Université Mohamed Khider de Biskra
Faculté des Sciences et de la Technologie
Département de Génie Electrique

MÉMOIRE DE MASTER

Sciences et Technologies
Filière
Spécialité

Réf. : Entrez la référence du document

Présenté et soutenu par :
Guenane salah eddine

Le : mercredi 30 septembre 2020

Le Protocol CAN POUR UN CAN pour un contrôle et un diagnostique sécurisé des moteurs de voitures

---

Jury :

Dr.	Maghrbi Mohamed el Arbi	MCA	Université de Biskra	Président
Dr.	Hmaizia Zahra	MCA	Université de Biskra	Examinateur
Dr.	Guesbaya Tahar	MCA	Université de Biskra	Encadreur

2020 - 2019Année universitaire :

Dédicaces

Au nom d'Allah, le Clément et le Miséricordieux. Tout d'abord je tiens à remercier le Tout

Puissant de m'avoir donné le courage et la patience pour arriver à ce stade afin de

réaliser ce travail que je dédie à:

A mes chers parents, et spécialement ma plus belle étoile qui puisse exister dans l'univers

ma chère mère

A mon Plus grande inspiration mon père

A ma famille, A mes amis sans exception, et toutes personnes que j'aiconnues.

A toute la promotion (2019,2020).

Remerciements

Nous remercions dieu le clément de nous avoir éclairé le chemin du
savoir afin de terminer ce travail.
Nous tenons à remercier: Mr Guesbaya Tahar pour son suivi et ses
orientations éclairées.
Nous remercions le président et les membres du jury de nous avoir
accordé l'honneur d'examiner et de valoriser notre travail.
Un grand merci à nos chers parents, frères, sœur, et amis pour leurs
soutiens et présences.

Sommaire

Chapitre II Présentation générale du Bus CAN

Chapter III Applications Sur Bus CAN

Liste des Figures

LESTE DU TABLEAUX

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
PLS	Physical Signalling
MA P	Physical Medium Attachment
REC	Receive Error Counter
RTR	Remote Transmission Request
RPM	Revolution (ou rotation)per minute
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

Introduction Générale

Introduction générale

Depuis la création de la première automobile fin XIXème siècle l'industrie automobile a évolué rapidement et pour cause plusieurs facteurs :

- Progrès technologiques.
- Pression des contraintes réglementaires.
- Pression des attentes clients : individuelles et collectives.

L'avènement de l'électronique, l'informatique embarquée et l'automatique moderne a dans ce contexte un rôle majeur à jouer. Il permet de remplacer les principaux composants mécaniques et hydrauliques (direction, freinage, suspension…). Tous les modules de la voiture sont devenus plus intelligents, le couplage des fonctions par l'électronique devient alors possible, la voiture devient communicante et les services associés se développent. En 1985, une voiture embarquait un calculateur d'injection réalisant 10 fonctions. Aujourd'hui, une voiture embarque 25 calculateurs électroniques sur lesquels se répartissent 90 fonctions. Ces chiffres parlent d'eux-mêmes.

L'électronique est devenue incontournable dans l'automobile. Impossible de s'en passer. Elle est d'ailleurs la principale source d'innovations. 80 à 90 % des innovations dans une voiture viennent de l'électronique. Alors que les calculateurs se sont multipliés à bord, leurs fonctionnalités et leurs interactions sont devenues plus complexes. Aussi, au fil des ans, les constructeurs se sont dotés de compétences techniques et d'outils de test de plus en plus spécifiques pour valider le fonctionnement des systèmes électroniques et électrotechniques embarqués.

De plus, il est actuellement difficile de considérer une fonction indépendamment des autres ; par exemple, la vitesse véhicule qui est élaborée notamment par le système de contrôle du moteur est nécessaire à la réalisation d'autres fonctions comme le contrôle de la suspension. C'est ainsi que, actuellement, sur un véhicule il peut y avoir jusqu'à 2500 informations, ou signaux, échangés entre des fonctions réparties sur environ 70 ECUs. Dés lors, pour des raisons de coût, de poids d'encombrement, de complexité de câblage, il est devenu rapidement impossible de supporter ces échanges par des connexions point à point et l'utilisation de réseaux locaux embarqués est apparue comme une solution incontournable qui a donné naissance au multiplexage de données (CAN et VAN) et par la suite, le LIN a été introduit pour encore simplifier l'échange d'informations entre les différents ECUs**.**

Chapitre I

Les systèmes embarqués dans l’automobile

I-1.Introduction

Aujourd'hui, un véhicule contient une grande quantité d'électronique et d'informatique : on retrouve plus de 100 capteurs, 30 à 50 calculateurs selon le type de véhicule et parfois près d'un million de lignes de codes pour les véhicules de dernière génération. Cette évolution s'explique par les demandes exigeantes des consommateurs et l'envie de différenciation des concurrents sur marché de l'automobile. S'ajoute à cela les contraintes économiques et écologiques où l'électronique embarqué répond à ces nouvelles attentes. De nouvelles fonctionnalités impliquent parfois une intégration électronique et informatique par le biais de systèmes embarqués. Voici une représentation des systèmes intégrés d'un véhicule moderne. [1]

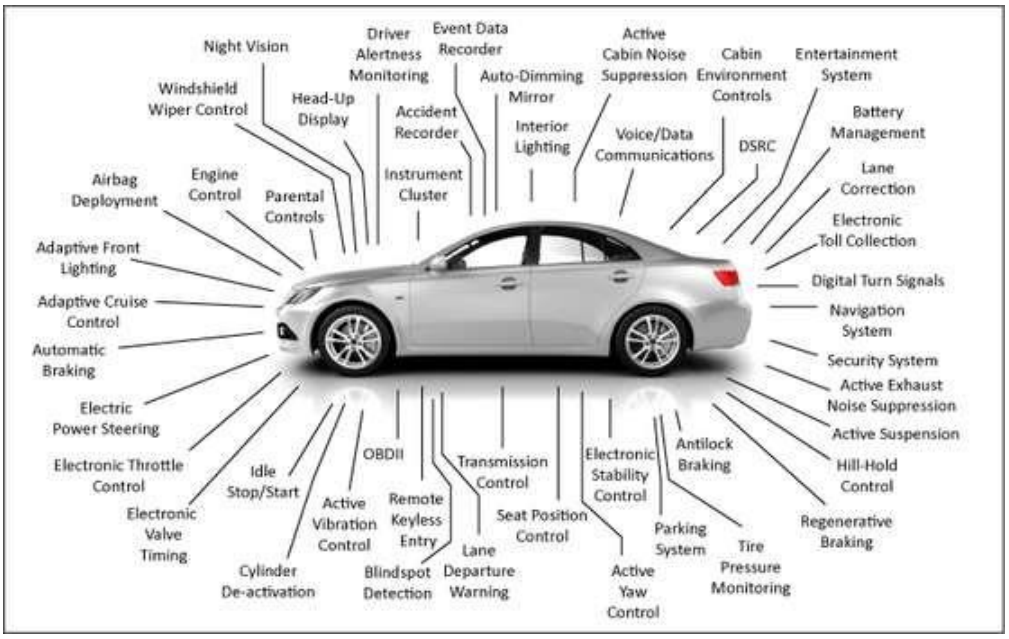


Figure I-1 : Fonctions d'un véhicule moderne

Dans un système automobile, on trouve plusieurs sous-systèmes. On distingue ici 5 types de tels sous-systèmes automobiles :

