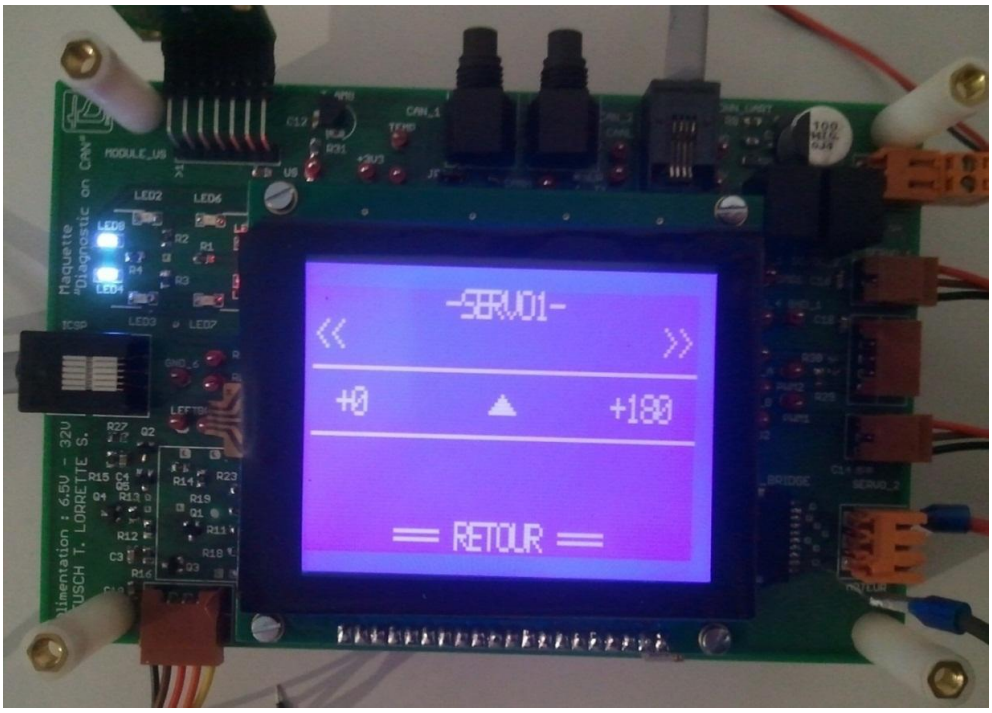
Réalisation d'une maquette de test permettant de simuler du diagnostic véhicule via le protocole ISO 14229 (UDS)

Projet de fin d'étude



Sébastien LORRETTE

Maître de stage : Matthieu ROTH

Du 30 janvier au 3 Aout 2012

Professeur suiveur : Bertrand BOYER

INSA STRASBOURG

engineering

CAHIER DES CHARGES

Le cahier des charges se décompose en 2 parties et englobe le travail à réaliser durant le PRT et durant le PFE.

- **Développement hardware**: Conception et réalisation d'une maquette de développement permettant le développement d'une librairie générique répondant à la norme standard UDS. Plusieurs contraintes doivent être prises en compte :
  - Utilisation d'un microcontrôleur de type DSP (« Digital Signal Processor ») de la famille DsPIC 33FJ.
  - Pilotage d'au moins un servomoteur.
  - Pilotage d'un moteur à courant continu (12V – 1A) par le biais d'un pont en H avec l'utilisation d'un codeur incrémental pour mesurer la vitesse.
  - Mesure de la tension d'alimentation et de la température ambiante,
  - Insertion d'un port de type CAN et d'une liaison série de type RS232,
  - Création d'une interface Homme/Machine à l'aide par exemple de LEDs, d'un LCD.
  - Intégration d'une liaison ICSP pour la programmation du microcontrôleur.
  - Utilisation d'une plage d'alimentation allant de 7 à 30V,
  - Résister à des décharges électrostatiques (ESD) typiques de 8 KV.
  - Prise en compte des contraintes électromagnétiques (CEM).

- **Développement software**: Cette partie se concentre sur 2 axes entre lesquels le protocole UDS est utilisé.
  - Côté Client (carte de développement):
    - Implémentation software au niveau du microcontrôleur des différentes sous-parties de la carte de développement (IHM, liaison I2C, partie moteur, etc.).
    - Implémentation des couches bases sur lesquelles repose le protocole UDS.
    - Implémentation de plusieurs services UDS : « Diagnosticsessionservices », « TesterPresent », etc.
  - Côté Serveur (Logiciel sur ordinateur):
    - Prise en main et modification d'un logiciel réalisé en C# par un ancien stagiaire. Il doit permettre de simuler un outil de diagnostic automobile.
    - Création d'une librairie dynamique contenant les fonctions essentielles aux différentes couches du modèle OSI nécessaires aux protocoles UDS.

L'objectif final est d'être capable d'interroger ou de dialoguer avec la carte de développement depuis l'outil de diagnostic de manière standardisée par l'intermédiaire de différents services UDS.

PROJET DE FIN D'ETUDES

Auteur : Sébastien LORRETTE	Promotion : GE5S
Titre : Réalisation d'une maquette de test permettant de simuler du diagnostic véhicule via le protocole ISO 14229 (UDS).	Soutenance : 20 Septembre 2012
Structure d'accueil : Technology and strategy 4 Avenue de la paix 67000 Strasbourg	
Nb de volume(s) : 1	Nb de pages : 56

Résumé :

Le stage s'est déroulé au sein du laboratoire de Recherche et Développement de la société T&S Engineering géré par Monsieur Matthieu Roth. Ce laboratoire permet à l'entreprise de développer sa capitalisation des connaissances dans le domaine des systèmes embarqués automobile. L'objectif du stage est découpé en 2 parties. La première correspond à la conception d'une carte de développement permettant de simuler le fonctionnement d'un calculateur automobile. La seconde est l'implémentation software de cette carte ainsi que celle de l'outil de diagnostic. La communication de ces 2 entités se produit grâce au protocole de diagnostic UDS.

Mots clés :

CAN, UDS, CANalyzer, langage C et C#, DLL

Traduction :

Title: Design and achievement of development model to simulate an on-board diagnostic via the protocol ISO 14229 (UDS).

My internship took place in the Research and Development laboratory of the T&S Engineering company managed by Mr. Matthieu Roth. This laboratory allows at the company to increase her capitalization of knowledge in the automotive embedded systems. The objective of this internship is cut in two parts. The first correspond with the conception of the development model which allows simulating the working of ECU (Electronic Control Unit). The second is the software implementation of development model and of diagnostic tool. The communication between the both is realized via an on-board diagnostic protocol UDS (Unified Diagnostic Session).

SOMMAIRE

CHAPITRE I : PRESENTATION DE T&S ET DU STAGE

1.1 Présentation générale

Le groupe T&S (Technology & Strategy) a été créé en janvier 2008. Il fait partie de la catégorie des SS2I (Société de services en ingénierie informatique). C'est-à-dire qu'il répond aux besoins d'externalisation des expertises, des services et des projets informatiques des directions informatiques d'entreprises clientes. T&S est composé de 3 entités (cf. figure 1 ci-dessous): une holding ainsi que 2 entités opérationnelles : T&S Information Technology et T&S Engineering. Les 3 associés fondateurs sont Frédéric Bonnet, Jérémie Huss et Jérôme Martinelle.

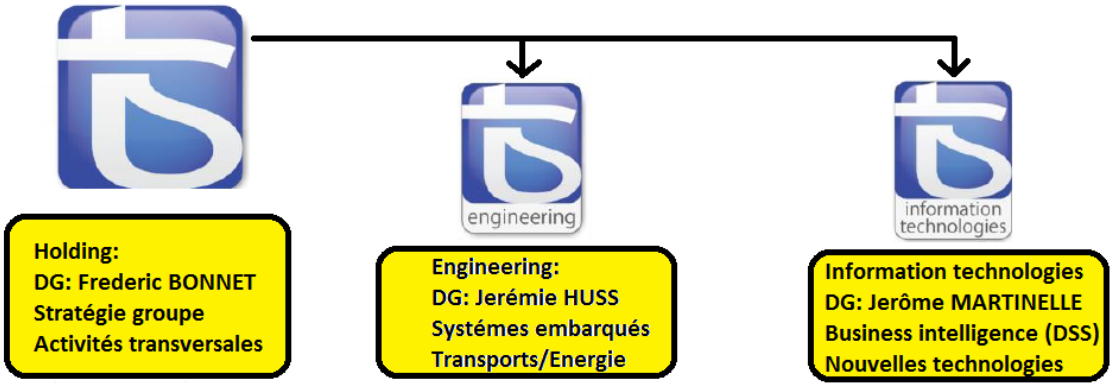


Figure 1 Organigramme organisationnelle de T&S

L'entreprise T&S compte à ce jour plus de 200 collaborateurs. Son objectif est de doubler ses effectifs d'ici à fin 2012. Pour réaliser cela, elle peut compter sur une forte croissance du nombre de ses clients depuis 2008. Le chiffre d'affaires cumulé en 2011 s'élevait à 21 millions d'euros (cf. figure 2 ci-dessous).

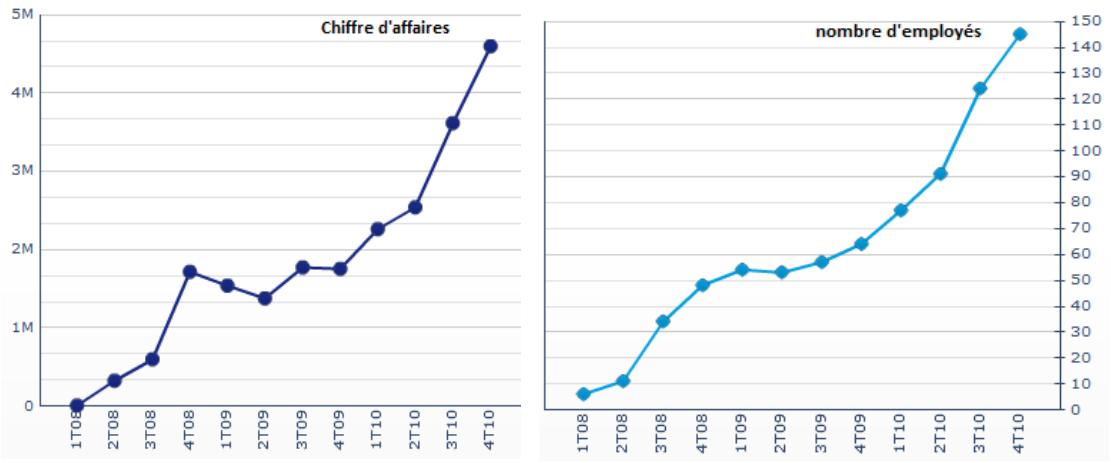


Figure 2 Evolution du chiffre d'affaires et du nombre d'employés

1.2 T&S Holding

Cette entité assure les fonctions de gestion transversale indispensable au fonctionnement des deux sociétés filles : T&S engineering et T&S information technologies. Elle gère, entre autres, l'administration centralisée, les aspects financiers, les ressources humaines et la gestion du personnel. L'entité Holding assure le développement stratégique du groupe en jouant son rôle de société mère. Elle met en place une structure décisionnaire et organisationnelle afin que l'entreprise se développe dans le domaine franco-allemand.

1.3 T&S Information technologies

T&S Information technologies regroupe les techniques utilisées dans le traitement et la transmission des informations, principalement de l'informatique, de l'Internet et des télécommunications. Le but de cette société est de devenir le leader de la Business Intelligence et des solutions applicatives orientées Microsoft dans l'Est de la France. 2 parties la composent :

- « Application Development and Solutions » (Développement d'applications et de solutions) qui permet d'offrir des prestations informatiques comprenant la vente de licence, du conseil et de l'hébergement de données.
- « Decision Support System » (Système interaction d'aide à la décision) qui désigne les moyens, les outils et les méthodes qui permettront de collecter, consolider, modéliser et restituer les données d'une entreprise en vue d'offrir une aide à la décision et de permettre aux responsables de la stratégie d'entreprise d'avoir une vue d'ensemble de l'activité traitée.

1.4 T&S Engineering

Cette entité est spécialisée dans le domaine des systèmes embarqués principalement dans le monde des transports (automobile, aéronautique, ferroviaire) mais également de l'énergie (cf. figure 3 ci-dessous). Elle se caractérise également par un fort positionnement franco-allemand car 80% de son activité se déroule outre-rhin. Elle compte parmi ses partenaires des acteurs majeurs de l'industrie automobile ainsi que des équipementiers et fournisseurs de premier rang tels que : Bosch, BMW, Porsche, Valeo, Hager, Socomec, etc.

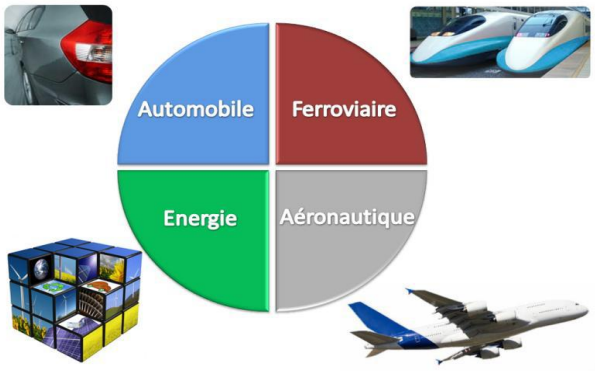


Figure 3 Domaine d'activités de T&S engineering

Le stage s'est déroulé au sein du laboratoire de T&S engineering qui intervient entre autres dans les domaines des transports ou de l'énergie. Le laboratoire est spécialisé dans l'étude, la conception et la réalisation de systèmes embarqués.

1.5 Déroulement du stage

Le stage s'est déroulé entre le 30 janvier 2012 et le 3 aout 2012. Cependant le projet a lui commencé plus tôt durant le 1er semestre de la 5eme année à l'INSA de Strasbourg. Un module était consacré à la réalisation de projet en association avec des entreprises de la région. Avec un autre étudiant du génie électrique Thomas HEUSCH, nous devions concevoir et réaliser une maquette de développement permettant le déploiement d'une librairie générique répondant à la norme standard UDS. Ce projet est en réalité la première partie du PFE consacré au développement hardware et à la partie recherche et documentation. C'est pourquoi ces parties n'apparaîssent pas dans le planning (cf. figure 4 ci-dessous).

Nom de la tâche	Durée	Début	Fin	Prédécesse
Développement hardware de la carte	20 jours?	Lun 30/01/12	Ven 24/02/12	
Développement software fonctions peripheriques	15 jours?	Lun 27/02/12	Ven 16/03/12	1
Soudure des composants	5 jours?	Lun 19/03/12	Ven 23/03/12	2
Developpement IHM et correctifs	15 jours?	Lun 26/03/12	Ven 13/04/12	3
Mise en place fonctions peripheriques	15 jours?	Lun 16/04/12	Ven 04/05/12	4
Etude documentation UDS	5 jours?	Lun 07/05/12	Ven 11/05/12	5
Développement UDS côté serveur	60 jours?	Lun 14/05/12	Ven 03/08/12	6
Développement UDS côté client - C# DLL	35 jours?	Lun 04/06/12	Ven 20/07/12	

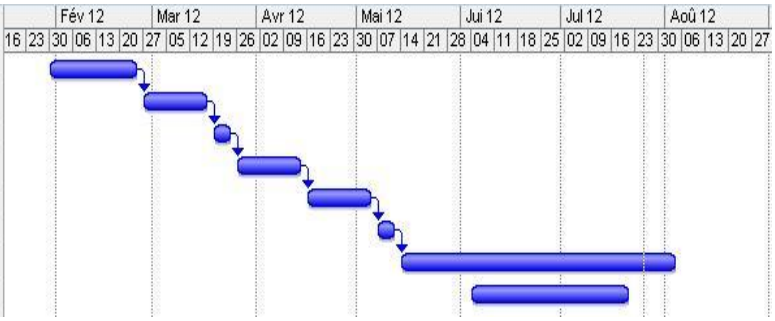


Figure 4 Planning du PFE

CHAPITRE II CONCEPTION DE LA CARTE DE DEVELOPPEMENT

2.1 Introduction

L'objectif est la réalisation et la conception de la maquette de développement permettant le développement d'une librairie répondant à la norme standard UDS (Unified Diagnostic Services). Elle doit contenir tous les éléments définis dans le cahier des charges tels que :

- Utilisation d'un microcontrôleur de type DSP (« Digital Signal Processor ») de la famille DsPIC 33FJ.
- Pilotage d'au moins un servomoteur.
- Pilotage d'un moteur à courant continu (12V – 1A) par le biais d'un pont en H avec l'utilisation d'un codeur incrémental pour mesurer la vitesse.
- Mesure de la tension d'alimentation et de la température ambiante,
- Insertion d'un port de type CAN et d'une liaison série de type RS232,
- Création d'une interface Homme/Machine à l'aide par exemple de LEDs, d'un LCD.
- Intégration d'une liaison ICSP pour la programmation du microcontrôleur.
- Utilisation d'une plage d'alimentation allant de 7 à 30V,
- Résister à des décharges électrostatiques (ESD) typiques de 8 KV.
- Prise en compte des contraintes électromagnétiques (CEM).

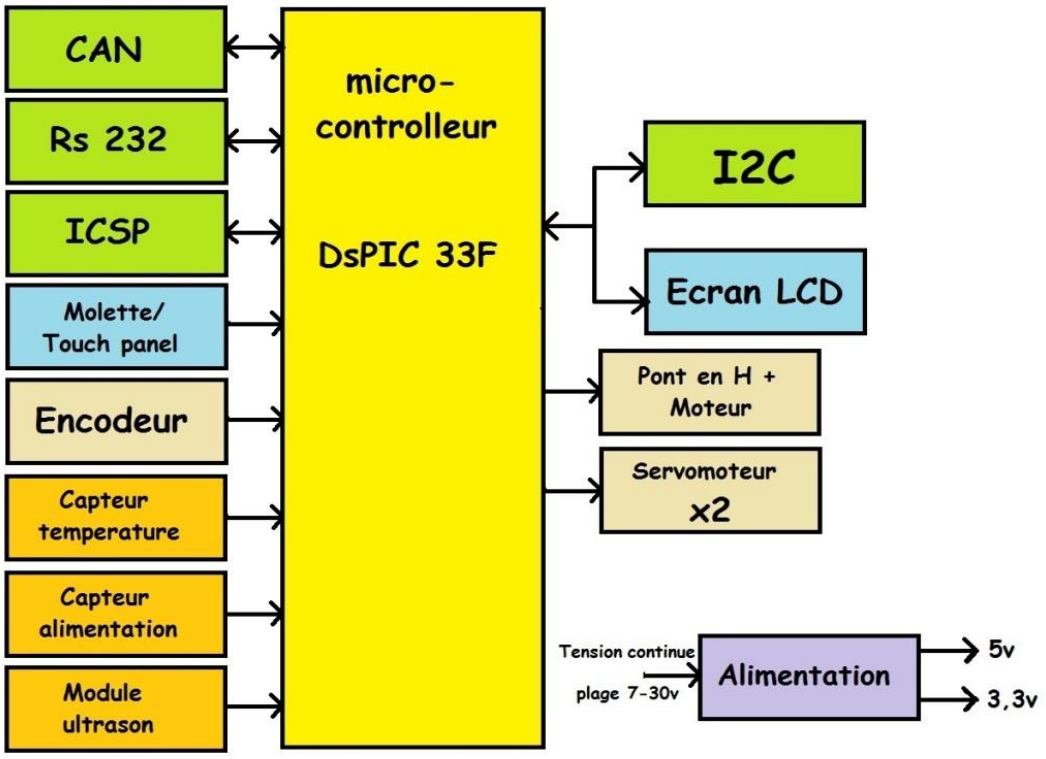


Figure 5 Schéma bloc des fonctions de la carte de développement

2.2 Choix des composants

2.2.1 Choix du microcontrôleur

Le 1er choix à réaliser est celui du microcontrôleur. En effet, c'est le cœur de la carte vers lequel tous les périphériques vont converger. Pour cela, plusieurs paramètres sont à prendre à compte. Le premier et principal est le nombre d'entrées/sorties utilisées qu'il faut recenser :

- Bus CAN : **2 pins** (CANH et CANL),
- Liaison RS232 : **2 pins** (RX1 et TX1),
- Liaison ICSP avec bus RS232 : **3 pins** (MCLR, RX2, TX2),
- Ecran LCD : **7 pins** (SCL, SDA, RW, RS, E, CS1, CS2 et RETRO),
- Liaison I2C : **1 pin** (RP + SCL et SDA commun au LCD)
- Molette et Touchpanel : **4 pins** (READX, READY, LEFT et BOTTOM),
- Servomoteurs : **2 pins** (SMO1 et SMO2),
- Pont en H : **3 pins** (PWM1, PWM2 et ENABLE),
- Encodeur : **2 pins** (ENCOD_A et ENCOD_B (HEDS-5500) : sortie collecteur ouvert),
- Capteur de température : **1 pin** (TEMP),
- Mesure de la tension d'alimentation : **1 pin** (ADC_ALIM),
- Quartz : **2 pins** (OSC1 et OSC2),
- LEDs : **4 pins**.

Le choix s'est donc porté sur un microcontrôleur avec **44** pins. Dans cette gamme, nous avons choisi le **DsPIC 33 FJ128MC804** dont voici les caractéristiques principales :

- Nombre de pins : **44**.
- RAM de programme : **128 Ko**.
- RAM de données : **16 Ko**.
- Nombre de pins remappables : **26**.

2.2.2 Alimentation de la carte

La tension d'alimentation en entrée de la carte peut varier entre 7 et 30V continu. Les composants de la maquette sont eux alimentés soit en 5V soit en 3,3V. Il faut donc réaliser une adaptation en tension. La première pour passer du 7-30V au 5V et la seconde pour passer du 5V au 3,3V.

- La conversion 7-30V vers 5V se fait à l'aide d'une alimentation à découpage Recom R-78B5.0-1.0L. La tension d'entrée peut varier de 6,5V à 32V. Le courant de sortie est de 1A. Cela risque d'être insuffisant. En effet, 2 servomoteurs sont connectés à la carte et ils peuvent consommer jusqu'à 800mA chacun lors des déplacements. Pour régler ce problème, nous mettons donc deux alimentations en parallèle.
- Le convertisseur 5V vers 3,3V se situe sur la platine du microcontrôleur. Sa référence est LT1521. C'est un régulateur avec un courant de sortie allant jusqu'à 300 mA.

