Dossier de projet tutoré

Train miniature HO et norme DCC
(Digital Command Control)

IUT Belfort-Montbéliard
19 avenue Maréchal Juin
90000 Belfort

Réalisé par :
MONSEL Rémi et MOLINA Manlyo

Professeur tuteur : M. Mickaël HILAIRET

REMERCIEMENTS

Nous avons réalisé ce projet tutoré au sein de l'IUT Belfort-Montbéliard et nous tenons à remercier toutes les personnes qui ont intervenues dans le bon déroulement de ce projet.

Tout d'abord, nous souhaitons remercier l'IUT Belfort-Montbéliard pour nous avoir fournis une salle nous permettant de travailler pendant les heures dédiées au projet tutoré et également en dehors de ces horaires et également de nous avoir fournis tout le matériel dont nous avions besoin (ordinateur, appareil de mesures …).

Nous voulons également remercié, Vincent Hubert de nous avoir fournis des conseils avisés lors de nos choix de composants et d'avoir réalisé le circuit imprimé dont nous avions besoin.

D'autre part, nous souhaitons remercié Mr Gavignet de nous avoir fournis des cours VHDL en amont de notre projet tutoré car cela nous a servis tout au long de notre projet tutoré.

Nous désirons également attribuer un remerciement spécial à notre professeur tuteur : Mr Hilairet qui a su nous encadré et nous aidé tout au long de l'année.

SOMMAIRE DU DOSSIER

INTRODUCTION

La licence professionnelle VEGA (Véhicules : électronique et Gestion des Automatismes) nous a permis de réaliser un projet tutoré. Nous avons choisis et réalisé durant cette année le projet « train miniature HO et norme DCC* (Digital Command Control) ». Dans un premier temps, ce projet a pour but d'analyser, de comprendre le fonctionnement d'un train numérique selon la norme DCC* et de connaître le fonctionnement des produits vendus dans le commerce. Ensuite, nos connaissances nous ont permis de réaliser notre propre codeur grâce au logiciel Quartus II d'ALTERA pour la programmation de circuit logique FPGA.

Cette norme permet de contrôler plusieurs trains sur un même rail dans le cas où l'on utilise un train numérique. Dans le cas où l'on souhaite commander un train analogique, on peut disposer d'un mode numérique-analogique permettant a la fois de commander les trains numériques et analogiques.

Nous avons également réalisé un « Booster» constitué d'un hacheur 4 quadrants qui nous a permis de commander des locomotives analogiques et des locomotives numériques. Par ailleurs, pour commander des trains numériques, nous avons deux choix de contrôle grâce à la carte FPGA. Le premier choix que nous avons est d'utiliser les boutons poussoirs et les interrupteurs ou bien d'utiliser une télécommande infrarouge.

* : Voir lexique

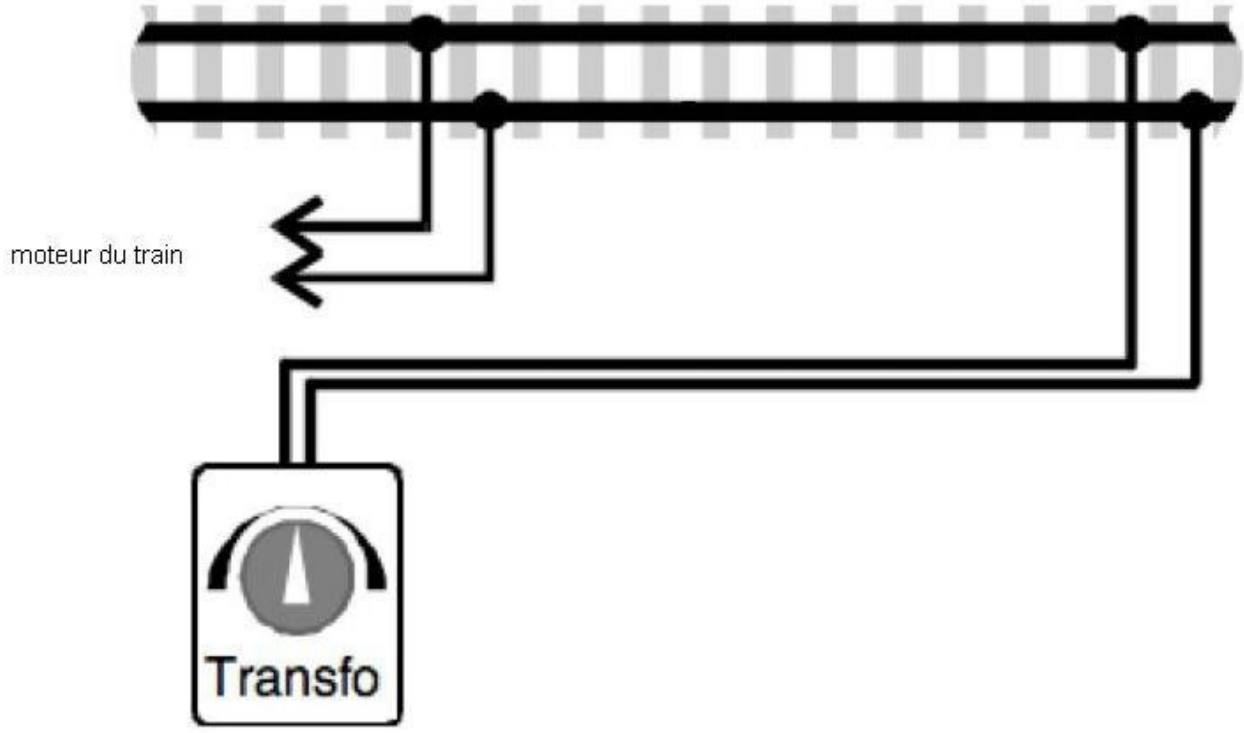
I - PRESENTATION GENERALE

I - 1 Expression du besoin

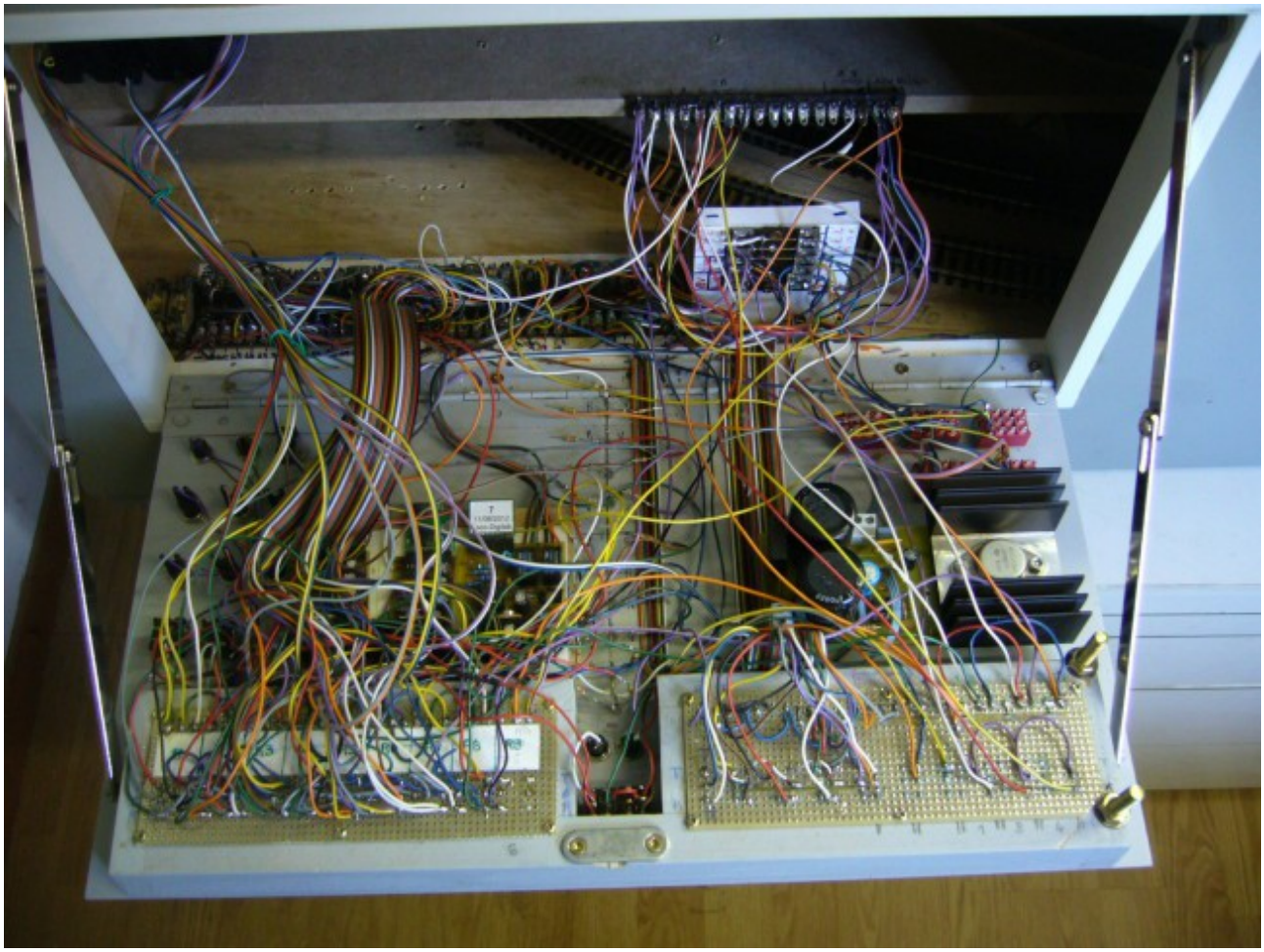
Depuis son invention, le modélisme ferroviaire suscite un engouement important. Depuis les années 1980, il est possible de commander son train au moyen d'un système numérique. Les voies ferroviaire ont de plus en plus d'équipements telle que la commande de plusieurs trains avec différentes fonctions sur une même voie : des aiguillages, des fumigènes, des klaxons, vitesse variable, éclairage, signalisation, arrêts et démarrages progressifs automatisés... qui n'ont pas uniquement besoin d'une alimentation mais nécessite également un moyen de transmission de toutes ces fonctions sur une seule voie.

I - 2 Locomotive analogique ou numérique ?

Lorsque l'on veut faire du modélisme ferroviaire, deux choix s'ouvrent à nous. On peut choisir un système analogique ou un système numérique. Le système analogique peut être mis en œuvre rapidement avec un hacheur 2 quadrants ou avec un autotransformateur et un redresseur à diode pour permettre au train d'avancer de reculer et de s'arrêter. Le schéma suivant permet de comprendre le fonctionnement du système analogique.

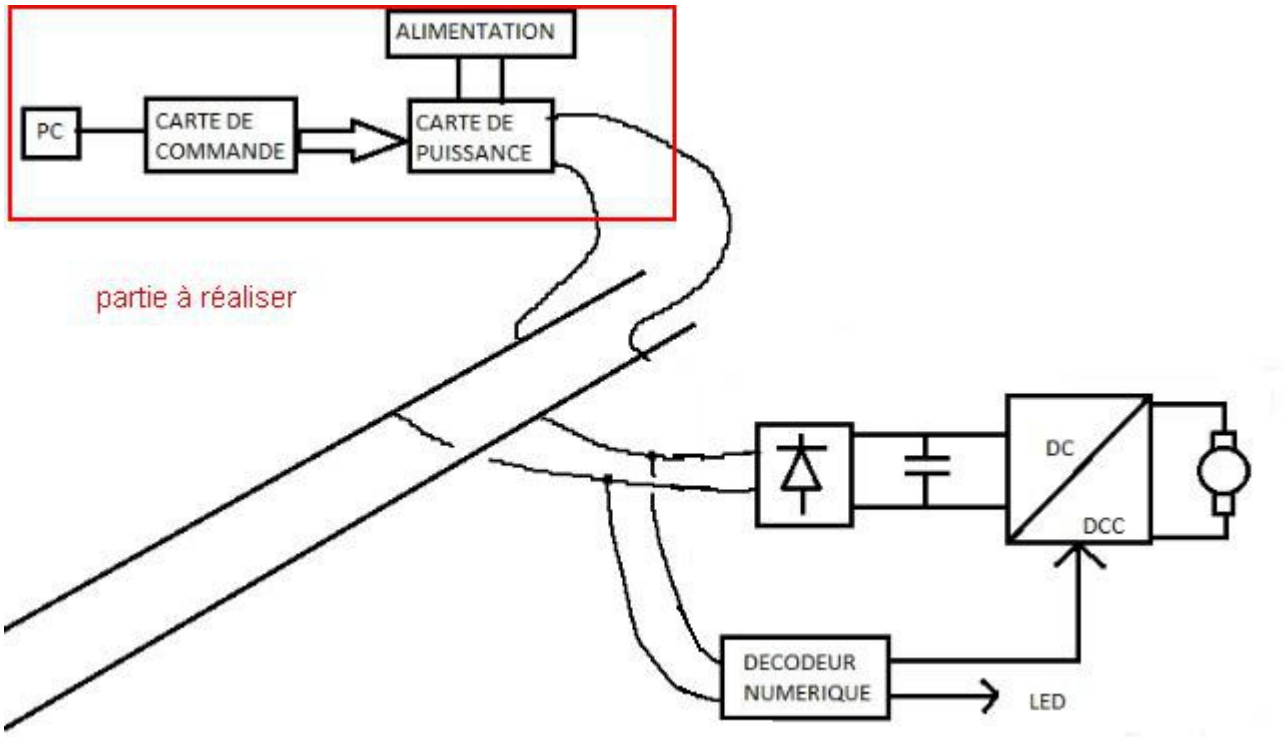


Dans le cas où l'on voudra plus de possibilités par exemple commander uniquement un train parmi ceux présent sur le circuit, il faudra utiliser un système de canton et un câblage complexe ressemblant à celui de la figure suivante.



Par contre, le système numérique est simple à mettre en œuvre avec uniquement deux fils. Il permet de commander des trains avec une tension alternative, ce qui signifie que l'on pourra commander le train miniature à différentes vitesses. Cependant, on se retrouve assez rapidement limité en utilisation lorsque l'on utilise un système analogique. En effet, il est impossible de faire varier la vitesse des différentes locomotives indépendamment. Par contre, ce n'est pas le cas, d'un système numérique car avec un système numérique, on pourra avoir de nombreuses options envisageable, toutes celle du système analogique et les possibles suivantes : commander plusieurs locomotives sur une voie en les alimentant tous mais en les commandant différemment, on peut faire circuler plusieurs trains sur une même section de voie à différentes allures et dans des directions opposées, on pourra également commander les trains en vitesse avec une grande précision et mettre en œuvre d'autres fonctions sur la locomotive comme des phares, un klaxon…

La figure suivante permet d'illustrer le système numérique.



Cependant, le temps de développement du programme et la mise en œuvre d'un système numérique est plus longue mais plus intéressante pour des passionnés d'électronique et d'informatique.

La différences majeures entre la commande d'une locomotive analogique et numérique résident dans le faite que l'alimentation n'est pas faite de la même manière. En effet, une locomotive analogique est directement commandé avec une PWM* alors qu'un train miniature numérique est commandé avec un signal codé numériquement portant à la fois le signal de puissance et des informations de commande d'une manière similaire au courant porteur.

* : Voir lexique

II – MATERIEL ET METHODES

II – 1 Cahier des charges initial

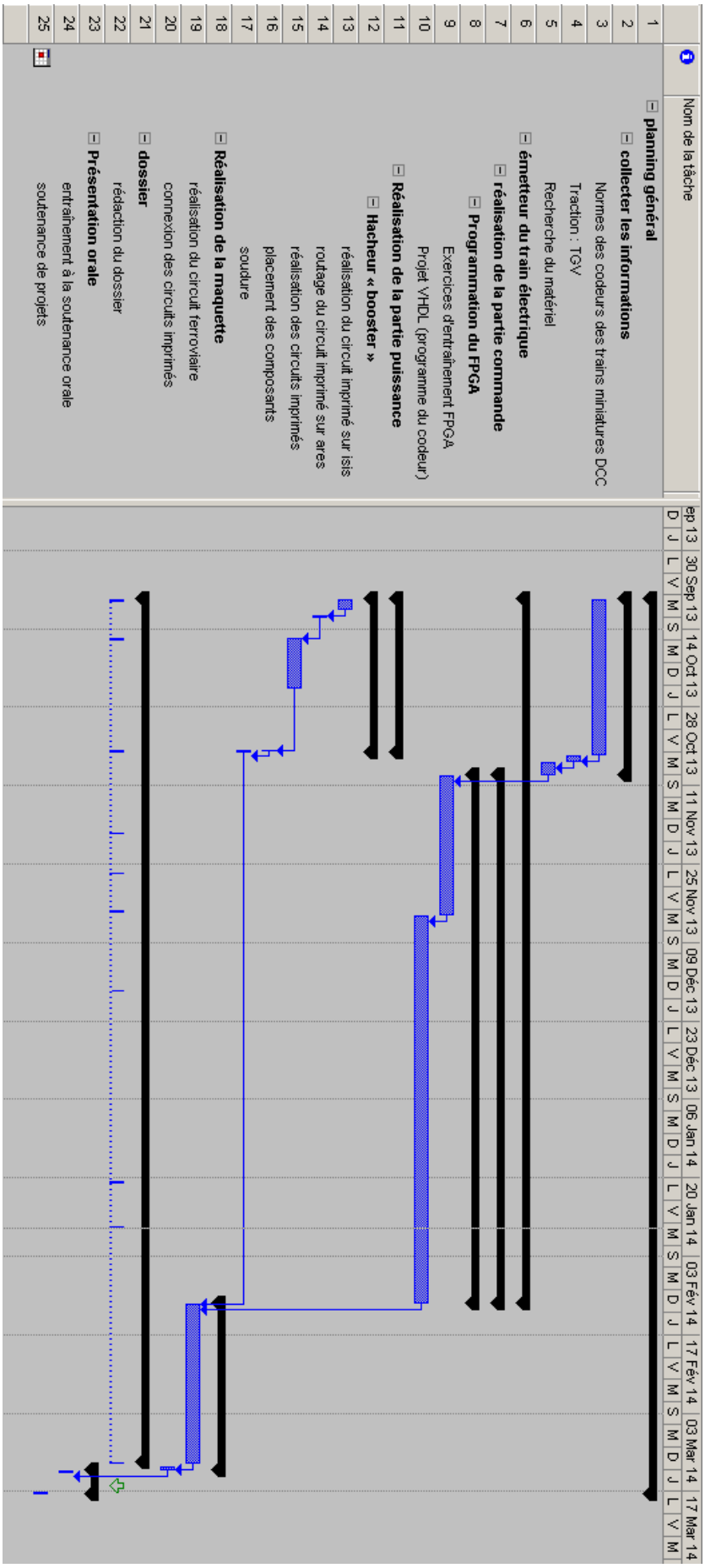
Le cahier des charges initialement prévu pour notre projet tutoré est le suivant : réaliser un codeur pour un train miniature appartenant à la norme DCC* et un hacheur 4 quadrants permettant de faire le lien entre la carte FPGA et les rails.