I-2.Les systèmes de châssis (Chassis Domain)

qui font partie des systèmes de sécurité active du véhicule, comprennent le contrôle de la dynamique du véhicule (VDC), également appelé programme de stabilité électronique (ESP). Les VDC/ESP sont con-çus pour assister le conducteur dans les situations de survirage, de sous-pilotage et de capotage. Le système de freinage antiblocage (ABS) empêche les roues de se bloquer en cas de freinage intensif. Cela aide le conducteur à maintenir sa capacité de

direction et à éviter de déraper pendant la pause. Tous ces systèmes nécessitent un contrôle en retour. [2]

I-2-1.Les systèmes d'airbag

font partie des systèmes de sécurité passive du véhicule et contrôlent le fonctionnement des sacs gonflables du véhicule. En règle générale, un véhicule contient plusieurs airbags. Ces sacs gonflables sont connectés à des capteurs qui détectent des situations anormales, par exemple une accélération ou une décélération soudaine du véhicule. Une fois qu'une situation anormale est détectée, les sacs gonflables appropriés sont gonflés en environ une demi-milliseconde après la détection d'un accident.

1-2-2.Le groupe motopropulseur (Power train Domain)

est l'ensemble par lequel la puissance est transmise du moteur du véhicule, via la boîte de vitesses, à l'axe d'entraînement. Le groupe motopropulseur comprend la commande du moteur, qui implique la coordination de l'injection du carburant, du régime moteur, du contrôle des soupapes, du calage de la came, etc.

1-2-3.L'électronique de corps et de confort (Body Domain)

nécessite un contrôle discret. Parmi les exemples de ces sous-systèmes, on trouve la climatisation, le régulateur de vitesse, les lève-vitres et le contrôle des sièges, etc. Ces systèmes reposent généralement sur l'interaction du conducteur et ne sont pas critiques pour la sécurité.

I-2-4.Les systèmes multimédias et télématiques (Multimedia, Telematic and HMI)

Comprennent, par exemple, l'interconnexion de périphériques sans fil, les haut-parleurs, le GPS, les moniteurs, les jeux vidéo, le traitement de la voix, les interfaces homme machine (IHM), la connectivité Internet, etc. [2]

Chaque sous-système nécessite différents niveaux de service, tels que le temps de réponse, labande passante, la redondance et la détection d'erreur, souvent appelés niveaux de qualité de service

. Celles-ci présentent également différentes.

Toutes ces fonctions sont gérées par des ECUs (Electronic Control Unit) qui représentent les calculateurs présents dans les véhicules. Ce sont de petits boitiers noirs ayant chacun leurs spécificités et leurs rôles liés à des capteurs et des actionneurs.

Figure I-2: Illustration d'un calculateur moteur

Celui-ci est dédié uniquement au contrôle moteur, mais de nombreux autres calculateurs (ou ECU) sont présents dans le véhicule pouvant ainsi gérer d'autres fonctions. Voici une liste non exhaustive de différents types d'ECU

Abréviation	Désignation	Utilité
ECU ou ECM	Engine Control Unit	Système permettant la gestion du bloc moteur
SCU	Speed Control Unit	Système de régulation de vitesse, permet de rouler à une vitesse constante
TCU	Telematic Control Unit	Permet de connaitre le positionnement du véhicule et les coordonnées GPS en temps réel
BCM	Brake Control Module	Système représentant l'ABS, permettant l'aide au freinage lors des freinages d'urgences
BMS	Battery Management System	Système permettant de réguler la batterie du véhicule

Tableau I-1 : Les différents types d'ECU

Tous ces ECUs sont reliés à des capteurs et des actionneurs leur permettant d'envoyer et de traiter les informations. Une communication est donc présente entre tous ces composants électroniques via des bus de communication. Toute cette composition forme l'électronique embarqué du véhicule.

I-3.Le temps réel

Comme déjà spécifié, les systèmes embarqués sont soumis à des contraintes différentes selon leur domaine d'utilisation. Et bien, le temps réel est l'une des contraintes primordiales dans le secteur de l'automobile au niveau de la performance mais surtout au niveau de la sécurité. Devant un ordinateur classique, quelques minutes de latence ne pourront affecter que l'humeur de l'utilisateur, sur un système embarqué d'automobile seuls quelques secondes de latence suffisent à provoquer un accident avec des conséquences terribles.

Le temps réel est le fait d'être constamment en adéquation temporelle avec la réalité. Un système en temps réel est un système qui doit, non seulement, produire un résultat juste mais dans une durée limitée, sans quoi ce résultat deviendrait erroné. Ainsi, le système en temps réel doit fournir un résultat avec une contrainte de temps. Le temps est déterminé par l'environnement dans lequel se trouve le système, celui-ci doit avoir l'image la plus réaliste de celle de son environnement externe qui évolue lui-même avec le temps.

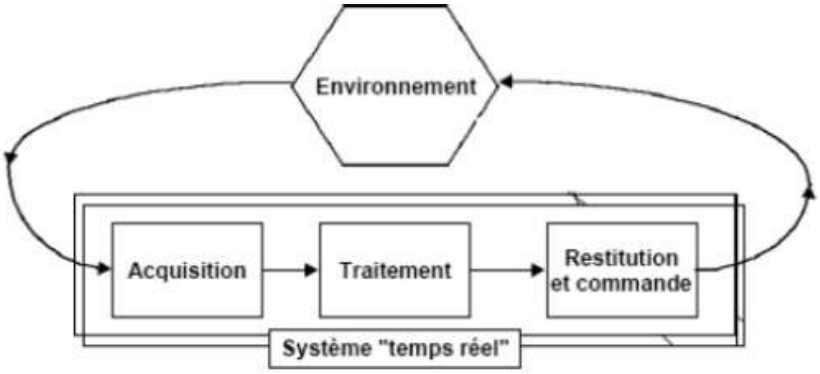


Figure I-3: Représentation d'un système en temps réel avec son environnement

L'échelle de temps dépend de l'environnement dans lequel le système est utilisé, il peut varier de quelques millisecondes à plusieurs heures. Lorsque nous sommes dans notre véhicule, il est préférable de voir l'allure à laquelle nous roulons aux millisecondes prés, l'affichage de la vitesse à quelques minutes d'intervalles rendrait les données complètement fausses

I-4. Composition d'un système embarqué

I-4-1. Calculateur

Le calculateur est l'élément principal d'un système embarqué automobile où régit la mémoire, la carte-mère ou encore le traitement logiciel. Chacun des calculateurs automobiles sont dédiés au pilotage d'une ou certaines tâches bien précises, ainsi de nombreux calculateurs sont présents dans les véhicules formant son système électronique.

