JAILLOT Etienne

Etudiant en 5ème année
Génie Electrique Option Systèmes
INSA Strasbourg

Du 30 Janvier au 3 Août 2012

Rapport de Projet de Fin d'Etudes
Combi Instrument CAN

Lieu de stage :
Technology & Strategy
4 Avenue de la paix
67000 STRASBOURG

Responsable en entreprise : Mr Matthieu Roth
Responsable pédagogique : Mr Bertrand Boyer

Cahier des charges du projet

Le cachier des charges se décompose en deux parties :

- 1ère partie : conception et réalisation hardware :
  - Etude du microcontrôleur dsPIC33E et de ses fonctionnalités.
  - Dimensionnement et choix des composants.
  - Plage de fonctionnement de 7 à 18V.
  - Réalisation d'une carte prototypage regroupant l'ensemble des fonctionnalités.

- 2ème partie : réalisation software
  - Etude de la spécification CAN2.0.
  - Réalisation de la programmation microcontrôleur en C permettant de :
    - Générer des trames CAN de contrôle du tableau de bord.
    - Acquérir et traiter des informations analogiques des pédales et des capteurs.
    - Gérer une communication USB 2.0.
    - Gérer une communication série.
  - Réalisation d'une IHM (interface homme-machine) en C# communiquant avec la carte par USB et permettant d'allumer les voyants du tableau de bord, d'accélérer, de freiner et d'afficher la vitesse et le régime moteur.
  - Réalisation d'une application Androïd en Java destinée à contrôler les voyants du tableau de bord, ainsi que l'accélération et le freinage via une communication Bluetooth.
  - Gestion de l'interactivité et du passage d'un mode de commande à l'autre.

PROJET DE FIN D'ETUDES

Auteur : Etienne Jaillot	Promotion : GE5 S 2011-2012
Titre : Combi Instrument CAN	Soutenance : 20 septembre 2012

Structure d'accueil :

T&S Engineering
4 avenue de la Paix
67000 Strasbourg

Nb de volume(s) : 1 Nb de pages : 55

Nb de références bibliographiques : 12

Résumé :

J'ai effectué mon stage dans le laboratoire de Recherche & Développement chez T&S Engineering. Au sein de ce laboratoire sont développés des projets majoritairement internes dans le but de capitaliser des connaissances et compétences dans le domaine des systèmes embarqués. L'objectif de mon projet est de réaliser une maquette de démonstration permettant de piloter un tableau de bord de Peugeot 207 RC par le CAN. Le tableau de bord peut être commandé de différentes manières : soit par comodo et pédalier (accélérateur et frein), soit à partir d'une application Androïd communiquant en Bluetooth, soit à partir du PC via une communication USB. La réalisation du projet comprend deux parties majeures : le développement hardware et software. La carte électronique alors réalisée sert d'interface entre les différents modes de commande et le tableau de bord : elle réalise l'acquisition des données de commande et génère les trames CAN correspondantes pour piloter le tableau de bord.

Mots clés :

CAN, USB, dsPIC33E, Bluetooth

Traduction :

Titel:

Combi Instrument CAN

Ich habe mein Praktikum in dem R&D Labor bei T&S Engineering durchgeführt. Inmitten dieses Labors werden hauptsächlich inneren Projekten entwickelt, um Kapital aus Kenntnissen und Kompetenzen in den Embedded Systems zu schlagen. Das Ziel meines Projektes ist, ein Demonstrationsmodell zu entwickeln, das ermöglicht, ein Kombi durch den CAN-Bus zu steuern. Dieser Kombi kann auf drei verschiedene Arten gesteuert werden: entweder durch das Gaspedal, die Bremse, den Lenkstockschalter, oder durch eine Androïd-Schnittstelle (Bluetooth-Kommunikation), oder durch eine Computerschnittstelle (USB-Verbindung). Die Durchführung dieses Projektes besteht aus zwei Hauptteilen: Hardware und Software Design. Die hergestellte elektronische Karte dient als Schnittstelle zwischen den unterschiedlichen Steuerungsarten und dem Kombi: sie erfüllt die Verarbeitung der Daten und sendet die entsprechenden CAN-Frames, um den Kombi zu steuern.

Sommaire

Remerciements

Le projet de fin d'étude décrit dans ce mémoire marque la fin de ma formation Génie Electrique à l'INSA de Strasbourg, c'est pourquoi je souhaite remercier tout l'encadrement enseignant et l'administration de la section Génie Electrique, sans qui ce stage n'aurait été rendu possible.

Plus particulièrement, je tiens à remercier mon tuteur de stage Mr. Bertrand BOYER, pour son enseignement, son suivi et ses conseils pédagogiques réguliers aussi bien en amont qu'au cours de cette période de stage.

Mes remerciements vont également à mon maître de stage en entreprise, Matthieu ROTH, pour son accompagnement tout au long du stage et ses conseils précieux. Je remercie aussi Sébastien JULIEN pour son aide concernant notamment le développement Hardware. Je remercie Adeline CLAUSS, qui fut notre première interlocutrice en début de stage et qui a su nous guider pendant cette période.

Je souhaite également remercier l'ensemble du personnel de Technology & Strategy à savoir la direction, les managers, le service des ressources humaines et l'administration pour leur accueil, leur disponibilité, l'intérêt qu'ils ont porté à nos travaux et leur implication dans notre réussite. L'opportunité qui m'a été offerte de travailler sur un projet industriel pour un client a été un plus à ce projet de fin d'étude, me permettant de mettre en application les compétences acquises pendant le début du projet et d'observer le métier de consultant T&S sur site client, pour cela, je souhaite également remercier Patrice PAULUS et Pascal TOURENNE, les coordinateurs de ce projet.

Enfin, je souhaite remercier l'ensemble des stagiaires présents pendant cette période, pour la bonne ambiance et l'entraide qui ont régnées tout au long du stage.

Introduction

Mon stage de fin d'étude s'est déroulé sur une durée de six mois au sein de T&S Engineering, entité du groupe Technology & Strategy, entreprise de consulting dans les domaines de l'électronique embarquée et de l'informatique décisionnelle. C'est dans ce cadre que j'ai effectué la réalisation d'une maquette de démonstration permettant de piloter un tableau de bord de Peugeot 207 RC de différentes manières interactives, en mettant en application différentes techniques de communication, dont une désormais très utilisées dans l'automobile, à savoir le CAN (Controller Network Area).

Le tableau de bord, aussi appelé combiné d'instrumentation, peut ainsi être piloté soit à partir d'un pédalier et d'un comodo (commande au volant des phares et clignotants), soit par un appareil disposant du système d'exploitation Androïd via une connexion Bluetooth, soit par ordinateur relié par USB au système de contrôle du combiné. Les informations transitent par une carte électronique, qui se charge d'envoyer les trames CAN correspondantes au combiné. Le projet comprend alors les phases de développement Hardware et Software de la carte électronique ainsi que le développement des deux applications sur Androïd et ordinateur.

Au cours du dernier mois de mon stage, j'ai été amené à effectuer une mission pour le compte de Bosch en Allemagne. J'ai pu ainsi participer au développement d'une interface de signaux pour le projet Airbag d'Audi jusqu'à la livraison du projet au client.

La première partie est consacrée à la présentation de l'entreprise. Dans un second temps, nous nous intéresserons en détail à la réalisation de la maquette de démonstration ; la planification du développement et les deux phases principales de la réalisation, le Hardware et le Software, seront ainsi décrites.

1. Présentation de l'entreprise

1.1. L'entreprise

Le groupe Technology & Strategy (T&S) a été créé en janvier 2008, par l'association de trois codirigeants encore actuellement en activité. Ils sont chacun à la tête d'une entité, et sont tous issus du métier d'ingénierie de conseil.

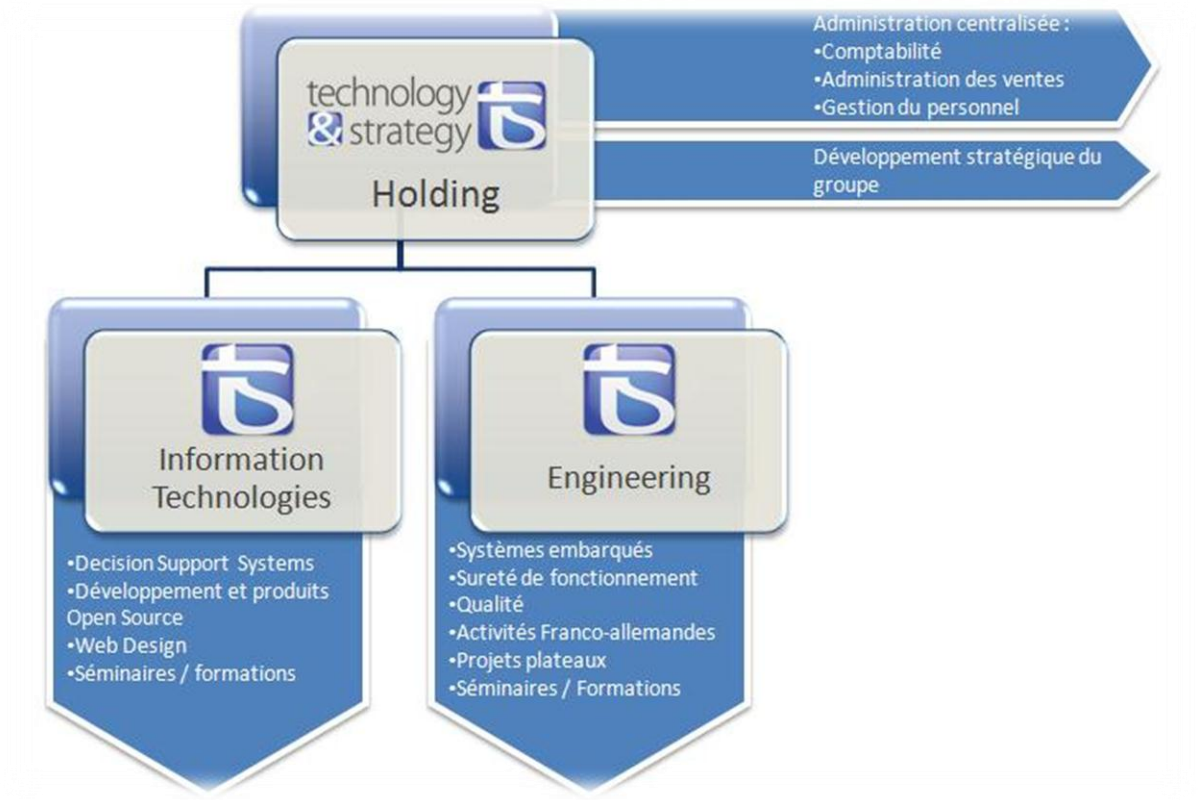

Figure 1 - Organigramme du fonctionnement de T&S

T&S est donc une société de conseil permettant à ses consultants d'apporter leur expertise, dans les métiers de l'Engineering et de l'Information Technologies, à ses clients. Les valeurs « transparence, créativité, partage et excellence », défendues par le groupe T&S, animent ses consultants en France et à l'étranger. Grâce à une écoute attentive des souhaits du consultant, la relation de confiance instaurée permet une évolution de celui-ci conformément à ses inspirations professionnelles.

La forme juridique retenue lors du démarrage se compose de deux entités opérationnelles (Information Technologies et Engineering) et d'une société de Holding. Un passé professionnel commun a déjà soudé les trois associés fondateurs autour d'un fort esprit d'entreprenariat. La somme des compétences acquises par les consultants au sein du groupe a pour vocation d'apporter les meilleures solutions à ses clients. La spécification Franco-allemande (cf. Figure 2) permet à T&S d'ouvrir à ses consultants les portes de carrières européennes.

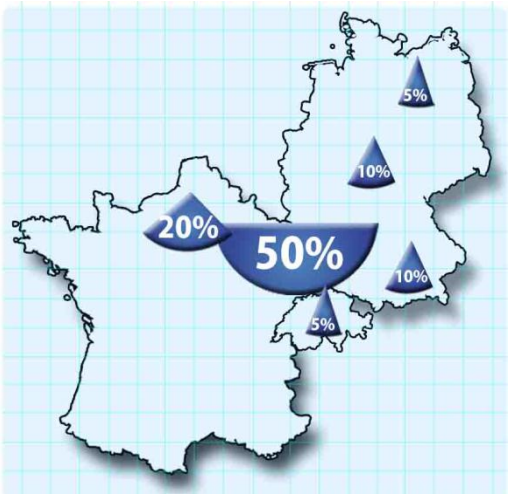

Figure 2 - implantation de T&S

1.1.1. T&S Holding

T&S Holding assure les fonctions de gestion transversale indispensables au fonctionnement de T&S Eng et T&S IT. Elle gère, entre autres, l'administration centralisée, les aspects financiers, les ressources humaines, l'infrastructure 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.

1.1.2. T&S Engineering

T&S Engineering a pour but de devenir un acteur majeur dans l'industrie des systèmes embarqués. C'est pourquoi la société compte sur son laboratoire de Recherche et Développement (R&D) et son positionnement Franco-allemand pour mener à bien cet objectif.

Le rayon d'action de T&S Engineering se situe principalement dans les domaines des transports (automobile, ferroviaire et aéronautique) mais aussi dans les domaines de l'Energie, en plein essor vu le contexte actuel.

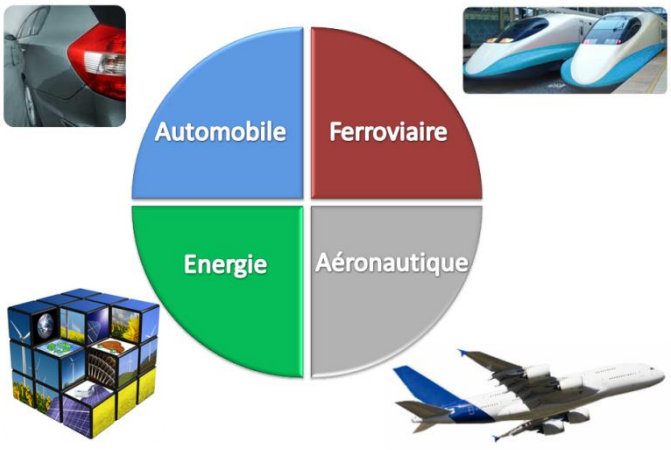

Figure 3 - Domaines d'intervention de T&S Engineering

Cette société est au jour d'aujourd'hui capable d'apporter à ses clients la meilleure des expertises en logiciels embarqués, Hardware embarqué, management de projet et qualité en intervenant sur les processus et les produits.

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, Porsche, Valeo mais également des grands groupes comme Hager et Socomec.

1.1.3. T&S Information Technologies

T&S Information Technologies (technologies de l'information) 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. T&S Information Technologies est divisée en trois parties :

- ICS : Solutions collaboratives, A/MOA, Spécifications, Conception, Design & Communication, Création et Mise en Œuvre de Portails Collaboratifs, Développement Web (ASP, Silverlight, Java, PHP), Convergence et Intégration, Hébergement et Infrastructure.
- ADS : Nouvelles technologies, Conception, Architecture, Design, IHM, Génie Logiciel (.NET, C#, VB.NET, C++, PB), Solution Web, Applications Industrielles, Expertise SGBD (Oracle, SQL Server, DB2), Gestion de Parcs de Licence.
- DSS : Business Intelligence, Outils d'aide à la Décision, Méthodologie Projet, Gestion de la Performance, Modélisation Décisionnelle, Intégration de Données, Tableau de Bord, Data Mining, Gestion de Système.

1.1.4. Quelques chiffres

De par ses valeurs et son positionnement, le groupe T&S bénéficie d'une très forte croissance comme le montre l'évolution du chiffre d'affaire et de l'effectif du groupe sur les trois premières années.

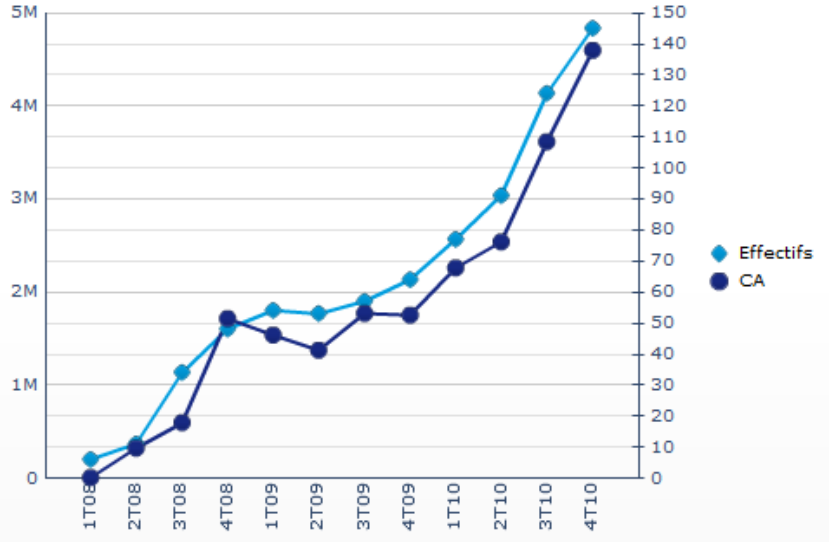

Figure 4 - évolution des chiffres T&S entre janvier 2008 et décembre 2010

1.2. RheinBrücke Consulting

Figure 5 - logo Rheinbrücke Consulting

A ces deux sociétés, se rajoute une société de conseil spécialisée dans le domaine franco-allemand, **RheinBrücke Consulting**. Elle se concentre dans le recrutement de profils bilingues par approche directe. La connaissance des différences juridiques et culturelles permet à RheinBrücke d'aborder aussi bien les entreprises allemandes actives en France que les sociétés française ayant une activité en Allemagne.

2. Réalisation de la maquette de démonstration

2.1. Planification du développement de la maquette

2.1.1. Etapes de développement

La réalisation de la maquette de démonstration comprend plusieurs étapes clés :

- Etude du cahier des charges et des spécifications nécessaires.
- Prise en main des outils de CAO.
- Conception de la carte électronique.
- Programmation du microcontrôleur.
- Réalisation des interfaces graphiques.
- Tests unitaires et d'intégration des différents modes de commandes.
- Validation des fonctionnalités.

La démarche suivie s'effectue de manière à respecter le plus possible le cycle en V (cf. figure …) représentant le cycle de développement traditionnel d'un produit :

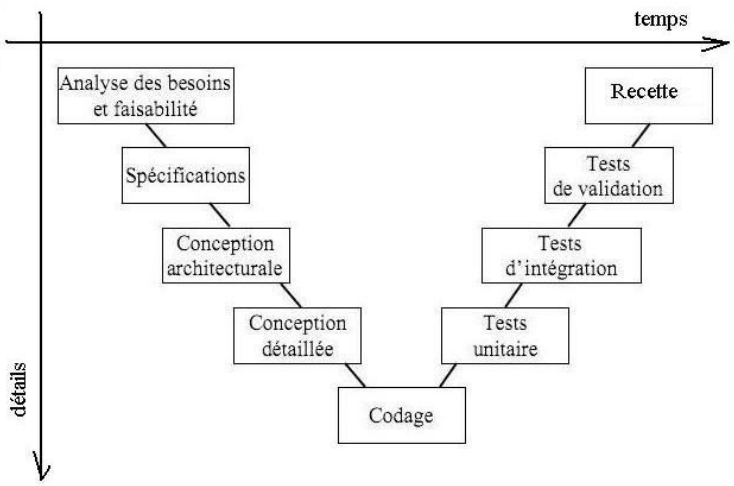

Figure 6 - Représentation du cycle en V

2.1.2. Cahier des charges initial

Les points suivants constituent le cahier des charges initialement prévu :

- Gérer 2 bus CAN, un Low-Speed (faible vitesse de communication) pour communiquer avec le combiné et un High-Speed pour communiquer, entre autres, avec le PC via un CANalyzer. Le CANalyzer permet de générer des trames CAN à partir du PC et de les transmettre par USB. La CANcaseXL contient un transceiver CAN qui se charge de l'envoi de la trame par câble null-modem.
- Pouvoir recevoir et envoyer des informations par USB, le but étant de pouvoir contrôler le combiné avec l'ordinateur.
- Ajout d'un module Bluetooth, qui laisse la possibilité de contrôler le combi avec un téléphone ou ordinateur via Bluetooth.
- Intégrer le mode de programmation ICSP.
- Plage d'alimentation de 7 à 18V.
- Réaliser des interfaçages divers pour le contrôle du tableau de bord.
- Mesurer la température ambiante.
- Mesurer la luminosité ambiante pour adapter la luminosité du combiné.