2.2.3 L'interface homme-machine

L'interface homme machine est composée de plusieurs éléments :

- Tout d'abord pour permettre la visualisation de différents paramètres nous utilisons un écran LCD de Vishay LCD-128G064I. Il utilise des drivers de type KS0108. Il possède un écran de 128x64 pixels. Nous utilisons le mode 8 bits du LCD. Les pins SCL, SDA, RW, E, CS1, CS2 et RS sont reliés au PIC. Un convertisseur série/parallèle (MCP23009) est inséré entre les 8 fils de données du LCD et le microcontrôleur. Nous multiplexons le bus de données à l'aide d'une liaison I2C. Elle nous permet d'utiliser seulement 2 broches du microcontrôleur : SDA et SCL. Une régulation est réalisée au niveau du rétro éclairage à l'aide d'un transistor Darlington.

Figure 6 Ecran LCD et Touchpanel

- Pour permettre à l'utilisateur de commander la carte, une tablette tactile est insérée par-dessus l'écran LCD. Le microcontrôleur récupère la position sur X et sur Y via 2 ADCs internes lors d'un appui sur la dalle. Il est ensuite assez facile de gérer les résultats de cette appui.
- Une molette multi-tour avec bouton poussoir peut être ajoutée en remplacement de la dalle tactile. Les empreintes coïncident avec celle-ci. Elle remplit les mêmes fonctions et peut donc être mise en remplacement de la dalle selon la préférence des utilisateurs.
- 8 LEDs sur la carte permettent de simuler les feux d'une voiture. Elles sont groupées par 2: les feux avant, stop, clignotants droit et clignotants gauche.

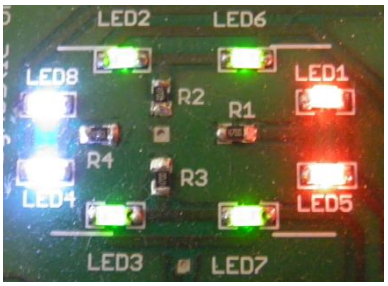


Figure 7 LEDs sur la carte

2.2.4 Partie moteur

La partie « moteur » est composée de plusieurs éléments. Le plus important est le pont en H (cf. figure 8 ci-dessous). Il fait varier la vitesse de la machine à courant continu et la pilote dans les 2 sens. Il est connecté avec 2 pattes du microcontrôleur : la première est une PWM qui permet la variation de vitesse et la seconde est à « 1 » ou à « 0 » permettant l'activation du sens de rotation.

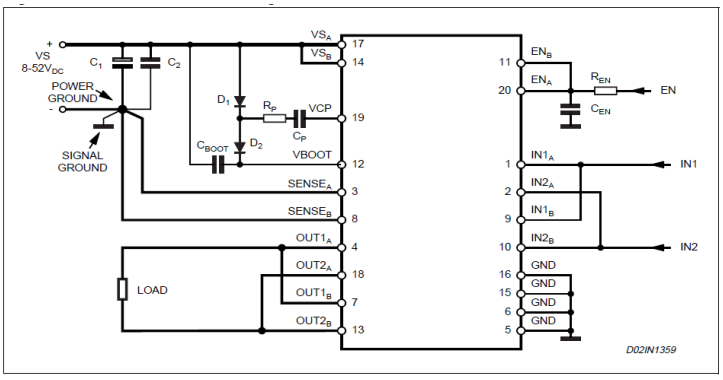


Figure 8 Pont en H

Le L6205 de STMicroelectronic est composé de 2 ponts en H. Le montage retenu utilise les 2 ponts en H en parallèle et permet d'augmenter le courant de sortie et également de réduire la dissipation dans le composant.

2.2.5 Capteurs

3 capteurs sont présents sur la carte :

- **Un capteur d'alimentation** : c'est un simple pont diviseur qui permet de faire correspondre la tension maximale d'alimentation de la carte avec la tension d'alimentation du microcontrôleur (3.3V). Il suffit ensuite de réaliser une conversion en numérique de cette valeur et de traiter cette information.
- **Un capteur de température** : C'est un capteur analogique linéaire. C'est-à-dire que la tension en sortie est proportionnelle avec la température ambiante. Un module ADC interne au DsPIC est également nécessaire.
- **Un transducteur à ultrasons** (cf. figure 9 ci-contre): Il est capable de mesurer des distances allant de 0 à 6,45m. Il possède une sortie PWM. La largeur d'impulsion est proportionnelle à la distance de l'obstacle. Pour gérer cela, une entrée de capture PWM est disponible dans le microcontrôleur.

Figure 9 module ultrasons

2.2.6 Les liaisons séries

Plusieurs périphériques sont connectés au microcontrôleur :

- **La liaison I2C**: Elle permet de multiplexer les données pour l'écran LCD et d'économiser des pattes sur le DsPIC (2 pins nécessaires SCL et SDA). Un connecteur est également disponible pour réaliser une possible extension du bus en ajoutant un périphérique communiquant par I2C.
- **Les liaisons séries**: 2 liaisons séries sont présentes sur la carte. La première est une liaison RS232. Elles passent par un MAX233 qui adapte les tensions de l'UART du DsPIC en tension -/+12V. La seconde est une liaison série avec les niveaux de tension du microcontrôleur 0-3.3V.
- **La liaison ICSP** : Elle permet de programmer le microcontrôleur via un PICKIT2 ou PICKIT3 par exemple. Cette liaison possède 5 fils : le MCLR, l'alimentation, la masse, l'entrée de l'horloge et l'entrée de données. Les entrées « clock » et « data » sont superposées avec la seconde liaison série.
- **La liaison CAN** : Elle sert de support au protocole UDS. Un transceiver est nécessaire pour la faire fonctionner avec le microcontrôleur en effet le DsPIC ne possède pas de transceiver interne. Il a plusieurs rôles : tout d'abord il permet de convertir les signaux CAN_TX et CAN_RX en signaux différentiels CAN_H et CAN_L, ensuite il protège la liaison contre des décharges électrostatiques pouvant aller jusqu'à 6KV. D'autres composants permettent de protéger la ligne (cf. figure 10) : Une self de mode commun permet d'augmenter le CMRR et

2 TVS (Transcient Voltage Suppressors ou diode à avalanche) permettent la protection contre les pics d'énergies. Enfin une résistance de 120 Ω est placée en bout de ligne.

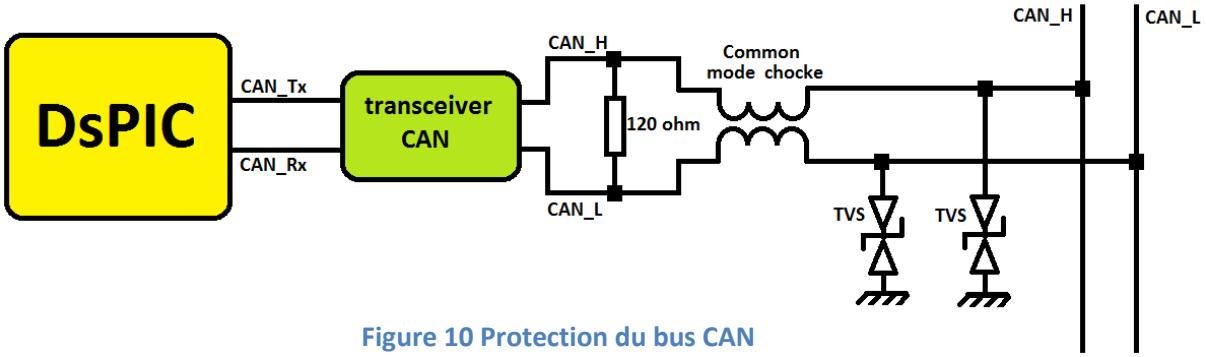


Figure 10 Protection du bus CAN

2.3 Conception de la carte

Pour concevoir la carte, nous avons utilisé le logiciel EAGLE de CAD Soft. C'est un logiciel de conception assisté par ordinateur de circuits imprimés. L'objectif est la création d'une carte avec une taille maximum au format européen (100x160 mm). Toutefois, après plusieurs modifications, nous avons réussi à diminuer la taille de la carte pour la ramener à 96.2x148.8 mm.

Pour cela, nous disposions de quatre couches : deux couches de signaux, une couche pour les alimentations et un plan de masse.

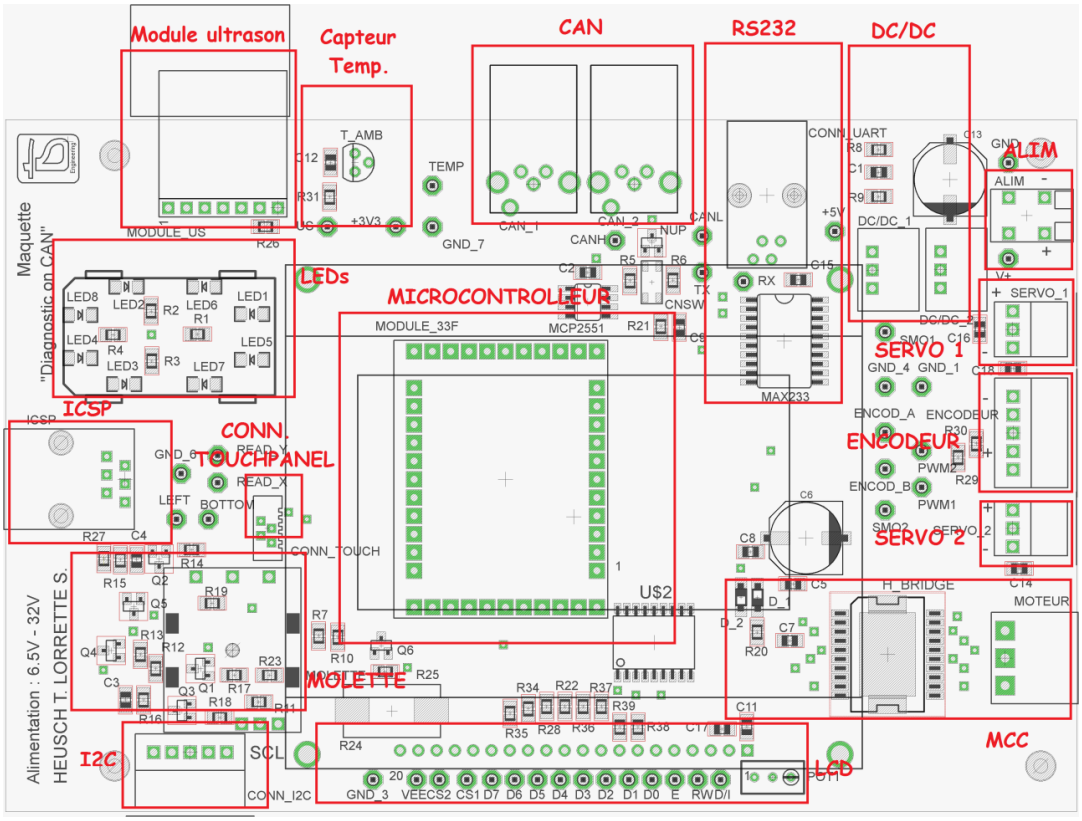


Figure 11 Couche Top et Bottom du typon

Au centre de la maquette « Diagnostic on CAN » se trouve le module 33F. Ce module a été conçu dans le cas où le DsPIC rencontrerait un problème. Il devient plus facile de le remplacer. Le module 33F comprend le dsPIC33FJ128MC804, le quartz ainsi qu'un régulateur 5V vers 3,3V.

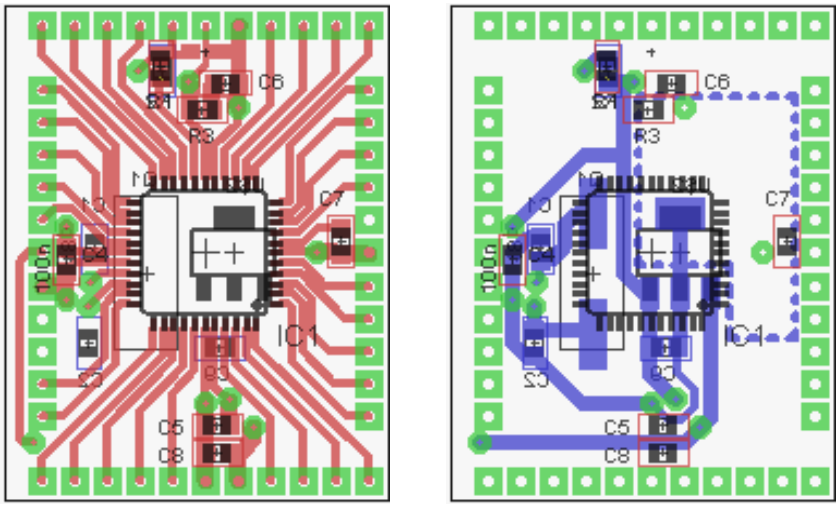


Figure 12 Couche TOP et BOTTOM du module 33F

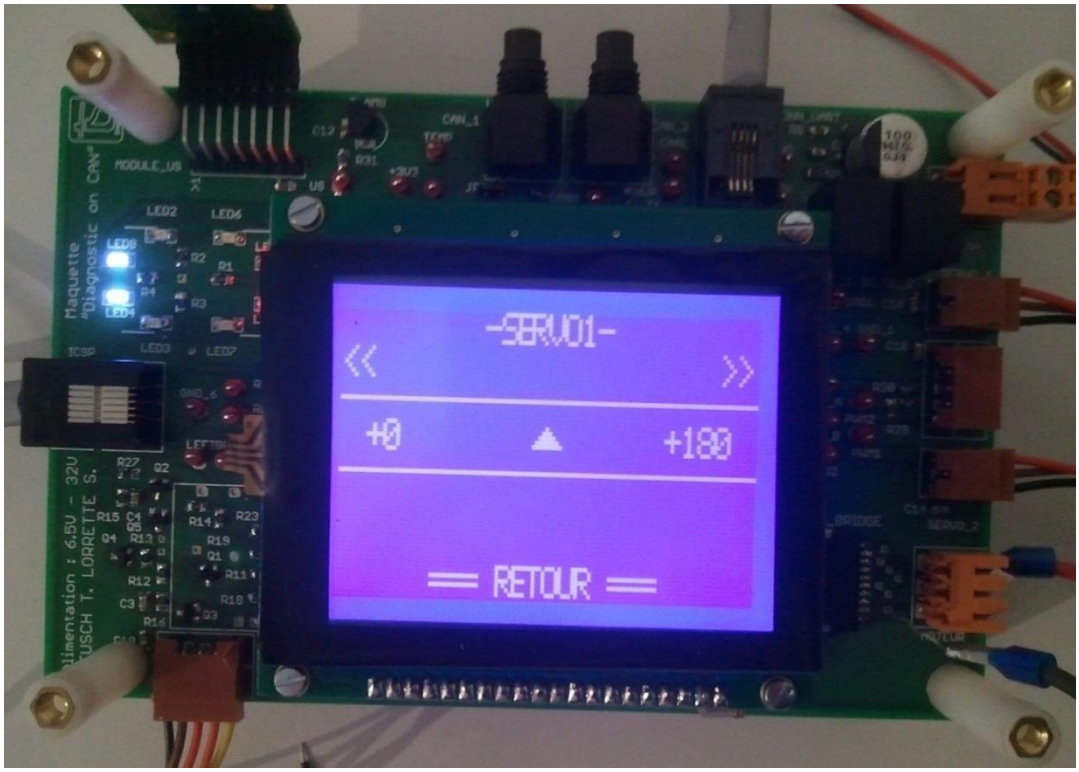


Figure 13 Carte finalisée

CHAPITRE III : DEVELOPPEMENT SOFTWARE DE LA MAQUETTE

3.1 Introduction

Dans cette partie, nous allons parler du développement software des différentes fonctions de la carte hormis la partie protocole CAN et UDS. C'est-à-dire principalement le développement de l'interface homme/machine, de la partie moteur et le traitement des signaux venant des différents capteurs de la maquette.

3.2 Conception de l'IHM

L'interface homme/machine est composée principalement de 2 éléments. Le premier l'écran LCD permet une communication dans le sens « logiciel -> utilisateurs », le second, la dalle tactile permet un échange dans le sens contraire « utilisateurs -> logiciel ». Pour créer le menu, il est nécessaire de faire interagir ces 2 éléments ensembles.

3.2.1 Correction Hardware et Software

Lors de la conception hardware de la carte, une erreur est intervenue. En effet, le brochage de l'écran LCD n'était pas correct. Il a donc était nécessaire de faire des modifications hardware et software pour faire correspondre les bonnes pins du microcontrôleur avec celles de l'écran LCD. Certaines modifications étaient sans conséquence grâce à l'utilisation des pins remappables, d'autres plus gênantes comme la pin E sur laquelle, on génère l'impulsion d'écriture de l'écran déplacée sur le bus I2C. Cette durée d'impulsion devenait par conséquence plus longue, il a donc fallu augmenter la vitesse du bus I2C au maximum, c'est à dire à plus de 1 Mbits/s.

3.2.2 Menu

Plusieurs fichiers sont consacrés à la réalisation de l'ensemble du menu:

- **LCD.c**: Fonctions permettant d'écrire du texte ou des logos sur l'écran LCD.
- **Touchpanel.c**: Fonctions permettant l'acquisition des coordonnées d'un appui sur la dalle tactile.
- **Menu.c** : Ce fichier fait le lien entre les 2 premiers cités. On peut ainsi créer différentes pages liées entre elles avec des boutons virtuelles.
- **I2C.c**: Permet l'écriture de données sur le bus I2C et donc sur l'écran LCD.
- **Font5x7.h**: Contient le tableau avec les caractères ASCII.
- **Logo.h**: Contient le logo de T&S.

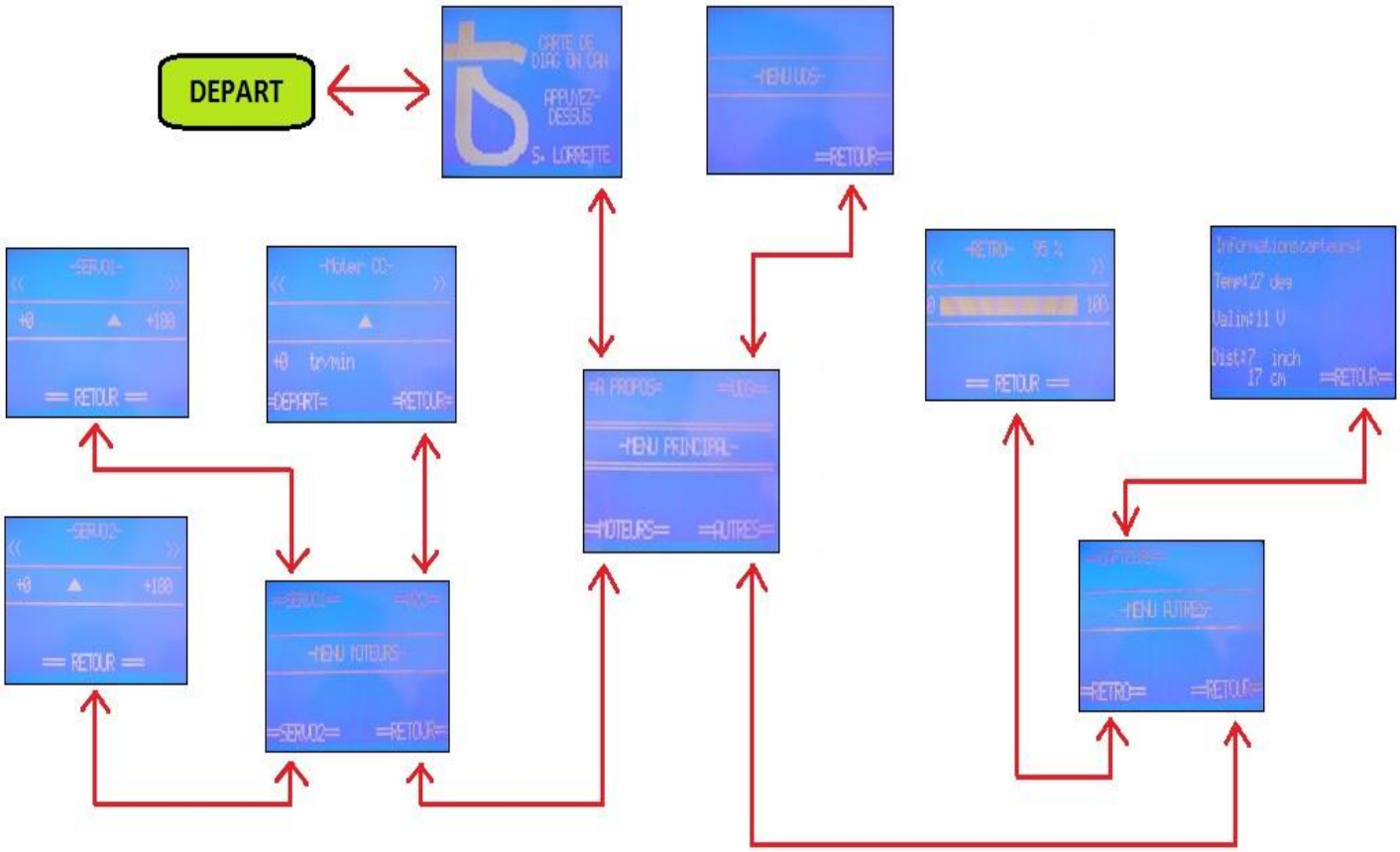


Figure 14 Organigramme du menu

Le menu contient en tout 11 pages différentes (cf. figure 14 ci-dessus). Lors du démarrage de la carte, l'utilisateur visualise un écran de présentation avec le logo de T&S et le nom du projet, il peut passer à l'écran principal en appuyant sur l'écran. La page principal permet d'accéder aux différents sous-menus :

- Sous-menu « **MOTEURS** » : Il contient les pages pour déplacer les 2 servomoteurs et mettre en route la machine à courant continu.
- Sous-menu « **AUTRES** » : Il permet l'accès au réglage du rétro éclairage entre 0 et 100% avec un pas de 5% et l'état des capteurs.
- Sous-menu « **UDS** » : La page est actuellement vide et existe dans l'optique de l'extension du menu.
- « **A PROPOS** » : Retourne à la page de présentation du départ.