II – 2 Cahier des charges révisé

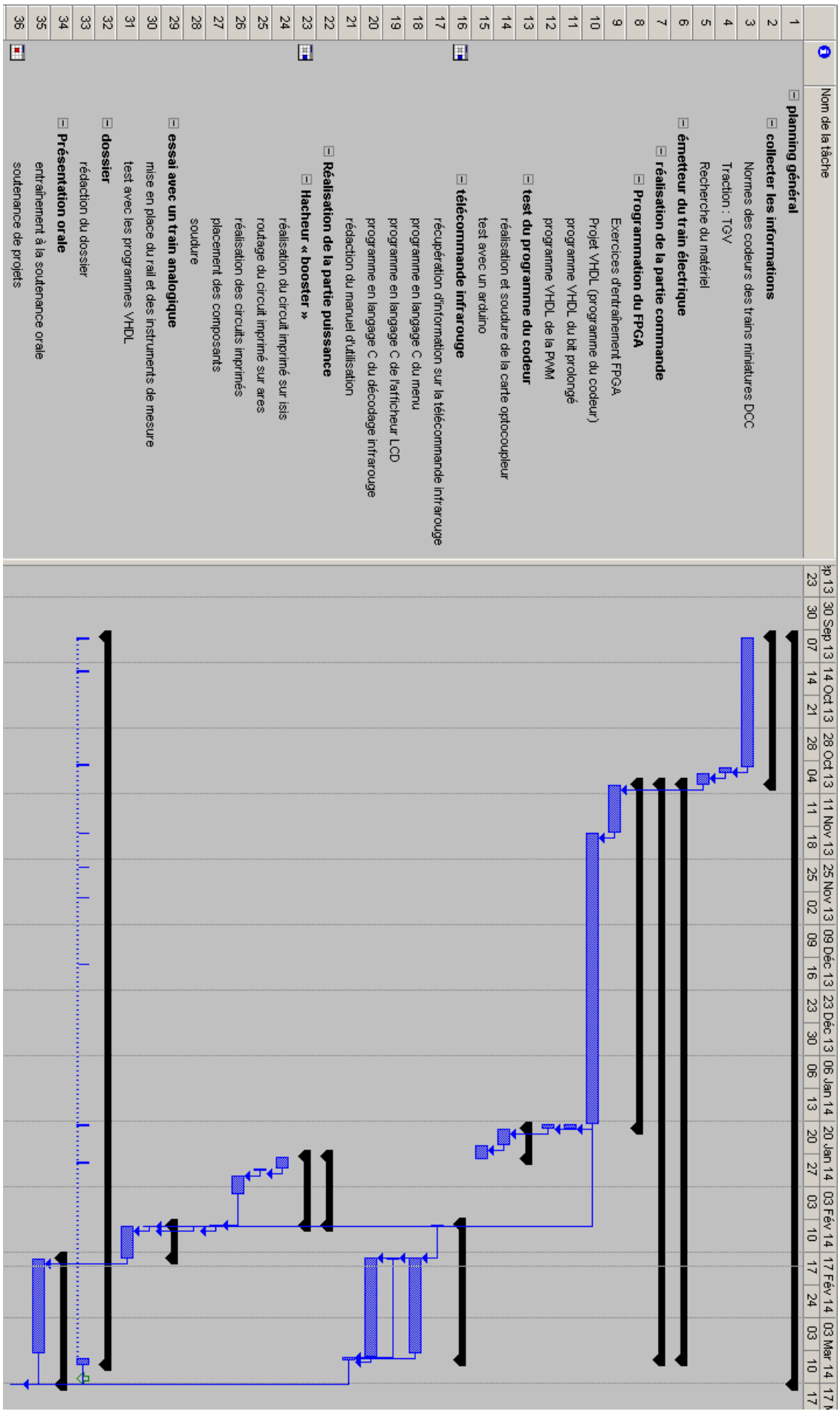
Finalement, le cahier des charges a été modifié en cours d'année. En effet, nous avons réalisé le codeur du train miniature appartenant à la norme DCC* et le hacheur 4 quadrants mais nous avons également réalisé la programmation de la carte FPGA de manière à recevoir les commandes du train miniature a partir d'une télécommande infrarouge.

* : Voir lexique

II – 3 Planification du projet : Diagramme de Gant prévisionnel



II – 4 Planification du projet : Diagramme de Gant rétrospectif



II – 5 Méthodes suivies

Pour réaliser notre projet dans les meilleures conditions, on a suivi une méthode simple. On a commencé par réaliser le diagramme de Gantt, de manière à avoir un planning. Cela nous a permis de savoir si nous étions en avance ou en retard par rapport au prévision et donc d'adapter notre méthode de travail.

On a commencé par collecter des informations sur la norme DCC* et notamment les normes NEM* 670 et NEM* 671. Cela nous a permis de comprendre le fonctionnement de la norme DCC* et de savoir ce que nous devions mettre en œuvre pour réaliser le codeur du train miniature.

Nous avons fais des exercices d'entraînement VHDL de manière à maîtriser la programmation en VHDL et nous avons fais des exercices sur les structures conditionnelles et également des exercices sur les machines d'état. Ces exercices étant maîtrisés, nous sommes passés à la partie la plus importante de notre projet : la réalisation du programme du codeur.

En parallèle, nous avons réalisé le schéma et le typon de la carte « Booster » contenant un hacheur 4 quadrants et permettant la communication entre les rails et la carte VHDL puis nous avons soudé les composants et les connecteurs sur la carte.

Après avoir réalisé la carte « Booster », nous avons réalisé un test avec un circuit arduino et un programme trouvé sur internet, ce programme permet d'identifier s'il s'agit d'une trame DCC* ou non.

D'autre part, nous avons rajouté quelques sous-programmes permettant d'utiliser notre programme pour commander des trains miniatures analogiques. Cela nous a permis de savoir si nous pouvions commander des trains analogiques en réalisant un test avec un rail et un train analogiques.

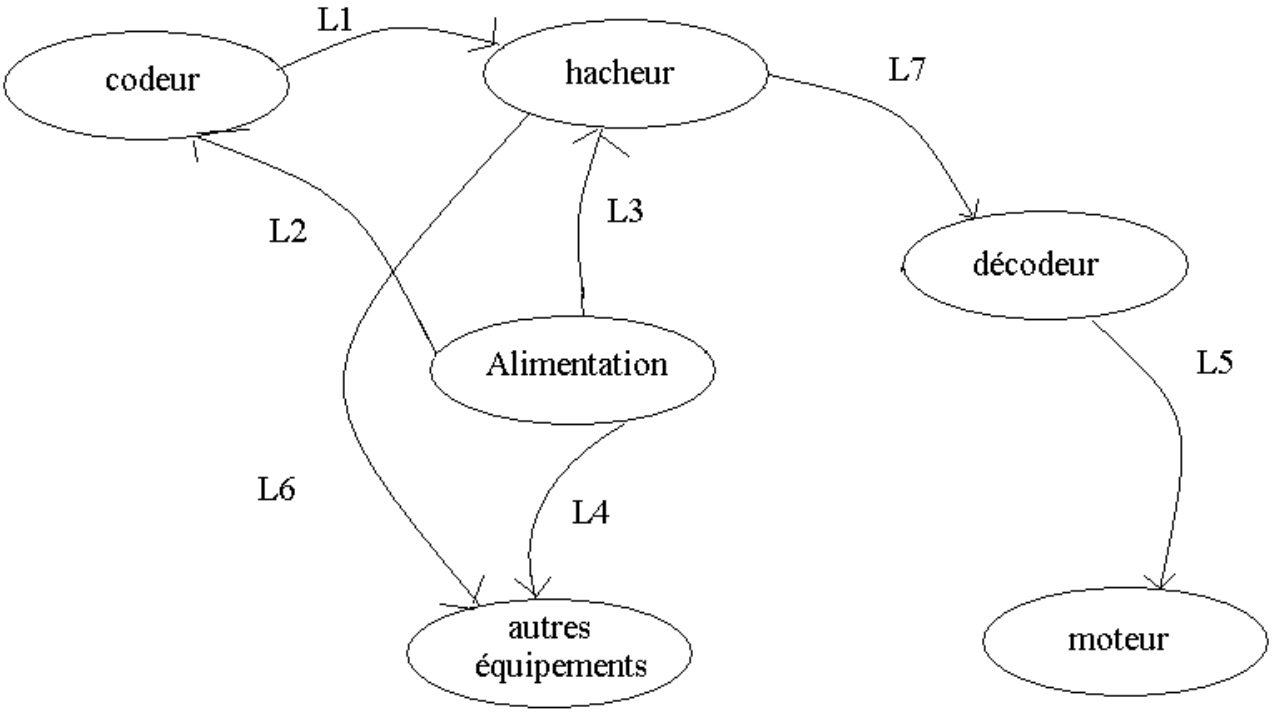
Par la suite, nous avons mis en œuvre la programmation de la carte FPGA permettant d'utiliser la télécommande pour contrôler la locomotive.

II - 6 Matériels utilisés

Pour concrétiser notre projet, nous avons eu besoin d'un ordinateur, d'une carte FPGA DE2 cyclone II fabriqué par Altera, d'un hacheur 4 quadrants basé sur le composant L6203. Nous avons également eu besoin d'une alimentation stabilisé permettant d'alimenter avec une tension de 18 V, d'un oscilloscope et de la télécommande.

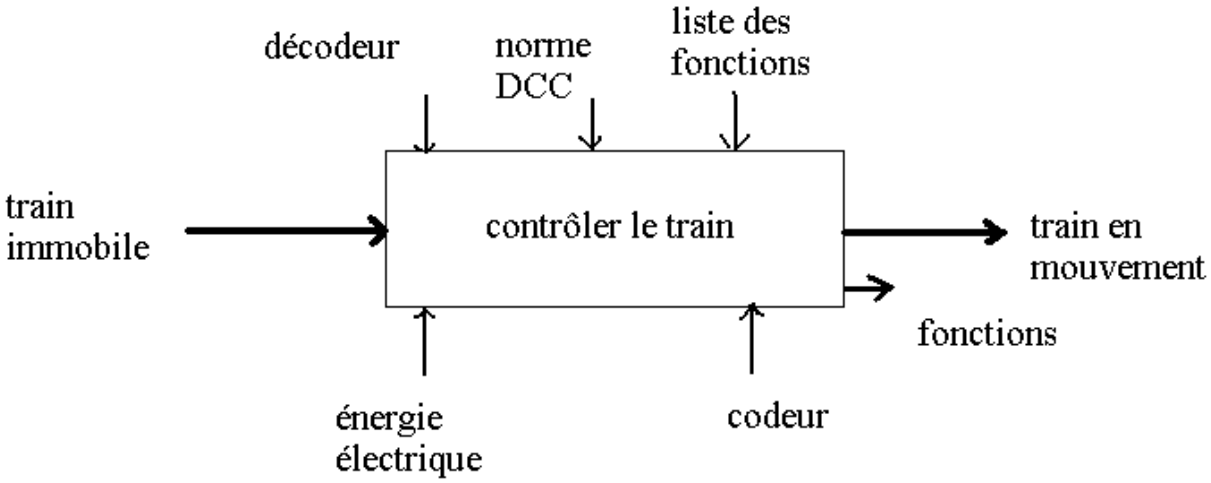
II - 7 Mise en situation

Description des liaisons :



L1 : décodage DCC*
L2 : alimentation secteur de la carte FPGA
L3 : alimentation du décodeur
L4 : alimentation des autres équipements
L5 : alimentation du hacheur en 18V
L6 : Signal PWM*
L7 : décodage DCC* pour les autres équipements (fumigène, éclairage...)
L8 : Amplification du signal

II - 8 Diagramme SADT



Ce diagramme SADT caractérise les liens qui existent entre les différents éléments de notre système. On a l'origine le train immobile et grâce à la norme DCC*, on a pu réalisé un codeur qui permettra par la suite, d'envoyer des informations au décodeur (locomotive sélectionnée, vitesse, sens de déplacement...) et finalement de mettre en mouvement le train et de mettre en action les diverses fonctions du train (klaxon,éclairage …).

* : Voir lexique

III - ASSOCIATIONS NMRA ET NORME NEM DEFINITION DE LA NORME DCC

L'association NMRA* est une association américaine qui s'investit énormément dans le développement du modélisme ferroviaire. Cette association fut fondée en 1935 aux États-Unis, à l'heure actuelle cette association est présente dans de nombreux pays telle que : le Royaume-Uni, le Canada, l'Australie mais également dans d'autres pays comme la France. Le but de cette association est de promouvoir les différentes normes du modélisme ferroviaires, et fournir également des recommandations liés au modélisme ferroviaires. En Europe, c'est la norme NEM* qui prime.

La norme DCC* est définie par les différentes normes NEM*, nous en retiendrons deux : la norme NEM* 670 [1] et NEM* 671 [2].

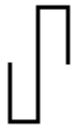
III -1 NEM* 670 [1]

Le contenu de la NEM* 670 [1] est conforme au Standard NMRA* S 9.1.
Le but de la norme est d'encoder des bits selon le standard DCC*. Pour ce faire, l'encodage du bit se décompose en plusieurs éléments caractéristiques :

- La transmission des données est effectuée par l'émission d'une série de bits. Cette transmission se fait au travers des rails et est appelé le signal de voie, ces bits prennent deux valeurs 0 et 1.
- Le signal de voie se compose d'une succession de transitions entre deux niveaux de tension de polarité opposée, appelées passages à zéro et s'il y a deux passages à zéro successifs de même sens indiquent que l'on passe au bit suivant.

III - 1.1 Description du bit « 1 » :

- Un bit « 1 » est composé de deux tensions de signes opposées ayant une durée identique de 58µs chacune, ainsi la durée d'un bit « 1 » est de 116 µs.
- La tolérance de durée admise pour un bit « 1 » est de ± 3µs pour le signal de voie à l'émission alors que pour le signal à la réception à une tolérance est de ± 6µs.



bit "1"

* : Voir lexique

III – 1.2 bit « 0 » :

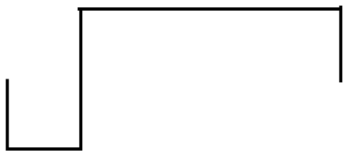
III – 1.2.1 Description du bit « 0 » :

Le bit « 0 » est construit de la même manière, à l'exception que la durée d'alternance doit être supérieure ou égale à 100µs.

bit "0"

A la réception, une alternance doit avoir une durée comprise entre 97 et 9900 µs. De plus, la durée total du bit « 0 » ne doit pas dépasser 12ms. Cependant pour le décodeur, la durée de chaque alternance doit est comprise entre 94 et 10000 µs.

III – 1.2.2 Description du bit « 0 » prolongé :



bit '0' prolongé

Le bit « 0 » prolongé a une durée à l'état bas identique au bit « 0 », c'est-à-dire supérieure ou égale à 100µs. Par contre, sa duré à l'état haut est plus grande. En faite, la durée à l'état haut est supérieure à 100 µs et est inférieure à 9,9 ms. Avec ce bit, il est possible de commander des trains miniatures analogiques car la valeur moyenne de la tension devient non nulle, ce qui n'est pas le cas avec le bit '0' « classique » de la norme DCC* car les alternances ont la même durée.

III - 1.3 Transport d'énergie et tension :

Toutes les stations de commande numérique et les décodeurs sont alimentés à l'aide d'un redresseur double alternance.
La tension supportées par les décodeurs est une tension de 27 V continue sur la voie.

* : Voir lexique

III - 2 NEM* 671 [2]

Cette norme a pour but de décrire le protocole permettant d'émettre les données par une centrale DCC*. Les données sont une combinaison de bits décrit dans la norme NEM* 670 [1] comme étant un signal de voie. On appelle paquet de données un nombre minimum de bits, un groupe de 8 bits appelé également octet.

III - 2.1 Description générale des éléments constitutifs d'un paquet de données :

Le paquet de données est constitué de divers éléments :

- **la synchronisation**
- **le bit de start**
- **l'octet d'adresse**
- **le bit de start de l'octet de donnée**
- **l'octet de donnée**
- l**e bit de stop**

La **synchronisation** est une suite de bits à « 1 ». Le paquet de données reçu par le décodeur doit contenir entre 10 et 12 bits à « 1 ». La station de commande doit envoyer 14 bits au minimum.

Le **bit de start** est un bit à « 0 » qui suit les bits de synchronisation ce qui indique au décodeur que les bits suivant représentent un octet d'adresse.

L'**octet d'adresse (0AAAAAAA)** est le premier octet reçu par le décodeur après le bit de start, il contient l'identité du décodeur que l'on souhaite commander. Certaines adresses ont des fonctions spéciales comme l'adresse 0, 254 et 255. Ces adresses doivent être utilisées uniquement dans le cas où l'on veut utilisé les fonctions spéciales. Le bit 7 est un bit à « 0 », dans la condition où il s'agit d'un octet d'adresse. Les autres bits de l'octet de donnée désigne le décodeur que l'on souhaite commander.

Le **bit de start de l'octet de donnée** est un bit à « 0 ». Il est envoyé au décodeur avant d'envoyer l'octet de donnée.

L'**octet de commande (01DCSSSS)** est utilisé comme adresse d'instruction. Son but est soit d'envoyer des données telle que le sens de marche ainsi que la vitesse de la locomotive. Le bit 7 est à l'état bas et le bit 6 est à l'état haut dans le cas où l'on souhaite que l'octet de donnée soit un octet de commande. Le bit 5 défini le sens de marche. Un état logique « 1 » signifie que la locomotive avance alors qu'un état logique bas signifie que la locomotive est en marche arrière. Le bit 4 a une fonction spéciale de commande (lumière, klaxon, fumigène...) et est également le bit de poids faible pour définir la commande de vitesse. Le tableau ci-dessous permet de connaître l'état de fonctionnement de la locomotive en terme de vitesse et d'arrêt, le bit C étant le bit 4 et Estop caractérisant un arrêt d'urgence.

* : Voir lexique

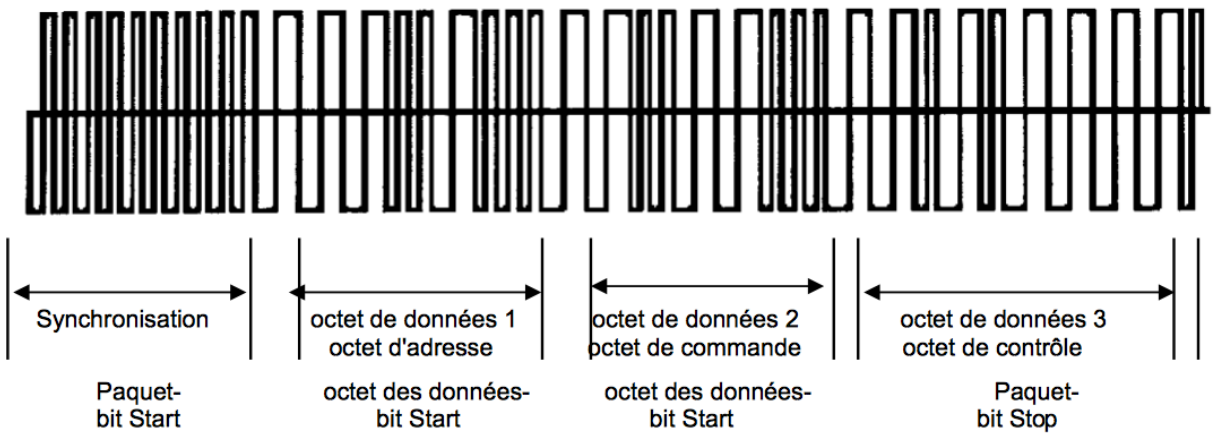
$S_3S_2S_1S_0C$	graduation	$S_3S_2S_1S_0C$	graduation	$S_3S_2S_1S_0C$	graduation	$S_3S_2S_1S_0C$	graduation
0 0 0 0 0	Stop	0 1 0 0 0	5	1 0 0 0 0	13	1 1 0 0 0	21
0 0 0 0 1	Stop+	0 1 0 0 1	6	1 0 0 0 1	14	1 1 0 0 1	22
0 0 0 1 0	EStop*	0 1 0 1 0	7	1 0 0 1 0	15	1 1 0 1 0	23
0 0 0 1 1	EStop*+	0 1 0 1 1	8	1 0 0 1 1	16	1 1 0 1 1	24
0 0 1 0 0	1	0 1 1 0 0	9	1 0 1 0 0	17	1 1 1 0 0	25
0 0 1 0 1	2	0 1 1 0 1	10	1 0 1 0 1	18	1 1 1 0 1	26
0 0 1 1 0	3	0 1 1 1 0	11	1 0 1 1 0	19	1 1 1 1 0	27
0 0 1 1 1	4	0 1 1 1 1	12	1 0 1 1 1	20	1 1 1 1 1	28

Le tableau ci-dessus montre tous les possibilités de vitesses envisageables. En effet, Stop permet d'arrêter le train, EStop permet de réaliser un arrêt d'urgence, on dispose également de 28 vitesses.

Dans le cas où l'on veut mettre en place un **octet de contrôle**, il faut mettre un bit de start c'est à dire, un bit à l'état bas puis envoyer l'octet de contrôle de manière à détecter les erreurs lors de la transmission.Pour détecter les erreurs, on fait une addition logique bit à bit de l'octet d'adresse avec l'octet de commande. Cette addition logique est réalisé a l'aide d'une fonction ou exclusif. Par la suite, le décodeur compare l'octet de contrôle avec le ou exclusif réalisé précédemment. Le paquet de données est ignorer si la comparaison échoue.