Le fonctionnement de l'ensemble peut être représenté comme suit :

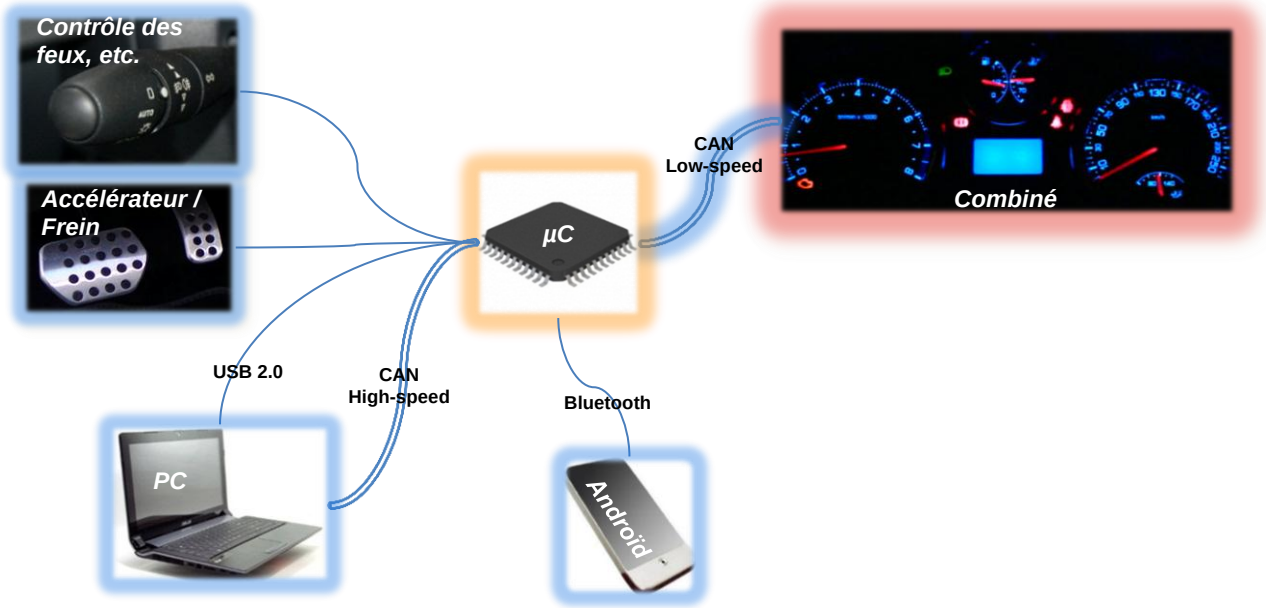

Figure 7 - Schéma de fonctionnement de la maquette

2.2. Développement hardware de la carte de commande du tableau de bord

2.2.1. Recherches bibliographiques

2.2.1.1. Protocole CAN 2.0

2.2.1.1.1. Intérêt et positionnement du protocole CAN

Etant donnée la croissance sans limite du nombre de commandes électroniques dans les véhicules, il était nécessaire de trouver une communication série permettant de s'affranchir d'un câblage trop lourd. Le CAN (Controller Area Network) permet d'échanger entre les nœuds d'un même réseau jusqu'à 2048 variables différentes, chacune comportant 8 octets de données.

Le concept de communication du bus CAN est celui de la diffusion d'information (broadcast) : les messages ne sont pas envoyés à un destinataire en particulier mais sont envoyés à tous les nœuds du réseau. Chacune des stations connectées au réseau écoute les trames transmises par les stations émettrices et décide de prendre ou non en compte le message circulant sur le bus.

Le modèle OSI établit une forme de représentation hiérarchisée (en couches) d'une architecture réseau, comme décrit dans l'article « Modèle OSI » [1].

Le protocole CAN ne concerne que les couches basses du modèle OSI, à savoir la couche de liaison de données et la couche physique. Son utilisation est donc facilement généralisable.

- La couche de liaison de données gère les communications entre 2 machines directement reliées entre elles par un support physique. Le type de donnée concerné par cette couche est la trame de bits.
- La couche physique, couche la plus basse, concerne les niveaux de signaux et la représentation des bits.

2.2.1.1.2. Arbitrage

Etant de type multi-maître, le protocole CAN autorise différentes stations à accéder simultanément au bus. C'est un procédé rapide et fiable d'arbitrage qui détermine la station qui émet en premier. C'est l'identificateur qui va indiquer la priorité du message, qui détermine l'assignation du bus lorsque plusieurs stations émettrices sont en concurrences.

Dans le cas de CAN 2.0A, l'identificateur est codé sur 11 bits (format standard), ce qui permet de définir jusqu'à 2048 messages plus ou moins prioritaires sur le réseau. Chaque message peut contenir jusqu'à 8 octets de données, ce qui correspond par exemple à l'état de 64 capteurs. L'adressage par le contenu assure une grande flexibilité de configuration. Il est ainsi possible d'ajouter des stations réceptrices à un réseau CAN sans modifier la configuration des autres stations. La spécification CAN 2.0B permet de définir des identificateurs sur 29 bits (format étendu), ce qui laisse la possibilité d'accroître considérablement le nombre de messages transmissibles au sein d'un même réseau, et par conséquent d'accroître le nombre de capteurs dans un véhicule.

L'arbitrage bit à bit (non destructif) est effectué tout au long du contenu de l'identificateur. Ce mécanisme garantit qu'il n'y aura ni perte de temps, ni perte d'informations. Dans le cas de deux identificateurs identiques, la trame de données gagne le bus. Tous les nœuds vérifient l'état du bus après émission d'un bit :

- Si un bit récessif est émis et qu'un bit récessif est lu, le nœud continue d'émettre.
- Si un bit récessif est émis et qu'un bit dominant est lu, il y a conflit entre plusieurs nœuds et perte d'arbitrage. Le nœud cesse d'émettre et repasse en mode réception.

2.2.1.1.3. Routage des informations

Dès que le bus est libre, n'importe quel nœud peut envoyer un message sur le bus. Un message est une trame de bits ayant un format bien défini. On en distingue deux types : les trames de requête et les trames de données (cf. partie 2.3.2.1. pour le détail sur la constitution de ces trames).

L'identificateur d'un message n'indique pas la destination du message mais la signification des données du message. Ainsi tous les nœuds du bus reçoivent le message, et chacun est capable de savoir grâce au système de filtrage de message si ce dernier lui est destiné ou non. Chaque nœud peut également détecter des erreurs sur un message qui ne lui est pas destiné et en informer les autres nœuds.

2.2.1.1.4. Types de trames

On distingue deux types de trames CAN :
- Data Frame (trame de données) : transporte les données.
- Remote Frame (trame de requête) : envoyée par un nœud désirant recevoir une certaine trame de donnée (de même identificateur).

La forme d'une trame CAN 2.0A est la suivante :

Figure 8 - Structure d'une trame CAN 2.0A [2]

- Le bit SOF est un bit dominant signalant le début de la trame.
- Les 11 bits suivants concernent l'identificateur.
- Le bit RTR permet de différencier la trame de requête (bit récessif) et de données (bit dominant).
- Les deux bits suivants sont réservés.
- Le champ DLC représente le nombre d'octets contenus dans le champ de données.
- Le champ de données contient les données transmises, d'une longueur de 0 à 8 octets. Dans le cas d'une trame de requête, aucune donnée n'est transmise.
- Le champ CRC permet de détecter une erreur de transmission d'information dans la trame : si la valeur transmise et la valeur calculée diffèrent, alors il y a erreur de CRC.
- Le champ Acknowledge est composé de deux bits :
  - Ack Slot : bits récessif et dominant sont superposés. Si un nœud détecte que ce bit est récessif, alors il doit signaler l'erreur.
  - Ack Delimiter : bit toujours récessif.
- Fin de trame composée de 7 bits récessifs.

2.2.1.1.5. Gestion des erreurs

Dans le but d'obtenir la plus grande sécurité lors de transferts sur le bus, des dispositifs de signalisation, de détection d'erreurs, et d'autotests ont été implémentés sur chaque noeud d'un réseau CAN.

Les types d'erreurs que l'on peut rencontrer sont :

- « Bit Error » : le transmetteur émet un bit dominant mais détecte un bit récessif sur le bus.
- « Acknowledge Error » : le slot Acknowledge est un bit récessif (signifie qu'aucun nœud n'a reçu le message correctement).
- « Form Error » : un bit des segments « CRC Delimiter », « ACK Delimiter » ou « End Of Frame » est dominant.
- « Bit Stuffing Error » : après 5 bits dominants ou récessifs successifs, un bit respectivement récessif ou dominant doit être généré. Dans le cas contraire, cette erreur est générée.
- « CRC Error » : la valeur du CRC reçu ne correspond pas à celle calculée.

Les éléments dont on dispose pour détecter les différentes erreurs sont les suivants :

- Le monitoring bus (vérification du bit émis sur le bus),
- Calcul du CRC (Cyclic Redundancy Check),
- La procédure de contrôle de l'architecture du message,
- La méthode de Bit-Stuffing.

Toutes les erreurs globales et locales au niveau des émetteurs sont alors détectables. Il en résulte une probabilité totale résiduelle de messages entachés d'erreurs inférieure à $4{,}6.10^{-11}$.

Chaque nœud signale s'il détecte une erreur en émettant immédiatement une trame d'erreur. A ce moment, le message n'est pris en compte par aucun des nœuds du réseau, et le nœud émetteur recommence la transmission de ce même message.

Si l'erreur persiste, le nœud émetteur est considéré comme défectueux et doit se déconnecter (électriquement) du réseau. Des compteurs d'erreurs sont mis en place pour déterminer le degré de perturbation du nœud. Chaque message erroné va incrémenter le compteur, tandis que chaque message correct va décrémenter celui-ci.

2.2.1.2. Protocole USB 2.0

La communication entre l'hôte (l'ordinateur) et les périphériques se fait selon un protocole basé sur l'interrogation successive de chaque périphérique par l'hôte. L'hôte émet un jeton désignant un périphérique lorsqu'il désire communiquer avec un périphérique ; le jeton est un paquet de données, codé sur sept bits et contenant l'adresse du périphérique. Le périphérique renvoie un paquet de données de 8 à 255 octets s'il reconnaît son adresse dans le jeton. L'adresse étant codée sur 7 bits, 128 périphériques peuvent être connectés simultanément à un même port.

On distingue trois modes de vitesse d'échange de données permettant de sélectionner le débit souhaité :

- Low Speed : 1,5 Mbps pour les périphériques qui ont besoin de transférer peu de données.
- Full Speed : 12 Mbps pour les données imprimantes, scanner ou encore disques durs.
- High Speed : 480 Mbps pour la plupart des périphériques de stockage de masse.

On dénombre 4 types de transferts différents, selon l'application effectuée :

- Transfert de commande : employé pour la configuration des périphériques principalement, donc pour des données de petites tailles.
- Transfert d'interruption : l'hôte interroge le périphérique avant que celui-ci ne puisse effectuer un transfert. Utilisé par exemple pour le clavier et la souris.
- Transfert isochrone : utilisé pour des échanges de données volumineux, en revanche, le succès de transmission des données n'est pas garanti, très utilisé pour la vidéo et l'audio.

- Transfert de masse (Bulk) : transfert de données volumineuses avec un succès de transmission des données assuré, mais sans garantie de bande passante.

Dans la partie 2.3.2.6 sont décrits les choix effectués pour notre application.

2.2.1.3. Etude et choix du microcontrôleur

Etant donné le peu de place disponible sur la carte, le microcontrôleur doit lui-même fournir un maximum de fonctionnalités.

Le choix s'est dans un premier temps porté sur le dsPIC33FJ de Microchip car il possède deux modules ECAN. La présence d'un transceiver interne USB dans le dsPIC33EP256MU806 nous a cependant amenés à le préférer plutôt que le dsPIC33FJ.

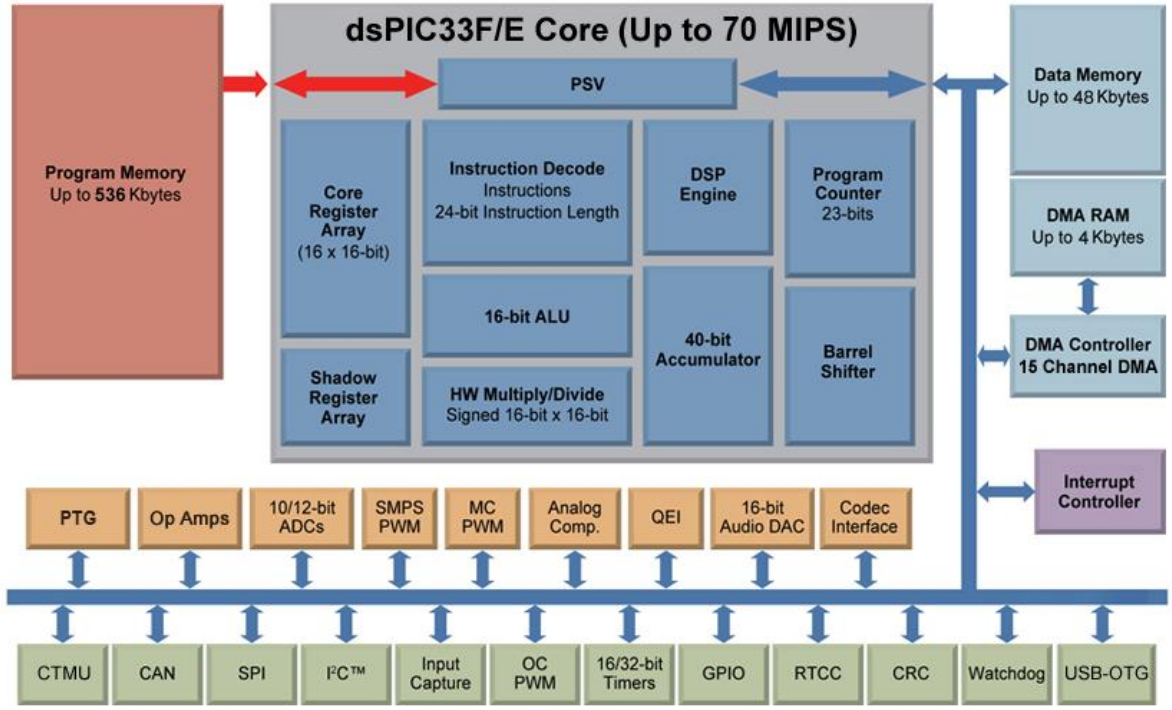

Figure 9 - Architecture de la famille de microcontrôleurs dsPIC33

Parmi les principales caractéristiques énoncées dans la documentation technique du dsPIC33E [3] qui vont être utiles dans notre projet, se trouvent :

- 2 modules ECAN indépendants qui gèrent les communications CAN pour des vitesses allant de 125 kb/s à 1 Mb/s,
- Une mémoire Flash de 256Kbytes (largement suffisante pour notre application),
- Une mémoire RAM de 28KBytes permettant de réaliser par exemple des calculs flottants,
- 1 module USB, qui offre différentes configurations. Le dsPIC peut être utilisé comme device (périphérique USB), host ou bien en mode On-The-Go si l'on souhaite pouvoir changer de mode (host ou device) en cours de fonctionnement.
- 24 entrées analogiques pour la conversion ADC de résolution 10 bits à 1100 ksps.
- Il possède au total 64 broches, 53 pour les entrées/sorties dont 48 d'entre elles sont remappables. On a alors le choix d'utiliser tel ou tel périphérique sur telle ou telle Pin remappable, ce qui laisse une grande liberté dans le routage de la carte. Le grand nombre mis à disposition permet d'utiliser tous les périphériques dont nous aurons besoin pour notre application (2x UART, 2x ECAN, 1x USB, 5x ADC).

2.2.2. Réalisation Hardware

2.2.2.1. Outils de CAO

Le logiciel de CAO utilisé pour effectuer le schéma et le routage de la carte est CAD Eagle.

Ce logiciel ne permet pas de faire de simulation du fonctionnement du circuit, il n'est pas nécessaire de réaliser le comportement d'un composant et il est donc possible de créer facilement les composants absents des librairies disponibles. Tous les composants d'une même carte doivent se trouver dans une même librairie.

2.2.2.2. Contraintes

Le PCB de la carte principale ne doit pas excéder le format européen, soit une dimension de 100x80mm. Ceci constitue la principale contrainte du routage. La difficulté résultant de cette contrainte est la réunification de manière logique des composants sur la carte (par blocs de préférence), tout en minimisant la longueur des pistes, notamment des pistes d'alimentation.

L'emplacement de certains composants doit être optimisé :

- les connecteurs doivent se trouver de préférence en bord de carte,
- le capteur de luminosité doit être inséré dans une zone suffisamment aérée pour ne pas fournir une information non représentative de la réalité,
- les boutons doivent être accessibles,
- etc.

2.2.2.3. Module microcontrôleur

Comme c'est souvent le cas dans le développement de cartes électroniques, une première version Hardware est mise au point avec une carte mère et des cartes modulaires. Ainsi, en cas de panne du microcontrôleur ou de modifications nécessaires sur les cartes modulaires, il n'est pas nécessaire de refaire une carte complète. Dans notre cas, cela s'avère encore plus utile du fait de la difficulté de souder les microcontrôleurs étant donnée leur petite taille. De plus, ces cartes modulaires vont être amenées à être réutilisées dans de futurs projets, il s'agit donc d'un gain de temps dans les prochaines phases de recherche et de développement.

L'objectif étant de pouvoir conserver toutes les fonctionnalités du dsPIC33E, il est évident que toutes les pins d'entrées et de sorties doivent être accessibles et directement retranscrites sur les Pins externes de la carte. Les composants supplémentaires que l'on va intégrer sur le module vont réaliser l'adaptation en tension 5V-3.3V, ou encore la génération de l'horloge par la présence d'un quartz, ces deux éléments étant de toute façon liés directement aux caractéristiques du dsPIC utilisé. Des condensateurs de découplages sont ajoutés au niveau des pins d'alimentation afin de garantir une bonne stabilité en tension.

Les contraintes à prendre en compte sont principalement des contraintes de place disponible dans l'espace 3D. En effet, il peut être intéressant de laisser la possibilité de placer des composants de la carte mère sous le module, une fois celui-ci inséré. L'utilisation de composants CMS est donc incontournable.

La carte mère ne doit pas dépasser le format européen, à savoir 100x80mm. Le module doit donc être suffisamment petit pour faire insérer sur la carte mère la totalité des composants, notamment ceux qui peuvent nécessiter une grande place (relais, module bluetooth, connecteurs, etc.). Finalement, le module a une dimension de 62x41mm.

Ci-contre le routage de la carte modulaire :

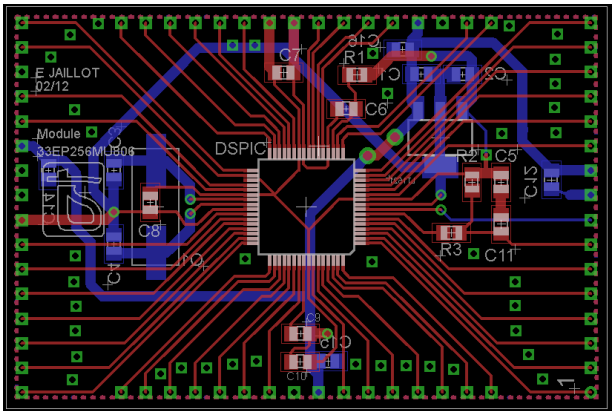

Figure 10 - Routage de la carte modulaire contenant le dsPIC

Le dsPIC est alimenté en 3,3V. La carte principale fournit au module une tension régulée de 5V. Le régulateur utilisé pour transformer la tension de 5V à 3,3V est le LT1521-3.3. Ce régulateur peut fournir au maximum 300mA pour une tension de sortie de 3,3V.

Il faut s'assurer que ce courant sera suffisant pour alimenter tous les composants fonctionnant sous cette tension. Les éléments alimentés en 3.3V sont les suivants :

- Bluetooth : consommation de 45mA maximum en transmission,
- dsPIC : consommation de 120mA maximum,
- MCP9700 (capteur de température) : consommation légèrement supérieure à 100uA,
- BH1600FVC (capteur de luminosité) : consommation de 7,5mA maximum,
- TXS0102 (adaptateur de niveau bidirectionnel) : conso de l'ordre de la dizaine de uA,
- 3 trimmers de 100kΩ : consommation totale de 100uA environ,

La consommation totale de ces éléments est donc au maximum de 173mA. Le régulateur délivre donc un courant suffisant pour alimenter tous ces composants.

La fréquence du quartz utilisé est de 8MHz. Cette valeur a été choisie afin de pouvoir générer, via les registres de configuration, la fréquence de fonctionnement du module USB à 48MHz.

