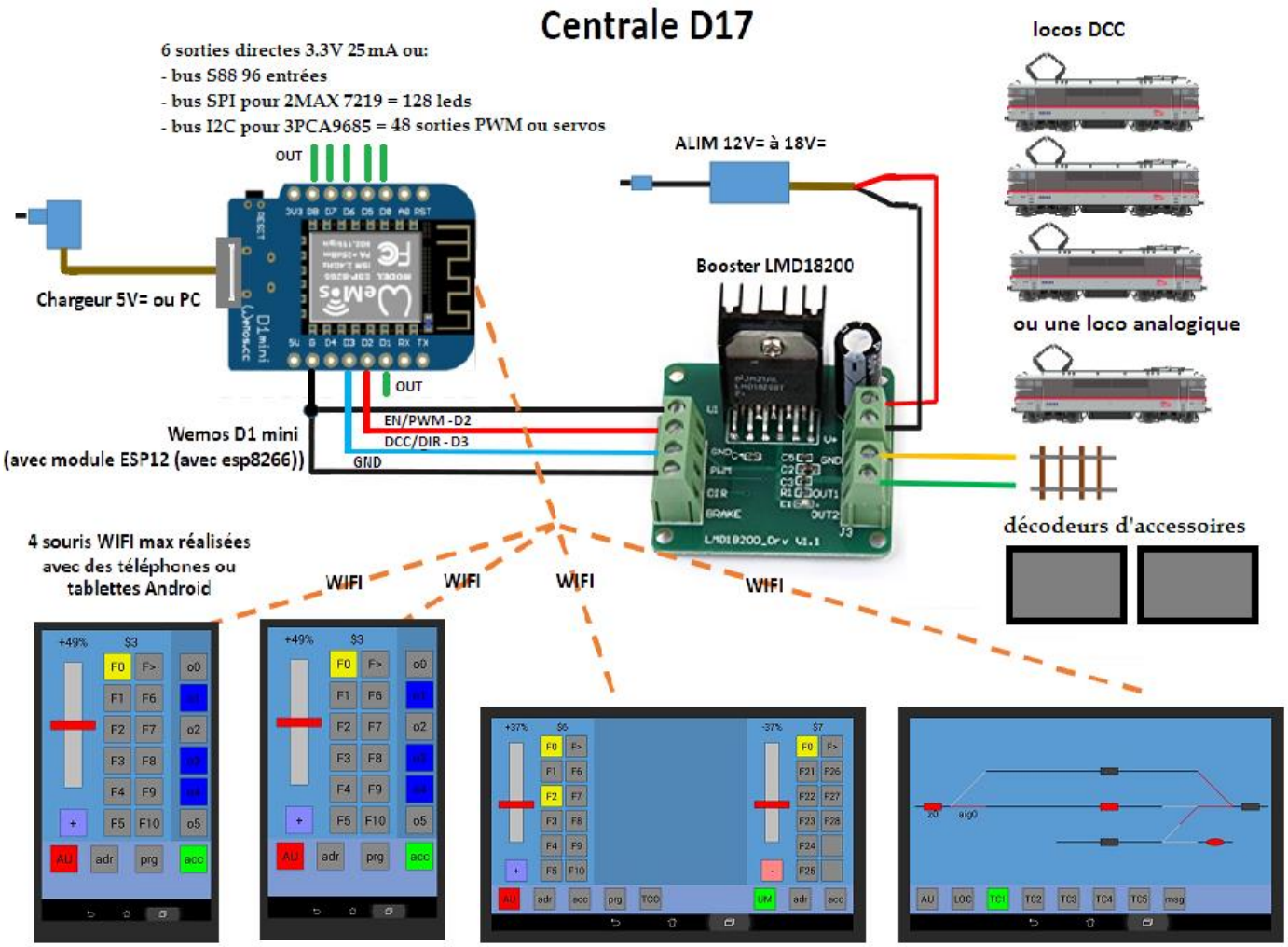


Table des matières :

(Ulysse Delmas-Begue documentation du 31 juillet 2018)

1. Introduction

** Présentation

La centrale D17 est une petite centrale DCC et analogique permettant de conduire des locomotives DCC ou analogiques avec des téléphones ou tablettes Android par WIFI. La centrale est également capable de piloter des accessoires (aiguillages, feux ...) via des décodeurs d'accessoires DCC ou directement via ses 6 sorties ou par bus SPI et MAX7219 pour des LEDs ou par bus I2C et PCA9685 pour des sorties PWM ou servos. La centrale propose aussi un bus de rétrosignalisation S88 pour les entrées. C'est une centrale à réaliser soi-même qui revient à 10€.

Le cœur de la centrale est un module Wemos D1 mini sur lequel est soudé le module ESP12 qui contient le circuit esp8622 qui est un microcontrôleur avec un système WIFI complet.
Le microcontrôleur peut se programmer avec l'environnement de développement Arduino. Le système WIFI peut fonctionner en mode point d'accès ou station. Nous l'utiliserons en mode point d'accès sur lequel se connecteront les téléphones et tablettes Android. Il n'y a donc pas besoin d'utiliser un point d'accès WIFI externe. Le module créant lui-même son propre réseau WIFI ! Ce point d'accès autorise la connexion de 4 appareils WIFI. J'ai programmé le microcontrôleur afin qu'il généré un signal DCC ou analogique suivant les commandes des souris Android. Le module est alimenté via son port micro-USB par un chargeur USB ou un PC.

Le cout total de l'électronique de la centrale est minime. Environ 10 euros. C'est pourquoi je qualifie cette centrale de "centrale a 10 balles".
- 4 euros : module Wemos D1 mini (qui contient le module ESP12 (qui contient l'esp8266))
- 6 euros : module LMD18200 pour réaliser le booster
- (+ les alims)
- il n'y a pas besoin de rajouter un point d'accès WIFI, c'est le module esp8266 qui le réalise !

Pour avoir un nom simple, j'ai nommé cette centrale D17 (D pour DCC ou Dix balles et 17 pour 2017)

La centrale permet de conduire simultanément 8 locomotives DCC (avec 4 souris doubles) ou une locomotive analogique. Bien entendu, il peut y avoir plus de 8 locomotives DCC mais à l'arrêt. Il ne peut pas y avoir de locomotives DCC et analogique sur la même voie. SI on veut mixer les 2 type, il faudra gérer des zones de coupure. Il n'y a rien besoin de modifier au niveau électronique pour passer du mode DCC au mode analogique. Une simple sélection sur une souris suffit.

La centrale possède un bus de rétrosignalisation S88 pour les entrées. Cela est par exemple utile pour voir l'occupation des voies ou mettre en place la signalisation.

La centrale peut aussi gérer les accessoires via :
- des décodeurs d'accessoires DCC (basiques et étendus)
- ses 6 sorties directes 3.3V 25mA. (Si vous n'utilisez pas le bus S88, le bus SPI et le bus I2C)
- son bus SPI où vous pouvez brancher 2 modules MAX7219 de 64 leds chacun soit 128 leds au total
- son bus I2C où vous pouvez brancher 3 modules PCA9685 de 16 sorties chacun, soit 48 sorties au total. De plus, ces sorties peuvent être variable (PWM) ou piloter des servos.

La réalisation est facile car le module Wemos D1 mini et le booster sont disponible sur Internet (Amazon...). Il faudra juste souder 3 fils.

Au niveau logiciel, il faudra programmer le module Wemos. Pour cela, installez l'environnent Arduino et le support de l'esp8266 (instructions sur Internet) puis téléverser le programme « centrale_d17.ino ». Si vous le souhaitez, vous pouvez changer le nom du réseau wifi par défaut "D17-0001" et le mot de passe par défaut "ulysse31". Si la programmation vous dépasse, vous trouverez surement un "geek" qui se fera un plaisir de vous montrer comment faire car l'environnement Arduino est très rependu. Dans le pire des cas, vous pouvez m'envoyez votre Wemos par la poste avec une enveloppe de retour et je vous le programmerai (gratuitement bien entendu).

Sur les souris Android, il faudra installer l'application RFO basic disponible sur le Market Android (gratuite). Cette application permet d'exécuter des programmes basic. En effet j'ai écrit les programmes des souris en basic afin qu'il soit facilement compréhensible et modifiable. En effet rare sont les personnes qui auraient pu faire une modification si j'avais fait une application Android en Java. Lorsque vous connectez votre appareil Android au Pc, il se comporte comme une clef USB. Il suffit de copier les programmes des souris « rfo_souris_simple.bas » et « rfo_souris_double.bas » dans le répertoire rfo-basic/sources de votre appareil Android. Chargez le programme de la souris de votre choix le avec le bouton LOAD du RFO basic, puis faites RUN pour l'exécuter. Il est possible de créer un lien sur le bureau Android pour lancer directement les souris. Vous pouvez aussi créer 5 TCO en métant les fichiers tco1.txt à tco5.txt dans le répertoire rfo-basic/data

Ce document est divisé en 2 parties :

- La première partie traite de la partie sur la conduite des locomotives. Le module Weemos sera décrit ainsi que le module LMD18200T pour réaliser le booster

- La seconde partie traite du pilotage des accessoires comme les aiguillages, feux. Elle décrit aussi comment connecter des modules MAX7219 pour ajouter des LEDs et PCA9685 pour ajouter des sorties PWM et servos. Cette partie décrit aussi le bus de rétrosignalisation. Elle explique comment réaliser des TCOs. Enfin elle explique comment réaliser des automatismes.

Bien entendu, cette réalisation est entièrement gratuite dans l’esprit de « open source » / « open hardware »

2. Partie Locomotives

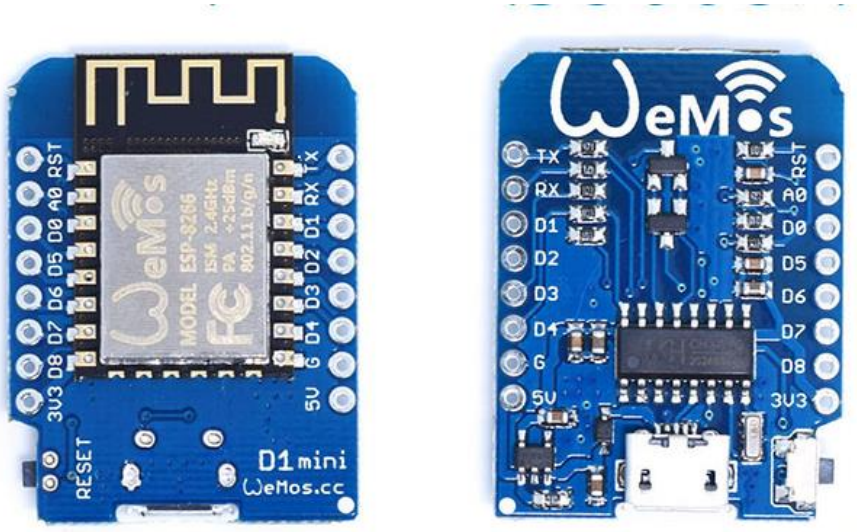
2.1. Electronique

2.1.a. Le module Wemos D1 min

Le module Wemos D1 mini contient un module ESP12 qui contient lui-même le composant esp8266.
L'esp 8266 est un circuit qui contient un microcontrôleur (sorte de petit ordinateur) et un système Wifi complet.
Le module ESP12 ajoute de la mémoire ainsi que l'antenne wifi
Le module Wemos ajoute l'USB pour l'alimentation et convertit ce dernier en lien série pour programmer et communiquer avec le 8266. Il ajoute aussi un régulateur 5V vers 3.3V.

Le microcontrôleur peut se programmer avec l'environnement de développement Arduino. Le système WIFI peut fonctionner en mode point d'accès ou station. Nous l'utiliserons en mode point d'accès sur lequel se connecteront les téléphones et tablettes Android. Il n'y a donc pas besoin d'utiliser un point d'accès WIFI externe. Le module créant lui-même son propre réseau WIFI ! Ce point d'accès autorise la connexion de 4 appareils WIFI. J'ai programmé le microcontrôleur afin qu'il généré un signal DCC ou analogique suivant les commandes des souris Android. Le module est alimenté via son port micro-USB par un chargeur USB ou un PC.

La carte étant en « open hardware », le schéma électrique est disponible (en annexe)
La figure suivante présente les différentes connexions :



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 qui est la masse (Si vous utilisez d'autres alimentations, connectez toutes les masses ensembles)
- +5V, c'est le +5V tiré de l'USB: pour alimenter les circuits sous 5V qui ne consomment pas trop (fusible de 500mA sur l'USB). Ne tirez donc pas plus de 400mA. Si vous avez besoin de plus ajouter une alimentation 5V sans oublier de connecter les masses. Ne jamais connecter les +5V entre eux !
- +3.3V, l'esp8266 fonctionne en 3.3V. Cette sortie est la sortie du régulateur du module Wemos qui transforme le 5V en 3.3V. Ne pas tirer plus de 400mA sur cette sortie. A noter que les sorties sont en 3.3V ce qui n'empêche pas de commander des circuits 5V qui basculent à 2.5V. Les entrées sont en 3.3V et basculent donc à 1.65V mais elles résistent à 5V.
- TX/RX sont utilisés pour la connexion USB
- A0 : entrée analogique 0-1V, nous l'utiliserons pour mesurer le courant fournit aux voies par le booster
- D0-D8 : 9 entrées/sorties numériques 3.3V pour nos signaux

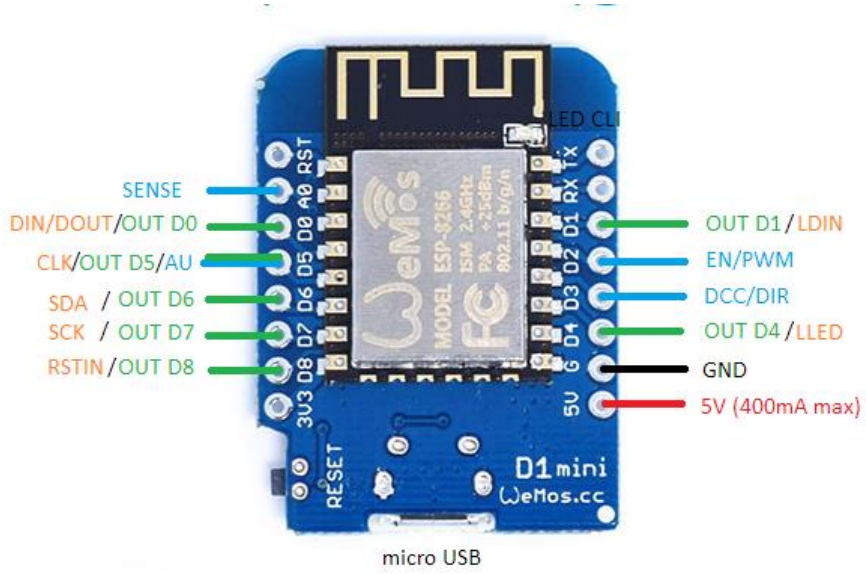
Les caractéristiques du module sont les suivantes :

- Processeur : 32bits à 80MHz
- Mémoire RAM : 520kB
- Mémoire Flash : 4MB
- E/S : 9 digitales D0-D8 (hors TX/RX) et 1 analogique A0
- UART : 2 dont une connectée à l'USB par un convertisseur USB/série

Pour donner vie à ce module, il faudra le programmer avec le programme de la centrale « centrale_d17.ino ». Pour ce faire, vous devrez compiler les ources du programme puis le télécharger dans la mémoire du module.
Pour cela, rendez-vous sur le site de la fondation Arduino https://www.arduino.cc/ afin d'installer l'environnement de développement Arduino (on parle d'IDE). La carte Wemos D1 mini n'est pas prise en charge de base par l'IDE, il faudra la rajouter (instructions sur Internet). Je vous conseille ensuite d'essayer l'exemple « blink led » (led clignotante) qui fait clignoter la led « L » présente sur la carte. Ne me demandez pas mon aide pour cette phase, tout est très bien expliqué sur Internet et rien ne vient de moi ;-) Maintenant que vous êtes des pros d'Arduino, vous pourrez télécharger le programme de la centrale. Dans le jargon Arduino, on dit que l'on « téléverse » un « sketch » 😉

Vous pouvez bien entendu modifier le programme pour l'adapter à vos besoins. Vous pouvez par exemple changer le nom du réseau wifi par défaut "D17-0001" et le mot de passe par défaut "ulysse31". Pour la partie des locomotives vous n'aurez normalement rien à modifier, par contre pour la partie TCO, il faudra indiquer dans le programme en C de la centrale comment réagir aux commandes du TCO, pour par exemple commander un aiguillage ou un feu sur un décodeur d'accessoires, allumer une LED par SPI sur un MAX, bouger un servo par I2C sur un PCA ...

