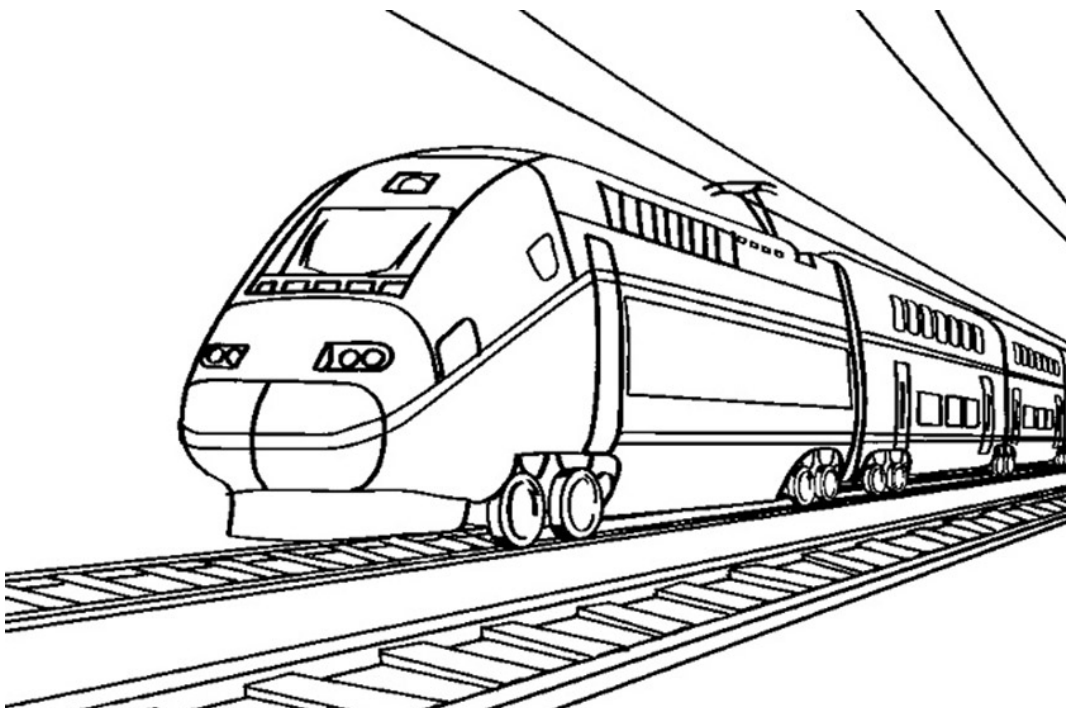


# Travail Pratique

## Gestion de station de train

---

Berger Antoine, Marques Rafael et Da Silva David  
Cours de Microcontrôleur ITI2



## **TABLE DES MATIERES**

### INTRODUCTION

### UTILISATION

Démarrage / Stop

Sélectionner un train

Changer la vitesse d'un train

Allumer les lumières d'un train

Changer le sens d'un train

### GESTION DE L'ECRAN

Initialisation de l'écran

Les fonctions créées

### COMMUNICATION CAN

Initialisation

Ecriture et lecture BusCan

Préparation des trames d'envoi

### COMMUNICATION UART

### COMMUNICATION ZIGBEE

### COMMUNICATION SPI

### CARTE SD

Initialisation

Le stockage des images

Les fonctions créées

Matériel et logiciels utilisés pour la gestion de la carte SD

### DALLE TACTILE

### INTERFACE GRAPHIQUE

### CONCLUSION GENERALE

Problèmes rencontrés

Améliorations possibles

## INTRODUCTION

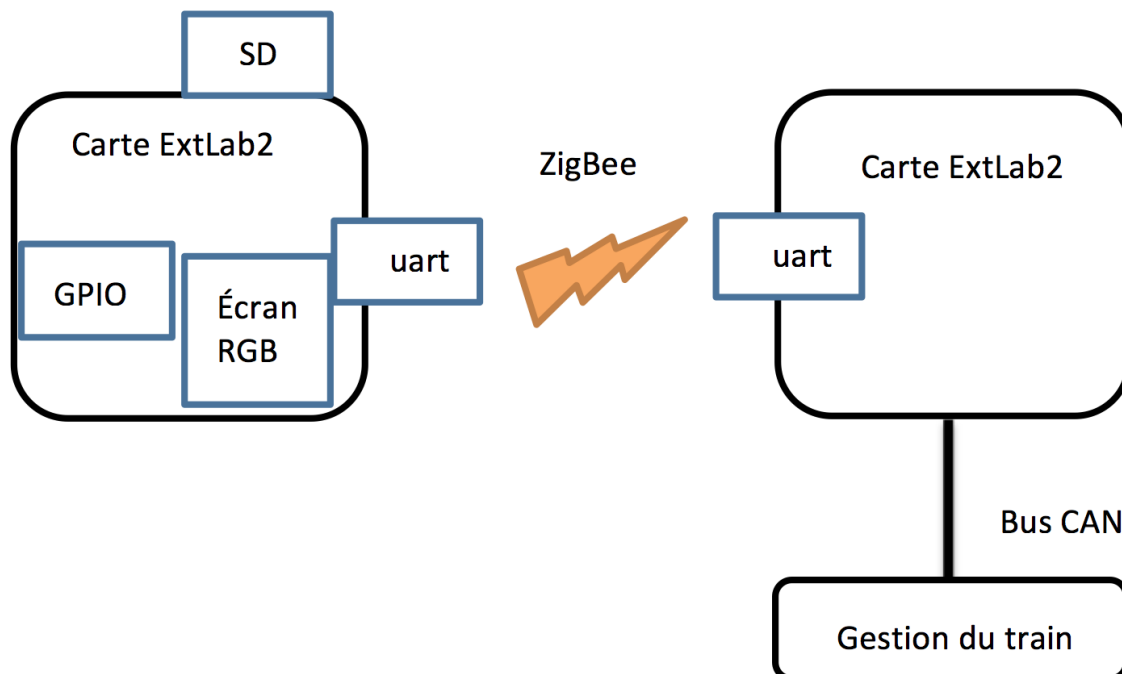
Dans le cadre du cours de microcontrôleur, un sujet de semestre nous à été imposé, la manipulation de train à distance à l'aide de carte ExtLab2. Le cahier des charges était pour le moins libre.

Nous avons choisi d'utiliser deux cartes ExtLab2, une qui sert à contrôler le train à distance et l'autre qui est utilisé pour la réception des commande. La communication entre les deux cartes se fait avec UART par le protocole ZigBee

Sur la carte de contrôle, nous utilisons l'écrans tactile pour afficher l'interface utilisateur. Cette interface qui est elle même stocké dans la carte SD. L'écran tactile permet aussi de sélectionner les différents paramètre modifiable tels que, n° de locomotive, vitesse, sens, lumière et le power.

Le seul GPIO utilisé est le bouton rotatif, ce dernier permet de incrémenter ou décrémenter le n° de locomotive ou la vitesse.