Le **bit de stop** est un bit à l'état haut, il termine la transmission.

Trame de données DCC* de base :



11111111111111 0 0AAAAAAA 0 01DCSSSS 0 EEEEEEEE 1

synchronisation bit de start octet d'adresse bit de start octet de commande bit de start octet de contrôle bit de stop

* : Voir lexique

III - 2.2 Remise à zéro générale du décodeur **:**

Pour mettre le compteur à zéro d'une locomotive, c'est-à-dire effacer la mémoire non permanentes, il faut envoyer les trois octets de données à 0.

III - 2.3 Signal d'arrêt général :

Pour activer le signal d'arrêt général et enclencher une procédure d'arrêt progressif (ralentissement puis arrêt), il faut que le premier octet soit à 0 et que le second soit le suivant 01DC0000. Si le dernier bit du deuxième octet est à 1, les décodeurs doivent s'arrêter immédiatement et couper l'alimentation du moteur.

III - 2.4 Espacement des paquets de données :

Pour que le décodeur réagisse aux paquets de données qu'il reçoit, il faut que l'intervalle de temps défini par le temps d'attente entre la réception du dernier bit d'un « paquet » de données et du premier bit du « paquet » de données suivant soit supérieur à 5 ms. Lors de la réception des données si un bit start, un bit stop ou un octet de contrôle invalide est reçu, le décodeur doit reconnaître la séquence de synchronisation suivante.

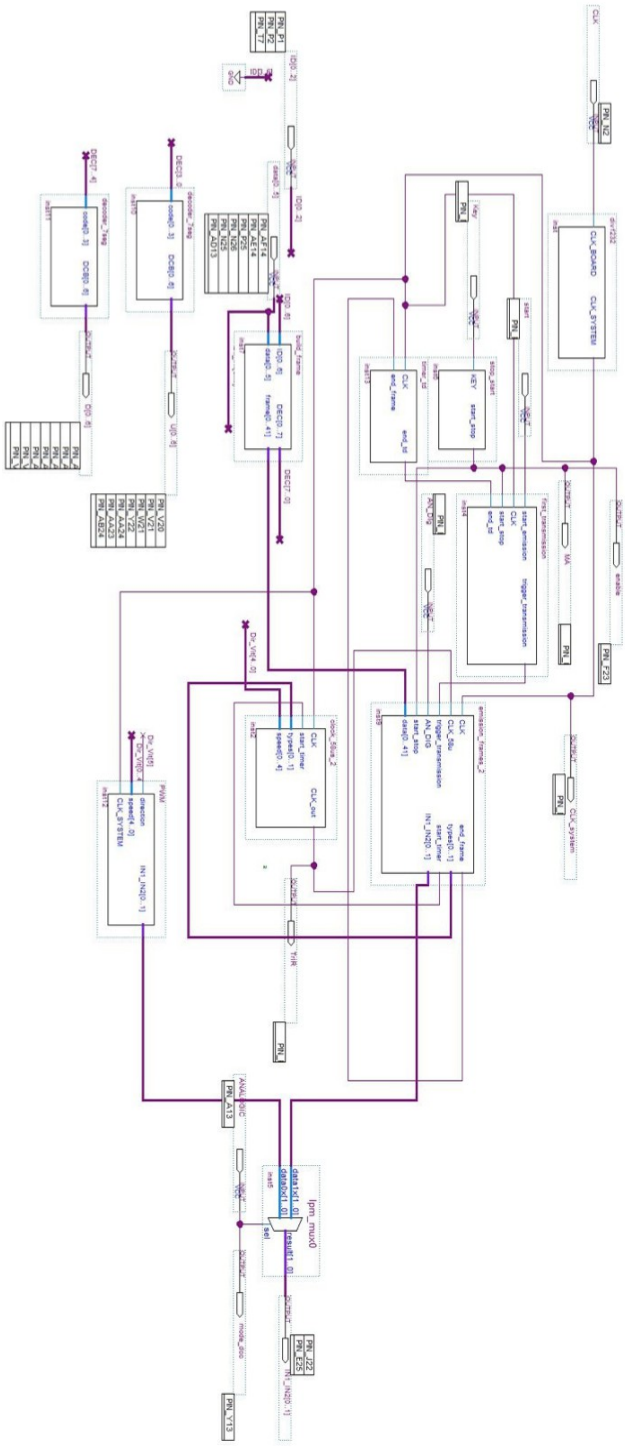
Une centrale de commande DCC* doit émettre un paquet de données au moins toutes les 30 ms (ce temps est mesuré entre les bits de start des paquets).

* : Voir lexique

IV – ARCHITECTURE MATERIELLE DU CODEUR

Le programme VHDL du codeur a été découpé en 9 sous-programmes que nous allons vous décrire dans cette partie.

IV – 1 Diagramme du système

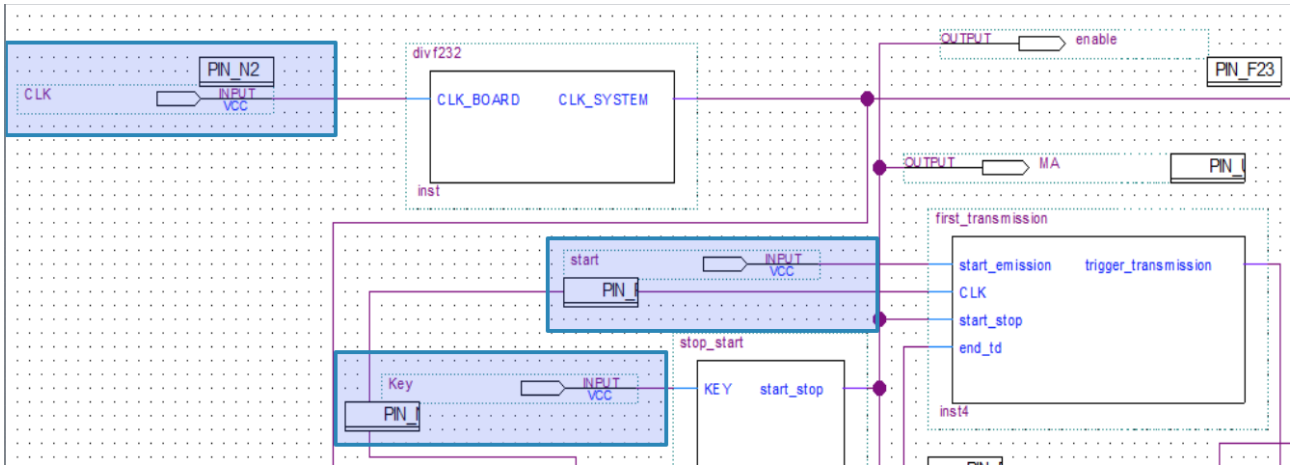


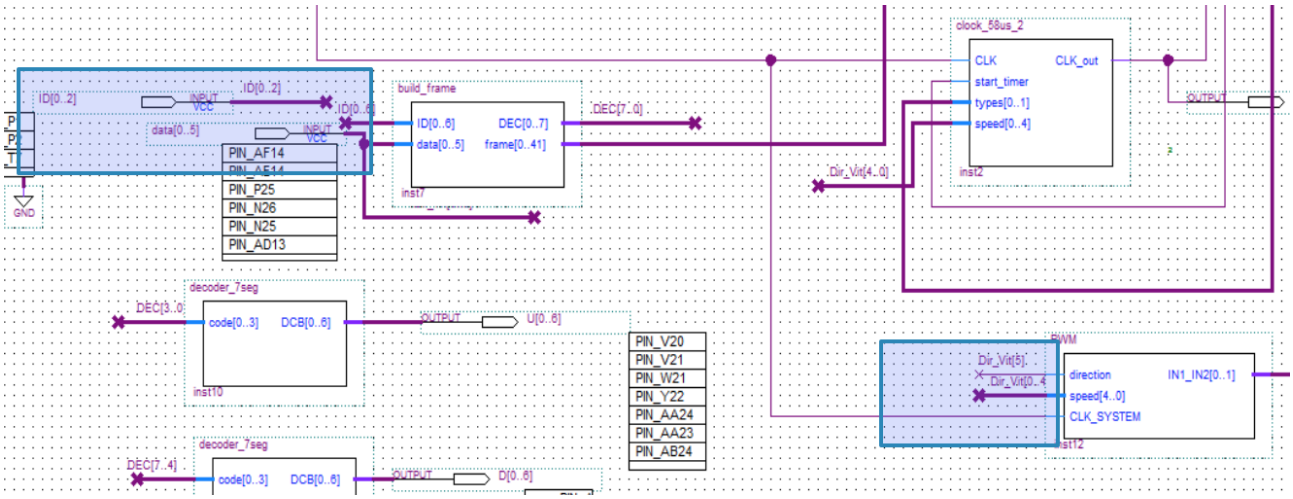
IV – 2 Description du système

Les photos suivantes permettent de décrire l'architecture matérielle du codeur que nous avons réalisé.

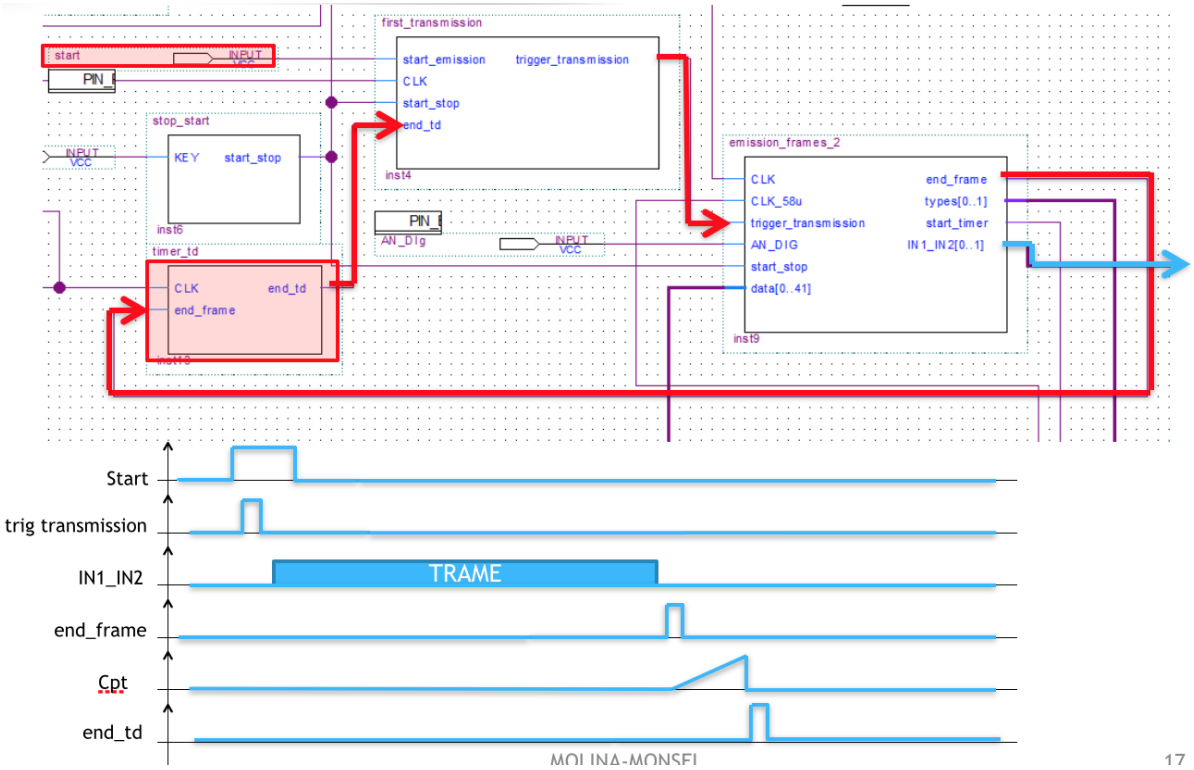
Tout d'abord, commençons par décrire les entrées encadrées en bleu dans les deux images ci-dessous :

- CLK : entrée d'horloge de 50MHz issue de la carte FPGA
- start : bouton d'autorisation d'envoi de trame
- Key : bouton de mise en marche ou d'arrêt du système
- ID[0..2] : adresse des locomotives
- Dir_Vit[5] : bit D de l'octet de commande
- Dir_Vit[0..4] : bit vitesse de l'octet de commande (CSSSS)

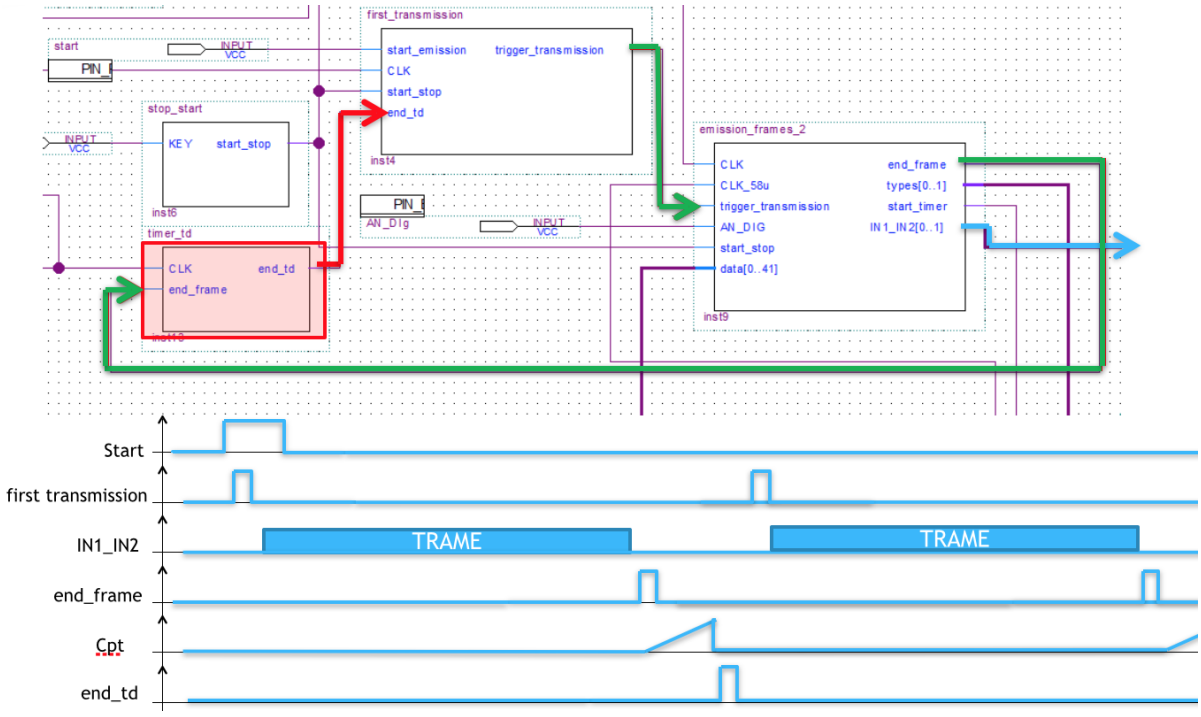




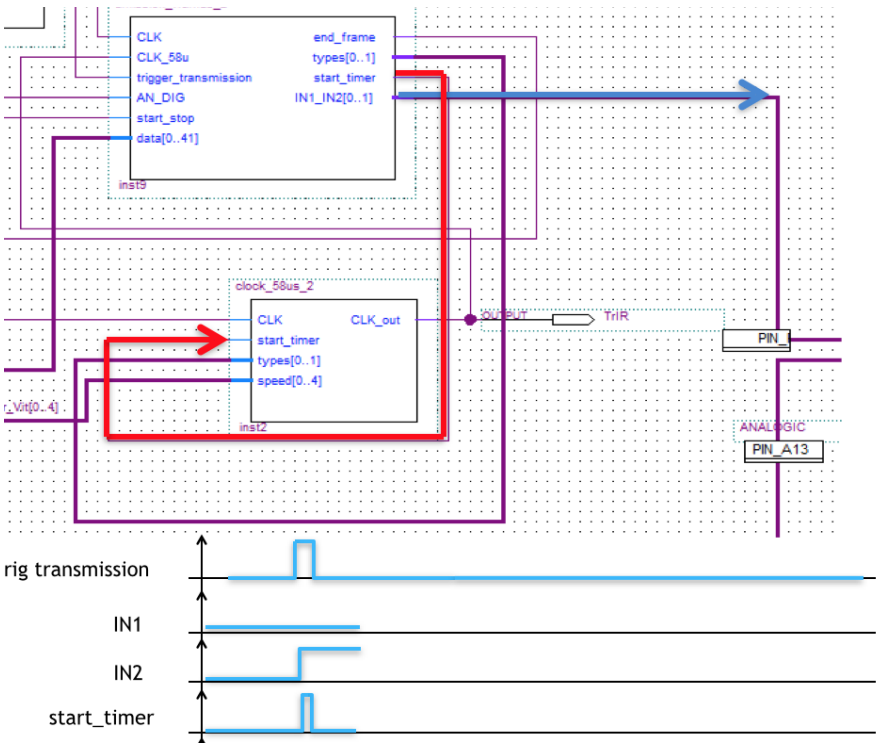
Lors de l'appui sur la touche start avec start_stop actif (mise en marche du système), la sortie trigger_transmission passe à 1 et autorise le bloc emission_frames à envoyer des trames. Lorsque la trame est complètement envoyé, la sortie end_frame passe à 1 et informe le bloc timer_td que la trame est complète, ce dernier déclenche un compteur de 5 ms et à la fin des 5 ms active la sortie end_td.



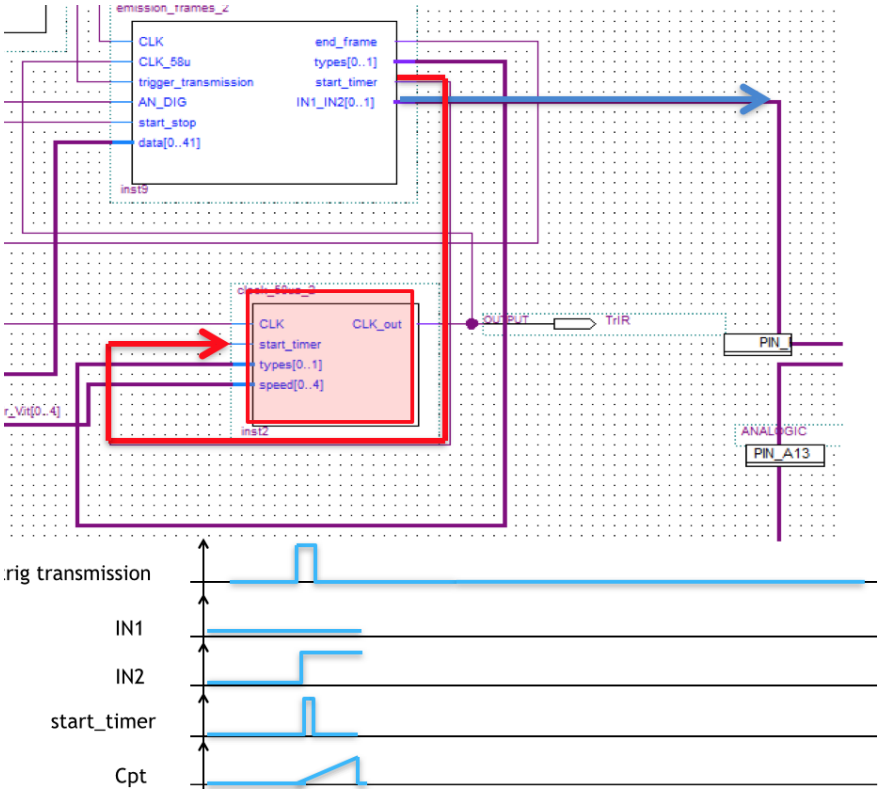
Une fois que la sortie end_td passe à 1, la sortie trigger_transmission passe à 1 et autorise l'envoi d'une nouvelle trame. Conformément à la norme nem 670, une trame doit être envoyé 3 fois avec un intervalle de temps entre chaque trame de 5ms.



Une fois que trigger emission est activée, on commence par l'alternance négative en envoyant la sortie IN2 à l'état haut.