2.2.2.4. Carte principale

Ci-après le schéma ainsi que le routage de la carte principale :

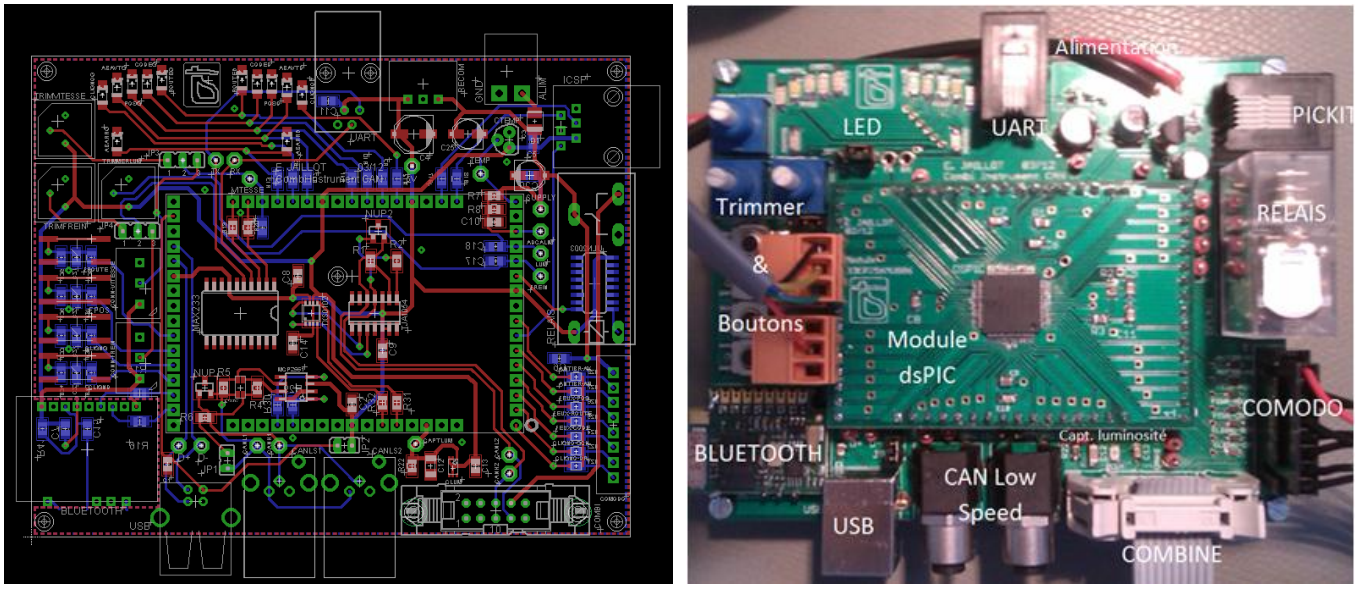

Figure 11 - Routage et photo du rendu final de la carte principale

La schématique de chacun des modules identifiés sur la photo ci-dessus est décrite dans les sous-parties suivantes.

2.2.2.4.1. Gestion de l'alimentation (régulateurs RECOM)

Comme décrit dans le cahier des charges, la plage de tension doit s'étendre de 7 à 18V. Le combiné est réalisé de telle manière qu'il fonctionne sous n'importe quelle tension comprise dans cette plage.

Il est donc simplement nécessaire de disposer d'une tension de 5V pour l'alimentation du circuit. Les caractéristiques du convertisseur de tension doivent être les suivantes :

- plage de tension de fonctionnement 7-18V incluse dans la plage de tension d'entrée du régulateur,
- tension de sortie de 5V.

Le choix s'est donc porté sur le régulateur RECOM R-785.0-1.0, dont la plage de tension en entrée s'étend de 6,5 à 18V et la tension de sortie de 5V. Le RECOM peut fournir jusqu'à 1A, ce qui est largement suffisant pour alimenter le circuit.

Le schéma concernant l'adaptation d'alimentation est le suivant :

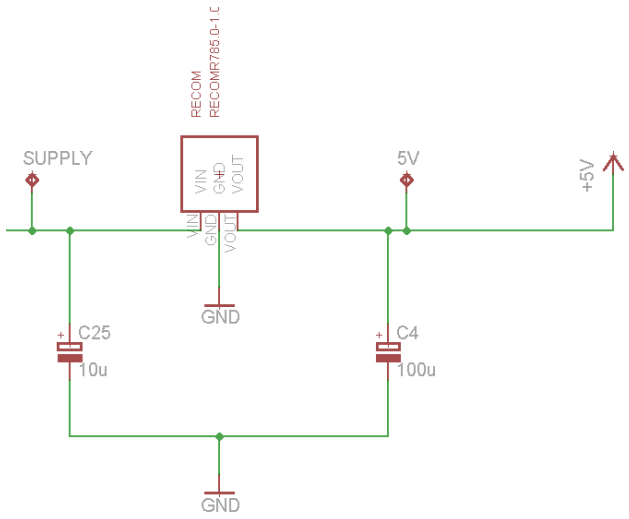

Figure 12 – Schéma de l'adaptation de la tension d'alimentation

2.2.2.4.2. Boutons et LEDs

L'interface homme-machine de la carte principale est composée de boutons et de LEDs. Le but premier de ces éléments est de faciliter le débogage. Les boutons sont également utilisés pour démarrer/arrêter le combiné et pour changer de mode de contrôle du combiné.

4 boutons offrent ces fonctionnalités. Ci-après, on retrouve la partie du schéma dédiée aux boutons :

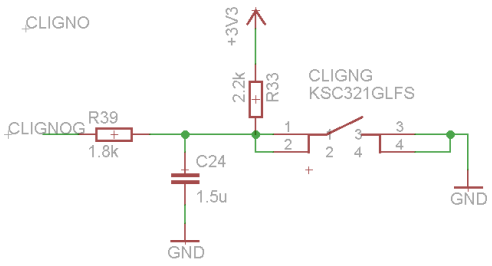

Figure 13 - Schématique dédiée aux boutons poussoirs

« CLIGNOG » représente l'entrée du PIC. Les 2 résistances et le condensateur forment le filtre anti-rebond du bouton.

Ce filtre est dimensionné afin d'atteindre la tension de 3,3V en entrée du PIC en 10ms lors du relâchement du bouton.

Dans la figure n°11, on peut constater que les LEDs sont disposées de manière à reproduire les phares d'un véhicule. Les résistances placées en série avec celles-ci dépendent de chaque type de LED utilisé et de l'intensité souhaitée. L'ordre de grandeur est de 1kΩ par sortie pour l'alimentation d'une LED.

2.2.2.4.3. Entrées analogiques, Conversion ADC

Nous disposons de 24 entrées analogiques et nous n'avons besoin que de 4 d'entre-elles.

Les contraintes inhérentes aux entrées analogiques sont le contrôle du niveau de tension et de courant. En effet, il faut que la tension en entrée soit inférieure ou égale à 3,3V et que le courant entrant n'excède pas 8mA pour ne pas détériorer l'entrée du dsPIC.

Les cinq entrées analogiques sont les suivantes :

- capteur de température MCP9700 :

Alimenté en 3,3V, la tension de sortie Vout reliée à l'entrée analogique du PIC est proportionnelle à la température extérieure.

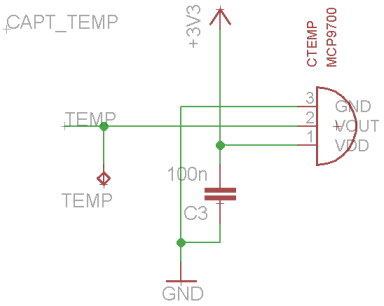

Figure 14 - Schématique du capteur de température

L'abaque suivant représente la tension de sortie en fonction de la température ambiante :

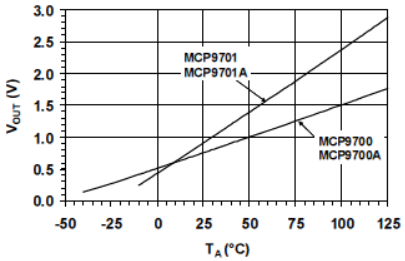

Figure 15 - Abaque tension de sortie vs. température ambiante

- capteur de luminosité ambiante BH1600FVC :

Le capteur de luminosité fournit un courant proportionnel à la luminosité. Le courant maximal de sortie est 7,5mA. Ainsi, les 8mA maximum tolérés par le dsPIC sont respectés.

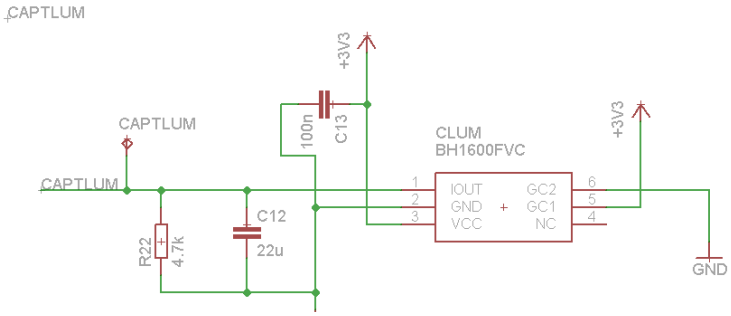

Figure 16 - Schématique du capteur de luminosité

L'abaque ci-contre représente le courant de sortie en fonction de la luminance :

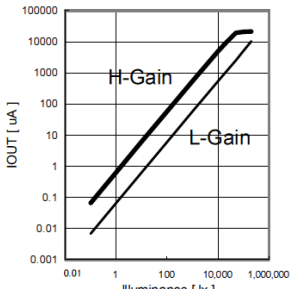

Figure 17 - Abaque courant de sortie vs. luminosité ambiante

- trimmers pour les pédales d'accélération et de freinage :

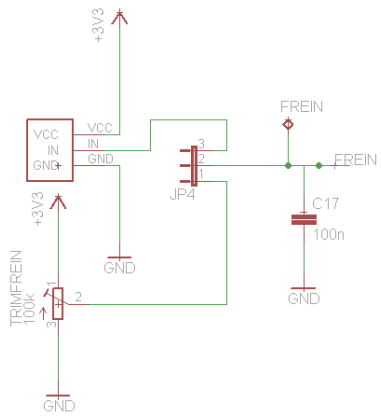

Figure 18 - Schématique de la partie dédiée aux trimmers et pédales

Un jumper permet de choisir le trimmer du pédalier ou celui disponible sur la carte électronique.

La tension est alors comprise entre 0 et 3,3V, dépendant de la position des trimmers, et est liée à l'une des entrées analogiques du dsPIC.

2.2.2.4.4. Bluetooth

Comme vu dans la partie de recherche bibliographique, le protocole SPP de la communication Bluetooth simule un port série. Il s'adapte particulièrement bien à notre application étant donné le peu de données à transmettre. De plus, ces modules Bluetooth ont l'avantage d'être petits et peu chers.

Le module Bluetooth choisi est le ARF7069B, il communique par UART avec le dsPIC.

Ci-après le schéma électrique du module Bluetooth :

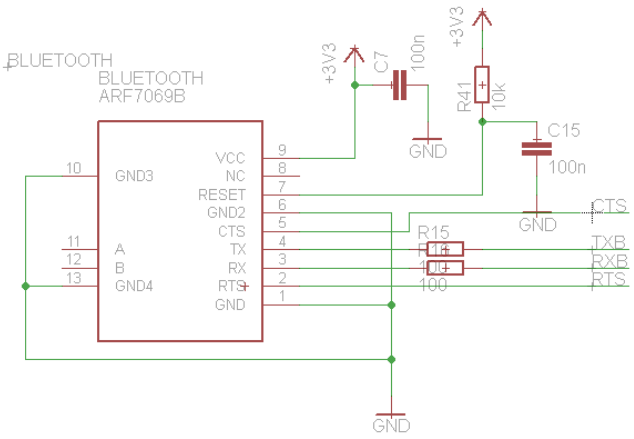

Figure 19 - Schématique de la partie dédiée au Bluetooth

Les liens TXR et RXR sont reliés au dsPIC. TXR et RXR étant respectivement les broches de réception et d'envoi des données, si on se place du point de vue du dsPIC.

Les entrées et sorties CTS et RTS étant gérées du point de vue du module Bluetooth, il convient de les relier aux entrées/sorties du dsPIC correspondantes.

2.2.2.4.5. Contrôle de l'alimentation du combiné

Afin d'être démarré, le combiné doit recevoir des trames CAN très peu de temps après sa mise sous tension. Il est donc indispensable de contrôler cette alimentation. Cela est réalisé par l'intermédiaire d'un relais et d'une commande de relais.

Le relais choisi est le G2R-1 5DC de OMRON. Ce relais peut laisser passer un courant de 1A maximum et la bobine doit être alimentée sous une tension de 5V continu et à un courant de 106mA au minimum pour rendre le relais passant.

Une fois la bobine du relais alimentée, ce dernier fournit la tension d'alimentation de la carte au combiné, soit une tension comprise entre 7 et 18V.

Le composant qui réalise la commande du relais est le ULN2003. Il est composé de 7 drivers à base de transistors Darlington. Le courant collecteur en sortie peut atteindre jusqu'à 500mA pour un courant de base maximum d'environ 1,5mA. Ce composant offre donc largement le courant nécessaire à alimenter la bobine du relais.

Ci-dessous le schéma correspondant. Le lien « COMB » est relié à une pin de sortie du dsPIC dédiée à la commande du combiné. Le connecteur nommé « COMBI » est relié aux entrées et sorties du combiné. La pin 8 de ce connecteur représente la pin d'alimentation du combiné.

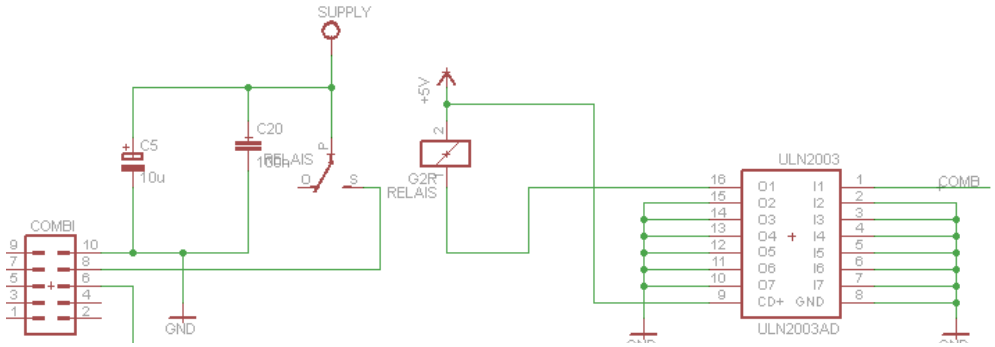

Figure 20 - Schématique de la commande de l'alimentation du combiné

2.2.2.4.6. Mise en œuvre du comodo

Le comodo est alimenté en 12V. Afin que les niveaux de tensions correspondent à ceux acceptés par le dsPIC, les tensions des différentes sorties du comodo sont converties en 3,3V par l'intermédiaire de diodes Zéner 3,3V afin d'être adaptées en entrée du dsPIC. Le schéma suivant illustre cette conversion :

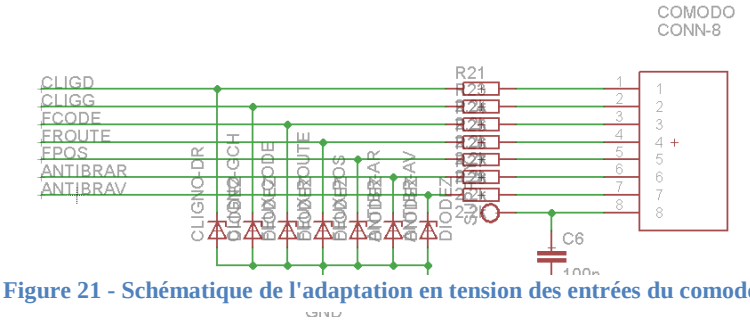

Figure 21 - Schématique de l'adaptation en tension des entrées du comodo

Il s'agit de diodes DZ2J033 de tension de Zéner 3,3V à 5mA. Comme la plage de tension s'étend de 7V à 18V, si l'on choisit de placer une résistance de 2,2k, les courants inverses dans la diode iront de 3,2mA à 8,2mA. D'après les abaques, la tension de Zener sera alors comprise entre 3 et 3,4V environ.

2.2.2.4.7. Bus CAN Low-Speed

Le bus CAN Low-Speed est le bus sur lequel transitent les trames destinées au contrôle du combiné.

La figure ci-dessous représente le schéma de la partie CAN Low-Speed.

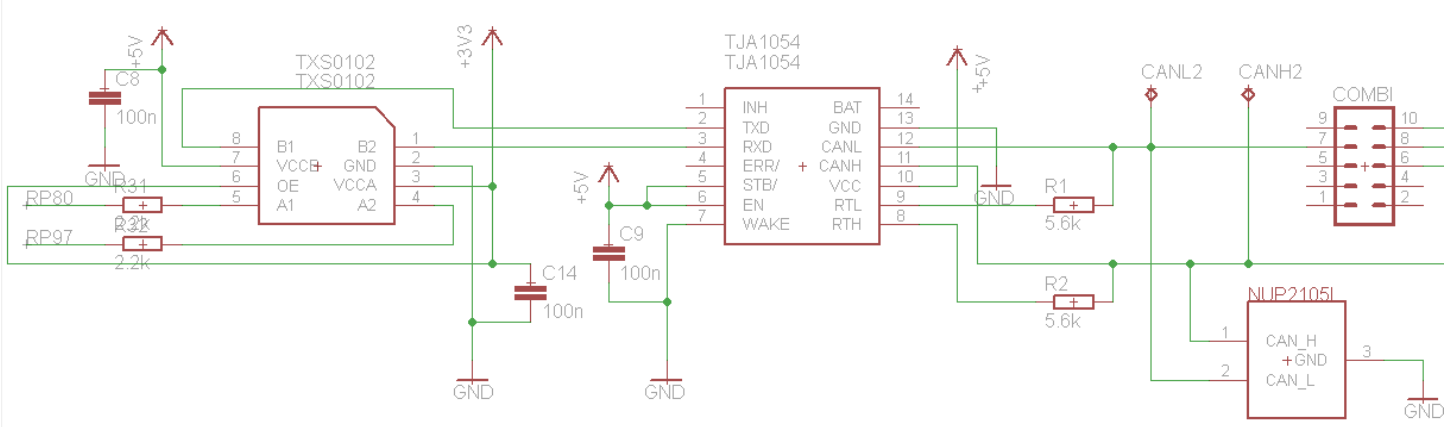

Figure 22 - Schématique de la partie dédiée au bus CAN Low-Speed

Elle est composée des éléments suivants :

- Pin de sortie remappable RP80 du dsPIC, pour la transmission des données CAN.
- Pin d'entrée remappable RP97 du dsPIC, pour la réception des données CAN.
- Un adaptateur de tension TXS0102.
- Un transceiver CAN Low-Speed TJA1054.
- Circuit de diodes de protection NUP2105L pour protéger contre les surtensions en régime transitoire.

Le TJA1054 est un des seuls transceivers CAN Low-Speed sur le marché. Il est très utilisé dans le domaine automobile pour les applications Low-Speed allant jusqu'à 125kbps. Il sert d'interface entre le contrôleur de protocole CAN (module ECAN du dsPIC) qui génère les signaux CAN numériques et le bus CAN physique. Le TJA1054 est alimenté sous 5V.

Les données sont envoyées/reçues par le dsPIC via son module ECAN. La tension en sortie du dsPIC est de 3,3V et la tension d'entrée maximale vaut 5V car il s'agit d'une pin tolérant le 5V. En revanche, la tension d'entrée à l'état haut V_{IH} du TJA1054 vaut :

$$V_{IH} = 0{,}7 * V_{CC} = 0{,}7 * 5 = 3{,}5V$$

Sans translation de la tension de 3,3V à 5V, cette tension est trop élevée pour que le TJA1054 détecte les états hauts transmis par le dsPIC. Ajouter une résistance de pull-up suivie d'un inverseur serait une solution au problème, mais la place occupée par l'ensemble serait beaucoup plus importante que celle occupée par le TXS0102. De plus, il faut que l'adaptation de tension puisse s'effectuer à la vitesse de 125kbps. Le TXS0102 assure une transmission pour des vitesses allant jusqu'à 24Mbps. Etant bidirectionnel, il permet également d'adapter la tension du signal RXD entrant de 5V à 3,3V pour les niveaux hauts.

2.2.2.4.8. Bus CAN High-Speed

Nous utilisons un second bus CAN High Speed afin de communiquer avec d'autres systèmes présents sur le bus.

Ce bus a permis par exemple de communiquer les données de vitesse et d'état des voyants à la carte Linux embarquée d'un autre stagiaire, qui a ainsi pu afficher un combiné numérique sur son écran. Je me suis également servi du CAN High-Speed pour utiliser le CANalyzer et observer les trames que j'envoyais.

Le schéma suivant représente la partie consacrée au CAN High-Speed :

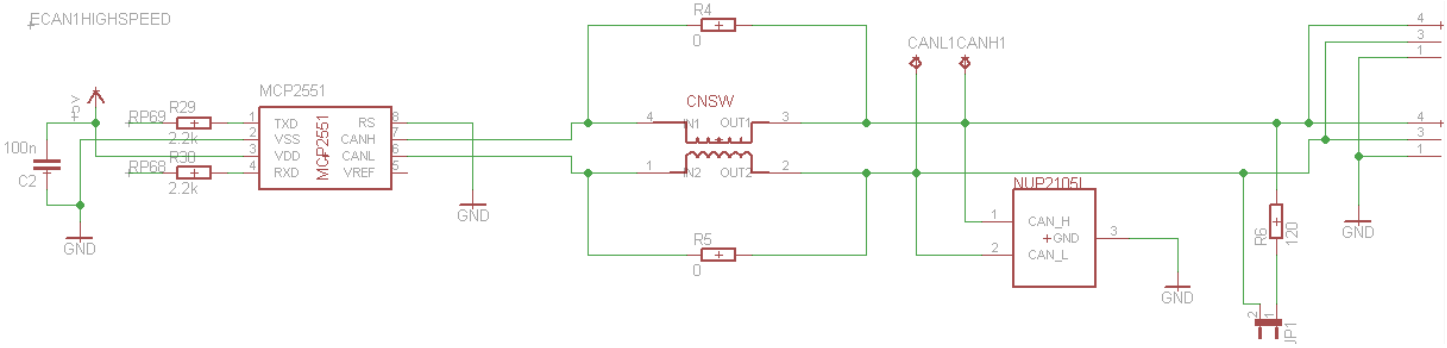

Figure 23 - Schématique de la partie dédiée au bus CAN High-Speed

Cette partie du schéma est composée des éléments suivants :

- Transceiver CAN High-Speed MCP2551.
- Circuit Diodes de protections : NUP2105L, pour la protection contre les surtensions en régime transitoire.
- Common Mode Choke : bobines couplées insérées en sortie du transceiver.
- Résistance de terminaison de 120Ω.