Sur la carte de réception, il y a l'interprétation des données envoyées par la carte de contrôle et la préparation des trames CAN à envoyer sur la station.



## UTILISATION

### **Démarrage / Stop**

Il suffit d'appuyer sur le bouton en haut à gauche de l'écran. Lorsque le bouton est rouge le courant est coupé sur la console Marklin et les trains s'arrêtent. Il suffit de d'appuyer à nouveau pour que le bouton redevienne vert et que les trains redémarre.

### **Sélectionner un train**

Pour sélectionner un train, appuyé sur le bouton "*Train*" puis utiliser le bouton rotatif afin de choisir un train.

### **Changer la vitesse d'un train**

Pour changer la vitesse d'un train, appuyé sur le bouton "*Vitesse*" puis utiliser le bouton rotatif pour choisir une vitesse entre 0 et 1000.

### **Allumer les lumières d'un train**

Pour allumer ou éteindre les lumières du train, appuyé simplement sur le bouton lumière.

### **Changer le sens d'un train**

Pour changer le sens d'un train, appuyer sur le bouton "*sens*", la fleche du bouton indique le sens actuel du train. Attention, lors d'un changement de sens, le vitesse revient à 0. Il faut donc changer la vitesse apres chaque changement de sens.

## GESTION DE L'ECRAN

Les dimensions de l'écran sont de 320x240 pixels. On peut comparer l'écran à un tableau à deux dimensions de 320 par 240 cases. L'écran est initialisé en RGB, chacun des pixels de l'écran prend 3 valeurs en parametres pour définir sa couleur. Le curseur a pour rôle de parcourir chacun des pixels de gauche à droite et de leur attribuer une couleur.

### Initialisation de l'écran

L'initialisation se fait par le biais des différents GPIO utilisé par l'écran.