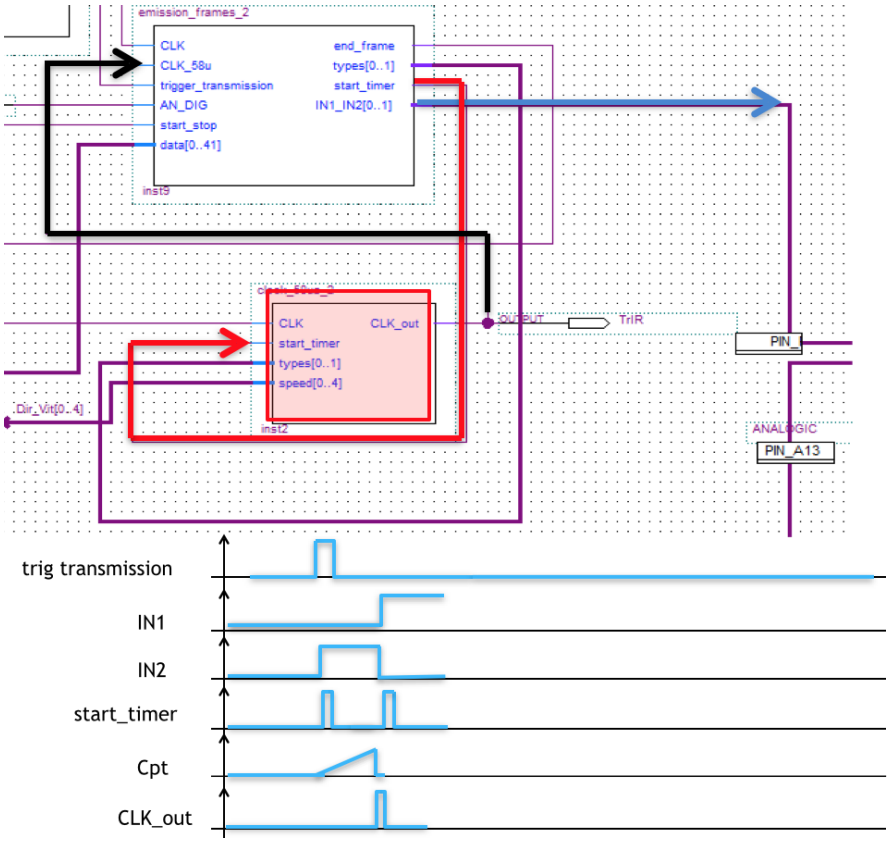


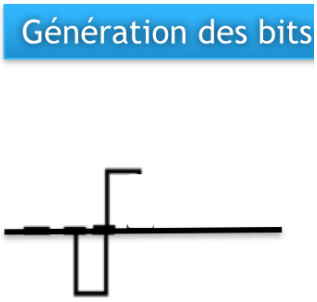
Le changement des sorties active aussi une tempo basée sur un compteur utilisé pour définir la durée de chaque alternance.



Génération des bits

A la fin du comptage le timer est réinitialisée et averti au bloc d'émission qui doit inverser les sorties pour changer la polarité de l'alternance. Quand le timer est terminé, on inverse de nouveau les sorties. On a bien un bit '1' car la durée est égale 116 µs. Puis on a la même procédure pour le bit '0' sauf au niveau du compteur. C'est due à la différence de durée des impulsions entre un bit '1' et un bit '0'. Le bit '0' à une durée supérieure ou égale à 100µs, donc au total, on aura un temps de 200µs pour un bit '0'.





19

IV – 3 Affectation des broches

KEY 0 : rien

KEY 1 : marche / arrêt

KEY 2 : émission de trame

KEY 3 : rien

SW0 : bit V0 (bit 0 de la vitesse)

SW1 : bit V1 (bit 1 de la vitesse)

SW2 : bit V2 (bit 2 de la vitesse)

SW3 : bit V3 (bit 3 de la vitesse)

SW4 : bit V4 (bit 4 de la vitesse)

SW5 : bit D (sens de déplacement)

SW6 : rien

SW7 : rien

SW8 : An_Dig(0)

SW9 : An_Dig(1)

SW10 : rien

SW11 : A(0) (de l'octet d'adresse)

SW12 : A(1) (de l'octet d'adresse)

SW13 : A(2) (de l'octet d'adresse)

SW14 : A(3) (de l'octet d'adresse)

SW15 : A(4) (de l'octet d'adresse)

SW16 : A(5) (de l'octet d'adresse)

SW17 : A(6) (de l'octet d'adresse)

IV – 4 choix du mode de fonctionnement

An_Dig(0)	An_Dig(1)	Mode choisi
0	X	analogue
1	0	numérique
1	1	Numérique-analogique

IV – 5 Correspondance entre les bit V et les bits S et C

vitesse	V4	V3	V2	V1	V0	C	S3	S2	S1	S0
STOP	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	1	0
2	0	0	0	1	0	1	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	1	0	0	1	1
5	0	0	1	0	1	0	0	1	0	0
6	0	0	1	1	0	1	0	1	0	0
7	0	0	1	1	1	0	0	1	0	1
8	0	1	0	0	0	1	0	1	0	1
9	0	1	0	0	1	0	0	1	1	0
10	0	1	0	1	0	1	0	1	1	0
11	0	1	0	1	1	0	0	1	1	1
12	0	1	1	0	0	1	0	1	1	1
13	0	1	1	0	1	0	1	0	0	0
14	0	1	1	1	0	1	1	0	0	0
15	0	1	1	1	1	0	1	0	0	1
16	1	0	0	0	0	1	1	0	0	1
17	1	0	0	0	1	0	1	0	1	0
18	1	0	0	1	0	1	1	0	1	0
19	1	0	0	1	1	0	1	0	1	1
20	1	0	1	0	0	1	1	0	1	1
21	1	0	1	0	1	0	1	1	0	0
22	1	0	1	1	0	1	1	1	0	0
23	1	0	1	1	1	0	1	1	0	1
24	1	1	0	0	0	1	1	1	0	1
25	1	1	0	0	1	0	1	1	1	0
26	1	1	0	1	0	1	1	1	1	0
27	1	1	0	1	1	0	1	1	1	1
28	1	1	1	0	0	1	1	1	1	1

V - HACHEUR « BOOSTER »

V – 1 Description

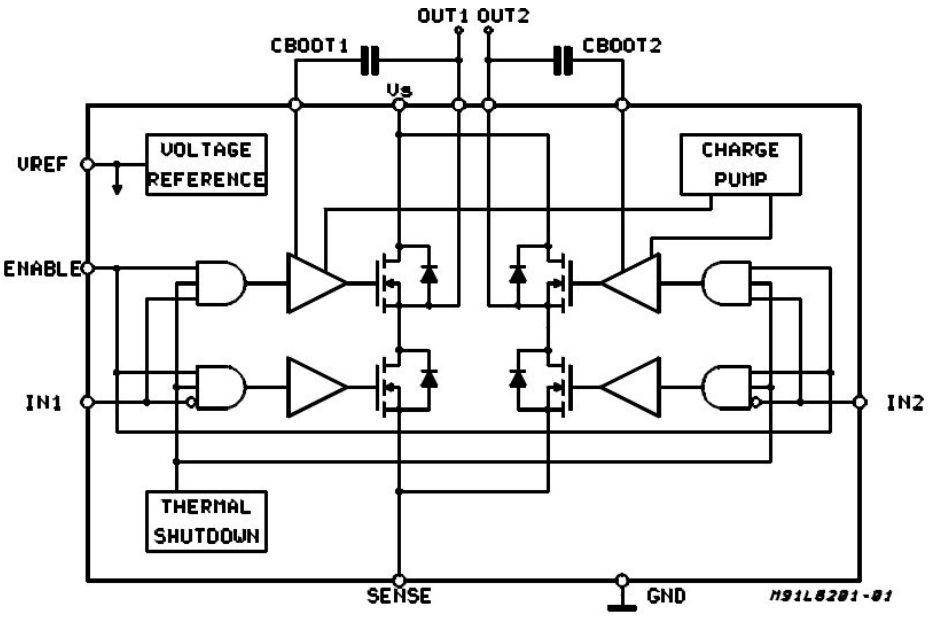
Le « Booster » est le nom utilisé dans le domaine du modélisme ferroviaire ou il s'agit d'un hacheur 4 quadrants (nom utilisé en électronique de puissance).

Ce hacheur permet de réaliser la commande en puissance du train miniature en contrôlant l'alimentation des rails avec un signal de commande. Par la suite, ce signal est reçu par le train miniature, plus précisément, par le décodeur du train qui reçoit l'information et la traduit grâce à un microcontrôleur. Le microcontrôleur transforme ce signal en une PWM* qui permet à son tour de régler la vitesse du train.

V – 2 Fonctionnement

On relie la carte FPGA à notre hacheur L6203. On utilise les signaux EN, IN1 et IN2 issue de la carte FPGA pour commander le hacheur L6203 soit en +18V soit en -18V. La tension entre OUT1 et OUT2 est égale à ±18v et est commandé par l'activation des entrées du hacheur et l'enable.

V – 3 Description des cas de fonctionnement



EN	IN1	IN2	Vout = OUT1- OUT2
0	X	X	0V
1	1	0	+ 18V
1	0	1	- 18V
1	1	1	0V

* : Voir lexique

V – 4 Carte réalisée

RPD1 et RPD2 sont des résistances de tirage, elles permettent d'assurer la tension aux bornes des entrées IN1 et IN2. Pour les hautes fréquences, les condensateurs permettent de filtrer le bruit à la masse. La masse de la carte « Booster » et la masse de la carte FPGA sont reliées ensemble pour créer une masse commune.

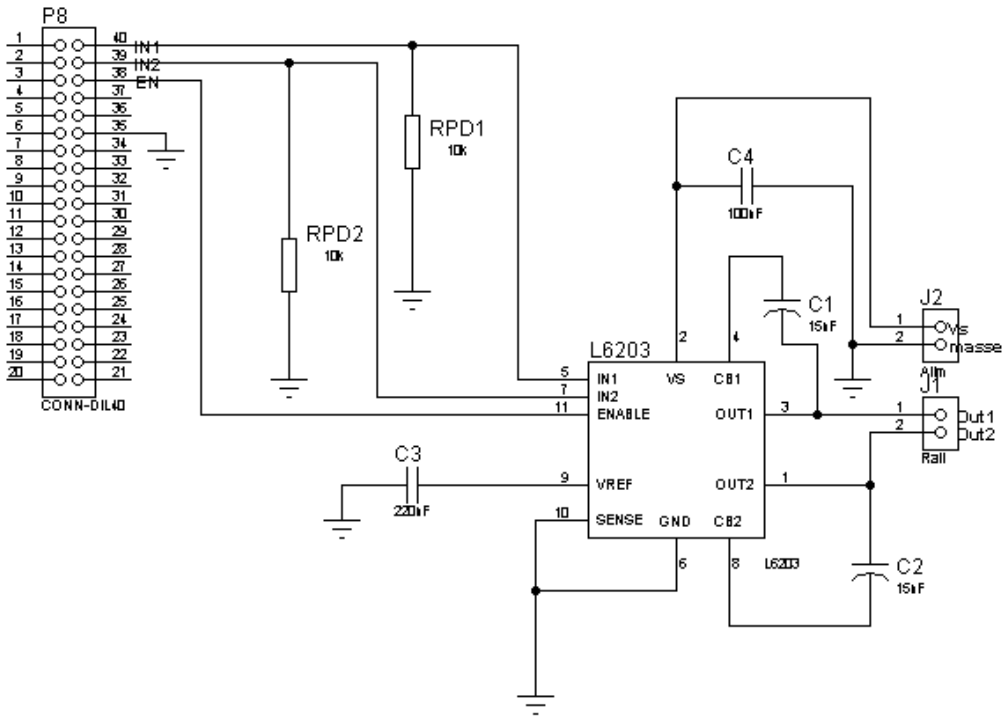


schéma réalisé sous isis

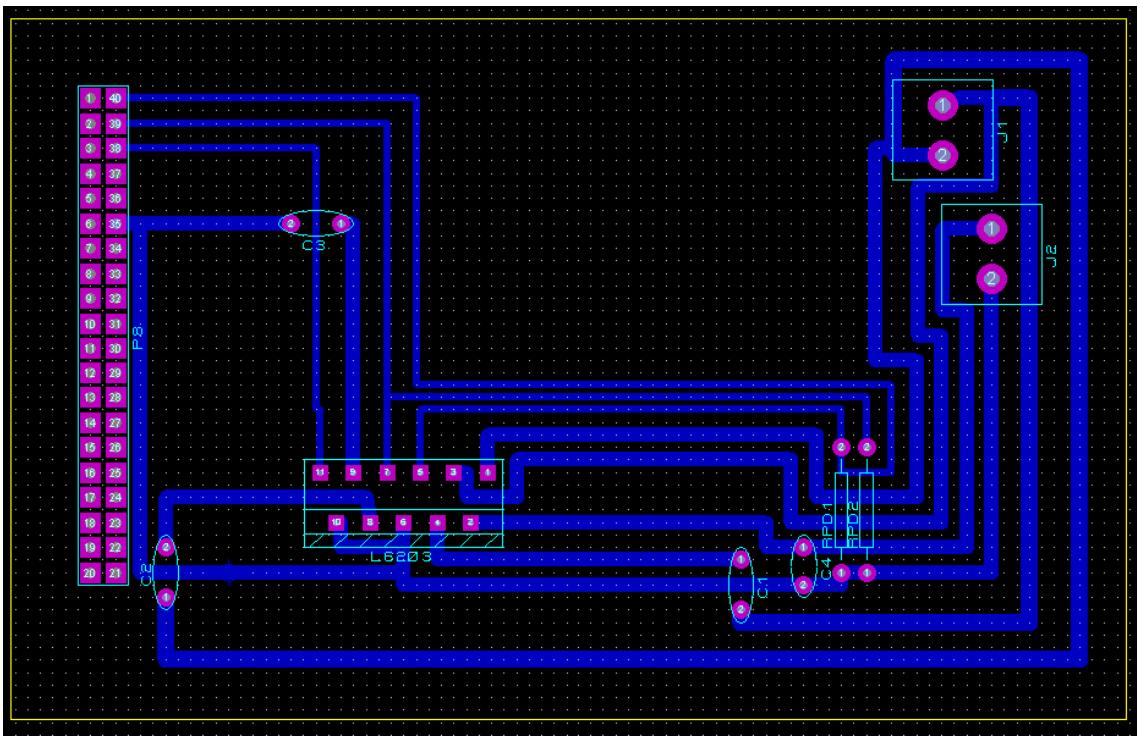


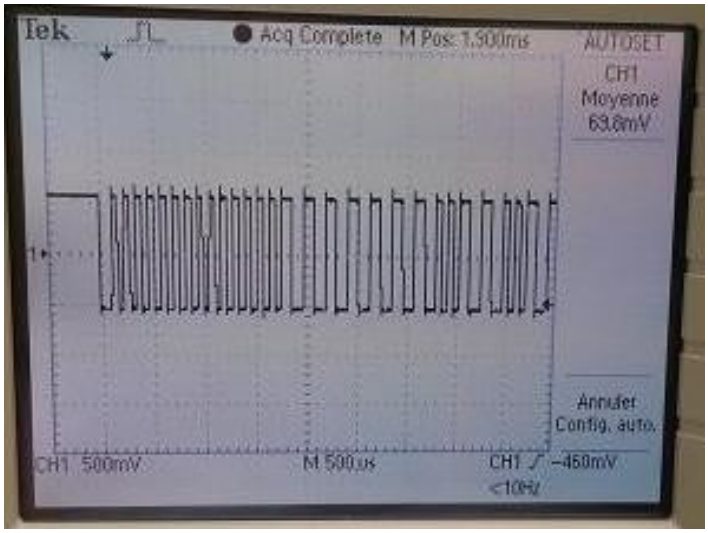
schéma réalisé sous ares

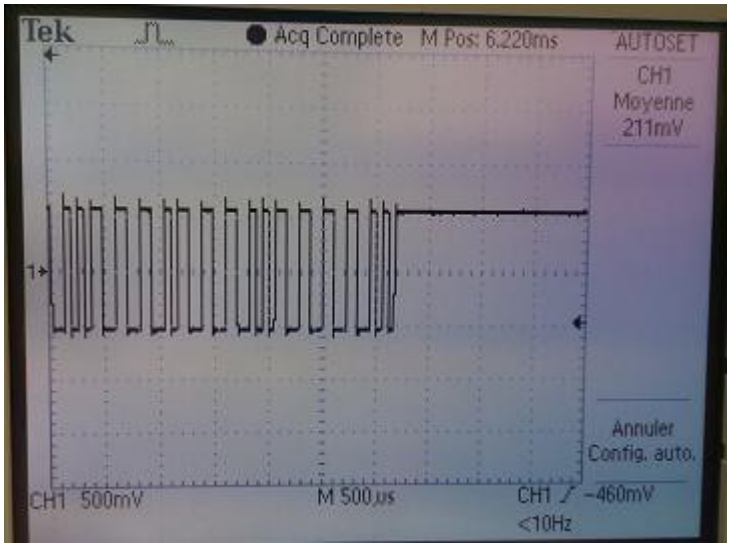
VI - REALISATION

VI – 1 Réalisation permettant de valider le système

VI – 1.1 Visualisation de trames sur l'oscilloscope

Sur les deux prochaines photos, on peut voir une trame reçu par le rail avec une division d'amplitude par 20. Cela nous a permis de vérifier que l'on recevais bien la trame voulue avec des durées cohérentes (58µs de durées d'alternance pour un bit '1' et 100µs de durées d'alternance pour un bit '0').

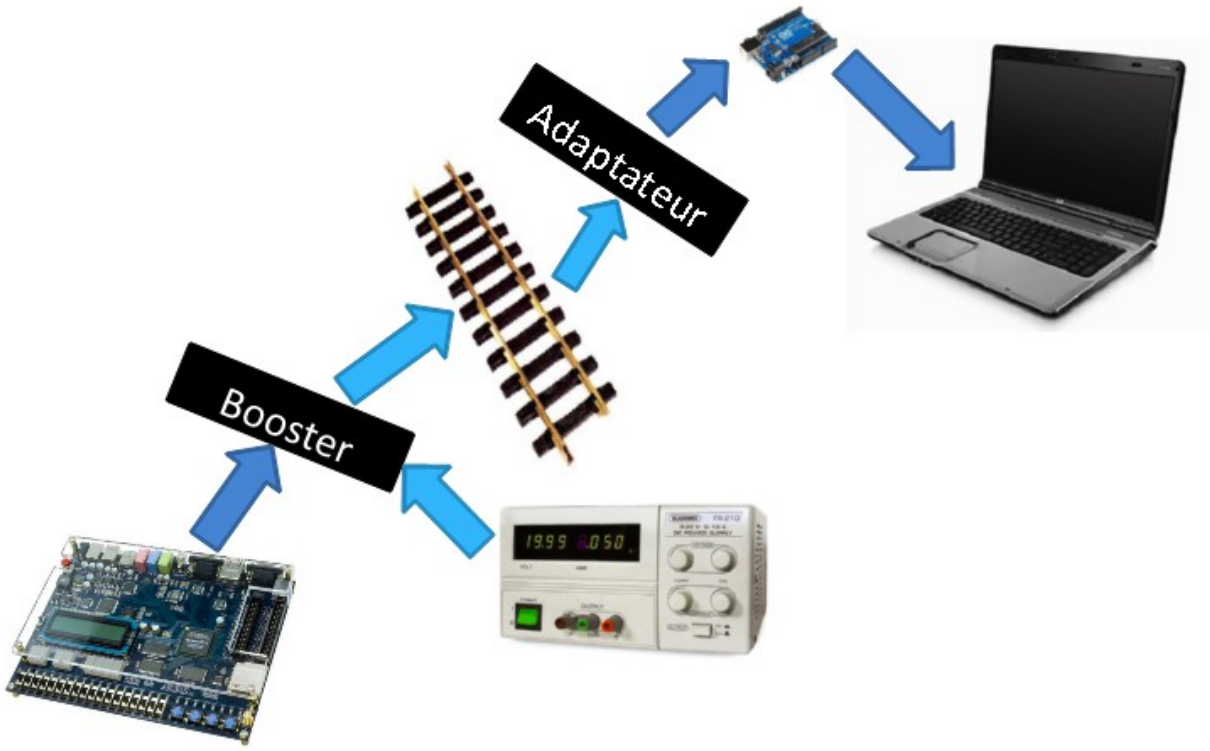




VI – 1.2 Validation de l'émission de trame DCC

Ne disposant pas de locomotive numérique à ce stade du projet, afin de valider l'émission de trame DCC, nous avons récupéré sur internet une bibliothèque DCC implémentée sur une carte Arduino [ref]

Ce circuit arduino est un outil de test et de validation selon la configuration ci-dessous :



On a réalisé ce montage pour faire le lien entre le hacheur « Booster » et le circuit arduino. Ce montage permet de réaliser un isolement galvanique. Cet isolement galvanique nous permet d'éliminer les connexions électriques entre le hacheur « Booster » (circuit de puissance) et le circuit arduino. L'isolement galvanique est réalisé grâce à un optocoupleur 2601, nous avons choisi cet optocoupleur car il présente des caractéristiques qui correspondent à nos besoins, nous avions besoin que l'optocoupleur fonctionne à une fréquence à 100 KHz, ce qui est le cas car l'optocoupleur peut fonctionner à une fréquence de 10 MHz.