Le calculateur correspond à un boitier contenant des broches électriques dotées de nombreux ports d'entrées et de sorties afin de permettre à la gestion des instruments de bord. Une carte programmable composée de circuits imprimés contient tout le traitement informatique du système, principalement codé en langage C++ ou Java, et s'accompagne d'autres éléments formant le calculateur. [1]

Nous pouvons les qualifier de systèmes « intelligents » due à leur capacité de prise de décision en fonction des paramètres d'entrées via des capteurs (ou sondeurs). Dans le cas d'un calculateur moteur, son but précis sera d'assurer les fonctions de pilotage d'un moteur en ajustant en temps réel les besoins du moteur. En recevant des signaux électriques de la part des capteurs (sonde de température, capteur de pression…), le calculateur peut traiter ces informations pour les transformer en actions par l'intermédiaire d'actionneurs (injecteur, vanne EGR…).

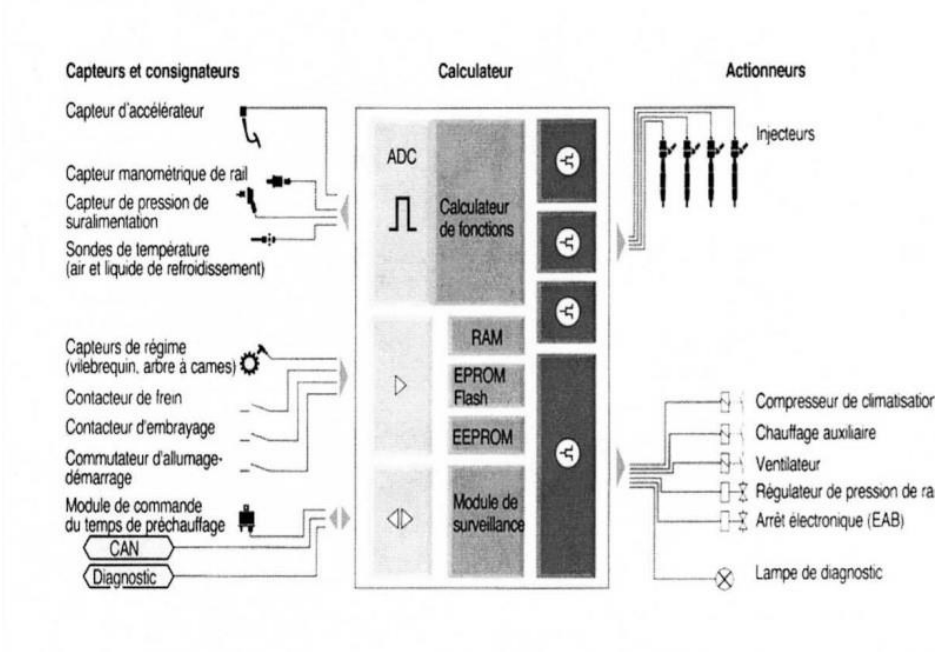


Figure I-4 Représentation d'un calculateur moteur et ses interactions

I-4-2. Actionneurs

Lorsque le traitement est réalisé par le calculateur, un signal électrique est transmis aux actionneurs permettant une action physique sur le véhicule. [1]

Ces actionneurs (ou actuateurs) transforment le signal électrique reçu en énergie mécanique. Cette transformation d'énergie peut être réalisée par moteur, de façon magnétique, hydraulique ou optique.

I-4-3. Capteurs

Les capteurs sont des éléments essentiels au fonctionnement des calculateurs puisque ce sont ces composants qui sont en charge de transmettre l'information afin d'être traitée de manière optimale.
Leur principal objectif est donc de renseigner le calculateur qui va pouvoir agir en temps réel avec l'environnement, c'est pourquoi, ces capteurs envoient de façon constante désinformations en continu au calculateur relié. De plus en plus de capteurs sont élaborés due à la sophistication des nouveaux véhicules.

Précisément, leurs tâches consistent à pouvoir transformer une grandeur physique (température, pression…) en un signal électrique afin de le transmettre au calculateur.

En effet, des interrupteurs peuvent être considérés comme des capteurs puisque les informations qui résultent de l'action émise par l'utilisateur sont directement transmises au calculateur. Fonctionnement des systèmes embarqués [1]

I-4-3-1.Capteur inductif

Pour la saisie de mouvements (vitesses de rotation, rotations de vilebrequin, etc.) et de positions(position de vilebrequin) on utilise par exemple des capteurs qui fonctionnent selon le principe d'induction (dénommés aussi capteurs inductifs). Le principe physique concernant la production d'une tension inductive repose sur la variation avec le temps du champ magnétique. Par exemple, le capteur de régime balaye les dents de la couronne du volant moteur et fournit une impulsion de sortie par dent.

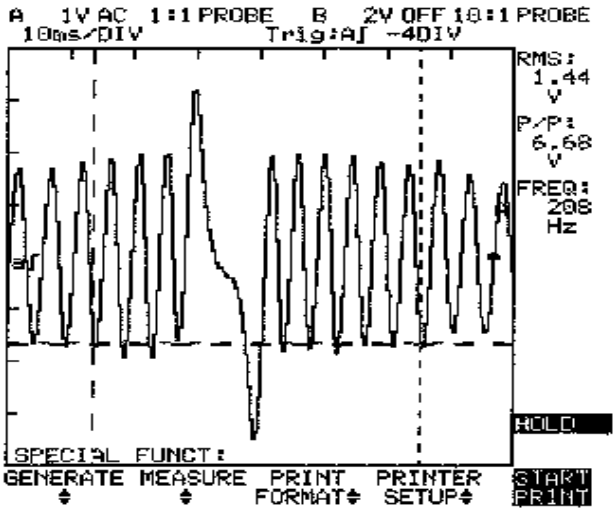


Figure I-5: l'allure du signal d'un capteur de position de vilebrequin à la vitesse de rotation du démarreur.

Figure I-6: capteur inductif

I-4-3-2. Capteur à effet Hall

Il est également possible de déterminer des vitesses de rotation (capteur de vitesse de rotation, capteur de vitesse du véhicule) et des positions (point d'allumage) à l'aide d'un capteur à effet Hall. Dans la sonde à effet Hall, une tension UH (tension de Hall) proportionnelle à la densité de champ magnétique B est crée. Un écran rotatif permet de modifier le champ magnétique en phase avec la vitesse de rotation de l'allumeur et il est ainsi possible de créer un signal de tension variant avec le champ magnétique B. [3]

La tension UH mesurée sur le générateur Hall est de quelques millivolts et doit être amplifiée à l'aide d'un circuit intégré Hall et transformée en signal de tension rectangulaire (signal binaire).

