Projet SPI: Le Bus CAN

FIPA 18 S3E Promotion 1
Tuteur Projet : Thierry Perisse
BAUDE | BOUYSSOU | DAZA | FEYSSEL | GAMBLIN

Cette page a été laissée blanche intentionnellement

Remerciements

Nous tenons à remercier les personnes et les organisations suivantes :

- Thierry Périsse pour nous avoir fourni de précieux conseils, un support matériel pour effectuer notre réalisation et son soutien technique.
- Stewart Allwood, directeur de la société Galloise Artic Consultants pour nous avoir fourni un logiciel de calcul nous ayant grandement aidé pour comprendre le fonctionnement du CAN dans les microcontrôleurs PIC.
- L'équipe pédagogique de l'école d'ingénieur du CESI de Labège et plus particulièrement notre pilote de formation Mme Céline Viazzi pour l'organisation et la répartition des groupes, la définition des sujets, et pour son apport pédagogique concernant la recherche documentaire.
- Nicolas Herbere, apprenti CESI spécialité Système Electriques et Electroniques embarqués en troisième année pour son aide précieuse sur la partie électronique et PIC de notre réalisation.

Table des matières

Table des illustrations

Indexation des abbreviations

ARINC	Aeronautical Radio, Incorporated
CAN	Controller Area Network
CAN FD	Controller Area Network Flexible Data rate
CAN H	Controller Area Network High
CAN L	Controller Area Network Low
CiA	CAN in Automation
CRC	Cyclic Redundancy Code
DLC	Data Length Code
DLL	Data link layer
ECU	Electronic Control Unit
EOF	End Of Frame
iCC	International CAN Conference
IEC	Commission Électrotechnique Internationale
ISO	Organisme International de Normalisation
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
LLC	Logical Link Control
MAC	Medium Access Control
MAU	Medium Access Unit
MDI	Medium Dependent Interface
NI	National Instrument
NRZ	Non return to Zero
OSEK	Systèmes ouverts et interfaces correspondantes pour l'électronique des véhicules automobiles
OSI	Open Systems Interconnection
PIC	Programmable Intelligent Computer
PLS	Physical Signalling
PMA	Physical Medium Attachment
RBS	Resource Breakdown Structure
REC	Receive Error Counter
RTR	Remote Transmission Request
SAE	Society of Automotive Engineers
SOF	Start Of Frame
SPI	Sciences Physiques pour l'Ingénieur
TEC	Transmit Error Counter
TRMC	Taux de Réjection du Mode Commun
TTCAN	Time-triggered communication protocol for CAN
UCE	Unités de Contrôle Électronique
VDX	Vehicle Distributed eXecutive
VI	Instruments Virtuels
WBS	Work Breakdown Structure

Introduction

A travers nos réalisations en entreprise et de nos études d'ingénieur en Systèmes Electriques et Electroniques Embarqués par l'apprentissage, nous sommes amenés à prendre part à un projet de Sciences Physiques pour l'Ingénieur. L'objectif est de mettre en œuvre nos compétences acquises durant notre formation par l'étude d'une technologie. Ce projet se traduit par l'élaboration de ce mémoire, d'une réalisation technique et d'une soutenance orale.

Dans le cadre de notre formation et de nos projets en entreprise, nous sommes amenés à utiliser des protocoles de communication entre calculateurs ou entre ordinateurs. Or, nous avons, pour la quasi-totalité des personnes de notre groupe, mis en œuvre, ou simplement étudié, le bus/réseau CAN. Cela nous a conduits à nous demander : Pourquoi le bus/réseau CAN s'est-t-il imposé dans le domaine automobile et pourquoi se démocratise-t-il de plus en plus dans le domaine aéronautique ?

La pertinence de cette problématique s'est d'ailleurs confirmée au cours des travaux préparatoires de la présente étude : le protocole CAN étant très utilisé dans des environnements sévères comme ceux d'une automobile ou d'un avion, il nous parait intéressant de nous poser ces questions : Comment s'est développé le bus/réseau CAN ? Quel est son principe de fonctionnement ? Quelles sont ses caractéristiques garantissant sa qualité de transmission ? Comment le mettre en œuvre ?

Afin de répondre au niveau de détail et de technicité demandé par l'exercice du projet SPI, nous nous baserons sur l'abondante bibliographie consacrée à la matière, notamment la norme CAN de Bosch. Aussi, il sera nécessaire de se fonder sur nos expériences en entreprise et sur le retour d'expérience apporté par la réalisation technique que nous allons vous présenter. L'exploitation de ces sources permettra de répondre à la série d'interrogations inhérentes à notre sujet.

Nous allons démontrer que les propriétés de ce bus permettent une grande fiabilité dans la transmission et un faible coût de mise en œuvre. Pour ces raisons, le Bus CAN s'est imposé comme le réseau de terrain le plus utilisé dans l'automobile.

Dans une première partie, nous définirons des termes techniques, un bref historique, les normes applicables et les domaines d'application. Ensuite, nous procéderons à une analyse technique approfondie du réseau CAN à travers l'étude de toutes les caractéristiques essentielles le définissant. Enfin, nous verrons dans une troisième partie la mise en œuvre du Bus CAN dans une réalisation technique concrète.

I- Présentation générale du Bus CAN

1. Définitions

Afin de bien comprendre le Bus CAN, il est nécessaire de définir les différents termes techniques abordés dans le dossier.

Bus de communication : Un bus de communication est un dispositif non bouclé reliant plusieurs composants, sous-ensembles ou matériels pour permettre entre eux l'apport d'énergie et la circulation d'informations.

Bus CAN : Le protocole de couche liaison CAN est le protocole dominant pour les systèmes de contrôle-commande dans les véhicules.

Il comporte :

- Une capacité à travailler avec plusieurs maîtres,
- Une communication par diffusion,
- Des fonctions sophistiquées de détection d'erreurs.

Réseau : Moyens de télécommunications entre les équipements informatiques. Le réseau peut être :

- En étoile : les équipements du réseau sont reliés à un système matériel central qui a pour rôle, d'assurer la communication entre les différents équipements du réseau,
- Maillé : chaque équipement du réseau doit recevoir, envoyer et relayer des données,
- Bus : tous les équipements sont reliés à une même ligne de transmission,
- Anneau : les ordinateurs sont situés sur une boucle et communiquent chacun à leur tour.

Trame : Une trame est un ensemble structuré d'éléments numériques consécutifs, spécifié par un protocole de communication.

Architecture de communication : Structure d'éléments définissant la communication :

- Les entités communicantes,

- Les règles d'échange entre les entités communicantes.

Protocole de communication : Un protocole est une méthode standard qui permet la communication entre des processus (s'exécutant éventuellement sur différentes machines), c'est-à-dire un ensemble de règles et de procédures à respecter pour émettre et recevoir des données sur un réseau.

Nœud : Un nœud représente un objet relié à un réseau.

Multiplexage : Action d'assembler des signaux indépendants en un seul signal composite à partir duquel ils peuvent être restitués. Il existe différents types de multiplexage : multiplexage en fréquence, dans le temps, en code, en longueur d'onde, etc.

Couche OSEK :

OSEK est le sigle pour « *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug* », en français Systèmes ouverts et interfaces correspondantes pour l'électronique des véhicules automobiles.

OSEK a été créé en 1993 par un consortium de constructeurs et équipementiers automobiles allemands (BMW, Bosch, Daimler, Chrysler, Opel, Siemens et Volkswagen) ainsi qu'un département de l'université de Karlsruhe. Leur but était de développer un standard pour une architecture ouverte reliant les divers contrôleurs électroniques d'un véhicule. En 1994, les constructeurs français Renault et PSA qui développaient un projet similaire, VDX (Vehicle Distributed eXecutive), rejoignirent le consortium.

L'architecture ouverte présentée par OSEK/VDX comprend trois parties :

- La communication (échange de données entre unités de contrôle),
- La gestion de réseau,
- Le système d'exploitation temps réel.

2. Historique

C'est en février 1986 que la société Bosch introduit le réseau CAN (Controller Area Network) pendant le congrès de la SAE (Society of Automotive Engineers). Et c'est en 1992 que le premier véhicule équipé du nouveau protocole est commercialisé.

Aujourd'hui, presque chaque nouvelle voiture de tourisme fabriquée en Europe est équipée d'au minimum un réseau CAN. Également utilisé dans d'autres types de véhicules de transport (trains, navires, avions..) ainsi que dans les secteurs industriels, le CAN s'est imposé en tant que premier protocole réseau.

1983:

- Lancement du projet interne Bosch en partenariat avec l'université Karlsruhe pour développer un réseau dans le véhicule.

1986:

- Bosch présente officiellement et pour la première fois le protocole CAN pendant le congrès de la SAE (Society of Automotive Engineers).

1987:

- Apparition des premiers circuits imprimés liés au CAN par Intel et Philips Semiconductors.

1991:

- Publication par Bosch des spécifications de la version 2.0.
- Kvaser introduit le CAN Kingdom (protocole de haut niveau basé sur CAN) utilisé par les fabricants de machines textiles.

1992:

- Création du groupe CiA (CAN in Automation) établi par des fabricants et utilisateurs du CAN.
- Publication de la première couche applicative du CAN (CAL) par la CiA.
- Premiers véhicules équipés du réseau CAN : Mercedes Classe S.

1993:

- Publication de la norme ISO 11898.
- Création du groupe OSEK (Systèmes ouverts et interfaces correspondantes pour l'électronique des véhicules automobiles).

1994:

- CiA organise la première conférence internationale sur le CAN (iCC).
- Allen-Bradley introduit le protocole DeviceNet utilisant la technologie du CAN.
- PSA (Peugeot – Citroen) et Renault rejoignent le groupe OSEK.

1995:

- Publication d'un amendement pour la norme ISO 11898 (format de trame étendu).
- CiA publie le protocole CANopen.

2000:

- Développement du protocole TTCAN (time-triggered communication protocol for CAN).

2012:

- Bosch présente officiellement l'évolution du Bus CAN : le CAN-FD lors de la 13^{e} conférence internationale (iCC).

3. Principe de fonctionnement

Le Bus CAN est un bus de communication qui permet d'effectuer des échanges de données entre plusieurs nœuds ou ECU (Electronic Control Unit). Chaque nœud ou ECU peut communiquer avec les autres. La transmission des données s'effectue sur une paire filaire par émission différentielle : une mesure de la différence de tension entre les deux lignes (CAN H et CAN L) est réalisée. La ligne du bus doit se terminer par des résistances de 120 ohms à chacun des bouts. Les communications sont réalisées avec des paquets de messages et la vitesse de transmission peut atteindre jusqu'à 1Mb/s (Pour le Can High Speed).

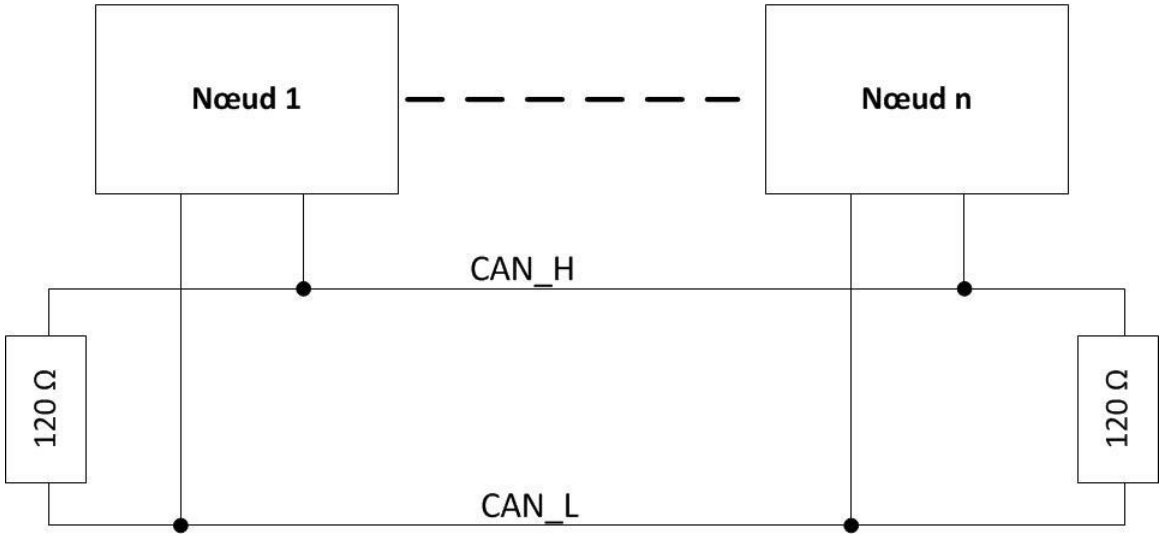


Figure 1 : Fonctionnement du Bus CAN

4. Normes et spécifications

Il existe deux types d'organismes de normalisation, nationaux et internationaux.

- Les internationaux : L'Organisme International de Normalisation (ISO) est responsable de toutes les applications non-électriques et la Commission Électrotechnique Internationale (IEC) responsable de tous les équipements électriques et électroniques.
- Les nationaux : Cenelec est l'homologue Européen de IEC et le Comité Européen de Normalisation équivalent à l'ISO

Le protocole CAN a été décrit pour la première fois dans une spécification publiée par Bosch.

4.1 Les normes ISO

En 1993 l'Organisme International de Normalisation (ISO), publie la norme ISO 11898 sur le protocole CAN dans les voitures, substituant tous les prédécesseurs, incluant la spécification de Bosch.

La norme ISO 11898 se décompose en plusieurs parties, qui sont les suivantes :

- **L'ISO 11898-1:2003** spécifie la couche liaison de données (DLL) et la signalisation physique du gestionnaire de réseau de communication (CAN): un protocole de communication série qui prend en charge la commande répartie en temps réel et le multiplexage, pour les besoins des véhicules routiers. Elle décrit l'architecture générale du CAN, en termes de couches hiérarchiques, conformément au modèle de référence ISO pour l'interconnexion de systèmes ouverts (OSI) spécifié dans l'ISO/CEI 7498-1, et fournit les caractéristiques de configuration d'un échange d'informations numériques entre modules par mise en œuvre de la DLL du CAN, celle-ci étant spécifiée conformément à l'ISO/CEI 8802-2 et à l'ISO/CEI 8802-3, avec des spécifications détaillées de la sous-couche de contrôle de liaison logique (LLC) et de la sous-couche de contrôle d'accès au support (MAC).

- **L'ISO 11898-2:2003** spécifie l'unité d'accès au support (MAU) à haute vitesse (vitesses de transmission atteignant 1 Mbit/s) et certaines caractéristiques de l'interface dépendant du support (MDI) (conformément à l'ISO/CEI 8802-3) de la couche physique du gestionnaire de réseau de communication (CAN): un protocole de communication série qui prend en charge la commande répartie en temps réel et le multiplexage, pour les besoins des véhicules routiers.

- **L'ISO 11898-3** spécifie les caractéristiques d'établissement d'un échange d'informations numériques entre des unités de contrôle électroniques de véhicules routiers équipés du gestionnaire de réseau de communication (CAN, de l'anglais «Controller Area Network») à des débits de transmission supérieurs à 40 kbit/s et pouvant atteindre 125 kbit/s.

- **L'ISO 11898-4:2004** spécifie le déclenchement temporel des communications des gestionnaires de réseau de communication (CAN): un protocole de communication série qui prend en charge la commande répartie en temps réel et le multiplexage, pour le besoin des véhicules routiers. Elle est applicable à la mise en place d'un échange d'informations

numériques, avec déclenchement temporel, entre les unités de contrôle électronique (UCE) de véhicules routiers équipés d'un CAN, et spécifie l'entité de synchronisation des trames qui coordonne le fonctionnement du contrôle de liaison logique et du contrôle de l'accès au support, conformément à l'ISO 11898-1, pour produire un ordonnancement des communications par déclenchement temporel.

- **ISO 11898-5:2007** : Véhicules routiers, Gestionnaire de réseau de communication (CAN), Partie 5: Unité d'accès au médium haute vitesse avec mode de puissance réduite

- **ISO 11898-6:2013** Véhicules routiers, Gestionnaire de réseau de communication CAN, Partie 6: Unité d'accès au médium haute vitesse avec fonctionnalité de réveil sélectif

L'organisme ISO a publié d'autres normes sur diverses applications :

- La norme ISO 11992 basée sur les caravanes,
- La norme ISO 15765-2 qui standardise le protocole de transport,
- La norme ISO 15765-4 sur les diagnostics,
- La norme ISO 16844 sur le tachygraphe dans les véhicules utilitaires,
- La norme ISO 11783 sur la communication entre les tracteurs et les équipements pour l'agriculture,
- La norme ISO 13628-6 décrit les exigences générales pour les équipements sous-marins qui utilisent le réseau CAN pour lier des capteurs et des mètres à l'unité de contrôle.

4.2 Les normes IEC

On trouve aussi des normes publiées par l'organisme IEC sur le CAN :
- IEC 61375-3-3 : décrit la mise en œuvre de la couche applicative CANopen spécifique au réseau CAN dans les véhicules ferroviaires, les locomotives que dans les autocars,
- IEC 61800-7-201/301 : spécifique pour le CIA 402 CANopen sur le dispositif qui établit le profil du contrôle des mouvements et des énergies.

4.3 Divers organismes

Dans les autres organismes qui publient des normes sur le CAN il y a :
- les deux organismes Européens, aujourd'hui, ces deux organismes travaillent très étroitement avec IEC et ISO, afin d'éviter de se retrouver avec des normes en doubles,
- Le Comité Génie Électronique des compagnies Aériennes : qui a commencé le développement de la spécification de l'ARINC (Aeronautical Radio, Incorporated) 825 qui est une normalisation générale pour l'utilisation du CAN dans les avions.

5. Domaines d'application

D'abord créé par Bosch pour l'industrie automobile, avec une application de mise en réseau électronique dans l'automobile, il s'est démocratisé au cours des 20 dernières années.

Son domaine d'application s'étend des réseaux à haut débit aux réseaux de multiplexage faible coût. De nombreuses industries ont adopté le Bus CAN pour une grande variété d'application telles que :

- Applications ferroviaires (tramways, métros, TGV…) relayant les commandes de freinage ou de contrôle des portes,
- Applications aéronautiques, capteurs de vol, système de navigation, commande des moteurs, pompes et actionneurs,
- Applications aérospatiales,
- Applications médicales pour les équipements médicaux ou la gestion des hôpitaux (gestion des salles d'opération, équipement des chambres),
- Ascenseurs et escaliers mécaniques,
- Applications non industrielles telles que les équipements de laboratoire, appareils de sport, des télescopes, des portes automatiques, et même des machines à café.

6. Intérêts du Bus CAN

Le principal intérêt du Bus CAN est la réduction des coûts. On peut classer ces intérêts en 3 catégories :

Réduction des coûts initiaux:

- Un seul câble nécessaire pour tous les équipements au lieu d'en utiliser un par équipement,
- Réutilisation du câblage analogique existant dans certains cas,
- Réduction du temps d'installation,
- Réduction du matériel pour l'installation du système.

Réduction des coûts de maintenance:

- Moins de maintenance due à la fiabilité accrue du système,
- Maintenance plus aisée: réduction du temps de dépannage, localisation des pannes possibles avec des diagnostics à distance,
- Flexibilité de l'extension à un nouveau bus de terrain,
- Simplifications des raccordements.

Performances globales accrues :

- Amélioration de la précision : Il n'y a pas d'erreurs de distorsion ni de réflexions dû à la précision du signal, ce qui n'est pas le cas sur un signal analogique
- Les données et mesures sont généralement disponibles à tous les équipements de terrain,
- Communications possibles entre 2 équipements sans passer par le système de supervision.

Dans l'exemple qui suit, on peut constater la réduction des liaisons afin de faire communiquer plusieurs unités au sein d'une automobile. Les unités communiquent ensemble avec un bus unique.