L'affectation des pattes pour D17 est la suivante :

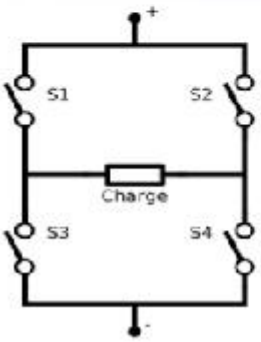


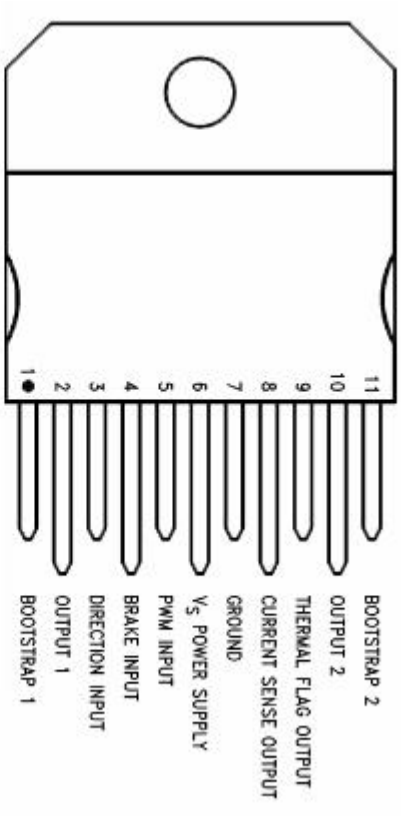
- EN/PWM D2 et DCC/DIR D3 permettent de piloter le booster
- SENSE A0 permettra de mesurer le courant du booster (pas encore dispo)
- AU D5 connecté à un ou des poussoirs permettra de générer un arrêt d'urgence et désactiver le booster (pas encore dispo). Cette fonction fonctionnera en même temps que OUT ou CLK.
- OUT D0,D1, D4, D5, D6, D7, D8 sont des sorties 0-3.3V 25mA qui peuvent être remplacées par des bus
- DIN DO, CLK D5, LDIN D1 et RSTIN D8 est le bus de rétrosignalisation S88
- DOUT D0, CLK D5, LLED D4 est le bus SPI pour commander les MAX7219 et leurs 128 LEDs
- SDA D6 et SCK D7 est le bus I2C pour commander les PCA 9685 et leurs 48 sorties PWM/Servos

2.1.b Le booster LMD18200T

Le signal DCC ou analogique pilote un booster fabriqué à l'aide d'un module LMD18200T qui alimente les voies. Ce circuit est capable de fournir 3A et est totalement protégé contre les court-circuits, surchauffe ... Il est alimenté par une alimentation continue à choisir entre 12V mini et 18V max. Bien entendu, on choisira une alimentation d'intensité suffisante. On prend en générale 500mA par loco (mais cela peut varier énormément).

Le LMD18200T permet d'alimenter les voies avec l'alimentation de puissance. Il est capable de :
- connecter ou déconnecter l'alimentations aux sorties grâce à l'entrée EN/PWM. 5V pour activer, 0V pour désactiver
- d'inverser la tension grâce à l'entrée DCC/DIR. 5V pour OUT1=ALIM/OUT2=GND, 0V pour OUT1=GND/OUT2=ALIM.





En analogique :
- PWM permettra de faire tourner le moteur plus ou moins vite en générant des impulsions plus ou moins longues. La fréquence à était choisie à 20KHz afin d'être inaudible.
- DIR permettra de définir le sens de rotation

En numérique :
- EN sera mis à 1 pour activer le booster
- DCC sera utilisé pour amplifier le faible signal DCC 0-3.3V en +ALIM/-ALIM

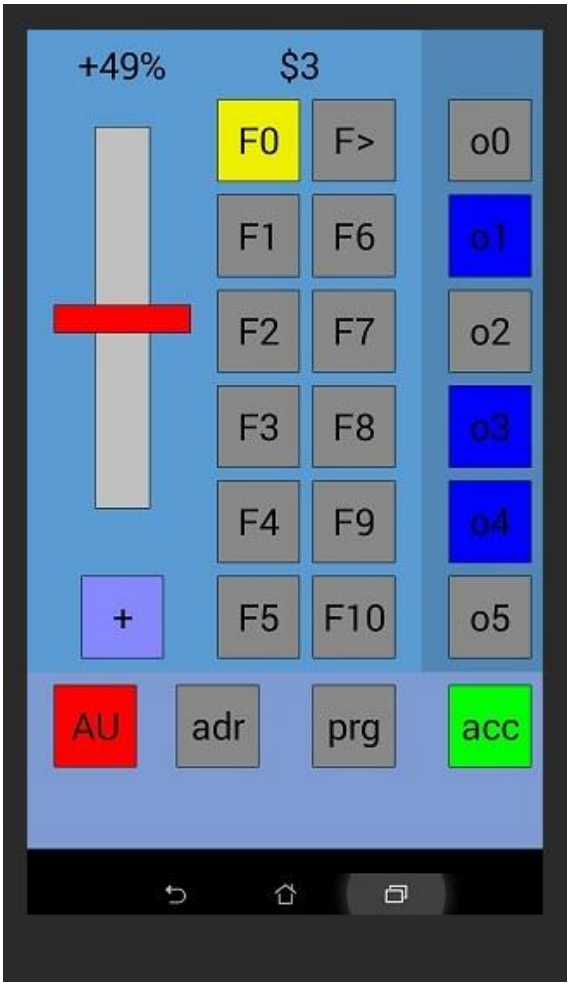
En arrêt d'urgence :
- EN sera mis à la masse afin de désactiver le booster afin de ne plus avoir aucun courant dans les voies. Les locomotives ne bougeront plus, mais il ne sera plus possible non plus de commander les décodeurs d'accessoires commandés par les voies. Il serait bien entendu possible d'utiliser le signal DCC 0-3.3V en aval du booster pour commander les décodeurs d'accessoires.

La sortie Sense reflète le courant en sortie du booster par une tension variable. Avec un montage calculé pour réduire la tension à 1V, il est possible de donner l'information de courant à la centrale.

La figure de la page de garde explique comment connecter le LMD18200T au module Wemos.
Il faudra connecter D3 à DIR et D2 à EN.

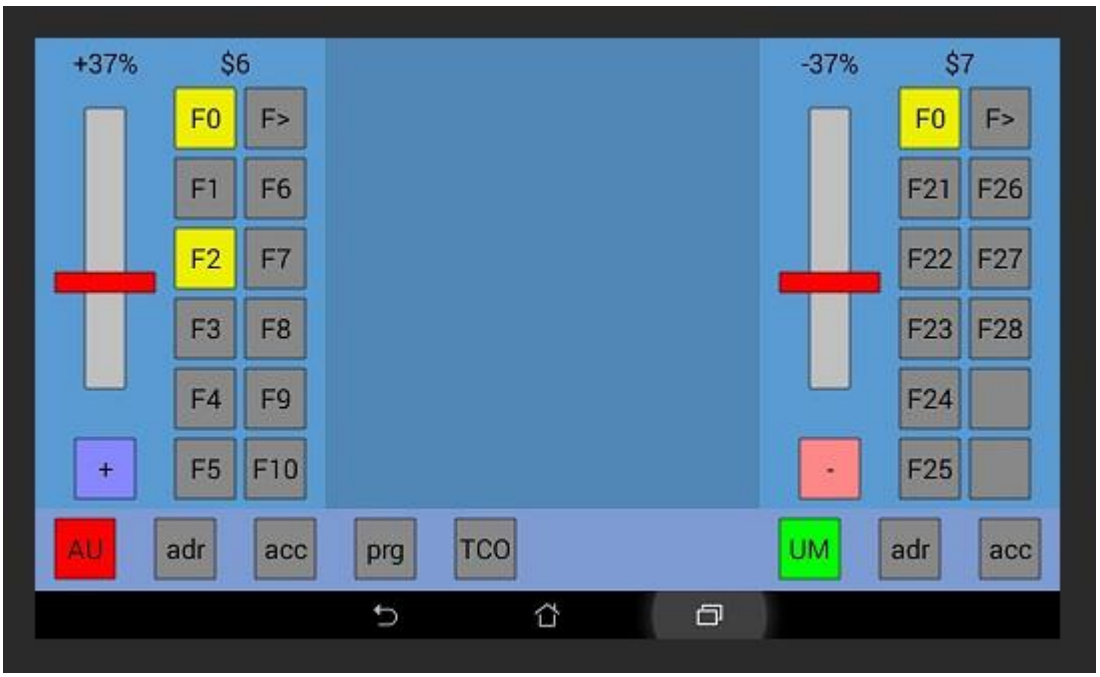
2.2. Utilisation des souris

2.2.a. La souris simple



- Cliquez sur le bouton ADR pour changer l'adresse de la locomotive. La proposition par défaut est 0.0. Ne tenez pas compte de la partie décimale, c'est une bizarrerie du RFO basic. Saisissez 10 par exemple pour l'adresse 10.
- Si une des souris utilise l'adresse 1 alors la centrale génère un signal PWM pour conduire une loco analogique. Dans le cas contraire, elle génère un signal DCC pour conduire des locos numériques.
- Les adresse 2 à 99 sont utilisables pour le DCC
- Utilisez le potentiomètre pour accélérer ou freiner.
- Le bouton +/- pour permet de changer le sens.
- Le bouton AU permet un arrêt d'urgence de l'ensemble des locomotives.
- Les boutons des fonctions F0 à F28 changent l'état des fonctions auxiliaires. (F0 commande souvent les phares).
- Le bouton F> permet d'afficher F1-F10 ou F11-F20 ou F21-F28 (F0 est toujours disponible)
- Le bouton ACC permet de conduire la loco en penchant la tablette. Pour inverser l'effet de l'accéléromètre, appuyez sur le bouton de sens +/-
- Le bouton PRG permet de programmer les décodeurs. Attention, toutes les locos alimentées par le booster seront programmées ! N'en laisser donc qu'une !!
Afin d'éviter les erreurs, il est nécessaire de rentrer un code de programmation (qui est 1234 par défaut). Ensuite l'adresse du CV est demandée, puis la valeur.
Par exemple le CV pour changer l'adresse de la loco est le numéro 1. La valeur est à choisir entre 2 et 99. Ne pas utiliser 0, ni 1 qui est utilisé pour l'analogique sur D17.
- Les boutons o0 à à5 permettent de changer l'état des sorties OUT0-5 (3.3V 25mA)

2.2.b. La souris double



La souris double s'utilise exactement comme la souris simple. Seuls les boutons adr et acc sont spécifiques à une locomotive.

Le programme utilise le "multitouch" ce qui permet de détecter 2 doigts et autorise à bouger les 2 potentiomètres simultanément.

Le nouveau bouton UM est utilisé pour envoyer les mêmes ordres de vitesse aux 2 locomotives. Avant d'activer l'UM, mettez les 2 locomotives dans le bon sens avec les boutons sens (+/-). Après activation de l'UM, le déplacement d'un potentiomètre, déplace l'autre potentiomètre et l'appuie sur un des bouton sens, inverse les 2 sens. Par soucis de simplicité accéléromètre et UM ne fonctionnent pas ensembles, mais si vous voulez faire de l'UM avec l'accéléromètre, activez simplement les 2 accéléromètres. En mode UM les locos doivent avoir à peu près les mêmes caractéristiques. L'UM n'a aucun sens en analogique car par nature toutes les locos alimentées bougent ensembles. Enfin notez qu'il est possible de programmer les décodeurs afin de leurs faire gérer l'UM. Dans ce cas, les 2 locomotives réagiront à l'adresse UM spécifiée ...

Le bouton TCO (Tableau de Control Optique) permet de passer à l'écran des TCOs

Les boutons des sorties ont disparu car vous pouvez contrôler les sorties et bien plus par le TCO.

3. Partie Accessoires

Notez que vous n'êtes pas obligé de commander vos accessoires par la centrale. Vous pouvez très bien déplacer vos aiguillages à la main ou électriquement avec des boutons poussoirs. C'est un peu moins vrai avec la signalisation, car cela va vite nous souler ou vous allez oublier, mais c'est possible. Vous pouvez aussi utiliser une électronique distincte pour la gérer comme un BAL à relais utilisant des détecteurs de courant ou ILS et relais bistables ... Le seul cas où il faut obligatoirement relier les accessoires à la centrale c'est pour l'automatisation ! Souvent avoir la possibilité de contrôler les accessoires ne sera pas suffisant, il faudra aussi avoir des entrées. Par exemple vous pouvez contrôler les leds d'un feu mais si vous n'avez pas d'entrée vous renseignant sur l'occupation des voies ce ne sera pas pratique.

3.1. Electronique

3.1.a. Les alimentations → Dans la partie TCO

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

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

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

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

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

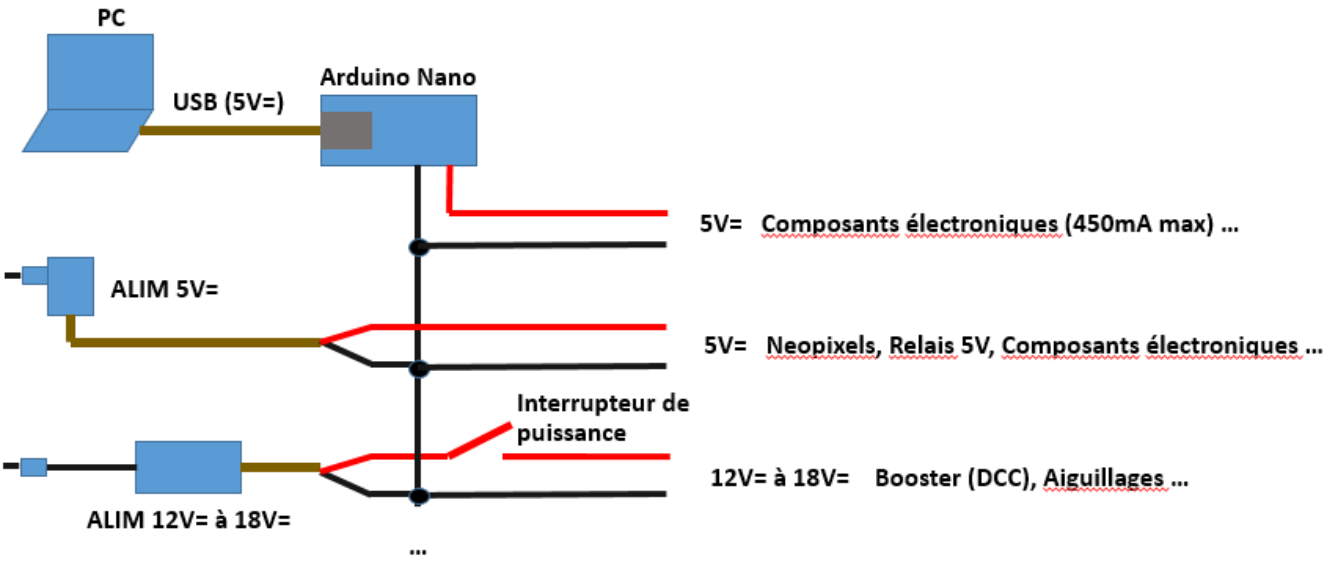
Utiliser un PC permet d'utiliser le port série de l'Arduino pour par exemple voir les paquets DCC décodés par le décodeur ou contrôler le décodeur autrement que par le DCC.

Exemple de quelques consommations :

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

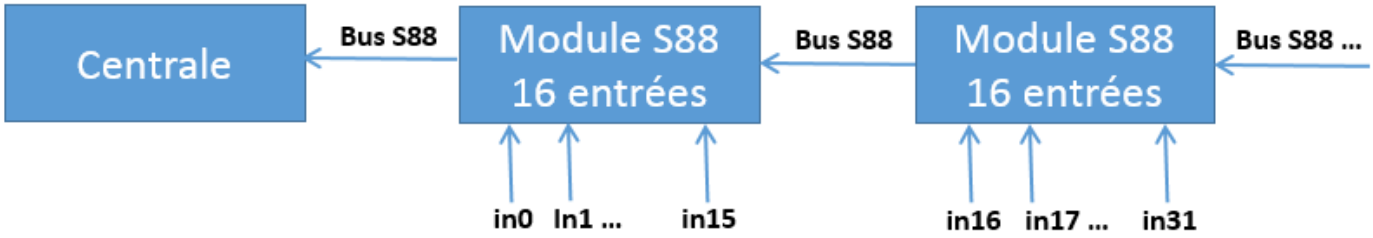
Il est de bonne pratique de prévoir un interrupteur sur l'alimentation de puissance.

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

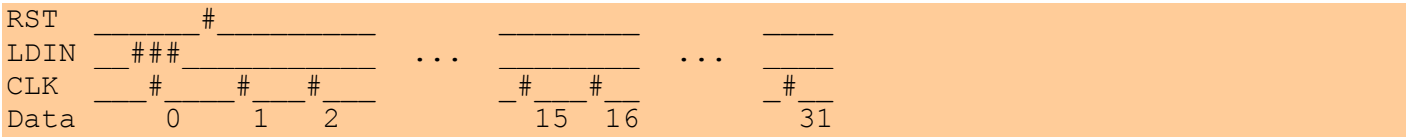


3.1.c. Les entrées par le Bus de rétrosignalisation S88

Le bus de rétrosignalisation S88 permet de récupérer l'état des entrées des modules S88. Généralement, les modules S88 permettent de rajouter 8 ou 16 entrées au système. La centrale D17 supporte au maximum 96 entrées (par exemple 6 modules de 16). Vous pouvez trouver dans le commerce des modules S88 ou les réaliser vous-même. De nombreux montages sont disponibles sur Internet.



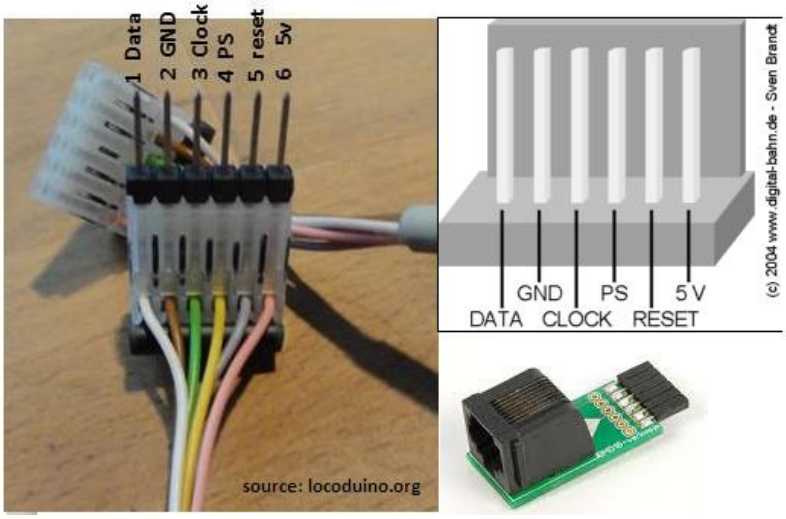
Le bus S88 fonctionne de la façon suivante : Une impulsion sur LDIN (Load IN = Charger les entrées), copie l'état des entrées dans un registre à décalage. L'entrée 0 (la première entrée du premier module (le plus proche de la centrale)) apparait sur la ligne DATA connectée sur l'entrée DATIN de la centrale. 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 résistances de 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.