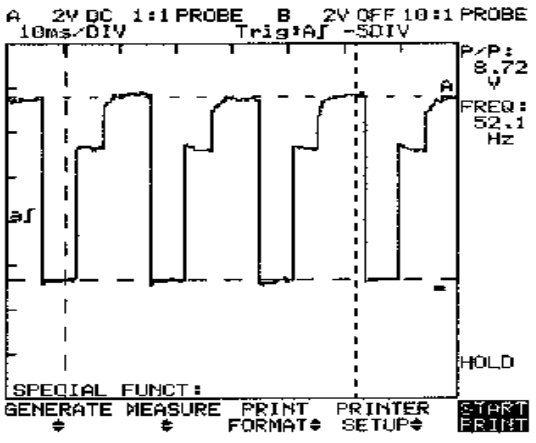


Figure I-7 :l'allure du signal d'un capteur à effet Hall dans le distributeur d'allumage au ralenti.

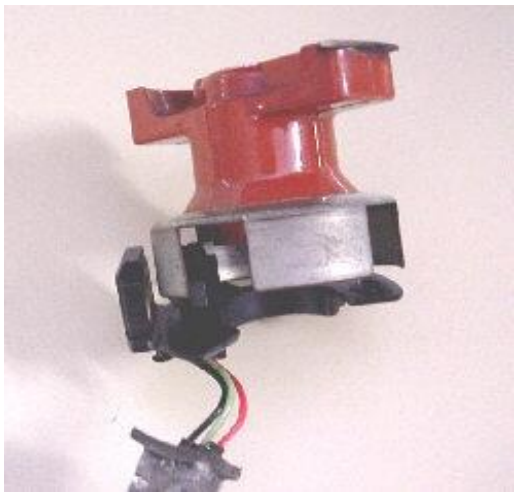


Figure I-8: capteur a effet hall

I-4-3-3.Capteur de température

Les mesures de température du moteur et de l'air aspiré fournissent à l'appareil de commande électronique des données importantes relatives aux phases de charge du moteur. Les capteurs de température mesurent électroniquement la température à partir des modifications de résistances au moyen de résistances NTC ou de résistances PTC. La plupart du temps des résistances NTC sont utilisées. [3]

L'abréviation NTC signifie Coefficient de Température Négatif : en cas d'une augmentation de température la valeur de la résistance diminue. L'abréviation PTC signifie Coefficient de Température

Positif : en cas d'une augmentation de température la valeur de la résistance augmente. Les valeurs de résistance correspondantes aux valeurs de températures sont transmises à l'appareil de commande sous forme d'un signal de tension

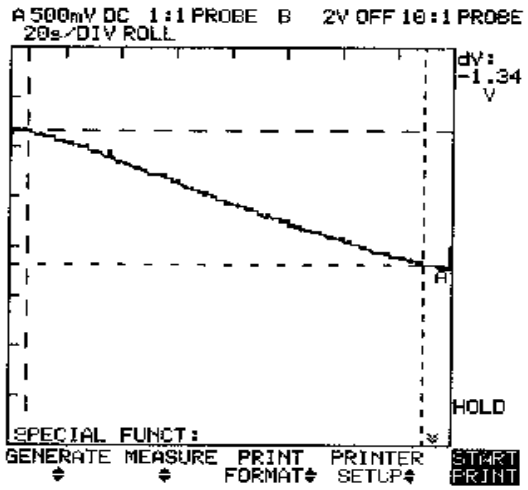


Figure I-9.signal de tension d'un capteur de température de liquide de refroidissement pour une température de 80°C

Figure I-10.capteur de température

I-4-3-4.Capteur de pression

Pour la mesure des pressions absolues ou bien relatives on utilise des capteurs piézoélectriques ou capacitifs. Ces derniers créent une tension électrique lorsqu'ils sont soumis à une pression.

Dans le domaine du moteur ces capteurs piézoélectriques sont utilisés comme capteurs de cliquetis et comme capteurs de pression dans le collecteur d'admission p. ex. dans des installations d'injection, et signalent l'état de charge du moteur à l'appareil de commande.

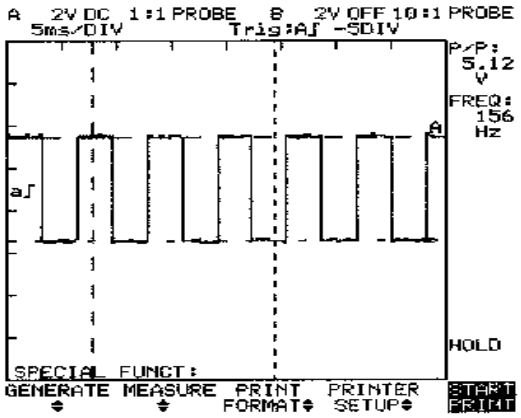


Figure I-11. signal d'un capteur de dépression, dont la fréquence se modifie selon la pression du collecteur d'admission.

Figure I-12. capteur de pression

I-4-3-5. Sonde d'oxygène (sonde lambda)

Pour qu'on puisse respecter le plus exactement possible une valeur lambda de ⌊ = 1,00 pour le traitement des gaz toxiques dans le catalyseur, le système d'échappement est pourvu d'une sonde à oxygène connue sous le nom de sonde lambda. Le capteur se compose d'une pièce creuse spéciale qui est fermée d'un côté et dont la partie intérieure est connectée avec l'air extérieur, tandis que la paroi extérieure est en contact avec les gaz d'échappement chauds. S'il y a de l'oxygène dans les gaz d'échappement, la sonde réagit en créant un signal de tension U⌊ .[3]

La tension varie suivant la richesse du mélange. La tension est transmise à l'appareil de commande età partir de là, le mélange air/carburant est mis à ⌊ = 1,00 par l'intermédiaire du circuit de réglage ⌊ .

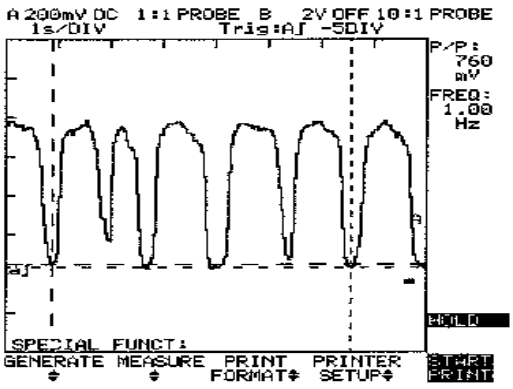


Figure I-13.signal d'une sonde lambda zirconium au régime de ralenti.

Figure I-14.sonde lambda

I-4-3-6. Potentiomètre

Pour la détermination de la position du papillon des gaz, de la pédale de l'accélérateur etc. on utilise des capteurs potentio métriques, c'est-à-dire des capteurs qui modifient leur résistance effective.