3.2.3 Gestion de l'affichage

L'écran LCD est inversé par rapport au sens de lecture car la nappe de la dalle tactile est trop courte pour mettre l'écran dans le bon sens directement. Plusieurs opérations sont nécessaires pour respecter le bon sens de lecture:

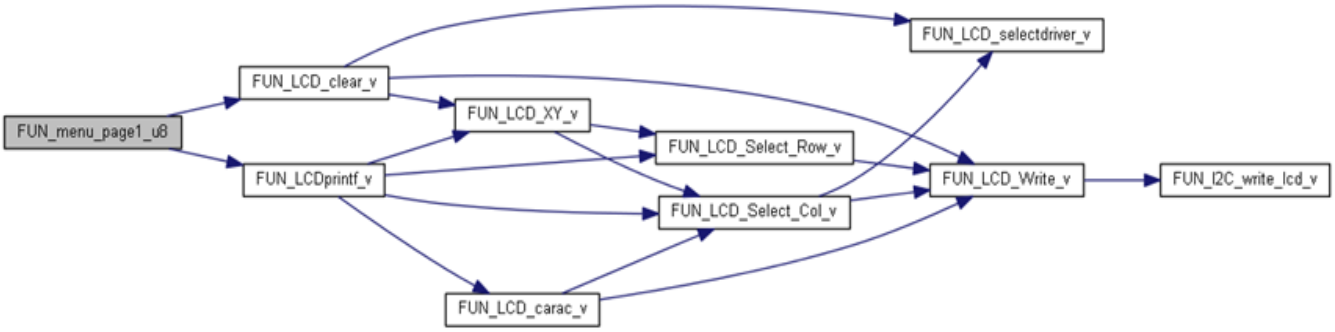


Figure 15 Organigramme de création de la page 1

- Affichage des caractères à l'envers grâce à La fonction « FUN_LCD_carac_v » dans LCD.c.
- Inversement de l'adressage des lignes: la ligne 0 correspond à la ligne7, la 1 à la 6, etc. grâce à « **FUN_LCD_Select_Row_v** ».
- Inversement de l'adressage des colonnes: la colonne 0 correspond à la colonne 63, la 1 à la 62, etc. grâce à « **FUN_LCD_Select_Col_v** ».
- La colonne s'incrémente automatiquement à chaque opération d'écriture. Il est donc nécessaire de soustraire l'adresse de 2 pour aller dans le bon sens. Cette opération se réalise lors de l'utilisation de « **FUN_LCDprintf_v** ».
- La fonction « **FUN_LCDprintf_v** » gère aussi automatiquement le passage du driver gauche au driver droit de l'écran LCD.

3.2.4 Gestion de la dalle tactile

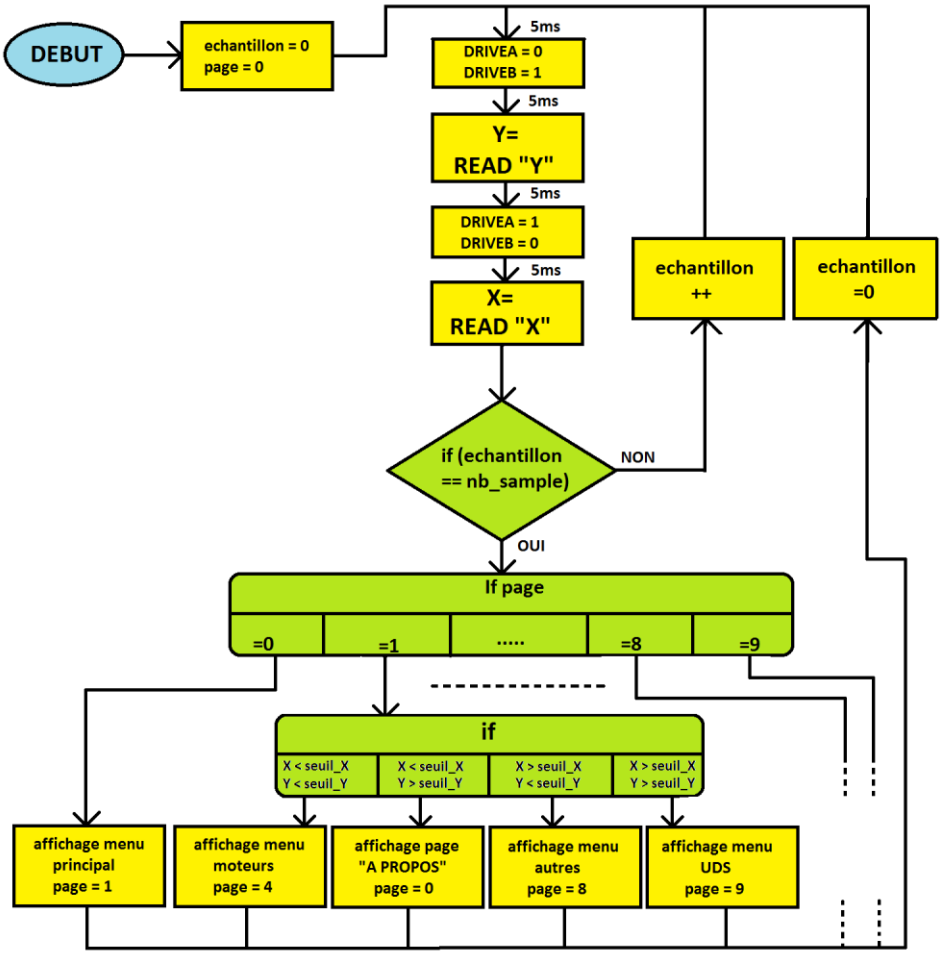


Figure 15 Gestion de l'appuie sur la dalle tactile

La dalle tactile possède 2 entrées: DRIVEA, DRIVEB et 2 sorties: READ_X, READ_Y. Une boucle infinie se produit (cf. figure 15) en continu et se décompose comme suit :

- DriveB est mis à « 1 » puis 5 ms plus tard la coordonnée sur Y est lue. On attend 5 ms de plus et on recommence l'opération avec DriveA et la coordonnée sur X est lue également. On faits ceci plusieurs fois pour stocker plusieurs échantillons et avoir une valeur affinée. On récupère les valeurs à l'aide de 2 convertisseurs analogique/numérique interne au microcontrôleur.
- A partir de cet instant, il est possible de déterminer s'il y a eu appui sur la dalle tactile. Selon la page, les boutons auront des fonctions différentes. Par exemple, un appui sur la page 0 aura pour conséquence l'apparition de la page 1. Celle-ci contenant le menu ouvrira 4 pages différentes selon les coordonnées de l'appuie. Un appuie en bas à gauche par exemple fera apparaître la page 4 avec le menu moteurs.

3.2.5 Utilisation du bus I2C

Le bus I2C nous permet de multiplexer une partie des liaisons avec l'écran LCD. Pour réaliser cela, un composant intermédiaire est inséré entre la liaison I2C et l'écran LCD : le MCP23009. Il réalise une conversion série/parallèle. Il est constitué de plusieurs registres à configurer. L'opération d'écriture suit toujours la même procédure (cf. figure 16 ci-dessous) dans le cas d'une écriture séquentielle utilisée exclusivement dans ce projet :

- Un bit de START est envoyé par le maitre, c'est-à-dire par le DsPIC, pour ouvrir la trame.
- L'adresse du MCP23009 est ensuite envoyée : les 4 bits de poids forts sont fixés à 0b100, les 3 bits suivants sont fixés par la tension analogique mise sur l'entrée « ADDR » dans notre cas 0b111 et enfin le dernier bit définit la lecture (0b1) ou l'écriture (0b0). L'esclave répond en envoyant un « acknowledge ».
- En troisième position, c'est l'adresse du registre dans lequel on veut écrire qui est transmise avec toujours un « acknowledge » en réponse.
- Et enfin pour finir, les données sont envoyées sur 8 bits. L'esclave envoie un « acknowledge » pour conclure cet échange.

Les opérations de lecture ne sont pas utilisées car l'écran LCD est utilisé simplement en mode écriture.

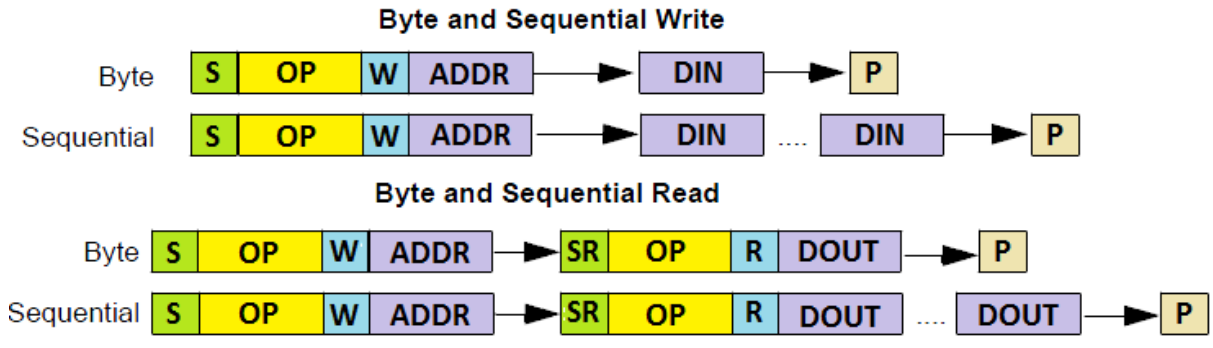


Figure 16 Lecture et écriture I2C du MCP23009

Plusieurs registres sont utilisés lors de la transmission de données vers l'écran LCD (cf. figure 18 ci-dessous):

- A l'initialisation le registre « **IODIR** » (0x00) est mis à 0x00 permettant d'avoir les 8 broches de données en sortie.
- Au même moment le registre « **GPPU** » (0x06) est mis à 0xFF, il active les pull-up sur les 8 broches de données.
- Le 3eme registre nécessaire est « **OLAT** » (0x0A). La valeur de ce registre se retrouve sur les broches de données.

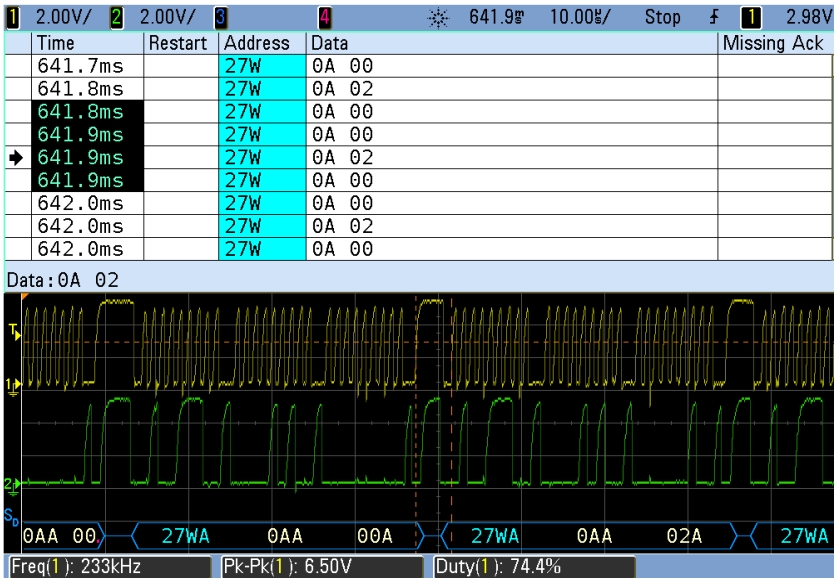


Figure 17 Capture d'une trame I2C vers l'écran LCD

3.2.6 Rétro éclairage

Il est géré depuis l'écran LCD lui-même. La cathode du rétro éclairage est reliée au microcontrôleur via un transistor et l'anode est reliée à l'alimentation de la carte. Un transistor Darlington de type NPN permet de réaliser une amplification en courant nécessaire au rétro éclairage (cf. figure 18 ci-dessous). Pour faire varier l'intensité de la LED de rétro éclairage, un module PWM du DsPIC est utilisé. Il fait varier la tension sur la base entre 0 et 3.3V et régule ainsi la luminosité de l'écran. L'utilisateur paramètre la luminosité depuis le menu où une page est dédiée à cette fonction.

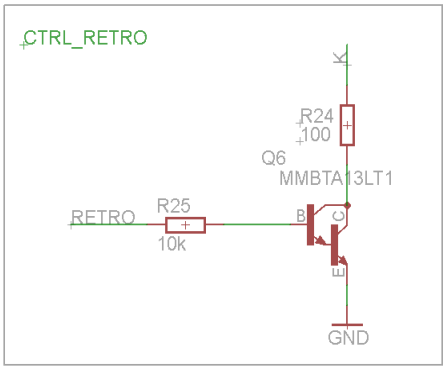


Figure 18 Schéma structurel du rétroéclairage

3.3 Commande des différents capteurs

3.3.1 Capteur à ultrasons

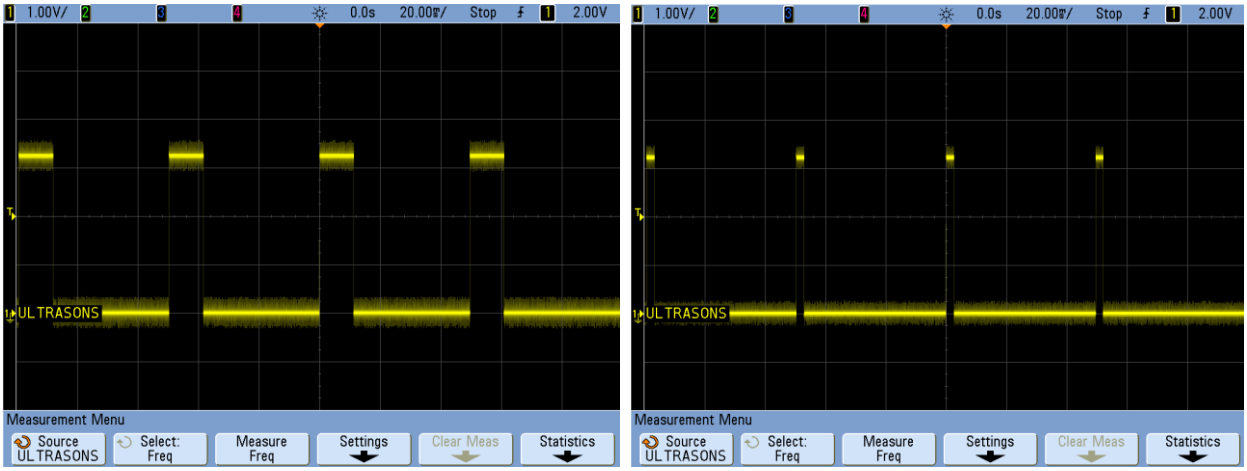


Figure 19 PWM venant du transducteur à ultrasons

Le transducteur à ultrasons possède une sortie PWM avec laquelle la largeur d'impulsion varie en fonction de la distance de l'obstacle (cf. figure 19 ci-dessus). Le microcontrôleur récupère ce signal et le traite à l'aide de son module « Input capture ». Ce module fonctionne en association avec un timer.

- Lors de chaque front du signal, la valeur du compteur associé au timer est stockée dans une FIFO.
- Une soustraction est réalisé entre la valeur du compteur au dernier front descendant et celle lors du dernier front montant. Cela permet d'avoir une représentation temporelle de la distance.
- A partir de là, il ne reste plus que des calculs. La 1ere conversion permet d'obtenir la durée entre les 2 fronts. La seconde conversion donne la valeur en inch sachant que 147 us représentent 1 inch. Enfin la dernière conversion donne la valeur en centimètre : 1 inch<->2,54 cm.
- Les distances sont ensuite affichées sur l'écran LCD et rafraichies régulièrement.

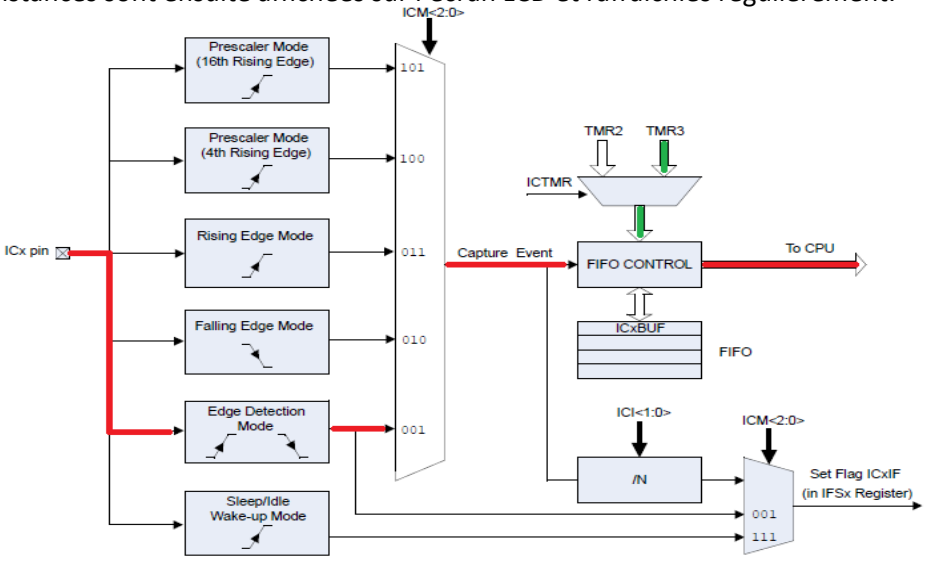


Figure 20 Schéma bloc du module "Input capture"

3.3.2 Capteur de températures

Le DsPIC récupère la tension analogique sur une de ses bornes représentant la température ambiante. Cela impose l'utilisation d'un convertisseur analogique/numérique. Pour réaliser cela, on utilise une librairie déjà faite. Elle a été conçue par M. Roth sur un ancien projet pour T&S et utilise le même type de capteur de température. Il suffit de donner les paramètres de configuration dans le programme :

- Le facteur de conversion ou quantum : Rapport entre la valeur pleine échelle en entrée du convertisseur (3.3V) et le nombre de valeurs possibles en sorties (2^10=1024).
- La température suivant la tension analogique du CNA suit une fonction affine : Tamb=a*V+b. les 2 derniers paramètres à définir sont donc le coefficient directeur (0.01) et l'offset (0.5).

La valeur de la température est récupérée et affichée de la même manière que le capteur à ultrasons à intervalle régulier.

3.3.3 Capteur d'alimentation

Le capteur d'alimentation est composé de 2 éléments :

- Un pont diviseur avec la tension d'alimentation en référence dimensionné pour avoir 3.3v sur le microcontrôleur dans le cas d'une l'alimentation maximale.
- Un convertisseur analogique/numérique à l'intérieur du DsPIC commandé grâce à un module écrit en C par M. Roth. Les paramètres à fournir sont le quantum, la valeur des résistances du pont, et l'unité dans laquelle on souhaite récupérer la donnée (en V, en 1/10 de V ou en 1/100 de V). La tension est affichée sur l'écran LCD et rafraichie régulièrement.

Figure 21 Page du menu concernant les capteurs

CHAPITRE IV : PRESENTATION DE L'UDS

4.1 Introduction

Cette partie est consacrée à l'explication générale du fonctionnement du protocole UDS. Nous verrons en quoi ce protocole de diagnostic repose sur le modèle OSI et nous décrirons les différentes couches qu'il utilise, par exemple, celles occupés par le bus CAN sur lequel il repose.

4.2 Présentation générale du modèle OSI

Le début des années 1980 a été marqué par une croissance exceptionnelle du nombre et de la taille des réseaux. En raison de cette extension rapide, les sociétés installant des réseaux commencèrent à rencontrer des problèmes tout comme il est difficile pour les gens qui ne parlent pas la même langue de communiquer. Les technologies réseau qui suivaient strictement des règles propriétaires ne pouvaient pas communiquer avec des technologies qui respectaient des règles propriétaires différentes. Le modèle de référence OSI (Open System Interconnection) publié en 1984 proposa donc aux fournisseurs un ensemble de normes assurant une compatibilité et une interopérabilité accrues entre divers types de technologies réseau. Il décrit l'architecture en 7 couches logicielles présentant chacune des interfaces standard pour communiquer entre elles.

	Modele OSI	
7	Application	Interface vers les programmes utilisateurs
6	Présentation	Conversion de formats
5	Session	Synchronisation – Reprise sur erreur
4	Transport	Fiabilité – Qualité de services
3	Réseau	Echange des données via des nœuds intermédiaires - Routage
2	Liaison	Accès entre nœuds voisins – Gestion de l'accès au médium – détection et correction des erreurs
1	Physique	Adaptation des informations au médium de transmission

Figure 22 les 7 couches du modèle OSI

4.3 L'UDS dans le modèle OSI

Le protocole UDS (Unified Diagnostic Services) repose sur le modèle OSI. 5 couches y sont implémentées.

- Les couches 1 (physique) et 2 (liaison) : ISO 11898 1-2-3 Protocole CAN
- Les couches 3 (Réseau) et 4 (Transport) : ISO 15765-2 Diagnostic on CAN
- La couche 7 (Application) : ISO 15765-3
- Une couche supplémentaire assimilée au niveau 8 : protocole UDS ISO 14229

4.4 Le protocole CAN

4.4.1 Présentation générale

Le protocole CAN (Control Area Network) fut créé par l'entreprise Bosch, équipementier automobile, en association avec l'université de Karlsruhe dans le milieu des années 80. Il est né du besoin de trouver une solution de communication série dans les véhicules automobiles, qui intégraient de plus en plus de commandes électroniques. Tous les organes de commande des véhicules échangeaient les données par l'intermédiaire de lignes séries point à point dédiées (jusqu'à 2 km de câbles par véhicule). Le CAN permet l'utilisation de la technique du multiplexage qui permet le raccordement à un même câble d'un grand nombre de calculateur. Cette approche limite le nombre de câbles et donc le cout global de l'installation (cf. figure 23 ci-dessous). Il permet également de conserver un très haut niveau d'intégrité au niveau des données dans des milieux bruyants tels que les milieux industriels. Il couvre les 2 plus basses couches du modèle OSI : la couche physique et la couche liaison.

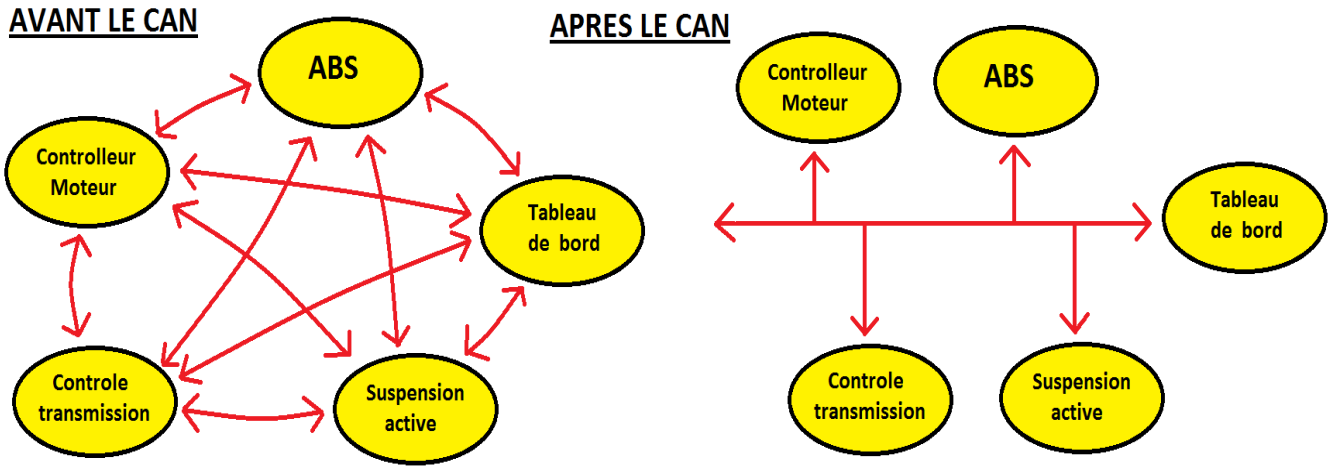


Figure 23 Comparaison avant et après le CAN

4.4.2 Bus différentiel

Le protocole CAN utilise un bus différentiel constitué de 2 éléments : CAN_H et CAN_L. Cela donne un plus fort taux de réjection en mode commun (CMRR) et donc de s'affranchir des problèmes de parasites. Ce type de protection se rajoute à l'utilisation de blindage ou de pair torsadée. En temps normal, un signal électrique est transmis à l'aide d'une référence et d'un signal utile. Malheureusement, si une interférence survient sur la ligne au cours du transport, elle n'est pas détectable et entraîne une déformation du signal (cf. figure 24).