Finalement, le cheminement est le suivant : on a le hacheur « Booster » qui reçoit les signaux IN1 et IN2 issue de la carte FPGA. Le signal en sortie du « Booster » est un signal de puissance et est également un signal de commande car le signal est reçu par le décodeur du train miniature, le décodeur récupère les octets envoyés et les décodes pour savoir s'il s'agit d'une trame qui permet de commander le train miniature en question et comment il faut le commander. Le signal en sortie du décodeur est transformé par un hacheur en un signal de type PWM, cela permet par la suite de commander le train. D'autre part, le montage ci-dessus réalise l'isolement galvanique et envoie le signal sur la patte 2 de l'arduino.

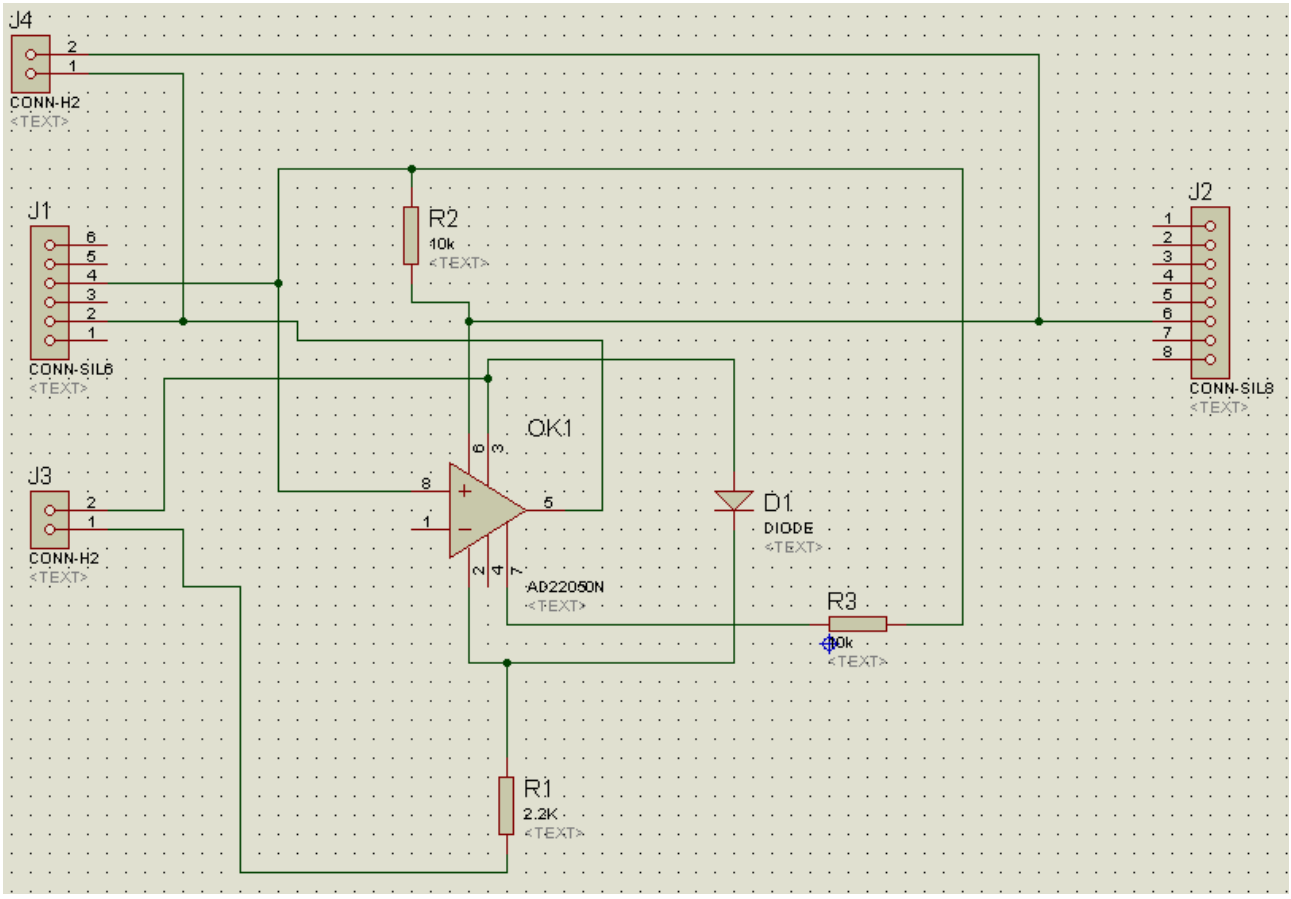


schéma de la carte optocoupleur réalisé sous isis

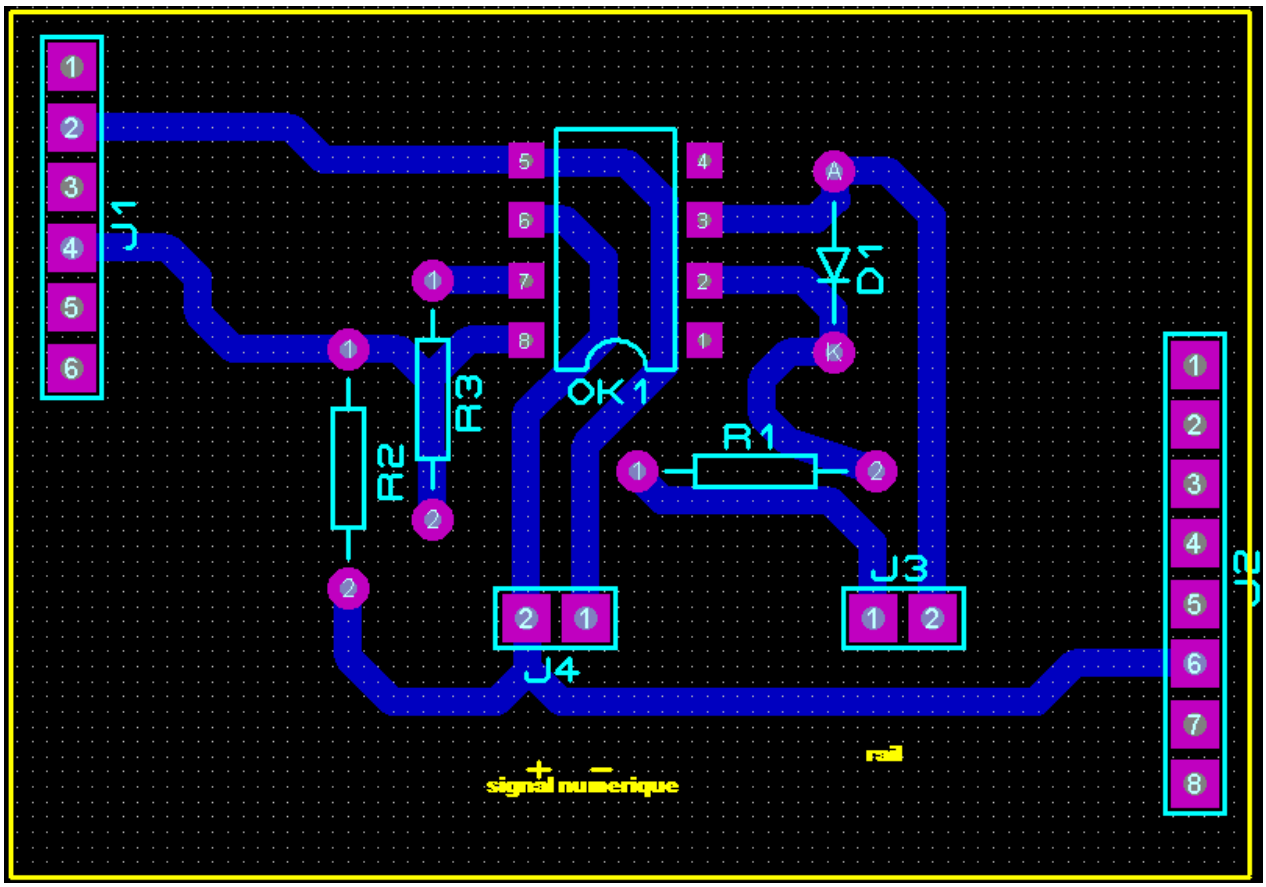
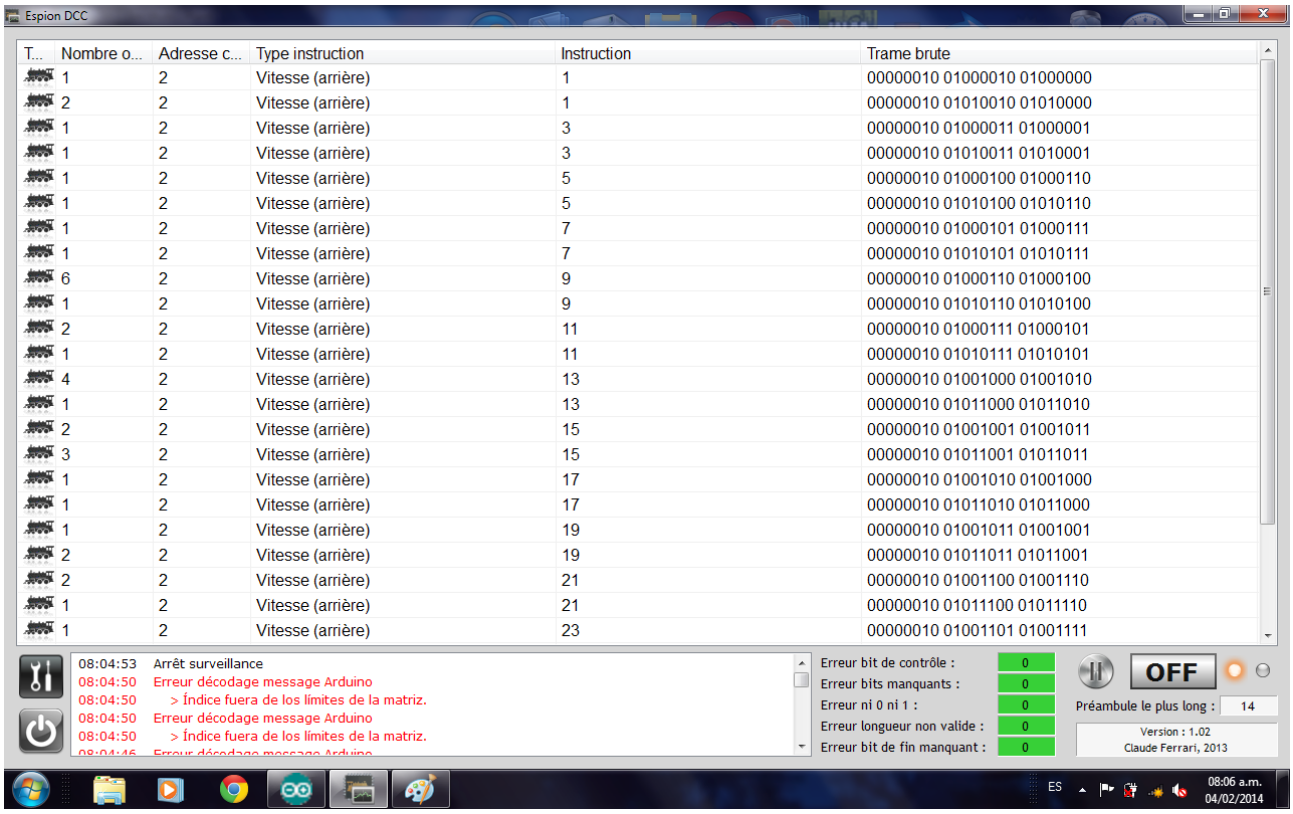
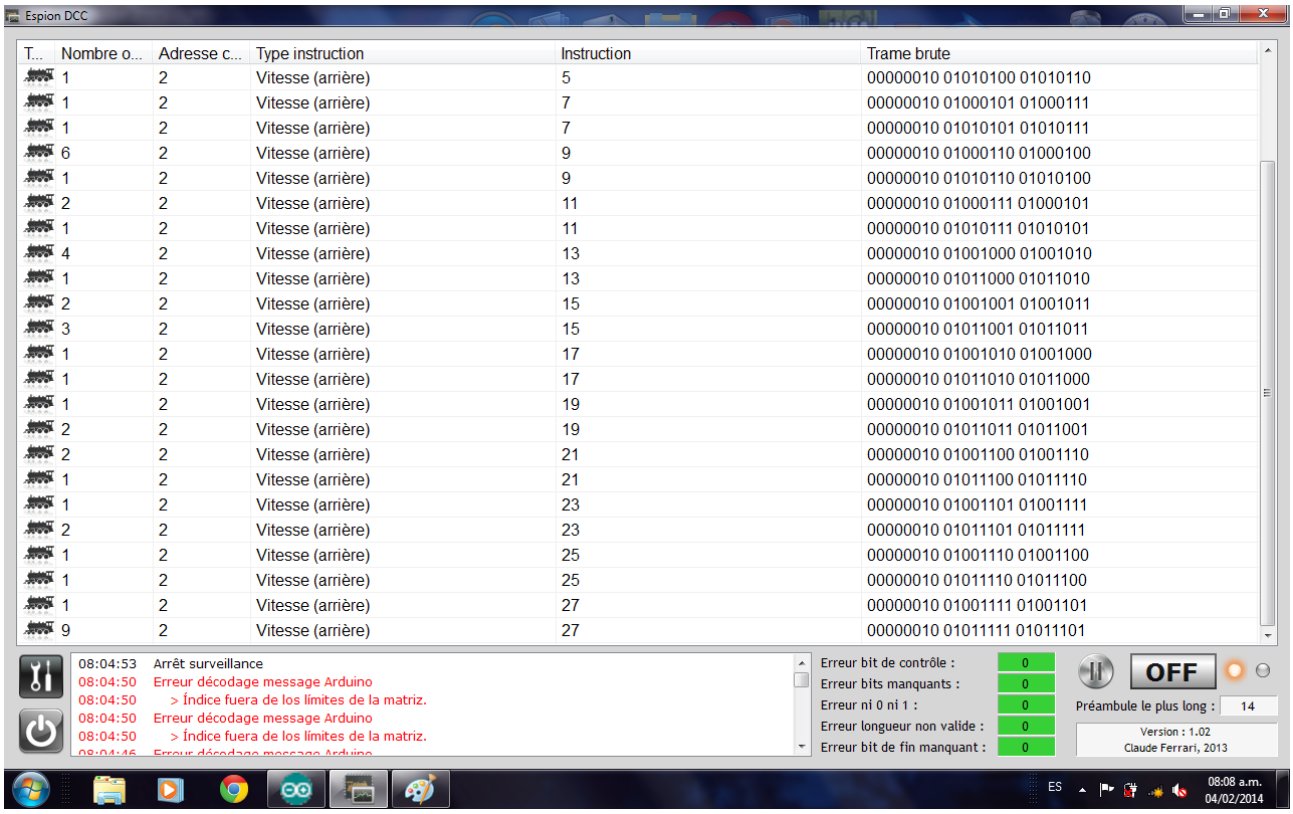


schéma de la carte optocoupleur réalisé sous ares

D'un point de vue informatique, le signal est traité par un programme (réalisé sur le logiciel arduino) que nous avons trouvé sur internet et qui a pour but d'analyser si nous envoyons bien une trame appartenant à la norme DCC*. Un autre logiciel appelé espion DCC* reçoit la trame et l'analyse en nous indiquant l'adresse du train, la vitesse (instruction), le sens de déplacement, la trame reçu et le nombre de fois qu'elle a été reçu. Ces informations sont reçues dans le cas où la trame envoyée appartient bien à la norme DCC*.

Sur les deux images suivantes, nous avons réalisé un test pour savoir si nous recevions bien les trames voulues, nous avons envoyé les trames correspondant aux 28 vitesses disponibles en marche arrière (décrit dans la norme NEM 671 page 15)





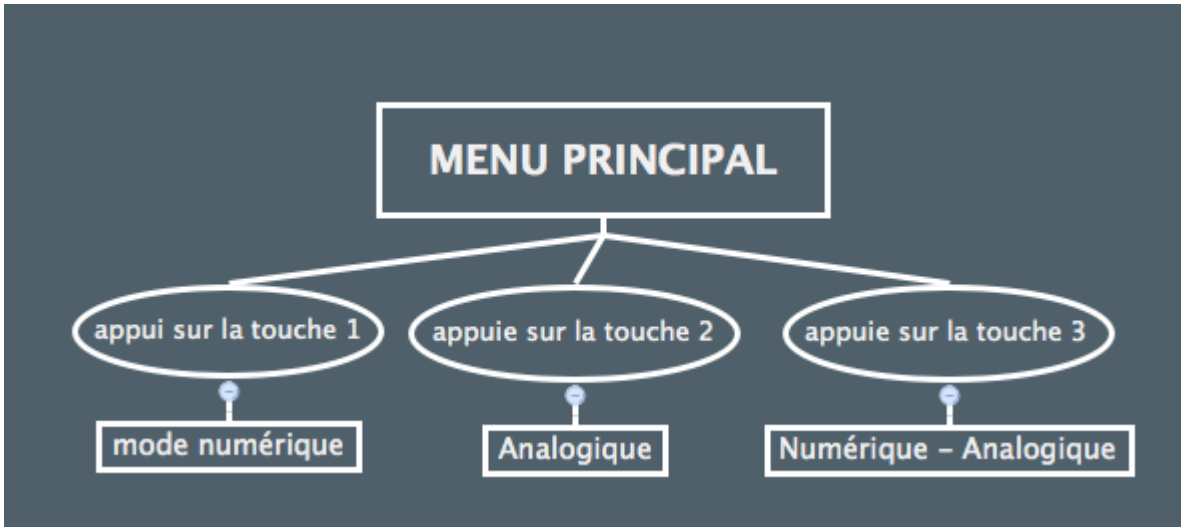
L'image suivante montre l'instruction stop et plusieurs vitesse en marche avant.

VII - TELECOMMANDE

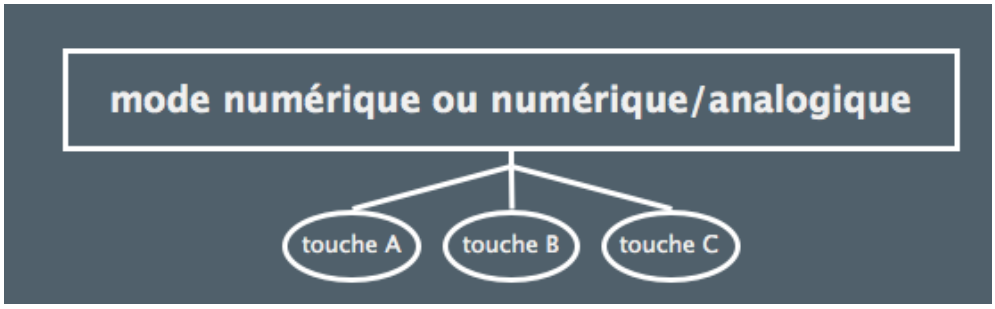
VII – 1 Manuel d'utilisation de la télécommande

Dans cette partie, nous allons détaillé le manuel d'utilisation de la télécommande infrarouge. Nous débutons avec un organigramme d'utilisation de la télécommande, de manière à connaître l'utilité de chaque touche.