Certains modules intègrent directement des détecteurs de courant afin de détecter l'occupation des voies. Dans le cas contraire vous devrez intercaler un détecteur de courant entre les voies et le module S88. Vous pouvez bien entendu utiliser les entrées pour connecter ce que vous voulez, vous n'êtes pas obligé d'utiliser un détecteur de courant.

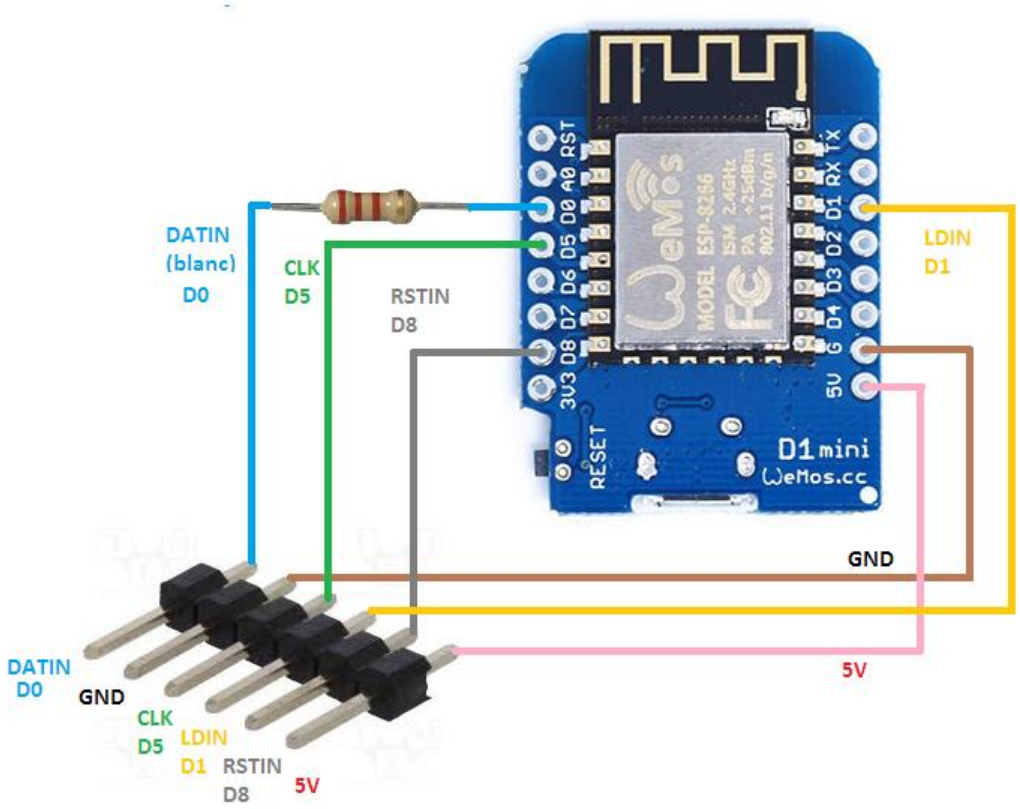
La connectique est standardisée sur un connecteur à 6 broches espacées de 2.54mm. Le même pas que les headers Arduino. Il est apparu récemment le S88-N qui est identique au S88 standard mais qui utilise des câbles réseaux et connecteurs RJ45. Vous pouvez utiliser un adaptateur pour passer d'une connexion à l'autre.



La figure suivante montre comment ajouter un connecteur S88 à D17. Bien entendu, on pourra aussi réaliser quelque chose de plus propre qu'un connecteur volant. On pourra par exemple mettre la 17 et le connecteur sur une plaque à trou du type veroboard ou réaliser un véritable circuit imprimé.

Si les besoins en 5V des modules S88 dépassent les possibilités du Wemos (pas plus de 400mA sur la broche 5V), il faudra couper le fil 5V, et utiliser une autre alimentation 5V sans oublier de relier les masses des 2 alimentations entre elles.

Vous avez sans doute remarqué la présence d'une résistance de 2200 Ω. Comme le nombre de pattes est limité, j'utilise la patte D5 aussi bien pour les données en entrée en provenance du S88 que pour les données SPI en sortie à destination des MAX7219. Lorsque D0 est utilisé en sortie, elle impose sa loi et les MAX voient les données mises sur la patte D0 par le Wemos car la résistance de D0 en sortie est inférieure à la résistance de 2200Ω. Lorsque D0 est en entrée, l'impédance de la patte D0 ainsi que les entrées DATA des MAX sont tellement grandes par rapport à 2200 Ω que les données S88 arrivent intactes sur la patte D0. Oublier la résistance détruit la sortie D0 ainsi que la sortie DATA du module S88 … La valeur n'a pas besoin d'être super précise, une valeur entre 1kΩ et 10kΩ devrait fonctionner.



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

Vous pouvez bien entendu, réaliser un module s88 avec un Arduino. Vous trouverez pleins de montages sur Internet. Pour ma part, j'en avais fait un pour mon système free-dcc 2017. Si vous voulez le réaliser, aller dans le projet free-dcc 2017. 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).

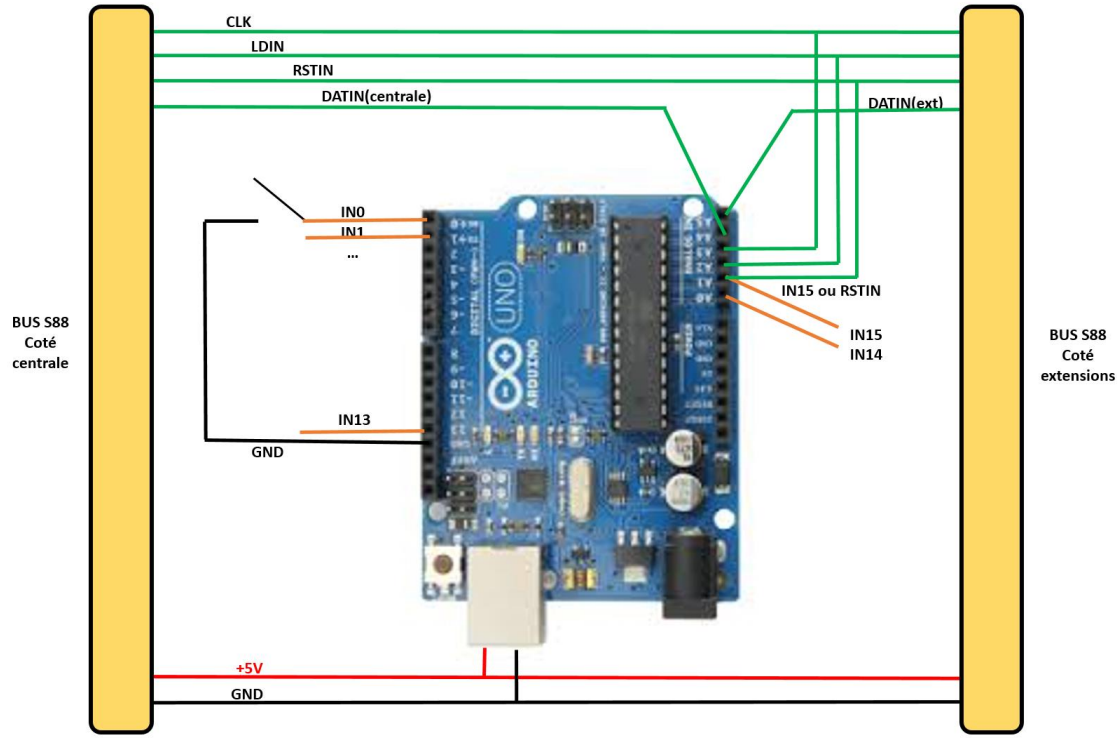
La constante **use_numeric_filter** permet d'activer un filtrage numérique des entrées afin d'éviter les « glitches ». Une entrée est déclarée active seulement lorsqu'elle a été vu 3 fois de suite au niveau bas.

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 :