### Les fonctions créées

- `Init_ports_display()` :
- `Index_out(uint8_t idx):`
- `Parameter_out(uint16_t param):`
- `Send_color(uint8_t color):`
- `Init_display():`
- `Write_pixel(uint8_t red, uint8_t green, uint8_t blue):` Définis le pourcentage de couleur pour le pixel.
- `Set_cursor(uint16_t x, uint16_t y):` Place le curseur sur un point de l'écran
- `Create_partial_screen(uint16_t v_start, uint16_t v_end, uint8_t h_start, uint8_t h_end):` crée une fenetre sur l'écran

## COMMUNICATION CAN

Sur la carte ExtLab2, nous avons 2 port RJ11 qui utilise la communication CAN. Dans le cadre de notre Travail Pratique, nous avons besoin que d'un seul port. Ce dernier, est relier avec la station Märklin.

La Central Station Märklin, assure la prise en charge des formats usuels Motorola. Elle possède un grand écran tactile couleur et comprend également 2 boutons rotatifs de commande. La Station permet d' interagir directement avec les locomotives se trouvant sur les rails.

### **Initialisation**

Pour initialiser le BusCan il faut :	mettre le courant
	mettre l'horloge des périphériques
	standardiser la lecture de 1 bit (BTR)
	sélectionner le pin pour émettre
	sélectionner le pin pour recevoir
	si il y a besoin gérer les différents mode

### **Ecriture et ecriture BusCan**

Nous avons utilisé une structure qui possède identifiant du bus, dlc et les datas. Avant toutes écritures, on vérifie que le buffer d'écriture est libre. Lorsqu'il est libre, on va sélectionner quel longueur de trame il va écrire, quel identifiant il va transmettre, la longueur des données envoyées et finalement il sélectionne le buffer pour écrire et transmet la trame.

Pour la lecture, nous remplissons simplement notre structure avec les valeurs qui sont récupéré dans les registres servant à recevoir les données du BusCan.

### **Préparation des trames d'envoi**

Les trames sont préparé grâce aux fonctions : StopGoTrain, ChanceDirection, ChangeSpeed et TurnLight. Chaque paramètre de la structure sont initialisés avec les bonnes valeurs souhaitées. Ces trames sont expliquées dans la documentation allemande de Märkiln.

## COMMUNICATION UART

L'UART (*Universal Asynchronous Receiver Transmitter*) est un protocole composé de deux fils, pour l'envoi des données c'est le Tx et pour la réception c'est le Rx. Une trame UART est constitué des bits suivants:

Une trame UART est constituée des bits suivants<sup>1</sup> :

- un bit de *start* toujours à 0 : servant à la synchronisation du récepteur
- les données : la taille peut varier (généralement entre 5 et 9 bits)
- éventuellement un bit de parité paire ou impaire
- et un bit de *stop* toujours à 1 (la durée peut varier entre 1, 1,5 et 2 temps bit)



L'UART utilise des vitesses de transmissions normalisées, dans notre cas nous avons choisi une vitesse de 9600 bauds, car le nombre de données reçues par secondes est très faible. Nous avons également choisis d'avoir uniquement un bit de stop et sans parité.

Dans le cadre du projet nous avons tout d'abord commencé avec l'UART0, mais il s'est avéré qu'en raison des connections disponibles sur la carte ExtLab2 on devait utiliser l'UART3.

Le fichier `uart.c` contient toutes les fonctions pour initialiser, lire et écrire sur l'UART 0 et 3.

Pour la lecture il y a deux fonctions, la première qui se nomme "`uart3_read_one_char`" permet simplement de lire un caractère contenu dans le buffer de réception de l'UART, si il n'y a pas de caractères ou bien que les données reçues contiennent des erreurs il retourne un 0, si un caractère a été lu il retourne un 1. Le caractère lu est retourné via le pointeur en paramètre.

La deuxième fonction permet de lire un nombre de données fixes dans le buffer, la lecture s'arrête lorsqu'il n'y a plus de données. La fonction retourne le nombre de caractères lus.



L'écriture s'effectue simplement avec une fonction qui se nomme `uart3_send`, qui prend en paramètres le texte à envoyer ainsi que la taille à envoyer (nombre de bytes).