La première image représente le choix que l'on a à l'allumage du système. On a le choix entre le mode numérique, le mode analogique et le mode numérique-analogique (pour commander des locomotives analogiques avec une trame DCC*).

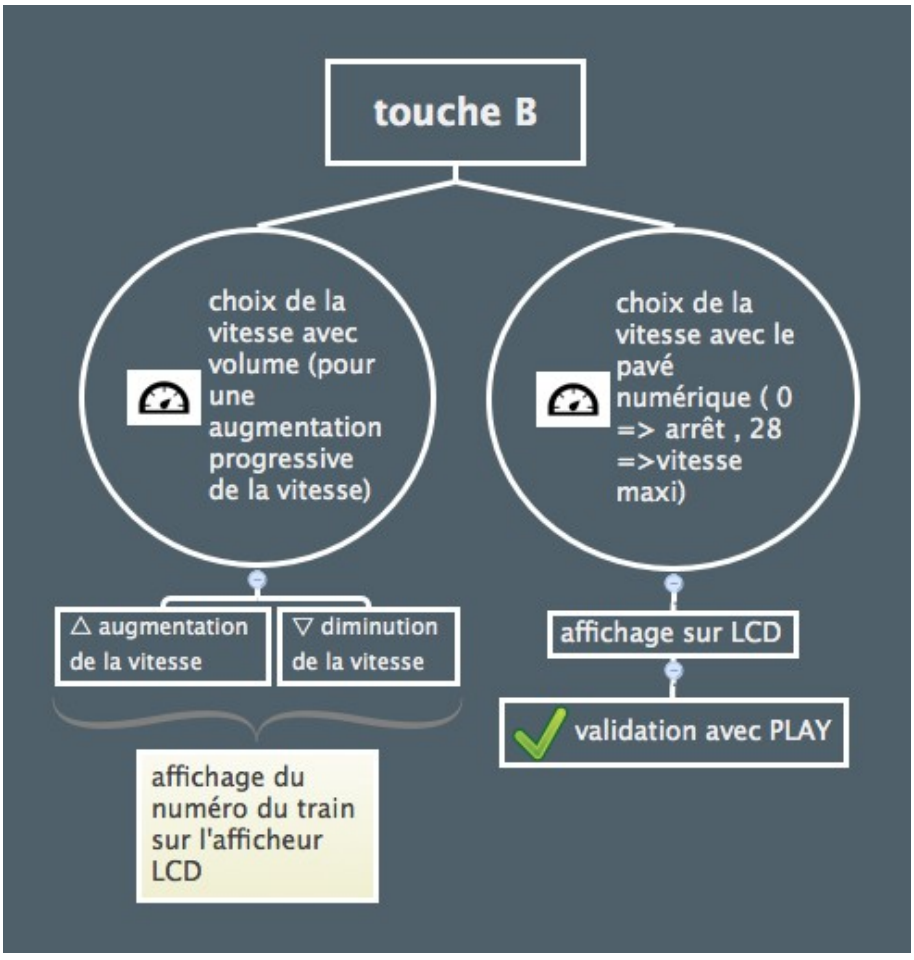


Après l'appui sur la touche 1 ou 3, on peut choisir entre la touche A, B et C dont on va détaillé les fonctions à la page suivante.

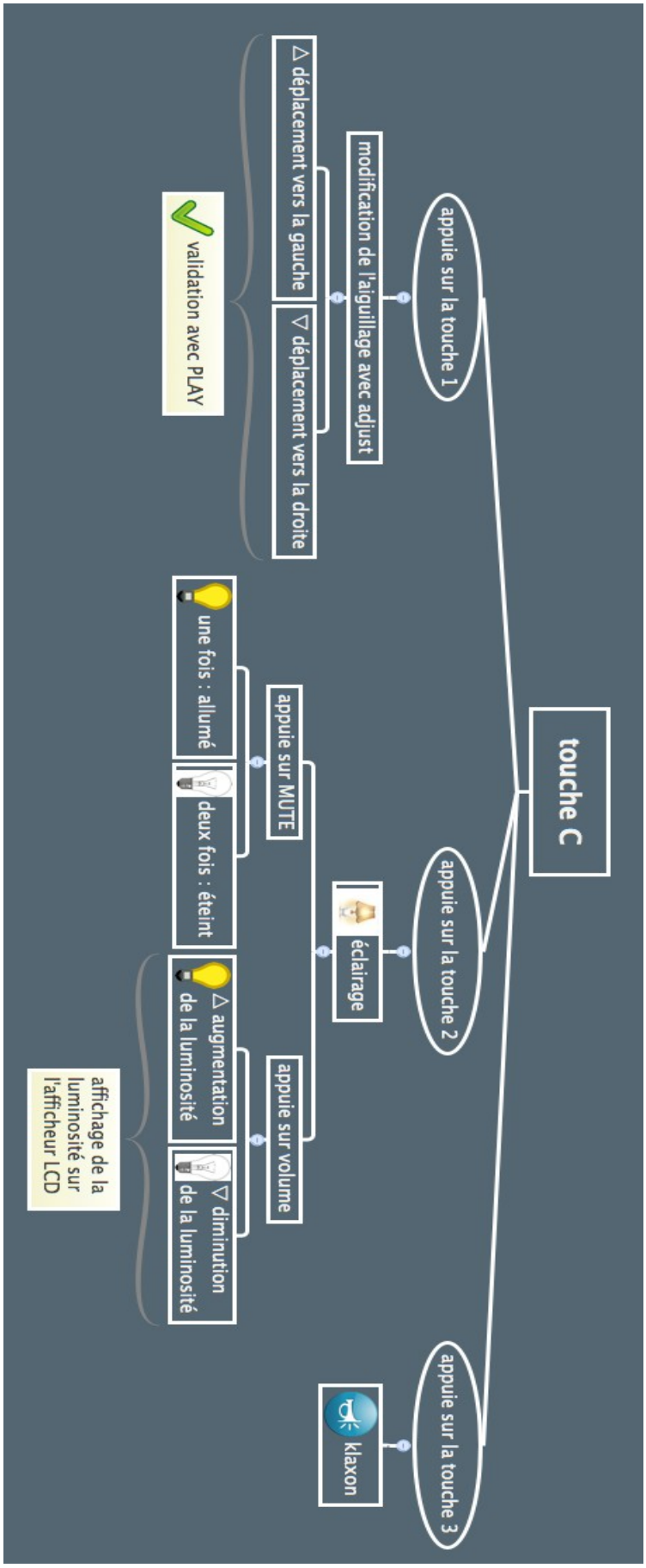


Lorsque l'on appuie sur la touche A, cela permet de choisir la locomotive. Il ne reste plus qu'à choisir avec channel la locomotive que l'on désire commander.

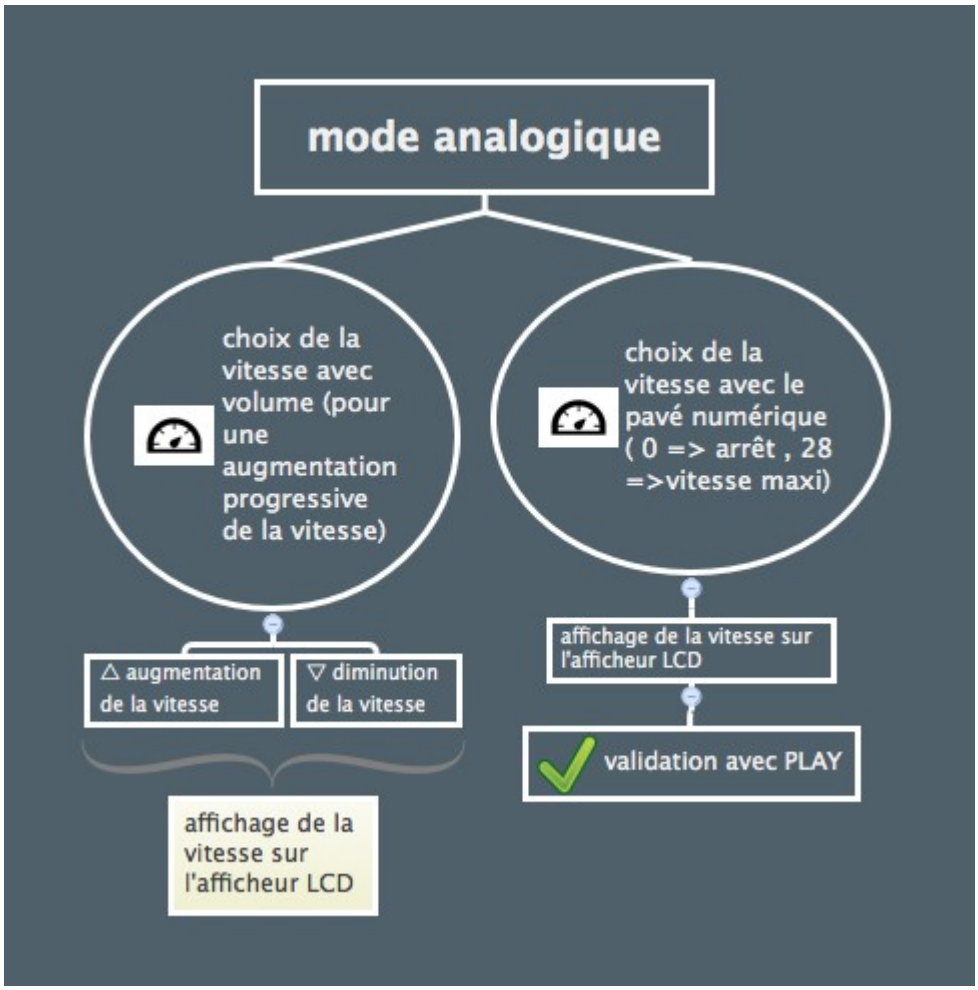
Dans le cas où l'on appuie sur la touche B, on dispose de deux possibilités pour choisir la vitesse : soit avec les touches volumes ou avec le pavé numérique.



La dernière touche que l'on peut appuyé dans le sous-menu numérique ou dans le sous-menu numérique-analogique est la touche C qui nous permet d'accéder à toutes les options : aiguillage, éclairage et klaxon.

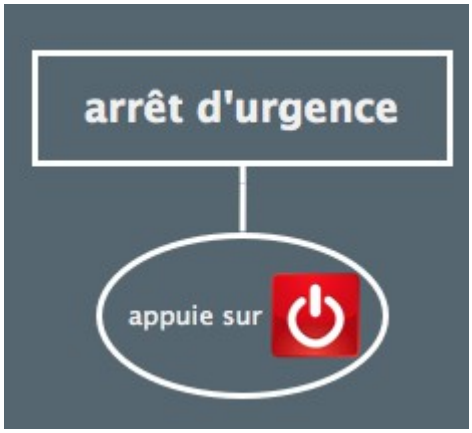


Dans le cas où l'on appuie sur la touche 2 à partir du menu principal, on arrive dans le sous menu du mode analogique et l'on peut choisir uniquement la vitesse soit avec les touches volumes, soit avec le pavé numérique.



A n'importe quel moment, lorsque l'on appuie sur la touche return, on retourne en arrière (par exemple du sous menu du choix de l'aiguillage au menu du choix des options).

Une autre touche qu permet de faire une action à n'importe quel moment est la touche POWER, elle permet de faire un arrêt d'urgence.



VIII - BILAN

L'objectif de ce projet était de réaliser un codeur de train miniature appartenant à la norme DCC*. Ce codeur a permis de communiquer des signaux au hacheur « Booster ». Ces deux éléments ont été réalisés. Ils vont pouvoir être exploités par un autre groupe de projet, un groupe qui réalisera le décodeur et le hacheur. Ce dernier permettra de faire le lien entre le codeur, le décodeur et le moteur.

Nous avons réalisé des sous programmes pour rendre le programme le plus clair possible et dans le cas où le groupe de projet réalisant le décodeur doit faire quelques modifications pour s'adapter à leur système, elles seront plus facilement réalisable car le programme n'est pas en un seul programme compacte mais en plusieurs sous programmes correspondant à une fonction précise.

Durant ce projet, nous avons toujours eu une bonne entente au sein de notre binôme. Cela nous a permis de travailler ensemble et de répartir le travail convenablement tout en faisant des petits bilans du travail réalisé pour pouvoir confronter les avancées de chacun régulièrement et par la suite savoir quelle direction nous devions prendre dans le projet.

Ce projet nous a permis de nous améliorer considérablement en langage VHDL et en langage C et nous a permis de mener à bien un projet, ce projet nous a permis de changer notre manière de travailler, nous avons commencé par collecter des informations plutôt que de débuter le projet sans se renseigner sur ces réelles attentes, on a également appris à faire des brouillons pour pouvoir réfléchir avant de faire le travail demandé.

IX – CONCLUSION

Ce projet est intéressant car l'engouement pour les trains miniatures est grand, il y a de nombreuses possibilités envisageable car ils existent de nombreuses fonctions et systèmes que l'on peut mettre sur un train miniature. Ce sujet est également intéressant car il est pluridisciplinaire : on a dans ce projet de l'informatique industrielle, de l'électronique de puissance… L'objectif que nous avons eu durant cette année est de créer notre propre système et ce système devait être l'image des produits du commerce autrement dit que notre système soit utilisable par une locomotive vendu dans le commerce.

X – PERSPECTIVES

Sur l'image suivante, nous pouvons voir un exemple de suite à notre projet. Nous pouvons envisager d'utiliser des réseaux sans fils (bluetooth, wifi, infrarouge), des réseaux filaires (port USB, ethernet, Liaison RS232) disponible sur la carte FPGA pour commander nos trains à distance et grâce à des applications disponible sur internet.

D'autres fonctions peuvent être mis en œuvre barrière qui descendent toutes seules à l'approche d'un train, klaxon et éclairage automatique en entrant dans un tunnel, allumage de l'éclairage public la nuit...

XI - ANNEXE

XI – 1 Programme VHDL

XI – 1.1 diviseur de fréquence

Dans le sous-programme suivant, nous adaptons la fréquence du quartz (50MHz) issue de la carte VHDL que nous utilisons pour obtenir un signal de 1us en sortie du sous-programme.

```
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--                              Diviseur de la fréquence                                                    --
--    Horloge d'entrée : Quartz 50MHz                                                                       --
--    Horloge de sortie: 10MHz                                                                              --
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
```

```
library leee;
use ieee.std_logic_1164.all;

ENTITY divf232 IS
port(
  CLK_BOARD  : IN std_logic;
  CLK_SYSTEM : out std_logic
);
END divf232;

ARCHITECTURE archi of divf232 IS
signal cpt : integer range 0 to 5:=0;              -- variable de rapport cyclique
signal next_cpt    : integer range 0 to 5:=0;      -- variable de comptage
BEGIN
  -- Séquentielle
  process
  begin
    wait until CLk_BOARD'event and CLK_BOARD='1';
      cpt <= next_cpt ;
  end process;

  -- Combinatoire
  next_cpt <= 0 when cpt = 4 else cpt + 1;
  CLK_SYSTEM <= '1' when cpt < 2 else '0';             -- rapport cyclique 50%

END archi;
```

XI – 1.2 Programme permettant la mise en marche ou l'arrêt du système

```
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--                    programme permettant la mise en marche ou l'arrêt                                          --
--    Entrée "Key"  : bouton poussoir                                                                            --
--    sortie              : signal marche/arrêt                                                                  --
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------

library leee;
use ieee.std_logic_1164.all;

ENTITY stop_start IS
port(
  KEY          : IN std_logic;
  start_stop   : buffer std_logic        -- sortie bidirectionelle
);
END stop_start ;

ARCHITECTURE archi of stop_start IS
BEGIN
  Process (KEY)

  Begin

    if KEY='0' and KEY'event then
      start_stop <= not(start_stop); --inversion de la sortie start_stop à chaque appui sur
.                                    --le bouton poussoir
    end if;

  end process;
END archi;
```

XI – 1.3 Programme du temps d'attente de 5ms

```
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--                          Programme du temps d'attente de 5ms                                             --
--    Horloge d'entree   : 10MHz                                                                            --
--    Entree "end_frame" : provenant du bloc "emission_frame"                                               --
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------

library leee;
use ieee.std_logic_1164.all;

ENTITY timer_td IS
  port(
        CLK      : in std_logic;
    end_frame : in std_logic;
        end_td    : out std_logic
         );
END timer_td;

ARCHITECTURE archi of timer_td IS
SIGNAL cpt   : integer range 0 to 70002;        -- variable de comptage de 5 ms
SIGNAL state : integer range 0 to 3 := 0;       -- Auxiliaire de la machine d'état
BEGIN
  Process(end_frame,CLK)
  BEGIN
    IF CLK='1' and CLK'event then
      CASE state is
-----  démarrage et test de fin de trame -----
       WHEN 0      => end_td<='0';                  -- Remet la sortie à l'état bas
                      cpt<=0;                       -- Réinitialisation du comptage
                       if end_frame='1' then        -- Attente de la fin de la trame
                        state<=1;
                       else
                        state<=0;
                       end if;

----- comptage de 5000 fronts montants d'horloge -----
            WHEN 1      => cpt<= cpt+1;
                           end_td<='0';             -- Remet la sortie à l'état bas
                            if cpt>50001 then
                              state<=2;
                            else                    -- Boucle de comptage
                              state<=1;
                            end if;

----- fin de comptage et réinitialisation-----
       WHEN 2      => cpt<=0;
                      state<=0;
                      end_td<='1';              -- Met la sortie à l'état haut
```

```
            WHEN OTHERS => state<=0;
                           end_td<='0';
                           cpt<=0;
     end CASE;
    end if;
  end process;

end archi;
```

XI – 1.4 Programme de la construction de la trame

```
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--                    Programme de la construction de la trame                                                    --
--    Entrée "ID" et "data"  : interrupteurs                                                                      --
--    Sortie "frame" : trame complète (basé sur la norme DCC mais sans '0' et '1' DCC)    --
-- Format de la trame DCC :1111111111111 0 0AAAAAAA 0 01DCSSSS 0 EEEEEEEE 1 –
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------

library leee;
use ieee.std_logic_1164.all;

ENTITY build_frame IS
              port(
                ID    : IN std_logic_vector(0 to 6);  -- numéro d'identification de la      .
.                                                     -- locomotive =[0AAAAAAA]
                      data   : IN std_logic_vector(0 to 5);  -- fonction =  [DCSSSS]
                      DEC    : out std_logic_vector(0 to 7);
                      frame  : out std_logic_vector(0 to 41)
                      );
       END build_frame ;

ARCHITECTURE archi of build_frame IS
signal builder : std_logic_vector(0 to 41);
signal S4C     : std_logic_vector(0 to 4);
  begin
  builder(0 to 15)  <= "1111111111111100";        -- synchronisation + le 0 de start avant le
.                                                 -- premier octet de donnée  et le 0
                                                  -- du premier octet de données
  builder(23 to 25) <= "001";                  -- bit de start  + "01" appartenant au deuxième
.                                              -- octet de données
  builder(32)       <= '0';                  -- bit de start avant le troisième octet de données
  builder(41)       <= '1';                  -- bit de stop correspondant à la fin de la trame
  builder(16 to 19) <= ID(3 to 6);           -- numéro d'identification de la locomotive
  builder(20) <= ID(2);                      -- numéro d'identification de la locomotive
  builder(21) <= ID(1);                      -- numéro d'identification de la locomotive
  builder(22) <= ID(0);                      -- numéro d'identification de la locomotive
```

```
  with data(0 to 4) select
    S4C <= "00000" when "00000", -- CSSSS = "00000" (STOP) quand aucun interrupteur
.                                   -- n'est actionné
          "00010" when "00001", -- CSSSS = "00001" (vitesse 1) quand SW0 ='1'
          "10010" when "00010", -- vitesse 2
          "00011" when "00011", -- vitesse 3
          "10011" when "00100", -- vitesse 4
          "00100" when "00101", -- vitesse 5
          "10100" when "00110", -- vitesse 6
          "00101" when "00111", -- vitesse 7
          "10101" when "01000", -- vitesse 8
          "00110" when "01001", -- vitesse 9
          "10110" when "01010", -- vitesse 10
          "00111" when "01011", -- vitesse 11
          "10111" when "01100", -- vitesse 12
          "01000" when "01101", -- vitesse 13
          "11000" when "01110", -- vitesse 14
          "01001" when "01111", -- vitesse 15
          "11001" when "10000", -- vitesse 16
          "01010" when "10001", -- vitesse 17
          "11010" when "10010", -- vitesse 18
          "01011" when "10011", -- vitesse 19
          "11011" when "10100", -- vitesse 20
          "01100" when "10101", -- vitesse 21
          "11100" when "10110", -- vitesse 22
          "01101" when "10111", -- vitesse 23
          "11101" when "11000", -- vitesse 24
          "01110" when "11001", -- vitesse 25
          "11110" when "11010", -- vitesse 26
          "01111" when "11011", -- vitesse 27
          "11111" when others;  -- vitesse 28 quand la valeur sur les interrupteurs est
.                                 -- supérieure à 27



  builder(26) <= data(5);             -- bit D de la trame

  builder(27)<= S4C(0);               -- bit C de la trame
  builder(28)<= S4C(1);               -- bit S3 de la trame
  builder(29)<= S4C(2);               -- bit S2 de la trame
  builder(30)<= S4C(3);               -- bit S1 de la trame
  builder(31)<= S4C(4);               -- bit S0 de la trame

  builder(33 to 40) <= builder(15 to 22) XOR builder(24 to 31); -- octet de contrôle =
.                                                               -- [EEEEEEEE]
  DEC<= builder(15 to 22);                        -- Affichage sur afficheur
  frame <= builder;                               -- affectation finale
end archi;
```