- Pin de sortie remappable RP69 du dsPIC, pour la transmission des données CAN.
- Pin d'entrée remappable RP68 du dsPIC, pour la réception des données CAN.

Le transceiver CAN High Speed MCP2551 de Microchip permet de générer les signaux CAN adaptés au bus High Speed à partir des signaux numériques délivrés par le module ECAN du dsPIC. Ce transceiver peut réaliser cette fonction pour des vitesses de communication allant jusqu'à 1Mbps. Ce transceiver est alimenté en 3,3V et le niveau de tension haut pour l'entrée TXD et la sortie RXD est de 3,3V également. Un étage d'adaptation de tension n'est donc pas nécessaire ici.

Il est préférable d'ajouter quelques éléments de protection et de stabilisation du signal. Le même circuit de diodes de protections NUP2105L que pour le bus CAN Low-Speed est ajouté, il permet de protéger le transceiver contre les surtensions en régime transitoire. Les bobines CMC (Common Choke Mode) permettent quant à elles d'augmenter le taux de réjection de mode commun du transceiver et ainsi de filtrer les distorsions du signal.

La valeur de la résistance de terminaison est normalisée, elle doit faire 120Ω. Le jumper JP1 laisse la possibilité d'ajouter ou non cette résistance, selon que notre système est inséré à l'une des extrémités du bus CAN ou non.

2.2.2.4.9. Communication série

La communication série va nous permettre de déboguer le software plus aisément en communiquant avec l'ordinateur.

Le codage NRZ est utilisé pour la communication RS232 d'un ordinateur. En effet, du point de vue du port RS232 de l'ordinateur, une tension de +12V représente un « 1 » logique et -12V représente un « 0 » logique. Pour cette raison, il faut que la tension soit adaptée en sortie du PIC. Le MAX233 effectue cette opération d'interfaçage. Le schéma de fonctionnement est le suivant :

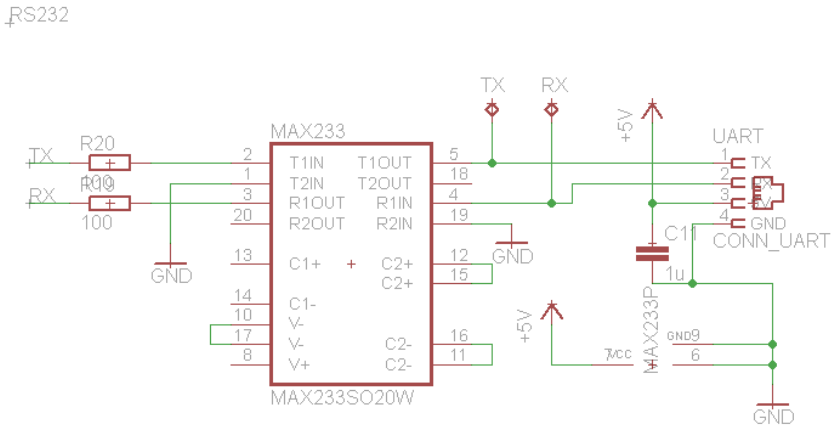

Figure 24 - Schématique de la partie dédiée à la communication UART

Les liens RX et TX représentent les entrées et sorties du dsPIC. Il s'agit de pins remappables auxquelles est associé un des deux modules UART disponible. Le MAX233 est alimenté en 5V. Les niveaux de tensions sont les suivants : en transmission, $V_{IL} = V_{IH} = 1{,}4V$; en réception $V_{OL} = 0{,}2V$ et $V_{OH} = 3{,}5V$. En transmission, les niveaux de tensions bas et haut sont respectivement supérieurs et inférieurs aux niveaux de tensions fournis par le dsPIC. En réception, la pin du dsPIC tolérant le 5V, ces niveaux de tensions sont adaptés à ceux du dsPIC. Il n'y a donc besoin d'aucune adaptation de tension dans cette partie.

Le connecteur utilisé est un RJ11. Le port série de l'ordinateur attend un connecteur D-SUB 9. La fiche RJ11 a été préférée étant donné le peu de place qu'elle occupe. Il a alors fallu créer un câble ayant l'un des deux connecteurs à chacune des extrémités.

2.2.2.4.10. USB

Le contrôleur USB est intégré au dsPIC. Les entrées et sorties du dsPIC dédiées à la communication USB sont donc directement reliées au bus.

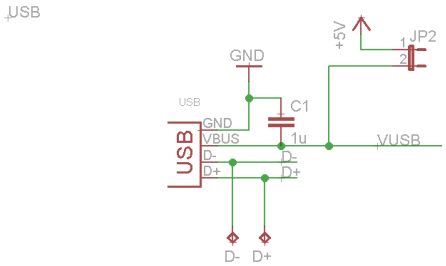

Figure 25 - Schématique de la partie USB

Selon les spécifications USB 2.0, un condensateur de 1µF environ est nécessaire entre l'alimentation du bus et la masse.

Le jumper JP2 permet de relier ou non l'alimentation 5V à la piste d'alimentation du bus. Le host alimente le bus. Dans notre application, le dsPIC est le device, il n'est donc pas nécessaire d'alimenter le bus. Ce jumper laisse simplement la possibilité de passer dans le mode hôte.

Différents types de connecteurs peuvent être utilisés : connecteur A, B, mini A ou mini B :

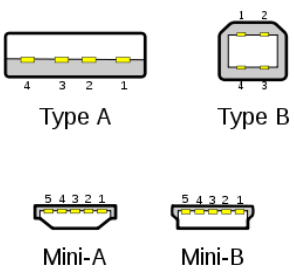

Figure 26 - Types de connecteurs USB

La norme impose, pour éviter tout type d'erreur de branchement, de relier l'hôte à un connecteur de type A et le périphérique à un connecteur de type B. C'est pourquoi dans notre cas, le type B est utilisé.

2.3. Réalisation Software

2.3.1. Structure générale du software

Le fonctionnement du software est régi par les interruptions des différents modules le composant. On distingue 5 interruptions différentes, qui permettent l'acquisition, l'enregistrement, le traitement, l'envoi et la réception de données.

Sur le schéma en figure n°26 de la page suivante, on peut observer la structure générale du software.

Signification des flèches :

Echange de données avec un dispositif externe (Androïd, PC ou tableau de bord)

- Envoi de requêtes de démarrage ou d'arrêt de communication
- Récupération des données (Getter)

Appel de fonctions

Transmission des données (requêtes de démarrage ou arrêt de la communication, état des voyants, consignes d'accélération, freinage)

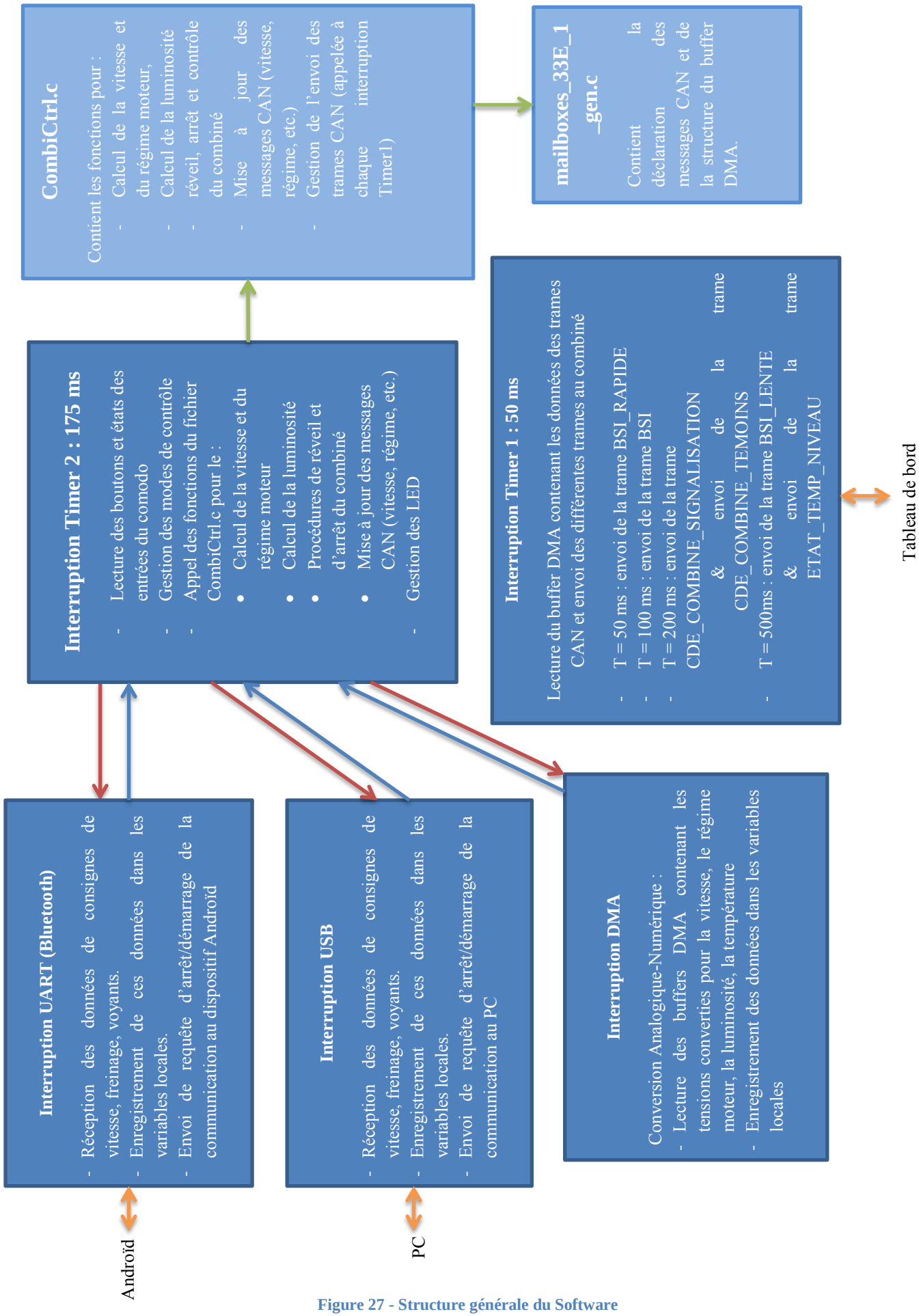

Figure 27 - Structure générale du Software

2.3.2. Interfaçage avec le combiné

2.3.2.1. Communication CAN

Pour démarrer le combiné, il faut passer par une phase de réveil, qui consiste à lui envoyer des trames à intervalles de temps réguliers (toutes les 50ms, 100ms, 200ms et 500ms selon le type de trame) indiquant le passage du mode veille à une phase de vie normale (voir la désignation des trames page 4). A partir de ce moment, le combiné commence à communiquer et génère également des trames CAN sur le bus (dont on ne connaît pas la signification).

Le combiné est contrôlé par l'envoi de trames ayant chacune un ID spécifique. Les différentes trames utilisées pour le combiné sont les suivantes :

Figure 28 - Description des trames CAN de contrôle du tableau de bord

En règle générale, lorsque l'on définit une communication CAN ou Flexray, la totalité des messages ainsi que les signaux qui les composent doivent être recensés dans un tableau. On y retrouve des informations sur la longueur de chacun des signaux, leurs valeurs maximum, minimum, l'offset, la résolution, leur valeur initiale, leur unité ainsi que sur leur type.

Ces tableaux sont utilisés dans l'industrie automobile, j'ai eu l'occasion de les utiliser lors du projet effectué chez Bosch au mois de juillet sur la réalisation d'une interface de signaux entre les bus CAN/Flexray et le module PreSensePro. J'y reviendrai plus largement dans la partie 4 (Projet complémentaire). Ci-après, la définition par exemple des signaux composant le message BSI :

	1	2	3	4	5	6	7	8	9	10	11
2	Msg_Name	Type	Sig_Name	Len	min	MAX	Offset	Reso	Unit	Init	Commentaire
3											
4	BSI	ULONG	Dummy	27	0	0	0	1	/	0	
5	BSI	BITFIELD	Mode_Lum	1	0	1	0	1	/	0	1: mode nuit - 0 : mode jour
6	BSI	UBYTE	Luminosite	4	0	15	0	1	/	0	Luminosité
7	BSI	UBYTE	AskWakeUp	8	0	255	0	1	/	0	0x03 : demande réveil - 0x01: phase de vie normale - 0x22 : demande de mise en veille - 0x20 : mise en veille
8	BSI	UWORD	Dummy0	16	0	0	0	1	/	0	
9	BSI	UBYTE	Off_APC	8	0	255	0	1	/	0	0x00 : Arrêt - 0x0C : +APC

Figure 29 - Extrait du tableau de définition des messages CAN

Les signaux Dummy sont des signaux non-utilisés, sans signification.

D'un point de vue software, l'interface de communication entre l'application et l'envoi des signaux CAN est effectuée par un buffer. Les valeurs des signaux sont stockées dans des structures, chacune d'entre elles représentant un message CAN (une trame). Les données sont stockées dans ce buffer par l'application pour l'envoi du message.

Si on prend l'exemple du message CDE_COMBINE_SIGNALISATION, la structure contenant les signaux est la suivante :

```
typedef struct tagMsg_CDE_COMBINE_SIGNALISATION
{
        EcanBufferConfig_t config; /* Structure des octets de configuration (ID, DLC, etc) */
        union
        {
                ULONG64 FullMsg_UN;
                UBYTE   FullMsg_TAB_UB[8];
                struct
                {
                   ULONG      Dummy5          :4  ;  /* valeur nulle */

                   ULONG      Voy_carb_min     :1  ;  /* carburant mini */

                   ULONG      Voy_p           :1  ;  /* voy_p  */

                   ULONG      Voy_ceinture     :1  ;  /* ceinture */

                   ULONG      Voy_airbag       :1  ;  /* airbag passager */

                   ...

                };
        };
        EcanBufferRcvConfig_t rcv_config;  /* Contient le FILHIT, le sender ID et le flag new msg ds le cas d'un msg reçu */
}Msg_CDE_COMBINE_SIGNALISATION_Type;
```

La DMA (Direct Memory Access) est utilisée pour envoyer des données. Il s'agit d'un sous-système du dsPIC qui facilite le transfert de données entre le CPU et les périphériques, sans assistance du CPU. Le transfert est effectué entre les registres de données des périphériques et la mémoire SRAM. Elle utilise la mémoire DPSRAM (dual-ported SRAM) et une structure de registres qui lui permettent d'opérer sur son propre bus d'adresses et de données. Son fonctionnement est donc indépendant du CPU et le transfert se fait en tâche de fond, un plus grand nombre d'opérations peuvent ainsi être effectuées en un temps donné.

Ci-après le schéma bloc illustrant cette séparation de fonctionnement entre la DMA et le CPU :

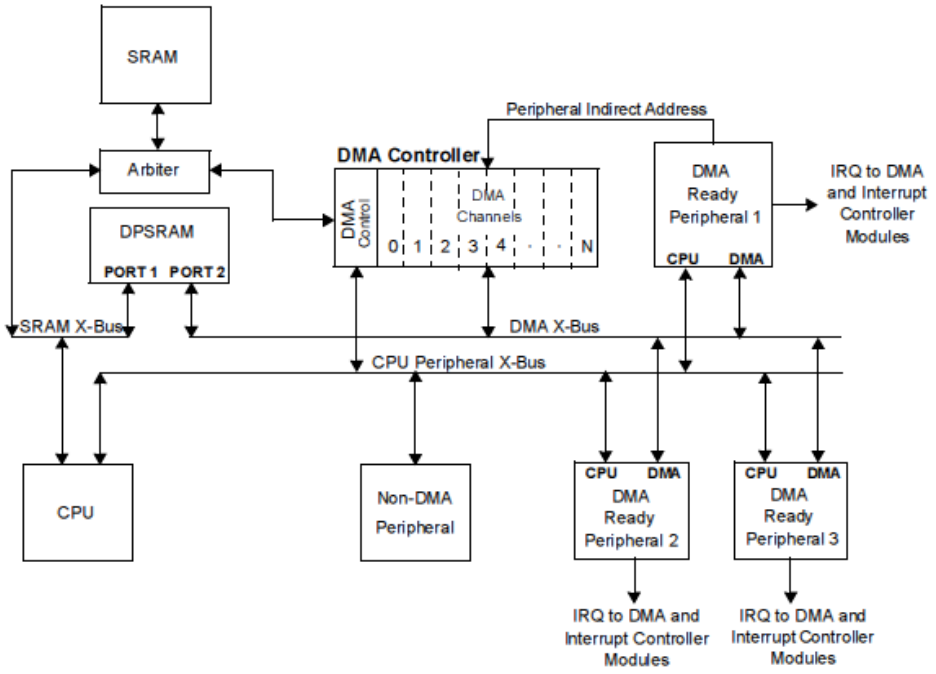

Figure 30 - Schéma bloc du fonctionnement du module DMA [4]

Dans le cas des deux modules ECAN, la DMA va être utilisée en transmission et en réception. Ce sont donc 4 canaux de la DMA qui vont être configurés pour effectuer la communication. La transmission s'effectue par le stockage des données dans les registres C1TXD ou C2TXD (selon le numéro de canal). La DMA se charge de transmettre les données automatiquement au module ECAN pour l'envoi sur le bus.

Inversement, dans le cas d'une réception, la donnée reçue et décodée par le module ECAN est transférée au registre du canal concerné (C1RXD ou C2RXD) (voir figure n°31). Le module DMA se charge en tâche de fond de transmettre les données dans les buffers renseignés lors de la configuration. Des interruptions peuvent être générées lors de l'envoi ou de réception de données. Celles-ci ne sont pas utilisées pour la communication CAN. Les données utilisées dans l'algorithme principal d'acquisition et de traitement des données sont donc en permanence les dernières données CAN reçues. En effet, à chaque calcul de la vitesse ou du régime moteur, la première chose effectuée est la récupération des données converties par l'ADC. L'interruption DMA est en revanche utilisée pour le module de conversion AD C que nous détaillerons par la suite.