Pour la position du papillon des gaz, le balai d'un potentiomètre est actionné de façon proportionnelle à la position du papillon des gaz de sorte qu'une chute de tension correspondante se produit et est transmise à l'appareil de commande. [3]

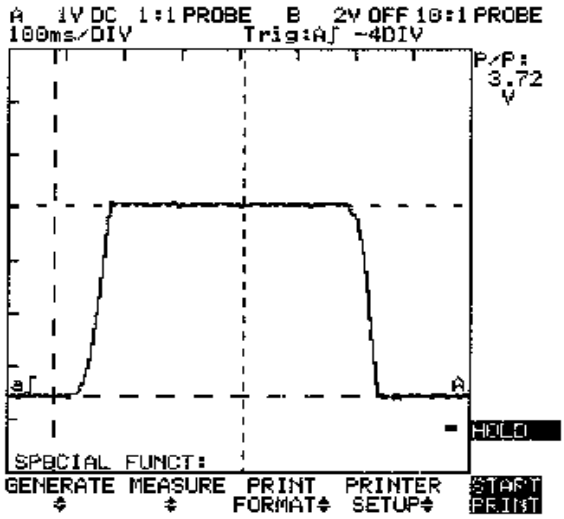


Figure I-15.le signal d'un capteur de papillon des gaz lors d'une accélération suivie d'une décélération.

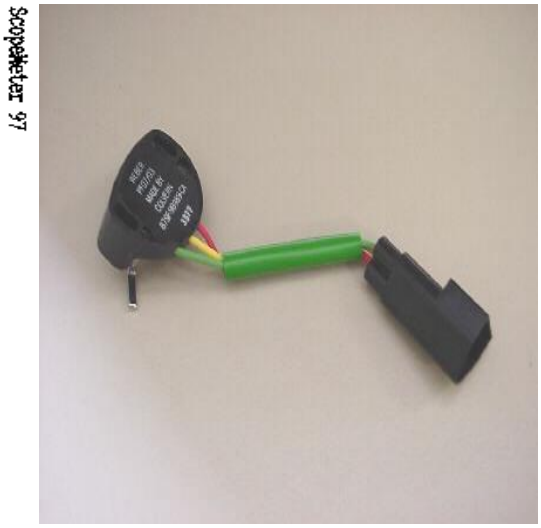
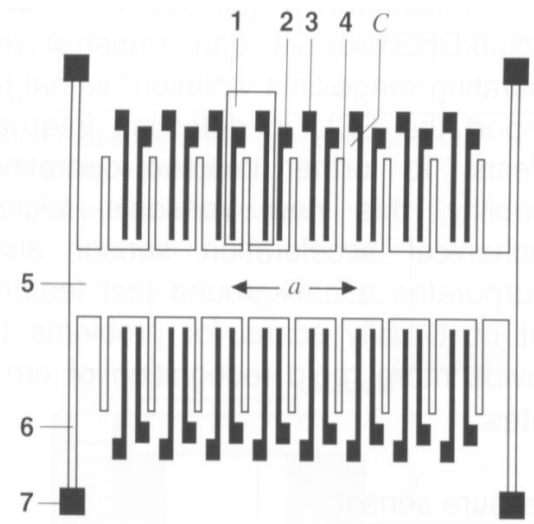


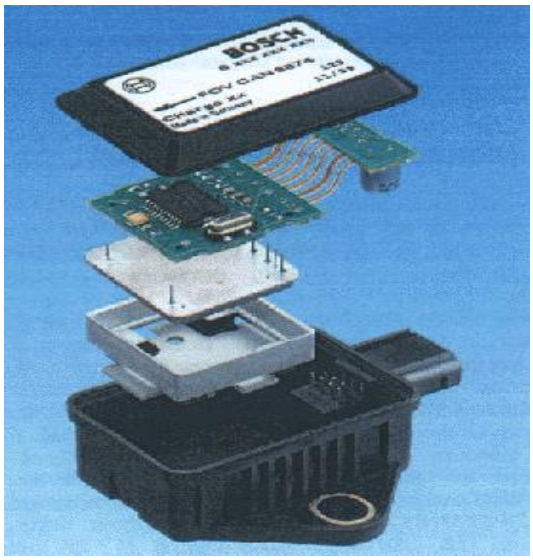
Figure I-16.potentiomètre

I-4-3-7. Capteurs capacitifs

Actuellement, le secteur automobile fait de plus en plus usage de capteurs capacitifs (mesure du niveau d'huile, suspension pilotée, capteur d'accélération). A cet effet, on utilise par exemple la modification de la capacité des deux condensateurs avec une électrode centrale.

La position de l'électrode centrale change sous l'influence d'une force. A ce moment, elle s'éloigne d'une électrode et se rapproche de l'autre. La capacité diminue ou augmente en conséquence. En calculant la différence, on obtient la mesure de l'accélération. Un tel condensateur différentiel est composé d'un matériau à base de silicium et peut donc être produit en grandes quantités et à bas prix. [3]





1 = Elément de condensateur; 2 + 3 = Electrodes fixes;

4 = Electrode mobile; 5 = Masse mobile; 6 = Barrette à ressort;

7 = Ancrage; C = Entrefer (diélectrique); a = Sens de l'accélération

I-4-3-8. Capteurs optiques

Les capteurs optiques utilisent des modifications détectées sur la lumière émise par réflexion, diffraction ou absorption. Dans le cas le plus simple, de même que dans un appareil de lecture de codes à barres, ils distinguent uniquement le clair et le sombre. A cet effet, une photodiode éclaire un champ dans lequel est présenté unco de correspondant, et un capteur photosensible mesure sil 'intensité de la lumière réfléchie se situe au-dessus ou en dessous d'une valeur de seuil. [3]

Ce principe peut être utilisé de manière appropriée pour la mesure d'un déplacement linéaire. Des marques sombres présentent des écarts à intervalles fixes et un compteur détecte le nombre des détections. On peut également mesurer des angles. À cet effet, le code à barres est par exemple appliqué sur un disque circulaire qui tourne autour d'un axe. On utilise par exemple huit pistes qui sont marquées de la manière suivante : la piste 1 est pour moitié claire et pour moitié sombre ; sur la piste 2, la clarté change chaque quart de piste ; sur la piste 3, la cadence est d'un huitième, et ainsi de suite. Si le motif instantané de clarté est détecté par plusieurs cellules, la position angulaire absolue peut être définie. [1]

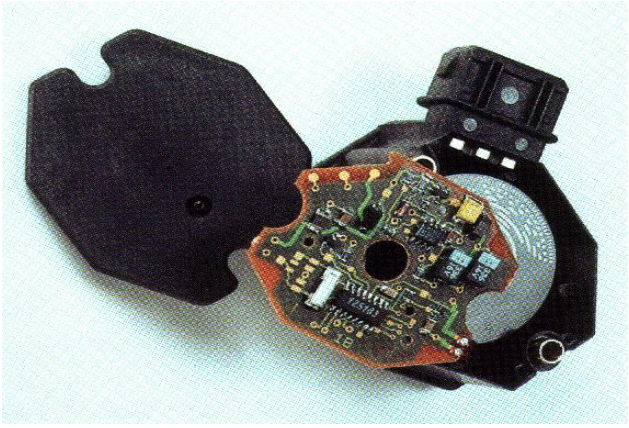


Figure I-18 Capteurs optiques

I-5.Communication

Tous ces composants échangent entre eux par l'intermédiaire de faisceaux, ils correspondent à de petits câbles permettant la transmission des signaux électriques contenant les informations recueillies et à transmettre. [1]