---

<sup>1</sup> <http://fr.wikipedia.org/wiki/UART>

## COMMUNICATION ZIGBEE

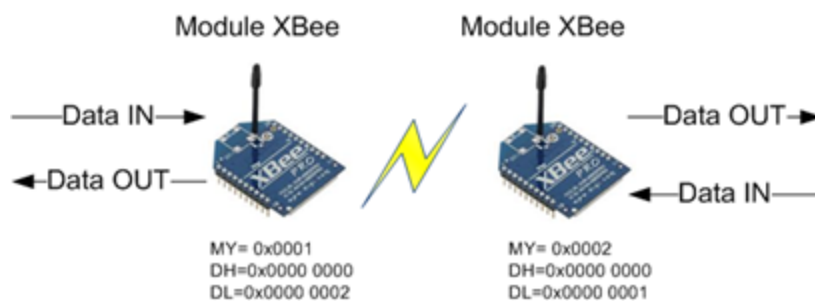
Dans le cadre de notre projet nous avons utiliser des modules XBee, ces modules utilisent une communication de type ZigBee et ont les caractéristiques suivantes :

Specifications		XBee 	XBee-PRO 
Performance	Indoor/Urban Range	up to 100 ft. (30 m)	up to 300 ft. (100 m)
	Outdoor RF line-of-sight Range	up to 300 ft. (100 m)	up to 1 mile (1.6 km)
	Transmit Power Output	1 mW (0 dBm)	60 mW (18 dBm)*, 100 mW EIRP*
	RF Data Rate	250,000 bps	250,000 bps
	Receiver Sensitivity	-92 dBm (1% PER)	-100 dBm (1% PER)
Power Requirements	Supply Voltage	2.8 – 3.4 V	2.8 – 3.4 V
	Transmit Current (typical)	45 mA (@ 3.3 V)	215 mA (@ 3.3 V, 18 dBm)
	Idle / Receive Current (typical)	50 mA (@ 3.3 V)	55 mA (@ 3.3 V)
	Power-down Current	< 10 µA	< 10 µA

Dans le cadre de notre projet nous avons des modules de type XBee et non XBee-PRO, mais il faut savoir que les deux modèles sont complètement compatibles entre eux.

L'avantage des modules XBee c'est que c'est des composants extrêments simple à utiliser pour des petits projets car leurs configurations sont êxtremements faciles à mettre en place.

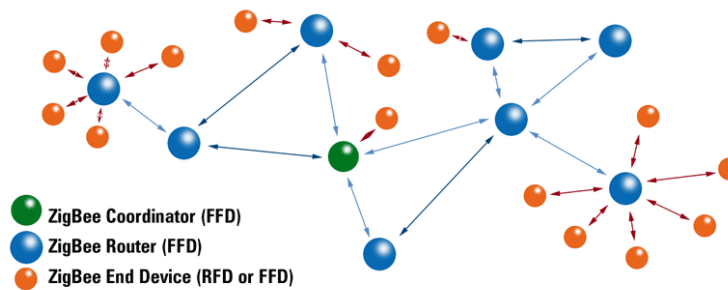
Il existe plusieurs modes de configurations, le plus simple est sûrement le point à point:



Dans ce type de configuration l'adresse MY (qui est l'adresse du XBee 1, à gauche) est insérée dans les adresse de destination du XBee 2 (les paramètres DH et DL contiennent alors l'adresse du premier module) et vise-versa. Le problème de cette configuration c'est que c'est absolument pas modulable et que surtout c'est uniquement point à point (risque de conflits si on fait du multipoint avec cette méthode).

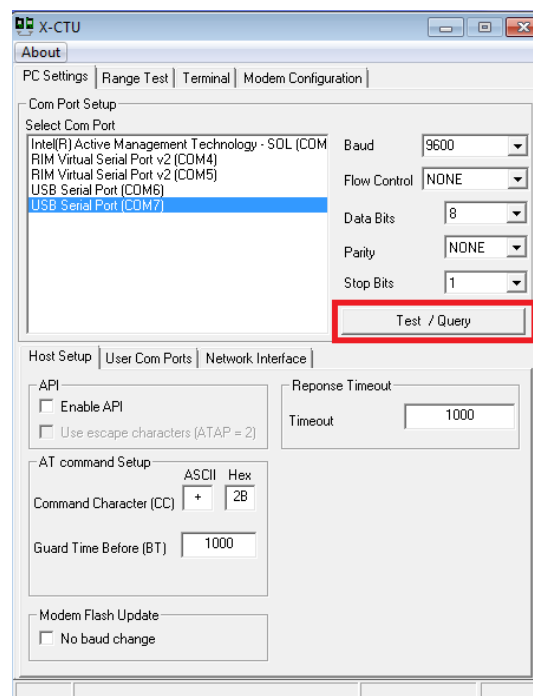
Alors dans le cadre de notre projet on a opté pour un mode coordinateur-router. Dans le schéma ci-dessous ce type de liaison est expliqué :