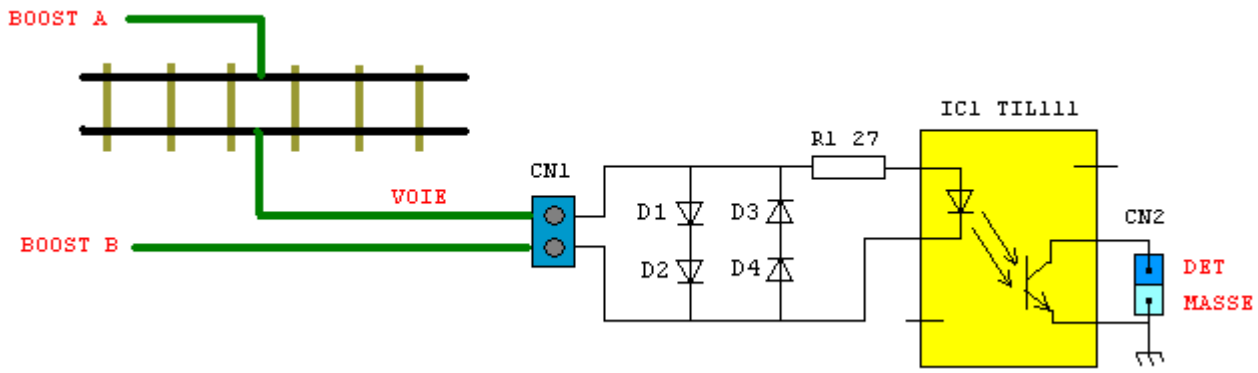
```
D0  = IN0    D1  = IN1    D2  = IN2    D3  = IN3
D4  = IN4    D5  = IN5    D6  = IN6    D7  = IN7
D8  = IN8    D9  = IN9    D10 = IN10   D11 = IN11
D12 = IN12   D13 = IN13   A0  = IN14   A1  = IN15 ou RSTIN

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 aussi utiliser le connecteur d'alimentation avec une tension de 7 à 12V ou directement injecter du 5V sur la patte 5V. Dans ce cas, merci de n e pas utiliser l'USB.



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.

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. Il faudra la aussi utiliser un autre montage.

3.1.c. Décodeurs d'accessories

Les décodeurs d'accessoires permettent de contrôler tous vos accessoires comme les aiguillages, feux, lampes, sServo-moteurs, moteurs …

La norme DCC prévoie :
- 510 décodeurs basiques à 8 sorties
- 2044 décodeurs étendus

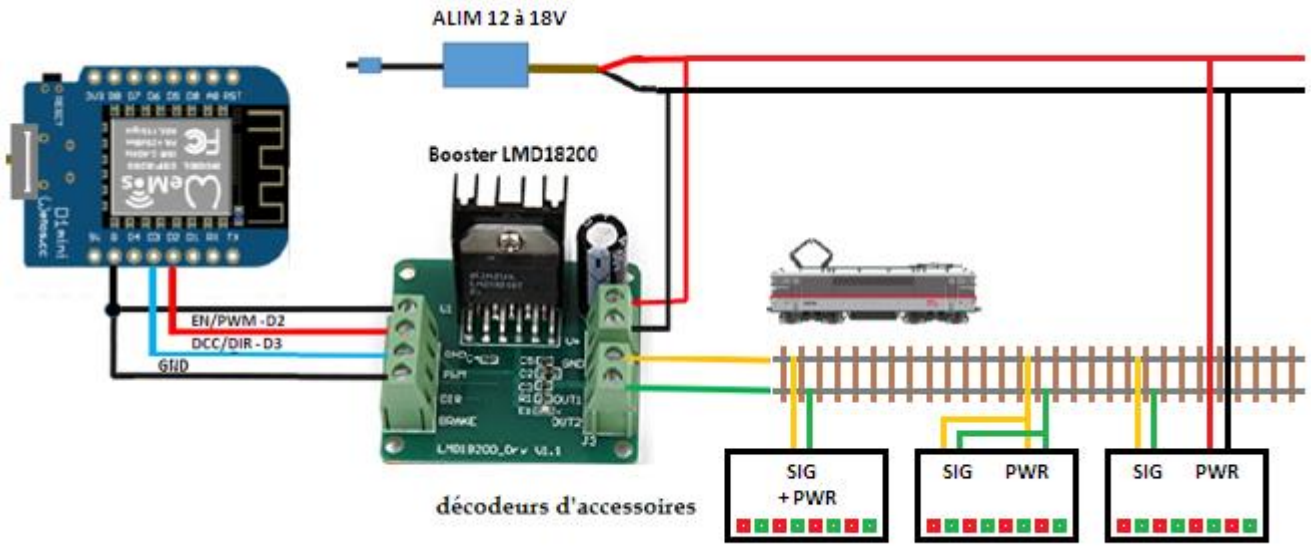
Pour plus d'information sur les décodeurs d'accessoires, merci de lire la doc de mon décodeur d'accessoire D18.
La doc explique également comment interfacer les actionneurs avec les sorties.

La centrale D17 sait contrôler tous les décodeurs d'accessoires.

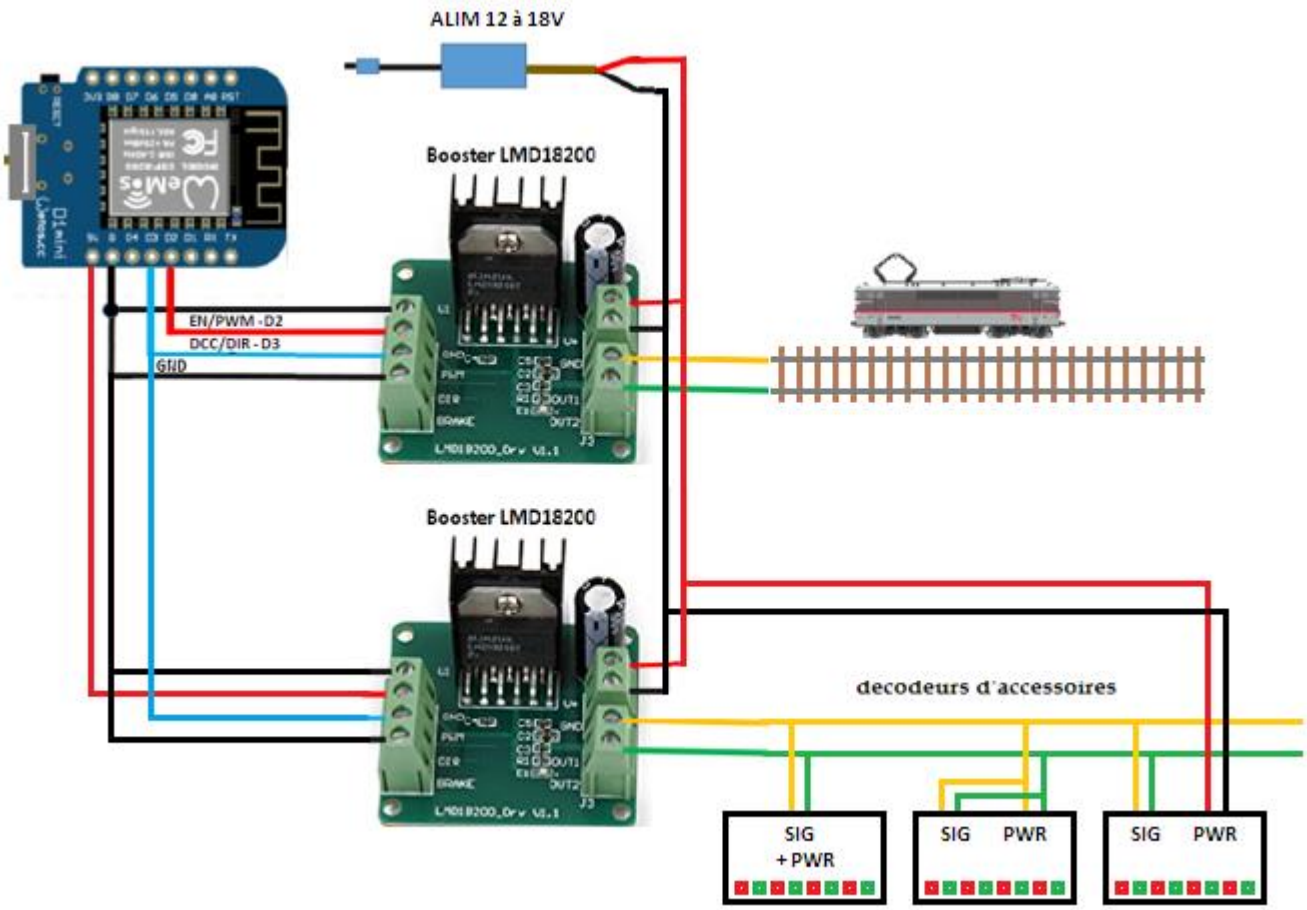
Vous pouvez les tester en leurs envoyant des ordres avec le bouton MSG de la souris TCO
Pour un décodeur basique :
  x<ADR 1-510>.<sortie 0-7>+/-
  Ex pour activer la sortie 0 du décodeur d'adresse 10: `x10.0+`
Pour un décodeur étendu :
  y<ADR 1-2044>=<valeur 0-31>
  Ex, envoyer la valeur 25 au décodeur d'adresse 12 : `y12=25`

En plus des décodeurs d'accessoires, D17 peut piloter des accessoires avec ses sorties directes, son bus SPI et I2C.

Pour connecter un décodeur d'accessoire à la centrale, le plus simple est de le brancher directement sur les voies car elles véhiculent le signal DCC et la puissance. Pour de rares décodeurs vous n'aurez pas le choix car ils ne possèdent qu'une connexion (2 fils) pour le signal DCC (SIG) et la puissance (PWR). La plupart des décodeurs possèdent 2 connexions, une pour le signal DCC et une autre pour la puissance. Vous pouvez les câbler en parallèle ce qui revient au cas précédent. Vous pouvez aussi connecter la puissance à une alimentation (généralement entre 12 et 18V). Cela permet de ne pas charger le booster inutilement afin que son courant serve exclusivement aux locomotives.



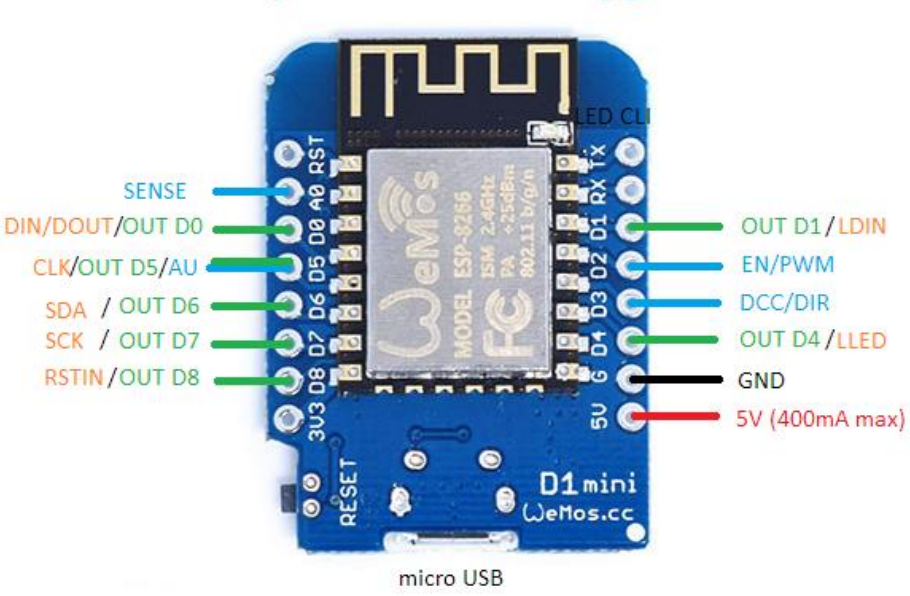
Dans le cas précédent, s'il y a un court-circuit sur la voie, les décodeurs ne recevront pas leurs commandes. Il est possible de résoudre le problème en utilisant un booster dédié pour les accessoires. La broche activation sera mise à 5V afin que ce second booster soit toujours actif. En mode arrêt d'urgence D17 désactive le booster des locos avec la broche d'activation et donc cela n'affectera pas le booster des accessoires. Bien entendu, vous pouvez utiliser une électronique moins puissante que le LMD18200T surtout si les décodeurs sont alimentés par une alimentation externe.



Il y a encore des montages plus simples mais qui dépendent de l'étage d'entrée des décodeurs. L'entrée est souvent réalisée à l'aide d'un optocoupleur. La masse du décodeur n'est pas forcément la masse de l'alim car il y a souvent un pont redresseur en après les bornes PWR.
Dans le cas du décodeur D18, le plus sage est d'utiliser un optocoupleur, mais vous pouvez aussi directement utiliser le signal DCC 0-3.3V de la centrale. La masse est commune avec ce décodeur.

3.1.d. Les sorties directes

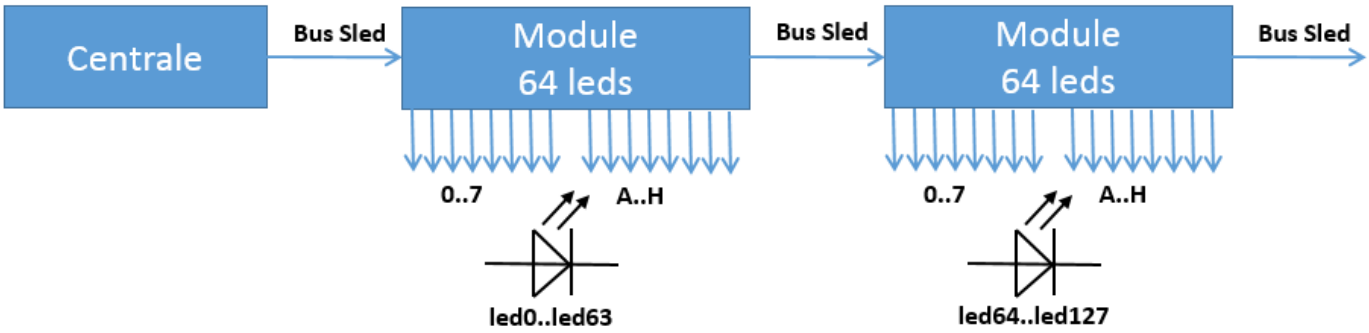
D17 propose 6 sorties directes 0-3.3V 25mA : OUT D0, OUT D1, OUT D5, OUT D6, OUT D7 et OUT D8



Elles sont partagées avec les bus S88, SPI et I2C. Plus vous utiliserez de bus, moins vous aurez de sorties directes. Vous pouvez y connecter des LEDs, lampes, moteurs, relais, aiguillages par l'intermédiaire ou non de transistors et drivers. Lisez le chapitre sur les sorties de la documentation du décodeur d'accessoires D18 pour plus d'informations.

3.1.e. LEDs via modules MAX7219 sur bus SPI

Le bus SPI (ou Sled) permet de raccorder jusqu'à 2 modules de 64 leds. Chaque module utilise un driver de LED MAX7219 ou 7221 capable de gérer 64 leds chacun, soit 128 leds au total.

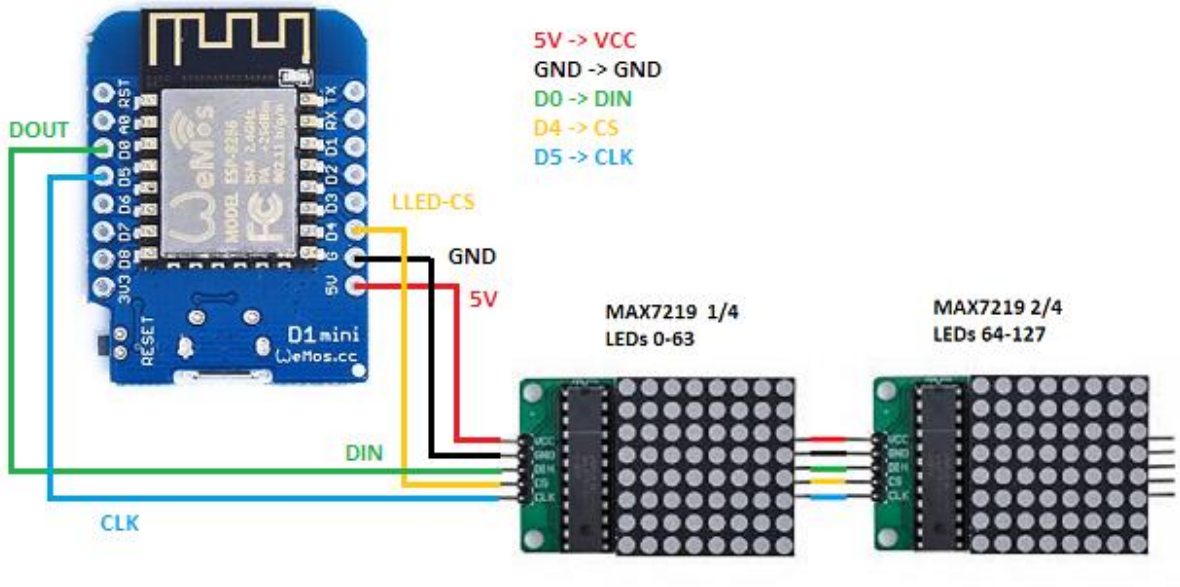


Un bus série de type SPI permet de communiquer avec les 7219. Chaque échange commence par la mise à l'état bas du signal d'activation LDLED (Load Led = chargement des LED) puis d'un décalage de 4 mots de 16 bits (un pour chaque MAX7219), puis se termine par une validation de l'échange par la mise à 1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

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

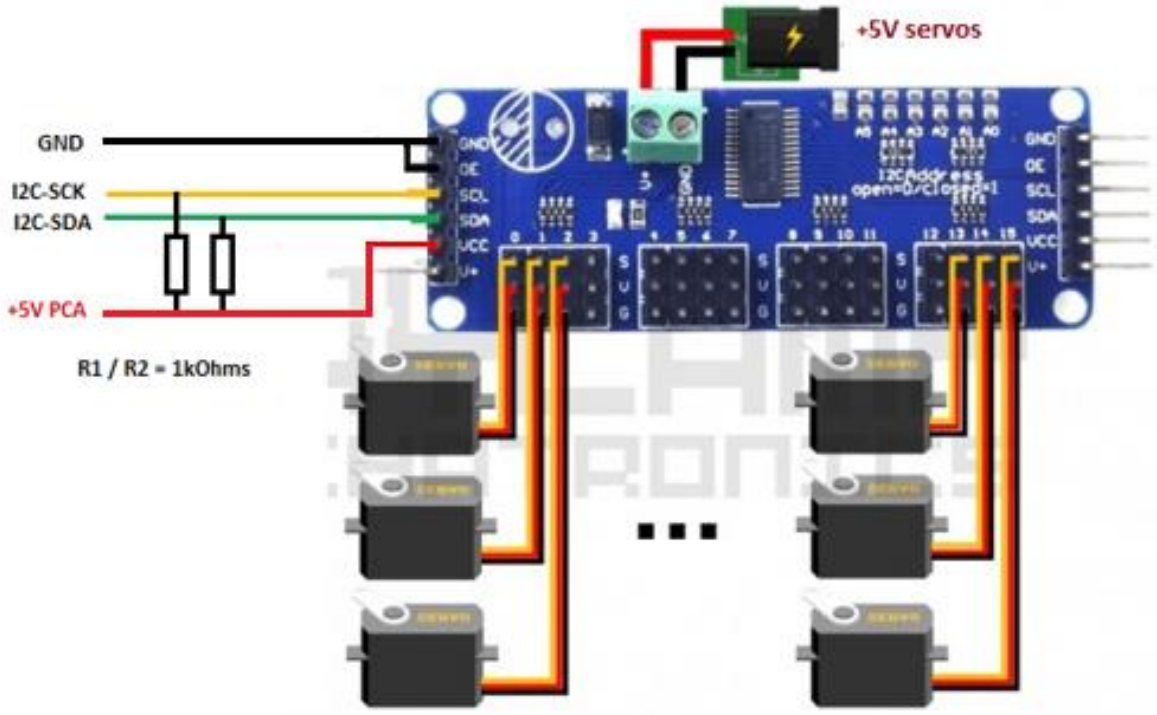
Il est possible de configurer la luminosité de l'ensemble de LED de chaque 7219.
Vous pouvez réaliser des modules avec le circuit MAX7219 (fonctionne aussi avec le 7221) ou alors les acheter directement sur Amazon par exemple pour 4€. Il existe des modules avec une matrice de 64 LEDs (8*8). Il existe aussi des modules avec un afficheur comprenant 8 afficheurs à 7 segments. Il suffit d'enlever la matrice ou l'afficheur pour accéder au connecteur afin d'y connecter vos LEDs. La matrice ou l'afficheur, pouvant servir à tester votre montage.

Le circuit commande les LEDs en courant, il n'y a donc pas besoin de rajouter une résistance en série avec les LEDs. L'intensité globale des LEDs peut être réglée par la programme. Ceci peut être utile pour voir la signalisation en plein jour et éviter qu'elle ne se transforme en lampadaire dans l'obscurité. Comme les LEDs sont commandés an matrice, vous disposez de 8 pattes pour les cathodes et 8 pour les anodes. Vous pouvez donc commander aussi bien des signaux à anodes ou cathodes communes. Le programme permet aussi de faire clignoter les LEDs que vous voulez à 1Hz et d'inverser la phase si nécessaire. Si toutes les LEDs sont utilisées alors la consommation sera de 330mA. Si cela excède les capacités du Wemos, utiliser le 5V d'une autre alimentation. Penser à relier sa masse à celle de l'Arduino. Le schéma suivant montre comment connecter des modules MAX7219 à la centrale D17.



3.1.f. Sorties PWM/Servos sur PCA9685 par 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. Les 2 signaux du bus I2C sont SDA et SCK. Actuellement, la centrale supporte 3 PCA8596. Chaque circuit dispose de 16 sorties pouvant être utilisées individuellement en mode PWM ou Servo. Donc 48 sorties au total ! Vous trouverez sur Amazon des modules PCA9685 à 8€ ainsi que des micro-servos 9G à 2€ pièce afin de contrôler tout ce que vous voulez sur votre réseau



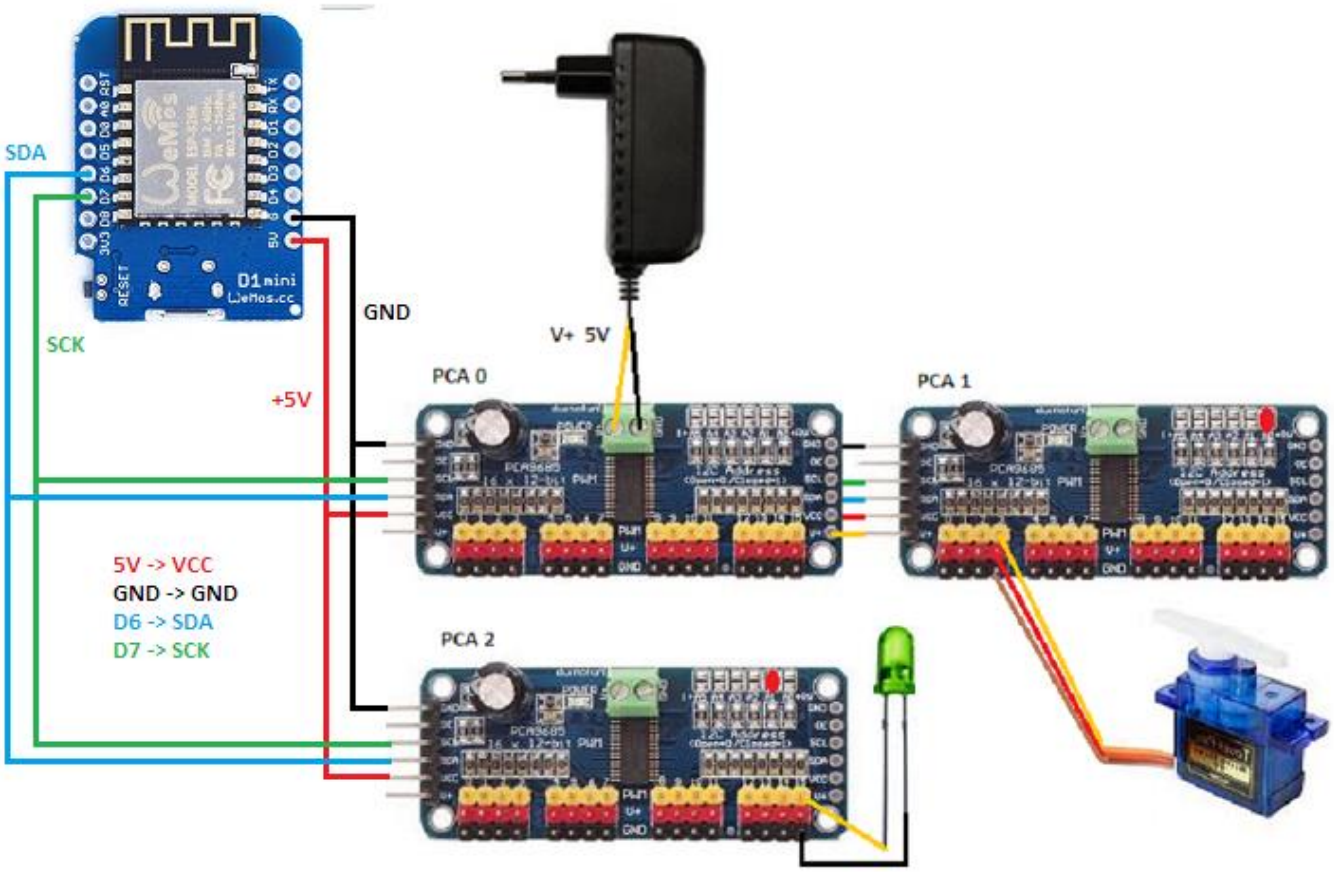
credit: image source: http://www.naylampmechatronics.com/blog/41_Tutorial-M%C3%B3dulo-Controlador-de-servos-PCA9685.html

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

Pour utiliser plusieurs modules, il suffit de les brancher en // ou les chainer. Ne pas oublier de changer les adresses. (Ajoutez un point de soudure sur A0 pour le 2eme module, et point de soudure sur A1 pour le 3eme module)
Pour utiliser des sorties au lieu des servos, utilisez les pattes jaunes. Ces sorties ne sont pas des sorties simples mais des sorties PWM (voire le chapitre PWM). Vous pouvez néanmoins les commander comme des sorties normales en mettant la PWM à 0% pour un état bas et à 100% pour un état haut.

Une résistance de 220Ω est en série avec chaque sortie, vous pouvez donc vous passer de résistances si vous contrôler des LEDs.

Ne pas utiliser le 5V de l'Arduino pour alimenter les servos car ils perturberaient ce dernier. L'alimentation +5V des servos se connecte sur le bornier à vis. La masse est commune avec les masses (GND) des connecteurs latéraux. Ce +5V nommé V+ est également disponible sur les connecteurs latéraux. Sur le schéma suivant ce 5V passe ainsi entre les cartes PCA0 et 1. Il est également disponible sur la broche centrale des 16 connecteurs des servos.
Le circuit est quant à lui alimenté par la broche VCC des connecteurs latéraux. C'est lui qui alimente aussi les broches PWM. Si vous utiliser 3 modules pour commander 48 leds cela tirera 48*15mA = 700mA sur le 5V de la centrale ce qui est trop et il vous faudra donc alimenter la patte VCC par une nouvelle alimentation 5V en gardant une masse commune avec la centrale.



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

La documentation du D18 montre comment utiliser un ruban de LEDs RVB pour éclairer votre réseau.

3.1.g. Multiplexage

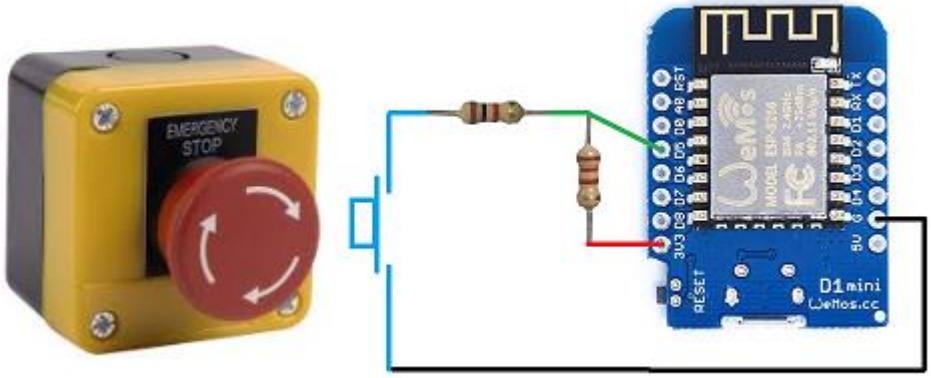
Pour économiser des sorties et électronique de puissance, il est possible de multiplexer !

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

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

3.1.h. Rajout d’un bouton d’arrêt d’urgence

Vous pouvez connecter un bouton d'arrêt d'urgence sur D5 en utilisant le montage suivant :



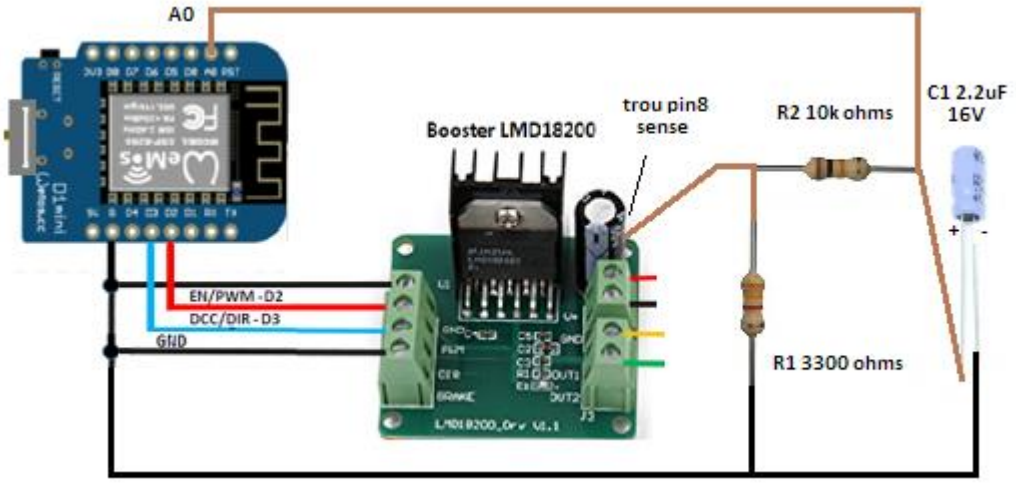
La résistance de 2200ohms sur le 3.3V fixe un état haut lorsque le circuit électrique est ouvert. La résistance de 1000ohms en série avec le bouton, fixe un état bas lorsque le circuit est fermé via le bouton. Cette dernière résistance évite les conflits lorsque la patte est utilisée en sortie pour générer le signal CLK pour les entrées S88 et les LEDs des MAX7219. Idem pour le signal OUT. Ne pas mettre cette résistance courcircuitera la sortie à la masse lorsque le circuit est fermé ce qui la fera cramer ...

Pour activer la fonctionalité d'arrêt d'urgence, vous devrez mettre dans le code de la centrale, la constante suivante à 1:
#define USER_USE_AU 1
Vous pouvez aussi choisir la polarité active avec:
#define USER_AU_LEVEL 0
Utilisez 0 pour un bouton normalement ouvert NO. (bouton qui ferme le circuit électrique lorsque l'on appuie dessus). Si vous utilisez plusieurs boutons, cablez les en parallèle.
Utilisez 1 pour un bouton normalement fermé NF. (bouton qui ouve le circuit électrique lorsque l'on appuie dessus). Si vous utilisez plusieurs boutons, cablez les en série.

le bouton est uniquement utilisé pour déclencher l'arrêt d'urgence. la désactivation se fait avec les souris

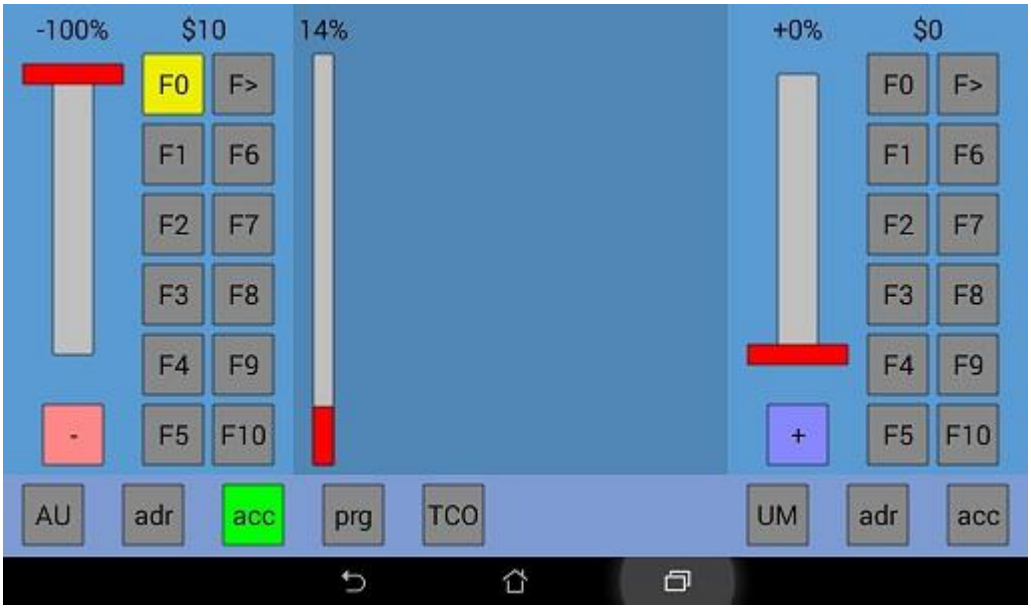
3.1.h. Rajout de la mesure du courant du booster

Vous pouvez mesurer le courant du booster avec l'entrée analogique A0 en utilisant le montage suivant :

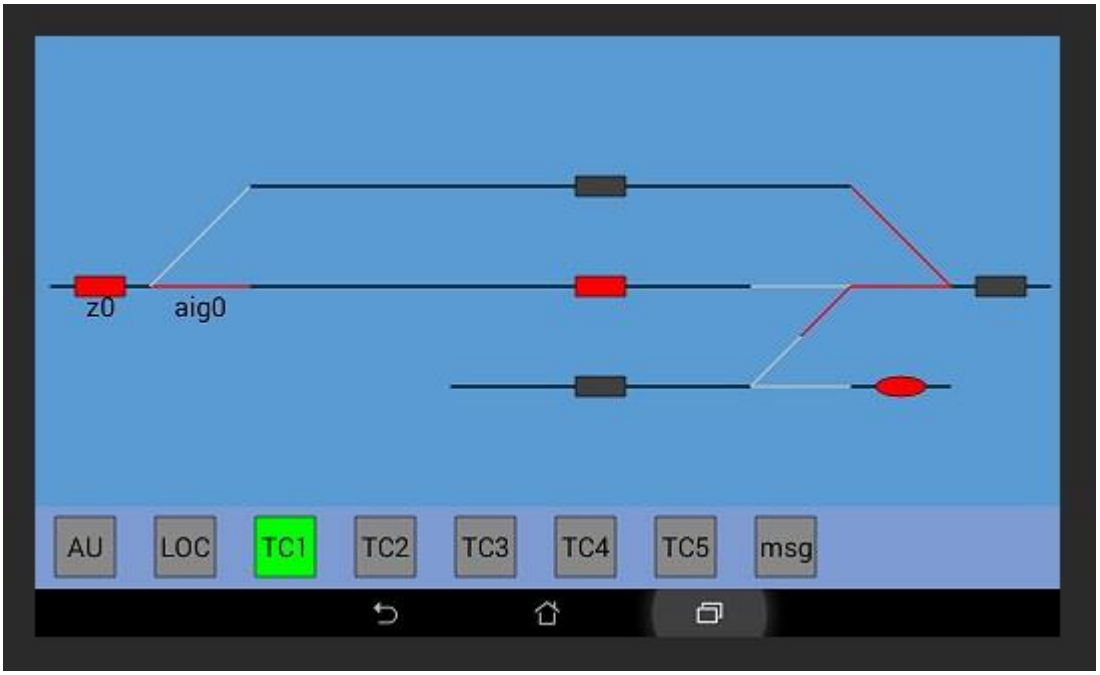


La patte 8 du LMD18200T fournit une image du courant du booster 377uA/A. En gros cette patte donne 1mA pour 3A fournit. Vous pouvez souder un fil sur la patte 8 ou alors entrer en force un câble Dupont dans le trou métallisé sérigraphié "pin8 sense" qui se trouve derrière le condensateur.
L'entrée analogique du Weemos peut mesurer des tensions jusqu'a 3.3V. Dans le weemos, une résistance de 220K et une autre de 100K forment un pont diviseur afin d'abaisser la tension à 1V qui est le max que l'esp8266 peut mesurer). Pour être précis, nous devons utiliser toute la plage de mesure. Nous devons donc générer une tension de 3.3V pour le courant moyen max du booster qui est de 3A.
La résistance R1 de 3300 ohms transforme l'image du courant en tension. Pour 3A, l'image est de 1mA, soit 3.3V aux bornes de la résistance (U=RI=3300 * 0.0001=3.3).
Le couple R2/C1 moyenne cette tension sur un intervalle de 22ms (t=10 000*0.000 002 2) afin de s'affranchir des pics et signaux rectangulaires (DCC ou PWM).

Le convertisseur analogique numérique mesure cette tension et l'envoie aux souris qui demande l'état de la centrale. Les souris doubles peuvent alors afficher la mesure sous forme de bargraphe (en % 0%=0A 100%=3A).



3.2. Création de TCOs sur la souris double



Chaque souris double dispose de 5 TCOs pour commander les aiguillages, sorties et visualiser les entrées.

Le principe :
- Dans le fichier tco1.txt à mettre dans le répertoire rfo-basic/data de l'appareil Android :
  Vous dessinez le TCO avec des lignes, rectangles, ovales et du texte.
  Vous pouvez faire changer la couleur des lignes, rectangles et ovales en fonction des entrées de rétrosignalisation, de la position des aiguillages et états des variables "u"
  Vous définissez des zones de click pour envoyer des commandes a la centrale pour changer la position des aiguillages ou la valeur des variables "u"
- Dans les fonctions **user_notify_aig()** et **user_notify_u()** du programme de la centrale
  Vous indiquez ce qu'il faut faire quand une commande d'aiguillage ou un changement d'une variable "u" arrive

Les variables "u" pour utilisateurs sont 96 variables qui peuvent prendre comme valeur 0 ou 1.
C'est à vous de définir a quoi elles servent.
Par exemple vous pouvez décider que u0 gère les lampadaires des quais
Ainsi les TCO peuvent mettre cette variable a 0 ou 1 et la centrale devra commander les lampadaires

Toutes les 100ms, les TCO envoies la commande ? a la centrale
En réponse à cette commande la centrale répond par l'état des entrées/aiguillages/variables "u"