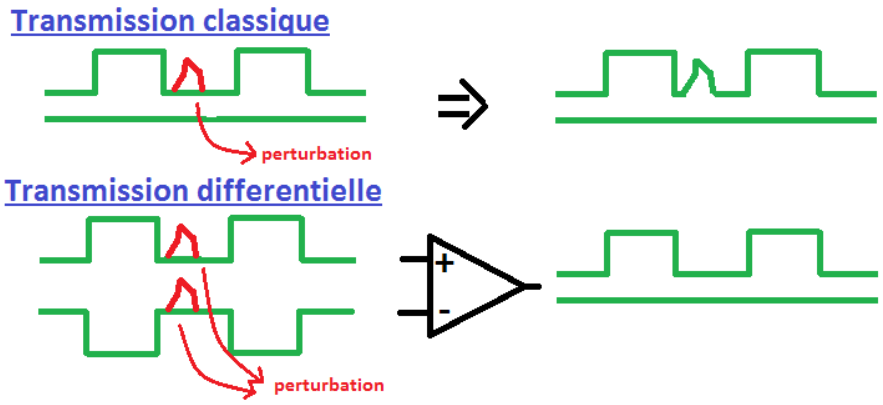


Figure 24 Comparaison transmission classique/différentielle

2 types de transmission existent avec des niveaux de tension différents:

- Transmission en bus CAN « low speed »: vitesse de transmission jusqu'à 125Kb/s.
- Transmission en bus CAN « high speed »: vitesse de transmission jusqu'à 1Mb/s.

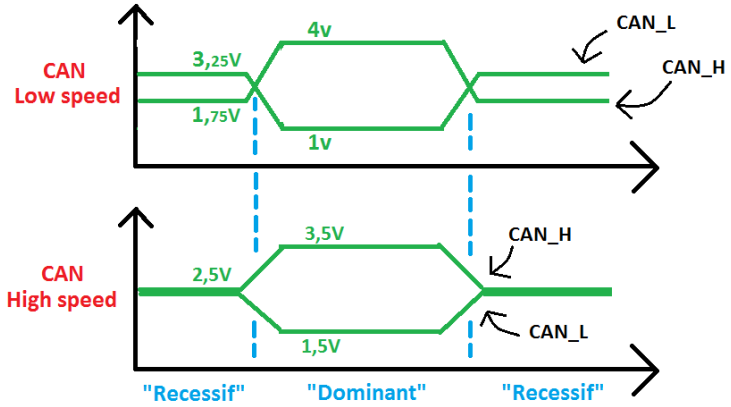


Figure 24 "low speed" contre "high speed"

Le CAN utilise le codage NRZ constitué de 2 niveaux pour chaque bit: un niveau récessif et un niveau dominant. Il utilise moins de transition mais nécessite une horloge très stable. Pour palier à ce problème, la CAN utilise la technique du « bit-stuffing » qui insère un bit récessif après 5 bits dominants consécutifs et inversement.

4.4.3 Standard des trames de données

Il existe 2 spécifications concernant le standard des trames de données.

- **CAN 2.0A** : « Standard frames », trames comportant un identifiant standard de 11 bits, ce qui permet d'accepter théoriquement jusqu'à 2 048 types de messages. C'est la spécification la plus ancienne.

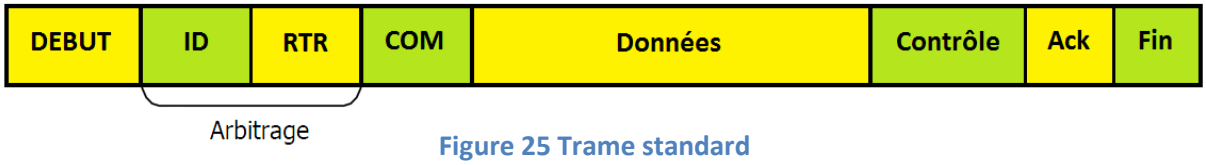


Figure 25 Trame standard

- **CAN 2.0B** : « Extended frames », trames avec un identifiant étendu de 29 bits, ce qui permet d'accepter théoriquement jusqu'à 536 870 912 types de messages. Cette spécification peut fonctionner en cohabitation sur un même bus avec le CAN 2.0A.

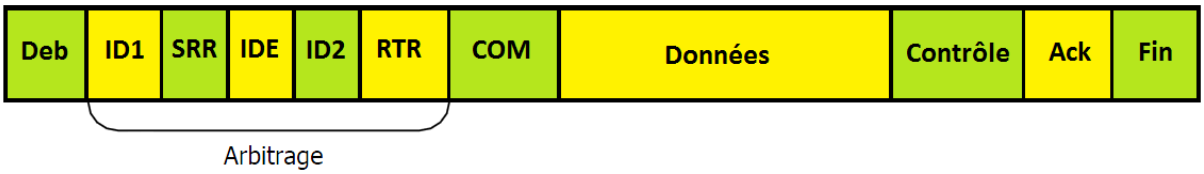


Figure 26 Trame étendue

Voici la décomposition d'une trame CAN :

- **Bit de début** : Il est constitué d'un seul bit dominant et signale le début d'un échange.
- **Champ d'arbitrage** : Le champ arbitrage est constitué de l'identificateur et du bit RTR. L'ID permet d'identifier le message, Il est coupé en 2 dans le cas du CAN 2.0B (ID1: bits 28 à 12 et ID2: bits 11 à 0). Le RTR (Remote Transmission Request) est utilisé pour les messages de demandes de transmission.
- **COM** : Ce champ est composé de 6 bits. Les 2 premiers sont réservés et les 4 suivants vont indiquer le nombre d'octets du champ de données. Ces 4 bits sont appelés le DLC (Data Length Code) et sont compris entre 0 et 8. La taille maximum des données est donc de 8 octets.
- **Données** : Contenant jusqu'à 8 octets selon la valeur du DLC.
- **Champ de contrôle** : 15 bits de CRC. 5 erreurs bits indépendantes sont détectables à 100%. Cette protection se rajoute à la protection hardware.
- **ACK** : est composé de 2 bits, le « ACK Slot » et le « ACK Delimiter » (1 bit récessif). Un nœud, en train de transmettre, place un bit récessif dans le « ACK Slot ». Un nœud, qui reçoit correctement le message, envoie un bit dominant pendant le « ACK Slot ».
- **Bits de fin** : Séquence de 7 bits récessifs. La logique du « bit-stuffing » est désactivée.

4.4.4 Logique du « ET-CABLE »

Le protocole CAN permet l'utilisation de niveau de priorité des messages CAN. En effet dans un véhicule par exemple, la vitesse d'un moteur varie plus vite que la température de l'habitacle. Il est donc logique que la vitesse soit prioritaire sur la température. Le CAN utilise le mécanisme dit du CSMA/CD (Carrier Sense Multiple Acces with Collision Detection). La première partie de cette technique permet à une station d'écouter le support physique de liaison pour déterminer si une autre station transmet une trame de données. La seconde partie permet la détection des collisions et

le traitement de celles-ci. Cette technique s'oppose au mécanisme TDMA (Time-Division Multiple Access) utilisé par exemple par le protocole Flexray où le temps est partagé entre les différents nœuds du réseau. Il est possible de schématiser le CSMA/CD par le principe du ET-Câblé (cf. figure 27 ci-dessous).

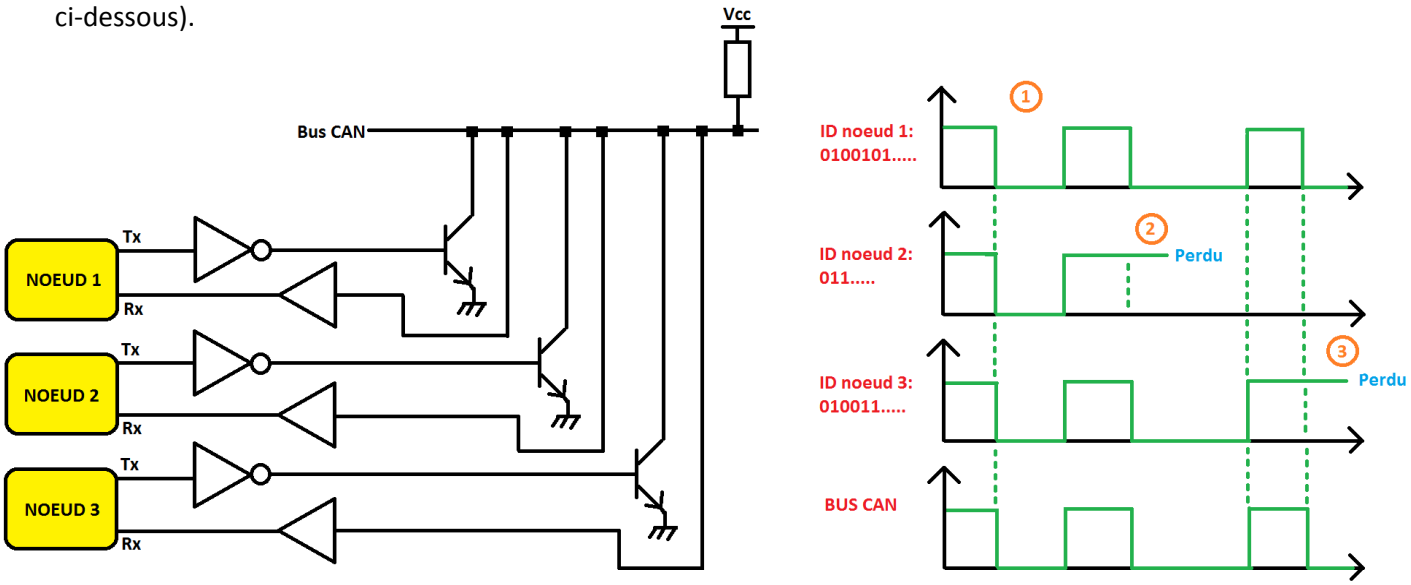


Figure 27 Principe du ET-Câblé

Exemple: Nous avons 3 nœuds connectés au même bus CAN souhaitant envoyer une trame au même moment. Le mécanisme du ET-câblé va permettre de départager le message le plus prioritaire. L'arbitrage commence sur le MSB de l'identifiant et continue tant qu'aucun nœud ne sort vainqueur.

1. Tous les nœuds sont dominants en même temps, aucun nœud ne sort vainqueur. On passe au bit suivant.
2. Le nœud 2 est récessif. Les nœuds 1 et 3 sont dominants et imposent leur niveau au bus. Le nœud 2 perd l'arbitrage et se met en écoute.
3. Le nœud 1 est dominant, le nœud 2 est récessif. Le nœud 1 sort vainqueur de cet arbitrage. Il peut donc émettre son message.

4.5 Diagnostic On CAN

4.5.1 Contexte

À partir des années 1980, les constructeurs automobiles ont commencé à intégrer massivement de l'électronique dans leurs véhicules, en particulier à travers l'utilisation d'un calculateur de contrôle moteur destiné à gérer le fonctionnement du moteur et à diagnostiquer ses défaillances. L'évolution progressive des architectures électroniques a introduit de nouvelles technologies d'échange des données (passage du filaire aux architectures VAN, CAN, LIN), de nouveaux besoins (augmentation de la diagnosticabilité des fonctions et du nombre de calculateurs), et une nécessité de gérer les évolutions (compatibilité entre applications, évolution de la législation). De nos jours, une voiture voit cohabiter de nombreux bus de communication dédié à des fonctions différentes (cf. figure 28) :

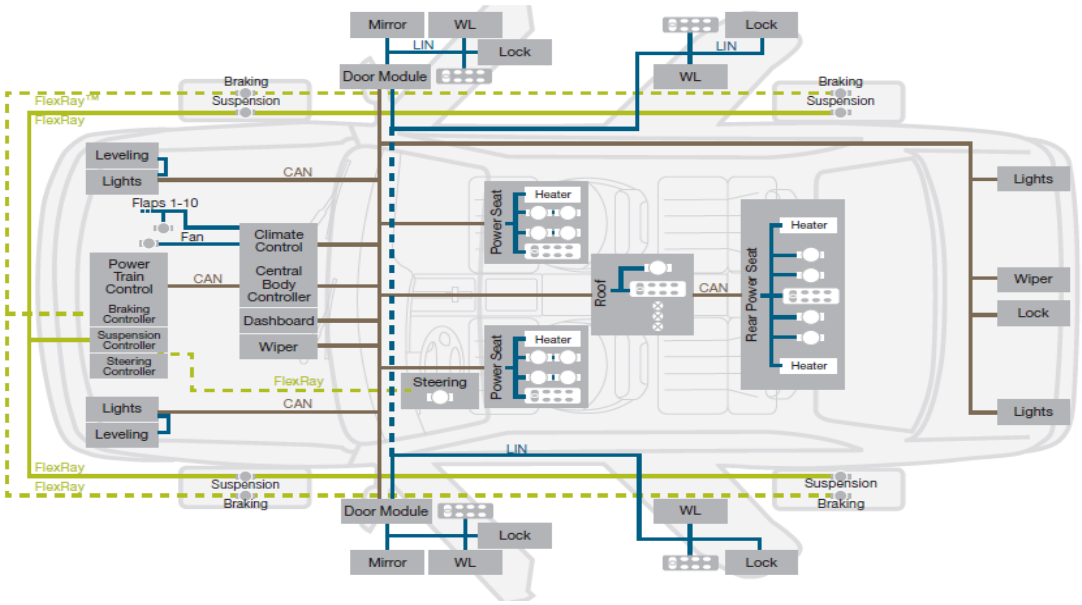
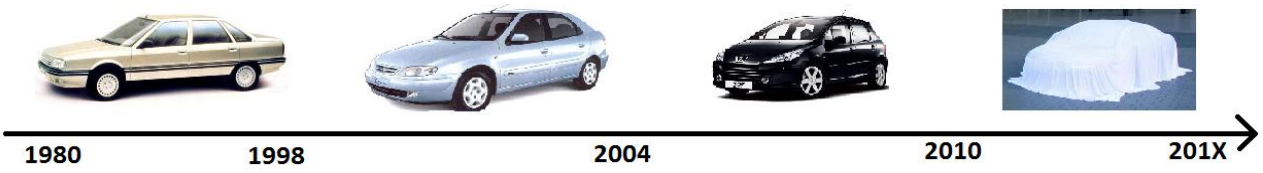


Figure 28 Exemple d'architecture automobile

- **Bus LIN** (Max. 20 kbps) : Dédié au système non-critique tel que la fermeture des portes, les rétroviseurs, ou le toit panoramique. Bus très lent mais peu coûteux car il peut utiliser le module UART d'un microcontrôleur.
- **Bus CAN** (Max. 1 Mbps): Dédié au système de confort, gestion du moteur et de la transmission, etc.
- **Bus Flexray** (Max. 10 Mbps) : Dédié au système de freinage ou de sécurité avancée. Il se distingue du CAN par son débit et sa fiabilité mais également par un prix plus élevée. Il est programmé pour être son remplaçant.
- **Bus MOST (**25 ou 50 Mbps) : dédié aux systèmes de communication entre le système mains-libres et l'autoradio ou le lecteur DVD par exemple. Il utilise de la fibre optique, il est par conséquent très couteux et compliqué à mettre en œuvre.

Les diagnostics embarqués sont devenus progressivement de plus en plus sophistiqués (cf. figure 29 ci-dessous), pour permettre aux moteurs de respecter les seuils d'émissions polluantes, dans un cadre de réglementation de plus en plus strict et contraignant.



KWP2000	KWP2000	KWP2000	UDS	DIAG ON CAN
LIGNE K	OBD	OBD	KWP 2000	DIAG ON LIN
	DIAG ON VAN	DIAG ON CAN	OBD	DIAG ON FLEXRAY
	LIGNE K	LIGNE K	DIAG ON CAN	DIAG ON IP
		LIN 1.X	DIAG ON LIN	WWH-OBD
				UDS
1 ECU	20 ECUs	30 ECUs	60 ECUs	

Figure 29 Evolution du diagnostic

Le rôle du diagnostic des calculateurs (ECUs) répond à 2 besoins principaux : tout d'abord la gestion et la maintenance des fonctions embarquées pour les besoins des constructeurs, ensuite l'accès à des données internes permettant par exemple: le respect des normes anti-pollution. Il permet d'accéder aux informations de défaillance détectées (DTC) par un calculateur (ECU).

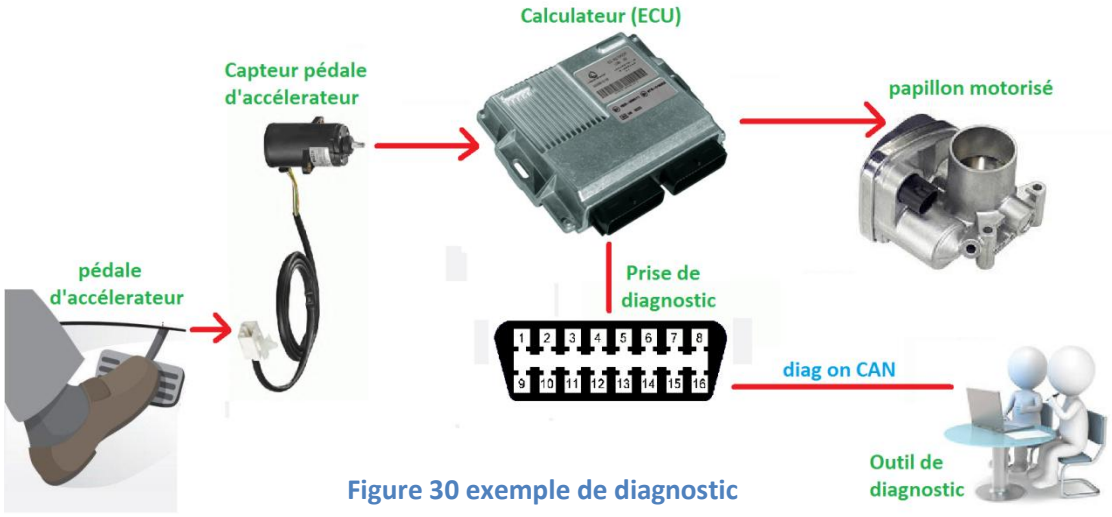


Figure 30 exemple de diagnostic

Dans cet exemple (cf. figure 30 ci-dessus), de nombreuses fonctionnalités sont disponibles depuis l'outil de diagnostic communiquant avec le calculateur via le « Diag On CAN » :

- Lecture de la position de la pédale d'accélérateur.
- Lecture de la position du papillon motorisé.
- Lecture des défauts mémorisés par le calculateur.
- Pilotage du papillon motorisé sans appui sur la pédale.
- Réglage des butées haute et basse de pédale.

4.5.2 Couche réseau

La couche réseau est décrite par la spécification ISO 15765-2. Cette couche a pour objectif l'émission et la réception de messages de taille variable pouvant être supérieur à la trame CAN. La taille maximale étant de 4095 octets.

4.5.2.1 Type de trames

Si le message est supérieur à 7 octets, les données sont segmentées en plusieurs trames. Pour permettre ce résultat, 4 types de trames CAN sont échangés :

- « **SF** » ou « **Single Frame** » : Trame unique pour les messages inférieurs à 8 octets.
- « **FF** » ou « **First Frame** » : Transmission de la taille globale du message et des 6 premiers octets de données.
- « **CF** » ou « **Consecutive Frame** » : Transmission de 7 octets de données et de l'index de la trame.
- « **FC** » ou « **Flow Control** » : Trame de contrôle du flux.

Nom N_PDU	CAN id	Zone de données							
		1	2	3	4	5	6	7	8
Single Frame (SF)	N_AI	N_PCI	N_DATA						
First Frame (FF)	N_AI	N_PCI		N_DATA					
Consecutive Frame (CF)	N_AI	N_PCI	N_DATA						
Flow Control (FC)	N_AI	N_PCI			N_DATA				

Figure 31 Composition des trames CAN

3 zones principales composent ces trames CAN (cf. figure 31 ci-dessus) :

1. « **N_AI** » : Cette zone contient l'ensemble des adresses (cf. figure 32 ci-dessus).

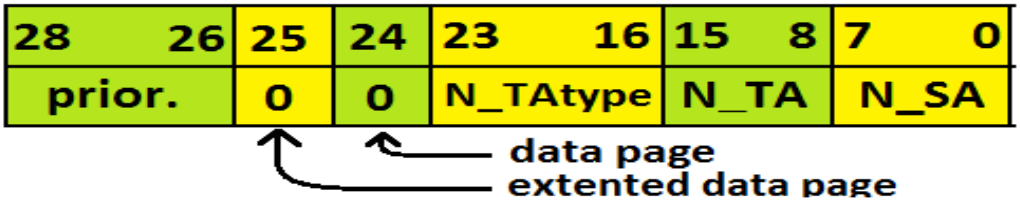


Figure 32 Composition identifiant CAN

- ✓ « **Priority** » : niveau de priorité du message allant de 0 à 7.
- ✓ « **Extented Data page** » et « **Data page** » : Définition du format de l'identifiant.
- ✓ « **N_TAtype** » : 2 valeurs sont possibles : 218 pour une « physical addressing » (communication 1 vers 1) et 219 pour une « functional addressing » (communication 1 vers n).
- ✓ « **N_TA** » : addresse sur 8 bits de la cible de la communication.
- ✓ « **N_SA** » : addresse sur 8 bits de la source de la communication.

L'identifiant peut être étendu, il empiète pour cela sur la zone de données.

2. « **N_PCI** » : Cette zone permet l'identification du type de trame et contient les informations propres à chacune (cf. figure 33 ci-dessous). Elle peut occuper jusqu'à 3 octets.