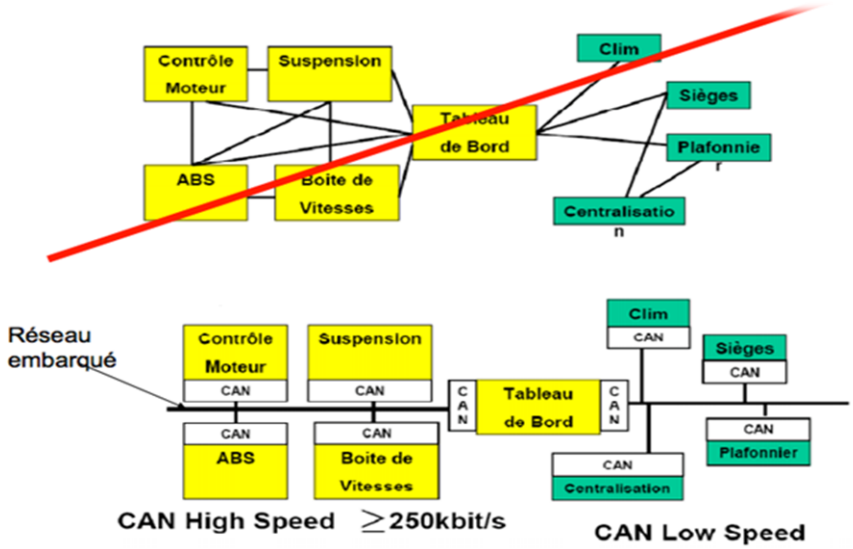


Figure 2 : Intérêts du Bus CAN dans l'automobile

7. Situation par rapport au modèle OSI

Le Bus CAN étant un protocole réseau, il doit répondre à la norme du modèle OSI qui définit en 7 couches les fonctionnalités nécessaires à la communication et l'organisation d'un protocole en réseau. Les couches (et sous-couches) du modèle OSI liées au CAN sont :

- La couche application (7)
- La couche liaison de données (2)
  - Sous-couche MAC (Medium Access Control)
  - Sous-couche LLC (Logical Link Control)
- La couche physique (1)
  - Sous-couche PLS (Physical Signalling)
  - Sous-couche PMA (Physical Medium Attachment)
  - Sous-couche MDI (Medium Dependant Interface)

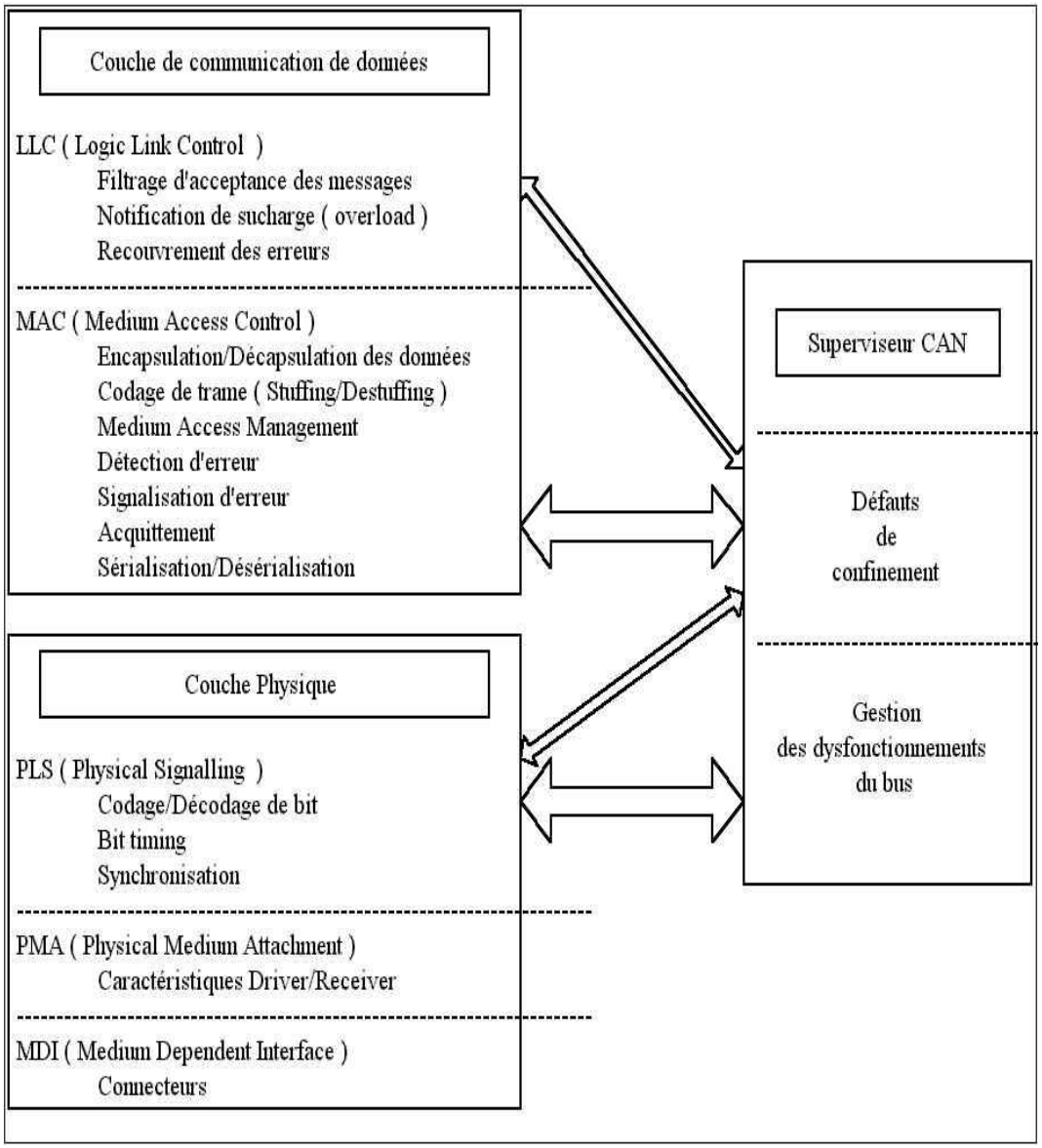


Figure 3 : Application du CAN au modèle OSI

7.1 Couche applicative

Cette couche est vide, mais sera complétée par l'utilisateur en fonction de l'application à laquelle sera affecté le Bus CAN.

7.2 Couche liaison de données

Cette couche va fournir les moyens nécessaires à l'établissement, la libération et au maintien des connexions entre les différentes entités du bus. Elle est aussi en charge de corriger les erreurs du premier niveau.

La sous-couche MAC est en charge de :

- La mise en trame du message,
- L'arbitrage,
- L'acquittement,
- La détection des erreurs,
- La signalisation des erreurs.

La sous-couche LLC est en charge de :

- Filtrer les messages,
- Notifier les surcharges,
- Procéder au recouvrement des erreurs.

7.3 Couche physique

La couche physique va définir la façon dont le signal est transmis. Elle assure le transfert physique des bits entre chaque nœud, en accord avec toutes les propriétés du système (électriques, électroniques…). A note que la couche réseau doit être la même pour chaque nœud.

Elle remplit les fonctions suivantes :

- Synchronisation des bits,
- La représentation des bits (timing, codage…),
- Définit les niveaux électriques des signaux,
- Définit le support de transmission.

II- Analyse technique

1. Caractéristiques électriques

Avant d'aborder l'aspect logiciel du Bus CAN, nous débutons par les caractéristiques électriques.

1.1 Support de transmission

Le câblage se présente par une paire filaire de type différentielle :

- CAN H (CAN High),
- CAN L (CAN Low).

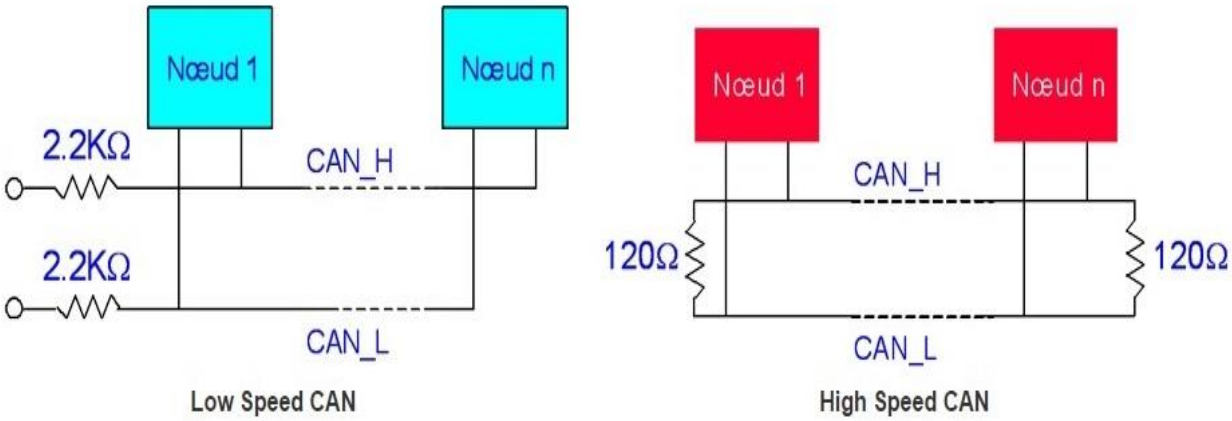


Figure 4 : Schéma de câblage Bus CAN

Le fait de transmettre en paire différentielle permet de supprimer les parasites qui peuvent être induits sur les lignes de communication.

Il existe trois principaux types de transmission possibles en CAN :

- CAN Low Speed,
- Can High Speed,
- CAN FD (Flexible Data Rate).

	CAN Low Speed	Can High Speed	CAN FD
Débit	125 kb/s	De 125 kb/s à 1 Mb/s	De 500 kb/s à 4 Mb/s
Nombre de nœuds	2 à 20	2 à 30	2 à 30
Courant de sortie	> 1 mA sur 2,2kΩ	25 à 50 mA sur 60Ω	25 à 50 mA sur 60Ω
Niveau dominant	CAN H = 4V CAN L = 1V	CAN H = 3.5V CAN L = 1.5V	CAN H = 3.5V CAN L = 1.5V
Niveau récessif	CAN H = 1.75V CAN L = 3.25V	CAN H = CAN L = 2.5V	CAN H = 2.5V CAN L = 1.5V
Caractéristiques du câble	30 pF entre CAN H et CAN L	Résistance de 120Ω aux deux extrémités du bus entre CAN H et CAN L (=60Ω)	Résistance de 120Ω aux deux extrémités du bus entre CAN H et CAN L (=60Ω)
Tensions d'alimentation	5V	5V	5V

Figure 5 : Types de transmission en CAN

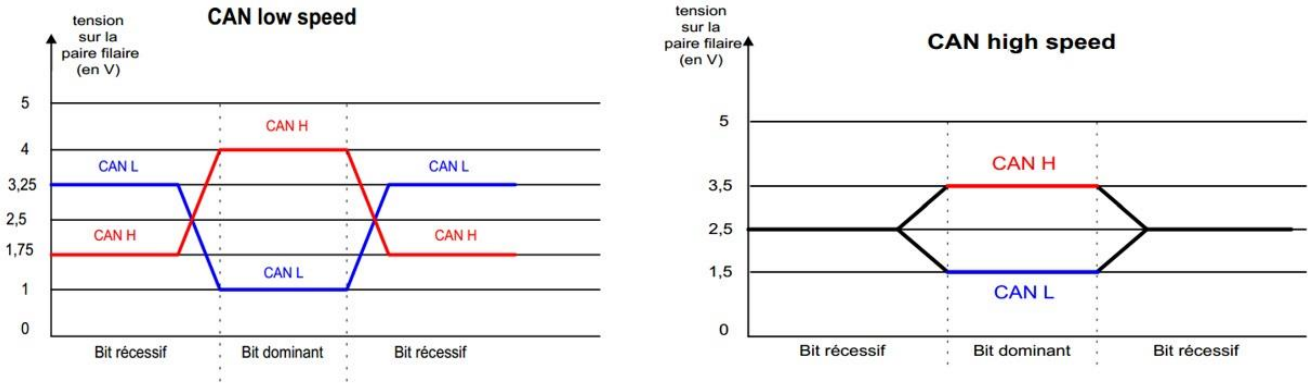


Figure 6 : Niveaux de tension en Low Speed & High speed

Il y a une nécessité d'utiliser des résistances de terminaison aux extrémités du bus afin de:

- Minimiser les réflexions sur le câble en adaptant l'impédance caractéristique de la ligne,
- Adapter l'impédance en continu de la ligne.

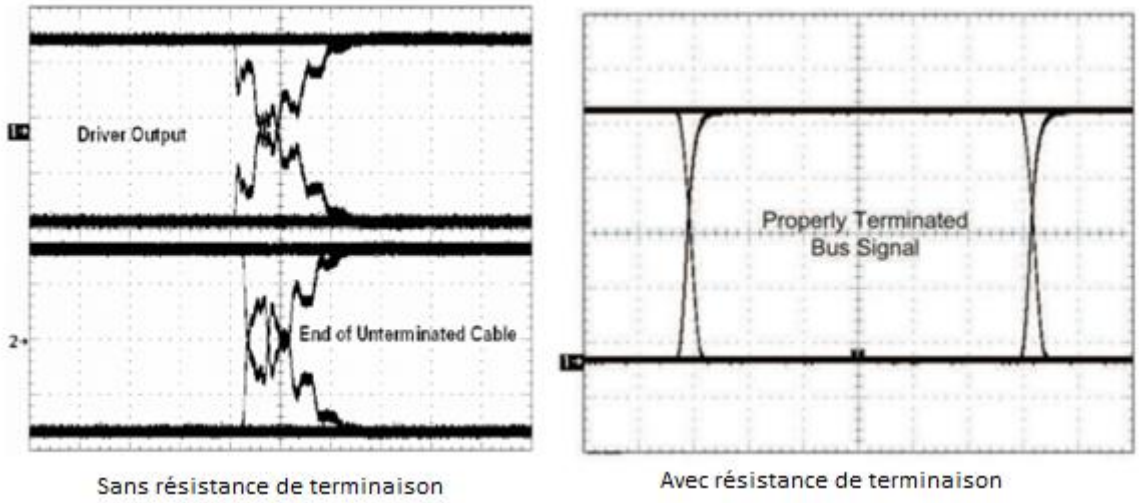


Figure 7 : Bus CAN sans résistance de terminaison et avec résistance de terminaison

À gauche, nous pouvons voir sur cette capture d'écran d'oscilloscope d'un bus sans résistance de terminaison que le signal est réfléchi et s'ajoute au signal original.

Les types de câbles à utiliser sont:

- Paire torsadée non blindée (UTP) avec impédance caractéristique de 120Ω,
- Paire torsadée blindée (STP) avec impédance caractéristique de 120 Ω.

1.2 Effet des longueurs de câble

La longueur du bus dépend des paramètres suivants :

- Le temps de propagation sur les lignes physiques du bus,
- La différence due au quantum à cause du temps de propagation défini précédemment. Cela est dû aux différences de cadencement des oscillations des nœuds,
- La variation de l'amplitude du signal en fonction de la résistance du câble et de l'impédance d'entrée des nœuds.

Pour faire fonctionner le Bus CAN avec une longueur de câble supérieure à 200 mètres, il est nécessaire d'utiliser un optocoupleur. Dans le cas où la longueur serait supérieure à 1 kilomètre, il est nécessaire d'utiliser des systèmes d'interconnexion. Un répéteur qui permet de régénérer un signal ou des ponts qui relient des réseaux locaux de même type peuvent être utilisés. Les modules connectés au Bus CAN doivent pouvoir supporter un débit supérieur à 20kbit/sec.

Débit (kb/s)	Longueur (mètres)	Longueur d'un bit (µs)
10	5 000	100
20	2 500	50
62.5	1 000	16
125	500	8
250	250	4
500	100	2
800	50	1,25
1 000	30	1

Figure 8 : Propagation dans les câbles

Le signal qui correspond à un bit émis par un nœud se propage à une vitesse de 200000 km/s sur les lignes électriques et optiques. La plus longue durée de propagation est celle où un bit doit parcourir le bus d'une extrémité à l'autre (on note cette durée *t_bus*).
Afin de ne pas créer de conflits entre les nœuds, le temps nominal d'un bit (*tn_bit*) doit être égal à deux fois le *t_bus*.

$$tn_{bit} > 2 * t_{bus} \quad \text{or} \quad t_{bus} = longueur\_du\_bus/200000$$

Sachant que le débit du réseau (Débit_du_réseau) = $1/tn_{bit}$, on obtient la relation suivante :

$$\text{Débit\_du\_réseau} = \frac{1}{tn_{bit}} = \frac{1}{2*t_{bus}} = \frac{1}{2*(\frac{longueur\_du\_bus}{200000})}$$

1.3 Le Non-Return to Zero

Dans le cas du Bus CAN, c'est le codage NRZ qui est employé (Non-Return to Zero). Concrètement, pendant toute la durée de génération du bit, le niveau reste constant, qu'il soit récessif ou dominant. Tous les schémas de trame qui suivent seront représentés de cette manière.

Cette méthode de codage possède un inconvénient majeur. Lors de la transmission d'une longue série de bit du même niveau, cela peut entrainer un problème de synchronisation. Pour ce type

de séquence, on préfèrera le codage Manchester qui consiste à effectuer un changement de niveau pour chaque bit transmis. Dans le cas du Bus CAN, on utilise le procédé du « bit stuffing ».

1.4 Le « bit-stuffing »

Une trame est codée en NRZ. Or, il se peut qu'une trame contienne beaucoup de bits d'un même niveau, ce qui peut laisser penser aux différents nœuds qu'une erreur existe sur le réseau. Ainsi, afin de « casser » ce rythme et affirmer que tout va bien, un bit de niveau opposé aux autres est ajouté à la suite de 5 bits successifs de même valeur lors de la transmission. On les appelle les bits de « remplissage » ou de « bourrage », en anglais stuff. Logiquement, cette technique allonge quelque peu le temps de transmission d'un message, mais elle en assure le bon transport de son contenu.

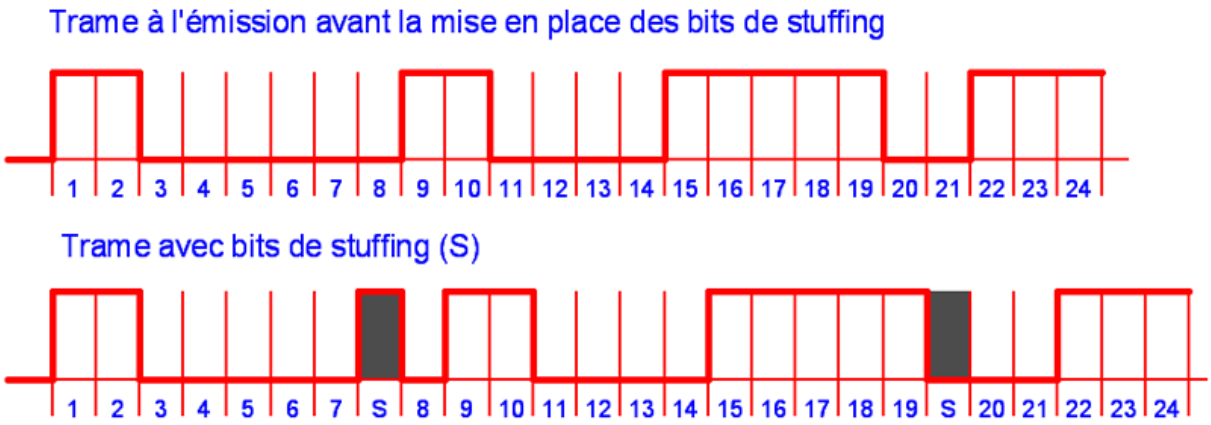


Figure 9 : Démonstration du "bit stuffing"

Pour que cette méthode fonctionne, les récepteurs CAN doivent connaître la technique de bourrage. Lors de la réception de la trame, le récepteur va procéder à la fonction inverse de « destuffing » en retirant ces bits de remplissage pour reconstituer le message d'origine.

Cette technique de « surcodage » est totalement transparente pour l'utilisateur, mais les erreurs sont remontées s'il s'en produit. À noter aussi que le fait d'insérer ces bits de stuffing permet de créer un plus grand nombre de transitions, et donc de favoriser la synchronisation de la communication malgré le codage NRZ des trames.