```
? --> au0 dcc i000000000000000000000000 dFFFFFFFFFFFF e000000000000 u000000000000000000000000 c00
```

i pour in (entrées en Anglais) indique l'état des entrées
Le premier 0 indique que les 4 premières entrées sont inactives ...
Pour ne pas avoir de trop longs messages 4 entrées sont mises par symbole. (Pas plus pour rester en ASCII Hex)

```
0 pour in3=0 in2=0 in1=0 in0=0
1 pour in3=0 in2=0 in1=0 in0=1
2 pour in3=0 in2=0 in1=1 in0=0
3 pour in3=0 in2=0 in1=1 in0=1
4 pour in3=0 in2=1 in1=0 in0=0
5 pour in3=0 in2=1 in1=0 in0=1
6 pour in3=0 in2=1 in1=1 in0=0
7 pour in3=0 in2=1 in1=1 in0=1
8 pour in3=1 in2=0 in1=0 in0=0
9 pour in3=1 in2=0 in1=0 in0=1
A pour in3=1 in2=0 in1=1 in0=0
B pour in3=1 in2=0 in1=1 in0=1
C pour in3=1 in2=1 in1=0 in0=0
D pour in3=1 in2=1 in1=0 in0=1
E pour in3=1 in2=1 in1=1 in0=0
```

```
F pour in3=1 in2=1 in1=1 in0=1
```

Le système supporte 96 entrées 0-95

d pour (position) directe = aiguillage droit

e pour (position) évitement = aiguillage dévié

Il y a 2 variables pour la position car :

- durant la manœuvre l'aiguillage n'est dans aucune position
- à l'initialisation le système ne connait pas la position des aiguillages

Le système supporte 48 aiguillages 0-47

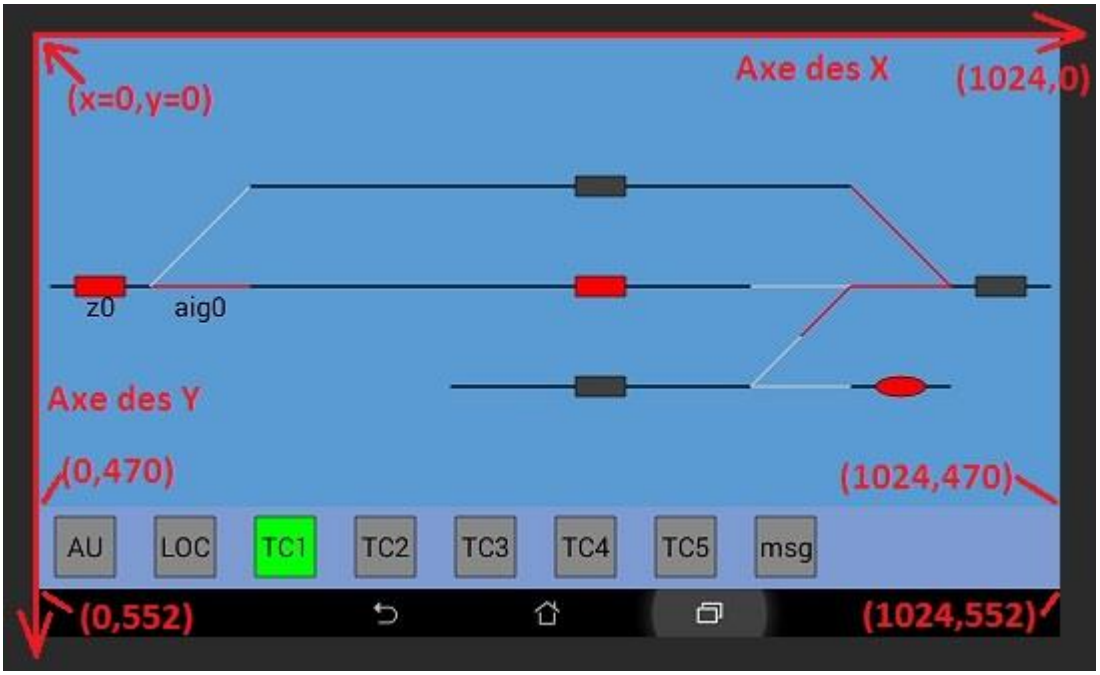
u est l'état des 96 variables "u" u0->u95

Si vous utilisez des TCO sur plusieurs tablettes, elles seront toutes au courant de l'état du réseau en temps réel grâce à cet état. Modifier un aiguillage ou une variable "u" sur une des tablettes, modifiera l'état sur toutes les autres

Vous pouvez utiliser les commandes suivantes suite à un appuie sur une zone du TCO

t0- commande pour mettre l'aiguillage 0 en position directe
t3/ commande pour mettre l'aiguillage 3 en position déviée
t47^ commande pour inverser la position de l'aiguillage 47
u0- commande pour mettre à 0 la variable u0
u5+ commande pour mettre à 1 la variable u5
u95^ commande pour inverser la variable u95
La plupart du temps l'utilisation des inversions est suffisantes

Pour le développement, j'ai utilisé une tablette avec un écran de 1024x600 pixels, la zone pour les applications était de 1024x552 et celle du TCO (sans les boutons du bas) est réduite à 1024x470 pixels
En horizontal, l'axe des x varie de 0 à 1023
En vertical, l'axe des y varie de 0 à 551
L'origine (x=0,y=0) se situe dans le coin supérieur gauche de l'écran
Si l'écran de votre tablette ne fait pas la même taille, le TCO sera automatiquement redimensionné pour occuper toute la hauteur mais pourra être tronqué sur la droite. Cela pour ne pas modifier la géométrie des éléments. Un développement est en cours pour faire tenir l'intégralité du TCO sur n'importe quel écran sans déformation de la géomètrie …



La configuration du TCO1 se fait dans le fichier tco1.txt à mettre dans le répertoire rfo-basic/data de l'appareil Android. Tco2.txt pour le second TCO …

La syntaxe de ce fichier est la suivante (elle est rappelée en commentaire au début des fichiers) :

```
# Fichier de configuration d'un TCO
#
# # commentaire
# ORIGINE,x0,y0
# COULEUR,r,g,b
# LIGNE,x1,y1,x2,y2[,var_type,id]
# RECT,xc,yc,xr,yr[,var_type,id]
# OVALE,xc,yc,xr,yr[,var_type,id]
# LABEL,xc,yc,txt
# ACTION,xc,yc,xr,yr,action
#
# var_type= in, - (aig_dir), / (aig_dev), u, u!
# alias: ORIGIN / COLOR / LINE / OVAL
```

Par défaut les coordonnées de l'origine (le coin supérieur gauche) sont x=0 et y=0
Vous pouvez la changer à tout moment. Les instructions suivantes utiliseront alors la nouvelle origine.
Cela est pratique lorsque l'on a dessiné une zone que l'on souhaite déplacer.