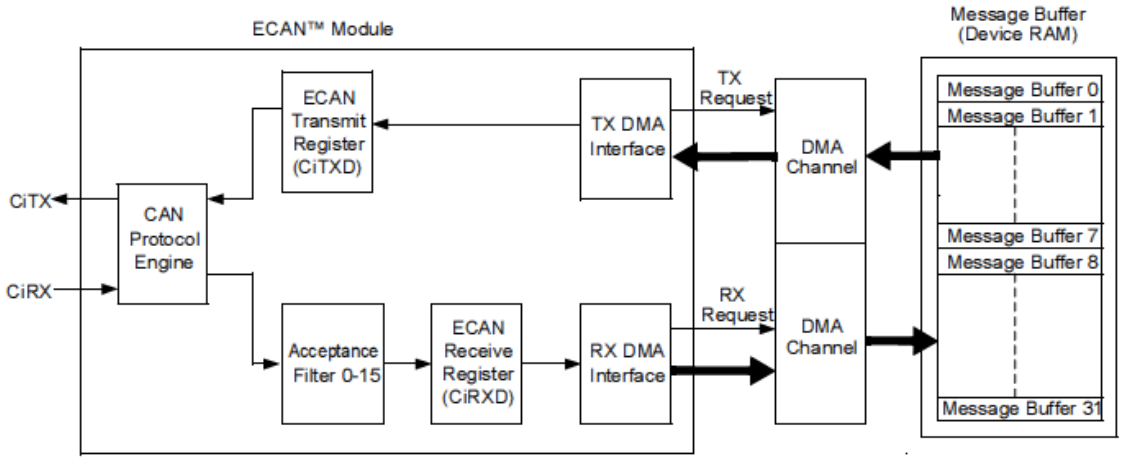

Figure 31 - Transmission des informations entre le module ECAN et le module DMA [5]

L'oscillogramme suivant représente la composition du message CAN « BSI RAPIDE » envoyé sur le bus et ayant pour identifiant standard 0x0B6 :

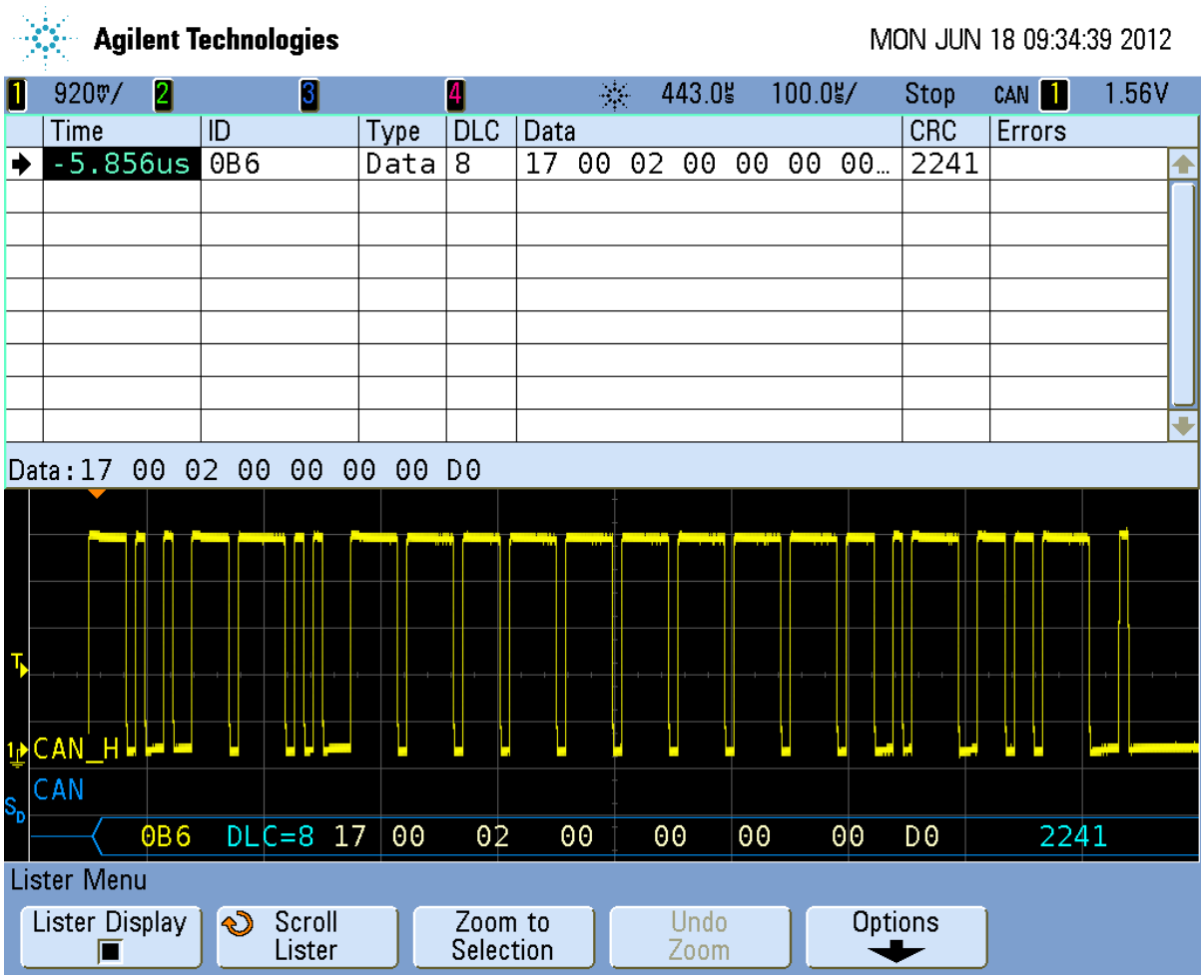

Figure 32 - Trame CAN "BSI_RAPIDE" (contient les informations de vitesse et de régime moteur)

Ici, on peut observer la composition de la trame. L'oscilloscope étant dans ce cas configuré pour détecter les trames CAN pour une transmission Low Speed 125kb/s, il peut extraire les valeurs des différents champs de la trame, à savoir :

- le début de trame (SOF),
- le champ d'arbitrage composé de l'ID (ici valant 0x0B6) et RTR (ici bit dominant car il s'agit d'une trame de données),
- le champ de commande composé du DLC (Data Length Control) valant 8 (nombre d'octets de données),
- le champ de données qui contient autant d'octets que le nombre défini par le DLC.
- le champ de CRC pour la vérification de la bonne transmission de la trame,
- le champ d'acquittement (Acknowledge),
- la fin de trames composée de 7 bits récessifs.

Le message BSI RAPIDE contient les signaux de régime moteur et de vitesse. Le premier octet de données représente la valeur des tr/min et le 3ème octet représente la valeur de la vitesse. La valeur du dernier octet est définie à l'initialisation.

L'oscilloscope peut également détecter la présence d'erreur et le type d'erreur rencontrée.

Les figures suivantes représentent la transmission des trames CAN sur le bus tout au long d'un cycle :

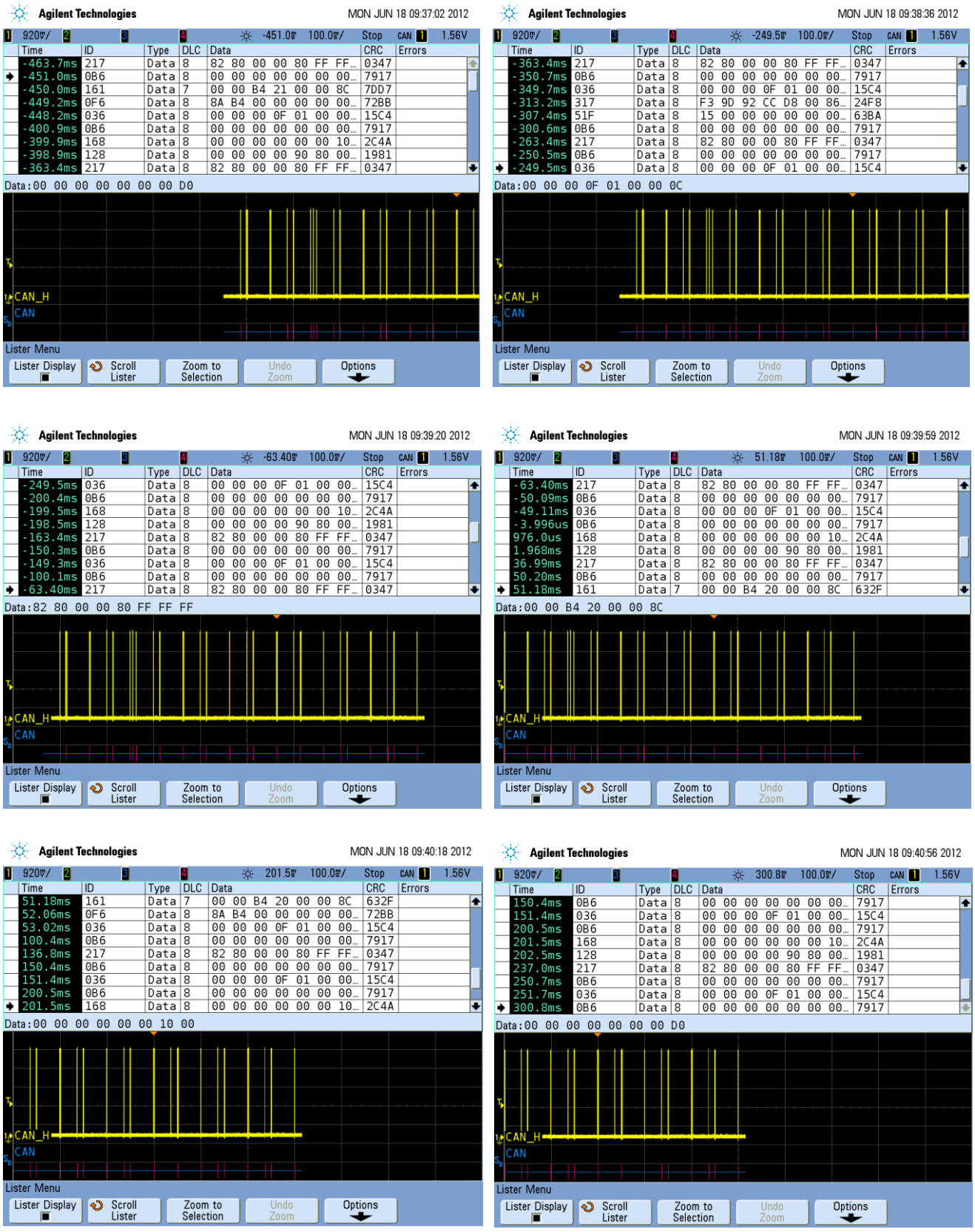

Figure 33 - Emission des trames CAN sur le bus tout au long d'un cycle

Un cycle est effectué en 500 ms car la plus longue périodicité de l'envoi de message sur le bus est de 500 ms (lors de l'envoi des messages BSI LENTE (ID = 0x0F6) et ETAT BSI LENTE NIVEAU (ID = 0x161)).

On constate bien que les messages sont envoyés à intervalles de temps réguliers, comme définit précédemment. Cette périodicité d'envoi est obtenue grâce à l'utilisation d'un timer. L'envoi est en effet effectué à chaque interruption du TIMER, dont la période d'interruption est définie à 50ms. Comme vu dans la strucutre générale du software, l'actualisation des données est effectuée toutes les 150 ms, ainsi, le champ de données de la structure du message CAN dans la DMA est actualisé toutes les 150 ms. Il faut ensuite, avant l'envoi de chaque trame, remplir les autres champs consacrés notamment à l'ID et au DLC. Sur la figure suivante, on peut observer comment est reconstituée la trame CAN à partir des registres du module ECAN.

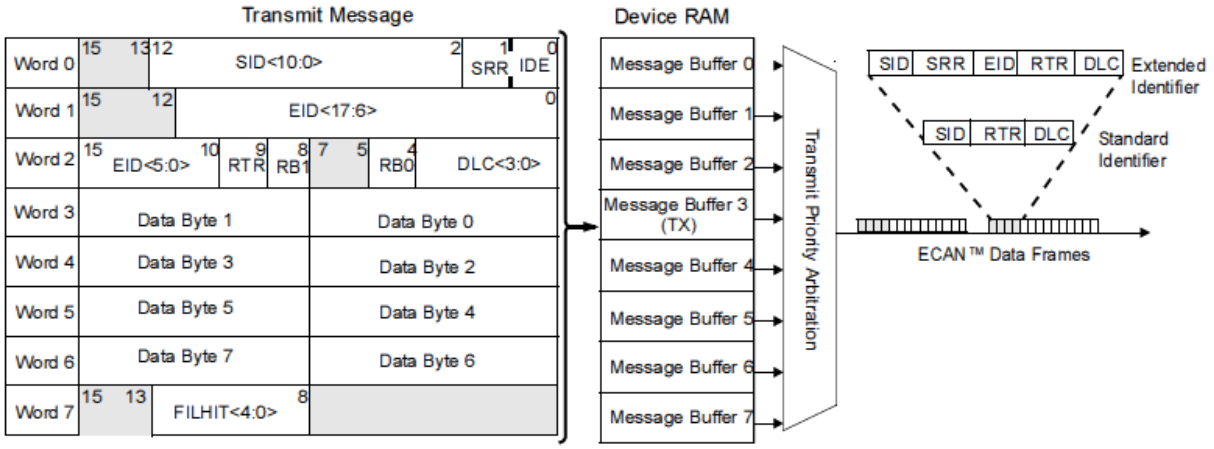

Figure 34 - Génération d'une trame CAN sur le bus à partir des registres du module ECAN [5]

Sur ces oscillogrammes, on peut également constater que certains messages circulant sur le bus ne font pas parties des messages générés par notre carte de contrôle. Il s'agit des messages ayant pour identifiants 0x217, 0x317 et 0x51F. Ces messages sont en fait générés par le combiné lui-même. Nous n'avons pas d'informations précises sur la signification de ces trames, c'est pourquoi nous les avons ici ignorées. Il s'agit de trames de commandes, probablement pour la gestion de la ventilation située à proximité du combiné entre autres

2.3.2.2. Acquisition des données analogiques

Le module ADC utilise également la mémoire DMA pour la conversion des données. Sur le schéma suivant, on peut observer un exemple de fonctionnement de cette conversion :

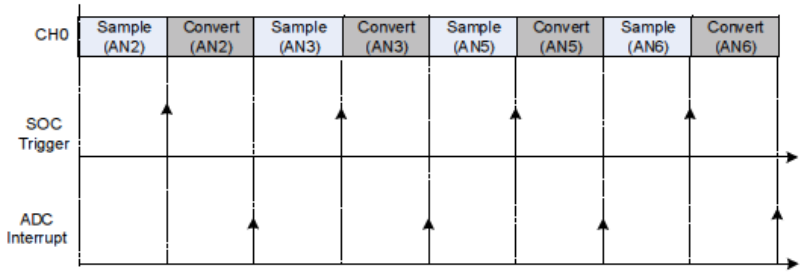

Figure 35 - Gestion de l'échantillonnage et de la conversion ADC pour de multiples canaux [6]

L'acquisition et la conversion des données analogiques en entrée sont effectuées les unes après les autres. 2 étapes composent la phase de conversion ADC : l'échantillonnage (« sample ») et la conversion (« convert »).

L'interruption ADC associée à la DMA n'est effectuée que lorsque toutes les entrées scannées sont converties une fois. A chacune de ces interruptions, les variables contenant les données converties sont mises à jour et mises en forme. En effet, la valeur d'une donnée convertie est codée sur 16 bits. Les 10 bits de poids fort représentent la partie significative de la conversion. Il faut donc effectuer un décalage de 6 bits pour obtenir la valeur convertie.

L'algorithme principal démarré à chaque interruption de timer (voir la structure générale du software) doit avoir accès à ces données converties. C'est accès est obtenu par l'intermédiaire de getteurs. Par exemple, la fonction suivante permet de récupérer la valeur de l'intensité lumineuse, comprise entre 0 et 1024. CaptLumi est la variable locale du fichier source consacré à l'ADC qui contient la valeur convertie actualisée à chaque interruption ADC.

```
UWORD FUN_GetValCaptLumi_UW()
{
   return CaptLumi;
}
```

2.3.2.3. *Modèle simulant la vitesse et le régime moteur du véhicule*

Nous souhaitons que la maquette soit la plus réaliste possible. Il convient donc de concevoir un système qui représente le plus fidèlement possible le comportement du véhicule lors de l'actionnement des pédales (accélération et freinage).

Le calcul de la vitesse et du régime moteur est effectué à chaque interruption timer, avant son envoi par CAN au combiné. Les consignes de couple et de freinage sont nécessaires à ces calculs. Le traitement des consignes de couple, qu'elles viennent de l'application Bluetooth, de l'application PC ou du pédalier, est réalisée par la fonction FUN_VitesseCombi_V. Cette fonction prend comme paramètres : les valeurs de consignes de vitesse, de freinage, ainsi que le numéro de rapport de vitesse enclenché.

Nous disposons des abaques suivants, représentant les caractéristiques techniques de la Peugeot 207 RC :

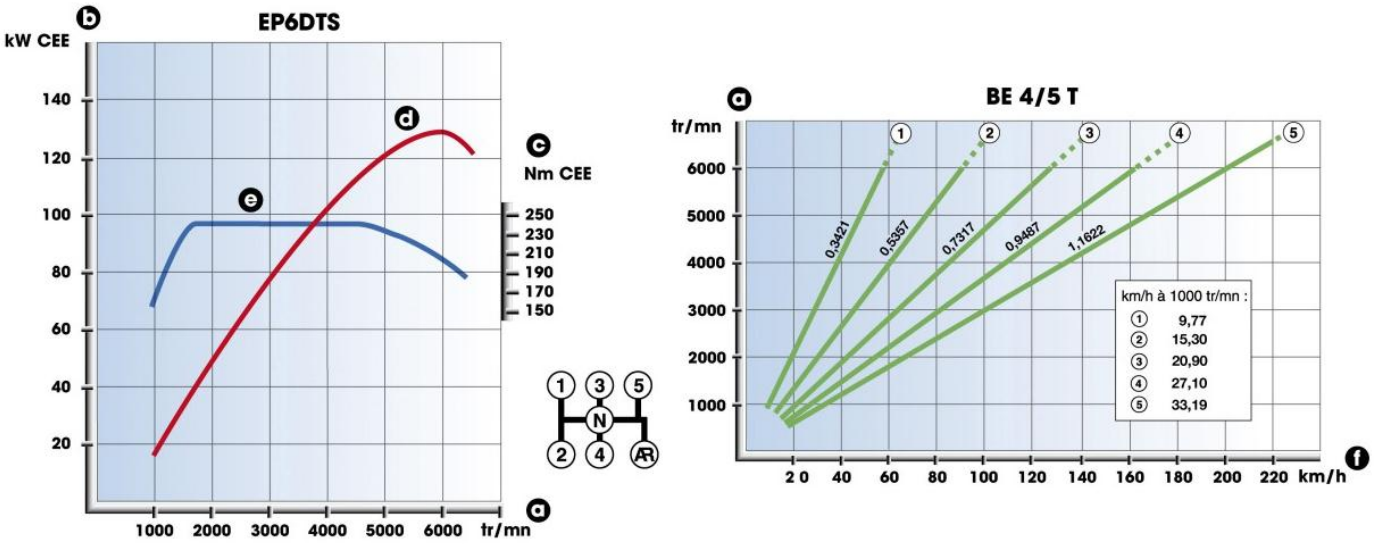

Figure 36 - Abaques de couple moteur/régime moteur et de régime moteur/vitesse du véhicule [7]

La consigne de rotation du moteur est déduite de la consigne angulaire d'appui sur la pédale par l'utilisateur. On considère qu'un appui maximal sur la pédale d'accélération consiste à atteindre à terme la vitesse de rotation maximale du moteur (6500 tr/min).

Le couple délivré par le moteur est connu grâce à l'abaque de gauche de la figure n°36 représentant la valeur du couple en fonction du régime moteur. On peut approximer la courbe par 4 droites, selon la valeur du régime moteur : de 0 à 1000 tr/min, de 1000 à 1700 tr/min, de 1700 à 4700 tr/min et pour des valeurs supérieures à 4700 tr/min.

Le couple à l'instant t est calculé par rapport au régime moteur réel : Couple = f(RotationReel).

Le régime moteur à l'instant t est calculé par rapport à la vitesse du combi à l'instant t.

Le régime moteur de consigne est calculé par rapport à la consigne de vitesse, en utilisant l'abaque régime/vitesse et la valeur du rapport enclenché : $N_{cons} = k(rapport) * v_{cons}$

Les données supplémentaires telles que la masse du véhicule, le coefficient de frottement, etc. sont tirés de la « Fiche technique - PEUGEOT - 207 RC » [8].

Application du principe fondamental de la dynamique :

$$\sum \vec{F} = m * \vec{a}$$

et

$$a = \frac{dv}{dt} \rightarrow v(t + dt) = v(t) + a * dt$$

Trois forces principales sont présentes dans notre système :

La force de traction T, la force de frottement de l'air $F_{frott\ air}$, la force de frottement des pneus F_{frott} et la force de freinage $F_{freinage}$.

- La force de frottement de l'air vaut :

$F_{frott\ air} = 0{,}5 * \rho * S * Cx * v^2$, avec :

- ρ la masse volumique du fluide (air sec) en kg/m^3
- S la surface frontale du véhicule
- Cx le coefficient de frottement dépendant de la forme du solide
- v la vitesse du véhicule

On connaît : ρ = 1,184 kg/m^3 et SCx = 0,7

- La force de frottement des pneus sur le sol vaut :

$F_{frott} = m * g * f$, avec :

- m la masse du véhicule,
- g la constante de pesanteur
- f le coefficient de frottement.

On connaît : m = 1250 kg, g = 9,81 m/s^2 et f = 0,015.

- La force de freinage vaut :

$F_{freinage} = m * g * f_{freinage}$, avec :

- m la masse du véhicule,
- g la constante de pesanteur
- $f_{freinage}$ le coefficient de freinage, dépendant de l'appui sur la pédale de frein.

On connaît : m = 1250 kg, g = 9,81 m/s^2.

- La force de traction vaut :

$T = \frac{P_{mot}}{v} = \frac{C_{réel} * \Omega_{cons}}{v_{réelle}} = \frac{C_{réel} * 2\pi * N_{cons}}{v_{réelle} * 60}$, avec :

- $C_{réel}$ le couple moteur à l'instant t
- $v_{réelle}$ la vitesse à l'instant t
- N_{cons} la consigne de régime moteur

On a donc la formule suivante :

$$a = \frac{T - F_{frott\ air} - F_{frott} - F_{freinage}}{m}$$

La vitesse à l'instant t+dt vaut donc :

$$v(t+dt) = v(t) + \frac{dt}{m} * \left(\frac{C_{réel} * 2\pi * N_{cons}}{v(t) * 60} - \left(0{,}5 * \rho * S * Cx * v(t)^2 + m * g * \left(f + f_{freinage}\right)\right)\right)$$

$$v(t+dt) = v(t) + \frac{T_{timer}}{1250} * \left(\frac{C_{réel} * 0{,}105 * N_{cons}}{v(t)} - \left(0{,}5 * 1{,}184 * 0{,}7 * v(t)^2 + 1250 * 9{,}81 * \left(0{,}015 + f_{freinage}\right)\right)\right)$$

2.3.2.4. Communication série UART

Le port série de l'ordinateur est lié par un câble null-modem à la carte. Un logiciel est ensuite utilisé pour lire et écrire sur le port série. Il suffit de définir les caractéristiques de la communication, à savoir la vitesse, la présence d'un bit de stop et d'un bit de parité.