Enfin, tous les champs d'une trame ne sont pas concernés par cette méthode. Le début de trame (Start Of Frame), le champ d'arbitrage (Arbitration Field), le champ de contrôle (Control Field), le champ de données (Data Field) et le champ de CRC (Cyclic Redundancy Code Field) sont codés avec le bit stuffing.

2. Trames CAN

Il existe quatre types de trames et un intervalle de temps dans le Bus CAN :

- La trame de donnée (*Data Frame*) : elle est générée par un nœud dit « producteur » qui désire transférer des données, ou comme réponse à la requête d'un autre nœud.
- La trame de requête (*Remote Frame*) : elle est générée par un nœud dit « consommateur » ou demandeur de données.
- La trame d'erreur (*Error Frame*) : elle est générée par n'importe quelle entité du bus lors de la détection d'une erreur.
- La trame de surcharge (*Overload Frame*) : elle est utilisée pour demander un laps de temps supplémentaire entre les Data Frames ou les Remote Frames lorsque celles-ci sont successives.
- L'intervalle de temps (*Interframe*) : les Data Frames et Remote Frames sont séparées temporairement par une Interframe.

2.1 Décomposition d'une trame

2.1.1 Trames de données

Une trame de données se décompose en 7 champs différents:

- le début de trame SOF (*Start Of Frame*), 1 bit dominant,
- le champ d'arbitrage, 12 bits,
- le champ de contrôle, 6 bits,
- le champ de données, 0 à 64 bits,
- le champ de CRC (*Cyclic Redundancy Code*), 16 bits,
- le champ d'acquittement (*Acknoledge*), 2 bits,
- le champ de fin de trame EOF (*End Of Frame*), 7 bits récessifs,

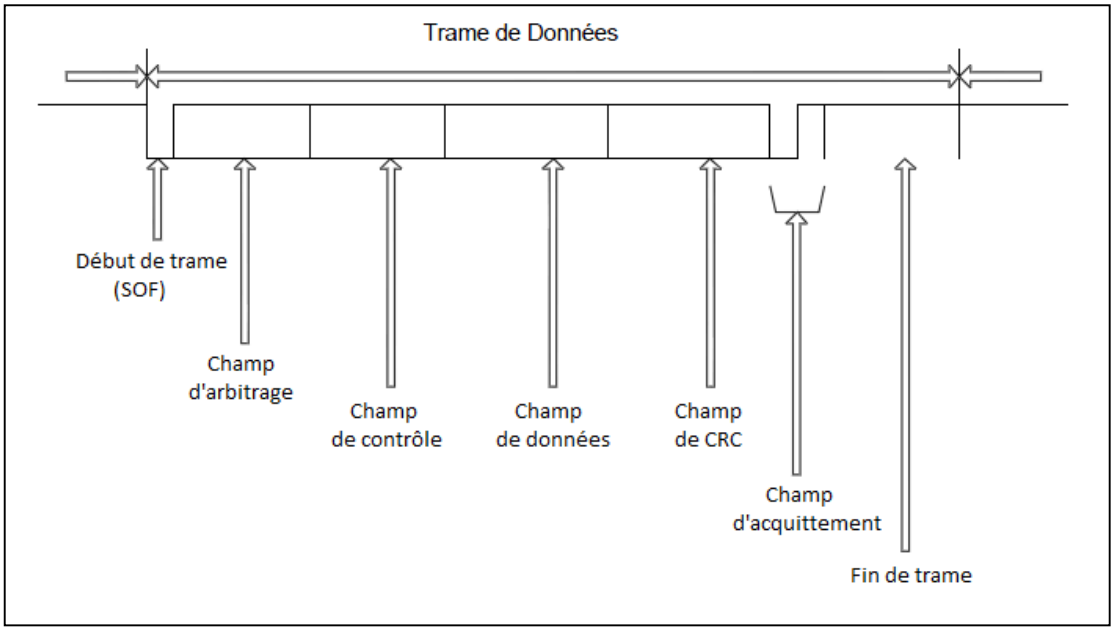


Figure 10 : Décomposition d'une trame de données

2.1.2 Trames de requête

Une trame de requête est constituée de la même manière qu'une trame de données sauf que le champ de données est vide.

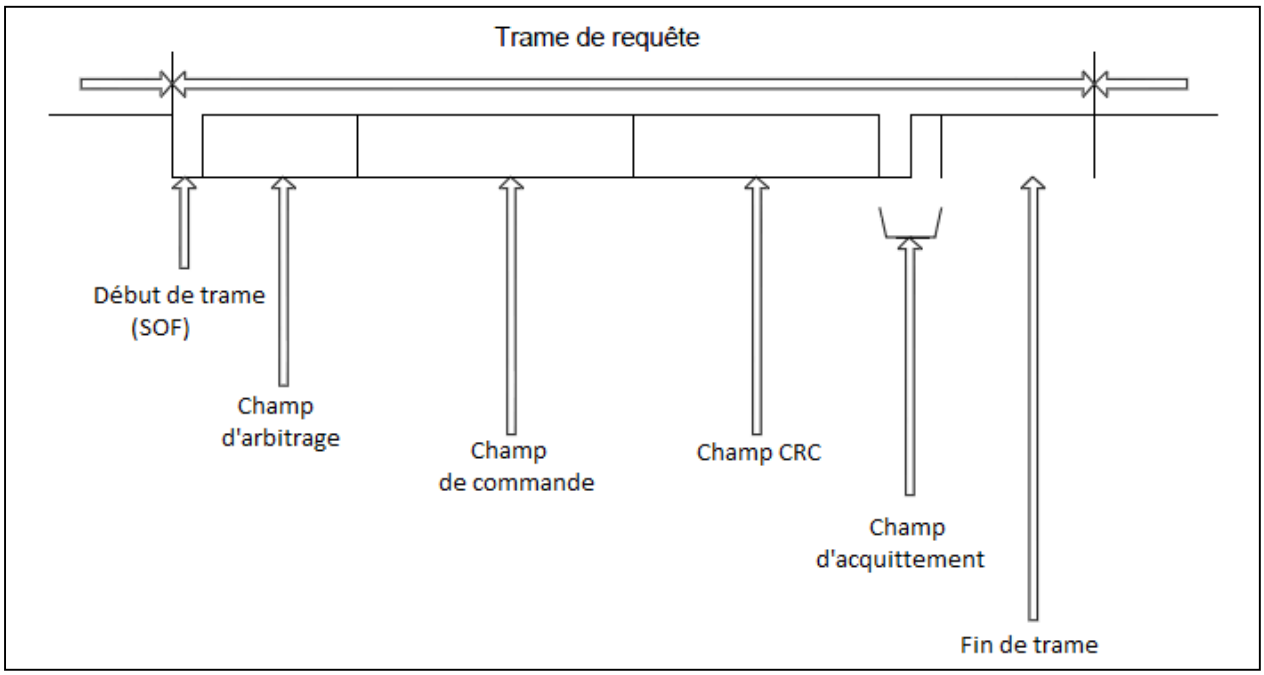


Figure 11 : Décomposition d'une trame de requête

2.1.3 Trames d'erreur

Une trame d'erreur permet de signaler aux autres nœuds CAN la présence d'une erreur (Passive ou Active).

La trame d'erreur est constituée de deux champs principaux : le drapeau d'erreur et le délimiteur de champ.

Ci-dessous voici la façon dont se construit une trame d'erreur :

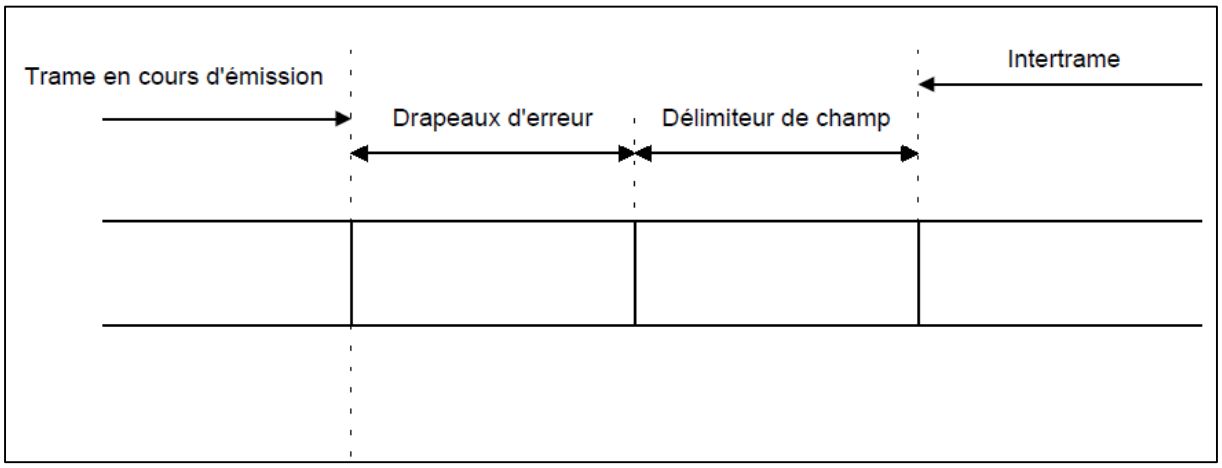


Figure 12 : Constitution de la trame d'erreur

Il existe deux sortes de champs drapeaux :

- La trame d'erreur active (Active Error Flag):

La trame d'erreur active est formée de six bits dominants consécutifs pour le drapeau (Flag error active) suivi de huit bits récessifs pour le délimiteur (Error Delimiter).

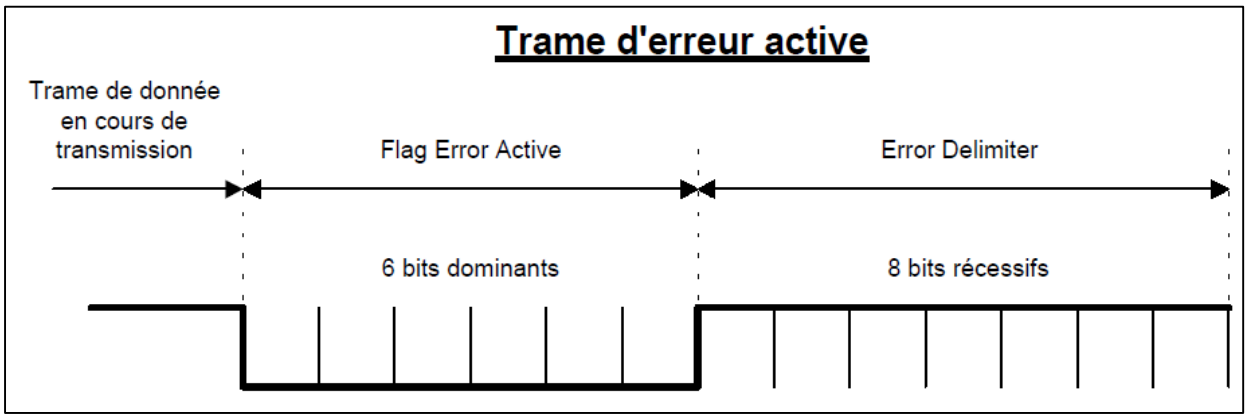


Figure 13 : Trame d'erreur active

- La trame d'erreur passive (Passive Error Flag) :

La trame d'erreur passive est formée de six bits récessifs pour le drapeau (Flag error active) et de huit bits récessifs pour le délimiteur (Error Delimiter).

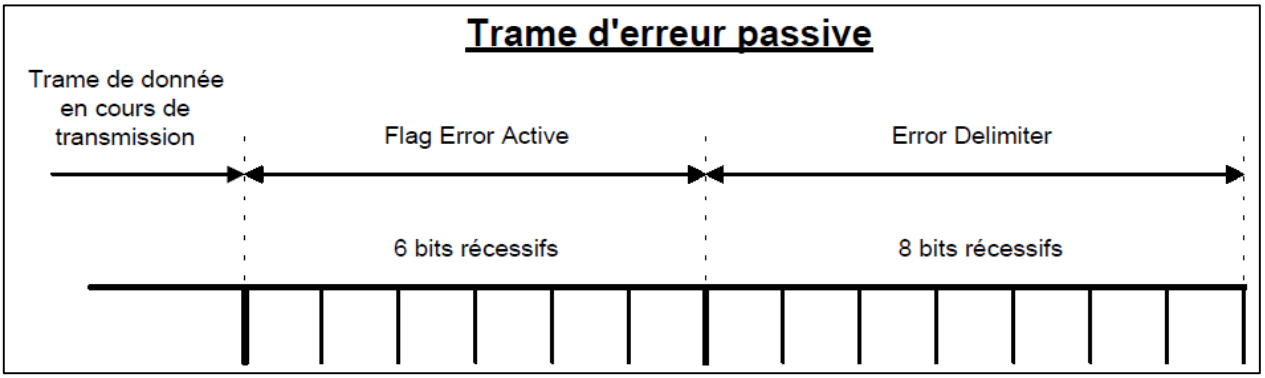


Figure 14 : Trame d'erreur passive

Le délimiteur d'une trame d'erreur est toujours constitué de 8 bits récessifs.

2.1.4 Trames de surcharge

Le but de cette trame est d'indiquer qu'une entité est surchargée pendant un certain laps de temps. Elle se déclenche sous deux conditions :

- Lorsqu'un nœud demande un certain temps avant d'accepter la prochaine trame de données ou de requêtes,
- Lorsqu'un nœud détecte un bit dominant pendant l'intertrame (3 bits récessifs entre les trames),

Dans le but de ne pas encombrer le bus indéfiniment, seules deux trames de surcharge peuvent être générées de manière consécutive. Elle se compose de deux champs : le champ des flags de surcharge et le champ délimiteur de surcharge.

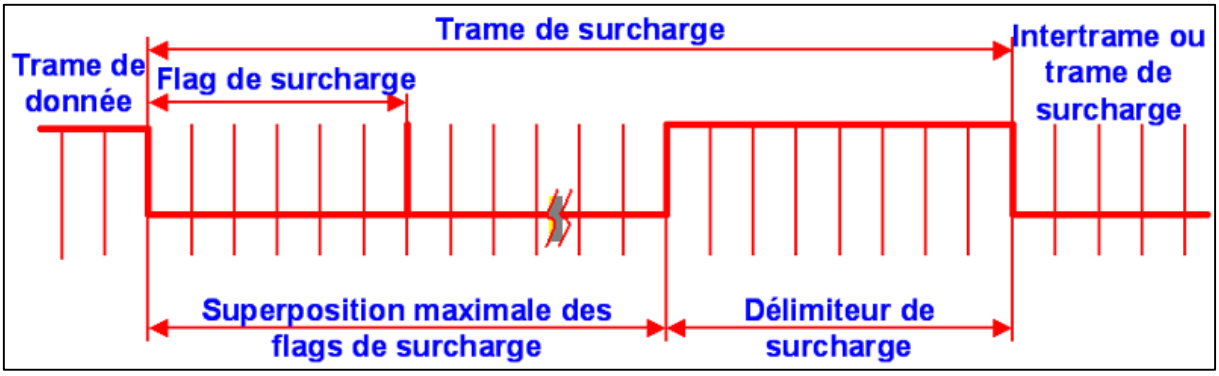


Figure 15 : Décomposition de la trame de surcharge

Le champ des flags de surcharges est composé de 6 bits dominants consécutifs. Il détruit le champ intermission de l'intertrame.

Le champ délimiteur de surcharge est composé de 8 bits récessifs consécutifs.

Après le passage d'un flag de surcharge, l'entité examine le bus jusqu'à ce qu'elle détecte une transition qui indique le passage d'un bit dominant à un bit récessif. À ce moment-là, chaque entité sur le bus a terminé d'envoyer son flag de surcharge et elles envoient donc 7 bits récessifs consécutifs pour former le délimiteur de surcharge.

2.2 Analyse des différents champs

Chaque trame démarre par un bit de SOF *(Start of Frame)* signalant le début d'un échange. Cet échange ne peut démarrer que si le bus était précédemment au repos. Le bit SOF marque le début d'une trame (*Data frame ou Remote frame*). Il consiste en un simple bit qui doit absolument être dominant. Tous les nœuds du réseau doivent se resynchroniser sur le bit de SOF.

2.2.1 Le champ d'arbitrage (Arbitration field)

Dans le champ d'arbitrage, on retrouve l'identificateur qui est composé de 11 bits *(ID[10..0])* et le bit de RTR (*Remote Transmission Request*).

Les 11 bits de l'identificateur sont transmis dans l'ordre, de ID[10] à ID[0] (MSB ID[0]). Il ne faut pas que les 7 bits les plus significatifs (ID[10..4]) soient tous récessifs. Pour des raisons de compatibilité avec des anciens circuits, les 4 derniers bits de l'identificateur (ID[3..0]) ne sont pas utilisés, ce qui réduit le nombre de combinaisons possibles pour l'identificateur.

Le bit RTR est dominant pour une trame de données et récessif pour une trame de requête.

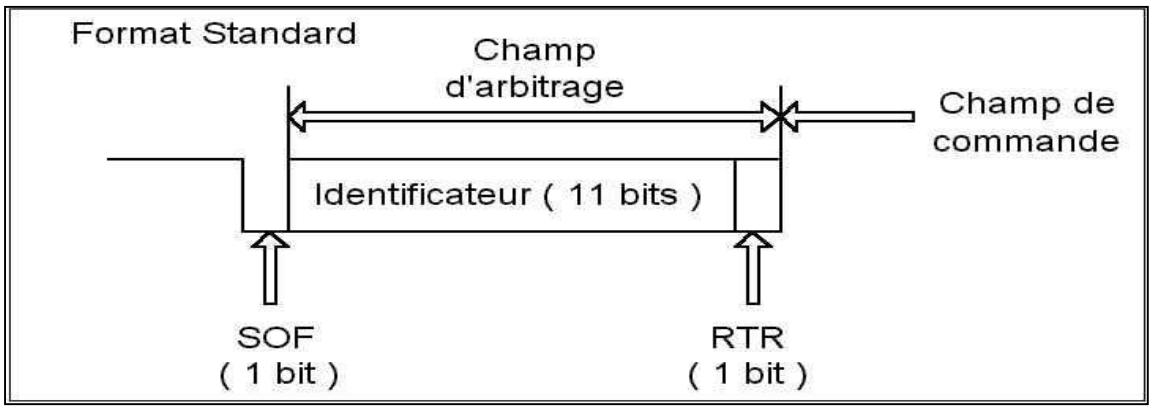


Figure 16 : Champ d'arbitrage

2.2.2 Le champ de contrôle (Control field)

Dans le champ de contrôle, on retrouve 2 bits réservés (r0 et r1), ainsi que 4 bits DLC[3..0] qui déterminent la longueur du champ de données (Data Length Code).

Les bits réservés (r0 et r1) sont des bits de réserve pour une expansion future. Ils doivent être envoyés "dominants".

Les 4 bits DLC permettent de déterminer le nombre d'octets qui seront contenus dans le champ de données. Ce nombre ne peut pas excéder la valeur de 8.

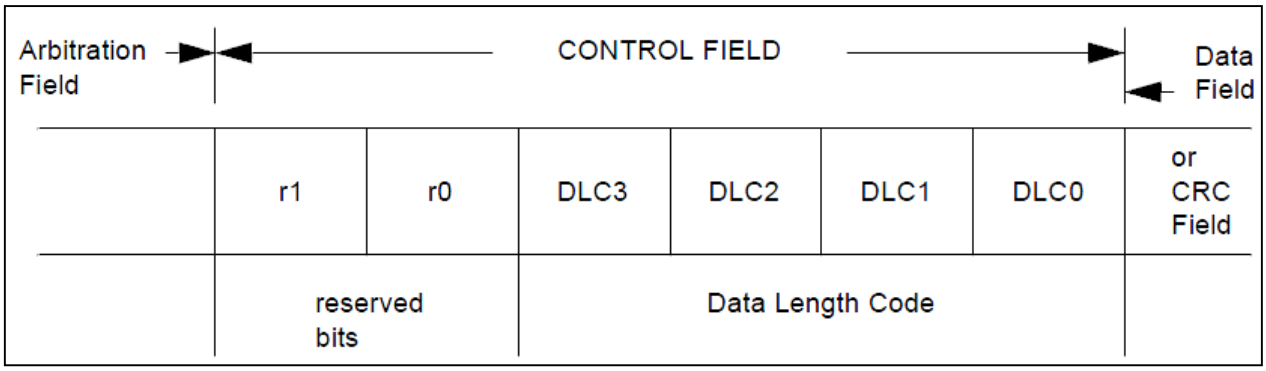


Figure 17 : Champ de contrôle

2.2.3 Le champ de données (Data field)

Le champ de données a une longueur qui peut varier de 1 à 8 octets (1 octet = 8 bits) donc de 0 à 64 bits. Cette longueur dépend du DLC du champ de contrôle. Pour une trame de requête, le champ de données est vide.