	OCTET 1		OCTET 2	OCTET 3
Nom N_PDU	Bits 7-4	Bits 3-0	Bits 7-0	Bits 7-0
Single Frame (SF)	0	SF_DL	N/A	N/A
First Frame (FF)	1	FF_DL		N/A
Consecutive Frame(FF)	2	SN	N/A	N/A
Flow Control (FC)	3	FS	BS	STmin

Figure 33 Composition du N_PCI

- ✓ « **SF_DL** » ou « **Single Frame Data Length** » : Contient la taille de la trame unique (0-7 octets).
- ✓ « **FF_DL** » ou « **First Frame Data Length** » : Contient la taille du message segmenté (8-4095 octets).

- ✓ « **SN** » ou « **Sequence Number** » : Index de la trame. Il s'incrémente à chaque nouveau « CF » et est remis à 0 lorsqu'il atteint 15.
- ✓ « **FS** » ou « **Flow Status** : Il permet d'indiquer à l'expéditeur s'il peut poursuivre la transmission.
  - ❖ FS=0 (« ContinueToSend ») : L'émission peut continuer.
  - ❖ FS=1 (« Wait ») : L'expéditeur doit attendre une autre trame de contrôle.
  - ❖ FS=2 (« Overflow ») : La « FF » reçu contient une taille de message supérieur à la taille du buffer en réception.
- ✓ « **BS** » ou « **Block Size** » : Indication du nombre de trames que peut envoyer l'expéditeur avant la prochaine trame de contrôle.
- ✓ « **STmin** » ou « **Separation Time min** » : Délai minimum à observer par l'expéditeur entre 2 trames consécutives.

3. « N_DATA » : Cette zone contient les données brutes du message et fait au maximum 7 octets.

4.5.2.2 Transmission de données non segmentées

Lors de la transmission de données non segmentées, une trame unique est utilisée (« SF »). Les services de lecture ont très souvent besoin d'une unique trame comme par exemple le service entre l'outil de diagnostic et le calculateur « readDataByLocalIdentifier » utilisé par le protocole de diagnostic « KWP2000 » (cf. figure 34 ci-dessous). Il permet de récupérer la valeur d'une variable avec son index.

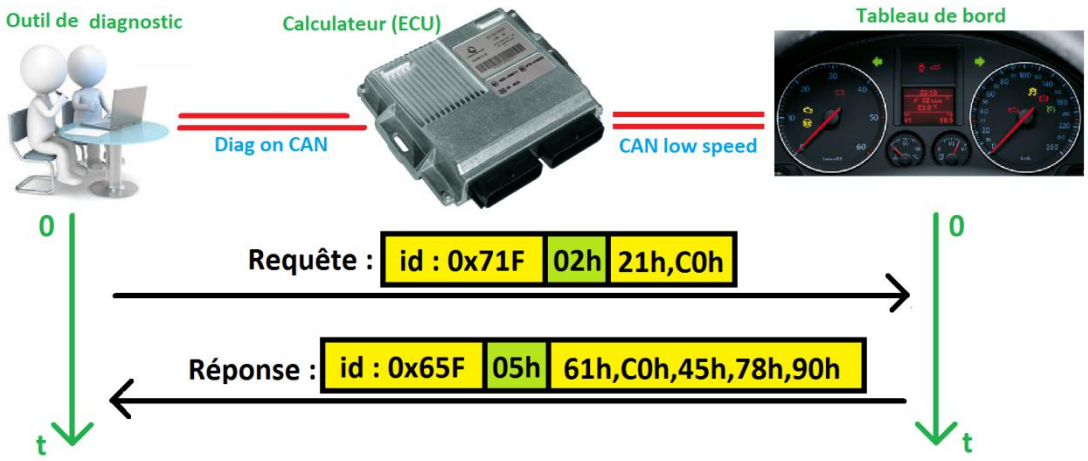


Figure 34 Echange de trames non-segmentées

4.5.2.3 Transmission de données segmentées

3 trames différentes sont utilisées pour ce type de transmission : « FF », « CF », et « FC ». L'émetteur envoie la première trame (« FF ») : elle initie la communication et envoie les premières données. Le récepteur répond juste après avec la première trame « FC » qui lui permet de contrôler le flux. L'émetteur poursuit l'échange en envoyant des « CF » contenant les données. Il les envoie sans discontinuer jusqu'à atteindre la valeur de « Block Size » délivrée dans le précèdent « Flow Control ».

Il doit attendre la réception d'un nouveau « Flow Control » pour reprendre la transmission de données. Ces opérations se réalisent jusqu'à l'envoi complet du message connu grâce à « FF_DL ».

Exemple : Transmission de « 123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ » du calculateur vers l'outil de diagnostic (cf. figure 35 ci-dessous).

T en s.	Trame	N_PCI	Data	Decodage
1.9016	Phys Tx ←	10 3F	61 80 31 32 33 34	First Frame, Data= « 1234 »
1.9117	Phys Rx →	30 03 64		Flow Control CTS, BS=3, STmin=100ms
2.0133	Phys Tx ←	21	35 36 37 38 39 41 42	Consecutive Frame, Data = « 56789AB »
2.1136	Phys Tx ←	22	43 44 45 46 47 48 49	Consecutive Frame, Data = « CDEFGHI »
2.2126	Phys Tx ←	23	4A 4B 4C 4D 4E 4F 50	Consecutive Frame, Data = « JKLMNOP »
2.3146	Phys Rx →	30 03 64		Flow Control CTS, BS=3, STmin=100ms
2.4156	Phys Tx ←	24	51 52 53 54 55 56 57	Consecutive Frame, Data = « QRSTUVW »
2.5155	Phys Tx ←	25	58 59 5A	Consecutive Frame, Data = « XYZ »

Figure 35 Echange de données segmentées

4.5.2.4 Services de communication

Au niveau de la couche réseau, 4 services existent. Leurs objectifs sont la transmission ou la réception des données ainsi que la transmission des informations entre la couche réseau et la couche application (cf. figure 36) :

- « **N_USData.request** » : Ce service est utilisé par l'expéditeur pour lancer l'émission de données. Il permet la transmission des données et les segmente si-nécessaire.
- « **N_USDataFF.indication** » : Il est appelé par le récepteur lors de la réception de la première trame du message et il en informe la couche supérieure.
- « **N_USData.indication** » : Il est appelé par le récepteur lors de la réception de la dernière trame du message et il en informe la couche supérieure.
- « **N_USData.confirm** » : Ce service est utilisé par l'expéditeur pour informer la couche supérieure de la fin de la transmission du message.

Sur le même principe, 3 services sont utilisés par la couche inférieure à la couche réseau : la couche liaison. Ils permettent entre autres, la transmission des informations entre la couche liaison et la couche réseau (cf. figure 36) :

- « **L_Data.request** » : transmission de la trame CAN de l'expéditeur vers le récepteur. Elle peut être de type « SF », « FF », « CF » ou « FC ».
- « **L_Data.indication** » : Confirmation du récepteur de la réception de la trame CAN à la couche supérieure: la couche réseau.
- « **L_Data.confirm** » : Confirmation de l'expéditeur de l'émission de la trame CAN à la couche supérieure: la couche réseau.

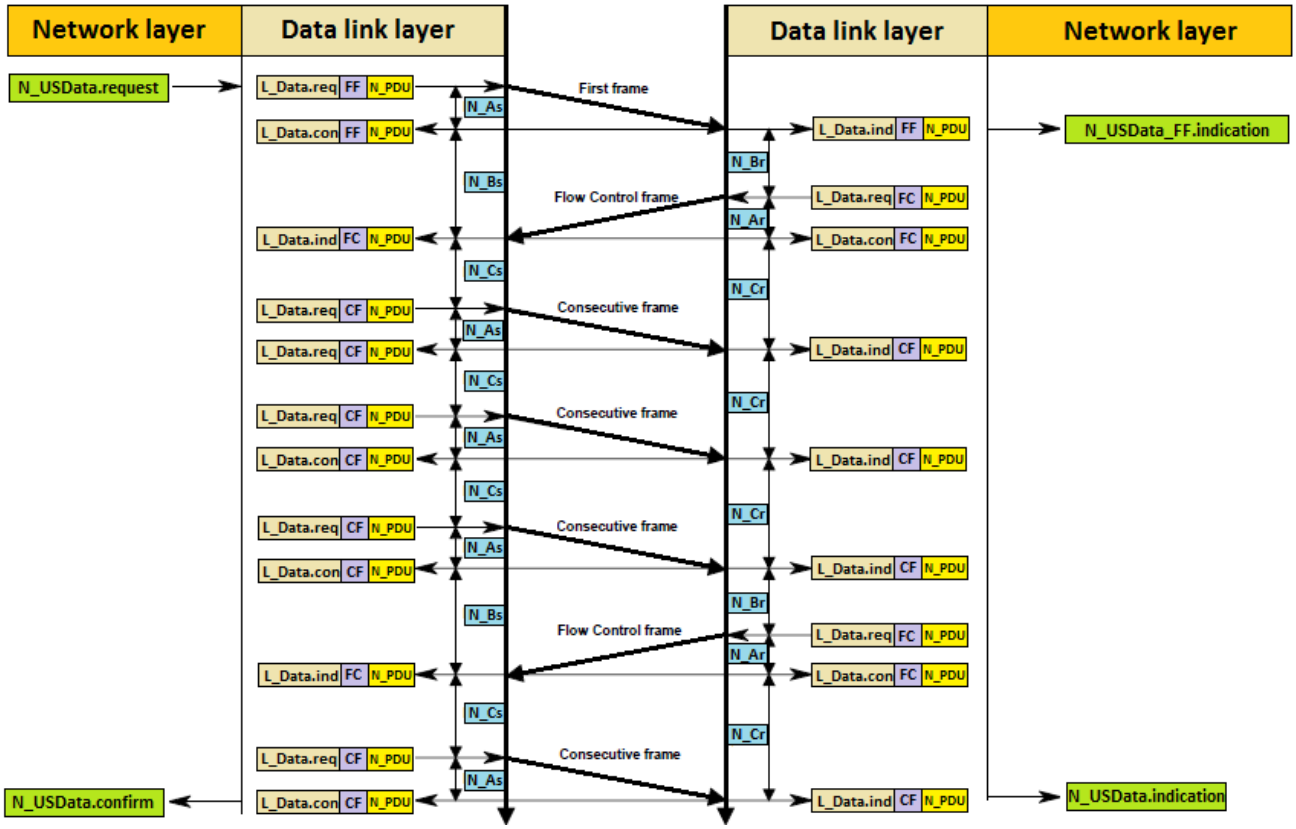


Figure 36 Echange de données segmentées

4.5.2.5 Timers

La supervision des échanges comporte la gestion d'un ensemble de timers permettant de détecter un disfonctionnement durant les phases d'échanges. En cas de détection d'une erreur, le récepteur abandonne la trame en cours et fournit à la couche supérieure une indication sur le type de l'erreur détectée en utilisant le service « L_Data.con ». L'émetteur doit prendre en compte cette erreur pour reprendre l'échange. 6 timers sont présents (cf. figure 36 ci-dessus) et imposent des délais au-delà desquels un message d'erreur est envoyée à la couche supérieure :

- « **N_As** » : Délai de transmission d'une trame CAN côté expéditeur.
- « **N_Ar** » : Délai de transmission d'une trame CAN côté récepteur.
- « **N_Bs** » : Délai jusqu'à la réception du prochain « Flow Control ».
- « **N_Br** » : Délai jusqu'à la transmission du prochain « Flow Control ».
- « **N_Cs** » : Délai jusqu'à la transmission du prochain « Consecutive Frame ».
- « **N_Cr** » : Délai jusqu'à la réception du prochain « Consecutive Frame ».

4.6 « Unified Diagnostic Services »

Le protocole « UDS » suit la spécification ISO14229-1 et se situe au sommet du modèle OSI. Il repose sur les spécifications vues précédemment concernant le « CAN » et le « Diag On CAN ». Il est basé sur le principe d'une communication client-serveur. Le client est en règle général l'outil de diagnostic

tandis que le serveur est le calculateur ou l'« ECU ». La communication est initiée généralement par le client et le serveur lui répond (cf. figure 37). Pour échanger une requête ou une réponse, il utilise les services de communication du « Diag On CAN » fournis au niveau de la couche réseau : « N_USData.request », « N_USDataFF.indication », « N_USData.indication », « N_USData.confirm ».

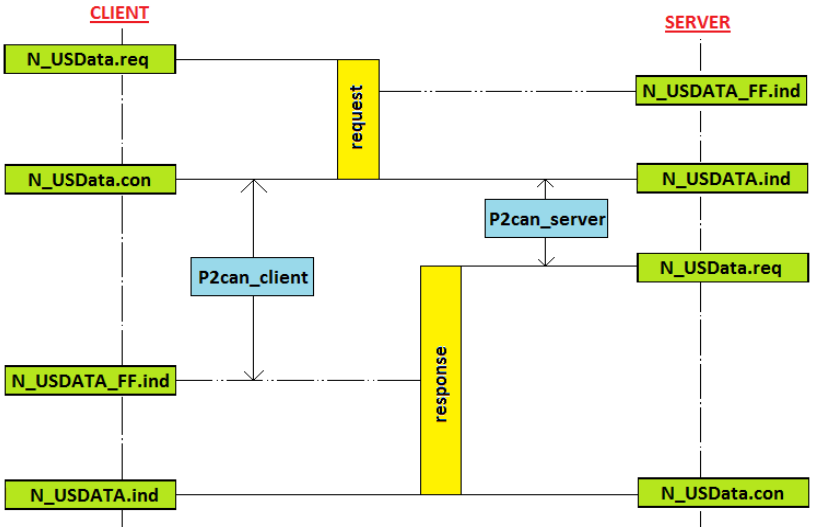


Figure 37 Echange client-serveur

Les services UDS sont utilisés pour communiquer entre le client et le serveur. Il est possible de les regrouper en 4 groupes principaux (cf. figure 38 ci-dessous) :

- **Les services de gestion du diagnostic et de la communication**. Ils sont nécessaires au contrôle des sessions et au maintien de la session en cours.
- **Les services d'accès aux données internes des « ECUs »** accessibles grâce à l'adresse mémoire ou à un identificateur.
- **Les services de diagnostic associés à la gestion des défauts**. Il est possible de lire les défauts mémorisés et de les effacer de la mémoire du calculateur.
- **Les services dédiés au téléchargement** permettent d'échanger une grande quantité de données pour reprogrammer un calculateur par exemple.

Functional Unit	SID	Available in Default Session	Available for RoE	Has Sub-Function	Service Name	Mnemonic
Diagnostic and Communication Management	$10	✓		✓	Diagnostic Session Control	DSC
	$11	✓		✓	ECU Reset	ER
	$27			✓	Security Access	SA
	$28			✓	Communication Control	CC
	$3E	✓		✓	Tester Present	TP
	$83			✓	Access Timing Parameter	ATP
	$84				Secured Data Transmission	SDT
	$85			✓	Control DTC Setting	CDTCS
	$86	✓		✓	Response On Event	ROE
	$87			✓	Link Control	LC
Data Transmission	$22	✓	✓		Read Data By Identifier	RDBI
	$23	✓			Read Memory By Address	RMBA
	$24	✓			Read Scaling Data By Identifier	RSDBI
	$2A				Read Data By Periodic Identifier	RDBPI
	$2C	✓		✓	Dynamically Define Data Identifier	DDDI
	$2E	✓			Write Data By Identifier	WDBI
	$3D	✓			Write Memory By Address	WMBA
Stored Data Transmission	$14	✓			Clear Diagnostic Information	CDTCI
	$19	✓	✓	✓	Read DTC Information	RDTCI
Input Output Control	$2F		✓		Input Output Control By Identifier	IOCBI
Remote Activation of Routine	$31	✓	✓	✓	Routine Control	RC
Upload Download	$34				Request Download	RD
	$35				Request Upload	RU
	$36				Transfer Data	TD
	$37				Request Transfer Exit	RTE

Figure 38 Liste des services UDS disponibles

CHAPITRE V : IMPLEMENTATION DE L'UDS

5.1 Introduction

Cette partie est consacrée à l'implémentation de l'UDS. Elle s'est faite en parallèle sur le client et le serveur. La programmation du serveur s'est réalisée en C dans le DsPIC en codant les différentes couches du modèle OSI. Une interface graphique C# a été conçue sur ordinateur pour faire office de client. Une librairie dynamique conçue en C contient les fonctions nécessaires au fonctionnement de l' « UDS ».

5.2 Le serveur

5.2.1 Composition du programme

Plusieurs fichiers sont affectés à l'implémentation de l'UDS. Les fichiers de plus bas niveaux concernant le CAN étaient fournis.

- **Comcan_33F.c** : Il est composé des fonctions permettant d'initialiser et d'utiliser le module CAN. Il comporte la fonction d'interruption en réception du bus CAN ainsi que les masques et les filtres d'acceptances.
- **Comcan_33F.h** : Définition et prototype du fichier « Comcan_33F.c ».
- **Comcan_33F_cfg.h** : Permet la configuration de la vitesse de transmission et du « bit time » (Durée des 4 différents segments composant la réception d'un bit).
- **Mailboxes_33F_gen.c :** Génération automatique depuis une macro Excel réalisée par T&S. Il permet de configurer les paramètres des messages CAN utilisés comme les filtres ou les buffers de transmission propres à chaque message.
- **Mailboxes_33F_gen.h :** Contient entre autres les structures des buffers et différentes définitions liées à chaque message.
- **ISO15765_2.c** : Les différentes fonctions de la couche liaison et réseau du « Diag On CAN » sont stockées dans ce fichier.
- **ISO15765_2.h** : Contient les structures des « N _PDU », les paramètres de temps, et les prototypes des fonctions du « Diag On CAN ».
- **ISO15765_3.c** : Les différentes fonctions de la couche application sont stockées dans ce fichier. Il contient les services UDS déjà implémentés.
- **ISO15765_3.h** : Contient les paramètres de temps de la couche application, la définition des mnémoniques et les prototypes des fonctions « UDS ».
- **UDS_cfg.h** : Configuration des adresses, des paramètres à l'intérieur des « Flow Control » et des paramètres de temps de la couche application.

En plus de tous ces fichiers, d'autres modules périphériques du DsPIC sont utilisés : Le module DMA pour la liaison CAN et le module Timer pour la gestion des paramètres de temps.

5.2.2 Fonctionnement du module CAN du DsPIC

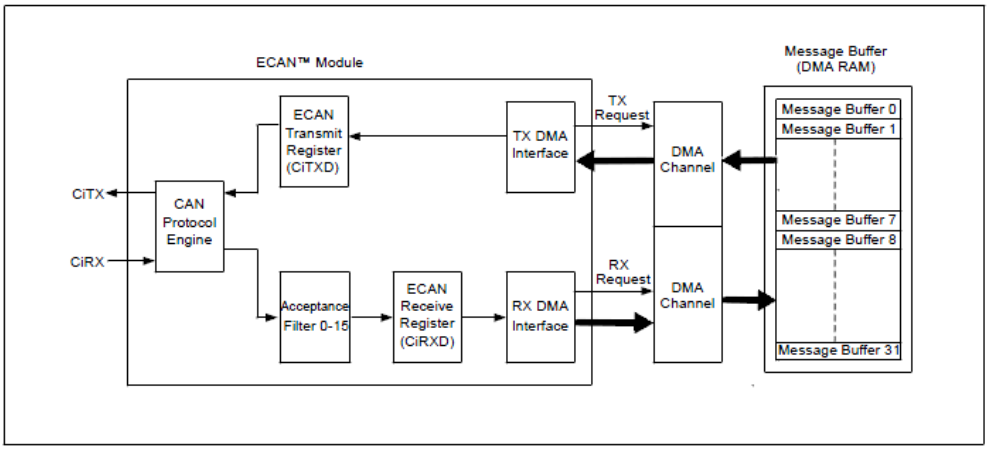


Figure 39 Module ECAN du DsPIC 33F

Le module ECAN fournit les fonctionnalités permettant de recevoir et d'émettre des messages sur le bus (cf. figure 39 ci-dessus) :

- Lorsqu'un message est reçu par le microcontrôleur, le champ « Identifiant » du message est comparé avec les valeurs des filtres. En cas de correspondance, le message est transféré dans le buffer de réception du filtre dans la DMA puis dans une FIFO. Les masques associés aux filtres spécifient les bits de l'identifiant à examiner.
- La transmission d'un message se fait par l'introduction du message dans un buffer de transmission au niveau de la DMA. Le niveau de priorité du message est requis.

5.2.3 Fonctionnement des couches basses

Toutes les fonctions de cette partie sont comprises dans le fichier ISO15765_2.c. On peut les décomposer en plusieurs groupes :

- **Fonctions de la couche physique**: « FUN_ISO15765_2_set_octet_CAN » et « FUN_ISO15765_2_get_octet_CAN ». Elles permettent de récupérer les octets d'une trame CAN en réception et d'imposer leurs valeurs en émission. L'envoi effectif de la trame se réalise à l'aide de la fonction : « FUN_ISO15765_2_Transmit ».
- **Fonctions de la couche liaison**. Elles correspondent aux différents services de cette couche : « L_Data.request », « L_Data.indication », et « L_Data.confirmation ». Dans chaque fonction se trouve des sélecteurs de type « Switch-case » déterminant le type de la trame en cours de traitement.
- **Fonctions de la couche réseau**. Elles correspondent aux différents services de cette couche : « N_USData.request », « N_USDataFF.indication », « N_USData.indication », « N_USData.confirm ».

- **Fonctions gérant les paramètres de temps de la couche liaison**. La fonction d'initialisation du module comporte les durées des Timers. Un unique timer du DsPIC est en réalité utilisé pour gérer les 6 timers de la couche liaison. C'est réalisable car les 6 timers ne fonctionnent jamais en même temps. Le module Timer du DsPIC sert donc simplement de base de temps pour savoir si un délai est dépassé.

```
typedef struct
{
   uint16_t N_PCItype;          /*!<Single,first ou consecutive frame ou flow control*/
   uint8_t  SF_DL :4;           /*!<Data length: gamme:0-6 (extented) ou 0-7 (normal)*/
   uint16_t FF_DL :12;          /*!<Data length: gamme:7-ffff(extented) ou 8-ffff(normal)*/
   uint8_t  SN :4;              /*!<Sequence number gamme:0-15 reinit à 0 quand on est à 15*/
   uint8_t  FS :4;              /*!<Flow status: ContinueToSend ou Wait*/
   uint8_t  BS :8;              /*!<Block size: Gamme: 01-FF*/
   uint8_t  STmin :8;           /*!<Separation_time_min: Gamme: 00-FF (mesure en ms)*/

}Network_PCI;
```

Figure 40 Structure avec les paramètres des N_PCI

Les structures permettent de faire passer les paramètres des messages entre les couches. Pour cela le pointeur de la structure est mis en paramètre dans le prototype de chaque fonction. Certaines structures sont par exemple composées des adresses, des données et des paramètres propres à chacun des messages. La structure « Network_PCI » est composée de tous les paramètres propres à chaque type de trame : « SF », « FF », « CF » et « FC » (cf. figure 40 ci-dessus).