La communication série avec l'ordinateur répond à deux objectifs principaux :

- Faciliter le débogage du programme : possibilité d'afficher certaines valeurs via le logiciel précédemment cité.
- Réaliser l'initialisation du module Bluetooth. Ce dernier est en effet livré avec un logiciel « Simply Blue Commander » qui permet d'envoyer les trames adaptées pour définir et modifier certaines de ses caractéristiques. Ce processus est décrit dans la partie suivante.

2.3.2.5. Communication Bluetooth

Le module communique directement avec le PIC par liaison UART. On distingue différents modes : un mode de commande du module et un mode transparent qui correspond à la transmission de données.

Lors de la première utilisation du module Bluetooth, il est indispensable de reconfigurer ses caractéristiques et d'obtenir des informations sur le module lui-même, comme son adresse. Le logiciel « Simply Blue Commander » permet de réaliser ces opérations plus facilement. Chacune des requêtes de commande est préenregistrée et le logiciel envoie la trame complète sur le port série.

Le module bluetooth ne peut pas communiquer directement avec le PC par UART. On utilise donc le dsPIC comme intermédiaire entre le PC et le module. Les données reçues sur un des modules UART par le dsPIC sont directement retransmises sur le second module.

Une trame de commande a cette forme-là :

Start delimiter	Packet type	Operation code	Data length	Check-sum	Data	End delimiter
1 byte	1 byte	1 byte	2 bytes	1 byte	<data length> bytes	1 byte

Figure 37 - Trame de commande du module Bluetooth

Les caractéristiques de la communication sont les suivantes :

- vitesse : 9600 bauds,
- bit de stop,
- pas de bit de parité.

2.3.2.5.1. Première utilisation

Pour notre application, la carte de contrôle du combiné est utilisée comme périphérique bluetooth et le dispositif Androïd comme Host. C'est ce dernier qui va établir la connexion avec le périphérique. Pour cela, il est nécessaire de connaître l'adresse du périphérique (module Bluetooth). On utilise le logiciel de configuration du module qui va générer une instruction de commande « Read Local Device Address », ayant la forme suivante : 02 52 05 00 00 57 03. Le module répond par l'envoi d'une trame de commande contenant son adresse en hexadécimal : 00 18 B2 01 58 0C.

Il faut ensuite mettre le périphérique en mode transparent pour l'envoi et la réception de données. L'instruction de commande est : 02 52 11 01 00 64 01 03.

2.3.2.5.2. Gestion de la réception des trames de commande et de données

Le dispositif Androïd génère des trames de commande et de données pour communiquer avec le périphérique.

Le protocole est le suivant. Lorsque le dispositif Androïd désire se connecter au périphérique, il envoie une requête à ce dernier. La connexion est alors établie et les octets suivants contiennent les données brutes. De même, le dispositif génère deux trames de commande bien particulières pour indiquer au périphérique le souhait d'interrompre la communication.

Il est nécessaire de détecter ces trames. Comme on ne reçoit qu'un octet à la fois, il faut détecter le début de trame et enregistrer l'état d'avancement dans la trame. On a défini trois types pour gérer la réception de trames :

Le type BTH_RECEIVED_MSG va permettre d'orienter l'enregistrement de chaque partie d'une trame de commande :

```
typedef enum{
   START_DELIMITER=0,
   PACKET_TYPE,
   OPERATION_CODE,
   DATA_LENGTH_FIRST_BYTE,
   DATA_LENGTH_SECOND_BYTE,
   DATA,
   END_OF_MSG
}BTH_RECEIVED_MSG;
```

Le second type est une structure qui va contenir les valeurs des différentes parties d'une trame de commande :

```
/* Structure d'une trame de commande */
typedef struct __attribute__((aligned(16)))
{
   uint8_t StartDelimiter_u8 :8;
   uint8_t PacketType_u8     :8;
   uint8_t OperationCode_u8  :8;
   uint16_t DataLength_u16    :16;
   uint8_t CheckSum_u8       :8;
   uint8_t EndDelimiter_u8   :8;
   uint8_t pdata[10];
}TRAME_BTH;
```

Le dernier type permet de définir quelles sortes de données sont reçues et quelles actions doivent donc alors être effectuées :

```
typedef enum{
    VOYANTS_MSG,
    RAPPORT_MSG,
    SPEED_MSG
}BTH_DATA_MSG;
```

A chaque fois qu'une donnée est reçue, il faut vérifier si la réception d'une trame de commande est déjà en cours et si oui, à quel stade (Start Delimiter, Data Length, Data, etc.). On détecte une trame de commande si le premier octet vaut 0x02. Si tel est le cas, le cheminement décrit à la figure n°38 est effectué.

Si en revanche la réception d'une trame de commande n'est pas en cours et que l'octet reçu ne vaut pas 0x02, il s'agit d'un octet de données indiquant les consignes demandées par l'utilisateur (accélération, freinage, changement de rapport, etc.). Les données sont alors enregistrées dans les variables locales correspondantes.

La figure n°38 décrit le fonctionnement de la réception Bluetooth.

Concernant la réception des données brutes (cas où l'octet reçu vaut 0x02), le protocole de communication est défini dans la partie n°3.2.3.

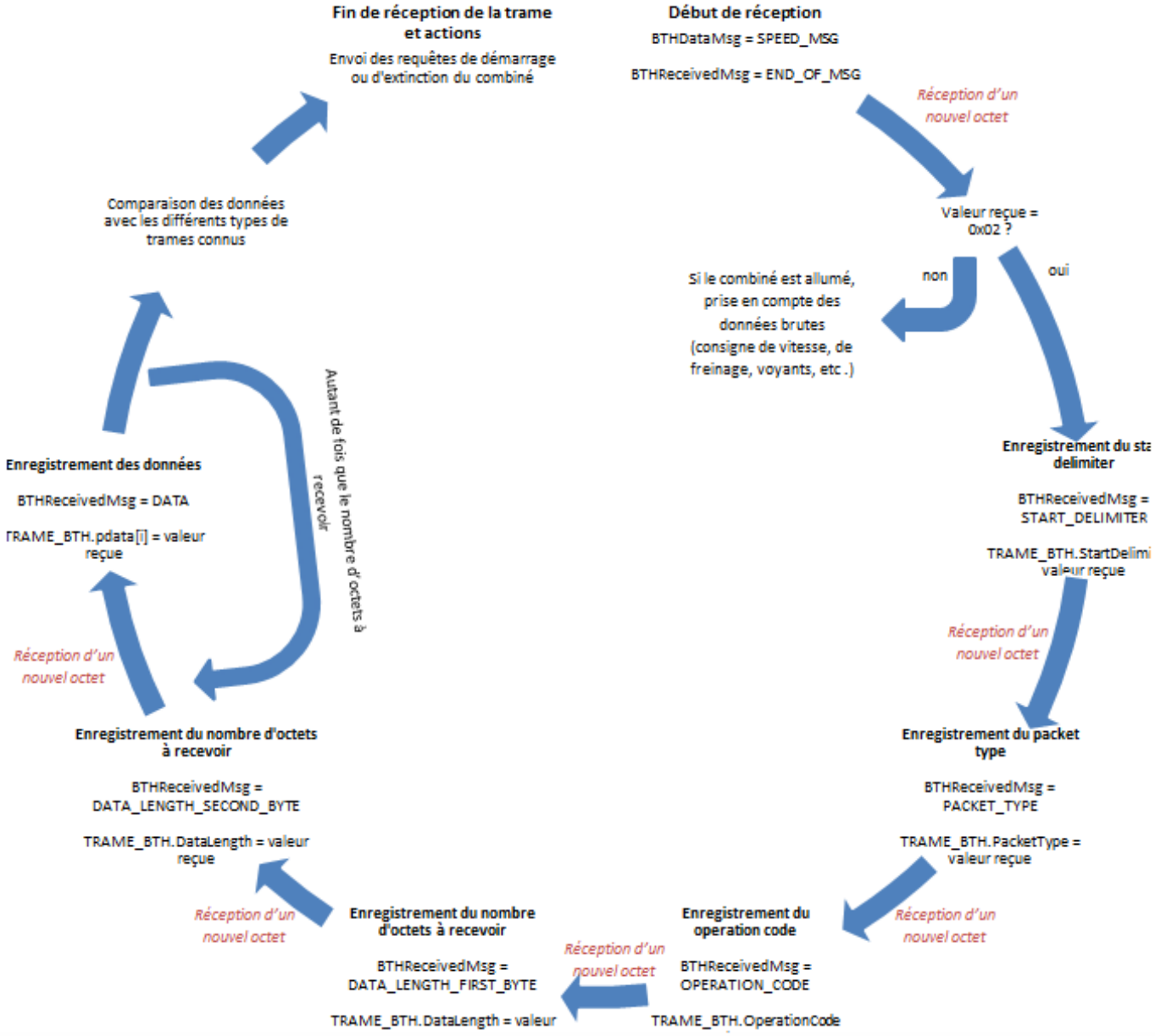

Figure 38 - Protocole de communication établi pour la réception Bluetooth

2.3.2.6. Communication USB avec le dsPIC33E

2.3.2.6.1. Etablissement de la communication USB

Le dsPIC33E256MU806 dispose d'un module USB intégré. Le diagramme d'interface se trouve en annexe n° 1.

Comme décrit dans la section USB de la documentation technique du dsPIC33E [9], le module USB est composé de plusieurs sous-éléments :

- Un transceiver USB interne réalisant la fonction d'intermédiaire entre le bus USB et les données numériques logiques.
- Une machine à état nommée Serial Interface Engine (SIE) permet le transfert de données en provenance et à destination des buffers Endpoint et génère le protocole hardware pour le transfert de données.
- Le USB Bus Master qui transfère les données entre les buffers de la RAM et la SIE.

Le module USB du dsPIC laisse la possibilité de choisir entre 3 modes d'opération :

- Mode périphérique (device),
- Mode master (host),
- Mode On-The-Go, qui permet de passer du mode master au mode périphérique et inversement.

Pour notre application, la carte est vue comme un périphérique. Ainsi, c'est le PC qui gère la communication.

La mise en place d'une communication USB à partir du dsPIC nécessite que les descripteurs de configuration, d'interface et d'endpoint soient en accord avec le driver utilisé et installé sur le PC.

Nous ne souhaitons pas que l'hôte interroge sans arrêt le périphérique, mais simplement qu'il lui envoie les données continuellement. Il est préférable également que toutes les données soient assurément transférées. C'est pourquoi nous choisissons ici un mode de transfert Bulk en Full Speed. Microchip annonce qu'avec cette configuration, le débit de transmission des données brutes est de 1,22Mbps.

Etant donnée l'utilisation du transfert Bulk, pour lequel la taille maximale d'un paquet de données est de 64 octets, l'utilisation d'un seul Endpoint suffit à satisfaire la transmission de toutes les données entre le microcontrôleur et le PC. Un driver qui utilise cette configuration est le LibUSB.

Microchip fournit un certain nombre de librairies et exemples d'applications. La librairie LibUSB fournie par Microchip permet d'établir la communication entre le PC et le dsPIC. Elle est composée de tous les fichiers headers et C contenant les fonctions qui effectuent les différentes étapes de la communication (initialisation, transfert de données, arrêt de la communication).

Il y a deux manières différentes de gérer la communication : soit en polling (géré par le concepteur), soit par interruption. Afin de garder le bus actif, la fonction réalisant les tâches quelles qu'elles soient en rapport avec l'USB doit être appelée toutes les 1,8ms au minimum. En utilisant les interruptions, on s'affranchit de cet appel, les tâches sont effectuées automatiquement lors de chaque interruption.

Selon l'état de connexion du périphérique, la tâche effectuée sera différente. On dénombre 7 états :

- DETACHED_STATE : périphérique détaché du bus.
- ATTACHED_STATE : périphérique attaché au bus, mais le port n'est pas configuré.
- POWERED_STATE : périphérique attaché au bus et le port est configuré.
- DEFAULT_STATE : état après réception de la commande RESET du maître.
- ADR_PENDING_STATE : le périphérique a reçu la commande SET_ADDRESS.
- ADDRESS_STATE : l'adresse propre au périphérique est sur le bus.
- CONFIGURED_STATE : le périphérique est correctement configuré, il peut alors procéder aux tâches spécifiques à l'application.

Un périphérique reconnu et prêt à être utilisé par l'ordinateur est nécessairement dans l'état CONFIGURED_STATE. Les échanges de paquets de données peuvent alors débuter.

2.3.2.6.2. Gestion de la réception et de l'envoi des trames

A chaque réception d'une trame complète, un flag est mis à 1 et lors de l'interruption, la fonction de callback associée « USBCBTransferCompleted » est appelée. Dans cette structure, on va récupérer les données reçues et les traiter.

La structure OUTPacket contient ces données. Les données sont enregistrées dans les variables locales en accord avec le protocole de communication défini dans la partie 2.3.4.3.

2.3.3. Interface graphique Androïd

L'interface graphique est réalisée en Java à l'aide de l'environnement de développement Eclipse. Cet environnement facilite la réalisation d'applications Androïd en offrant notamment un outil de création de la partie graphique, qui génère le code XML correspondant automatiquement. Il est évidemment possible d'avoir accès à ce code afin d'y apporter directement des modifications.

2.3.3.1. Présentation de l'interface

Figure 39 - Bluetooth déconnecté, combiné éteint

Figure 40 - Bluetooth en cours de connexion ou de déconnexion

Figure 41 - Bluetooth connecté, combiné allumé

Utilisation de l'interface :

- Le démarrage ou l'arrêt du combiné (et de la communication Bluetooth) est effectué par l'appui sur le bouton on/off situé en haut de l'interface. Un message apparaît quelques secondes lors du démarrage ou de l'arrêt du combiné.
- Les voyants sont en réalité des boutons, vides lorsque le voyant est éteint, pleins lorsqu'il est allumé.
- L'appui sur un des clignotants déclenche son clignotement.
- Les boutons + et – permettent d'augmenter ou diminuer les rapports de vitesse.
- Les boutons frein et accélérateur remplacent les pédales, il faut rester appuyé dessus pour avoir une action prolongée similaire au fonctionnement des vraies pédales.

2.3.3.2. Définition du protocole de communication

Le protocole de communication est défini de la manière suivante :

- Lorsque la connexion entre le périphérique et le téléphone est établie, le module Bluetooth génère une trame bien particulière au dsPIC. La réception de cette trame est testée pour autoriser ou non le démarrage du combiné et commencer à prendre en compte les valeurs reçues.

- Les informations suivantes doivent être transmises à la carte :
  - Appui ou non sur le bouton d'accélération,
  - Appui ou non sur le bouton de freinage,
  - Valeur du rapport enclenché,
  - Etat des voyants (clignotants et feux)

  Les informations d'accélération et de freinage sont envoyées toutes les 175ms. Ces deux données sont transmises chacune sur 1 octet :
  - Envoi de 0x76 → appui sur le bouton d'accélération
  - Envoi de 0x77 → bouton d'accélération relâché
  - Envoi de 0x66 → appui sur le bouton de freinage
  - Envoi de 0x67 → bouton de freinage relâché

  L'état des voyants et le rapport enclenché sont envoyés dès que leur état est modifié (donc dès que l'on appuie sur tout autre bouton que les boutons on/off, accélérateur et frein). Deux octets sont consacrés à l'état des voyants et deux autres sont consacrés au rapport enclenché :
  - 1er octet : 0x72 prévient le dsPIC que l'octet suivant contient les données concernant les voyants.
  - 2nd octet :
    - bit 1 : clignotant gauche,
    - bit 2 : clignotant droit,
    - bit 3 : antibrouillard arrière,
    - bit 4 : antibrouillard avant,
    - bit 5 : feux de route,
    - bit 6 : feux de croisement,
    - bit 7 : feux de position.
  - 3ème octet : 0x74 annonce la réception de l'octet contenant l'information sur le rapport enclenché.
  - Le dernier octet contient la valeur du rapport enclenché.

- A l'initialisation, un premier message est envoyé au dsPIC, indiquant que tous les voyants sont éteints et que le premier rapport est enclenché.

- Une requête d'arrêt de la communication peut être envoyée par le périphérique (la carte), il s'agit de la suite d'octets représentant la chaîne de caractères « stop ».

2.3.3.3. Structure et fonctionnement du programme

La programmation d'une application Androïd s'effectue en Java.

Le programme est constitué de 4 classes :

- **BtInterface** : gestion de la communication Bluetooth (démarrage et arrêt de la communication, réception et envoi des données).
- **Combi** : gère l'interface graphique, les différents évènements liés aux boutons et la mise en forme des trames avant envoi.
- **CommandeInterface** : contient le thread destiné à envoyer les informations d'appui des boutons d'accélération et de freinage.
- **ClignoInterface** : contient le thread de gestion du clignotement des clignotants dans l'interface graphique (clignotement de période 1s).

Le fonctionnement du programme est basé sur l'utilisation des Threads. Cela facilite l'utilisation « simultanée » des éléments d'interface (évènements) et de communication. Les délais d'affichage sont alors plus aisément maîtrisés et respectés.

A chaque thread est associé un handler. Les handlers permettent de communiquer entre les threads et les différentes classes. Ils peuvent contenir des informations sous forme de messages. Nous allons les utiliser pour communiquer entre l'affichage de l'interface graphique géré par la classe Combi et les threads définis dans les autres classes.

Nous allons utiliser plusieurs threads :

- un pour la connexion Bluetooth avec le périphérique (le module Bluetooth de la carte),
- un pour la réception des données Bluetooth,
- un autre pour gérer l'envoi périodique de l'information pédale de vitesse ou de frein appuyée ou relâchée,
- un dernier pour le clignotement des voyants de clignotants dans l'interface.

2.3.3.4. Gestion de la communication Bluetooth SPP

2.3.3.4.1. Etablissement et arrêt de la connexion Bluetooth

Différents états sont définis pour la connexion Bluetooth :

- STATE_NONE
- STATE_CONNECTED
- STATE_DISCONNECTED

Selon l'état actuel de la communication, l'appui sur le bouton On/Off va soit fermer la communication Bluetooth, soit créer une nouvelle instance de la classe BtInterface et donc une nouvelle communication. Cette création va initialiser la communication Bluetooth, c'est-à-dire :

- Réaliser la détection de tous les périphériques Bluetooth,
- Rechercher si l'adresse du périphérique Bluetooth souhaité fait partie de la liste trouvée.
- Créer le BluetoothDevice associé.
- Créer un « connecteur » (socket) qui va ouvrir une communication basée sur le protocole SPP et ouvrir les flux de données d'entrée et de sortie.

A l'appui sur le bouton on/off, un nouveau thread est créé. Dans ce cas, le message transmis au handler associé indique si la communication est établie ou non. Le handler est déclaré dans la classe Combi qui gère l'interface graphique.

Par exemple, lorsque la communication est établie lors de l'exécution du thread, le message est modifié et envoyé au handler via l'appel à la fonction handler.sendMessage(msg), « msg » étant le message précédemment modifié et « handler » le handler précédemment cité. Cet envoi déclenche l'exécution des tâches définies dans le handler, avec le message associé. Dans notre cas, un message va alors s'afficher à l'écran, informant l'utilisateur que le combiné a été démarré.

Comme vu précédemment dans le protocole de communication, la carte peut envoyer une requête au dispositif Android pour stopper la communication Bluetooth et ainsi demander l'arrêt du combiné. Il suffit que la carte envoie « stop » au dispositif Android pour déclencher cette procédure d'arrêt. Le thread de réception de données scrute en permanence le flux d'entrée de données et lorsqu'une donnée est disponible, la chaîne de caractères est stockée dans le message transmis au handler associé. Si la chaîne de caractères est « stop » et que la communication Bluetooth est bien active à cet instant, alors l'arrêt de la communication est désiré.

2.3.3.4.2. Acquisition et traitements des actions de l'utilisateur

L'interface dispose de différents boutons, chacun d'entre eux étant associé à un évènement précis. Pour notre application, on trouve deux types d'évènements distincts :

- OnClick : évènement déclenché lorsque l'utilisateur appuie et relâche un bouton

Cet évènement concerne les boutons on/off, de voyants, d'incrémentation ou décrémentation des rapports de vitesses. La fonction OnClick est donc appelée à chaque clic sur un de ces boutons. Les ID des boutons sont scrutés pour savoir lequel a été appuyé, puis l'action associée peut être effectuée.

Dans le cas des voyants, il faut gérer la mise en forme de l'octet contenant les informations sur les états des voyants en prenant en compte ces états avant le déclenchement de l'évènement actuel.

Si l'état des clignotants est modifié, le clignotement des clignotants peut être interrompu ou enclenché.