Ci-dessous le codage des bits DLC suivant la taille de données en octets.

Number of Data Bytes	Data Length Code			
	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d

d : bit dominant

r : bit récessif

Figure 18 : Codage des bits DLC suivant la taille des données en octets

2.2.4 Le champ de CRC (Cyclic Redundancy Code field)

Dans le champ de CRC, on retrouve la séquence CRC composée de 15 bits ainsi qu'un délimiteur de fin de champ CRC.

La séquence de contrôle de trame est dérivée d'un code de redondance cyclique qui convient mieux à des trames de 127 bits au moins. La séquence de CRC est calculée à partir de la trame SOF, des champs d'arbitrage, de contrôle et du champ de donnée (si trame de donnée). Elle est calculée de la façon suivante :

1) On interprète comme un polynôme tous les bits depuis le début de la trame (SOF), jusqu'à la fin du champ de données (Data field) pour une trame de données. S'il s'agit d'une trame de requête, on interprète jusqu'à la fin du champ de contrôle (Control field). On affecte des coefficients 0 ou 1 au polynôme suivant la présence de chaque bit.

2) Le polynôme obtenu P(x) est alors multiplié par x^{15} pour l'ajout du mot CRC.

3) Le polynôme obtenu est divisé par le polynôme générateur suivant :
$G(x) = X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$.
La chaîne de bits correspondante à ce polynôme est : "1100010110011001".

4) Le reste de la division du polynôme P(x) par le polynôme générateur G(x) représente la séquence CRC de 15bits.

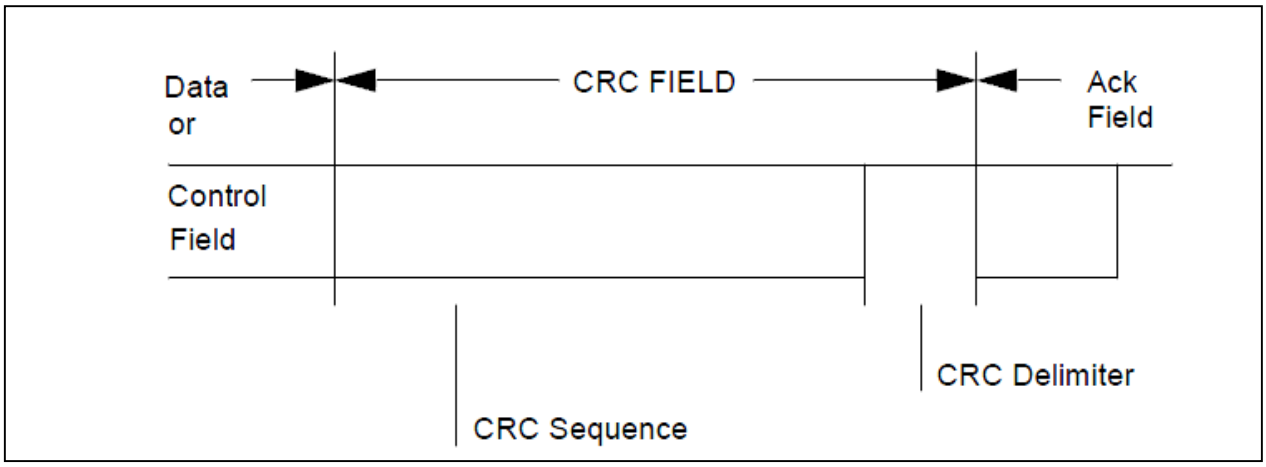


Figure 19 : Champ CRC

La norme Bosch décrit le programme informatique correspondant à l'algorithme présenté au-dessus.

```
CRC_REG=0 ;                                          // initialize shift register
REPEAT
        CRC_NXT_BIT=(NXT_BIT) XOR (CRC_REG(14)) ;
        CRC_REG(14:1)=CRC_REG(13:0) ;                // shift left by
        CRC_REG(0)=0 ;                                       // 1 position
        IF CRC_NXT_BIT THEN
                CRC_REG(14:0)=CRC_REG(14:0) XOR (4599hex) ;
        ENDIF
UNTIL(CRC SEQUENCE starts or there is an ERROR condition)
```

Figure 20 : Algorithme CRC Bosch

2.2.5 Le champ d'acquittement (ACK field)

Afin que les nœuds récepteurs puissent vérifier la consistance du message reçu, il y a la présence d'un champ d'acquittement.

Le champ d'acquittement est constitué des bits suivants :

- Un bit d'acquittement (récessif)
- Un bit délimiteur (récessif)

Ce champ d'acquittement permet aux nœuds récepteurs d'envoyer, pendant l'émission de ces deux bits, un bit dominant pour confirmer la bonne réception de la trame. En cas de mauvaise réception, le nœud récepteur enverra une trame d'erreur.

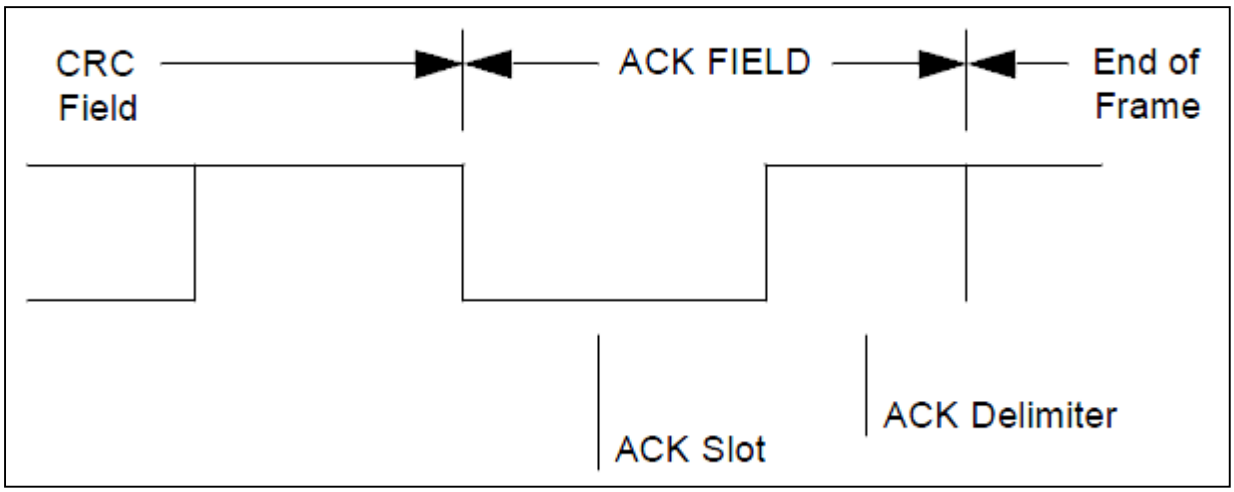


Figure 21 : Champ d'acquittement

2.2.6 Le champ de fin de trame (End of frame field)

À la fin de chaque trame, il est envoyé une salve de 7 bits récessifs.

2.3 Période d'intertrame

Les trames de données et de requêtes sont séparées des autres trames par une intertrame, alors que les trames d'erreur et de surcharges ne sont pas séparées par cette intertrame. Celui-ci se compose de 2 ou 3 champs selon les cas : le champ intermission, le champ bus libre et pour les unités qui ont été émettrices du message précédent, le champ suspension transmission.

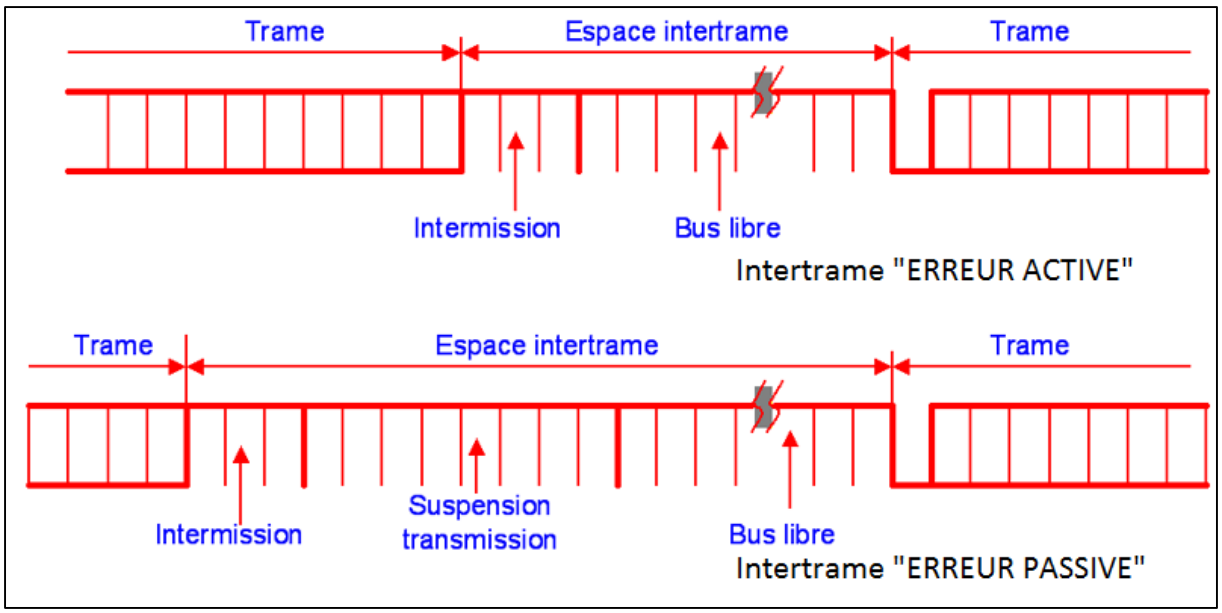


Figure 22 : Composition période d'intertrame

Le champ intermission est composé de 3 bits récessifs. Pendant l'intermission, aucune trame de type données ou requêtes n'est autorisée à démarrer, seule la trame de surcharge peut circuler.

La durée du champ bus libre peut être arbitrairement choisie. Pendant ce laps de temps, la ligne est « libre » et n'importe quelle entité peut accéder au bus. Si un message était en attente lors de la précédente transmission, alors celui-ci pourra démarrer dès le premier bit suivant l'intermission. Ce bit sera interprété comme le SOF (Start Of Frame) de la nouvelle transmission.

Après qu'une entité avec un statut « Error passive » ai transmis un message, elle envoie le champ suspension transmission composé de 8 bits récessifs avant de démarrer une nouvelle transmission ou de reconnaître que le bus est « libre ». Si pendant ce même laps de temps une transmission démarre (par une autre entité) alors la première entité deviendra réceptrice de ce message.

3. Gestion des priorités

Une trame de donnée est toujours prioritaire à une trame de requête. Lorsqu'un nœud souhaite recevoir un certain nombre d'octet, il emmétra une trame de requête avec dans son contenu le nombre d'octet dont il a besoin. Un bit, appelé RTR (Remote Transmission Request) permet d'abriter la priorité qui sera donnée aux trames. En effet, le bit RTR sera dominant dans le cas d'une trame de donnée et récessif dans une trame de requête. La priorité est toujours donnée à une trame de donnée grâce à la vérification du bit RTR.

Avec l'exemple suivant, on compare deux trames ayant les mêmes identificateurs, on peut remarquer que la trame de donnée est prioritaire sur la trame de requête, car le bit RTR est dominant.

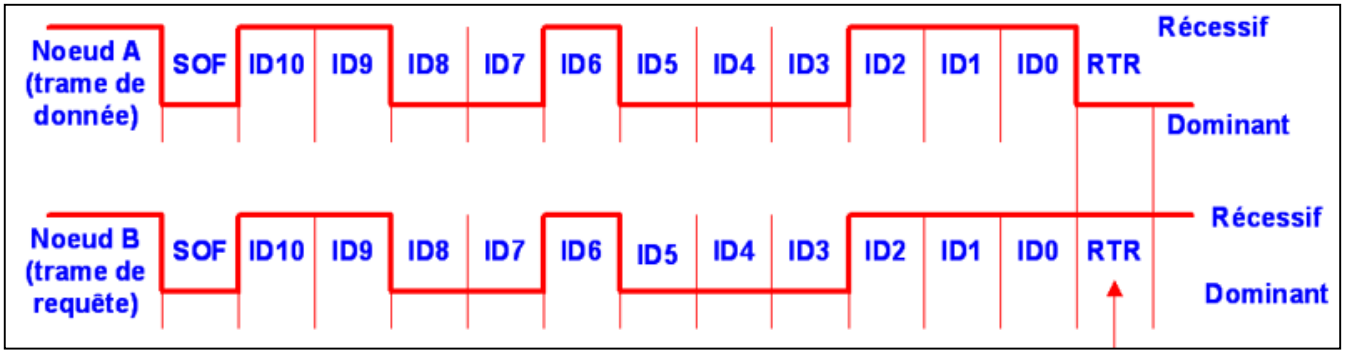


Figure 23: Comparaison de deux niveaux de priorité

4. Gestion des erreurs

Afin de comprendre comment se fait la gestion des erreurs, nous allons reprendre toutes les erreurs qui peuvent être rencontrées, les détailler et enfin expliquer comment le Bus CAN gère ces erreurs.

Des erreurs de transmission peuvent perturber le fonctionnement des différents nœuds sur le Bus CAN. Un nœud peut être à l'origine d'une communication perturbée ou impossible sur le BUS. Dans ces cas-là, il existe des méthodes afin de détecter les erreurs de transmissions dans le protocole CAN.

4.1 Les différents types d'erreurs

4.1.1 Le Bit Error

Lors de l'émission d'un bit sur le bus, une vérification est effectuée afin de surveiller si le niveau émis est le même que celui qui est attendu. S'il n'y a pas de correspondance, il s'agit d'un *bit error*. Mais cette erreur n'est pas signalée dans les cas qui suivent :

- Si un bit dominant se situe dans le champ d'arbitrage au lieu d'un bit récessif il s'agit d'une perte d'arbitrage sur le bit dominant.
- Si un émetteur envoie un « flag d'erreur passive » (bit récessif) et reçoit un bit dominant

4.1.2 L'erreur de Stuffing

Lorsqu'il y a au-delà de 6 bits consécutifs de même signe sur le bus, il s'agit d'une erreur de Stuffing. Elle est signalée uniquement dans les champs d'identificateurs, de commandes et de CRC. La règle du bit-Stuffing n'est pas déclarée après le CRC. Cette erreur n'est pas signalée dans le champ de fin de trame et le champ d'acquittement.

4.1.3 L'erreur de Cyclic Redundancy Code (CRC)

Lorsque la valeur reçue par le calcul du nœud récepteur est différente de celle envoyée par le nœud émetteur, il s'agit d'une erreur de CRC (CRC Error).

4.1.4 L'erreur d'Acknowledge Delimiter

Lorsque dans le champ d'Acknowledge Delimiter, le nœud récepteur ne voit pas de bit récessif, il s'agit d'une erreur d'Acknowledge Delimiter. Il s'agit de la même erreur pour le CRC Delimiter.

4.1.5 L'erreur de Slot Acknowledge (Acknowledgment Error)

Dans le champ de Slot Acknowledge, lorsque l'émetteur ne lit pas un bit dominant, il s'agit d'une erreur de *Slot Acknowledge*.

Voici sur la figure 24 les différents types d'erreurs ainsi que leurs validités en fonction de l'emplacement dans la trame.

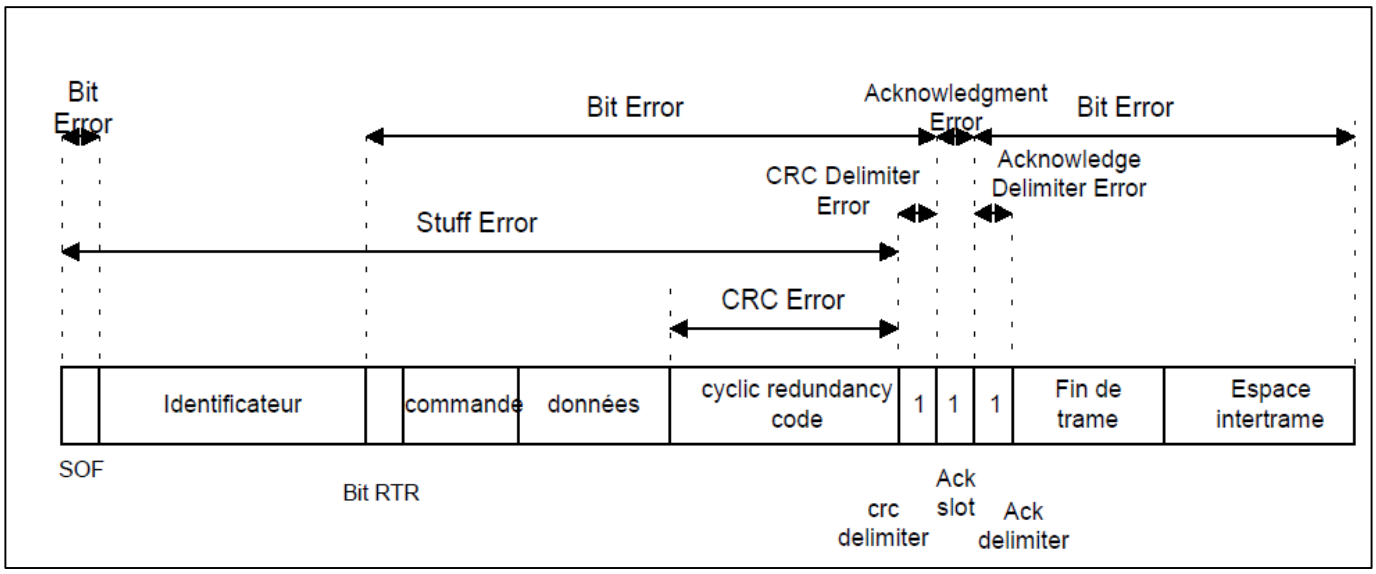


Figure 24 : Les sources d'erreur dans la trame CAN

4.2 Les modes d'erreur

En cas d'erreur(s) de transmission sur une trame, le nœud envoyeur retourne la trame jusqu'à que la transmission s'effectue sans erreur.

Il existe deux compteurs d'erreurs :

- Un compteur nommé TEC (Transmit Error Counter), pour les erreurs d'émission
- Un compteur nommé REC (Receive Error Counter), pour les erreurs de réception

La valeur de ces compteurs s'incrémente lorsqu'il existe des erreurs sur la transmission des trames. Si les trames se transmettent correctement, les compteurs se décrémentent à mesure que la transmission se fait normalement.

Selon la gravité des erreurs, les valeurs d'incrémentation des compteurs sont différentes (1 ou 8).

Si la valeur de ces compteurs est trop grande, le nœud incriminé se déconnectera du bus (Mode Bus Off) et sera donc dans l'incapacité d'émettre et de recevoir des trames.