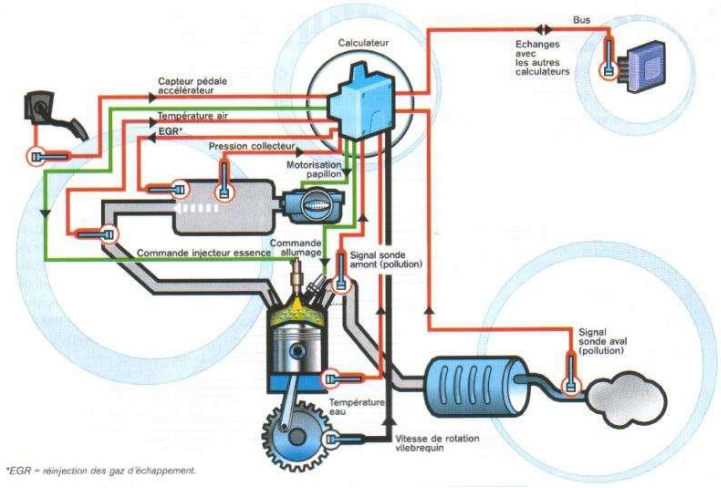


Figure I-19: Communication capteur – calculateur – actionneur

Cette figure **I-19** illustre la communication et les échanges entre le calculateur et ses actionneurs et capteurs. En vert, les interactions avec les actionneurs comme la commande d'allumage et en rouge, les interactions avec les capteurs, correspondant tous à des faisceaux électriques.

I-5-1.Multiplexage

Dans un véhicule, chaque ECU (ou calculateur) gère son propre système, cependant il est possible pour un système de pouvoir interagir et échanger des informations avec tous les autres calculateurs contenus dans le véhicule. [1]

Pour ce faire, les calculateurs communiquent entre eux par types de langages différents suivants les fonctions qui leurs sont dédiées (gestion moteur, gestion habitacle…). Ceci représente un réseau multiplexé.

Lors de l'introduction des systèmes embarqués dans les véhicules, le réseau utilisé était point-to-point, c'est-à-dire que chaque système était relié directement à un autre par l'intermédiaire de câbles. Cette méthode convient lorsque peu de systèmes sont installés car elle nécessite un nouveau câblage à chaque composant ajouté.

Ainsi, lors de l'augmentation du nombre de systèmes embarqués dans les véhicules, des kilomètres de fils et de câbles se sont vus entasser dans les véhicules impliquant une forte quantité de poids mais surtout un risque de panne plus élevé. Ceci représentait notamment un coût important de la part du constructeur ainsi qu'un espace restreint pour l'éventuel ajout de système. [1]

C'est ainsi que le multiplexage a dû faire place permettant les transitions d'informations sur un seul câblage via des protocoles de langages dédiés.

I-5-2.Classification des réseaux embarqués pour l'automobile

Une variété de réseaux dans les véhicules a évolué, principalement en fonction des exigences de coût et de performance. CAN (Controller Area Network) s'est avéré trop coûteux et compliqué pour des fonctions simples comme les fenêtres électriques ou la libération du démarrage. Des protocoles plus simples, tels que le réseau d'interconnexions locales (LIN) , pourraient offrir une fonctionnalité similaire à un coût et une consommation d'énergie moindres, et ont donc trouvé une adoption généralisée pour les fonctions non critiques. CAN s'est également révélé trop lent pour les applications qui nécessitent une bande passante élevée telles que le multimédia dans les véhicules haut de gamme, ce qui a conduit à la mise au point de protocoles à bande passante élevée, tels que MOST (Media Oriented Systems Transport) pour de telles applications. CAN à déclenchement temporel (TTCAN) est une évolution du CAN standard, qui corrige le manque de déterminisme en introduisant un mécanisme à déclenchement temporel au-dessus du cadre CAN. Le protocole FlexRay, développé par le consortium FlexRay, offre une combinaison de communication à déclenchement temporel et à déclenchement événementiel pour les applications embarquées dans les véhicules afin d'améliorer la fiabilité avec une bande passante élevé. L'extension Ethernet standard gagne également en popularité en tant que réseau fédérateur pour les futurs

véhicules. La Society for Auto motive Engainées (SAE) classe les réseaux dans les véhicules en fonction du débit et du domaine d'activité, comme indiqué dans le tableau 2. [4]

Classe	débit	Domain	Protocole
Classe A	Inférieur à 10 Kbps	Body Domain: bas de gamme	LIN
Classe B	10 à 125 Kbps	Body Domain: Non-critique & non-diagnostique	Single-wire CAN (SWC) & CAN 2.0
Classe C	125 Kbps à 1 Mbps	Powertrain Do-main: paramètres critiques et temps réel	HS-CAN
Classe D	Supé-rieur à 1 Mbps	Powertrain, Chassis: Temps réel dur & fiabi-lité	FlexRay
		Occupant Safety: Temps réel & fiabilité	Safe-by-wire
		Diffusion multi-media et diver-tissement	MOST

Tableau I-2: classification des réseaux embarqués automobile (SAE)

I.5.3. Les classes de multiplexage

Le multiplexage électronique consiste à faire circuler plusieurs informations entre divers équipements en utilisant un seul canal de transmission. Les informations sont présentées sous forme série. En fonction du débit de transmission, du niveau de sécurité envisagé, le multiplexage se décline en plusieurs classes :

· Classe A Multiplexage maître / esclave bas débits bas coût

· Classe B Multiplexage multi-maîtres moyens débits

· Classe C Multiplexage multi-maîtres hauts débits (applications sécuritaires)

· Classe D Pour les liens optiques de données (Applications télématiques et multimédia nécessitant le transport de la voix et de l'image).

On distingue deux sortes de Multiplexage dans les véhicules :

- VAN : (Véhicule Area Network)
- CAN : (Controller Area Network), à l'heure actuelle, c'est le réseau le plus employé.[5].