Si n'importe quel autre bouton que le bouton on/off est appuyé, l'envoi de l'état des voyants et du rapport de vitesse est effectué (utilisation de la fonction sendData de la classe BtInterface).

- OnTouch : évènement déclenché lorsque l'utilisateur touche un bouton. Il peut ensuite y avoir une distinction de l'action effectuée

Les boutons d'accélération et de freinage utilisent ce type d'évènement. On utilise deux booléens pour définir les états des deux boutons. Ainsi, si l'utilisateur appuie sur un des deux boutons (ACTION_DOWN), le booléen associé passe à true. Si en revanche il relâche le bouton (ACTION_UP), le booléen passe à false. On connaît donc en permanence l'état de ces deux boutons. Le message passé au handler associé au thread d'envoi de ces deux informations est directement mis à jour.

2.3.4. Interface graphique PC

2.3.4.1. Présentation de l'interface

Figure 41 - Interface graphique de l'application sur PC

Utilisation de l'interface :

- Les voyants sont en réalité des boutons, vides lorsque le voyant est éteint, plein lorsqu'il est allumé.
- L'appui sur un des clignotants déclenche son clignotement.
- Le champ de sélection des rapports permet d'augmenter ou diminuer les rapports de vitesse.
- Les boutons frein et accélérateur remplacent les pédales, il faut rester appuyé dessus pour avoir une action prolongée similaire au fonctionnement des vraies pédales.

2.3.4.2. *Structure du programme*

Le software est basé sur une architecture Modèle-Vue-Contrôleur (MVC). Cette architecture a pour avantage d'accroître l'indépendance entre les différents composants.

Les classes composant le programme sont les suivantes :

- Le modèle contient les données : états des voyants, vitesse du véhicule et régime du moteur. Il procède aux différents calculs et conversions de données pour fournir à la vue les données nécessaires à l'affichage dans le bon format.
- Le contrôleur reçoit les requêtes de l'interface graphique (valeur de la vitesse, etc.) et demande les infos au modèle qui calcule et renvoie les résultats. Il peut aussi demander à l'interface de se mettre à jour en générant un évènement qui va faire appel à une fonction de la vue.
- La vue contient tous les évènements de l'interface graphique et fait appel aux méthodes du contrôleur.
- La dernière classe, USBCommunication, contient les méthodes pour gérer la communication USB : connexion, déconnexion, envoi de trame, réception de trame.

L'instance de classe du contrôleur est créée dans la vue, les instances de classe du modèle et de la classe USBCommunication sont quant à elles créées dans le contrôleur.

Ainsi, la création d'une instance de la vue entraîne la création de tous les autres objets. Le problème se pose lorsque le contrôleur désire par exemple ordonner la mise à jour de l'affichage de la vue. Il n'a en effet pas la possibilité d'appeler une fonction de la classe Vue. Le diagramme de la figure n° 43 récapitule les différentes interactions entre les classes.

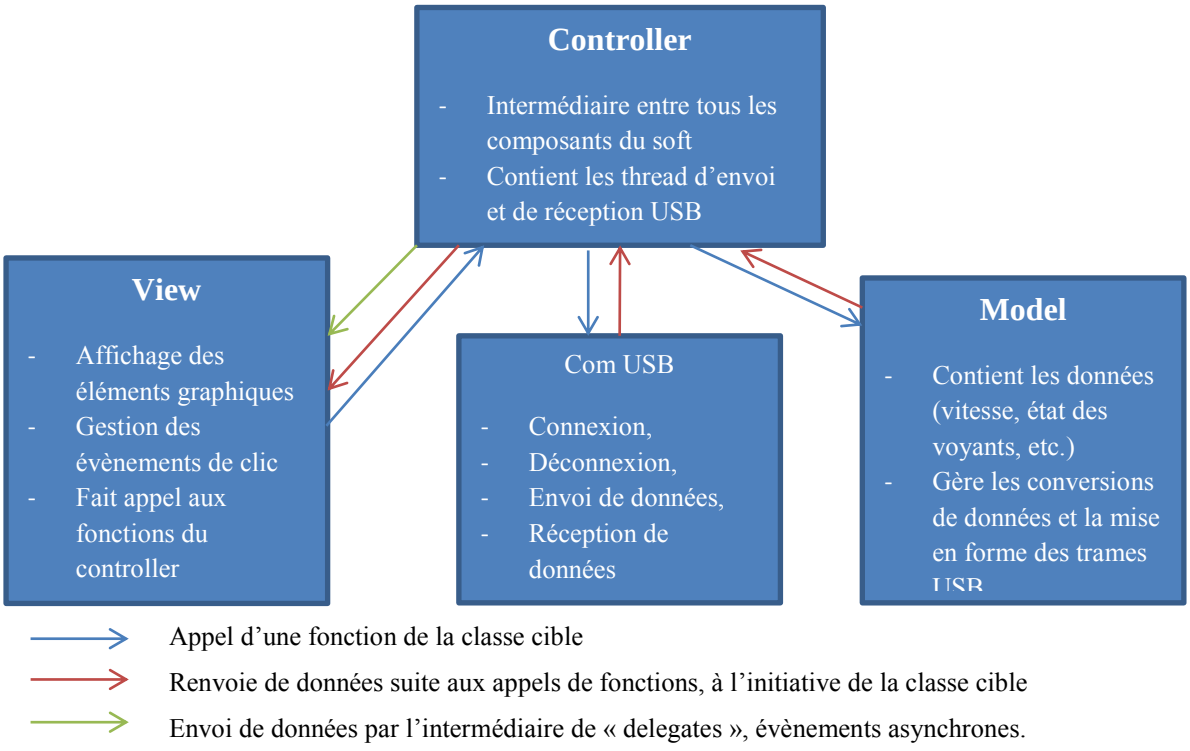

Figure 42 - Diagramme du modèle MVC

La flèche verte indique un flux de données du contrôleur vers la vue, à l'initiative du contrôleur. Cela signifie que le contrôleur a la possibilité de déclencher des actions dans la classe vue. Cela n'est possible que par l'intermédiaire de « delegates » qui déclenche un évènement dans la classe vue et l'exécution de la fonction qui est associée.

Dans notre application, cette technique est entre autres utilisée pour mettre à jour l'affichage de la vitesse, du régime moteur et la position des aiguilles à chaque réception d'une trame. La figure n°44 ci-dessous illustre les différentes déclarations nécessaires à l'utilisation de ce type d'évènements.

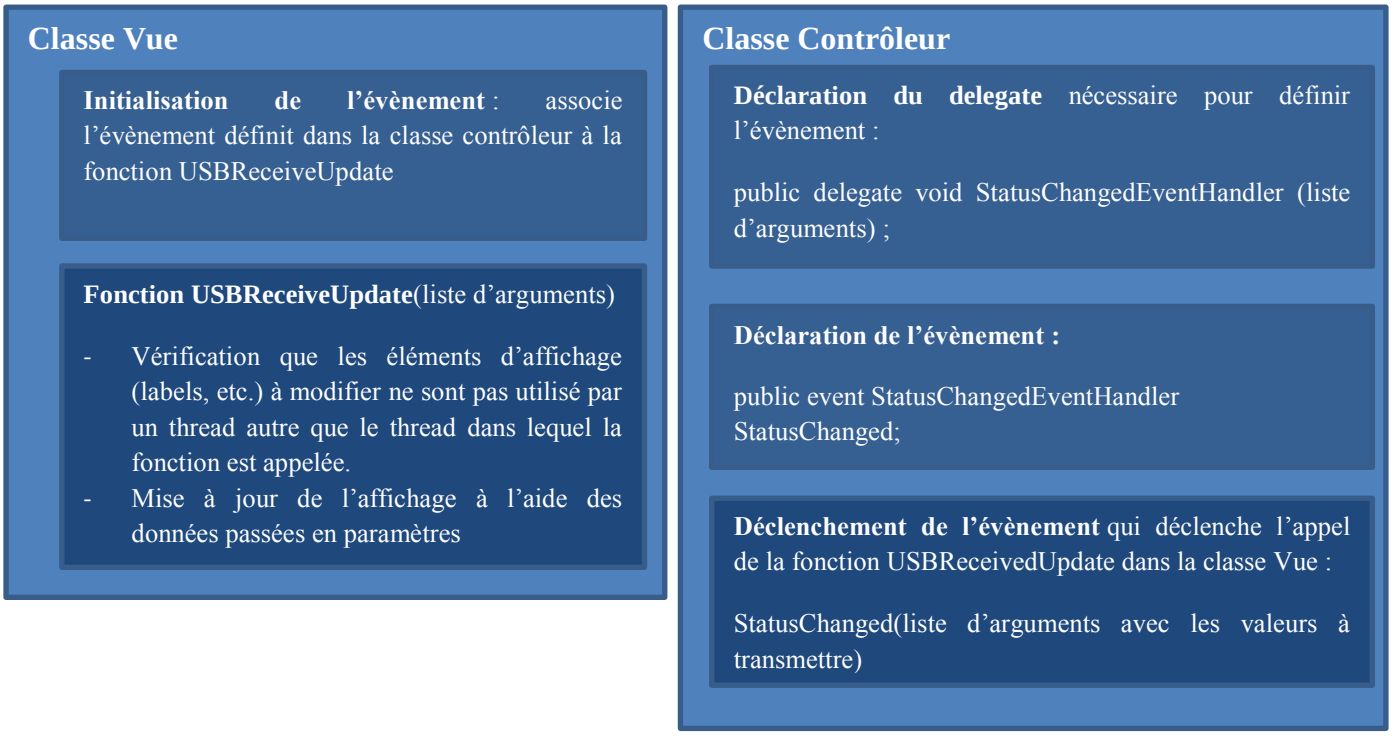

Figure 43 - Mise en place des évènements asynchrones dans le programme

2.3.4.3. *Définition du protocole de communication*

Bien que la carte soit considérée comme un périphérique USB par l'ordinateur, les informations transitent dans les deux sens. Les informations à transmettre sont similaires à celles énoncées pour l'interface graphique Androïd. Ici, la communication est simplifiée étant donné qu'une trame USB est composée de 64 octets. Toutes les informations nécessaires sont ainsi transmises par l'envoi d'une seule et même trame.

Le flux de données étant plus important, l'ajout de l'affichage du combiné dans l'application est possible. C'est pourquoi les informations de vitesse et de régime moteur sont envoyées par la carte au PC.

Des requêtes d'allumage, d'extinction ou de prise en main du contrôle du combiné peuvent être effectuées par l'application PC. De même, le dsPIC peut indiquer au PC que la communication USB doit être interrompue.

Le protocole établi est le suivant :

- Trame USB envoyée par la carte au PC :
  1er octet : vitesse du véhicule.
  2ème et 3ème octets : régime moteur.
  4ème octet : état du combiné (allumé/éteint).

- Trame USB envoyée par le PC à la carte :
  1er octet : état de la communication (requête d'allumage ou d'arrêt du combiné).
  2ème octet : état des voyants (1bit par voyant).
  3ème octet : état des boutons accélérateur et frein.
  4ème octet : numéro du rapport de vitesse enclenché.

2.3.4.4. *Gestion de la communication USB 2.0*

Le périphérique USB (le dsPIC) est détecté par l'ordinateur dès l'allumage de la carte, car l'initialisation de la communication USB est effectuée lors de l'initialisation du microcontrôleur.

2.3.4.4.1. Eléments de l'interface graphique destinés au contrôle de la communication :

Via la gestion des évènements, décrite dans la sous-partie suivante, l'appui sur le bouton « Connecter » va :

- ouvrir la communication et démarrer l'envoi des trames USB, si la communication n'est pas déjà ouverte (il faut donc que l'état actuel de la connexion soit STATE_DISCONNECTED),
- indiquer au PIC, via un des octets de la trame USB, qu'il doit:
  - passer dans le mode de contrôle du combiné par USB,
  - démarrer le combiné si celui-ci ne l'est pas déjà,
  - envoyer les données de vitesse et le régime moteur à l'ordinateur.

L'appui sur le bouton « Déconnecter » permet :

- l'envoi d'une dernière trame USB indiquant au dsPIC le démarrage de la procédure d'arrêt du combiné et la sortie du mode USB,
- l'arrêt d'envoi des trames USB,
- la fermeture de la connexion.

2.3.4.4.2. Ouverture de la communication :

Le driver utilisé pour la communication USB est le LibUSB. Une fois le périphérique reconnu, la communication peut être initialisée et démarrée. Les librairies LibUsbDotNEt sont incluses dans le code pour utiliser les fonctions dédiées à ce driver.

La communication est démarrée (ouverte) grâce à la ligne suivante :

```
MyUsbDevice = UsbDevice.OpenUsbDevice(MyUsbFinder);
```

2.3.4.4.3. Suivi et gestion de la communication active :

Pour suivre l'état de la communication USB, on définit un énumérateur « ConnexionState » dans la classe Model, qui va prendre soit la valeur STATE_CONNECTED, soit STATE_DECONNECTED. L'état de la communication ne passe à STATE_CONNECTED que si la communication USB vient juste d'être établie. De même, l'état de la communication ne passe à STATE_DISCONNECTED que si la communication USB vient juste d'être fermée.

Si la connexion est un succès, l'état de la connexion passe à STATE_CONNECTED alors les threads d'envoi et de réception des trames sont démarrés.

Ces threads sont actifs tant que l'état de la connexion est STATE_CONNECTED, et stoppés si différent.

Tant que la communication est active, une trame USB est envoyée par l'ordinateur toutes les 220ms. De même toutes les 400 ms, la routine de réception de trame est démarrée, pendant laquelle 200ms sont consacrées à l'attente de la réception d'une trame.

2.3.4.4.4. Arrêt de la communication

Il peut être demandé par l'ordinateur ou le microcontrôleur :

- Dans le premier cas, c'est l'appui sur le bouton « Déconnecter » qui va activer la requête en passant le booléen nommé STOPRequest à true. Ce booléen permet de ne pas arrêter directement la communication (les threads restent actifs) et d'envoyer ainsi une dernière trame USB indiquant au dsPIC la volonté d'arrêt de la communication et d'arrêt du combiné. Ce n'est qu'à la suite de cet envoi que l'état du booléen STOPRequest est vérifié et la communication stoppée.
- Dans le second cas, la réception de la requête entraîne automatiquement l'arrêt de la communication, l'état de la communication passe alors à STATE_DISCONNECTED, et les threads sont stoppés.

2.3.4.5. Acquisition et traitements des actions de l'utilisateur

Comme dans l'application Androïd, les actions de l'utilisateur (appuis sur les boutons) sont traitées via les évènements. Quelles que soient les actions, les états de tous les voyants et boutons sont modifiés et enregistrés dans le modèle.

- Evènements de clic pour les boutons de voyants et les boutons de connexion et déconnexion.

  Par exemple, l'appel à la fonction ci-dessous est effectué lors du clic sur le bouton d'allumage des antibrouillards arrières :

```
private void pictureBoxAbarr_Click(object sender, EventArgs e)
{
   this.setStateVoyants(c.AbarrClick());
}
```

  - La fonction « setStateVoyants » prend un tableau de booléen en paramètre, qui contient les états de tous les voyants. Cette fonction permet de n'allumer dans l'interface graphique que les voyants actifs. Les images de fonds des boutons actifs sont alors modifiées.
  - « c » représente un objet de la classe controller. Il faut passer par la fonction AbarrClick de la classe controller pour accéder aux données se trouvant dans la classe model. C'est dans cette classe que sont enregistrés les états des différents voyants. Ainsi, l'appui sur le bouton d'antibrouillard arrière va modifier l'état de ce voyant (false à true ou inversement) et retourner un tableau de booléens contenant les états de tous les voyants.
    Il est nécessaire de renvoyer la totalité des états des boutons car la logique de l'allumage des feux est également gérée. Par exemple, il n'est possible de démarrer les feux de croisement que si les feux de position sont déjà allumés. L'appui sur un bouton peut donc entraîner la modification de l'état d'autres boutons, il faut donc que l'affichage de tous les boutons de voyants soit mis à jour à chaque appui.

- Pour les boutons d'accélérateur et de frein, le fait d'appuyer ou de relâcher le bouton va générer deux évènements distincts : MouseDown et MouseUp.
  Le fait d'appuyer sur l'un des deux boutons passe l'état de ce dernier à true, jusqu'à ce que celui-ci soit relâché.

- L'évènement concernant la sélection du rapport de vitesse se déclenche lorsque la valeur sélectionnée est modifiée.

- Enfin, le clic sur la croix de fermeture de la fenêtre entraîne l'envoi d'une requête d'extinction du combiné et de fermeture de la connexion.

2.4. Intégration de l'ensemble et gestion des modes

2.4.1. Définition des outils

➔ Le combiné est soit allumé, soit éteint. Son état de fonctionnement est enregistré et suivi tout au long du fonctionnement du système. Cette information est contenue dans le fichier source CombiCtrl et peut être récupérée par l'appel à la fonction FUN_GetEtatCombi().

➔ Comme vu précédemment, il y a 3 modes de contrôle du combiné : comodo et pédalier, application Android via le Bluetooth ou encore application en C# sur PC via l'USB.
Pour mettre en place une bonne gestion des modes de fonctionnement, on doit pouvoir identifier facilement chacun d'entre eux. Le type MODE_FUNCTION permet de définir les différents modes possibles :

- COMODO_MODE
- BLUETOOTH_MODE
- USB_MODE

➔ Pour chaque mode de fonctionnement, on définit 4 états de communication.

- COM_*mode*_OFF → Mode de fonctionnement non-utilisé ou combiné éteint.
- COM_*mode*_ASK_FOR_ON → Demande de démarrage du combiné.
- COM_*mode*_ON → Combiné démarré et mode de fonctionnement actif
- COM_*mode*_ASK_FOR_OFF → Demande d'arrêt du combiné.

2.4.2. Gestion des modes

Le mode de commande du tableau de bord par défaut est le comodo + pédales.

Les applications Androïd et PC ont la priorité sur ce mode, ainsi, le démarrage de l'application Androïd ou PC permet de passer respectivement dans les modes de communication Bluetooth ou USB, et d'allumer le combiné si celui-ci est éteint.

La figure n°45 est une représentation du fonctionnement logique de la gestion des modes. COM_BTH_ASK_FOR_ON est activé seulement lorsque l'on appuie sur le bouton de démarrage de l'application Androïd. De même, COM_USB_ASK_FOR_ON est activé seulement lorsque l'on appuie sur le bouton « Connecter » de l'application sur ordinateur.

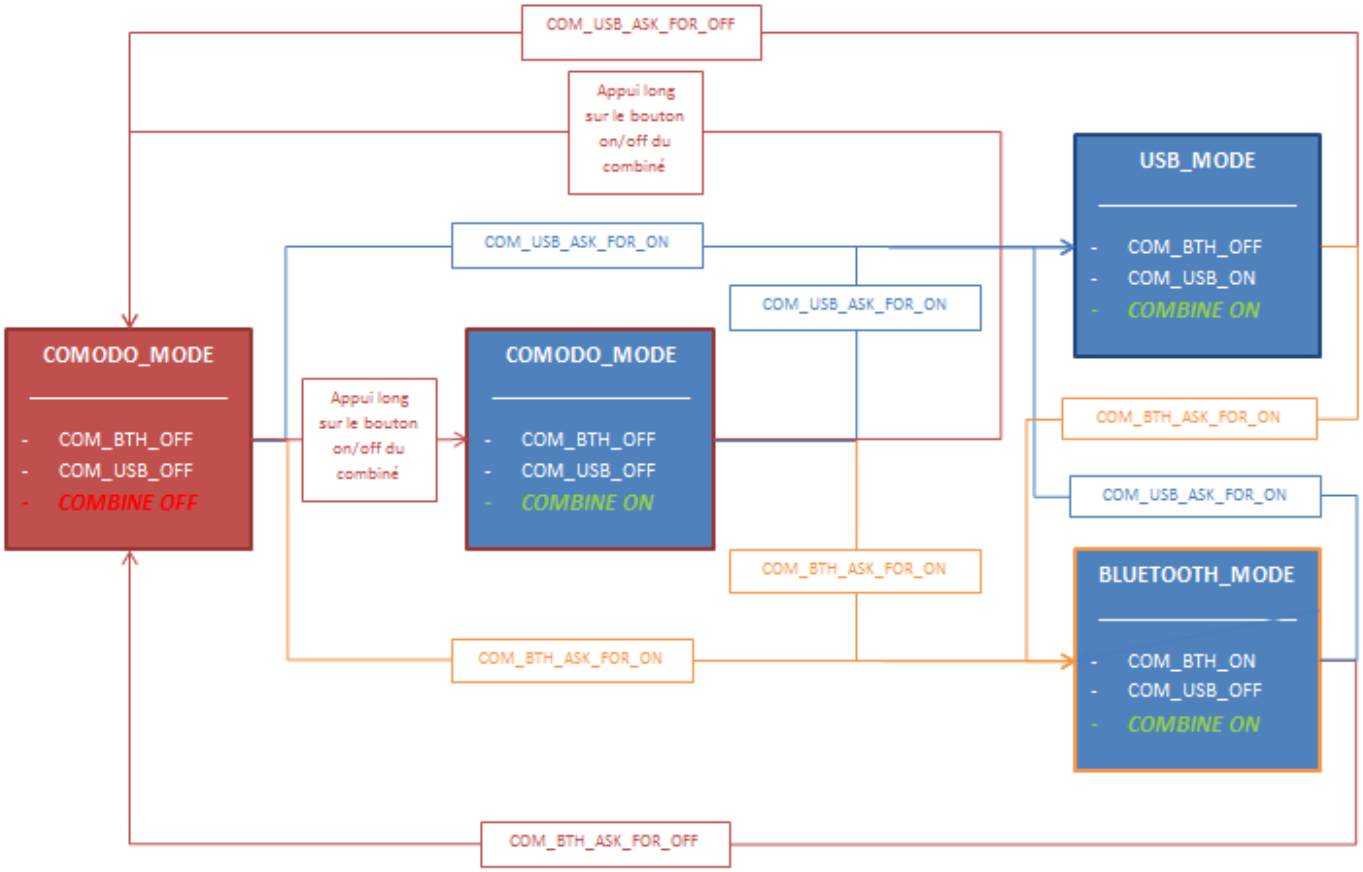

Figure 44 - Gestion des modes de commandes

Au démarrage de l'application, le mode de contrôle par défaut est le comodo et les pédales (encadré rouge).