Il existe différents modes d'erreurs dont leur règle de passage est régie par l'état des compteurs (figure 25):

- Le mode Error Active, lorsque la valeur d'un des deux compteurs est inférieure à 128
- Le mode Error Passive, lorsque la valeur d'un des deux compteurs est supérieure à 128
- Le mode Bus Off, lorsque le nœud CAN observe 128 occurrences de 11 bits récessifs

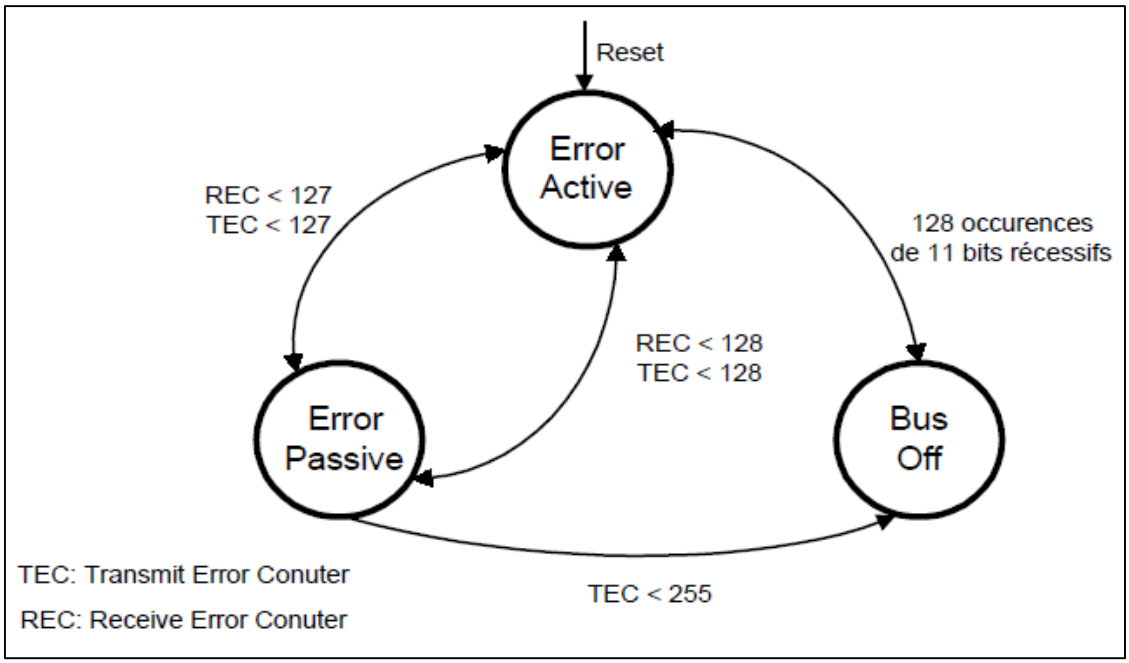


Figure 25 : Compteur d'erreur et état d'un nœud

5. Modes de fonctionnement

5.1 Le mode sommeil

Dans un but de réduction de la consommation d'énergie, les éléments du CAN peuvent se mettre en « Sleep mode ». Le nœud CAN concerné n'a pas d'activité et les drivers ne sont pas connectés au bus.

5.2 Le mode de réveil

Le mode « Wake-up » permet au bus de se réveiller après le « sleep mode ». Il s'effectue lors d'une reprise de l'activité sur le bus ou par décision interne à l'élément CAN. Un temps d'attente dû à la resynchronisation de l'oscillateur local est observé. En effet, il vérifie la présence de 11 bits consécutifs sur le bus. Ensuite, les drivers se connectent au bus, mais les premiers messages sont perdus à cause de la resynchronisation. Ils sont renvoyés afin de ne perdre aucune information.

Des oscillateurs à quartz sont utilisés afin d'obtenir de bonnes performances de débit sur le réseau.

III- Réalisation

1. Présentation de la réalisation

Nous avons décidé d'effectuer notre réalisation sur le Bus CAN avec une application qui se rapproche de celle de l'automobile. Elle est composée de 3 nœuds qui démontrent l'intérêt de ce bus de terrain :

- Le premier nœud simule une clé de démarrage d'un véhicule. Il détermine l'état moteur du véhicule. Il est réalisé avec un PIC18F4580.
- Le deuxième nœud se compose de différents capteurs que l'on retrouve dans un véhicule. Il est réalisé avec une carte Arduino. Trois capteurs déterminent la vitesse du véhicule ainsi que la température et la luminosité extérieure.
- Le troisième nœud représente un tableau de bord d'un véhicule. Il est réalisé sur une interface Labview. Cette interface affiche l'état du moteur ainsi que les données des différents capteurs.

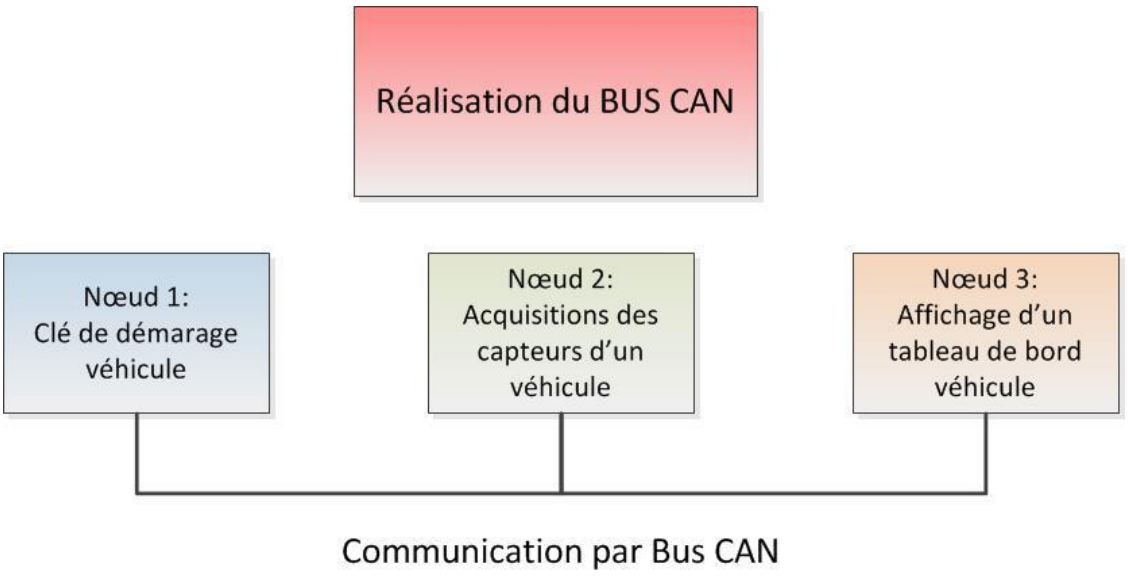


Figure 26: Présentation de la réalisation

2. Le découpage fonctionnel du projet (WBS)

Le Work Breakdown Structure (WBS) ci-dessous est un arbre représentant la liste structurée de toutes les fonctionnalités du projet. Il a pour but de nous aider à organiser notre projet. Il établit la planification et permet également de déléguer la mission confiée à chaque acteur.

Pour ce faire il faut :

- Effectuer l'inventaire exhaustif des tâches à réaliser
- Identifier les fonctions que doit avoir notre réalisation
- Attribuer à chaque fonction un responsable unique
- Définir de façon claire les niveaux d'exigence et leurs limites

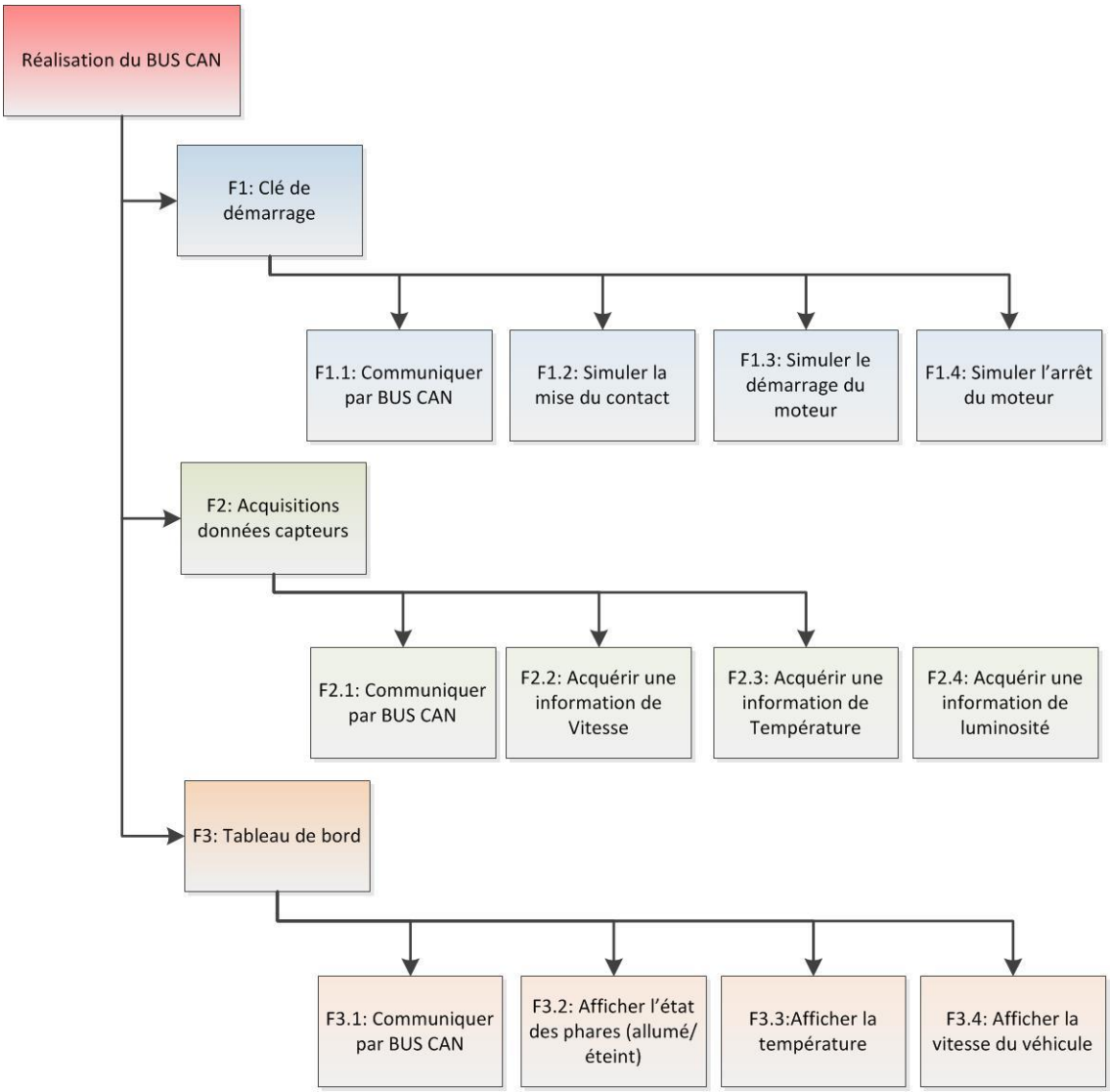


Figure 27: Arbre fonctionnel de la réalisation

3. L'organigramme des ressources du projet (RBS)

Le Resource Breakdown Structure a pour objectif de décomposer le projet en ressources et de les regrouper par nature ou en équipes. Le chef de projet doit, à travers le RBS, mettre en place des calendriers de travail, vérifier la disponibilité en termes de compétences et affecter les responsabilités hiérarchiques.

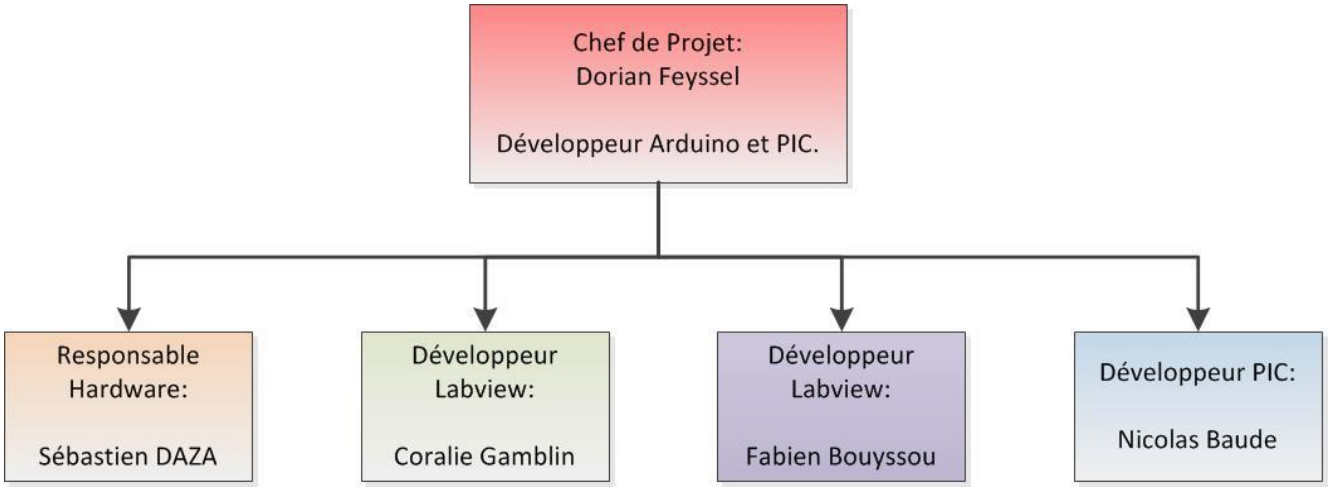


Figure 28: Organigramme ressources du projet

4. Cahier des charges de la réalisation

Nous établissons des conditions de fonctionnement pour notre réalisation lors de la communication des différentes trames. Afin de se rapprocher du fonctionnement d'un véhicule, voici les règles de priorités que nous avons déterminées:

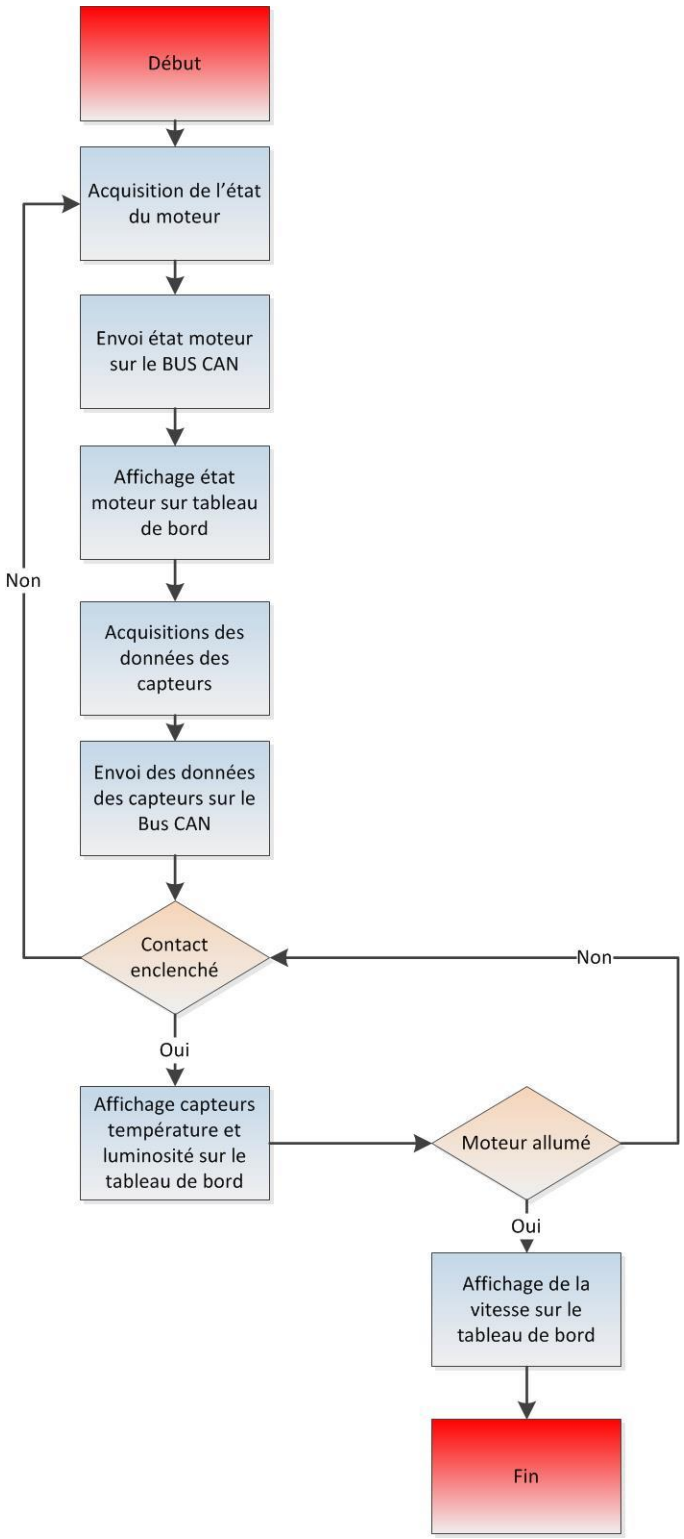


Figure 29: Cahier des charges de la réalisation

5. Aspects techniques

Ci-dessous est présenté la schématique électrique du projet que nous avons développé. Cette réalisation se répartit en 3 blocs fonctionnels :

- La partie Acquisition des données capteurs à base d'Arduino,
- La partie Clé de démarrage à base de PIC Microchip 18F4580,
- La partie Tableau de bord réalisée à partir du logiciel N.I Labview.

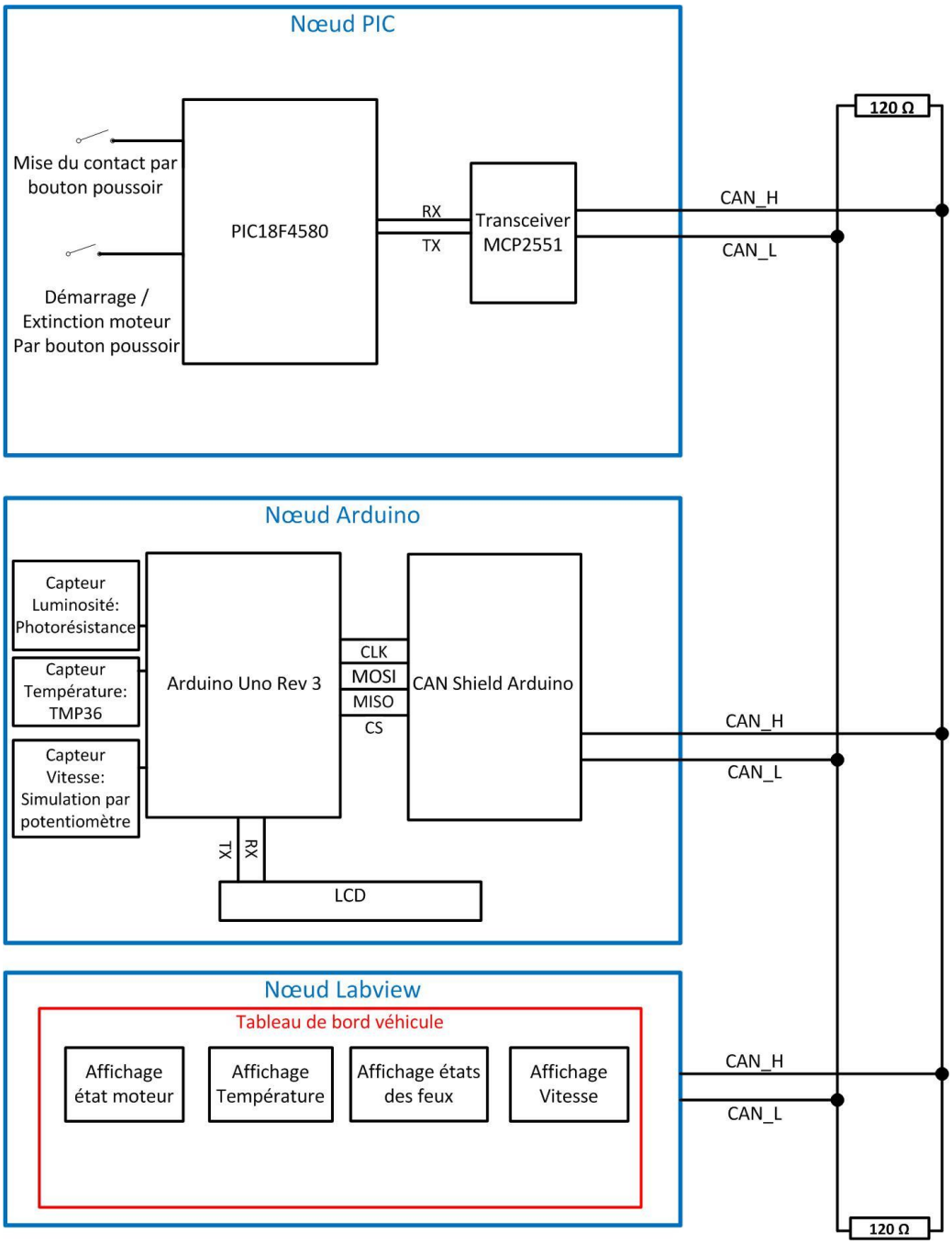


Figure 30: Schématique de communication de la réalisation

5.1 Clé de démarrage

Nous avons réalisé l'acquisition de l'état d'une clé de démarrage qui pourrait être présente dans un véhicule. L'état de la clé de démarrage est envoyé sur le Bus CAN afin d'y être traité par les autres nœuds.