I-6. Le protocole VAN : généralités et concept

Le protocole VAN est un protocole de communication utilisant des débits de communication moyens, bien adapté notamment pour la gestion des fonctions de carrosserie et de confort. En effet, ces fonctions nécessitant, pour la plupart d'entre elles, des temps de réaction, entre une commande et une action, de l'ordre de 100 millisecondes (100 ms), ce type de protocole est bien adapté. [5]

Le protocole VAN est découpé en couches selon le modèle OSI de l'ISO de la manière suivante(Figure I.20)

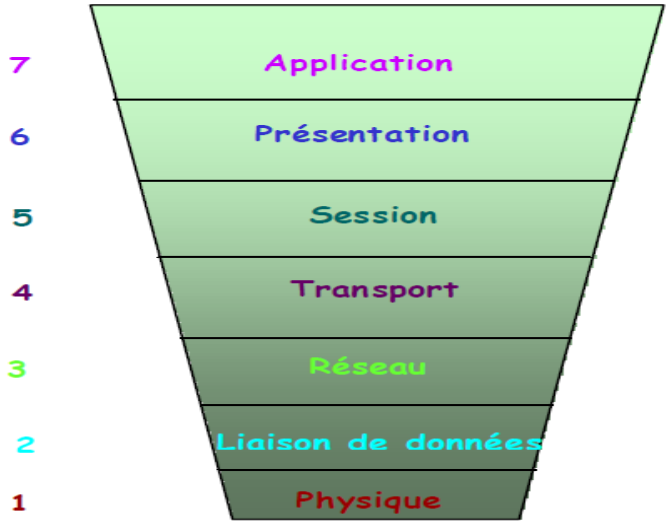


Figure I-20: Couche OSI du protocole VAN

Le protocole VAN a été inventé dans le but de mettre en relation des systèmes communicants complexes entre eux mais aussi (et peut être surtout) pour faire communiquer des éléments simples et esclaves avec un maître qui assurera le cadencement du réseau. [5]

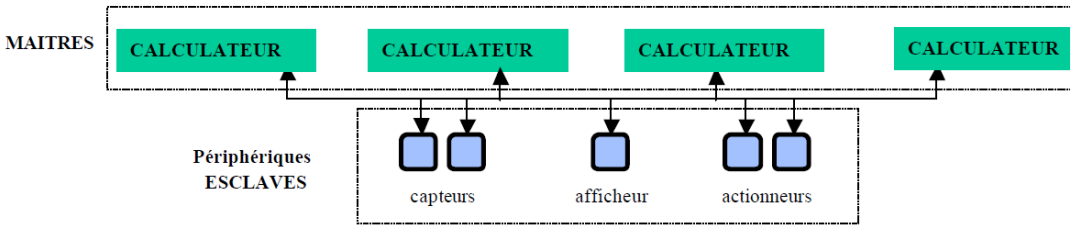


Figure I.21 : Concept VAN

I-6-1Topologie

Les topologies, c'est-à-dire l'arrangement des calculateurs, permis par le protocole VAN, sont à peuprès libres. Les calculateurs sont cependant le plus souvent connectés en obéissant à une topologie de type arbre-bus ou étoile-arbre-bus comme le montre la figure **I-22** d'une architecture bi VAN/CAN ci-dessous :

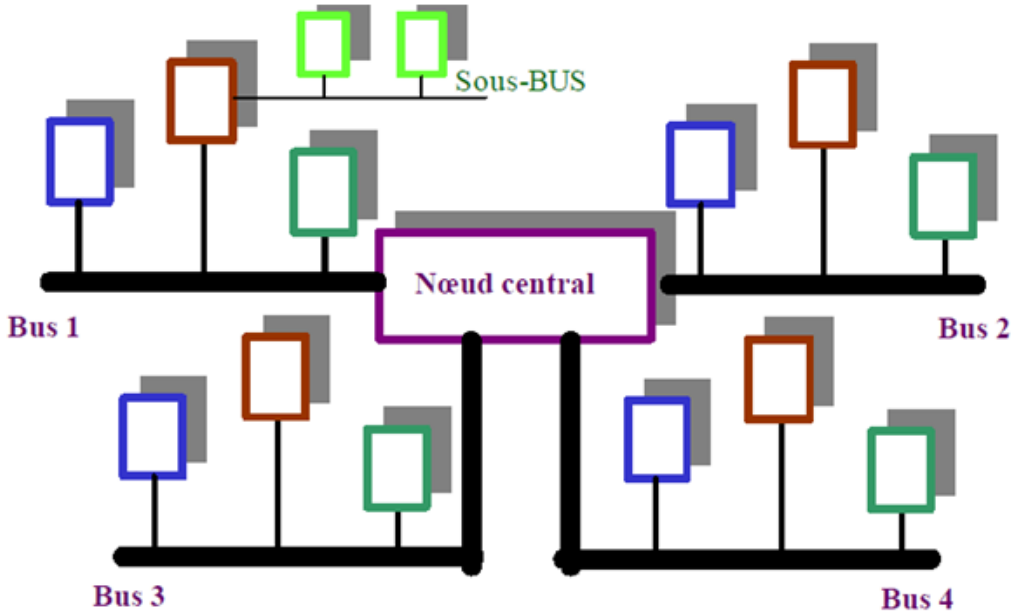


Figure I-22 : Topologie d'un réseau VAN

Le nombre maximum de nœuds VAN sur le même bus est 16 et la longueur maximale autorisée entre 2 calculateurs est 40 mètres. [5]

I-7. Le protocole LIN

I-7-1. Historique et concept du protocole LIN

Le protocole LIN a été créé en 1998, par un consortium d'entreprises regroupant Motorola, Bmw, Daimler Chrysler, Volkswagen, Volvo Et Volcano.

La création de ce protocole a suivi les étapes principales suivantes :

· Octobre 1998 : création d'un groupe de travail autour d'un protocole bas coût / bas débit.
· Juillet 1999 : Apparition d'un premier document de spécification.
· Mars 2000 : Création d'un consortium LIN par Audi, BMW, Daimler Chrysler, Volvo, Volkswagen, Volcano et Motorola.

· Novembre 2000 : Diffusion de la spécification LIN 1.2.

· Septembre 2003 : Diffusion de la spécification LIN 2.0.

· 2004 : 1ere application LIN en série chez CITROËN sur X3 (nouvelle C5) sur les applications [5]

AFIL (Alerte Franchissement Involontaire de Ligne) et Projecteurs Directionnels.

Le protocole LIN a été inventé dans le but de mettre en relation des systèmes communicants simples entre eux. Ce protocole est conçu afin de mettre en relation des éléments simples et esclaves avec un maître qui assurera le cadencement du réseau.

Seul le maître du réseau LIN est habilité à prendre la parole :

· Pour commander les esclaves.

· Pour récupérer l'état des esclaves.