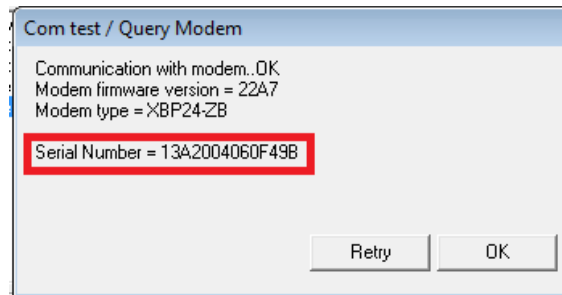


Dans notre cas nous n'utilisons pas de XBee de type "End Device", nous avons simplement un XBee de type coordinateur qui est placé sur la station qui reçoit les commandes et puis qui les envoient sur le bus CAN. La station de gestion (écran tactile) est équipée d'un module en mode routeur. Pour configurer ces modules on peut directement utiliser un logiciel fournis par le fabricant (Digi International) et qui se nomme XCTU.

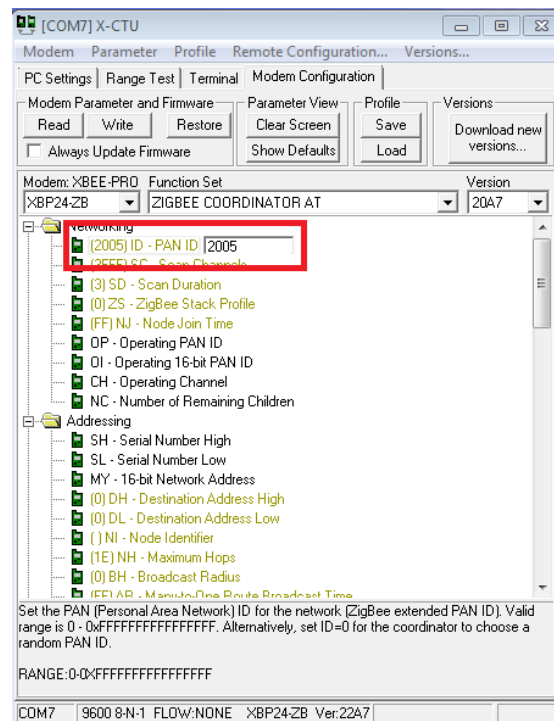
Pour configurer les modules il suffit de les connecter à l'ordinateur via un petite station qui se branche en USB et qui sera reconnue comme un port COM. Lorsque vous lancez le logiciel, il suffit de cliquer sur le bouton suivant pour se connecter au module et récupérer ses données :



Une fois connectez une fenêtre s'affiche, il faut noter le numéro de série du module XBee car ça sera celui qui est configurer en coordinateur :