Pour réaliser ce nœud, nous avons choisi une mise en œuvre par microcontrôleur. Un microcontrôleur est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires et interfaces d'entrées-sorties. Les microcontrôleurs se caractérisent par un plus haut degré d'intégration, une plus faible consommation électrique, une vitesse de fonctionnement plus faible et un coût réduit par rapport aux microprocesseurs polyvalents utilisés dans les ordinateurs personnels. Nous avons choisi un microcontrôleur de marque Microchip modèle PIC18F4580 car il intègre un module de communication spécialisé pour le Bus CAN.

Figure 31: Le microcontrôleur Microchip PIC18F4580

Pour pouvoir communiquer en CAN, nous devons rajouter un autre composant appelé Transceiver CAN. Ce composant permet d'adapter et de générer des signaux électriques compatibles avec les spécifications du Bus CAN à partir de la sortie CAN du PIC.

Figure 32: Le Transceiver CAN Microchip MCP2551

Pour développer ce nœud, nous avions à notre disposition une carte de développement gracieusement prêtée par l'Université Paul Sabatier :

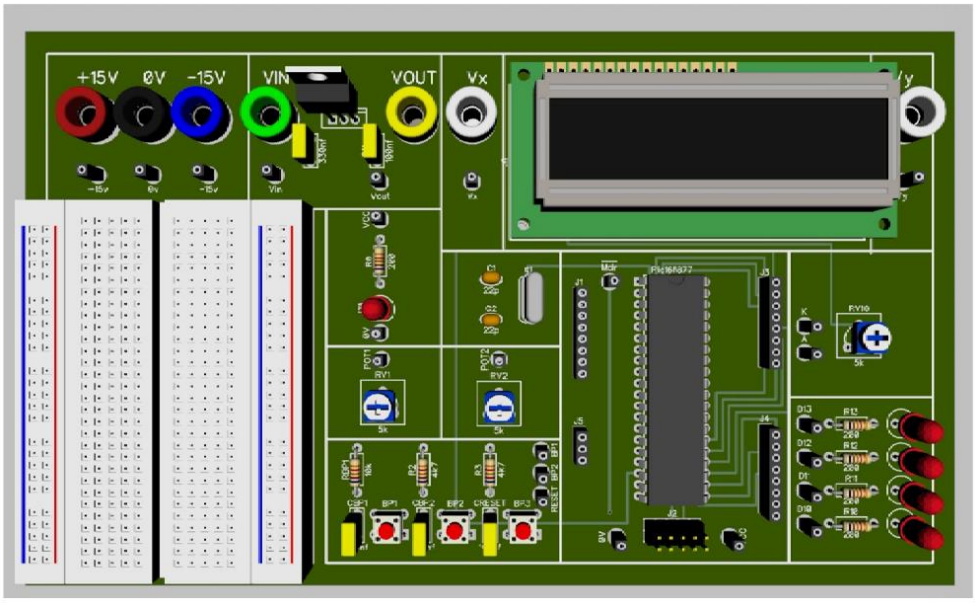


Figure 33: Modélisation 3D de la carte de développement PIC de l'Université Paul Sabatier

Cette carte de développement comporte plusieurs parties utiles à notre projet :

- Une partie « platine d'expérimentation » (à gauche) qui permet de facilement câbler des composants et de les connecter au PIC
- Une partie « boutons poussoirs » (au centre en bas) qui permet de simuler une clé de démarrage véhicule.
- Une partie « Affichage à LED » (à droite) qui permet d'afficher l'état de la clé tel qu'il est vu par le PIC

Afin de développer le logiciel qui régit le fonctionnement du PIC et ainsi satisfaire notre cahier des charges, nous avons fait appel au programme MikroElectronika MikroC :

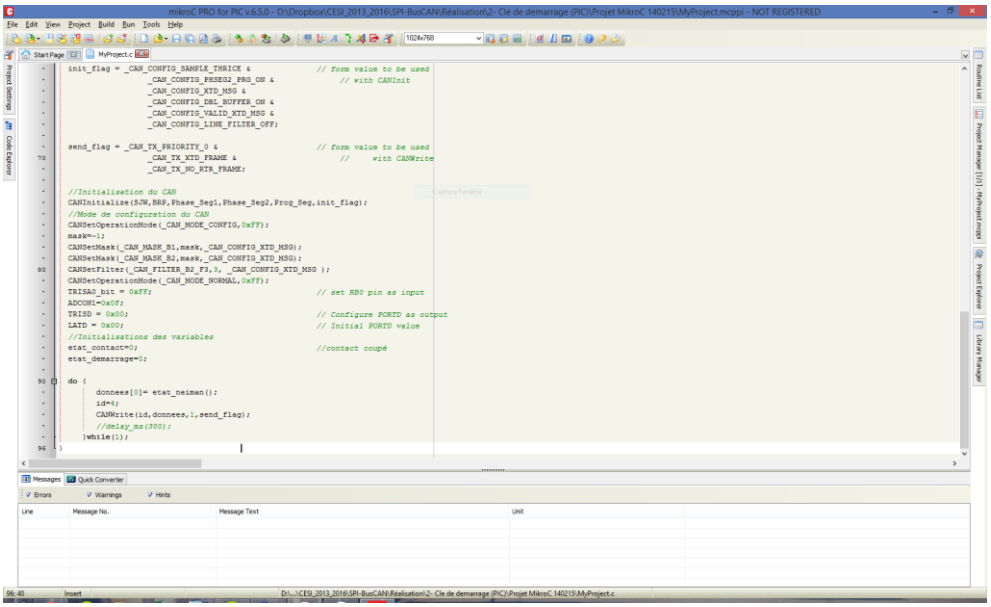


Figure 34: L'interface de développement MikroC

Le programme que nous avons développé peut se résumer par cet algorithme simplifié :

```
Initialisation_Bus_CAN() ;

Faire

{

    Acquisition_etat_clé() ;

    Envoi_Etat_Clé_CAN() ;

}Tant Que(Toujours) ;
```

On vient acquérir l'état de la clé, puis on envoie son état sur le Bus CAN en permanence à la vitesse de 1 Mbps.

Pour pouvoir programmer le logiciel présenté précédemment, nous avons besoin d'un programmateur spécial :

Figure 35: Le programmateur MikroElectronika MikroProg

Cet outil permet de transmettre le programme développé avec MikroC sur le PIC à travers une interface USB standard.

5.2 Acquisitions données capteurs

Nous avons décidé de réaliser l'acquisition de différents capteurs pouvant être présents dans un véhicule. Les données de ces capteurs sont envoyées sur le Bus CAN afin d'y être traité par les autres nœuds.

Afin de réaliser ce nœud, nous avons décidé d'utiliser les trois capteurs suivants qui représentent:

- La vitesse du véhicule,
- La luminosité extérieure du véhicule,
- La température extérieure du véhicule.

Pour réaliser l'acquisition, le traitement et l'envoi des données de ces trois capteurs sur le Bus CAN, nous avons choisi d'utiliser une carte Arduino Uno.

Figure 36: Carte Arduino Uno

La carte Arduino Uno est un module sur lequel se trouve un microcontrôleur qui peut être programmé pour analyser et produire des signaux électriques. Nous avons fait le choix d'utiliser cette carte car elle est simple d'utilisation et est parfaitement adaptée à nos capteurs. En effet, nous utilisons 3 capteurs analogiques qui s'interfacent parfaitement avec notre carte. L'acquisition et le traitement de leurs données sont réalisés par programmation en langage C simplifié à l'aide d'un logiciel fourni avec la carte.

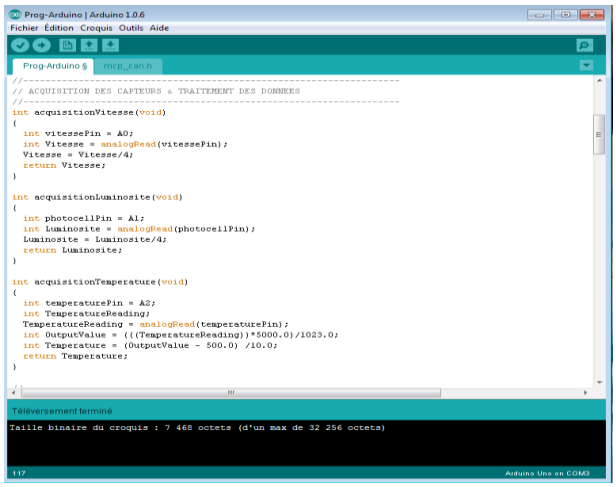


Figure 37: Logiciel développement Arduino

Notre choix des composants pour les trois capteurs présentés précédemment est le suivant :

- Un capteur de vitesse: Un potentiomètre, dont la tension représente la vitesse du véhicule.
- Un capteur de luminosité: Une photocellule dont la tension est proportionnelle à une valeur de la lumière en lux.
- Un capteur de température : Un TMP36, dont la tension est proportionnelle à la température en degrés Celsius.

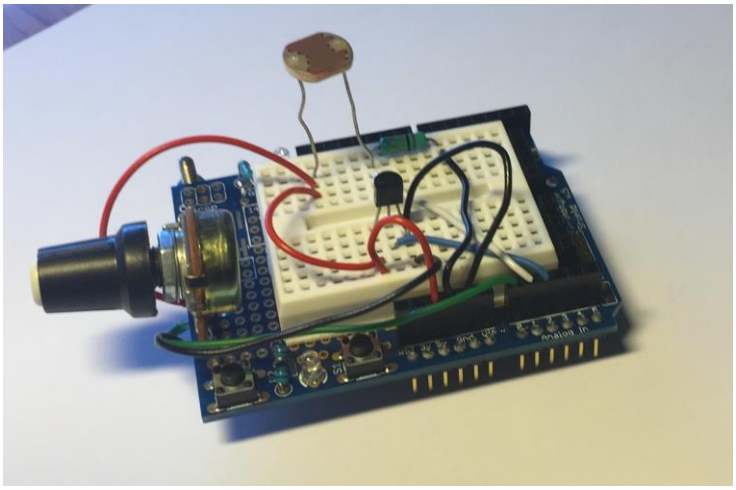


Figure 38: Carte prototype avec les capteurs

Pour réaliser l'envoi des données des capteurs par protocole Bus CAN, la carte Arduino ne possède pas le module nécessaire pour réaliser cette action. Nous avons donc utilisé un module CAN Shield Arduino qui est compatible avec notre carte et qui nous permet d'envoyer les valeurs de nos capteurs par Bus CAN :

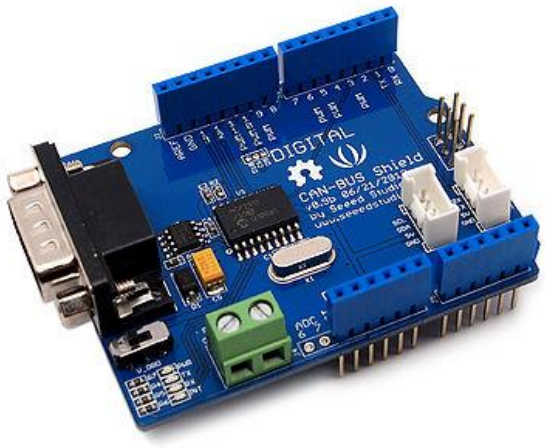


Figure 39: CAN Shield compatible Arduino Uno

Afin de visualiser les données des différents capteurs qui sont envoyées sur le Bus CAN, avons fait le choix d'ajouter un afficheur LCD qui est compatible avec la carte Arduino. Il permet de vérifier que les valeurs reçues par Bus CAN sur l'autre nœud sont correctes.

Figure 40: Afficheur LCD

L'ensemble des trois cartes ainsi que les différents capteurs sont rassemblés dans un boîtier :

Figure 41: Boîtier ensemble multi –capteurs

Le programme développé sur l'Arduino peut se résumer par cet algorithme simplifié :

```
Initialisation_Bus_CAN ();
Faire
        {
        Acquisition_des_capteurs () ;
        Envoi_données_capteurs () ;
        Affichage_LCD_données_capteurs () ;
        }
Tant Que (Toujours) ;
```

On vient acquérir l'état des capteurs, puis on envoie leurs données sur le Bus CAN en permanence à la vitesse de 1 Mbps et on affiche leurs données sur un écran LCD.

5.3 Tableau de bord

Le tableau de bord du véhicule est réalisé avec le logiciel Labview. Cet outil est le cœur d'une plate-forme de conception de systèmes de mesure et de contrôle, basée sur un environnement de développement graphique de National Instruments.

Figure 42: Logo du logiciel Labview

Labview est une plate-forme de programmation graphique qui va de la phase de conception jusqu'au test final. Cet outil est utilisé essentiellement pour la mesure par acquisition de données, pour le contrôle d'instruments et pour l'automatisme industriel.

Les programmes Labview sont appelés instruments virtuels (Vi). Leur apparence et leurs fonctionnements sont semblables à des instruments réels, tels que les oscilloscopes et les multimètres. Un VI, se compose de deux fenêtres:

- La face avant correspond à l'interface utilisateur du VI,
- Le diagramme représente le code, il utilise des représentations graphiques des fonctions afin de contrôler les objets de la face avant. La fenêtre du diagramme contient ce code source graphique.

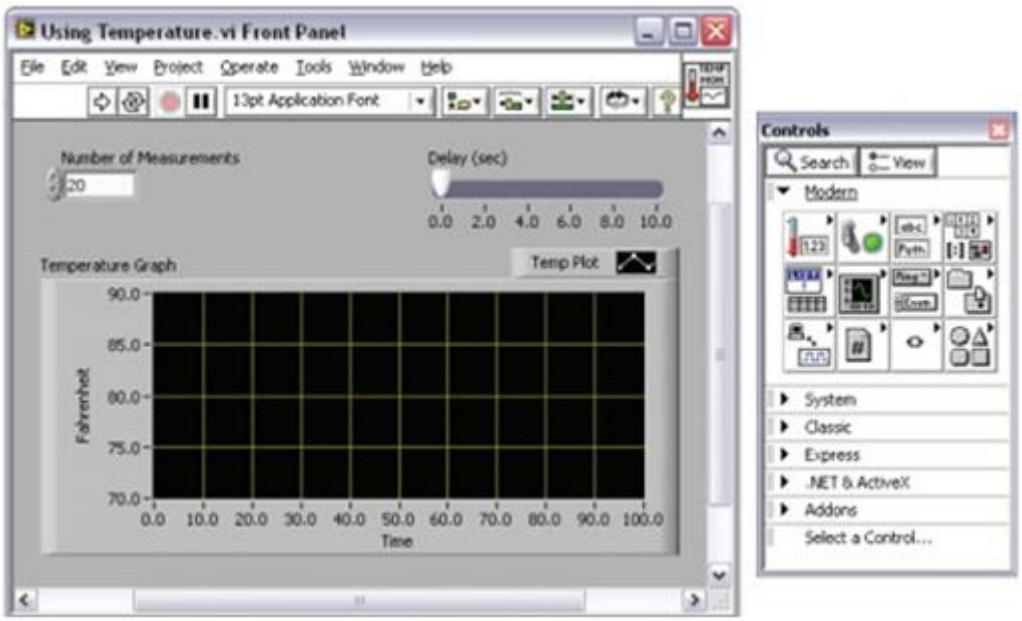


Figure 43: Fenêtre de la face avant

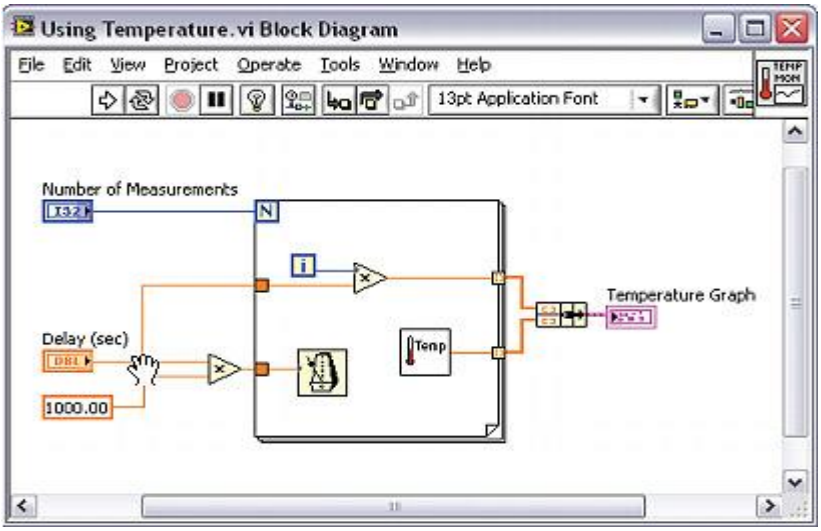


Figure 44: Fenêtre du Diagramme

Le CANUSB est une clé électronique qui se branche dans un Port USB de PC et qui fournit une connexion CAN instantanée. Afin de pouvoir utiliser la prise CANUSB avec Labview il faut au préalable télécharger plusieurs fichiers, qui vont dépendre du PC utilisé. Il faut télécharger un driver FTDI USB, et une bibliothèque CANUSB DLL sur le site suivant http://www.can232.com/ Sur ce site on peut aussi trouver quelques exemples de programmes avec plusieurs logiciels dont Labview.

Figure 45: Câble USB

Dans notre réalisation, Labview est un des nœuds du Bus CAN. Il permet d'établir une connexion avec le Bus CAN via un port USB. Notre programme reçoit des données des autres nœuds (PIC et Arduino) et les affiche sur la face avant. Comme décrit dans le cahier des charges, sur la face avant est visible les informations suivantes:

Informations	identifiant CAN
Vitesse véhicule	1
Luminosité extérieure véhicule	2
Température extérieure véhicule	3
Etat du moteur (allumé/éteint)	4

Figure 46: Identifications des trames

Ces différentes informations qui arrivent par le Bus CAN sont traitées avec des conditions sous Labview. Dans le tableau ci-dessus, un identifiant CAN, permettent de faire correspondre la valeur du capteur et l'identifiant utilisé sur la trame. L'état du moteur détermine si nous affichons les valeurs des capteurs sur le tableau de bord :

- Si le moteur est éteint, nous n'affichons aucune information,
- Si le contact est établi, l'état des phares ainsi que la température extérieure s'affichent,
- Si le moteur est démarré, nous affichons la vitesse du véhicule. Nous affichons toujours l'état des feux ainsi que la température.

Concernant l'allumage des feux, nous avons déterminé 3 états de luminosité qui ont pour conséquence de choisir les feux qui sont allumés :

- Pour un état « sombre », nous allumons les feux de route,
- Pour un état « Peu lumineux », nous allumons les feux de croisement,
- Pour un état « Lumineux », nous allumons les feux de position,
- Pour un état « Très lumineux », aucun feu n'est allumé.

Les résultats obtenus par notre programme seront visibles sur la fenêtre de la face avant de Labview.