```
# declaer les dessins suivants de +100 vers la droite et 100 pixels vers le bas
ORIGINE,-100,-100
```

Pour ceux qui préfèrent la langue de Shakespear à celle de Molière, vous pouvez utiliser les alias. Par exemple remplacer ORIGIN à la place de ORIGINE …

Pour dessiner, une ligne, il faudra passer les 2 points (x1,y1) et (x2,y2) à la fonction LIGNE (ou LINE)

```
# voie 1
LIGNE,0,100,200,100
```

Idem pour les rectangles et ovales, vous spécifierez le centre et la demi largeur (le rayon pour un cercle).

```
RECT,100,100,50,50
OVALE,200,100,50,50
```

Pour du texte, vous spécifierez son centre ainsi que son texte

```
LABEL,100,110,gare de Toulouse
```

Par défaut, les formes seront noires. Vous pouvez changer les couleurs en jouant avec les variables **r,g,b** (red=rouge,green=vert,blue=bleue). Leurs valeurs varient de 0 à 255.

```
# creation d'un rectangle noir
COLOR,0,0,0
RECT,100,100,50,50
# creation d'un rectangle rouge
COLOR,255,0,0
RECT,200,100,50,50
```

Pour les lignes, rectangles et ovals, vous pouvez changer les couleurs dynamiquement en fonction :
- des entrées (spécifier in dans le champ var_type ainsi que son numéro dans le champ suivant)
- de la position des aiguillages (le signe moins – pour la position directe ou / pour la position déviée + le numéro)
- de l'état des variables "u" (avec u ou u ! pour son complément + numéro de la variable)

```
# La couleur de la ligne dépend de l'entrée 5
LIGNE,100,100,200,100,in,5
# La couleur du rectangle depend de la variable u2
RECT,200,100,50,50,u,2
# Les couleurs des ovales dependent de la position de l'aiguille 7
OVALE,300,100,20,20,-,7
OVALE,300,150,20,20,/,7
```

Par défaut, les couleurs sont définis comme suit :

Pour les lignes
- in à 0 -> couleur noir / in à 1 -> couleur rouge
- aigok ou aigko à 0 -> blanc / aigok ou aigko à 1 -> rouge
- u à 0 -> gris foncé / u à 1 -> vert

Pour les rectangles et ovals
- in à 0 -> noir / in à 1 -> rouge
- aigok ou aigko à 0 -> blanc / aigok ou aigko à 1 -> jaune
- u à 0 -> gris foncé / u à 1 -> vert

Pour changer ces couleurs, recherchez dans le code de la souris double: `! colors for line` et modifier les RVB.
Attention de laisser le premier paramètre (alpha) à 255 et le dernier (fill) à 1

```
 ! colors for line

GR.COLOR 255, 255, 0, 0, 1 %fill     in=1
GR.PAINT.GET p_paint_seg_occ
GR.COLOR 255, 0, 0, 0, 1 %fill       in=0
GR.PAINT.GET p_paint_seg_libre
...
```

Tant que l'on est dans les couleurs par défaut, vous pouvez aussi changer le fond du TCO

```
! rectangle zone de fond du TCO

GR.COLOR 255, 91, 155, 213, 1 %fill
```

Ex pour changer la couleur d'un segment suite à l'occupation d'un tronçon repérée par l'entrée in3 :

```
LIGNE,0,100,200,100,in,3
```

Le segment sera noir en l'absence de détection et rouge s'il y a une détection.

Ex pour un voyant rectangulaire d'occupation sur un segment qui reste noir :

```
LIGNE,0,100,200,100
RECT,100,100,25,10,in,3
```

L'indicateur sera noir en l'absence de détection et rouge s'il y a une détection.
Le segment quant à lui, restera noir.

Ex pour un aiguillage :

```
LIGNE,200,100,300,100,-,8
LIGNE,200,100,300,200,/,8
```

La branche inactive sera blanche et l'active rouge.

Ex pour un aiguillage avec des voyants :

```
LIGNE,200,100,300,100
LIGNE,200,100,300,200
OVALE,250,100,15,15,-,8
OVALE,250,150,15,15,/,8
```

Les branches resteront noires mais le voyant sera jaune sur la branche active et gris sombre sur la branche inactive.

Ex d'un indicateur d'une variable "u" :

```
RECT,500,300,30,30,u,88
LABEL,500,300,L1
```

Le voyant sera vert si u88 est à 1 ou gris foncé si u88 est à 0

Nous allons maintenant envoyer des commandes en fonction de la zone du TCO ou l'on appuie.
Par exemple pour transformer l'indicateur précédent en interrupteur qui changera la valeur de la variable "u" à chaque appuie :

```
RECT,500,300,30,30,u,88
LABEL,500,300,L1
ACTION,500,300,30,30,u88^
```

Pour le debug, vous pouvez afficher les zones de click en mettant à 1 la variable suivante dans le code de la souris :

```
    ! visualisation des zones de click (0 normalement)
    see_click_zone = 1
```

Pour commander l'aiguillage, nous allons ajouter une zone englobant les 2 branches

```
LIGNE,200,100,300,100,-,8
LIGNE,200,100,300,200,/,8
ACTION,250,150,50,50,t8^
```

3.3. Commande des accessoires et automatismes sur la centrale

Vous pouvez utiliser les fonctions suivantes pour contrôler votre électronique :

** Pour les sorties directes de la carte Arduino

```
user_out(out#<D0,D1,D4,D5,D6,D7,D8>, val<0_1>)
```

** Pour les LEDs sur les MAX

```
user_led(led#<0-127>, val<0-1>)          //allumage/extinction
user_led_cli(led#<0-127>, val<0-1>)      //mode clignotant
user_led_pha(led#<0-127>, val<0-1>)      //inversion de la phase de clignotement
```

** Pour les sorties des PCA

```
user_pwm_0_100(out#<0-47>,<0-100%>)
```

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

```
user_servo_500_2500(out#<0-47>, <500-2500>)
```

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

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

vitesse de rotation du servo (pas encore dispo)

** Pour les décodeurs

```
user_bas_acc_dec_tx(adr#<1-510>, out<0-7>, val<0-1>)
```

Pour transmettre un ordre à un décodeur d'accessoire basique

```
user_ext_acc_dec_tx(adr#<1-2044>, val<0-31>)
```

Pour transmettre un ordre à un décodeur d'accessoire étendu

** Pour lancer des tempos

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

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

** Pour mettre à jour les indicateurs des positions des aiguillages (d=direct et e=evitement) et variables u

```
user_set_d_e(aig#<0-47>, val_d<0-1>, val_e<0,1>)
user_set_u(u#<0-95>, val_u<0-1>)
```

** Pour savoir l'état d'une variable u

```
user_get_u(u#<0-95>)    //retourne 0 ou 1
```

** Pour savoir l'état d'une entrée

```
user_get_in(in#<0-95>)    //retourne 0 ou 1
```

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

```
void user_init(void)
{
}

void user_notify_aig(byte num, byte cmd)    //cmd=0=direct cmd=1=devie
{
    user_set_d_e(num, 1-cmd, cmd);
}

void user_notify_u(byte num, byte val)
{
    user_set_u(num, val);
}

void user_notify_tempo_end(byte num_tempo)
{
}

void user_250ms(void)    // au lieu de 125ms
{
}
```

user_init() est appelée une fois au démarrage de la centrale. Vous pouvez ainsi y mettre vos initialisations.
user_notify_aig() est appelé chaque fois qu'une souris demande de modifier la position d'un aiguillage.
user_notify_u() est appelé chaque fois qu'une souris demande de modifier une variable u.
user_notify_tempo_end() est appelé chaque fois qu'une tempo que vous avez lancez se termine.
user_250ms() est appelé toutes les 250ms.

Vous remarquerez qu'il y a déjà du code dans user_notify_aig. Ce code par défaut met à jour les variables de position des aiguillages en fonction de la commande. Si cmd=0 (demande de position directe), alors d est mise à 1 et e à 0. Vous pouvez modifier cela, surtout si vos aiguillages prennent du temps à déplacer. Dans ce cas, vous mettrez à 0 les 2 variables d et e, puis mettrez la bonne à 1 en fin de déplacement. Pour commencer, vous pouvez laisser ce code et l'ignorer. Même comportement pour les variables u.

Suivant que vous voulez utiliser ou non des modules MAX, des modules PCA, des modules S88 vous devrez mettre à 1 ou 0 les constantes suivantes :

```
// PIN assignation
#define USER_USE_S88  0     // 1 to use S88 IN     (out D1, D6, D7, D8 no more available)
#define USER_USE_MAX  0     // 1 to use MAX7219/21 (out D4, D6, D7     no more available)
#define USER_USE_PCA  0     // 1 to use PCA9685    (out D0, D5         no more available)
#define USER_CLI_LED_MAX -1 // indicate which LED of MAX should blink to indicate sign of life
(-1 not used)
```

Si les MAX sont utilisés alors la LED de la carte ne clignotera plus pour montrer que la carte est vivante (sign of life) ou pour signaler un arrêt d'urgence. Pour combler ce manque, il est possible d'indiquer le numéro d'une LED du MAX qui clignotera a sa place. Si vous ne souhaitez pas avoir cette LED, spécifiez -1.

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

- Utiliser // pour un commentaire (tout ce qui suit jusqu'en fin de ligne est un commentaire)
- L'instruction if exécute ce qui est entre {} si le test entre () est vraie.

- On peut ajouter else (sinon) et ce qui est entre {} sera exécuté dans le cas contraire donc si le test est faux.
- Pour tester si une variable vaut une value, il faut utiliser == (2x=) et non 1 seul = qui est l'affectation.
- Ne pas oublier le ; après chaque appel de fonction
- Si vous utiliser une constante souvent, vous pouvez la définir avec #define, ex #define AIG_3_DIRCET 5

Exemples

Les pages suivantes donnent quelques exemples, bien entendu il faudra générer les commandes associées à partir du TCO.

Pour tester, vous pouvez utiliser le bouton MSG. Comme :
t0- commande pour mettre l'aiguillage 0 en position directe
t3/ commande pour mettre l'aiguillage 3 en position déviée
t47^ commande pour inverser la position de l'aiguillage 47
u0- commande pour mettre à 0 la variable u0
u5+ commande pour mettre à 1 la variable u5
u95^ commande pour inverser la variable u95

Ex: commande d'un aiguillage branché sur un décodeur :

Aiguillage numéro 18 pour la centrale et le TCO
Décodeur basique adresse 10
- commande sortie 0 pour mettre l'aiguillage en position directe
- commande sortie 1 pour mettre l'aiguillage en position déviée

```
void user_notify_aig(byte num, byte cmd)
{
    if(num = 18)
    {
        if(cmd==0) { user_bas_acc_dec_tx(10, 0, 1); }
        if(cmd==1) { user_bas_acc_dec_tx(10, 1, 1); }
    }

    user_set_d_e(num,cmd^1,cmd);
}
```

Ex: commande d'un itinéraire :

```
--aig12---aig13--- iti 0  (u10)
     \        \----- iti 1  (u11)
      \-------------- iti 2  (u12)

aig12 sur decodeur 10 (sortie0=direct 1=devie)
aig13 sur decodeur 10 (sortie2=direct 3=devie)
```

```
void user_notify_u(byte num, byte cmd)
{
    if(u = 10 &&  cmd==1)
    {
        user_bas_acc_dec_tx(10, 0, 1); user_set_d_e(12, 1, 0);
        user_bas_acc_dec_tx(10, 2, 1); user_set_d_e(13, 1, 0);
        user_set_u(11, 0); user_set_u(12, 0);
    }
    if(u = 11 &&  cmd==1)
    {
        user_bas_acc_dec_tx(10, 0, 1); user_set_d_e(12, 1, 0);
        user_bas_acc_dec_tx(10, 3, 1); user_set_d_e(13, 0, 1);
        user_set_u(10, 0); user_set_u(12, 0);
    }
    if(u = 12 &&  cmd==1)
    {
```

```
        user_bas_acc_dec_tx(10, 1, 1); user_set_d_e(12, 0, 1);
        user_set_u(11, 0); user_set_u(11, 0);
    }
    user_set_u(num, cmd);
}
```

J'ai poussé un peu le raffinement à mettre à 0 les itinéraires non compatibles. Ainsi les anciennes commandes s'éteindront sur le TCO.

On peut faire plus simple surtout si les aiguillages peuvent aussi se piloter manuellement.
Le code du pilotage des aiguillages étant dans user_notify_aig().

```
void user_notify_aig(byte num, byte cmd)
{
    if(num = 12)
    {
        if(cmd==0) { user_bas_acc_dec_tx(10, 0, 1); }
        if(cmd==1) { user_bas_acc_dec_tx(10, 1, 1); }
    }

    if(num = 13)
    {
        if(cmd==0) { user_bas_acc_dec_tx(10, 2, 1); }
        if(cmd==1) { user_bas_acc_dec_tx(10, 3, 1); }
    }

    user_set_d_e(num,cmd^1,cmd);
}

void user_notify_u(byte num, byte cmd)
{
    if(u = 10 &&  cmd==1)
    {
        user_notify_aig(12, 0);
        user_notify_aig(13, 0);
        user_set_u(11, 0); user_set_u(12, 0);
    }
    if(u = 11 &&  cmd==1)
    {
        user_notify_aig(12, 0);
        user_notify_aig(13, 1);
        user_set_u(10, 0); user_set_u(12, 0);
    }
    if(u = 12 &&  cmd==1)
    {
        user_notify_aig(12, 1);
        user_set_u(11, 0); user_set_u(11, 0);
    }
    user_set_u(num, cmd);
}
```

Ex de code qui n'active pas un itinéraire si un autre est déjà actif. L'instruction return sort de la fonction et donc ne met pas à jour la variable u qui reste à 0. Sur le TCO le bouton ne change donc pas de couleur. Il faudra desactiver l'itinéraire actif pour activer le nouveau. Dans ce cas sécurisé on ne codera pas la commande directe des aiguillages depuis le TCO.

```
void user_notify_u(byte num, byte cmd)
{
    if(u = 10 &&  cmd==1)
    {
        if(get_u(11)==1) { return; }
        if(get_u(12)==1) { return; }
        user_notify_aig(12, 0);
        user_notify_aig(13, 0);
    }
    if(u = 11 &&  cmd==1)
    {
        if(get_u(10)==1 || get_u(12)==1) { return; }
        user_notify_aig(12, 0);
```

```
        user_notify_aig(13, 1);
    }
    if(u = 12 &&  cmd==1)
    {
        if(get_u(10) || get_u(11)) { return; }
        user_notify_aig(12, 1);
    }
    user_set_u(num, cmd);
}
```

On peut encore raffiner, par ex en n'autorisant pas un itinéraire incompatible, en verrouillant les aiguillages d'un itinéraire, en gérant la signalisation ...

Ex: lampadaire branché sur la sortie pwm17 (2eme sortie du second PCA)

```
void user_notify_u(byte num, byte cmd)
{
    if(u = 88)
    {
        if(cmd==0) { user_pwm_0_100(17, 0); }
        if(cmd==1) { user_pwm_0_100(17, 100); }
    }

    user_set_u(num, cmd);
}
```

Ex: Carré violet à LEDs

```
#define LED_VIOLETTE 17
#define LED_BLANCHE 18

void user_notify_u(byte num, byte cmd)
{
    if(u = 10)
    {
        if(cmd==0) { user_led(LED_VIOLETTE, 1); user_led(LED_BLANCHE, 0); }
        if(cmd==1) { user_led(LED_BLANCHE, 0); user_led(LED_VIOLETTE, 1); }
    }

    user_set_u(num, cmd);
}
```

Ex: Carré violet mécanique

```
#define LED_VIOLETTE 17
#define LED_BLANCHE 18
#define SERVO_CV101 3
#define SERVO_CV101_FERME 1500
#define SERVO_CV101_OUVERT 2000

void user_init(void)
{
    user_servo_500_2500(SERVO_CV101_FERME);
    user_led(LED_VIOLETTE, 1); user_led(LED_BLANCHE, 0);
}

void user_notify_u(byte num, byte cmd)
{
    if(u = 10)
    {
        if(cmd==0)
        {
            user_servo_500_2500(SERVO_CV101_FERME);
            user_led(LED_VIOLETTE, 1); user_led(LED_BLANCHE, 0);
        }
        else
```

```
        {
            user_servo_500_2500(SERVO_CV101_OUVERT);
            user_led(LED_BLANCHE, 0); user_led(LED_VIOLETTE, 1);
        }
    }

    user_set_u(num, cmd);
}
```

Ex: Aubinage d'un sémaphore mécanique

L'utilisateur peut ouvrir et fermer le sémaphore, mais il se fermera automatiquement si in0 est active.