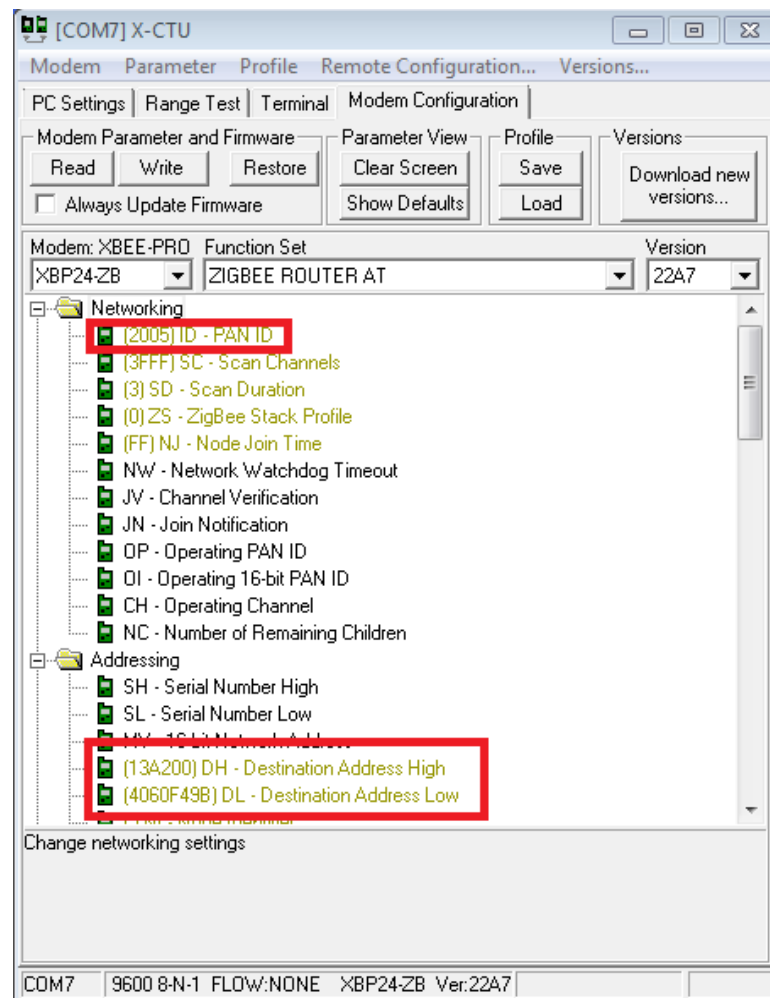


Une fois connecter il suffit de se déplacer dans l'onglet "Modem Configuration" afin de configurer le module. Pour le mode coordinateur il suffit de selectionner "Function Set" en tant que "ZIGBEE COORDINATOR AT". La seule chose a changer ensuite c'est le PAN ID, cette valeur correspond à quel "sous-réseau" appartient le module, la valeur choisie est totalement mais il faut la retenir car elle sera nécessaire pour la configuration router. Voici ce que vous devez avoir:



La dernière chose à configurer l'autre module en mode routeur. Pour se faire c'est la même manipulation que précédemment sauf qu'au lieu de choisir "ZIGBEE COORDINATOR AT" cette fois-ci il faut prendre "ZIGBEE ROUTER AT". Une fois cette fonction choisie il faut mettre le même PAN ID que le précédemment et dans DH et DL il faut mettre le numéro de série du module coordinateur.

Voici à quoi doit ressembler votre configuration pour le mode routeur :

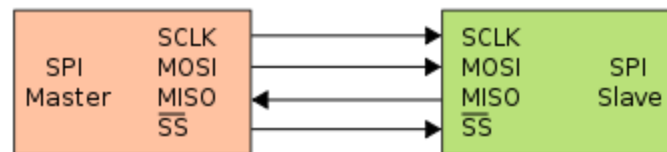


Après ces configuration vous avez deux modules qui fonctionnent parfaitement entre eux. Vous pouvez les données via un terminal (TeraTerm par exemple) afin de tester si la configuration est correcte.

## COMMUNICATION SPI

La communication SPI est un bus de données série synchrone qui fonctionne en mode full-duplex (deux fils, un pour l'envoi et l'autre la réception). Le SPI fonctionne avec les propriétés "master" et "slave". Dans ce type de conversation c'est toujours le master qui doit initier la conversation. Les périphériques SPI de type "slave" sont tous équipés d'un "chip select", cette ligne permet de dire que l'on désire parler avec ce périphérique en particulier et donc que celui-ci doit écouter ce qu'il lit sur son MOSI (master out slave in) et qu'il peut également envoyer ses données via le MISO (master in slave out).

Pour que les données soient transmises par l'esclave, le maître doit toujours envoyer lui-même des données afin de générer des coups d'horloges.



Dans le cadre de notre projet un fichier nommé SPI.c contient l'intégralité des fonctions nécessaires à l'utilisateur du SPI sur un microcontrôleur de type LPC1769.

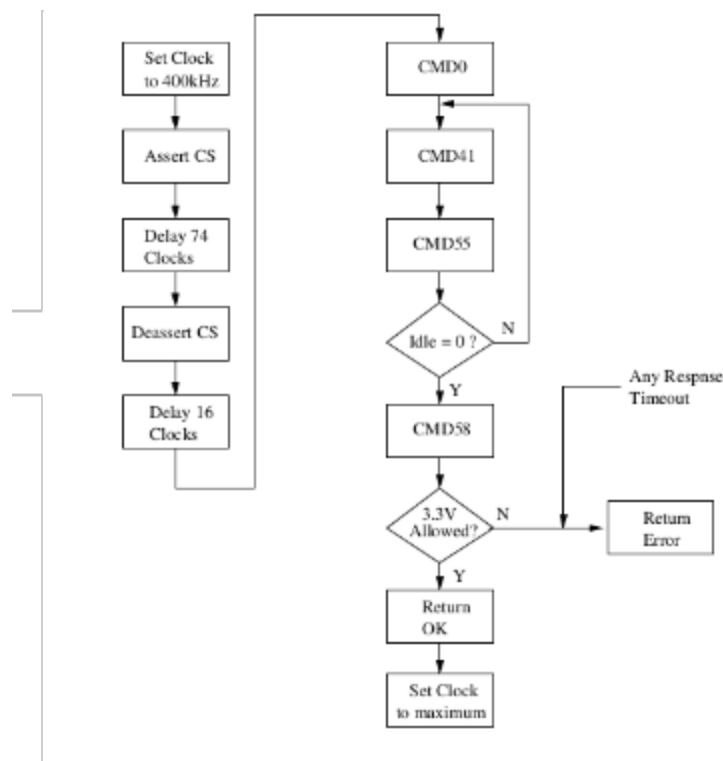
Une fonction d'écriture est disponible en mode écriture seule. Une autre fonctionne en mode écriture et lecture, c'est à dire que en retour on récupère des données qui sont reçues de la part de l'esclave.