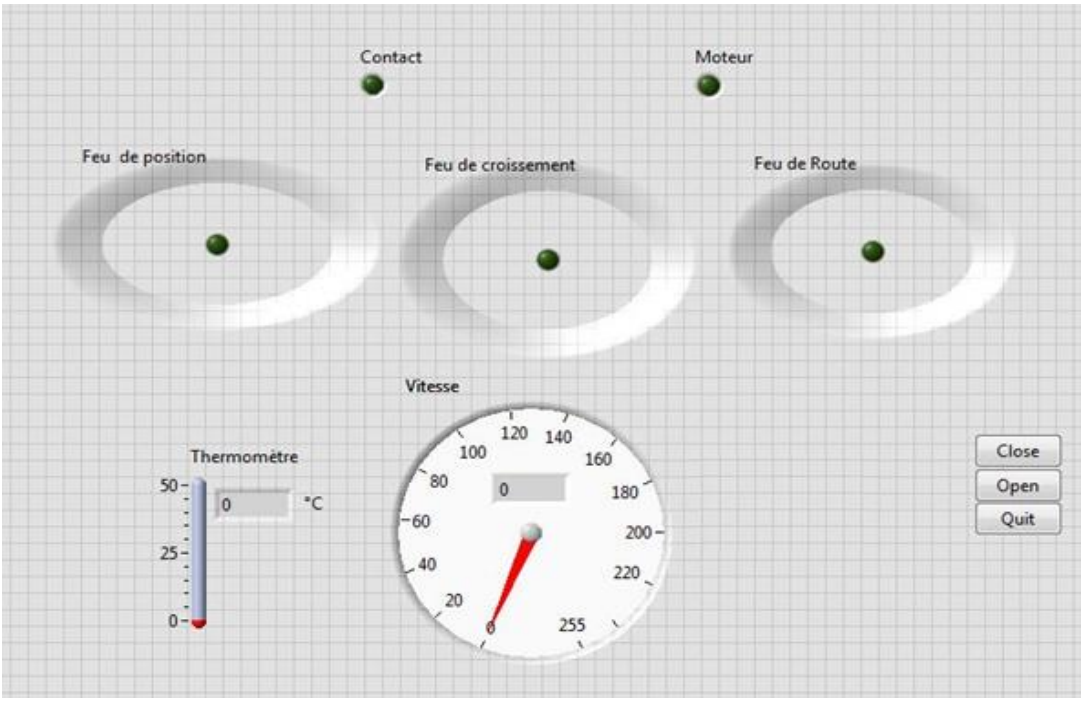


Figure 47: Face avant Labview (Tableau de bord véhicule)

Conclusion

L'objectif du projet de Science Physiques pour l'ingénieur est de mettre en œuvre nos compétences acquises durant notre formation par l'étude et la mise en œuvre d'une technologie : le Bus CAN. Ce projet se traduit par l'élaboration de ce mémoire, d'une réalisation technique et d'une soutenance orale. A travers ce document, dans une première partie, nous vous avons présenté le Bus CAN de façon générale : les définitions à connaitre, un bref historique, les normes, les domaines d'applications, les intérêts de ce bus de terrain et sa situation par rapport au modèle OSI. Ensuite, nous avons procédé à une analyse technique approfondie du Bus CAN, en décrivant les caractéristiques électriques, les différents types de trames de communication CAN, la façon dont sont gérées les erreurs et les modes de fonctionnements. Enfin, la réalisation est exposée par une présentation générale, une description des outils de gestion de projet que nous avons mis en œuvre et une description des aspects techniques.

Pour produire cet écrit, nous avons utilisé des méthodes de recherche et de gestion de projet que nous avons pu apprendre au CESI et en entreprise. En effet, après avoir défini le périmètre de l'étude et le plan associé, nous avons découpé le travail et alloué un responsable unique. Nous avons aussi élaboré un planning avec des livrables pour chaque jalon. Ensuite, chaque responsable a pu commencer ses recherches bibliographiques et webographiques, que nous mettions en commun régulièrement pendant des réunions. Ce même principe a été utilisé pour développer notre réalisation technique.

Cette analyse nous permet de comprendre pourquoi le Bus CAN s'est imposé dans le domaine automobile et pourquoi il se démocratise de plus en plus dans le domaine aéronautique : son faible coût de mise en place grâce à l'utilisation d'une paire filaire, sa robustesse de fonctionnement et son système de gestion des priorités en fait un réseau de choix dans les contextes sévères du transport automobile et aérien. Toutefois, dans le domaine des bus de terrain, le Bus CAN standard se doit d'évoluer car sa relative faible vitesse (1Mbps) et l'augmentation des débits demandés par les systèmes de divertissements embarqués mènent les entreprises et les laboratoires de recherches à développer de nouveaux protocoles plus rapides et robustes comme le CAN FD, le FlexRay et l'Ethernet.

Sources

[1] CAN Specification Version 2.0 – 1991 – Robert Bosch Gmbh

[2] Réseaux de communication pour systèmes embarqués - 2e éd. – 2014 – Dominique Paret, Hassina Rebaine

[3] Le Bus CAN – Patrice Kadionik – [En Ligne] :

https://kadionik.vvv.enseirb-matmeca.fr/enseirb/old/RE304/canbus_enseirb.pdf

[4] Seminaire Industriel CAN – CFAI Languedoc Roussillon – [En Ligne] :

http://www.cfai-languedocroussillon.com/telechargement/Seminaire_Bus_Industriel_CAN.pdf

[5] CAN Overview – National Instruments – [En Ligne] :

http://www.ni.com/white-paper/2732/en/#toc3

[6] Introduction aux Réseaux Informatiques – Aoun, Kacimi, Torguet, Truillet (Université Paul Sabatier, IRIT) – Cours CESI

[7] Commission générale de terminologie et de néologie - NOR : CTNX0306624X - JO du 14-06-2003, pp. 10047-10050

[8] The history of standardization and CAN in a nutshell – CiA (Can in Automation) – [En Ligne] : http://www.can-cia.org/fileadmin/cia/files/Newsletter_02-14/2-14_p3_the-history-of-standardization-and-can-in-a-nutshell.pdf

[9] Mise en œuvre d'une communication par Bus CAN - CHARLES LERY – [En Ligne] http://dirac.epucfe.eu/projets/wakka.php?wiki=P08AB12index/download&file=note_application_com_pic_mppt.pdf

[10] TP PIC Photovoltaïque – Thierry Perisse, Vincent Boitier (Université Paul Sabatier) – [En Ligne] http://thierryperisse.free.fr/documents/tppic/TP_PIC_PV_v2014-15.pdf

[11] Réseau CAN – Z. Mammeri (Université Paul Sabatier, IRIT) – [En Ligne] : http://www.irit.fr/~Zoubir.Mammeri/Cours/M2PRLI/Chap2IntroCAN.pdf

[12] AN754 Understanding Microchip's CAN Module Bit Timing – Société Microchip – [En Ligne] : http://ww1.microchip.com/downloads/en/AppNotes/00754.pdf

[13] PIC18F2585/2680/4585/4680 Data Sheet – Société Microchip – [En Ligne] : http://ww1.microchip.com/downloads/en/DeviceDoc/39625c.pdf

[14] A la découverte du Bus CAN – BDE ENSEEIHT – [En Ligne] http://www.bde.enseeiht.fr/clubs/robot/node/44

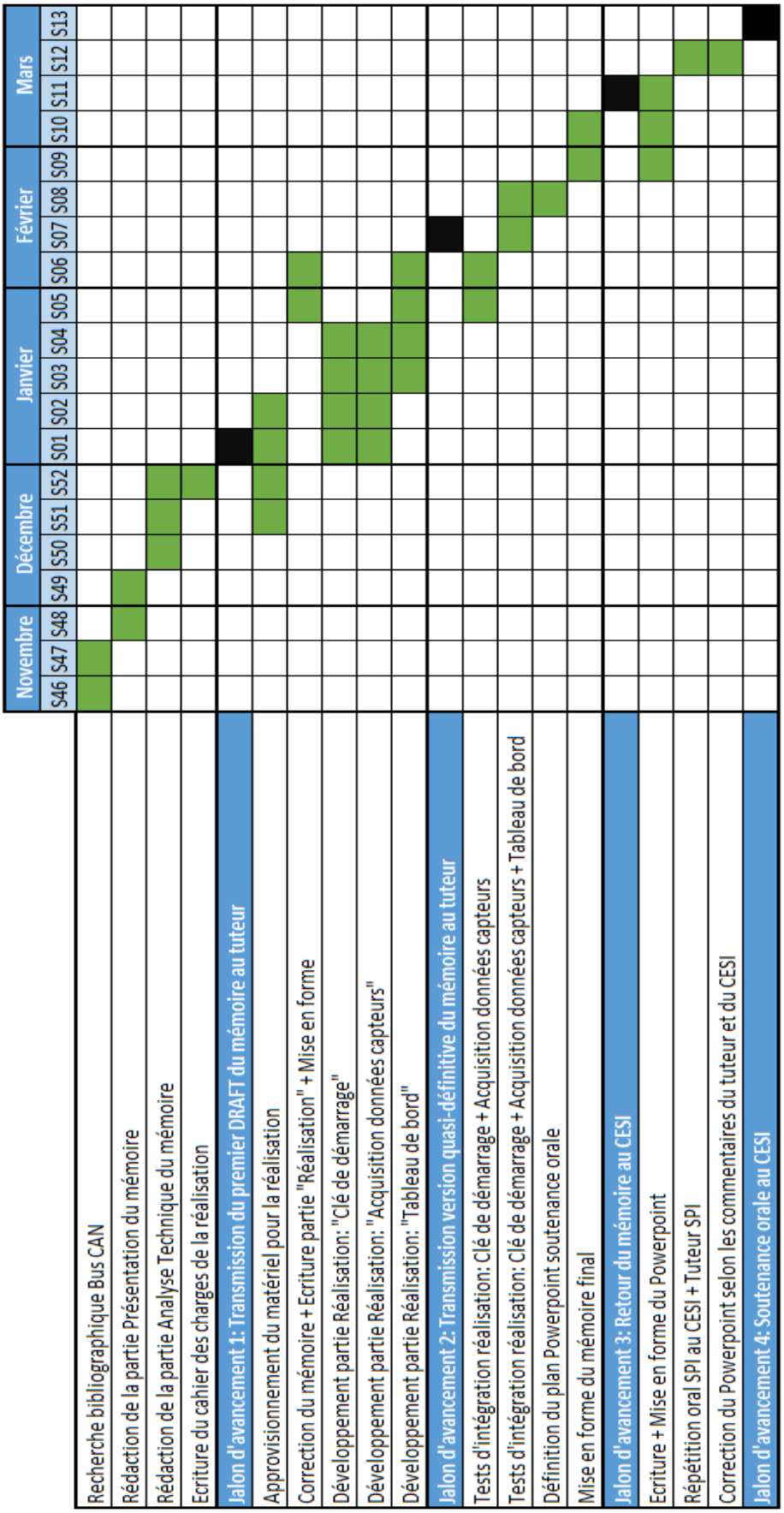
[15] Réalisation de systèmes de capteurs, une approche pratique – German Fabregat (Université Jaume I Castellon, Espagne) – [En Ligne] :

http://pagesperso.univ-brest.fr/~bounceur/ecole2013/pdf_presentations/cours_fabregat.pdf

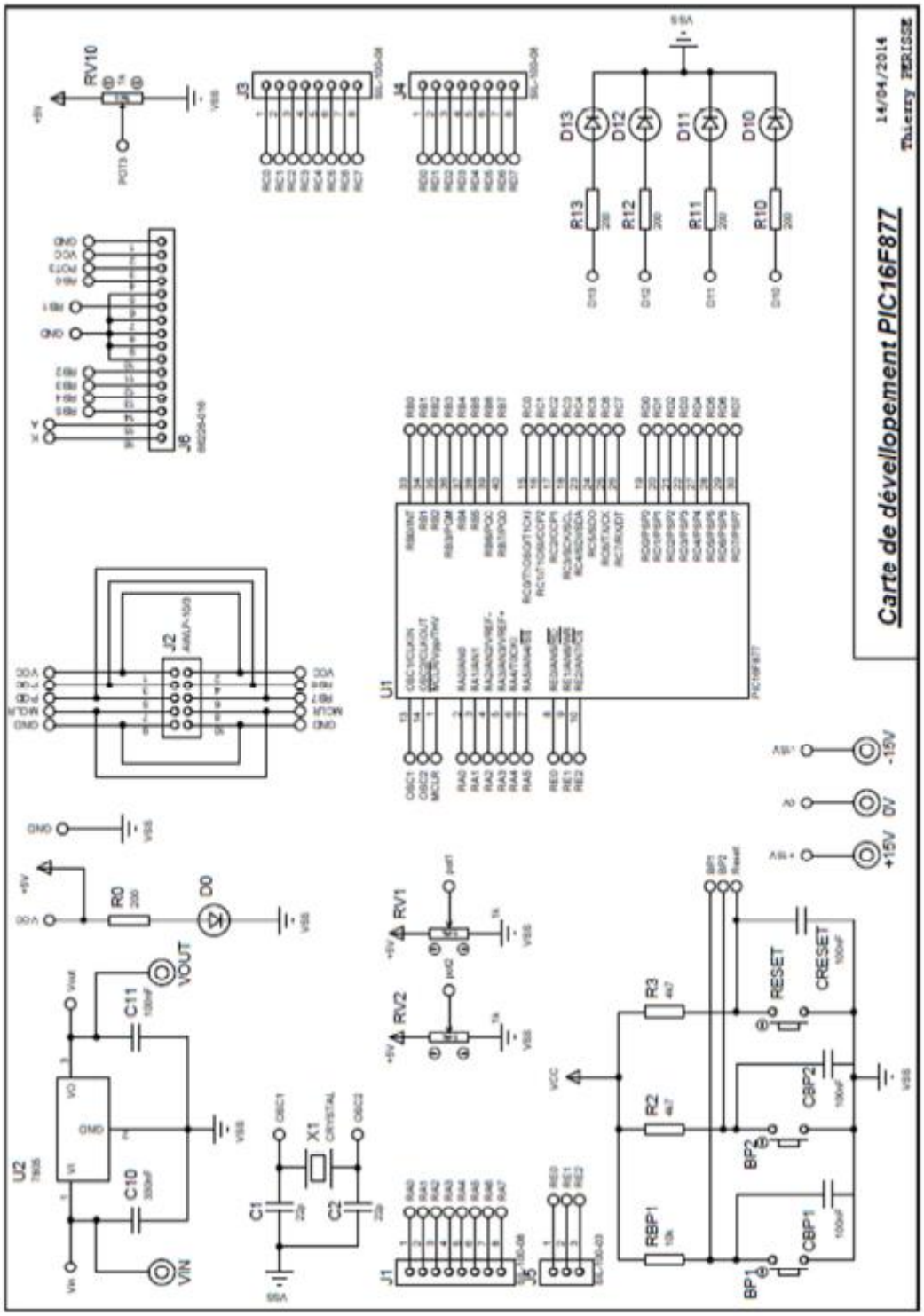
[16] Advanced PIC Microcontroller Projects in C, 1st Edition – Dogan Ibrahim

[17] Controller Area Network Physical Layer Requirements - Steve Corrigan (Texas Instruments)

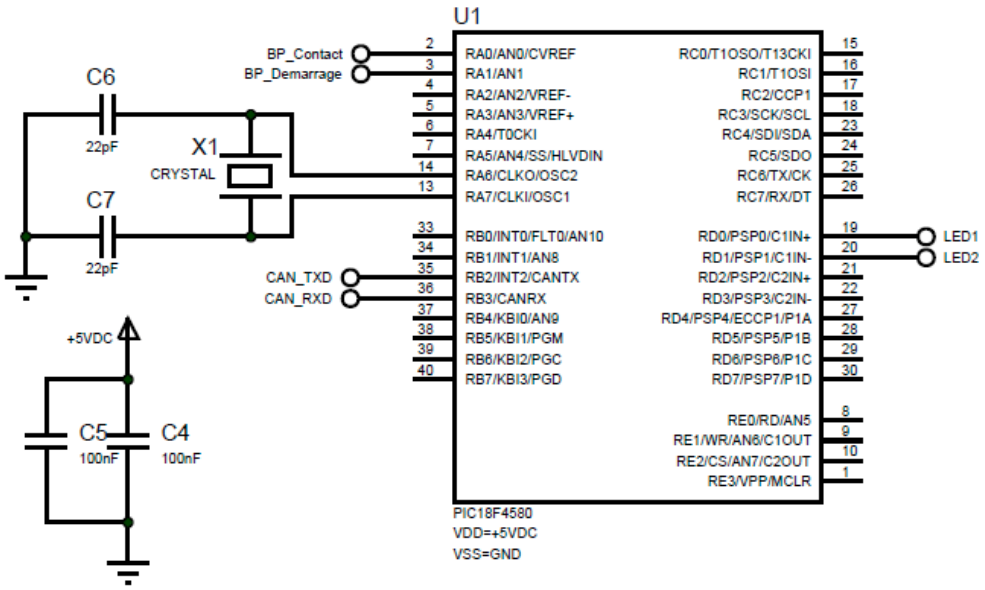
ANNEXE 1 : Planning du projet.



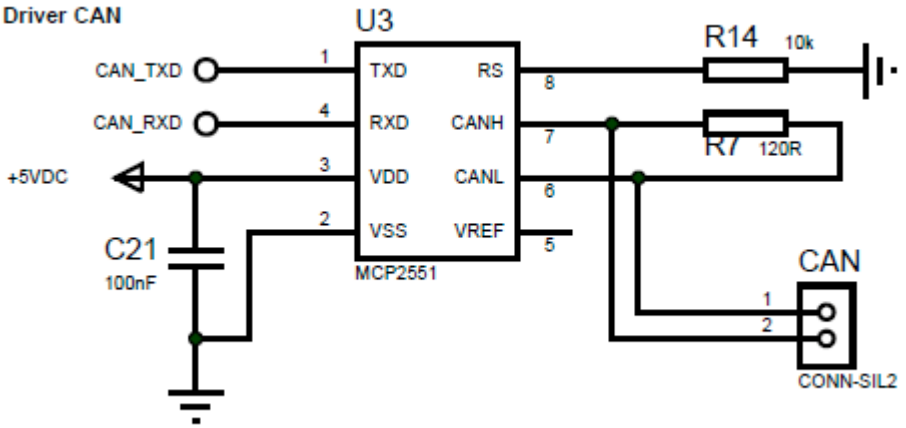
ANNEXE 2 : Schéma de câblage carte PIC Université Paul Sabatier.



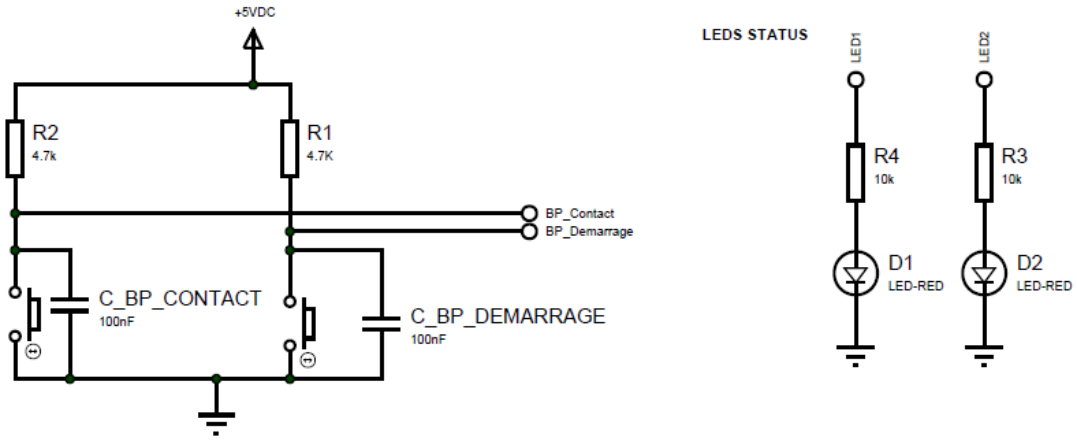
ANNEXE 3 : Schéma de câblage PIC18F4580.



ANNEXE 4 : Schéma de câblage Transceiver MCP2551.



ANNEXE 5 : Schéma de câblage boutons poussoir et LEDS.