5.2.4 Implémentation des services UDS

2 services UDS ont étés complètement implémentés durant ce stage. Ils permettent la gestion des différentes sessions du diagnostic. En plus, un troisième service est en cours d'implémentation permettant de récupérer des défauts détectés par le calculateur. Il sera plus rapide par la suite de développer d'autres services car les couches sur lesquelles reposent ces services sont finalisées.

5.2.4.1 « DiagnosticSessionControl »

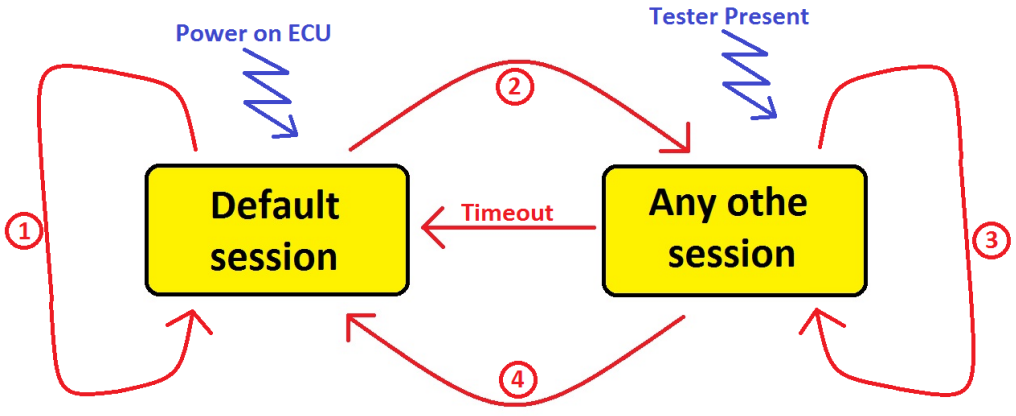


Figure 41 Gestion des différentes sessions

C'est le premier service codé. Il gère l'activation des différentes sessions et le passage d'une session à une autre. Les principales sessions sont la « Default session », la « Programming Session », l'« Extented Diagnostic Session » et la « Safety System Diagnostic Session ». On distingue la « Default session » des autres sessions (cf. figure 41) car tout diagnostic commence par cette session. Chaque session a la particularité d'autoriser l'utilisation de certains services spécifiques. Il suffit donc de bloquer les services non-concernés par la session en cours.

1. Le serveur est en « session default » et le client veut commencer une « session default ». Dans ce cas le serveur réinitialise la session en faisant un reset programme implémenté dans le code.
2. Le serveur transite de la « session default » vers un autre type de session. Il réinitialisera seulement les évènements configurés dans le serveur via le « ResponseOnEvent service » durant la session de défaut.
3. Lorsque le serveur transite entre 2 sessions différentes de la session de défaut, le serveur réinitialise les différents évènements.
4. Lors d'une transition d'une session différente de la session de défaut vers la session de défaut, le serveur réinitialise chaque évènement qui est configuré via le « ResponseOnEvent service ».

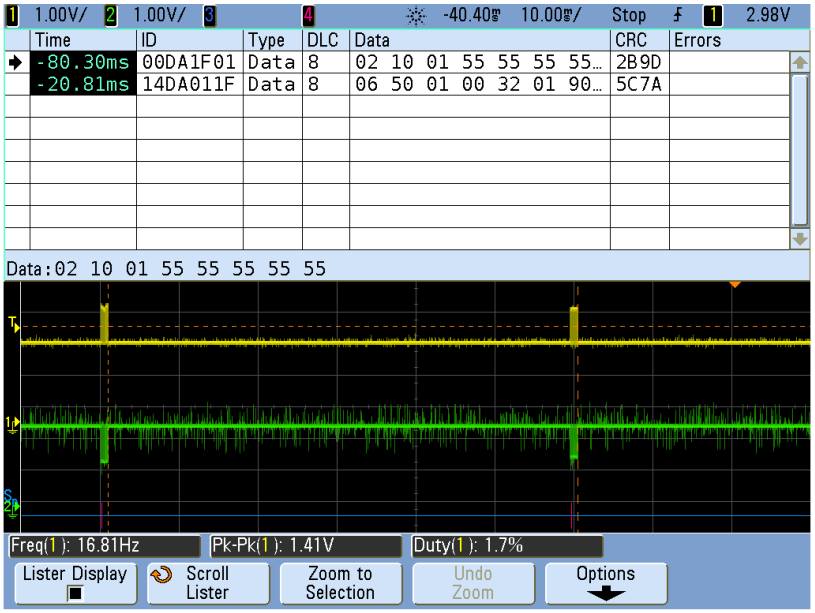


Figure 42 Requête et réponse pour entrer en "Default Session"

Lors de la communication (cf. figure 42 ci-dessus), le client envoie une requête avec l'identifiant du service (0x10). Le second paramètre est l'identifiant de la session souhaitée par exemple 0x01 pour la « Default Session » ou « 0x02 » pour la « Programming Session ». Le client utilise une « Single Frame » où les octets vides sont complétés avec 0x55. L'identifiant d'une réponse à un service est calculé comme suit : ID service + 0x40. Dans ce cas, le client répond avec l'identifiant 0x50 et l'écho de l'identifiant de la session. Il fournit en plus 2 paramètres de temps :

- **P2can_client** (0x0032 dans l'exemple = 50ms) : Durée maximale entre la confirmation d'une requête et l'indication du début de la réception d'une réponse. Le paramètre est configurable depuis le fichier « UDS_cfg.h ».
- **P2can_client*** (0x0190 dans l'exemple = 4000ms) : Ce paramètre est identique au précèdent dans le cas d'une réponse négative venant du serveur.

Une réponse négative peut être émise, si le serveur est dans l'impossibilité d'interpréter le service. Par exemple, si le client veut ouvrir une session avec un identifiant de session inconnu, le serveur enverra cette réponse :

- Octet 1 : **0x7F** (« NRSI » ou « Negative response service identifier ») : identifiant dédié aux réponses négatives pour tous les services.
- Octet 2 : **0x10** (« DSC » ou « Diagnostic Session Control ») : Identifiant du service concerné.
- Octet 3 : **0x12** (« SFNS » ou « Sub-function not supported»): Code pour signaler un défaut dans un sous-paramètre du service.

5.2.4.2 « TesterPresent »

Il permet de maintenir la session active ouverte en absence d'autre requête si la session n'est pas une « Default Session ». Ce service est donc émis automatiquement à intervalle régulier. Le paramètre de temps « S3client » définit la durée recommandée et la durée maximale entre 2 services « TesterPresent ». En l'absence de « TesterPresent », le calculateur (ECU) passe automatiquement en « Default Session ».

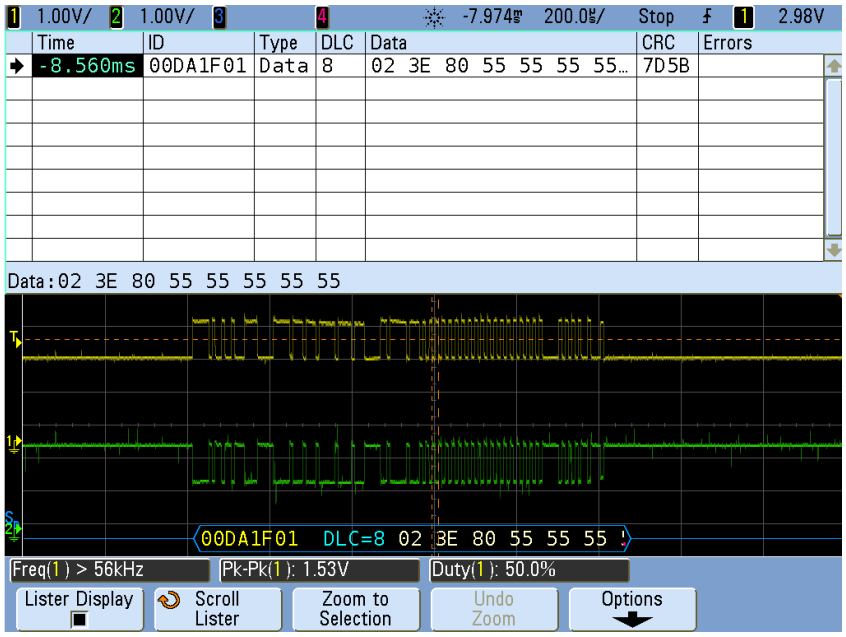


Figure 43 trame "SF" du service "Testerpresent"

Ce service comporte simplement 2 paramètres :

- Octet 1 : **0x3E** (« TP » ou « Tester Present »): identifiant du service « Testerpresent ».
- Octet 2 : **0x00** (« ZSUBF » ou « Zero Sub-Function ») : identifiant pour signifier que le service ne possède pas de sous-paramètre.

Dans notre cas (cf. figure 43), l'octet 2 possède la valeur 0x80. En réalité, seul le bit de poids fort du second octet est mis à 1 et modifie par conséquent la valeur du sous-service concerné. Ce bit s'appelle « SuppresPosRspMsgIndicationBit », il est disponible dans de nombreux services. Il permet d'utiliser une des particularités de l' « UDS » par rapport à d'autres outils de diagnostic tel que le « KWP 2000 ». La symétrie entre le nombre de requêtes et le nombre de réponses n'est pas systématique. En effet la mise à « 1 » de ce bit désengage le serveur de répondre au client.

5.2.4.3 « ResponseOnEvent »

Ce service fournit la possibilité d'automatiquement exécuté un service lors d'un évènement se produisant dans le serveur (cf. figure 44 ci-dessous). Durant le stage, une partie de ce service a été implémenté. Elle permet d'ouvrir une fenêtre temporelle à l'intérieur de l' « ECU ».

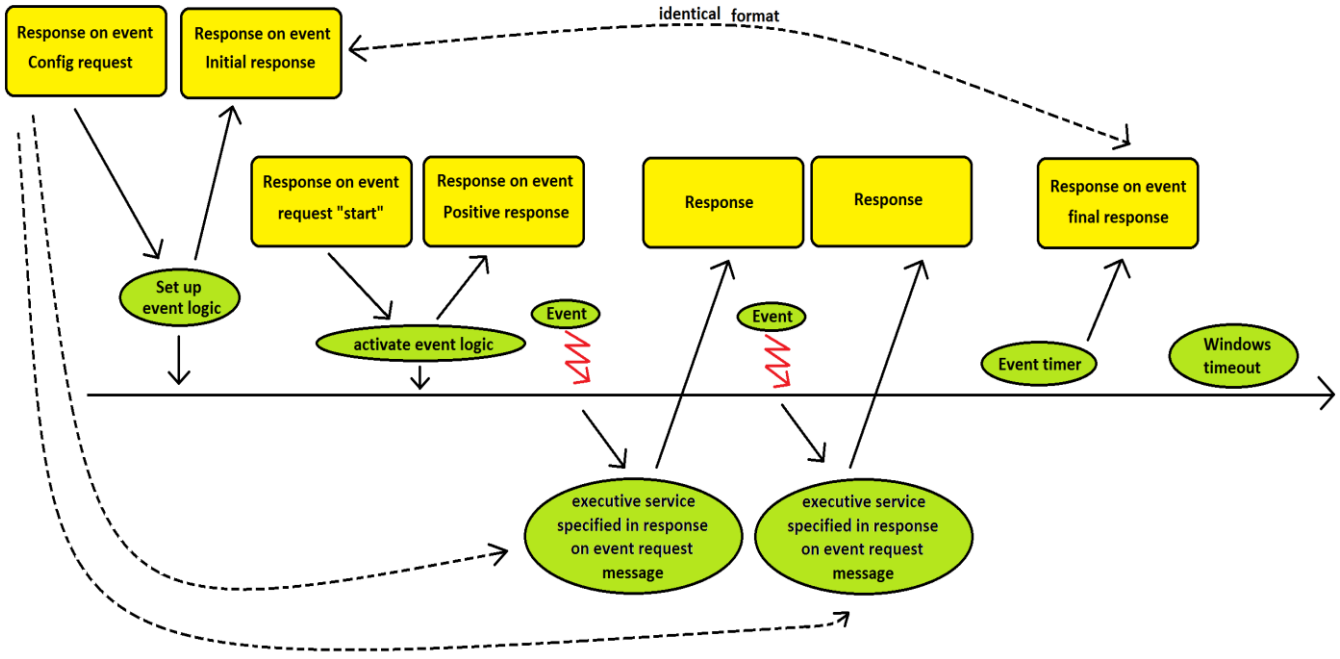


Figure 44 Fonctionnement du service Response On Event

Ce service possède l'identifiant 0x86, les réponses sont envoyées avec 0xC6. La partie implémentée permet d'ouvrir une fenêtre temporelle bornée dans laquelle le serveur pourra réagir à des évènements. Pour utiliser cette fenêtre, plusieurs trames sont échangées entre le client et le serveur :

1. **Client -> Serveur** : Requête de configuration. Elle possède plusieurs paramètres comme le choix du type d'évènements déclenchant une réponse de l' « ECU » (0x01 :

« OnDTCstatusChange »), La taille de la fenêtre en secondes, le service utilisé par le calculateur pour réagir à un évènement, etc.

2. **Serveur -> Client** : Réponse positive du serveur qui envoie l'écho des paramètres de configuration.
3. **Client -> Serveur** : Requête de « Start ». Le client demande au serveur de démarrer la fenêtre.
4. **Serveur -> Client** : Le calculateur signifie à l'outil de diagnostic que la fenêtre est ouverte.
5. **Serveur -> Client** : « Final Response ». L' « ECU » notifie la fermeture de la fenêtre.

Il est possible d'implémenter par la suite d'autres services pour permettre au serveur de répondre à un évènement. Le service avec lequel il répond, est donné dans les paramètres de la 1ere trame de configuration. Souvent le « readDTCInformation » est utilisé. Ce service permet de lire ce que l'on appelle les « DTC » ou « Diagnostic Trouble Code ». Ce sont des codes alphanumériques (cf. figure 45 ci-dessous) qui identifient un problème rencontré par l'un ou l'autre des systèmes supervisés par le calculateur. Chaque code de problème constitue un message, qui décrit le circuit, le composant ou le système à l'endroit où le problème est survenu.

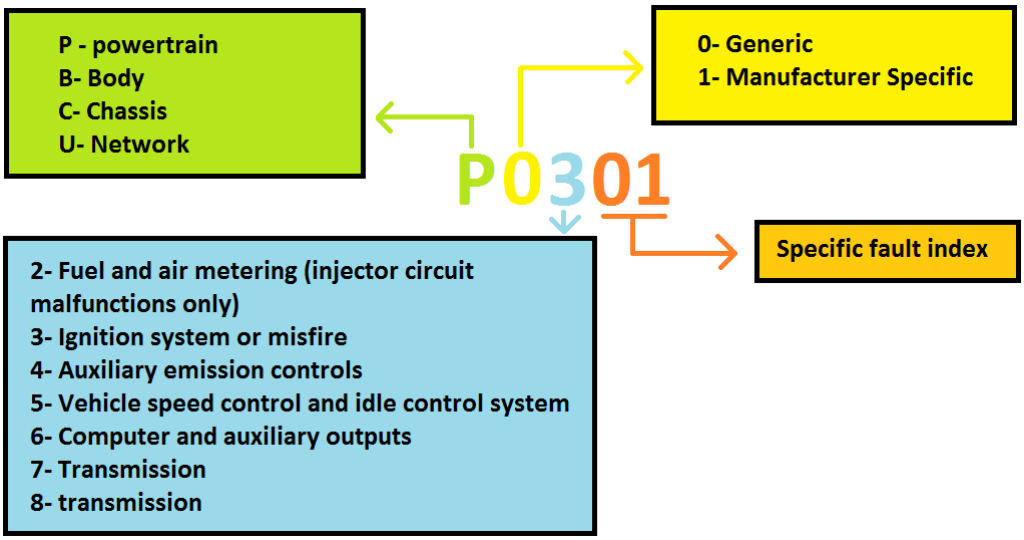


Figure 45 Composition d'un "DTC"

Ils sont constitués de 4 éléments :

- Le 1er caractère est une lettre qui identifie le système principal : carrosserie, châssis, etc.
- Le 2nd caractère est un chiffre qui identifie le type de code : code générique ou propre au fabricant (Audi, BMW, PDA, etc.).
- Le 3eme caractère est un chiffre qui identifie le système ou le sous-système spécifique où se situe le problème.
- Les 2 derniers caractères sont des chiffres qui identifient la section du système qui connaît des problèmes par exemple le cylindre numéro 1.

5.3 Le client

Le client (l'outil de diagnostic) permet d'entrer en communication avec le calculateur (la carte électronique). Dans notre cas, cet outil est développé en C# directement sur ordinateur. La communication entre les 2 entités se fait par le biais de l'outil « CANalyzer ». Il possède 2 « channels » : une contenant un transceiver « Low speed » et l'autre un transceiver « High speed » reliés directement à un bus CAN. Il faut par la suite configurer la vitesse de transmission et lier le programme C# au « CANalyzer » à travers le logiciel fourni avec (cf. figure 46 ci-dessous).

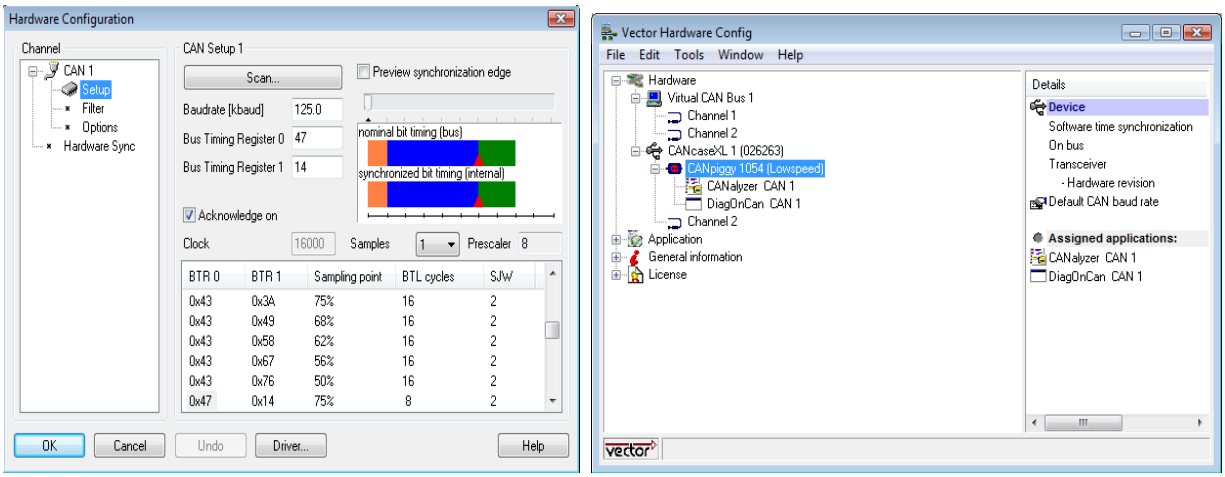


Figure 46 configuration hardware CANalyzer

Dans un projet plus ancien, un stagiaire a développé un outil de diagnostic en C#. Le 1er objectif était d'assimiler son fonctionnement et de l'adapter à mon projet. Le second était de rendre mon programme le plus générique possible par la création d'une librairie dynamique.

5.3.1 Utilisation de DLL

Une « DLL » est une librairie dynamique (« Dynamic link library »). Ses fonctions internes ne sont pas inclues dans l'exécutable mais sont appelées pendant l'exécution du programme d'où le nom de dynamique. Plusieurs programmes peuvent accéder à une « DLL » alors que la bibliothèque est chargée une seul fois. Elle s'oppose aux librairies statiques moins lourdes mais chargées directement dans l'édition des liens. Seul l'exécutable du programme est nécessaire au fonctionnement du code, par contre les fonctions sont dupliquées pour chaque programme utilisant cette librairie.

5.3.1.1 « XL Driver library »

Le précédent programme utilisait déjà une « DLL ». Elle est fourni par « Vector » le concepteur du « CANalyzer » et permet d'utiliser directement le bus CAN dans le logiciel C# en fournissant des fonctions de configuration, d'envoi et de réception. Il fait le pont entre le driver du « CANalyzer » et l'application développée (cf. figure 47 ci-dessous).

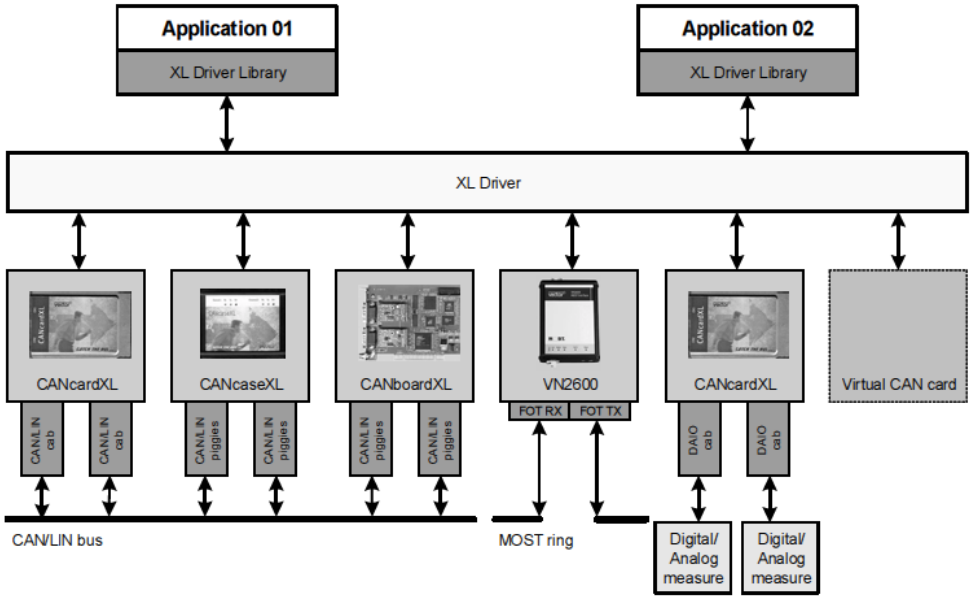


Figure 47 Lien entre l'application et le bus CAN

Les fonctions dans la librairie dynamique sont écrites en C tandis que le logiciel est en C#. Il est donc nécessaire d'utiliser un « wrapper .NET » ou « Adaptateur .NET ». Il convertit l'interface d'une classe en une autre interface que le client attend. L'Adaptateur fait fonctionner un ensemble de classes qui n'auraient pu fonctionner sans lui, à cause d'une incompatibilité d'interface. En effet, dans notre cas la « DLL » est écrite dans un code non managé et ne fonctionne que sur la plateforme pour laquelle elle a été compilée alors que le C# est un code managé. Il fonctionne sur tout support où la machine virtuelle du .NET est installée. La compilation ne donne pas un programme binaire mais un langage intermédiaire compris seulement par cette machine virtuelle.