Du point de vue software, à chaque interruption du timer, la première opération est de scruter les états de communications :

Si les états COM_BTH_ASK_FOR_ON ou COM_USB_ASK_FOR_ON sont actifs, le changement de mode va être opéré.

La fonction FUN_ChangeModeFunction_V(mode) permet d'effectuer les opérations pour sortir du mode actuel proprement et d'enregistrer le nouveau mode demandé. Ensuite, la fonction associée au mode de contrôle du combiné est appelée. Ces fonctions sont décrites dans la sous-partie suivante 2.4.3.

Exemple : passage du mode Bluetooth au mode USB

- COM_USB_ASK_FOR_ON est activé par la réception de la trame de démarrage du mode USB.
- La fonction de changement de mode FUN_ChangeModeFunction_V(USB_MODE) est alors appelée. Le mode actuel est BLUETOOTH_MODE, la fonction FUN_LeaveBluetoothMode_V() est alors appelée. Elle contient les opérations nécessaires à l'arrêt de la communication Bluetooth, à savoir l'envoi de la trame « stop » par Bluetooth au dispositif Androïd. L'état de la communication Bluetooth passe à COM_BTH_OFF.
- Le nouveau mode est enregistré : USB_MODE.
- La fonction FUN_Button_CombiByUSB_V() est appelée. Dans cette fonction vont être réalisées les opérations de contrôle du combiné (voir la sous-partie suivante). L'état de la communication USB passe à COM_USB_ON.

2.4.3. Réalisation des actions de contrôle du combiné selon le mode de contrôle

A chaque interruption du timer 2, soit toutes les 150ms, selon le mode activé, une des fonctions suivantes est appelée :

- Mode Comodo : appel de la fonction FUN_Button_CombiByCOMODO_V()
- Mode Bluetooth : appel de la fonction FUN_Button_CombiByBLUETOOTH_V()
- Mode USB : appel de la fonction FUN_Button_CombiByUSB_V()

Les deux fonctions FUN_Button_CombiByBLUETOOTH_V() et FUN_Button_CombiByUSB_V() ont la structure suivante :

➔ SI (étatDeCommande == COM_*mode*_OFF OU
(étatDeCommande == COM_*mode*_ASK_FOR_OFF ET combiAllumé) OU
(étatDeCommande == COM_*mode*_ASK_FOR_ON ET combiEteint))

ALORS Lancement des routines de démarrage ou d'arrêt du combiné

➔ SINON SI (combiAllumé)

ALORS

- Calcul de la valeur de luminosité du combi et mise à jour des buffers d'envoi associés du module CAN
- Calcul de la vitesse et du régime puis mise à jour des buffers d'envoi associés du module CAN
- Mise à jour des états des voyants et remplissage des buffers d'envoi associés du module CAN

La fonction FUN_Button_CombiByCOMODO_V() a la structure suivante :

➔ SI (appui long sur le bouton on/off du combiné)

ALORS Lancement des routines de démarrage ou d'arrêt du combiné

➔ SINON SI (combiAllumé)

ALORS

- Calcul de la valeur de luminosité du combi et mise à jour des buffers d'envoi associés du module CAN
- Calcul de la vitesse et du régime puis mise à jour des buffers d'envoi associés du module CAN
- Mise à jour des états des voyants et remplissage des buffers d'envoi associés du module CAN

L'envoi des trames CAN est réalisé lors de l'interruption du Timer 1 comme décrit dans la partie « Interfaçage avec le combiné » et est donc indépendant de la gestion des modes.

2.5. Ordre de tests de fonctionnement

Les soudures et les tests des fonctionnalités des composants ont été effectués progressivement.

La première opération a été la mise en place de la conversion de tension, permettant de fournir la tension de 5 et 3,3V au circuit.

- Une fois cela opérationnel, il était nécessaire d'établir la communication avec le microcontrôleur afin de pouvoir le flasher.

- Mise en place des boutons et LED pour un premier essai de contrôle via le microcontrôleur.

- Mise en place des trimmers afin de tester la conversion analogique.

- Soudage du relais et du transistor Darlington pour la commande du relais. Vérification de la bonne commutation du relais.

- Soudage du circuit de communication CAN Low-Speed :
    - TXS0102, test de la bonne adaptation de tension 5V-3.3V.
    - TJA1054, génération d'une trame CAN pour vérifier le bon envoi sur le bus.

- Premier essai de contrôle du combiné.

- Soudage du circuit de la communication CAN High-Speed : test de l'envoi des trames CAN avec l'envoi des trames à la carte Linux embarquée.

- Ne disposant pas d'informations suffisantes concernant le brochage du combiné, il a fallu le trouver en tâtonnant. Si le combiné est alimenté et ne reçoit aucune trame CAN, il se met dans le mode dégradé, c'est-à-dire qu'il affiche seulement quelques informations, on peut donc voir s'il est bien alimenté ou non. Il reste ensuite à trouver le brochage pour le CAN. Dès lors que le combiné envoie sur le bus les bits d'acknowledge en réponse à une trame générée par la carte, c'est que la trame CAN est bien envoyée sur le bus et est correctement reçue par le combiné.

- Soudage du module bluetooth, test de la transmission de données en envoi et en réception.

- Réalisation et débogage de l'application Androïd.

- Soudage du connecteur USB, test de la reconnaissance de périphérique par l'ordinateur.

- Réalisation et débogage de l'application sur PC.

- Intégration de l'ensemble et optimisation de la gestion des modes.

3. Projet complémentaire

A la suite du projet principal de mon PFE, j'ai participé à un projet pour le compte Bosch à Ditzingen. Ce projet a été réalisé dans le cadre du développement du Software de l'Airbag pour Audi. Plus précisément, nous avons travaillé sur le système Pre Sense Pro qui permet de détecter un obstacle et de déterminer si la collision est ou non évitable. Dans le cas où la collision est inévitable, les dispositifs de sécurité (airbag, ceinture de sécurité, etc.) sont enclenchés avant même l'accident. Le projet consiste alors en la réalisation d'une interface qui gère la transmission et la conversion des signaux (en réception pour les signaux venant des capteurs ou à en transmission pour les signaux de commandes) entre les bus de communications (CAN et Flexray) et le module PSP (Pre Sense Pro) qui traite les données reçues des capteurs et commande les différents dispositifs de sécurité.

3.1. Structure et fonctionnement du système

L'architecture du système est la suivante :

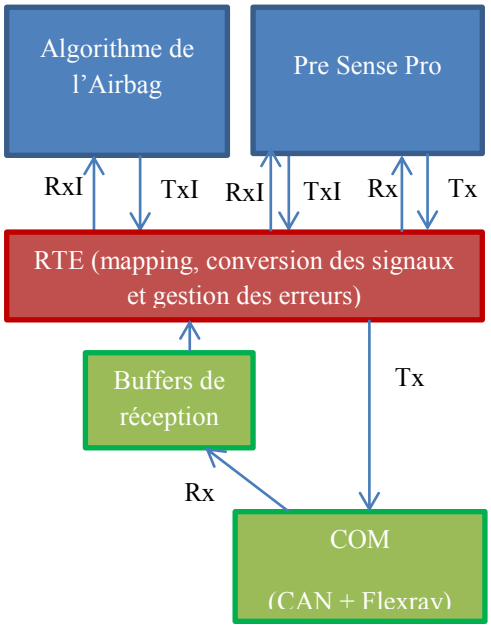

Figure 45 - Architecture du Software

La partie du travail consiste à réaliser la partie RTE (Real Time Environment) ainsi que le buffer de réception. Dans cette brique, le mapping et la conversion des signaux sont effectués. Audi fournit les librairies Rte.h et .c, qui contiennent les fonctions permettant de lire les différents signaux à partir du PSP, et de les écrire sur le bus. Une fonction permettant de lire une valeur reçue est de la forme :

```
Std_Return   Rte_Read_<InterfaceNameInRte>_<SignalNameInRte>( <StructOfInterface> * <NameOfStruct>)
{
   <NameOfStruct>➔<SignalNameInRte> = <Scale> * <StructOfPdu>.<SignalNameInPdu> + <Offset>;
   …
}
```

Prenons un exemple précis. Voici la fonction permettant de récupérer les signaux externes du message ACC10 et de les enregistrer dans l'API (interface) ActionsAcc du PSP.

```
Std_ReturnType Rte_Read_ActionsAcc_ActionsAcc(tActionsAcc * ActionsAcc)
{
  Std_ReturnType Rte_Status = RTE_E_OK;

  /* Check if signal qualifier is valid */
  if(S_PspRxACC10_XSR.E_PDUValid_XXX)
  {
    ActionsAcc->AccCollisionMitigationRequest = (S_PspRxACC10_XSR.E_CMAnforderung_XXX);
    ActionsAcc->AccBrakePartialModeRequest    = (S_PspRxACC10_XSR.E_InfoTeilbremsung_XXX);
    ActionsAcc->AccBrakePartialRequestEnabled = (S_PspRxACC10_XSR.E_TeilbremsungFreigabe_XXX);
    ActionsAcc->AccBrakeFullRequestEnabled    = (S_PspRxACC10_XSR.E_ZielbremsungFreigabe_XXX);
    ActionsAcc->AccDecelerationRequested      = (((tFloat32)(S_PspRxACC10_XSR.V_ZielbrmTeilbrmVerzAnf_U16X) *
C_Scale_RxACC10_ANBZielbremsTeilbremsVerzAnf) + C_Offset_RxACC10_ANBZielbremsTeilbremsVerzAnf);
  }
  else /* Signal qualifier is invalid, send init values */
  {
    ActionsAcc->AccCollisionMitigationRequest = (C_DefaultValuePhysic_RxACC10_ANBCMAnforderung);
    ActionsAcc->AccBrakePartialRequestEnabled = (C_DefaultValuePhysic_RxACC10_ANBTeilbremsungFreigabe);
    ActionsAcc->AccBrakeFullRequestEnabled    = (C_DefaultValuePhysic_RxACC10_ANBZielbremsungFreigabe);
    ActionsAcc->AccDecelerationRequested      = (C_DefaultValuePhysic_RxACC10_ANBZielbremsTeilbremsVerzAnf);
  }

  return Rte_Status;
}
```

Audi fournit également sous forme de tableau Excel la liste des messages CAN et Flexray ; chacun des messages contient une liste de signaux. Pour chacun des signaux, il est indiqué si une conversion est nécessaire avant transmission au PSP. Ce fichier est appelé K-Matrix, il s'agit du même type de fichier que nous avons utilisé pour le projet CombiCAN pour définir la composition des messages CAN.

Dans le fichier Rte.h sont définis des structures. Chacune de ces structures est une interface et a un nom bien défini. Chaque interface représente une liste de signaux. Un autre fichier Excel fournit les noms des signaux utilisés et l'interface qui leur est attachée. Les noms des signaux ne sont pas les même dans les structures du fichier header fournit par Audi que dans les K-Matrix, c'est pourquoi il a fallu créer un tableau qui recense tous les noms de signaux définis dans les structures du fichier Rte.h et qui les associe aux signaux définis dans la K-Matrix. Si une conversion est requise pour le signal, alors elle est aussi indiquée. Ce fichier permet de générer le code C qui va notamment compléter les fonctions définies dans le fichier Rte.c et de définir entre autres des structures de signaux pour les regrouper par message.

Ci-dessous, le format d'une structure de signaux :

```
typedef struct {
  U16 element1;
  U16 element2;
  U8 element 3;
  ...
  te_Boolean E_PDUValid_XXX;
} ts_PspRx<PduName>

ts_PspRx<PduName> S_PspRx<PduName>_XSR;
```

L'ensemble de ces structures va constituer le buffer de réception. Ce buffer a pour fonction de contenir les valeurs des signaux reçus. En passant par ce buffer, il n'est pas nécessaire de réaliser la conversion d'un signal à chaque réception de celui-ci. La conversion ne s'effectue alors que lorsque le signal est lu par le PSP (ce qui constitue une économie de temps d'exécution car la fréquence de lecture des signaux est plus faible que la fréquence de leur réception).

3.2. Tests unitaires

J'avais pour tâche de mettre en place la partie de tests unitaires du software. Il s'agit de tests unitaires RTRT (Rational Test Real Time) réalisés à l'aide du logiciel IBM Rational Test Real Time. Chaque test va permettre de vérifier que l'on obtient bien les valeurs attendues si l'on met des paramètres bien définis en entrée d'une fonction.

La planification des tests passe par le remplissage d'un tableau, dans lequel chaque test est littéralement décrit. A partir de ce tableau est généré le code nécessaire à la mise en œuvre des tests. J'ai jusqu'à présent défini la plupart des tests à effectuer : il y a 4 cas différents, selon que la fonction à tester est une fonction d'écriture « Write » ou de lecture « Read » des signaux présents sur les bus CAN ou Flexray.

Dans le cas d'une fonction Read, les champs de la structure passée en paramètre sont remplis avec les valeurs converties des signaux d'entrée. Les tests vont consister à vérifier que tous les champs de la structure sont remplis correctement, cela pour différentes valeurs des signaux d'entrée : valeur minimum, maximum, de défaut (valeur prise par le signal lors de l'initialisation par exemple), comprise entre le minimum et le maximum. Il faut également tester une valeur pour le signal d'entrée de telle manière que la valeur convertie soit positive/négative.

Dans le cas de fonction Write, ce sont les mêmes valeurs qui sont testées mais ici les valeurs à vérifier sont celles prises par les signaux sur le bus pour les valeurs données affectées au champ de la structure passée en paramètre.

Un test RTRT se présente sous la forme suivante. Ce test va permettre de tester le bon fonctionnement de la fonction prise en exemple dans la sous-partie précédente.

```
#tActionsAcc variable;

  TEST Rte_Read_ActionsAcc_ActionsAcc_01
  FAMILY nominal

      ELEMENT
          VAR S_PspRxACC10_XSR.E_PDUValid_XXX , init = 1, ev=init
              --output data
                  VAR variable.AccCollisionMitigationRequest, init = 0, ev = 0
                  VAR variable.AccBrakePartialModeRequest, init = 0, ev = 0
                  VAR variable.AccBrakePartialRequestEnabled, init = 0, ev = 0
                  VAR variable.AccDecelerationRequested, init = 0, ev = -20.016001
                  VAR variable.AccBrakeFullRequestEnabled, init = 0, ev = 0
              --input data
                  VAR S_PspRxACC10_XSR.E_CMAnforderung_XXX, init = 0, ev = init
                  VAR S_PspRxACC10_XSR.E_InfoTeilbremsung_XXX, init = 0, ev = init
                  VAR S_PspRxACC10_XSR.E_TeilbremsungFreigabe_XXX, init = 0, ev = init
                  VAR S_PspRxACC10_XSR.V_ZielbrmTeilbrmVerzAnf_U16X, init = 0, ev = init
                  VAR S_PspRxACC10_XSR.E_ZielbremsungFreigabe_XXX, init = 0, ev = init

          #Rte_Read_ActionsAcc_ActionsAcc(&variable);
      END ELEMENT

  END TEST -- Rte_Read_ActionsAcc_ActionsAcc_01
```

Pour chacune des variables présentes dans la fonction, on définit ici la valeur initiale « init » et souhaitée « ev » (expected value) en fin d'exécution de la fonction.

Toute expression précédée d'un « # » signifie l'insertion de code C. Ainsi les valeurs initiales sont passées aux variables avant d'exécuter la fonction à tester.

En annexe n°2, on retrouve un extrait du rapport de test obtenu. Ce document est généré par le logiciel et présente de manière claire les valeurs prises par les différentes variables, ainsi que le pourcentage de code testé. Il va de soi que la totalité du code doit être testée, pour différentes valeurs et que ces tests doivent être bons pour que cette étape de tests unitaires soit validée.

3.3. Livraison du projet

Une fois cette étape effectuée, notre partie du projet était terminée et nous avons pu livrer cette partie programme sous le logiciel de gestion de projet « MKS Source Integrity ». Ce logiciel donne accès à une base de données et permet de stocker toutes les versions d'un projet, c'est-à-dire toutes les différentes BaseLines et Bug Fixes d'un projet. Toutes les étapes du développement d'un projet sont alors consultables, mais non modifiables. En effet, après enregistrement d'une version d'un projet dans MKS (opération de Check-In), il n'est plus possible de modifier cette version, elle est simplement consultable et téléchargeable (Check-Out). Différents logiciels de ce type sont utilisés chez Bosch, parmi eux, on trouve eASEE ou TCM.

Cette opération de « Check-In » dans le logiciel MKS marque ainsi la livraison du projet au client.

Conclusion

Le but de ce projet de fin d'études effectué chez T&S était de réaliser une maquette de démonstration « Combi Instrument CAN » permettant de piloter un tableau de bord par le CAN, en mettant en œuvre différentes interfaces de commande. Le cahier des charges fixé en début de projet a été respecté. Il s'agissait d'un projet de recherche et développement, non destiné à la commercialisation, mais qui avait pour objectif de matérialiser et capitaliser la connaissance de différents protocoles de communication, notamment du CAN qui est très utilisé dans le domaine de l'électronique embarquée.

Le développement de ce projet m'a ainsi permis d'acquérir et de renforcer mes connaissances aussi bien dans le domaine Hardware que Software, étant donné que le projet contenait une phase de réalisation de cartes électroniques, puis une phase de programmation de microcontrôleur et d'applications. La carte électronique de la maquette a l'avantage de disposer de nombreuses fonctionnalités, dont 2 réseaux CAN de vitesses différentes. Une suite possible à ce travail pourrait donc être d'ajouter d'autres nœuds au réseau CAN afin d'échanger avec d'autres systèmes et pourquoi pas de les contrôler. On voit bien que les possibilités d'utilisation sont quasi-illimitées.

Le projet complémentaire de développement et tests d'une interface de signaux réalisé chez Bosch a été l'occasion non seulement de mettre en œuvre dans le cadre d'un projet industriel les connaissances acquises en début de stage, mais aussi de prendre en main les différents outils utilisés chez Bosch et dans l'industrie électronique en générale. J'ai aussi pu découvrir les processus qui sont utilisés là-bas et distinguer concrètement les différentes phases du cycle en V, de la rédaction des spécifications à la livraison du projet.

Bibliographie

[1] Modèle OSI ; An, http://fr.wikipedia.org/wiki/Modèle_OSI.

[2] « Le Bus CAN » ; DIOU Camille, Maitrise EEA 2001-2002, 66p., (2002).

[3] dsPIC33EPXXXMU806/810/814 and PIC24EPXXXGU810/814 Data Sheet ; An, Microchip Technology Inc., DS70616E, 570 p., (2009-2011).

[4] Section 22. Direct Memory Access (DMA) ; An, Microchip Technology Inc., DS70348B, 60 p., 2011.

[5] Section 21. Enhanced Controller Area Network (ECAN™) ; An, Microchip Technology Inc., DS70353C, 76 p., (2008-2011).

[6] Section 16. Analog-to-Digital Converter (ADC) ; An, Microchip Technology Inc., DS70621B, 92 p., 2011.

[7] Peugeot 207 RC : 1.6 l 16v THP, 4 cylindres EP6DTS : 128 kW (175 ch) – 240 Nm ; An, http://www.feline207.net/photos/modules/myalbum/photo.php?lid=3661, 2007.

[8] Fiche technique - PEUGEOT - 207 RC ; An, http://www.rsiauto.fr/peugeot/207-rc-542.php.

[9] Section 25. USB On-The-Go (OTG) ; An, Microchip Technology Inc., DS70571B, 64 p., (2010).

[10] Universal Serial Bus Specification ; Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC & Philips, Revision 2.0, 650p., (2000).

[11] L'USB en bref ; Bernard Acquier, http://acquier.developpez.com/cours/USB/, 2005.

[12] CAN Specification 2.0 Part B ; An, http://www.can-cia.org/fileadmin/cia/specifications/CAN20B.pdf