ANNEXE 6 : Code source PIC18F4580

```
1: //Variables etat contact et demarrage
2: bit etat_contact;
3: bit etat_demarrage;
4:
5: void lecture_entree(void)
6: {
7:  // Utilisation de la routine Button de mikroC permettant de gerer les rebonds su
   r les boutons poussoirs
8:   if(Button(&PORTA, 0, 10, 0)==255)    //appui sur bouton contact
9:   {
10:     etat_contact= ~etat_contact;
11:  }
12:
13:  if(Button(&PORTA, 1, 10, 0)==255)  //appui bouton demarrage
14:  {
15:     etat_demarrage= ~etat_demarrage;
16:  }
17: }
18:
19: char etat_neiman(void)
20: {
21:  char etat;
22:  lecture_entree();        // appel de la procedure lecture_entree permettant de re
   ecuperer la position de la clé
23:  if(etat_contact==0 && etat_demarrage==0)       // si on est dans la position mote
   eur coupé
24:  {
25:       etat=1; //moteur arreté
26:       LATD.F0=0;  // eteint la LED D0
27:       LATD.F1=0;  // eteint la LED D1
28:  }
29:  else if(etat_contact==1 && etat_demarrage==0)     // sinon si le contact est mis
30:  {
31:      etat=2; //contact mis
32:      LATD.F0=1;      // allume la LED D0
33:      LATD.F1=0;      // eteint la LED D1
34:  }
35:  else if(etat_contact==1 && etat_demarrage==1)     // sinon si le moteur est demarr
   ré
36:  {
37:      etat=3; //moteur demarré
38:      LATD.F0=1  // allume la LED D0
39:      LATD.F1=1; // allume la LED D1
40:  }
41:  return(etat);
42: }
43: void main()
44: {
45:   //CAN
46:   unsigned short init_flag,send_flag,dt,len,read_flag;
47:   char SJW,BRP,Phase_Seg1,Phase_Seg2,Prog_Seg,txt[4];
48:   char donnees[8];
49:   long id,mask;
50:   //PORTB = 0;                                          // Initialisation PO
   ORTB à 0
51:   TRISB = 0x08;                                       //Initialisation du
    port de sortie du CAN
52:   //Parametres de configuration du Baudrate (vitesse du bus = 1Mbps)
53:   SJW=1;
54:   BRP=1;
55:   Phase_Seg1=4;
56:   Phase_Seg2=3;
```

```
57:   Prog_Seg=2;
58:
59:   init_flag = 0;
60:   send_flag = 0;                                        // initialisation des flags
    s à 0
61:   read_flag  = 0;
62:   // initialisation des flags aux valeurs conseillées par le constructeur
63:   init_flag = _CAN_CONFIG_SAMPLE_THRICE &
64:                   _CAN_CONFIG_PHSEG2_PRG_ON &
65:                   _CAN_CONFIG_XTD_MSG &
66:                   _CAN_CONFIG_DBL_BUFFER_ON &
67:                   _CAN_CONFIG_VALID_XTD_MSG &
68:                   _CAN_CONFIG_LINE_FILTER_OFF;
69:
70:   send_flag = _CAN_TX_PRIORITY_0 &
71:                   _CAN_TX_XTD_FRAME &
72:                   _CAN_TX_NO_RTR_FRAME;
73:
74:   //Initialisation du CAN
75:   CANInitialize(SJW,BRP,Phase_Seg1,Phase_Seg2,Prog_Seg,init_flag);  // routine mik
    kroC permettant d'initialiser le CAN, elle annule toute les transmissions en att
    nte(voir annexe)
76:   //Mode de configuration du CAN
77:   CANSetOperationMode(_CAN_MODE_CONFIG,0xFF); // routine mikroC permettant de mett
    tre le CAN en mode requête, la valeur 0xFF permet de ne pas mettre le CAN en mod
     requête tant que cela n'est pas demandé (voir annexe)
78:   mask=-1;
79:   CANSetMask(_CAN_MASK_B1,mask,_CAN_CONFIG_XTD_MSG);      // routine mikroC permett
    tant d'identifier les messages avec un masque    (voir annexe)
80:   CANSetMask(_CAN_MASK_B2,mask,_CAN_CONFIG_XTD_MSG);
81:   CANSetFilter(_CAN_FILTER_B2_F3,3, _CAN_CONFIG_XTD_MSG );  // routine mikroC perm
    mettant d'appliquer un  filtre sur les differents messages (voir annexe)
82:   CANSetOperationMode(_CAN_MODE_NORMAL,0xFF);
83:   TRISA0_bit = 0xFF;                                    // Mise de la pin RB0 en en
    ntrée
84:   ADCON1=0x0F;                                          // Configuration des entrée
    es comme des entrées digitales
85:   TRISD = 0x00;                                         // Mise du PORTD en sortie
86:   LATD = 0x00;                                          // Initialisation de la val
    leur du PORTD
87:   id=4;                                                 // ID sur le bus CAN
88:   //Initialisations des variables
89:   etat_contact=0;                                       //contact coupé
90:   etat_demarrage=0;
91:
92:   do {
93:         donnees[0]= etat_neiman();
94:         donnees[1]= 0;
95:         donnees[2]= 0;
96:         donnees[3]= 0;
97:         donnees[4]= 0;
98:         donnees[5]= 0;
99:         donnees[6]= 0;
100:        donnees[7]= 0;
101:
102:        CANWrite(id,donnees,1,send_flag);               // routine mikroC permettan
    nt d'envoyer le message sur le bus CAN    (voir annexe)
103:        delay_ms(300);
104:     }while(1);
105: }
```

ANNEXE 7 : Routine PIC ADC0N1

PIC18F2480/2580/4480/4580

REGISTER 19-2: ADCON1: A/D CONTROL REGISTER 1

U-0	U-0	R/W-0	R/W-0	R/W-0(1)	R/W-q(1)	R/W-q(1)	R/W-q(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented:** Read as '0'

bit 5 **VCFG1:** Voltage Reference Configuration bit (VREF- source)
1 = VREF- (AN2)
0 = AVSS

bit 4 **VCFG0:** Voltage Reference Configuration bit (VREF+ source)
1 = VREF+ (AN3)
0 = AVDD

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A
0011	A	A	A	A	A	A	A	A	A	A	A
0100	A	A	A	A	A	A	A	A	A	A	A
0101	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Note 1: The POR value of the PCFG bits depends on the value of the PBADEN bit in Configuration Register 3H. When PBADEN = 1, PCFG<3:0> = 0000; when PBADEN = 0, PCFG<3:0> = 0111.

2: AN5 through AN7 are available only on PIC18F4X80 devices.

ANNEXE 8: Routine PIC CANinitialize

CANInitialize

Prototype	`void CANInitialize(char SJW, char BRP, char PHSEG1, char PHSEG2, char PROPSEG, char CAN_CONFIG_FLAGS);`
Returns	Nothing.

Description	Initializes CAN. All pending transmissions are aborted. Sets all mask registers to 0 to allow all messages.
	Filter registers are set according to flag value:
	`if (CAN_CONFIG_FLAGS & _CAN_CONFIG_VALID_XTD_MSG != 0)`
	`  // Set all filters to XTD_MSG`
	`else if (config & _CAN_CONFIG_VALID_STD_MSG != 0)`
	`  // Set all filters to STD_MSG`
	`else`
	`  // Set half of the filters to STD, and the rest to XTD_MSG.`
	Parameters:
	▪ `SJW` as defined in datasheet (1-4)
	▪ `BRP` as defined in datasheet (1-64)
	▪ `PHSEG1` as defined in datasheet (1-8)
	▪ `PHSEG2` as defined in datasheet (1-8)
	▪ `PROPSEG` as defined in datasheet (1-8)
	▪ `CAN_CONFIG_FLAGS` is formed from predefined constants (see CAN constants)

Requires	CAN must be in Config mode; otherwise the function will be ignored. Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.

Example	`init = _CAN_CONFIG_SAMPLE_THRICE &`
	`  _CAN_CONFIG_PHSEG2_PRG_ON &`
	`  _CAN_CONFIG_STD_MSG  &`
	`  _CAN_CONFIG_DBL_BUFFER_ON &`
	`  _CAN_CONFIG_VALID_XTD_MSG &`
	`  _CAN_CONFIG_LINE_FILTER_OFF;`
	`...`
	`CANInitialize(1, 1, 3, 3, 1, init);  // initialize CAN`

ANNEXE 9: Routine PIC CANSetFilter

CANSetFilter

Prototype	`void CANSetFilter(char CAN_FILTER, long value, char CAN_CONFIG_FLAGS);`
Returns	Nothing.
Description	Function sets message filter. Given `value` is bit adjusted to appropriate buffer mask registers. Parameters: ▪ `CAN_FILTER` is one of predefined constant values (see CAN constants) ▪ `value` is the filter register value ▪ `CAN_CONFIG_FLAGS` selects type of message to filter, either `_CAN_CONFIG_XTD_MSG` or `_CAN_CONFIG_STD_MSG`
Requires	CAN must be in Config mode; otherwise the function will be ignored. Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.
Example	`// Set id of filter B1_F1 to 3:` `CANSetFilter(_CAN_FILTER_B1_F1, 3, _CAN_CONFIG_XTD_MSG);`

ANNEXE 10: Routine PIC CANSetMask

CANSetMask

Prototype	`void CANSetMask(char CAN_MASK, long value, char CAN_CONFIG_FLAGS);`
Returns	Nothing.
Description	Function sets mask for advanced filtering of messages. Given `value` is bit adjusted to appropriate buffer mask registers. Parameters: ▪ `CAN_MASK` is one of predefined constant values (see CAN constants) ▪ `value` is the mask register value ▪ `CAN_CONFIG_FLAGS` selects type of message to filter, either `_CAN_CONFIG_XTD_MSG` or `_CAN_CONFIG_STD_MSG`
Requires	CAN must be in Config mode; otherwise the function will be ignored. Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.

Example	`// Set all mask bits to 1, i.e. all filtered bits are relevant:`
	`CANSetMask(_CAN_MASK_B1, -1, _CAN_CONFIG_XTD_MSG);`
	
	`// Note that -1 is just a cheaper way to write 0xFFFFFFFF.`
	`Complement will do the trick and fill it up with ones.`

ANNEXE 11 : Routine PIC CANSetOperationMode

CANSetOperationMode

Prototype	`void CANSetOperationMode(unsigned short mode, unsigned short wait_flag);`
Returns	Nothing.
Description	Sets CAN to requested mode, i.e. copies `mode` to CANSTAT. Parameter `mode` needs to be one of `CAN_OP_MODE` constants (see CAN constants). Parameter `wait_flag` needs to be either 0 or 0xFF: ▪ If set to 0xFF, this is a blocking call – the function won't "return" until the requested mode is set. ▪ If 0, this is a non-blocking call. It does not verify if CAN module is switched to requested mode or not. Caller must use `CANGetOperationMode` to verify correct operation mode before performing mode specific operation.
Requires	Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.
Example	`CANSetOperationMode(_CAN_MODE_CONFIG, 0xFF);`

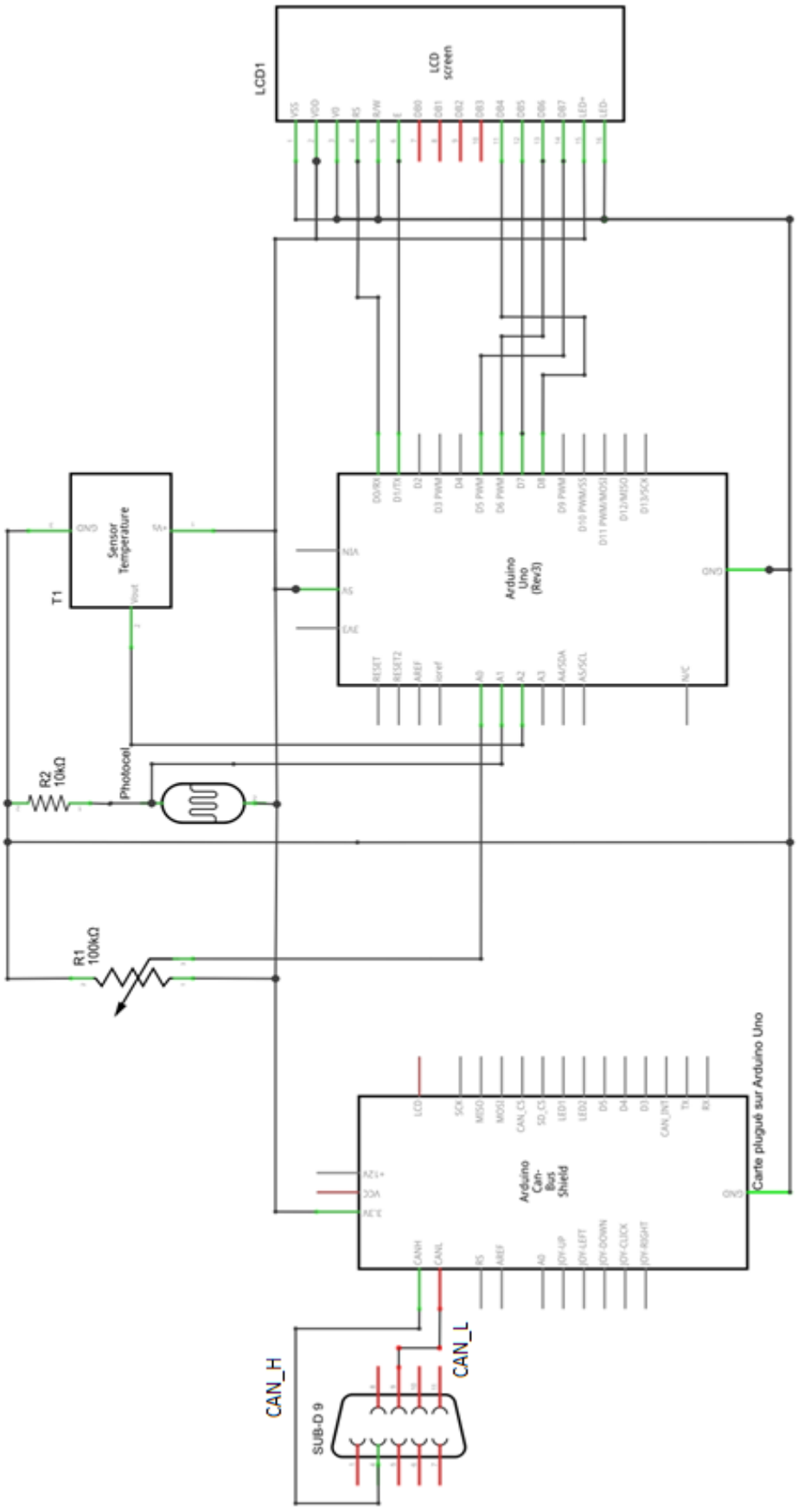
ANNEXE 12 : Routine PIC CANWrite

CANWrite

Prototype	`unsigned short CANWrite(long id, char *data, char datalen, char CAN_TX_MSG_FLAGS);`
Returns	Returns zero if message cannot be queued (buffer full).
Description	If at least one empty transmit buffer is found, function sends message on queue for transmission. If buffer is full, function returns 0. Parameters: ▪ `id` is CAN message identifier. Only 11 or 29 bits may be used depending on message type (standard or extended) ▪ `data` is array of bytes up to 8 bytes in length ▪ `datalen` is data length from 1–8 ▪ `CAN_TX_MSG_FLAGS` is value formed from constants (see CAN constants)
Requires	CAN must be in Normal mode. Microcontroller must be connected to CAN transceiver (MCP2551 or similar) which is connected to CAN bus.

Example	`char tx, data;`
	`long id;`
	
	`// ...`
	`tx = _CAN_TX_PRIORITY_0 &`
	`_CAN_TX_XTD_FRAME;`
	`// ...`
	`CANWrite(id, data, 2, tx);`

ANNEXE 13 : Câblage ensemble multi-capteurs



ANNEXE 14 : Code Source Arduino

```
#include <mcp_can.h>
#include <SPI.h>
#include <LiquidCrystal.h>

LiquidCrystal lcd(0, 1, 8, 7, 6, 5);  // Liaison LCD avec Arduino
MCP_CAN CAN(9);

//----------------------------------------------------------------------
// ACQUISITION DES CAPTEURS & TRAITEMENT DES DONNEES
//----------------------------------------------------------------------
int acquisitionVitesse(void)
{
  int vitessePin = A0;
  int Vitesse = analogRead(vitessePin);
  Vitesse = Vitesse/4;
  return Vitesse;
}

int acquisitionLuminosite(void)
{
  int photocellPin = A1;
  int Luminosite = analogRead(photocellPin);
  Luminosite = Luminosite/4;
  return Luminosite;
}

int acquisitionTemperature(void)
{
  int temperaturePin = A2;
  int TemperatureReading;
  TemperatureReading = analogRead(temperaturePin);
  int OutputValue = (((TemperatureReading))*5000.0)/1023.0;
  int Temperature = (OutputValue - 500.0) /10.0;
  return Temperature;
}


//-------------------------------------
// ENVOIE DES TRAMES SUR LE BUS CAN
//
//    ID 0x01: Vitesse
//    ID 0x02: Luminosite
//    ID 0x03: Temperature
//-------------------------------------


void envoieVitesse(int Vitesse)
{
  unsigned char test=Vitesse;
  CAN.sendMsgBuf(0x01, 0, 8, &test);
}

void envoieLuminosite(int Luminosite)
{
  unsigned char test=Luminosite;
  CAN.sendMsgBuf(0x02, 0, 8, &test);
}

void envoiTemperature(int Temperature)
{
  unsigned char test=Temperature;
  CAN.sendMsgBuf(0x03, 0, 8, &test);
}
```

Fonction *CAN.sendMsgBuf(ID, Ext, Len, Data)*

- *ID : identifiant de la trame*
- *Ext : 0-> Standard trame & 1-> Extended trame*
- *Len : taille*
- *Data : Données*

```
void affichageLCD(int Temperature, int Vitesse, int Luminosite)
{
  lcd.setCursor(0, 0);
    // Affichage température sur le LCD
    lcd.print(Temperature);
    lcd.write(4);
    lcd.print("C  ");

    // Affichage vitesse sur le LCD
  lcd.setCursor(7, 0);
    lcd.print(Vitesse);
    lcd.print( "KM/H   ");

    // Affichage luminosité sur le LCD
  lcd.setCursor(0, 1);
  if (Luminosite < 63)
  {
    lcd.print(" Sombre         ");
  }

  else if (Luminosite < 127)
  {
    lcd.print(" Peu Lumineux   ");
  }

  else if (Luminosite < 189)
  {
    lcd.print(" Lumineux       ");
  }

  else
  {
    lcd.print(" Tres luminineux");
  }
 delay(300);
}

void setup(void)
{
  lcd.begin(16, 2);
  lcd.print(" SPI - BUS CAN  ");
  delay(2000);
  lcd.clear();

  lcd.createChar( 4, degrees );
  lcd.write(4);
  lcd.clear();

  START_INIT:
    // Initialisation du BUS CAN
    if(CAN_OK == CAN.begin(CAN_1000KBPS))
    {

        lcd.print("CAN BUS INIT OK");  // Initialisation du BUS CAN OK
        delay(2000);
        lcd.clear();
    }
    else
    {

        lcd.print("CAN BUS INIT FAIL"); // Echec initialisation du BUS CAN
        delay(1000);
        lcd.clear();
        goto START_INIT;
    }
}
```

```
void loop(void)
{
  //Declaration des variables
  int Vitesse;
  int Luminosite;
  int Temperature;

  //Lecture des capteurs
  Vitesse = acquisitionVitesse();
  Temperature = acquisitionTemperature();
  Luminosite = acquisitionLuminosite();

  //Envoi des données sur le bus CAN
  envoiTemperature(Temperature);
  delay(1);
  envoieVitesse(Vitesse);
  delay(1);
  envoieLuminosite(Luminosite);
  delay(1);

  // Affichage données sur le LCD
  affichageLCD(Temperature, Vitesse, Luminosite);
  delay(300);
}
```

ANNEXE 15 : Diagramme Labview