Voici la liste des principaux éléments de cette librairie gérant la couche physique:

- La méthode **Command()** permet d'envoyer les ordres à la classe. Ces ordres sont de trois types :
  - ✓ « Start » pour démarrer et initialiser la classe.
  - ✓ « Stop » pour arrêter l'ensemble des threads créés par cette classe et fermer les ports CAN ouverts.
  - ✓ « Pause » pour stopper l'affichage des trames à l'écran.
- Le thread **RXThread()** a pour but d'écouter ce qui se passe sur le bus CAN et de transmettre ces informations à la méthode **Receive().**
- La méthode **Receive()** segmente les trames reçues et les range dans des N_PDU.
- La méthode **Transmit()** reçoit les N_PDU de la classe « Link_Layer », qu'elle segmente dans des trames CAN puis les transmets sur le bus de communication.

5.3.1.2 Librairie UDS

En plus de la première DLL utilisant les couches physiques du modèle OSI, une deuxième DLL est développée durant le stage. Elle est également écrite en C et nécessite un adaptateur. Elle doit contenir les fonctions utilisées par les différentes couches hautes de l' « UDS ». L'intérêt de cette création réside dans le fait de sa portabilité future. Elle pourra être utilisée directement dans d'autres programmes en C, en C# ou en Lab Windows. Cela permet un gain de temps important lors de développements futurs pour d'autres applications.

Cette librairie contient les mêmes fonctions que celles du serveur au niveau de la couche ISO15765-2. Elles sont en général très proches en termes d'écriture. Les paramètres quand à eux, sont parfois différents pour permettre au logiciel C# de récupérer les informations nécessaires à son fonctionnement. Ils envoient par exemple ces informations sur le bus CAN par le biais de la seconde DLL. Au niveau de la couche application, les fonctions sont en générales différentes car elles se complémentent. Par exemple, pour le service « TesterPresent » le serveur possède la requête et la confirmation tandis que le client possède l'indication.

La librairie contient 2 fichiers .C pour l'ISO15765-2 et l'ISO15765-3 ainsi que les 2 fichiers header avec la définition des structures. Pour faire fonctionner cette librairie avec le programme en C#, il faut concevoir un « wrapper ». Cet adaptateur est en réalité une classe avec les mêmes définitions de structures que la librairie mais adaptés au .NET. La classe possède aussi les prototypes des fonctions de la « DLL ». Ensuite, il est possible d'utiliser les fonctions de la librairie sans problème.

5.3.2 Interface graphique

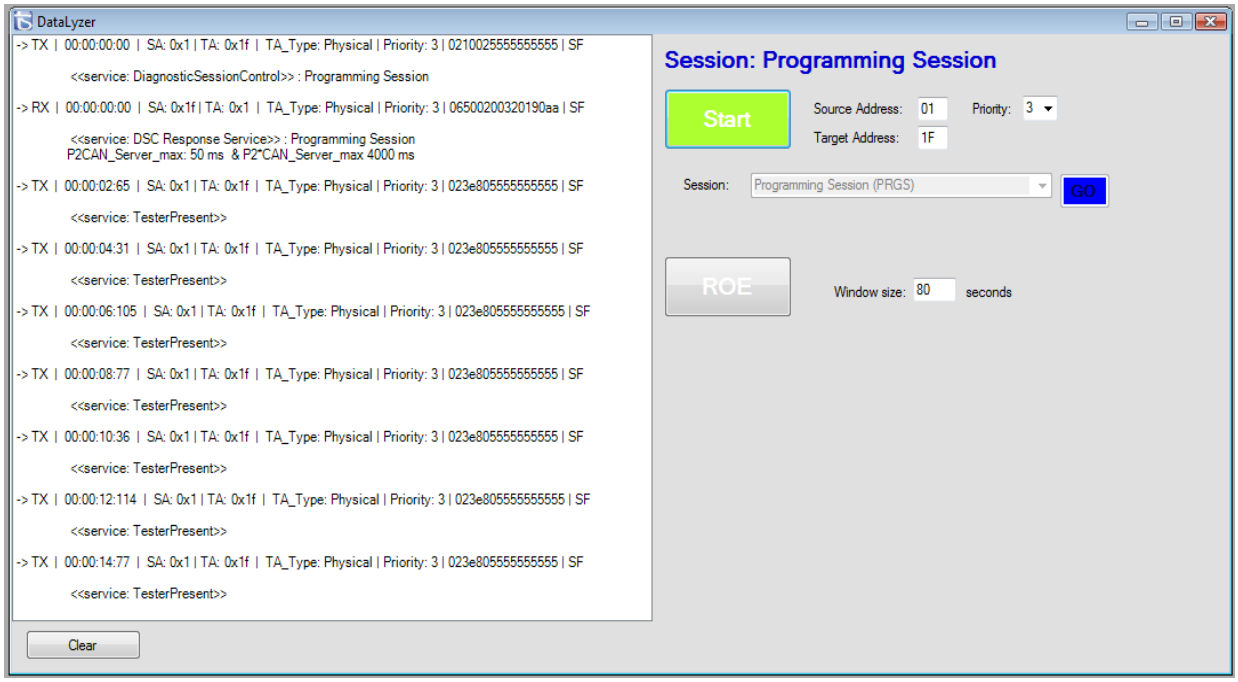


Figure 48 IHM de l'outil de diagnostic

Plusieurs fenêtres sont utilisées dans le logiciel permettant à l'utilisateur de communiquer avec l'outil de diagnostic. La principale permet d'utiliser les services UDS déjà implémentés (cf. figure 48). Il est possible de choisir les adresses de la cible et de la source ainsi que le niveau de priorité des messages CAN. En appuyant sur le bouton « start », la session de diagnostic commence. Elle débute en session par défaut. Il est possible de choisir la session de son choix via un menu déroulant. Le service « TesterPresent » est automatiquement lancé suivant la session en cours à intervalle régulier. Enfin, il est possible d'ouvrir une fenêtre temporelle essentielle au service « ResponseOnEvent ».

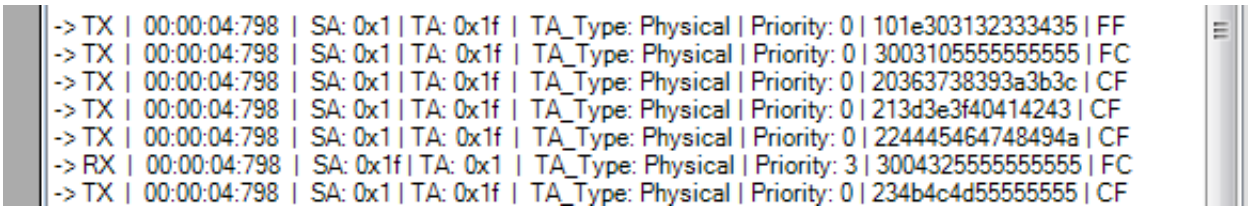


Figure 49 Visualisation de trames échangées

Les trames échangées sont visibles depuis une zone de texte où plusieurs informations sont présentes (cf. figure 49 ci-dessus):

- Le sens de l'échange : transmission (« TX ») ou réception (« RX »).
- Le moment de l'échange depuis l'ouverture du diagnostic.
- Les adresses sources et cibles ainsi que le type d'adressage physique ou fonctionnel.
- Le niveau de priorité.
- Les données brutes du message CAN.
- Le type de la trame CAN : « SF », « FF », « CF » ou « FC ».
- Le service UDS utilisé avec la traduction de la signification de ces données brutes.

Le logiciel est à ce jour fonctionnel avec des services UDS déjà implémentés. Il est possible facilement d'y implémenter de nouveaux services en modifiant la librairie dynamique et en ajoutant des fonctionnalités à l'interface graphique.

CONCLUSION

L'objectif du stage était la Réalisation d'une maquette de test permettant de simuler du diagnostic véhicule via le protocole ISO 14229. Il m'a permis d'utiliser de nombreuses compétences acquises durant mes années d'études. Tout d'abord en consolidant mes connaissances dans le domaine de l' « hardware » avec la conception complète du typon et du dimensionnement électrique et mécanique des composants. Ensuite, j'ai complété mes compétences en programmation en C avec la découverte d'un nouveau type de microcontrôleur et une nouvelle façon de voir l'architecture d'un code. Et enfin, j'ai également amélioré mes connaissances des langages orientés objets tel que le C#.

Ce stage m'a surtout permis d'obtenir des connaissances essentielles sur les systèmes embarqués dans le milieu automobile, d'approfondir mes connaissances du bus CAN et de découvrir le fonctionnement d'un outil de diagnostic automobile. J'ai à ce jour une vision beaucoup plus claire du fonctionnement globale de l'architecture électrique dans un véhicule avec les différents types de bus ou calculateurs présents.

Ce stage de fin d'étude est le plus enrichissant et le plus abouti des stages de ma formation. En effet, j'ai mobilisé la quasi totalité de mes connaissances techniques dispensées à l'INSA de Strasbourg. D'autre part, il a permis la découverte de l'environnement automobile, les responsabilités et les contraintes liées au travail d'un ingénieur et l'ouverture vers un monde technologique vaste et passionnant.

Je tiens à remercier la société Technology&Strategy pour m'avoir accueilli durant ce stage de fin d'étude et durant le projet en préambule de ce stage. Je voudrais particulièrement remercier 2 personnes : Matthieu ROTH, le responsable du laboratoire ainsi que Sébastien JULIEN, consultant pour T&S qui ont suivis l'avancée de mon projet du début à la fin. Ils m'ont donné des conseils techniques pour permettre la progression de mon travail. Enfin, je tiens à remercier les autres personnes qui m'ont entourées notamment les 6 autres stagiaires avec qui une entraide technique s'est créée, ainsi que Jéremie HUSS, le cogérant de l'entreprise qui m'a accueilli chez T&S et l'ensemble des interlocuteurs que j'ai pu rencontrer au sein de l'entreprise.

BIBLIOGRAHIE

- Specification ISO/DIS 15765-2: Road vehicles – Diagnostics on CAN – Part 2: Network layer services.

- Specification ISO 15765-3: Road vehicles - Diagnostics on Controller Area Networks (CAN) - Part 3: Implementation of unified diagnostic services (UDS on CAN).

- Specification ISO 14229: Road vehicles — Unified diagnostic services (UDS) — Specification and requirements.

- « Diag on CAN dossier de développement » : fournit par l'ancien stagiaire.

- « Diagnostic Communication of Vehicles » de Bosch.

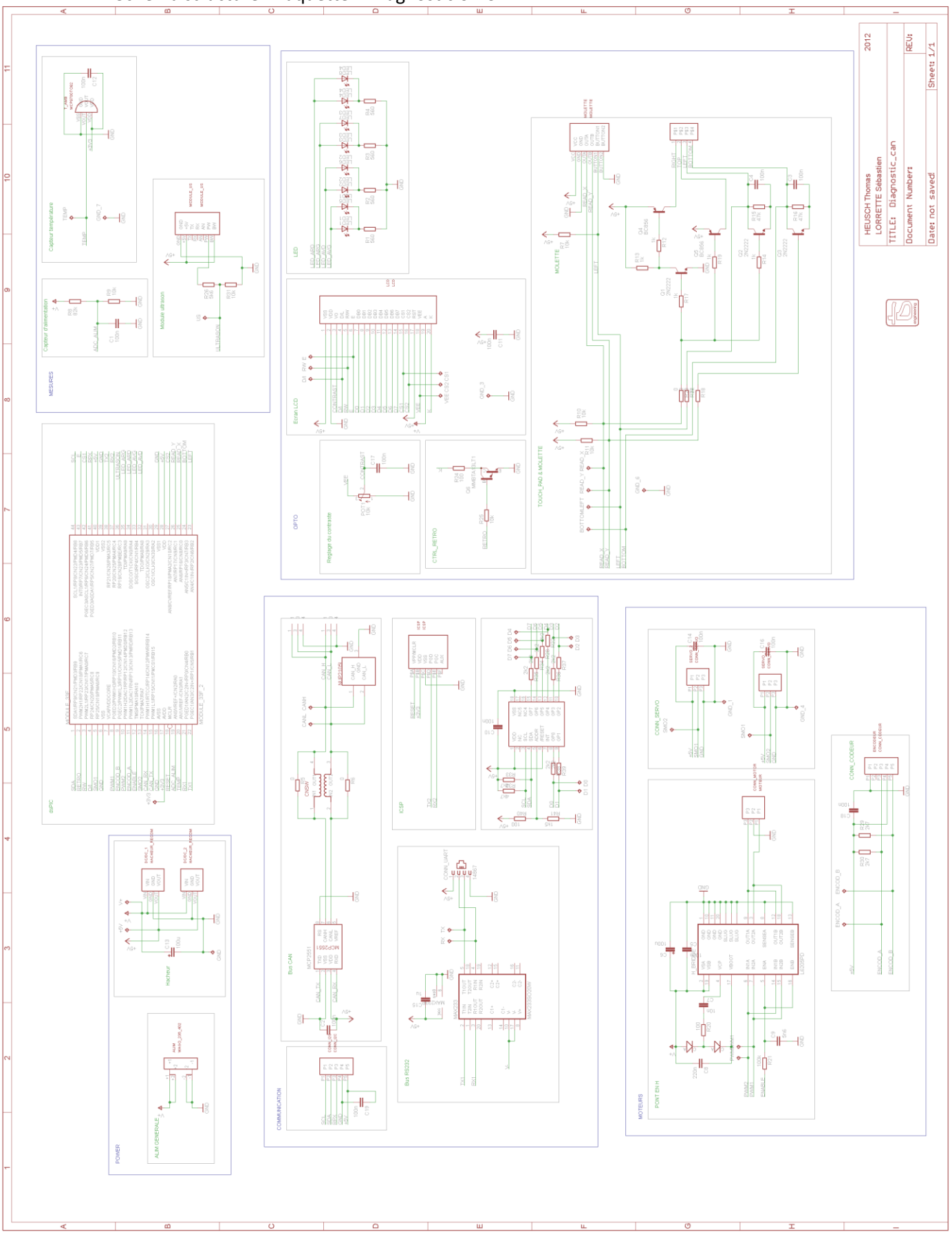
- « le diagnostic dans les architectures électroniques embarquées PSA » : Formation NSI.

- « Unified Diagnostic Services (UDS) – ISO14229 »: poster publié par "Softing"

- « La langage C » du « research & development LAB » de T&S : Coding rules concernant la programmation en langage C.

ANNEXE

1. Schéma structurel maquette « Diagnostic on CAN »



2. Liste composants maquette « Diagnostic on CAN »

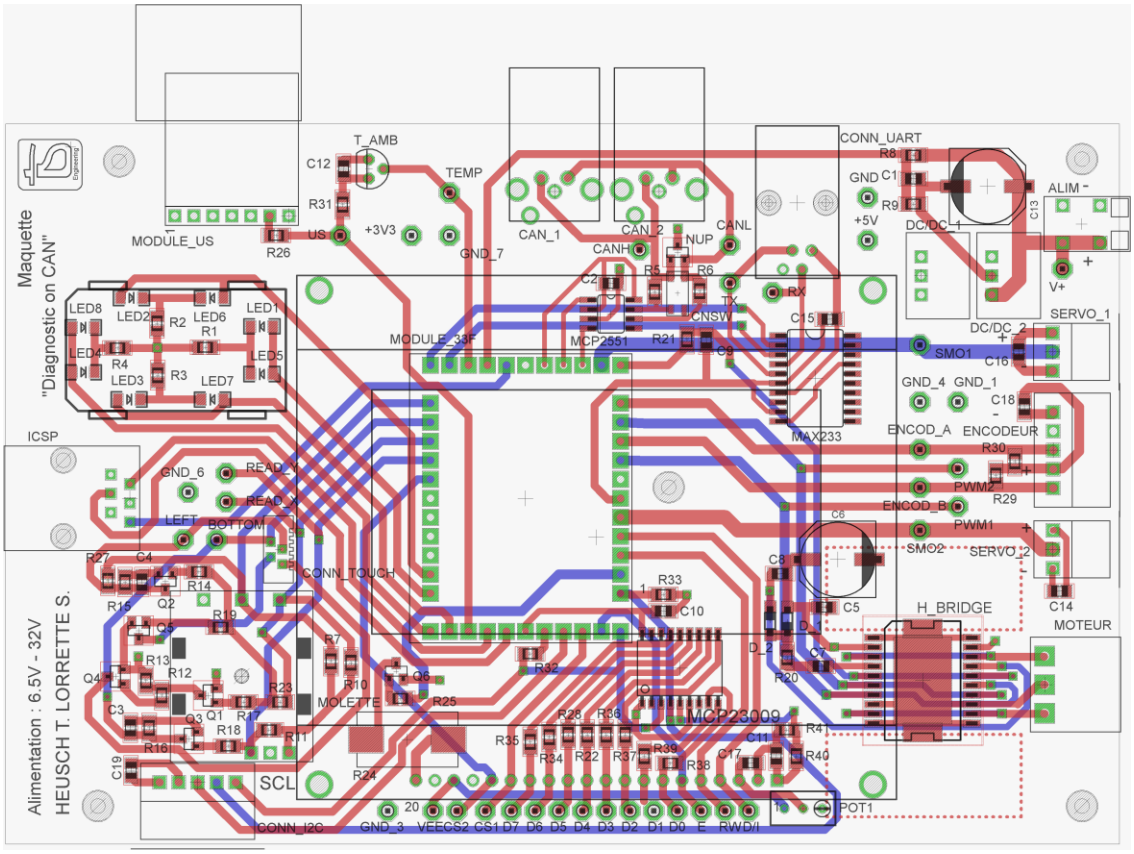
Ref. eagle	Valeur	nom eagle	Package	Ref. fourn.	Fournisseur	Ref. fabricant	Fabricant	Prix
POWER								
ALIM GENERALE								
ALIM	-	WAGO_236_402	WAGO_236_402	1283526	Farnell	236-402	WAGO	0,61
Hacheur								
C13	100u	C	CAP_PANASONIC-TG_BOITIER_G	568-553	Radiospares	EEETG1H101UP+	Panasonic	1,22
DC/DC_1	-	HACHEUR_RECOM	HACHEUR_RECOM	163-451	Radiospares	R-78B5.0-1.0	Recom	9,18
DC/DC_2	-	HACHEUR_RECOM	HACHEUR_RECOM	163-451	Radiospares	R-78B5.0-1.0	Recom	9,18
dsPIC								
IC4	-	MODULE_33F_2	MODULE_33F	-	-	-	-	
OPTO								
Ecran LCD								
LCD	-	LCD-G12864FULL	LCD-G12864	A/LCD12864	Lextronic	-	-	30
C11	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
Réglage du contraste								
POT1	22k	R-TRIMM3296W	RTRIM3296W	522-0293	Radiospares	3299W-1-103LF	Bourns	1,28
C17	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
CTRL_RETRO								
R24	150	R-EU_	POWER_RESISTOR	664-1654	Radiospares	SMW5150RJT	TE Connectivity	2,26
Q6	-	2N2222	SOT23	485-4586	Radiospares	PMBT2222A	Infineon	0,01
R25	47K	C-EUC0805	C0805	721-7889	Radiospares	ERJP06F4702V	Panasonic	0,01
LEDs								
R1	560	R-EU_M0805	M0805	153-668	Radiospares	ERJP06J561V	Panasonic	0,01
R2	560	R-EU_M0805	M0805	153-668	Radiospares	ERJP06J561V	Panasonic	0,01
R3	560	R-EU_M0805	M0805	153-668	Radiospares	ERJP06J561V	Panasonic	0,01
R4	560	R-EU_M0805	M0805	153-668	Radiospares	ERJP06J561V	Panasonic	0,01
LED1	-	LED1206	LED-1206	156314	Conrad	15-21SURC/S530-A2/TR8	Everlight	0,2
LED2	-	LED1206	LED-1206	156314	Conrad	15-21SURC/S530-A2/TR8	Everlight	0,2
LED3	-	LED1206	LED-1206	156314	Conrad	15-21SURC/S530-A2/TR8	Everlight	0,2
LED4	-	LED1206	LED-1206	156317	Conrad	15-21UYC/S530-A2/TR8	Everlight	0,2
LED5	-	LED1206	LED-1206	156314	Conrad	15-21SURC/S530-A2/TR8	Everlight	0,2
LED6	-	LED1206	LED-1206	156314	Conrad	15-21SURC/S530-A2/TR8	Everlight	0,2
LED7	-	LED1206	LED-1206	156314	Conrad	15-21SURC/S530-A2/TR8	Everlight	0,2
LED8	-	LED1206	LED-1206	156317	Conrad	15-21UYC/S530-A2/TR8	Everlight	0,2
Touch pad & Molette								
DALLE_TACTILE	-	-	-	A/TACT1	Lextronic	-	-	11
MOLETTE	-	MOLETTE	PKG_MOLETTE	1703831	Farnell	62P22-H4	Grayhill	8,68
R7	10k	R-EU_M0805	M0805	153-797	Radiospares	ERJP06J103V	Panasonic	0,01
R10	10k	R-EU_M0805	M0805	153-797	Radiospares	ERJP06J103V	Panasonic	0,01
R11	10k	R-EU_M0805	M0805	153-797	Radiospares	ERJP06J103V	Panasonic	0,01
R12	1k	R-EU_M0805	M0805	153-781	Radiospares	ERJP06J102V	Panasonic	0,01
R13	1k	R-EU_M0805	M0805	153-781	Radiospares	ERJP06J102V	Panasonic	0,01
R14	1k	R-EU_M0805	M0805	153-781	Radiospares	ERJP06J102V	Panasonic	0,01
R15	47k	R-EU_M0805	M0805	154-071	Radiospares	ERJP06J473V	Panasonic	0,01