XI – 5 Programme d'émission

```
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
--                                  Programme d'émission                                                              --
--    Horloge d'entrée                  : 10MHz                                                                       --
--    Horloge d'appel                   : 19.6KHz (51 µs)                                                             --
--    Entrée "trigger_transmission " : Signal d'autorisation d'émission                                               --
--    Entrée "start_stop"               : signal marche/arrêt                                                         --
--    Entrée "end_td"                   : Front montant provenant du "timer_td"                                       --
--    Sorties "IN1,IN2"                 : Broches du connecteur GPIO 40 pins                                          --
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------

library leee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

ENTITY emission_frames_2 IS
port(
  CLK                    : in std_logic;
  CLK_58u                : in std_logic;
  trigger_transmission   : in std_logic;
  AN_DIG                 : in std_logic;
  start_stop             : in std_logic;
  data                   : in std_logic_vector(0 to 41);
  end_frame              : out std_logic;
  types                  : out std_logic_vector(0 to 1);
  start_timer            : out std_logic;
  IN1_IN2                : out std_logic_vector(0 to 1)
  );
END emission_frames_2;

ARCHITECTURE v of emission_frames_2 IS
signal etat : integer range 7 downto 0 := 0;          -- Variable de la machine d'état
signal S : std_logic;                                 -- Variable utilisée pour complémenter les sorties
signal position : integer range -1 to 42:=-1;         -- Indice de position dans la trame
BEGIN
  Process(CLK,start_stop)
  Begin
    if start_stop = '0' then                          -- Positionnement initial en cas de stop général
           etat <= 0;
    elsif CLK='1' and CLK'event then
      CASE etat is

----- Attente de la fin d'émission de la trame et l'autorisation d'émission -----
        when 0 => end_frame <='0';
                  if trigger_transmission ='1' then
                    etat <= 1;
                  else
                    etat <= 0;
                  end if;
```

```
                S <='0';

----- Identification des bits -----
        ---------- select ----------
           when 1  => if data(position)='1' then
                types <= "00";              -- bit '1'
              elsif AN_DIG = '0' then
                types <= "01";              -- bit '0' numérique
              else
                types <= "10";              -- bit '0' analogique
              end if;
              etat <= 2;


 ----- Etat négatif -----
           when 2  =>
                   start_timer<='1';
                    if CLK_58u='1' then
                      etat <=  3;
                      S<='1';
                      start_timer<='0';
                    end if;

----- Etat positif -----
           when 3 =>
                   start_timer<='1';
                    if CLK_58u='1' then
                      etat <=  4;
                      S<='0';
                      start_timer <= '0';
                    end if;



----- Augmentation/réinitialisation de la position -----
           when 4  =>   if position = 41 then
                  end_frame <= '1';
                  etat <= 0;
                  position <= -1;
                else
                       position <= position+1;
                  etat <= 1;
                end if;

                 when others => etat <= 0;
    end CASE;
   end if;
  end process;

IN1_IN2(0)   <= S;                          -- affectation finale de la sortie IN1
IN1_IN2(1)   <= not(S);                     -- affectation finale de la sortie IN2
END v;
```

XI – 1.6 Programme d'autorisation d'émission

```
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--                          Programme d'autorisation d'émission                                              --
_                                                                                                            --
--    Horloge d'entrée        : 10MHz                                                                        --
--    Entrée "start_emission" : bouton poussoir                                                              --
--    Entrée "start_stop"     : signal marche/arrêt                                                          --
--    Entrée "end_td"         : Front montant provenant du "timer_td"                                        --
--    Sortie "trigger_transmission" : autorisation d'émission                                                --
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------

library leee;
use ieee.std_logic_1164.all;

ENTITY first_transmission IS
                port(
                        start_emission      : IN std_logic;
                        CLK                 : IN std_logic;
                        start_stop          : IN std_logic;
                        end_td              : IN std_logic;
                        trigger_transmission : out std_logic
                        );
        END first_transmission ;

ARCHITECTURE archi of first_transmission IS
signal state : integer range 0 to 4;       -- Auxiliaire de la machine d'état
signal N : integer range 0 to 5;           -- Variable de comptage du nombre de répétitions
BEGIN
  Process (CLK)
  Begin
    if CLK='1' AND CLK'event then
          CASE state IS

----- Attente du start et test du stop général -----
                        WHEN 0 => if start_emission='0'  AND start_stop='1' then
                            state<=1;
                                    else
                            state<=0;
                          end if;
                           N <= 0;          -- Réinitialisation du nombre comptant les répétitions

----- Comptage du nombre de répétitions -----
                    WHEN 1 =>  state<=2;
                               N <= N+1;


----- Attente de la fin du timer de 5 ms -----
                    WHEN 2 =>  if end_td ='1'then
                    state<=3;
```

```
                end if;


----- Test du nombre de répétitions et de stop général -----
                  WHEN 3 =>  if (N<1 ) then
                   state<=1;
                  else
                    state<=4;
                  end if;
                if start_emission='1'then
                 end if;

          WHEN 4 => if start_emission='1' OR start_stop='0' then
                     state<=0;
                   end if;

         WHEN OTHERS => state<=0;
     END CASE;
    end if;
  end process;


  trigger_transmission <= '1' when state = 1 else '0';          -- autorisation d'émission

END archi;
```

XI – 1.7 Programme PWM (pour train analogique)

```
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--                          Programme réalisant d'un signal PWM                                             --
--    Horloge d'entrée : Horloge du système 10 MHz                                                          --
--    Horloge de sorties: Commande de l'hacheur                                                             --
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;

ENTITY PWM IS
port(
  direction  : IN std_logic;
  speed      : IN integer range 0 to 31;
  CLK_SYSTEM : IN std_logic;
  IN1_IN2    : out std_logic_vector(0 to 1)
);
END PWM;

ARCHITECTURE archi of PWM IS
signal cpt        : integer range 0 to 336 := 0;
signal next_cpt   : integer range 0 to 336 := 0;          -- variable de comptage
signal alpha      : integer range 0 to 336 := 0;          -- variable de rapport cyclique
signal PWM        : std_logic;
BEGIN

 -- Séquentielle

 process
 begin
  wait until CLK_SYSTEM'event and CLK_SYSTEM='1';
    cpt <= next_cpt ;
 end process;

 -- Combinatoire

 next_cpt <= 0 when cpt = 336 else cpt + 1;
 PWM <= '1' when cpt < alpha else '0';


 IN1_IN2(0) <= PWM when direction = '1' else '0'; -- IN1_IN2(0) = IN1
 IN1_IN2(1) <= PWM when direction = '0' else '0'; -- IN1_IN2(1) = IN2

 WITH speed SELECT
  alpha <= 0    when 0,
           84   when 1,
           93   when 2,
          102   when 3,
```

```
111  when 4,
120  when 5,
129  when 6,
138  when 7,
147  when 8,
156  when 9,
165  when 10,
174  when 11,
183  when 12,
192  when 13,
201  when 14,
210  when 15,
219  when 16,
228  when 17,
237  when 18,
246  when 19,
255  when 20,
264  when 21,
273  when 22,
282  when 23,
291  when 24,
300  when 25,
309  when 26,
318  when 27,
327  when 28,
336  when others;

END archi;
```

XI – 1.8 Programme de la cadence de 58µs

```
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
--                          Programme de la cadence 58µs                                                              --
--    Horloge d'entrée   : 10 MHz                                                                                     --
--    Entrée "start_timer" : provenant du bloc "emission_frame"                                                       --
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------


library leee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

        ENTITY clock_58us_2 IS
                port(
                        CLK          : IN std_logic;
                        start_timer    : in std_logic;
                    types        : IN std_logic_vector(0 to 1);
          speed       : IN std_logic_vector(0 to 4);
                        CLK_out       : out std_logic
                       );
        END clock_58us_2;

ARCHITECTURE archi of clock_58us_2 IS
SIGNAL cpt        : integer range 0 to 2300;          -- variable de comptage de 58 µs
SIGNAL state      : integer range 0 to 50 := 0;        -- variable de la machine d'état
SIGNAL add        : integer range 0 to 1100:=0;       -- variable d'augmentation de la
                                                      -- durée du bit '0'
SIGNAL speed_an   : integer range 0 to 2300:=0;   -- variable de la vitesse analogique
BEGIN
  Process( start_timer,CLK)
  BEGIN
    IF CLK ='1' and CLK 'event then
      CASE state is

----- réinitialisation du compteur, attente du start et choix du type de bit -----

      WHEN 0      => CLK_out<='0';            -- Remet la sortie à l'état bas
                 cpt<=0;                      -- Réinitialisation du comptage
                                    if  start_timer='1' then      -- Attente de la start_timer
                                      if types = "00" then        --  s'il s'agit d'un bit '1'
                                        state<=1;
                                      else                        --  s'il s'agit d'un bit '0'
                                        state<=2;
                                      end if;
                                    end if;
```

----- comptage de 51 fronts montants d'horloge -----

```
        WHEN 1      => cpt <= cpt+1;
                       CLK_out<='0';          -- Remet la sortie à l'état bas
                        if cpt=580 then
                          state<=4;
                        else
                          state<=1;            -- Boucle de comptage
                        end if;
```

----- temporisation de la 1ère impulsion du bit '0' et sélection du type de bit '0' -----

```
      WHEN 2     =>  cpt<= cpt+1;
                     CLK_out<='0';           -- Remet la sortie à l'état bas
                     if cpt=1100 then
                       if types = "10" then      -- s'il s'agit d'un '0' analogique
                         CLK_out<='1';           -- Met la sortie à l'état haut
                         state<=3;
                         cpt<=0;
                       else
                         state<=4;
                       end if;
                     else
                       state<=2;                 -- Boucle de comptage
                     end if;
```

----- temporisation du bit '0' analogique -----

```
      WHEN 3      => cpt <= cpt+1;
                     CLK_out <= '0';           -- Remet la sortie à l'état bas
                     if cpt = speed_an then
                       state<=4;
                     else
                       state<=3;                 -- Boucle de comptage
                     end if;
```

----- fin de comptage et réinitialisation-----

```
      WHEN 4      => CLK_out <= '1';           -- Met la sortie à l'état haut
                      state<=5;
                      cpt <= 0;

      WHEN 5      => CLK_out <= '0';
                     state <= 0;

          WHEN OTHERS => state <= 0;
                         CLK_out <= '0';
                         cpt <= 0;
    end CASE;
  end if;
end process;
```

```
WITH speed SELECT
add <= 1   when "00000",
          1   when "00001",
         4   when "00010",
         8   when "00011",
        12  when "00100",
        16  when "00101",
        20  when "00110",
        24  when "00111",
        28  when "01000",
        32  when "01001",
        36  when "01010",
        40  when "01011",
        44  when "01100",
        48  when "01101",
        52  when "01110",
        56  when "01111",
        60  when "10000",
        64  when "10001",
        68  when "10010",
        72  when "10011",
        76  when "10100",
        80  when "10101",
        84  when "10110",
        88  when "10111",
        92  when "11000",
        96  when "11001",
        100 when "11010",
        104 when "11011",
        108 when "11100",
        108 when others;

speed_an<=(add*10)+1100;

end archi;
```

XI – 1.9 Programme pour l'affichage d'information sur la carte FPGA

```
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--                          Programme du décodeur pour l'affichage                                          --
--  Entrée "code" : identification du numéro du train (information issue des interrupteurs)  --
--  Sortie "DCB"  : sortie vers l'afficheur 7 segments                                                      --
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------


library leee;
use ieee.std_logic_1164.all;

ENTITY decoder_7seg IS
  port(
    code : IN std_logic_vector(0 to 3);
    DCB  : out std_logic_vector(0 to 6)
     );
END decoder_7seg;

ARCHITECTURE archi of decoder_7seg IS
BEGIN
  WITH code SELECT
  DCB<= "0000001"   when "0000", --on inverse les valeurs des bits car l'afficheur 7
                                                          --segments est actif à 0
            "1001111"   when "0001",
            "0010010"   when "0010",
            "0000110"   when "0011",
            "1001100"   when "0100",
            "0100100"   when "0101",
            "0100000"   when "0110",
            "0001111"   when "0111",
            "0000000"   when "1000",
            "0000100"   when "1001",

            "0001000"   when "1010",-- à partir d'ici on affiche les valeurs supérieures à
                                    -- 9 en hexadécimal
            "1100000"   when "1011",
            "0110001"   when "1100",
            "1000010"   when "1101",
            "0110000"   when "1110",
            "0111000"   when "1111",

            "1111110"   when others;
        END archi;
```