## CARTE SD

La carte SD est initialisée en mode SPI, c'est à dire qu'il n'y a pas de système de fichier présent sur la carte. En SPI la mémoire de notre carte est divisée en 3862528 blocs de 512 octets. Nous stockons les images des trains sur la carte que l'on va lire et afficher sur l'écran. Nous utilisons le protocole SPI pour lire les images.

### **Initialisation**

L'initialisation de la carte SD se fait par l'envoi de commandes spécifiques via SPI. Cette initialisation nécessite une fréquence SPI spécifique ainsi que quelques coups d'horloge entre les commandes. Voici le schéma d'initialisation de la carte SD en mode SPI.



## Le stockage des images

L'interface ainsi que les images des trains sont des fichiers bitmap de tailles différentes. Elles sont stockées sur la carte SD. La particularité du format bmp est qu'il stocke les images en commençant par le dernier pixel. Nous avons donc retourné les images verticalement avec GIMP, retiré l'entête bmp puis écrit sur la carte SD avec Hdx.

L'ensemble des images sur la carte SD remplissent 1249 blocks de 512 octets sur la carte. Les autres blocks ne sont pas utilisés et sont initialisés à 0 par défaut. Voici l'emplacement des images sur la carte ainsi que leurs dimensions.

Images	Dimension	block de départ	block d'arrivée
Interface	320x240	0	449
défault	180x150	451	609

Train n°5	180x150	611	769
Train n°66	180x150	771	929
Train n°14	180x150	931	1089
Train n°44	180x150	1091	1249

## Les fonctions créées

- `init_SD()`: Initialisation de la carte SD en SPI.
- `Read_SD_one_block()`: Lecture d'un bloc passé en paramètre.
- `Read_SD_multi_block()`: Lecture de l'intervalle de block passé en paramètre.
- `Clear_pin()`: Selectionne le chip select.
- `Set_pin()`: Selectionne le chip select.

Nous n'avons pas crée de fonctions d'écriture car nous ne faisons que lire des images sur la carte SD.

## Matériel et logiciels utilisés pour la gestion de la carte SD

- *Gimp* pour les images ( dimension, création interface, inversion vertical).
- *Hex Editor* pour la lecture de fichier bitmap et l'effacement de l'entete bmp.
- *Hxd* pour l'écriture par bloc sur la carte SD.
- *USBee* pour le débugeage et la vérification de l'initialisation SPI et des trames en général.

## DALLE TACTILE

La dalle tactile utilise un contrôleur de type TSC2046 qui est commandé via SPI. Lorsque qu'une pression est effectuée sur la dalle tactile nous avons décidé de faire en sorte de mettre une interruption qui est appelée.

Sur la dalle on a la possibilité de lire 3 type de données, le X, le Y et le Z. Dans le cadre de notre projet nous n'avons qu'utiliser le X et le Y. Pour lire ces valeurs il suffit d'envoyer une trame sur le SPI formée comme ceci :

BIT 7 (MSB)	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 (LSB)
S	A2	A1	A0	MODE	SER/DFR	PD1	PD0

Voici la signification des options :

BIT	NAME	DESCRIPTION
7	S	Start bit. Control byte starts with first high bit on DIN. A new control byte can start every 15th clock cycle in 12-bit conversion mode or every 11th clock cycle in 8-bit conversion mode (see Figure 13).
6-4	A2-A0	Channel Select bits. Along with the SER/DFR bit, these bits control the setting of the multiplexer input, touch driver switches, and reference inputs (see Table 1 and Figure 13).
3	MODE	12-Bit/8-Bit Conversion Select bit. This bit controls the number of bits for the next conversion: 12-bits (low) or 8-bits (high).
2	SER/DFR	Single-Ended/Differential Reference Select bit. Along with bits A2-A0, this bit controls the setting of the multiplexer input, touch driver switches, and reference inputs (see Table 1 and Table 2).
1-0	PD1-PD0	Power-Down Mode Select bits. Refer to Table 5 for details.