R16	47k	R-EU_M0805	M0805	154-071	Radiospares	ERJP06J473V	Panasonic	0,01
R17	1k	R-EU_M0805	M0805	153-781	Radiospares	ERJP06J102V	Panasonic	0,01
R18	1k	R-EU_M0805	M0805	153-781	Radiospares	ERJP06J102V	Panasonic	0,01
R19	1k	R-EU_M0805	M0805	153-781	Radiospares	ERJP06J102V	Panasonic	0,01
R23	0	R-EU_M0805	M0805	223-0146	Radiospares	CRG0805ZR	TE Connectivity	0,01
R27	0	R-EU_M0805	M0805	223-0146	Radiospares	CRG0805ZR	TE Connectivity	0,01
C3	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
C4	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
Q1	-	2N2222	SOT23	485-4586	Radiospares	PMBT2222A	Infineon	0,01
Q2	-	2N2222	SOT23	485-4586	Radiospares	PMBT2222A	Infineon	0,01
Q3	-	2N2222	SOT23	485-4586	Radiospares	PMBT2222A	Infineon	0,01
Q4	-	BC856	SOT23	445-2045	Radiospares	BC856B	Infineon	0,01
Q5	-	BC856	SOT23	445-2045	Radiospares	BC856B	Infineon	0,01
Moteurs								
Pont en H								
U1	-	L6205PD	SO20POWER	714-0644	Radiospares	L6205PD	STMicroelectronics	11,18
R20	100	R-EU_M0805	M0805	153-804	Radiospares	ERJP06J104V	Panasonic	0,01
R21	100k	R-EU_M0805	M0805	153-775	Radiospares	ERJP06J101V	Panasonic	0,01
C5	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
C6	100u	C	CAP_PANASONIC-TG_BOITIER_G	568-553	Radiospares	EEETG1H101UP+	Panasonic	1,23
C7	10n	C-EUC0805	C0805	624-2569	Radiospares	GRM216R71H103JA01D	Murata	0,01
C8	220n	C-EUC0805	C0805	723-6385	Radiospares	GRM219R71E224K	Murata	0,01
C9	5n6	C-EUC0805	C0805	723-6420	Radiospares	GRM2195C1H562J	Murata	0,01
MOTEUR	-	CONN_MOTOR	CONN_MOTOR	1121788	Farnell	SL 3.5/3/180G	WEIDMULLER	0,66
D1	-	1N4148	SOD123	700-3671	Radiospares	1N4148W-V-GS08	Vishay	0,01
D2	-	1N4148	SOD123	700-3671	Radiospares	1N4148W-V-GS08	Vishay	0,01
CONN_SERVO								
C14	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
C16	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
CONN_SERVO	-	CONN_SERVO	CONNN_SERVO	741221	Conrad	44.805	-	1,5
CONN_SERVO1	-	CONN_SERVO	CONNN_SERVO	741221	Conrad	44.805	-	1,5
CONN_CODEUR								
R29	2K7	R-EU_M0805	M0805	721-7788	Radiospares	ERJP06F2701V	Panasonic	0,01
R30	2K7	R-EU_M0805	M0805	721-7788	Radiospares	ERJP06F2701V	Panasonic	0,01
C18	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
CONN_CODEUR	-	CONN_CODEUR	CONN_CODEUR	741230	Conrad	44.806	-	1,75
Communication								
Bus CAN								
R5	0	R-EU_M0805	M0805	223-0146	Radiospares	CRG0805ZR	TE Connectivity	0,01
R6	0	R-EU_M0805	M0805	223-0146	Radiospares	CRG0805ZR	TE Connectivity	0,01
C2	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01

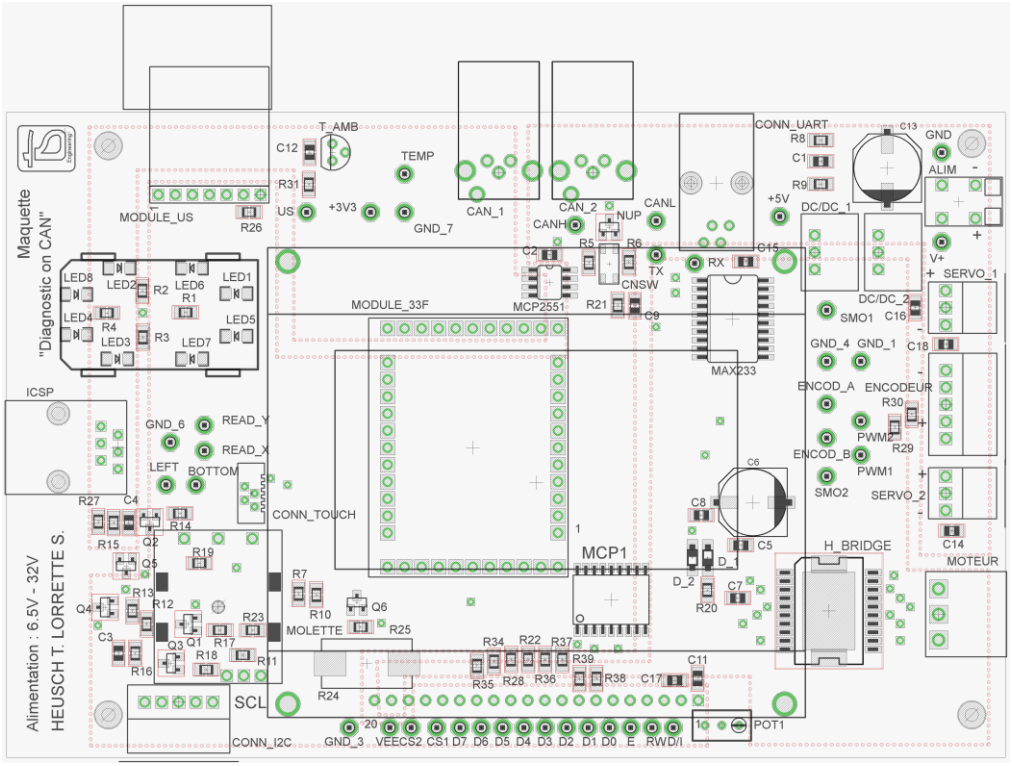
IC1	-	MCP2551/SN	SO08	402-904	Radiospares	MCP2551-I/SN	Microchip	1,09
NUP	-	NUP2105L	SOT23	464-255	Radiospares	NUP2105LT1G	ON Semiconductor	0,51
CNSW	-	CNSW	CNSW	500-3076	Radiospares	744232102	Wurth Elektronik	0,93
CAN_1	-	CONN_CAN	M8	-	-	-	-	-
CAN_2	-	CONN_CAN	M8	-	-	-	-	-
Bus RS232								
C15	1u	C-EUC0805	C0805	723-6067	Radiospares	GRM21BR71C105K	Murata	0,01
IC3	-	MAX233	SO-20W	732-8951	Radiospares	MAX233ACWP+G36	Maxim	9,34
CONN_UART	-	14867	14867	615-4282	Radiospares	5520257-2	TE Connectivity	3,33
ICSP								
ICSP	-	ICSP	ICSP	615-4260	Radiospares	1-1705950-1	Tyco	3,23
I2C								
R1	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R2	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R3	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R4	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R22	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R28	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R32	4K7	R-EU_M0805	M0805	721-7801	Radiospares	ERJP06F4701V	Panasonic	0,01
R33	4K7	R-EU_M0805	M0805	721-7801	Radiospares	ERJP06F4701V	Panasonic	0,01
R34	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R35	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R38	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R39	2K2	R-EU_M0805	M0805	566-474	Radiospares	ERA6AEB222V	Panasonic	0,01
R40	100	R-EU_M0805	M0805	153-804	Radiospares	ERJP06J104V	Panasonic	0,01
R41	1K5	R-EU_M0805	M0805	721-7766	Radiospares	ERA6APB152P	Panasonic	0,01
C10	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
C19	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
MCP23009	-	MCP23009	SOIC18-W	687-8492	Radiospares	MCP23009-E/SO	Microchip	1,28
CONN_I2C	-	CONN_CODEUR	CONN_CODEUR	741230	Conrad	44.806	-	1,75
MESURES								
Capteur d'alimentation								
R8	820	R-EU_M0805	M0805	721-7744	Radiospares	ERJP06F8200V	Panasonic	0,01
R9	100	R-EU_M0805	M0805	153-804	Radiospares	ERJP06J104V	Panasonic	0,01
C1	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
Capteur température								
C12	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
T_AMB	-	MCP9700	TO-92	403-838	Radiospares	MCP9700A-E/TO	Microchip	0,27
Module ultrason								
MODULE_US	-	MODULE_US	MODULE_US	MS-EZ1	Lextronic	-	-	24,5
R26	5K6	R-EU_M0805	M0805	742-5942	Radiospares	ERA6AEB562V	Panasonic	0,01
R31	10K	R-EU_M0805	M0805	153-797	Radiospares	ERJP06J103V	Panasonic	0,01
								139,73

3. Typon maquette « diagnostic on CAN »

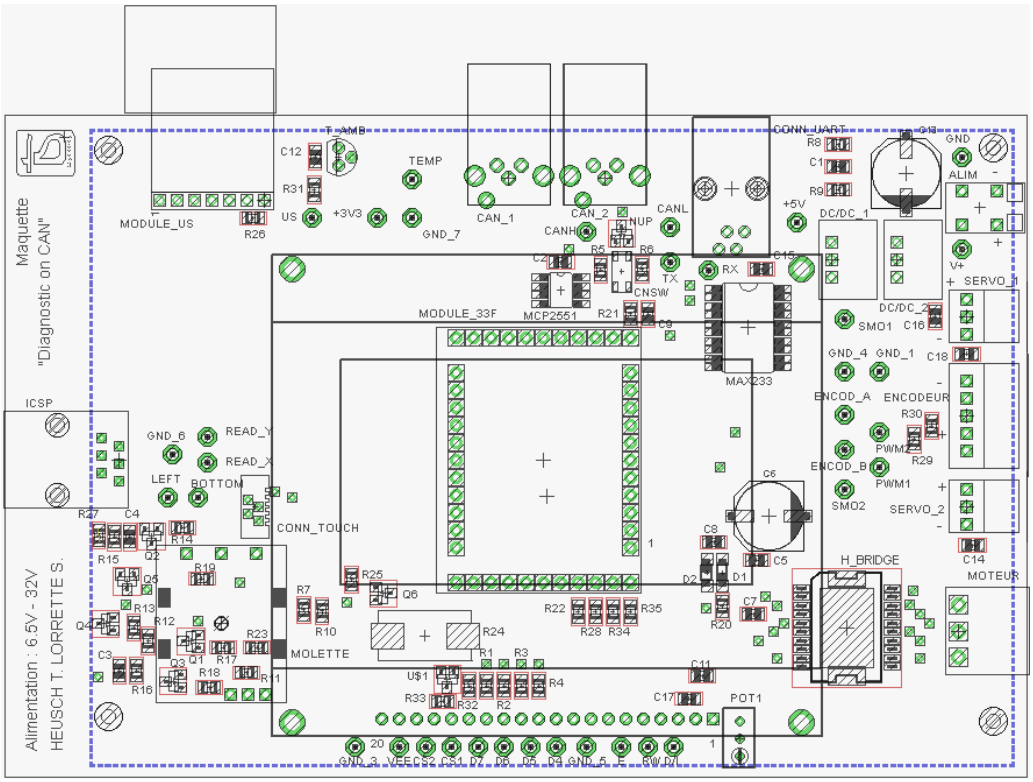
➢ TOP et BOTTOM



➢ Alimentation



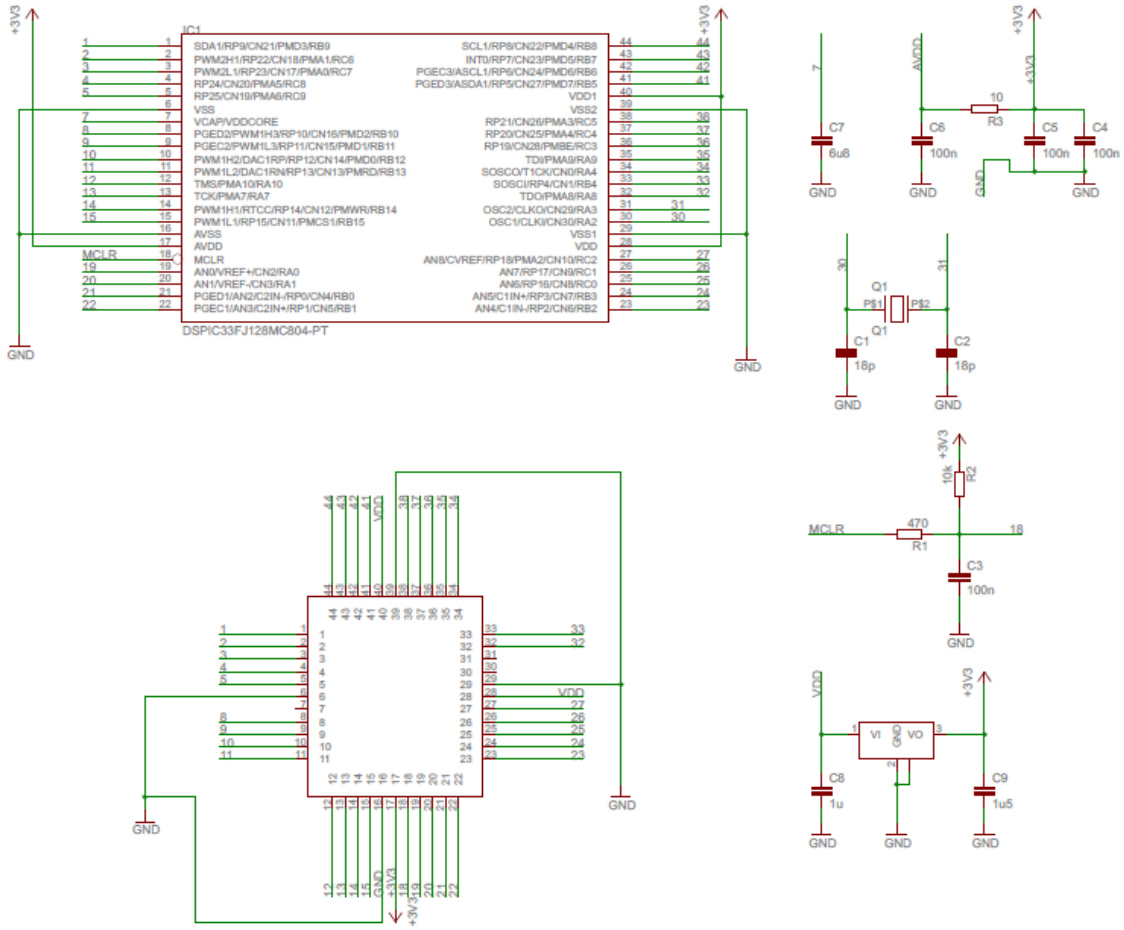
- Masse


Maquette
"Diagnostic on CAN"
Alimentation : 6.5V - 32V
HEUSCH T. LORRETTE S.
MODULE_US
T_AMB
TEMP
US
+3V3
GND_7
CAN_1
CAN_2
CANL
CANH
CANUP
MODULE_33F
MCP2551
CNSW
TX
RX
MAX233
CONN_UART
DC/DC_1
DC/DC_2
+5V
GND
ALIM
V+
SERVO_1
SERVO_2
SMO1
SMO2
GND_4
GND_1
ENCOD_A
ENCODEUR
ENCOD_B
PWM2
PWM1
ICSP
GND_6
READ_Y
READ_X
LEFT
BOTTOM
CONN_TOUCH
MOLETTE
H_BRIDGE
MOTEUR
POT1

4. Brochage microcontrôleur DsPIC 33FJ128MC804

numero pin	Nom de la broche	Entree/sortie	Analogic pin	Remappable pin		
1	SDA1/RP9(1)/CN21/PMD3/RB9	in/out		Rp9	SDA	Signal de donnée I2C
2	PWM2H1/RP22(1)/CN18/PMA1/RC6	out		Rp22	RETRO	Asservissement retroeclairage
3	PWM2L1/RP23(1)/CN17/PMA0/RC7	out		Rp23	RW	RW Read/write LCD Screen
4	RP24(1)/CN20/PMA5/RC8	out		Rp24	D/I	D/I Data/instruction LCD Screen
5	RP25(1)/CN19/PMA6/RC9	out		Rp25	SMO1	Servomoteur Output1
6	VSS	-			0v	-
7	VCAP	-			VCAP	-
8	PGED2/PWM1H3/RP10(1)/CN16/PMD2/RB10	out		Rp10	PWM1	PWM1 Moteur
9	PGEC2/PWM1L3/RP11(1)/CN15/PMD1/RB11	out		Rp11	QEB1	Module QEI1 phase B
10	PWM1H2/DAC1RP/RP12(1)/CN14/PMD0/RB12	in		Rp12	PWM2	PWM2 Moteur
11	PWM1L2/DAC1RN/RP13(1)/CN13/PMRD/RB13	out		Rp13	QEA1	Module QEI1 phase A
12	TMS/PMA10/RA10	out			ENABLE	Enable PWM
13	TCK/PMA7/RA7	out			SMO2	Servomoteur Output2
14	PWM1H1/DAC1LP/RTCC/RP14(1)/CN12/PMWR/RB14	out		Rp14	C1RX	Module ECAN reception
15	PWM1L1/DAC1LN/RP15(1)/CN11/PMCS1/RB15	out		Rp15	C1TX	Module ECAN transmission
16	AVSS	-			0v	-
17	AVDD	-			3,3v	-
18	MCLR	in			MCLR	-
19	AN0/VREF+/CN2/RA0	in	An0		Valim_rec	Module ADC
20	AN1/VREF-/CN3/RA1	in	An1		TEMP	Capteur de température
21	PGED1/AN2/C2IN-/RP0(1)/CN4/RB0	in	An2	Rp0	U1RX	Module UART1 reception MAX233
22	PGEC1/AN3/C2IN+/RP1(1)/CN5/RB1	out	An3	Rp1	U1TX	Module UART1 transmission MAX233
23	AN4/C1IN-/RP2(1)/CN6/RB2	in	An4	Rp2	LEFT	Touchpad Left
24	AN5/C1IN+/RP3(1)/CN7/RB3	out	An5	Rp3	BOTTOM	Touchpad Bottom
25	AN6/DAC1RM/RP16(1)/CN8/RC0	out	An6	Rp16	Read-X	READ-X Touch-panel
26	AN7/DAC1LM/RP17(1)/CN9/RC1	out	An7	Rp17	Read-Y	READ-Y Touch-panel
27	AN8/CVREF/RP18(1)/PMA2/CN10/RC2	out	An8	Rp18	CS2	Chip select LCD 2
28	VDD	-			3,3v	-
29	VSS	-			0v	-
30	OSC1/CLKI/CN30/RA2	out			OSC1	Quartz
31	OSC2/CLKO/CN29/RA3	out			OSC2	Quartz
32	TDO/PMA8/RA8	out			LED1	LEDs DROITE
33	SOSCI/RP4(1)/CN1/RB4	out		Rp4	LED2	LEDs ARRIERE
34	SOSCO/T1CK/CN0/RA4	out			LED3	LEDs AVANT
35	DI/PMA9/RA9	out			LED4	LEDs GAUCHE
36	RP19(1)/CN28/PMBE/RC3	in		Rp19	Ultrason	Module ultrason
37	RP20(1)/CN25/PMA4/RC4	out		Rp20	U2RX	ICSP RX
38	RP21(1)/CN26/PMA3/RC5	out		Rp21	U2TX	ICSP TX
39	VSS	-			0v	-
40	VDD	-			3,3v	-
41	PGED3/ASDA1/RP5(1)/CN27/PMD7/RB5	out		Rp5	RP	Liaison I2C
42	PGEC3/ASCL1/RP6(1)/CN24/PMD6/RB6	out		Rp6	CS1	Chip select LCD 1
43	INT0/RP7(1)/CN23/PMD5/RB7	out		Rp7	E	Enable LCD
44	SCL1/RP8(1)/CN22/PMD4/RB8	in/out		Rp8	SCL	Signal d'horloge I2C

5. Schéma structurel platine DsPIC

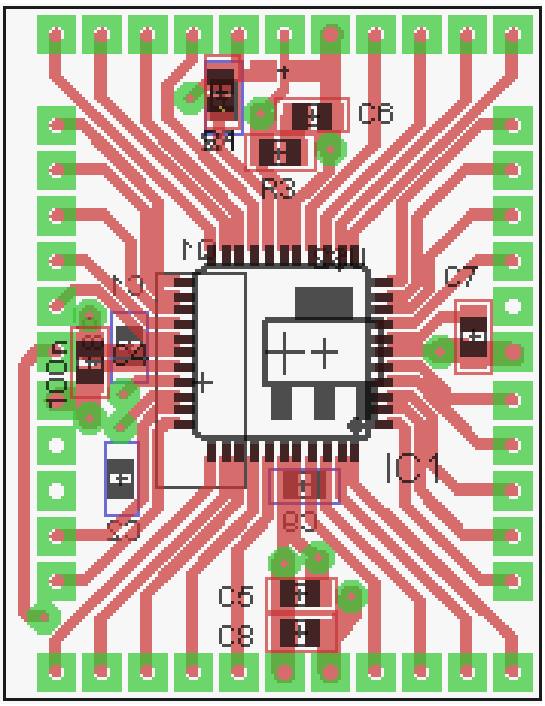


6. Liste composants platine DsPIC

Ref. eagle	Valeur	nom eagle	Package	Ref. fourn.	Fournisseur	Ref. fabricant	Fabricant	Prix(€)
Platine DsPIC								
C1	18p	C-EUC0805	C0805	723-6228	Radiospares	GRM2165C1H180JZ01D	Murata	0,01
C2	18p	C-EUC0805	C0805	723-6228	Radiospares	GRM2165C1H180JZ01D	Murata	0,01
C3	100n	CAP	805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
C4	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
C5	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
C6	100n	C-EUC0805	C0805	723-5058	Radiospares	GCM21BR72A104KA37L	Murata	0,01
C7	6u8	C-EUC0805	C0805	691-1192	Radiospares	C0805C685K8PAC7800	Kemet	0,01
C8	1u	C-EUC0805	C0805	723-6067	Radiospares	GRM21BR71C105K	Murata	0,01
C9	1u5	C-EUC0805	C0805	619-8030	Radiospares	LLA219R70J155MA01L	Murata	0,01
IC1	-	DSPIC33FJ128MC804-PT	TQFP44	666-8378	Radiospares	dsPIC33FJ128MC804-I/PT	Microchip	6,37
Q1	-	XTAL	QUARTZ	-	-	-	-	
R1	470	R-EU_R0805	R0805	154-150	Radiospares	ERJP06J471V	Panasonic	0,01
R2	10K	R-EU_R0805	R0805	153-797	Radiospares	ERJP06J103V	Panasonic	0,01
R3	10	R-EU_R0805	R0805	153-769	Radiospares	ERJP06J100V	Panasonic	0,01
U$2	LT1521-3	LT1521-3	SOT223	545-6672	Radiospares	LT1521CST-5#PBF	Linear Technology	4,43
								10,92

7. Typon platine DsPIC

➢ Top



➢ Bottom