```
#define LED_ROUGE_S103 19
#define LED_VERTE_S103 20
#define SERVO_S103 3
#define SERVO_S103_FERME 1500
#define SERVO_S103_OUVERT 2000
#define IN_S103 2

void user_init(void)
{
    user_servo_500_2500(SERVO_S103_FERME);
    user_led(LED_ROUGE_S103, 1); user_led(LED_VERTE_S103, 0);
}

void user_notify_u(byte num, byte cmd)
{
    if(u = 11)
    {
        if(cmd==0)
        {
            user_servo_500_2500(SERVO_S103_FERME);
            user_led(LED_ROUGE_S103, 1); user_led(LED_VERTE_S103, 0);
        }
        else
        {
            user_servo_500_2500(SERVO_S103_OUVERT);
            user_led(LED_ROUGE_S103, 0); user_led(LED_VERTE_S103, 1);
        }
    }

    user_set_u(num, cmd);
}

void user_250ms(void)
{
    if(get_in(IN_S103) { user_notify(11, 0) ; }
}
```

Ex: Clignotement des LEDs d'un passage à niveau sans barrières

Cet exemple ne nécessite pas de commande de la part du TCO

```
#define LED_PN_A 19
#define LED_PN_B 20

void user_init(void)
{
    user_led_cli(LED_PN_A, 1);
    user_led_cli(LED_PN_B, 1); user_led_pha(LED_PN_B, 1);
}

void user_250ms(void)
{
    if(user_get_in(14)==1)
    {
```

```
        user_led(LED_PN_A, 1);
        user_led(LED_PN_B, 1);
    }
    else
    {
        user_led(LED_PN_A, 0);
        user_led(LED_PN_B, 0);
    }
}
```

Ex: Feu de BAL (Block Automatic Lumineux)

Cet exemple ne nécessite pas de commande de la part du TCO.
Le canton suivant est relié à un détecteur de courant branché sur in1
Celui d'après sur in2
Pour que cet exemple puisse fonctionner, les essieux doivent être résistifs

```
  >S102
---  -----in1-----  -----in2-----
Z12       Z13            Z14
```

```
#define S102_LED_ROUGE 0
#define S102_LED_VERTE 1
#define S102_LED_JAUNE 2
#define IN_Z13   1
#define IN_Z14   2

void user_250ms(void)
{
    user_led(S102_LED_ROUGE, 0); user_led(S102_LED_VERTE, 0); user_led(S102_LED_JAUNE, 0);

    if(user_get_in(0)==1)
    {
        user_led(S102_LED_ROUGE, 1);
    }
    else
    {
        if(user_get_in(1)==1)
        {
            user_led(S102_LED_JAUNE, 1);
        }
        else
        {
            user_led(S102_LED_VERTE, 1);
        }
    }
}
```

Comme vous disposez de la position des aiguillages, de l'état des itinéraires vous pouvez faire des choses encore plus complexes. Si les essieux sont isolés, il faudra être capable de suivre la loco et la ça devient très compliqué !

Le protocole de communication dispose également d'autres fonctions pour envoyer des ordres aux décodeurs d'accessoire ou manipuler directement les LEDs, sorties, sorties PWM, servos rattachés à la D17.
Il est bien entendu préférable d'utiliser les variable "u" ne serait-ce que pour toutes les tablettes soient au courant des modifications. Si vous n'utilisez qu'une tablette en TCO, l'intérêt des variable u est plus limité

Le bouton « msg » peut être utilisé pour envoyer de tel messages.

Par exemple pour activer la sortie 0 du décodeur d'accessoire basique d'adresse 10
x10.0+
Pour allumer la led 28
l28+
Pour la mettre en mode clignotant
c28+
Pour faire clignoter la led 29
l29+c29+p29+
Mettre le servo 15 en position neutre
s15=150

Vous pouvez voire tous les échanges en utilisant le "Serial Monitor" de l'environnement Arduino réglé à 115200bps.

3.4. Scripts d'automatisation sur la souris double

Les bases :

Il est désormais possible d'écrire des scripts et de les exécuter sur les souris doubles afin d 'automatiser des séquences de jeu. Un script est un fichier texte qui contient une séquence d'instructions. La souris exécute les instructions une à une.

Les instructions sont les suivantes :

Commande	Description
CMD la_commande_sans_espace	
MSG le_message_sans_espace	Affichage d'un message « popup » sur la souris
TEMPO durée_en_s	Attente fixe
TEMPOR durée_min_en_s durée_max_en_s	Attente aléatoire
WAITIN numéro_de_l'entrée<0-47> WAITNOTIN numéro_de_l'entrée<0-47>	Attente jusqu'à ce que l'entrée spécifiée soit active (WAITIN) ou inactive (WAITNOTIN)
WAITU numéro_de_la_variable_u<0-95> WAITNOTU numéro_de_la_variable_u<0-95>	Attente jusqu'à ce que la variable U spécifiée soit active (à 1) (WAITU) ou inactive (à 0) (WAITNOTU)
END	Fin du script
LOOP	Recommencer le script depuis le début

Vous pouvez donc :
- envoyer des commandes à la centrale avec CMD
- attendre durant le temps spécifié avec TEMPO pour une attente fixe ou TMPOR pour une commande aléatoire
- attendre une entrée ou une variable U avec WAIT[NOT]<IN/U>
- afficher des messages « popup » avec MSG sur la souris afin de savoir où vous en êtes.

Les instructions peuvent être en majuscules ou minuscules.

Avec la commande CMD, vous pouvez commander tout ce que vous voulez en utilisant le protocole de communication des souris. Notez que vous pouvez à tout moment essayer d'envoyer une commande avec ce protocole en utilisant le bouton « MSG » du TCO. Vous trouverez la version à jours du protocole dans le code de la centrale. A la date d'écriture de cette section, voici ce protocole :

```
/* Protocole
 *
 * au1 = arret d'urgence, le booster est desactive
 * au0 = fin d'arret d'urgence
 *
 * a<1-99>   = adresse de la locomotive et selection du canal A de la souris
 * b<1-99>   = adresse de la locomotive et selection du canal B de la souris
 * s+        = sens 1 (pour le canal selectionne)
 * s-        = sens 0
 * v<0-100>  = vitesse en %
 * f<0-28>+  = allumage fonction auxiliaire
 * f<0-28>-  = extinction fonction auxiliaire
 *
 * cva<1-1024> cvd<0-255> cvp<XXXX> = programation d'un decodeur de locomotive cva=adresse du
CV, cvd=data cvp=code de protection a fixer dans ce programme afin d'interdir les programmation
non autorisees
 *
 * x<1-511>.<0-7>+  = activation d'une sortie d'un decodeur d'accessoire
 * x<1-511>.<0-7>-  = desactivation d'une sortie d'un decodeur d'accessoire (normalement jamais
utilise car l'activation de 0 desactive 1 et vice versa, ou alors si le decodeur est configure
en impulsion, les sorties reviennent a 0 toutes seules)
 * y<1-2044>=<0-31> = afectation de la valeur 0-31 a un decodeur d'accessoire etendu
 *
 * o<0-5>+   = mise a 1 d'une sortie directe
 * o<0-5>-   = mise a 0 d'une sortie directe
 *
 * l<0-127>+ = allumage d'une led
 * l<0-127>- = extinction d'une led
 * c<0-127>+ = mode clignotant
 * c<0-127>- = mode fixe
 * h<0-127>- = phase normale
 * h<0-127>+ = phase inverse
 *
 * p<0-47>=<0-100>  = pwm (sur i2c)
 * s<0-47>=<50-250> = servo(sur i2c) = pulsation en 10us, neutre en 150*10us=1.5ms
 * s<0-47>-         = mettre le servo en position 0
 * s<0-47>+         = mettre le servo en position 1
 * Le choix entre pwm et servo se regle au debut du fichier d ela centrale
 * Pour utiliser les sorties en 0/1, utiliser pwm0/100
 * La vitesse de rotation des servos, ainsi que les positions extremes sont reglables
 *
 * t<0-47>- = aiguillage en position direct (t=turnable, aiguillage en Americain)
 * t<0-47>/ = aiguillage en position devie
 * t<0-47>^ = inversion de la position de l'aiguillage
 * l'aiguillage peut etre de n'importe quel type(bobine, moteur, serv) et situe n'importe ou(en
sortie de l'esp, sur un module i2c pca9685, sur un decodeur d'accessoire),
 * c'est a l'utilisateur de le definir dans ce programme. Il est egalement possible d'indiquer
la duree de l'impulsion et inverser la commande
 * les rapport d et e sont mis a jour un efois la commande effectuee
 *
 * u<0-95>- = mettre a 0 la variable utilisateur
 * u<0-95>+ = mettre a 1 la variable utilisateur
 * u<0-95>^ = inverser la variable utilisateur
 *
 * ? = demande de l'emission de l'etat
 * au[0/1] ana/dcc iXXXXXXXXXXXXXXXXXXXXXXXXX dXXXXXXXXXXXX eXXXXXXXXXXXX
uXXXXXXXXXXXXXXXXXXXXXXXX cYY
 * 01 2   3456
78901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901
 * 0                 1         2         3         4         5         6         7         8
9          10
 * X = codage en hexa 0-F pour transmettre 4 bits par caracter afin de ne pas surcharger le lien
 *     on aurait pu passer en binaire pour encore plus gagner, mais je voulais rester en ASCII
 * i = entree (0-95)
 * u = etat des commandes utilisateur
 * d = aiguillage en position directe (0-47)
 * e = aiguillage en position devie   (0-47)
 * c = convertisseur analogique numerique (souvent utilise pour mesurer le courant
 * YY = 0(0V), 99(1V)
 *
 * ! = information sur la centrale: ie: D17 v20170106a
 *
 */
```

Pour les locos, chaque souris à la possibilité de contrôler 2 locos la « a » et la « b ». Il faut spécifier son adresse après la lettre a ou b. Si cela est omis, la centrale considère qu'il s'agit du canal a et de l'adresse précédemment enregistrée. Ensuite vous pouvez changer le sens avec s+ ou s-, la vitesse avec v<0-100>, les fonctions auxiliaires avec f<0-28><+/->.

Exemple de script pour faire avancer la loco 8 pendant 5s avec les feux allumés pendant le déplacement.

```
CMD a8s+v20f0+
TEMPO 5
CMD v0
TEMPO 2
CMD f0-
```

Pour commander les aiguillages, vous pouvez utiliser comme décrit dans le chapitre du TCO t<0-47><- />
Dans ce cas la fonction user_notify_aig() de la centrale doit décrire comment commander cet aiguillage.

Pour commander les accessoires, vous pouvez utiliser les variables U avec u<0-95><+/->
Dans ce cas la fonction user_notify_u() de la centrale doit décrire comment commander l'accessoire.

Il est également possible de commander directement :
- les décodeurs d'accessoires avec x<adresse1-510>.<sortie0-7><+/->
- les leds du bus SPI de la centrale avec l<0-127><+/-> pour l'allumage, c<0-127><+/-> pour le clignotement h<0-127><+/-> pour la phase
- les servos des modules PCA9685 du bus I2C de la centrale avec s<0-47>=<50-250> (durée de l'impulsion en 10ms, 150=1500ms=neutre)
- les sorties PWM des modules PCA9685 du bus I2C de la centrale avec p<0-47>=<0-100>
- les sorties directes de la centrale avec o<0,1,5,6,7,8>

Notez qu'il est préférable d'utiliser les commandes des aiguillages et variables U au lieu des commandes directes afin que l'état du système soit partagé avec les autres tablettes.

Exemples :

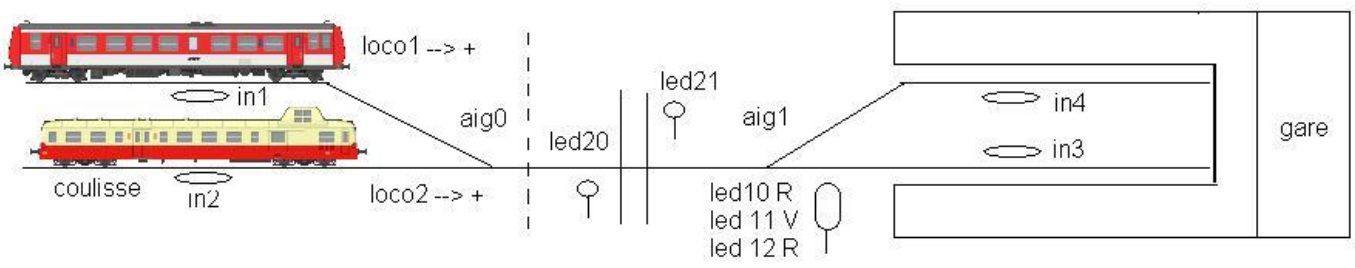
Utilisons le réseau suivant pour créer quelques scripts.

Ce petit réseau comporte :

- une coulisse de 2 voies. Les détecteurs de courant in1 et in2 détectent l'occupation. L'utilisation d'ILS n'est pas recommandé pour le moment car on peut les louper, mais je les supporterai un peu plus tard. Le choix de la voie est fait par l'aiguillage 0.

- Après la coulisse, on trouve un passage à niveau que nous commanderons avec la variable u32 (fermé à 1, ouvert à 0). Bien entendu, il faudra gérer ce passage à niveau dans la fonction user_notify_u() de la centrale.

- Après le PN, on trouve une gare terminus à 2 voies. Un Carré protège la voie unique avec la variable u28 (carré à 1, voie libre à 0).



Exemple avec un A/R du picasso in2 > in3 > in2

```
# Super script qui fait un A/R du picasso

MSG script_AR_picasso
CMD u32+u28+t0-t1-
TEMPO 2
CMD a2s+v20f0+
WAITIN 3
CMD v10
TEMPO 2
CMD v0f0-u32-

TEMPOR 20-60

MSG depart
CMD u32+u28-s-v10f0+
WAITNOTIN 3
CMD v20u28+
WAITIN 2
TEMPO 2
CMD v0f0-u32-

END  #faculatif
```

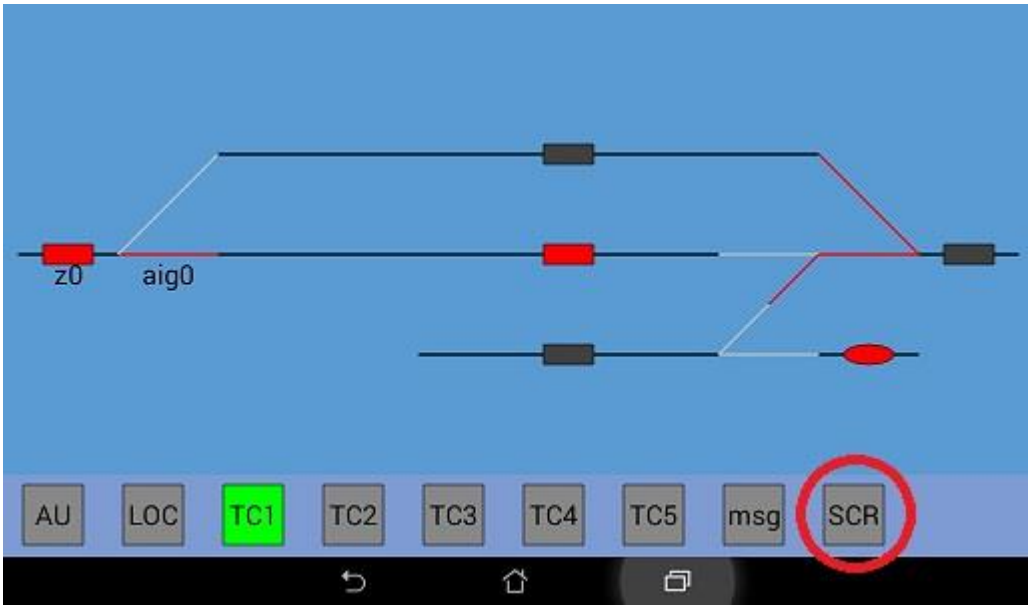
Exemple A de l'X2200 in1 > in4, A du picasso in2 > in3, R de l'X2200 in4 > in 1, R du Picasso in3 > in2

```
MSG A_X2200
CMD u32+u28+t0/t1/
TEMPO 2
CMD a1s+v20f0+
WAITIN 4
TEMPO 2
CMD v0f0-u32-

TEMPOR 10-30

MSG A_PICASSO
CMD u32+u28+t0-t1-
TEMPO 2
CMD a2s+v20f0+
WAITIN 3
TEMPO 2
CMD v0f0-u32-