Dans notre cas nous avons choisi la conversion sur 12 bits et sans mode différentiel. Pour choisir les A2 à A0 il faut s'aider des table ci-dessous :

A2	A1	A0	V <sub>BAT</sub>	AUX <sub>IN</sub>	TEMP	Y-	X+	Y+	Y-POSITION	X-POSITION	Z <sub>1</sub> -POSITION	Z <sub>2</sub> -POSITION	X-DRIVERS	Y-DRIVERS
0	0	0			+IN (TEMP0)								Off	Off
0	0	1					+IN		Measure				Off	On
0	1	0	+IN										Off	Off
0	1	1					+IN				Measure		X-, On	Y+, On
1	0	0				+IN						Measure	X-, On	Y+, On
1	0	1						+IN		Measure			On	Off
1	1	0		+IN									Off	Off
1	1	1			+IN (TEMP1)								Off	Off

**Table 1. Input Configuration (DIN), Single-Ended Reference Mode (SER/ $\overline{\text{DFR}}$  high)**

A2	A1	A0	+REF	-REF	Y-	X+	Y+	Y-POSITION	X-POSITION	Z <sub>1</sub> -POSITION	Z <sub>2</sub> -POSITION	DRIVERS
0	0	1	Y+	Y-		+IN		Measure				Y+, Y-
0	1	1	Y+	X-		+IN				Measure		Y+, X-
1	0	0	Y+	X-	+IN						Measure	Y+, X-
1	0	1	X+	X-			+IN		Measure			X+, X-

**Table 2. Input Configuration (DIN), Differential Reference Mode (SER/ $\overline{\text{DFR}}$  low)**

Dans notre cas on utilisera la première table car le mode différentiel n'est pas utilisé. Ce qui nous donne une valeur de trame de 0xD3 pour la lecture du X et de 0x90 pour la lecture du Y.

Suite à des problèmes de rebonds, nous avons mis en place un système d'anti-rebonds via un timer. Ce timer se déclenche à l'appui sur la dalle, et puis le timer se déclenchera, via une nouvelle interruption, après un temps imparti (dans notre cas 1ms) afin. Et afin d'être sûr que une pression est toujours effectué on test le GPIO d'interruption lié à la dalle. La lecture est effectuée dans le main (via la scrutation d'un "flag" global) afin de passer le moins de temps possible dans les interruptions.



## INTERFACE GRAPHIQUE

Pour l'interface utilisateur sur l'écran RGB, nous avons décidé de créer une image à l'aide du logiciel GIMP. Ensuite à l'aide d'une carte SD formatée en SPI (sans système de fichier), nous stockons l'image sans l'en-tête de fichier bmp.



- Le carré vert permet de mettre du courant sur les rails, ce même carré change de couleur en devenant rouge, ce qui permet de couper le courant.
- le carré contenant "Train" est utilisé pour afficher la locomotive qui est manipulée.
- Le carré contenant "Sens" permet d'afficher une flèche qui indique la direction que prend la locomotive.
- Le carré contenant "Lumière" permet d'indiquer si les lumières de la locomotive sont allumées ou non.
- Le carré contenant "Vitesse" affiche la vitesse de la locomotive de 0 à 1000.
- La zone où se situe le "?", permet d'afficher une photo de la locomotive. Si la photo n'est pas connue le "?" sera affiché.

En sélectionnant "Train" ou "Vitesse" les valeurs sont modifiables à l'aide du bouton rotatif.

## CONCLUSION GENERALE

Après un semestre de travail, nous sommes fiers de présenter une station de gestion de train fonctionnelle. Le cahier des charges présenté en début de semestre a été respecté. De bonnes relations, de l'entraide et de la coordination nous ont permis d'atteindre nos objectifs. Ce TP nous a permis de faire face aux problèmes liés au travail en groupe sur le long terme.

### **Problèmes rencontrés**

La séparation des tâches a permis un approfondissement personnel des connaissances au sein de la tâche même. Malheureusement, il n'y a eu qu'un survol du travail des autres, et donc une compréhension basique et peu approfondie de certains éléments du TP.

### **Améliorations possibles**

Pour les améliorations possibles, on pourrait envisager de programmer des trajets pour chaque locomotive. La gestion des aiguilleurs pourrait être implémentée rapidement car c'est pas compliqué. Une autre amélioration qui serait très intéressante, la gestion des collisions et l'utilisation des capteurs pour rendre la navigation plus sûre.