XI – 2 Programme C

XI – 2.1 Programme C permettant d'utiliser la télécommande

```
#include "address_map.h"
#include <stdio.h>
#include <string.h>

/*****************************************************************************************
 * Global variable
*****************************************************************************************/

/********** déclaration des messages affichés sur l'écran LCD **********/

char menu[40] = "Menu Principal\0";
char mode_numerique[40] = "1-Numerique\0";
char mode_analogique[40] = "2-Analogique\0";
char mode_num_an[40] = "3-Num-an\0";
char numerique[40] = "1-numerique\0";
char loco[40] = "A-loco\0";
char vitesse[40] = "B-vitesse    \0";
char options[40] = "C-options\0";
char choix_1[40]="appuyez sur\0";
char choix_loco_2[40]="channel\0";
char choix_loco_3[40];
char choix_loco_fait[40]=" A choisir   \0";
char choix_vitesse_1[40]="Choix avec\0";
char choix_vitesse_2[40]="volume ou avec\0";
char choix_vitesse_3[40]="pave numerique\0";
char choix_vitesse_4[40]="entre 0 et 28 \0";
char choix_vitesse[40];
char choix_vitesse_fait[40]="selectionne\0";
char aiguillage[40]="1-aiguillage\0";
char eclairage[40]="2-eclairage\0";
char eclairage_allume[40]="allume\0";
char eclairage_eteint[40]="eclairage eteint\0";
char klaxon[40]="3-klaxon\0";
char adjust[40]="adjust\0";
char aiguillage_1[40]="aiguillage +\0";
char aiguillage_2[40]="aiguillage -\0";
char choix_aiguillage_fait[40]="aiguillage\0";
char choix_aiguillage_fait_2[40]="deplace\0";
char choix_eclairage[40]="appuyez sur mute\0";
char choix_eclairage_2[40]="une fois allume\0";
char choix_eclairage_3[40]="deux fois eteint\0";
char augmentation_eclairage[40]="luminosite +\0";
char diminution_eclairage[40]="luminosite -\0";
char arret_urgence[40]="arret d'urgence\0";
char choix_valide[40]="Choix valide\0";
char mute[40]="mute\0";

char texte[40]; // utilisé pour afficher des texte variable avec incrémentation d'un chiffre
```

```
/********** déclaration des variables globales **********/

  long int SW_value, KEY_value, HEX_bits,i,k,last;
  int etat=0;          // état du switch 15 (utiliser pour l'éclairage)
  int etat_2=0;        // variable utilisée pour la détection des fronts montants
  int etat_3=0;        // variable utilisée pour la détection des fronts montants
  int etat_4=1;        // variable utilisée pour la détection des fronts montants
  unsigned int x;      // variable utilisée pour l'augmentation ou la réduction de la vitesse affichée
  volatile int delay=1; // variable utilisée pour compter un temps d'attente
  int data=0;          // variable de données
  int mod_data=0;      // variable de données avant validation
  long int Num,shift_menu;

/********** déclaration des adresses d'entrée **********/

  volatile int * red_LED_ptr        = (int *) RED_LED_BASE ;   // adresse des LED rouge
  volatile int * green_LED_ptr      = (int *) GREEN_LED_BASE;          // adresse des LED verte
  volatile int * HEX3_HEX0_ptr              = (int *) HEX3_HEX0_BASE;          // adresse du premier
afficheur 7 segments
  volatile int * HEX7_HEX4_ptr              = (int *) HEX7_HEX4_BASE;          // adresse du
deuxième afficheur 7 segments
  volatile int * SW_switch_ptr      = (int *) SLIDER_SWITCH_BASE;  // adresse des interrupteurs
  volatile int * KEY_ptr           = (int *) PUSHBUTTON_BASE;           // adresse des boutons
poussoirs

/*****************************************************************************************
 * Programme principal
*****************************************************************************************/
int main(void)
{

  main_menu();       // appel le sous programme du menu principal

  while(1)
  {
     submenu();     // appel le programme du sous menu
     LCD_Display(); // appel du sous programme d'affichage des messages
   }
}

/*****************************************************************************************
 * Sous programme d'affichage des messages
*****************************************************************************************/

void LCD_Display()
{

  while(1)
  {
    SW_value = *(SW_switch_ptr);
    KEY_value  = *(KEY_ptr);

   switch (shift_menu)
     {
```

```
        case 0:

/********** affichage LCD du menu principal  **********/

            main_menu(); //appel du sous programme du menu principal
            break;

        case 5:

/********** affichage LCD du choix entre la loco, la vitesse et les options dans le mode numérique
**********/

            affichage_tournant_3(3,0,0,1,mode_numerique,loco,vitesse,options);
              // appel du sous programme affichage tournant 3
            break;

        case 51:

/********** affichage LCD du choix de la loco avec channel **********/

            affichage_tournant_2(3,0,0,1,loco,choix_1,choix_loco_2);
              // appel du sous programme affichage tournant 2
            break;

        case 510:

/********** affichage LCD du choix de la locomotive  **********/

            sprintf(choix_loco_3,"train numero %d ",mod_data);
            affichage_deux_lignes(0,0,0,1,choix_loco_3,choix_loco_fait);
              // appel du sous programme affichage deux lignes
            break;

        case  52:

/********** affichage LCD du choix de la vitesse avec volume ou le pavé numérique dans le mode
numérique  **********/

affichage_tournant_4(3,0,0,1,vitesse,choix_vitesse_1,choix_vitesse_2,choix_vitesse_3,choix_vites
se_4);
              // appel du sous programme affichage tournant 4
            break;

        case  520:

/****************affichage LCD de la vitesse ************/

            sprintf(choix_vitesse," vitesse = %d  ",mod_data);
            affichage_deux_lignes(0,0,0,1,vitesse,choix_vitesse);
              // appel du sous programme affichage deux lignes
            break;

        case  53:

/********** affichage LCD du choix des options : aiguillage, éclairage et klaxon  **********/

            affichage_tournant_3(3,0,0,1,options,aiguillage,éclairage,klaxon);
```

```
            // appel du sous programme affichage tournant 3
            break;

        case 535:

/********** affichage LCD du menu de l'aiguillage  **********/

            Clear_LCD(); // on efface l'écran du LCD
            affichage_tournant_2(0,0,0,1,aiguillage,choix_1,adjust);
            // appel du sous programme affichage tournant 2
            break;

        case 536:

/********** affichage LCD du menu de l'éclairage  **********/

            Clear_LCD(); // on efface l'écran du LCD
            affichage_tournant_2(0,0,0,1,éclairage,choix_1,mute);
            // appel du sous programme affichage tournant 2
            break;

        case 537:

/********** affichage LCD du menu du klaxon  **********/

            Clear_LCD(); // on efface l'écran du LCD
            affichage_deux_lignes(0,0,0,1,klaxon,klaxon); // appel du sous programme affichage
deux lignes
            break;

        case 6:

/********** affichage LCD du choix de la vitesse avec volume ou le pavé numérique dans le mode
analogique  **********/

affichage_tournant_4(0,0,0,1,mode_analogique,choix_vitesse_1,choix_vitesse_2,choix_vitesse_3,
choix_vitesse_4);
            // appel du sous programme affichage tournant 4
            break;

        case 60:
/********** affichage LCD de la vitesse choisie  **********/

            sprintf(choix_vitesse," vitesse = %d  ",mod_data);
            affichage_deux_lignes(0,0,0,1,vitesse,choix_vitesse);
            // appel du sous programme affichage deux lignes
            break;

        case 5352:
/********** affichage LCD du déplacement de l'aiguillage vers la gauche  **********/

            affichage_deux_lignes(0,0,0,1,aiguillage,aiguillage_1);
            // appel du sous programme affichage deux lignes
            break;
```

```
        case 5353:

/********** affichage LCD du déplacement de l'aiguillage vers la droite  **********/

            affichage_deux_lignes(0,0,0,1,aiguillage,aiguillage_2);
              // appel du sous programme affichage deux lignes
            break;

        case 5364:

/********** affichage LCD après l'allumage de l'éclairage  **********/

            switch (KEY_value)
            {
              case 4 :
           // if (SW_value == 19){break;}
             if (SW_value==19)
            {
              etat=1;
              return_menu();
            }
            if (i==0  && etat==0)
            {
              i=1;         // variable de comptage (0 = éclairage allumé, 1 = éclairage éteint)
              Clear_LCD(); // on efface l'écran du LCD
              affichage_deux_lignes(0,0,0,1,eclairage,eclairage_allume);
              etat=1;      // affectation de la variable à 1 pour qu'on ne repasse
                           // dans la boucle qu'à la prochain allumage de l'éclairage
            }
            else if(i==1 && etat==0)/********** affichage LCD après l'extinction de l'éclairage
**********/
            {
              Clear_LCD(); // on efface l'écran du LCD
              affichage_deux_lignes(0,0,0,1,eclairage,eclairage_eteint);
              i=0;    // affectation de i à 0 pour le prochaine allumage de l'éclairage
              etat=1; // affectation de la variable à 1 pour qu'on ne repasse dans la boucle qu'à la
                      // prochaine extinction de l'éclairage
            }



            break;

            default :
                if (etat == 1)  // si on est passé une fois dans la boucle ci-dessus (SW14) on exécute
                                // cette boucle
            {
              etat = 0; // on remet etat à 0 pour le prochain front montant
            }
            else if (SW_value==19)
            {
              etat = 1;
            }
            break;
            }
        break;
```

```
        case 7:

 /***affichage LCD du choix entre la loco, la vitesse et les options dans le mode analogique ***/

            affichage_tournant_3(3,0,0,1,mode_num_an,loco,vitesse,options);
              // appel du sous programme affichage tournant 3
            break;

        case 1:

/********** affichage LCD d'un message lorsque le choix est validé  **********/
            LCD_cursor(0,0);
            LCD_text(choix_valide); // affichage du texte "choix validé"
            //break;
             return;

        case 4:

/********** affichage LCD de l'arrêt d'urgence  **********/
            LCD_cursor(0,0);  // position (0,0) du curseur (première ligne, premier case)
            LCD_text(arret_urgence); // affichage du texte "arrêt d'urgence"
            return;


        default:
            LCD_cursor_off(); // appel du sous programme LCD_cursor_off

            if (etat==1)  // si on est passé une fois dans la boucle ci-dessus (SW14) on exécute
                          // cette boucle
            {
              etat=0; // on remet etat à 0 pour le prochain front montant
            }

            if (etat_2==1)  // si on est passé une fois dans la boucle ci-dessus (SW10) on exécute
                            // cette boucle
            {
              etat_2=0; // on remet etat à 0 pour le prochain front montant
            }

            if (etat_3==1) // si on est passé une fois dans la boucle ci-dessus (SW9) on exécute
                            // cette boucle
            {
              etat_3=0;   // on remet etat à 0 pour le prochain front montant
            }
    }
  }
}
```

```
/*****************************************************************************************
 * sous programme du sous-menu
*****************************************************************************************/
void submenu()
{
  KEY_value  = *(KEY_ptr);
  if(KEY_value == 0x04)
  {
    Num = SW_value; // mémorisation de la touche appuyé
    SW_value = *(SW_switch_ptr);
    switch (SW_value)
    {
     case 0:
     case 4:  shift_menu = 4; // appel de l'arrêt d'urgence
              data = 0;       // mise à 0 de data
              mod_data = 0;   // mise à 0 de mod_data
              Clear_LCD();    // on efface l'écran du LCD
     break;

    case 17: shift_menu = 0;  // appel du menu principal
            mod_data = data; // permet d'effacer les données
    break;

    case 19: return_menu(); // appel du sous programme return_menu
    break;

    case 21: data = mod_data;
    break;


    case 5:
    case 6:
    case 7:
    case 8:
    case 9:
    case 10:
    case 11:
    case 12:
    case 13:
    case 14:
    case 15:
    case 16:
    case 18:
    case 20:
    case 22:
    case 23:
    case 24:
     if (etat_4==0)
              {
                etat_4=1;  // etat_4 passe à 1 pour que l'on ne repasse
                        // dans la boucle qu'au prochain front montant
                Clv_num(); // appel du sous programme clv_num
              }
    break;
    default: shift_menu = shift_menu; // shift_menu ne change pas
```

```
   }
  }
if(KEY_value == 0x00)
  {
   etat_4=0; // on remet la variable etat_4 à 0 pour détecter le front montant
  }
   switch (shift_menu)
   {
      case 0: // menu principal
            if (SW_value == 5)
             {
               shift_menu=5; // menu du mode numérique
             }
            else if(SW_value == 6)
             {
               shift_menu=6; // menu du mode analogique
             }
            else if(SW_value == 7)
             {
               shift_menu=7; //même menu mais à bit "0" prolongé
             }
      break;

      case 5:  // menu numérique
            if (SW_value == 1)
             {
               shift_menu=51; // Après appui sur la touche A, affichage du message
                              // du choix de la locomotive
             }
            else if(SW_value == 2)
             {
               shift_menu=52; // Après appui sur la touche B, affichage du message
                              // du choix de la vitesse
             }
            else if(SW_value == 3)
             {
               shift_menu=53; // Après appui sur la touche C, affichage du message
                              // du choix des options (éclairage, aiguillage, klaxon)
             }
      break;

      case 6:
            if (mod_data != data )
             {
               shift_menu=60;  // menu du choix de la vitesse avec les boutons volume
             }
      break;

      case 7: // menu numérique-analogique
            if (SW_value == 1)
             {
                  shift_menu=51; // Après appui sur la touche A, affichage du message
                                 // du choix de la locomotive
             }
            else if(SW_value == 2)
             {
                  shift_menu=52; // Après appui sur la touche B, affichage du message
```

```
                        // du choix de la vitesse
        }
       else if(SW_value == 3)
        {
             shift_menu=53; // Après appui sur la touche C, affichage du message
                        // du choix des options (éclairage, aiguillage, klaxon)
        }
break;

case 51: // menu du choix de la locomotive avec channel

       if (mod_data != data )
        {
            shift_menu=510;  // Après appui sur la touche channel, affichage
                         // du choix de la locomotive sur le LCD
        }
break;

case 52: // menu du choix de la vitesse avec volume ou avec le pavé numérique

       if (mod_data != data )
        {
            shift_menu=520; // affichage de la vitesse
        }
break;

case 53: // menu du choix des options (aiguillage, éclairage, klaxon)
       if (SW_value == 5)
        {
            shift_menu=535; // Après appui sur la touche 1, on arrive dans le
                        // menu du choix de l'aiguillage
        }
       else if(SW_value == 6)
        {
            shift_menu=536; // Après appui sur la touche 2, on arrive dans le
                        // menu du choix de l'éclairage
        }
       else if(SW_value == 7)
        {
            shift_menu=537; // Après appui sur la touche 3, on arrive dans le
                        // menu d'activation du klaxon
        }
break;

case 535: // menu de l'aiguillage
         if (SW_value == 22)// Après appui sur la touche adjust gauche, on affiche
                        // que l'aiguillage est déplacé vers la gauche
        {
            shift_menu=5352;
        }
         if (SW_value == 23)// Après appui sur la touche adjust droit, on affiche
                        // que l'aiguillage est déplacé vers la droit
        {
            shift_menu = 5353;
        }
break;
```

```
        case 60   :
        case 5352 :
        case 5353 :
                if (SW_value == 21) // Après un appui sur la touche PLAY, on affiche le
                                    // message de validation
                  {
                    shift_menu = 01;
                  }
        break;

        case 536: // menu de l'éclairage
                if (SW_value == 24) // Après appui sur la touche MUTE, on affiche
                                    // que l'éclairage est allumé (un appui sur MUTE) ou
                                    // on affiche que l'éclairage est éteint ( deux appui sur MUTE)
                {
                  shift_menu = 5364;
                }
        break;

        case 520:
                if (SW_value == 8 || SW_value == 12) // Après appui sur la touche channel + ou -
                                                     // on affiche la vitesse sélectionnée
                {
                  shift_menu = 60;
                }

        break;

        default: shift_menu = shift_menu; // shift_menu ne change que dans le cas où il
                                          // prend une valeur parmi celle indiquée ci-dessus
    }
}

/*********************************************************************************************
 * sous programme du menu du pavé numérique
*********************************************************************************************/
void Clv_num()
{
  float chiffre=0;
  if  (shift_menu == 52 || shift_menu == 520 || shift_menu == 51 ||shift_menu == 510||shift_menu ==
6||shift_menu == 60)
    {
     if ((Num!=8)&&(Num!=12))
      {
         mod_data = ((mod_data*10)+Num);
      }
     else if (Num == 8)
      {
         mod_data=(mod_data+1);
      }
     else if(mod_data > 0)
      {
         mod_data=(mod_data-1);
      }

     chiffre =(mod_data/100.0);
```

```
      if (chiffre >= 1)
        {
          mod_data=0;
        }
   }
}

/*******************************************************************************************
 * sous programme permettant de faire un retour au menu précédent
*******************************************************************************************/
 void return_menu()
{
    unsigned int last_menu;
    mod_data = 0;
     if(shift_menu != 0 )
       {
         last_menu = shift_menu/10;
         shift_menu = last_menu;
         while (KEY_value == 0x04)
          {
            KEY_value  = *(KEY_ptr);
          }
       }
}

/*******************************************************************************************
 * sous programme de déplacement du curseur
*******************************************************************************************/
void LCD_cursor(int x, int y)
{
          volatile char * LCD_display_ptr = (char *) 0x10003050;       // affichage de 16 caractères sur
deux lignes
          char instruction;

          instruction = x;
          if (y != 0) instruction |= 0x40;
          instruction |= 0x80;                                  // permet de connaître la position du curseur
          *(LCD_display_ptr) = instruction;               // écrire dans le registre d'instruction du LCD
}

/*******************************************************************************************
 * sous programme permettant d'envoyer un texte LCD
*******************************************************************************************/
void LCD_text(char * text_ptr)
{
          volatile char * LCD_display_ptr = (char *) 0x10003050;
                              // affichage de 16 caractères sur deux lignes
          while ( *(text_ptr) )
          {
                    *(LCD_display_ptr + 1) = *(text_ptr);  // écrire dans le registre de donnée du LCD
                    ++text_ptr;
          }
}
```

```
/*****************************************************************************************
 * Sous programme permettant d'effacer le LCD
*****************************************************************************************/
void Clear_LCD()
{
        volatile char * LCD_display_ptr = (char *) 0x10003050;
// affichage de 16 caractères sur deux lignes
   *(LCD_display_ptr) = 0x01;
// effacer le registre d'instruction du LCD
}

/*****************************************************************************************
 * sous programme permettant d'effacer le curseur LCD
*****************************************************************************************/
 void LCD_cursor_off(void)
{
        volatile char * LCD_display_ptr = (char *) 0x10003050;
// affichage de 16 caractères sur deux lignes
        *(LCD_display_ptr) = 0x0C;
                                // effacer l'écran LCD
}

/*****************************************************************************************
 * fonction d'affichage LCD avec deux lignes
*****************************************************************************************/
void affichage_deux_lignes(int x,int y,int z,int t,char * texte_1,char * texte_2)
{
  last=shift_menu;   // on informe qu'il s'agit du dernier menu sélectionné
  LCD_cursor(x,y);   // position du curseur pour la première ligne
  LCD_text(texte_1); // texte affiché sur la première ligne
  LCD_cursor(z,t);   // position du curseur pour la deuxième ligne
  LCD_text(texte_2); // texte affiché sur la deuxième ligne
  submenu();         // appel le programme du sous menu
  return;
}

/*****************************************************************************************
 * fonction d'affichage LCD avec deux lignes et deux écrans
*****************************************************************************************/
void affichage_tournant_2(int x,int y,int z,int t,char * texte_1,char * texte_2,char * texte_3)
{

  volatile char *texte_j="   Choisissez \0";
  long int affichage=0; // variable utilisée pour modifier le temps d'affichage
  int j=2;              // variable utilisée pour afficher plusieurs texte sur la deuxième ligne
  Clear_LCD();          // on efface l'écran du LCD
  last=shift_menu;      // on informe qu'il s'agit du dernier menu sélectionné
  while(1)
    {
   for (j=2 ; j!=4 ;j++)
     {
       for (affichage=0 ; affichage != 2000 ; affichage++) // cette boucle permet a l'utilisateur d'avoir
                                                           // le temps de voir le message affiché
      {
        LCD_cursor(x,y);       // position du curseur pour la première ligne
        LCD_text(texte_1);     // texte affiché sur la première ligne
        LCD_cursor(z,t);       // position du curseur pour les autres lignes
```

```
        LCD_text(texte_j);     // texte affiché pour les autres lignes
        submenu();             // appel le programme du sous menu
        if(shift_menu != last) // lorsque l'on change de menu, on sors de la boucle for
         {
           return;
         }
        }
        Clear_LCD(); // on efface l'écran du LCD

        switch (j)
         {
           case 2 : texte_j = texte_2; // premier texte affiché sur la deuxième ligne
         break;
           case 3 : texte_j = texte_3; // deuxième texte affiché sur la deuxième ligne
         break;

         }
      }
    }
}

/*****************************************************************************************
 *  fonction d'affichage LCD avec deux lignes et trois écrans
*****************************************************************************************/
void affichage_tournant_3(int x,int y,int z,int t,char * texte_1,char * texte_2,char * texte_3,char *
texte_4)
{
  volatile char *texte_j = "Choisissez \0";
  long int affichage=0; // variable utilisée pour modifier le temps d'affichage
  int j=2;              // variable utilisée pour afficher plusieurs texte sur la deuxième ligne
  Clear_LCD();          // on efface l'écran du LCD
  last = shift_menu;    // on informe qu'il s'agit du dernier menu sélectionné
  while(1)
    {
   for (j=2 ; j!=5 ;j++)
     {
     for (affichage=0 ; affichage != 2000 ; affichage++) // cette boucle permet a l'utilisateur d'avoir
                                                         // le temps de voir le message affiché
      {
        LCD_cursor(x,y);       // position du curseur pour la première ligne
        LCD_text(texte_1);     // texte affiché sur la première ligne
        LCD_cursor(z,t);       // position du curseur pour les autres lignes
        LCD_text(texte_j);     // texte affiché pour les autres lignes
        submenu();             // appel le programme du sous menu
        if(shift_menu != last) // lorsque l'on change de menu, on sors de la boucle for
         {
         return;
         }
      }

        Clear_LCD(); // on efface l'écran du LCD
        switch (j)
         {
           case 2 : texte_j = texte_2; // premier texte affiché sur la deuxième ligne
         break;
           case 3 : texte_j = texte_3; // deuxième texte affiché sur la deuxième ligne
         break;
```

```
            case 4 : texte_j = texte_4; // troisième texte affiché sur la deuxième ligne
           break;
          }
       }
    }
}

/*******************************************************************************************
 *  fonction d'affichage LCD avec deux lignes et quatre écrans
*******************************************************************************************/
void affichage_tournant_4(int x,int y,int z,int t,char * texte_1,char * texte_2,char * texte_3,char *
texte_4,char * texte_6)
{
  volatile char *texte_j="Choisissez \0";
  long int affichage=0; // variable utilisée pour modifier le temps d'affichage
  int j=2;              // variable utilisée pour afficher plusieurs texte sur la deuxième ligne
  Clear_LCD();          // on efface l'écran du LCD
  last=shift_menu;      // on informe qu'il s'agit du dernier menu sélectionné
  while(1)
    {
    for (j=2 ; j!=7 ;j++)
      {
      for (affichage=0 ; affichage != 2000 ; affichage++) // cette boucle permet a l'utilisateur d'avoir
                                                          // le temps de voir le message affiché
       {
        LCD_cursor(x,y);   // position du curseur pour la première ligne
        LCD_text(texte_1); // texte affiché sur la première ligne
        LCD_cursor(z,t);   // position du curseur pour les autres lignes
        LCD_text(texte_j); // texte affiché pour les autres lignes
        submenu();         // appel le programme du sous menu
        if(shift_menu != last) // lorsque l'on change de menu, on sors de la boucle for
          {
           return;
          }
        }

        switch (j)
          {
            case 2 : texte_j = texte_2; // premier texte affiché sur la deuxième ligne
              Clear_LCD();              // on efface l'écran du LCD
            break;
            case 3 : texte_j = texte_3; // deuxième texte affiché sur la deuxième ligne
              break;
            case 4 : texte_j = texte_4; // troisième texte affiché sur la deuxième ligne
              break;
            case 6 : texte_j = texte_6; // quatrième texte affiché sur la deuxième ligne
              break;
          }
      }
    }
}
```

```
/*****************************************************************************************
 * fonction d'affichage LCD du menu principal avec deux lignes et trois écrans
*****************************************************************************************/
void main_menu()
{
  affichage_tournant_3(0,0,0,1,menu,mode_numerique,mode_analogique,mode_num_an);
}
```

XII - BIBLIOGRAPHIE

[1] Norme nem 670, http://www.morop.eu/fr/normes/nem670_f.pdf

[2] Norme nem 671, http://www.morop.eu/fr/normes/nem671_f.pdf

[3] DE2 Media Computer, ftp://ftp.altera.com/up/pub/Altera_Material/11.0/Examples/DE2/NiosII_Computer_Systems/DE2_Media_Computer.pdf

[4] DE2_User_Manual, ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf

[5] DE2_115_User_Manual, http://www.element14.com/community/servlet/JiveServlet/previewBody/37305-102-1-219594/Altera.User_Manual.pdf

[6] Analog_Digital, http://astrain.skynetblogs.be/06-analogique-et-digital-dcc/

[7] Le digital pour les nuls, http://letrainpassion.fr/Debut_digit_0.htm

[8] le petit train de vincent, http://lepetitraindevincent.centerblog.net

[9] espacerails.com, http://www.espacerails.com

[10] le protocole DCC (ou protocole NMRA) – version 'BASIC', http://benoit.bouchez.free.fr/digitrain4.htm

[11] Märklin Digital et DCC : les protocoles en détails, http://benoit.bouchez.free.fr/digitrain1.htm

XIII - LEXIQUE

DCC : Digital Command Control ce qui signifie en français contrôle de commande numérique

IrDA : Infrared Data Association (IrDA en anglais) ou assiociation infrarouge de données (en français). L'IrDA permet l'envoi de donnée entre deux appareils par exemple une télécommande infrarouge permet d'envoyer des données de commande à une télévision.

NEM : Norme Européenne du Modélisme

NMRA : National Model Railroad Association (en français : Fédération Nationale de Modélisme Ferroviaire)

PWM : Pulse Width Modulation (PWM en anglais) ou Modulation en Largeur d'Impulsion (MLI en français)