TEMPOR 20-50

MSG R_X2200
CMD u32+u28-t0/t1/
TEMPO 2
CMD a1s-v20f0+
WAITNOTIN 4
CMD v28+
WAITIN 1
TEMPO 2
CMD v0f0-u32-

TEMPOR 10-30

MSG R_PICASSO
CMD u32+u28-t0-t1-
TEMPO 2
CMD a2s-v20f0+
WAITNOTIN 4
CMD v28+
WAITIN 2
TEMPO 2
CMD v0f0-u32-

TEMPOR 30-60

LOOP   #on execute en boucle tant que l'on ne desactive pas le script
```

Utilisation :

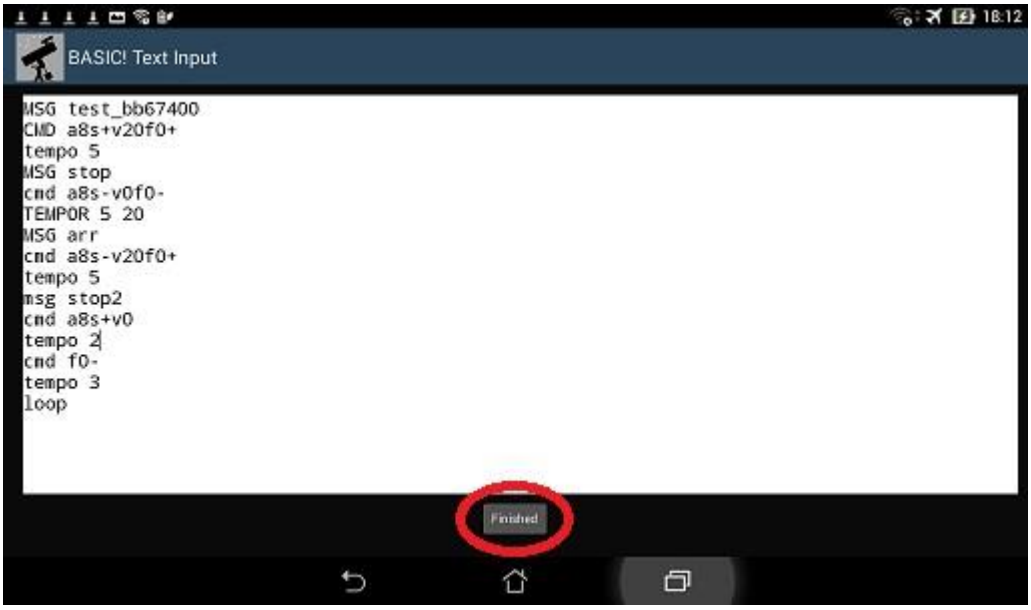
1) Premièrement, vous devez créer un fichier texte avec votre script dans le répertoire « rfo_basic/data » de votre appareil Android. Pour cela, vous pouvez utiliser une application Android capable d'éditer un fichier texte ou alors l'écrire sur un PC puis le transférer par USB dans l'appareil Android. Lorsque l'on branche un appareil Android sur un PC, on voit ses fichiers comme s'il s'agissait d'une clé USB. Ne mettez pas les scripts dans « rfo_basic/source » comme pour les programmes des souris mais bien dans « rfo_basic/data ».

2) Rendez-vous dans l'écran du TCO et appuyer sur le bouton SCR (pour SCRipt).

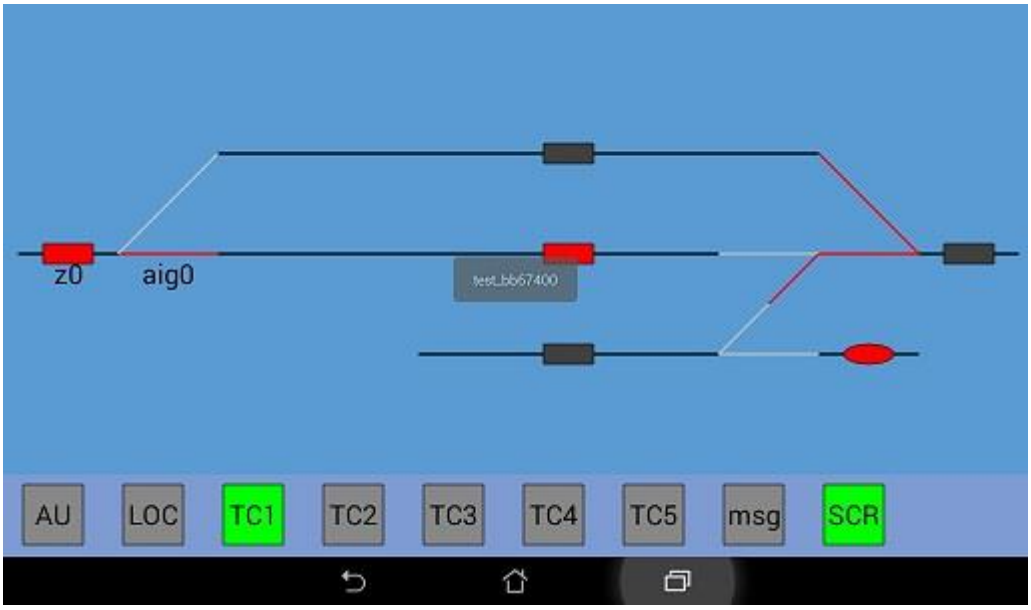


3) Sélectionnez votre script

4) Un éditeur s'ouvre pour vous laisser faire quelques réglages de dernières minutes. Sauvez pour sortir.



5) Le bouton SCR s'allume en vert puis le script commence son exécution. Vous verrez alors le résultat de votre travail. Sur la souris, vous verrez vos messages popup, les aiguillages et occupations changer si vous avez utilisé les variables des aiguillages et variables U.
Par contre vous ne verrez pas les commandes des locomotives bouger. Une fois le script lancé, vous n'êtes pas obligé de rester sur l'écran du TCO. Sur l'écran de conduite, évitez de jouer avec les potentiomètres des canaux (a ou b) des locos en mouvement. Mais si vous utilisez le canal a dans votre script, rien ne vous empêche de conduire une loco manuellement sur le canal b. Avant de lancer un script, vérifiez que vous n'êtes pas en AU et que ACC et UM ne sont pas utilisés.

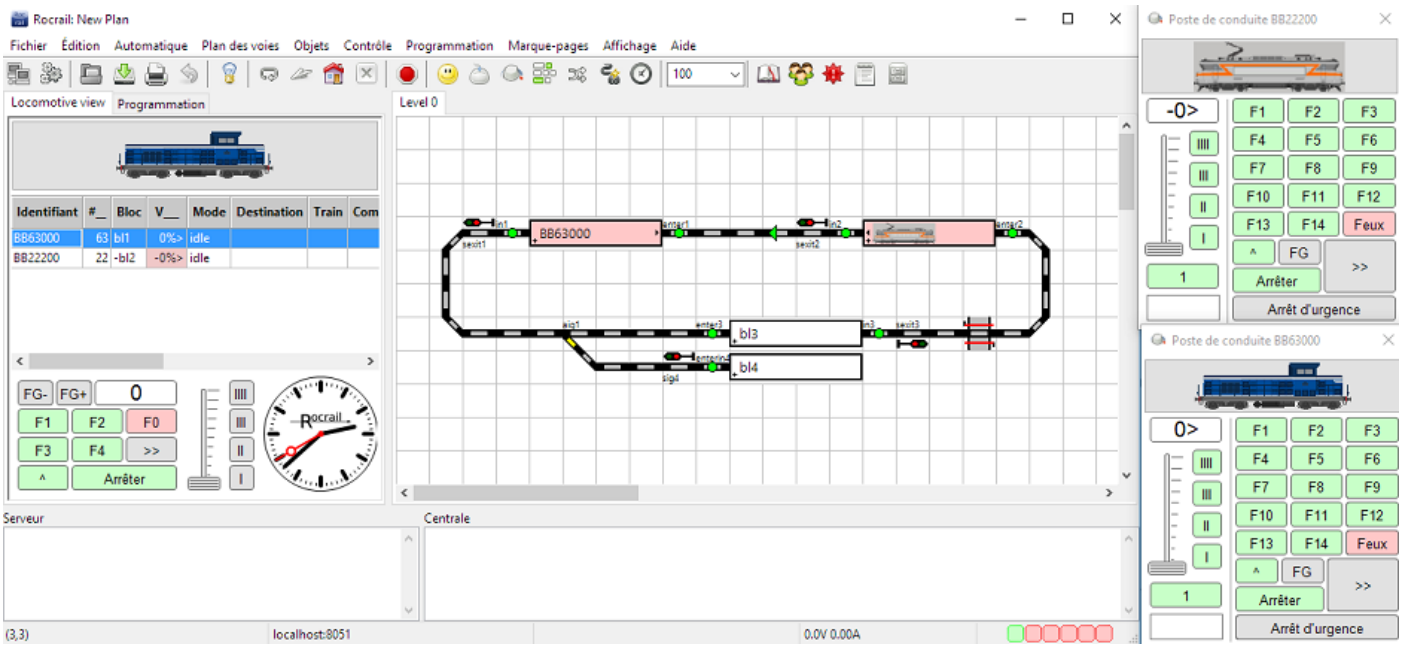


6) En fin de fichier ou à la suite de l'instruction END, le bouton SCR s'éteint et le script s'arrête
Vous pouvez à tout moment arrêter le script en appuyant sur le bouton SCR qui s'éteint.
Bien entendu en cas de problème, utilisez le bouton AU

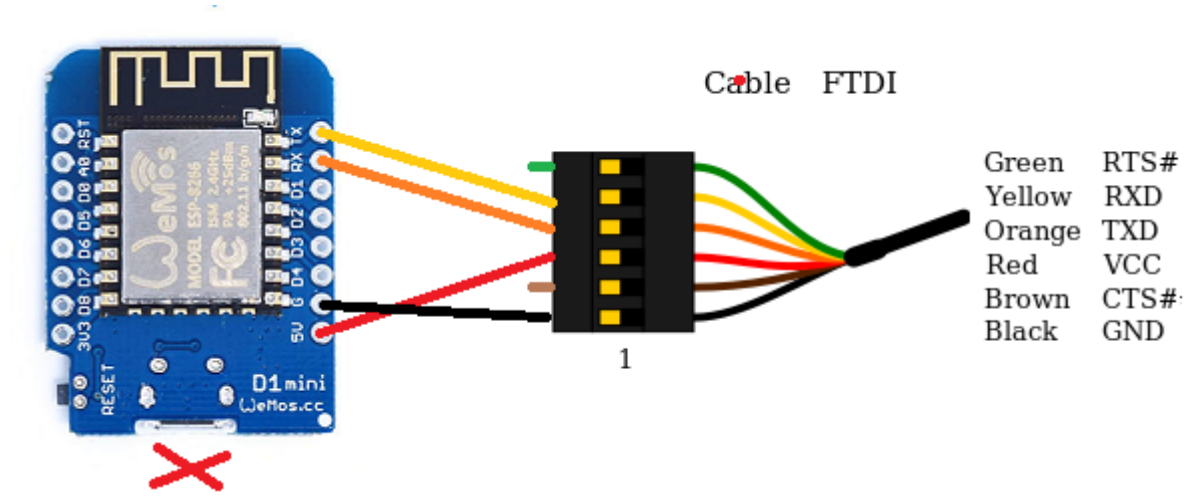
3.5. Pilotage par des logiciels PC

La grande force de D17 est de se passer de PC et être simple.
D17 est bien adapté pour une utilisation manuelle ou une automatisation légère.
Pour une automatisation plus lourde, il est possible d'utiliser un PC avec un bon logiciel.
Une version light du protocole DCC++ peut être activée dans le code de la centrale pour cela.

Le logiciel RocRail peut être utilisé pour piloter D17 par USB avec la version light du protocole DCC++.
Ce logiciel nécessite le cantonement du réseau avec idéalement 2 détecteurs par canton (un en début et un en fin de canton).
Il propose un mode manuel, un mode semi-automatique ou l'on glisse une loco d'un canton sur un autre et un mode automatique avec lequel les horaires peuvent être utilisées.



Malheureusement la DLL de RocRail qui gère DCC++ joue avec le signal RTS ce qui reset le Wemos.
Il est possible de résoudre ce problème en utilisant un câble USB-Serie (ex: FDTI 3.3V) à la place de l'USB du Wemos.



Le logiciel CDM-Rail devrait aussi être utilisable dans le futur car l'auteur m'a dit qu'il supporterai le protocole DCC++ prochainement.

Je ne sais pas encore si je modifierai le logiciel Free-DCC 2017 pour supporter le protocole DCC++ et donc la centrale D17 ...

4. Conclusion

Vous savez maintenant tout sur la centrale DCC WIFI D17. J'espère que ce document aura été suffisamment 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/

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

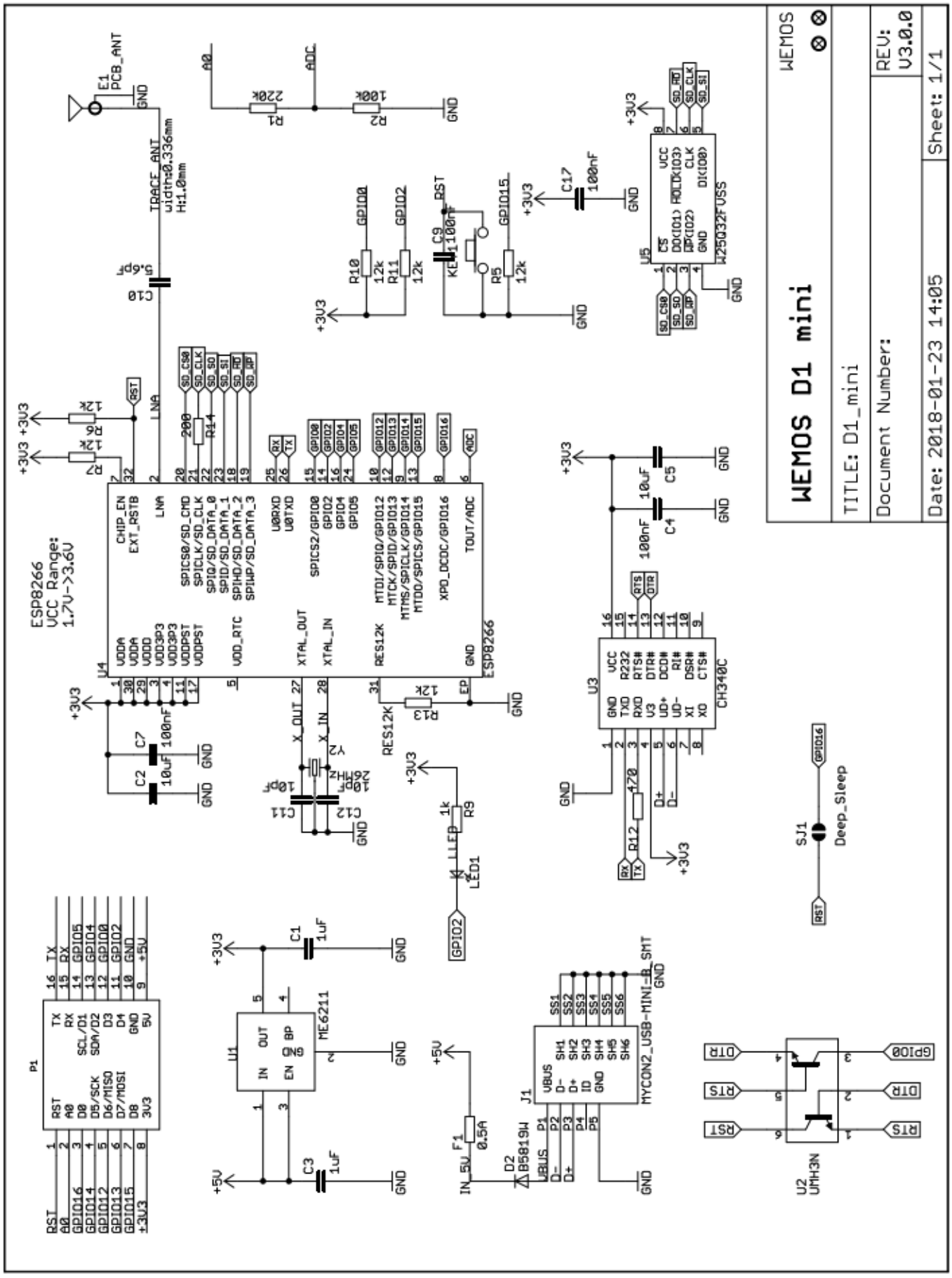
Quelques Liens :
- DCC NMRA: http://www.nmra.org/dcc-working-group
- Arduino: https://www.arduino.cc/
- Un bon article sur les trames DCC: http://trainminiature.discutforum.com/t12784-dcc-comment-ca-marche-place-a-la-technique

Limitations actuelles :
- Mouvements lents pour les servos pas encore disponible
- Multiplexage pas disponible
- Pas de protocole pour commander la centrale par un logiciel sur PC

La petite histoire :

Vous pouvez vous demander pourquoi ai-je réalisé cette centrale. En fait je voulais juste essayer le module esp8266 pour le contrôler une LED par WIFI avec une interface graphique sur une tablette Android. A ce moment-là, j'ai réalisé mon réseau à base de modulinos et je voulais un système simple pour le piloter. J'aurais pu utiliser mon système Free-DCC, mais je voulais essayer quelque chose sans PC. Vu l'enthousiasme autour de ce projet, j'ai décidé de le faire évoluer… J'espère que cette centrale vous apportera satisfaction si vous décidez de le réaliser. Si vous souhaitez me remercier, envoyez-moi quelques photos de vos réalisations 😉

Module Wemos D1 mini



Carte PCA 9685