La figure I-23 ci-dessous illustre le concept LIN :

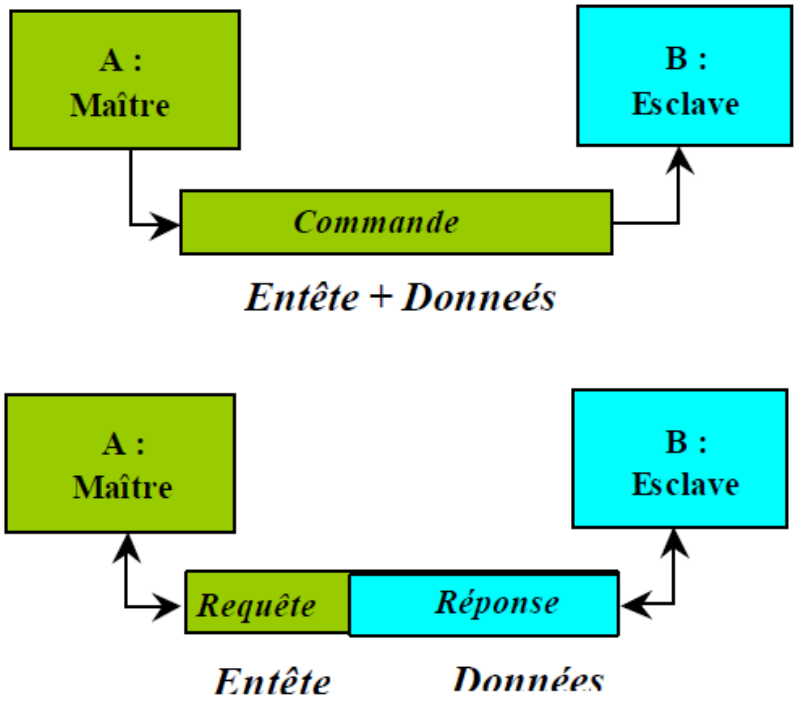


Figure I.23 : Concept LIN

I-7-2. Raisons de l'utilisation d'un sous réseau LIN (en complément du CAN)

Les protocoles VAN et CAN ont tous deux des compétences et performances similaires. Ce sont des protocoles de communication bien adaptés aux besoins de contrôle / commande, dont le coût reste moyen et dont la robustesse est nécessaire. Les débits maximum autorisés par ces protocoles sont de l'ordre de 1 Mbits/s (sur une distance maximale de 40 mètres). Par contre, ils restent encore chers et trop complexes lorsque les fonctions à multiplexer sont assez simples et ne présentent pas de possibilité d'économie significative en terme de suppression de fils(pilotage d'un moteur de serrure, …).[5]

Le protocole LIN est donc utilisé en complément des protocoles VAN et CAN et non en remplacement de ceux-ci. De plus, l'adoption du protocole LIN permet d'éradiquer bon nombre de liaisons séries en tout genre proposées par les équipementiers et dont les principales caractéristiques suivantes ne sont pas maîtrisées :

· Compatibilité Electromagnétique (CEM) non maîtrisée.

· Mécanisme de veille / réveil inexistant donc la consommation n'est pas maîtrisable.

· Absence de diagnostic.

I-7-3. Architectures actuelles

Les architectures actuelles faisant appel aux protocoles VAN et CAN ont permis de réaliser un premier niveau de multiplexage ayant abouti à la suppression d'un grand nombre de fils et d'interconnexions. [5]

Cependant, comme l'indique la figure I-24 ci-dessous, même si les liaisons entre les calculateurs VA Nou CAN sont aujourd'hui optimisées, les liaisons entre un calculateur et ses capteurs et/ou actionneurs ne sont pas toujours multiplexées. La raison principale avant l'adoption du protocole LIN, était principalement le coût, le VAN et le CAN étant tous deux trop chers.

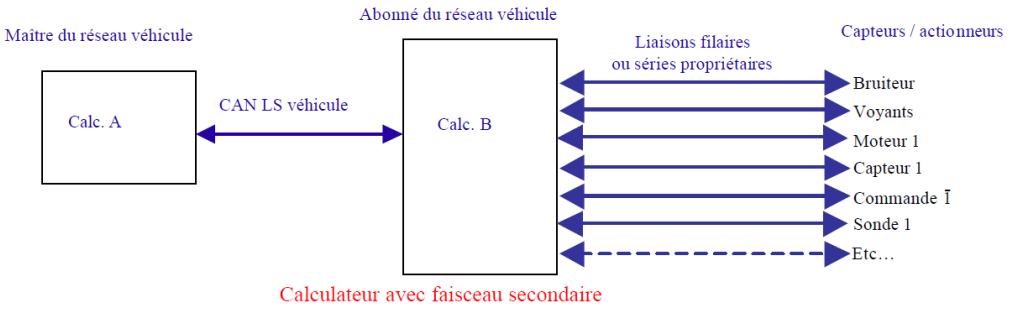


Figure I-24 : Architectures actuelles sans LIN

I-7-4. Architecture avec l'apport du protocole LIN

Avec l'apport du protocole LIN, les liaisons entre certains calculateurs et leurs capteurs et/ou actionneurs sont aujourd'hui multiplexées. [5]

Pour chaque calculateur, une balance économique est calculée et l'électronique à rajouter dans les capteurs / actionneurs est comparée aux fils supprimés.

Dans le cas où la balance économique est favorable, le faisceau secondaire est supprimé au profit d'un sous -réseau LIN reliant les capteurs / actionneurs au calculateur, comme illustré dans la figure ci-dessous :

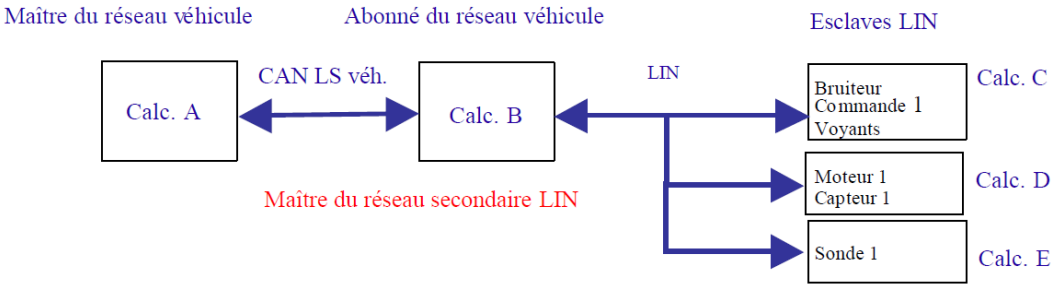


Figure I-25: Architectures actuelles sans LIN

I.CONCLUSION

Le multiplexage est présenté comme une révolution technologique majeure dans l'automobile parce qu'il permet aux constructeurs d'équiper de façon fiable les véhicules d'un grand nombre d'options.

Cela est vrai, mais cette évolution n'a pas été faite pour seulement satisfaire le client, mais aussi pour des questions de coût.

En effet, à équipements équivalents, la fabrication d'un véhicule multiplexé revient moins cher qu'une automobile non multiplexée. La fabrication en série des calculateurs électroniques (réalisée par des équipementiers à l'étranger) est bien meilleur marché que la pose d'une quantité plus importante de fils à base de cuivre dont le prix ne cesse d'augmenter. D'autre part, la gestion à base de calculateurs et de logiciels permet d'adapter facilement les véhicules au marché sans avoir à refaire de coûteux développements ; de même, une mise à jour logicielle (souvent opérée par les constructeurs sur les véhicules à problèmes) permet de ne plus déclarer certaines fautes en changeant tout simplement les critères de déclenchement et ainsi d'augmenter la fiabilité à moindre coût.

Chapitre II

Présentation générale du Bus CAN

II-Introduction

Controller Area Network (CAN) est un système de communication de type diffusion série, asynchrone, développé par Bosch GmbH dans les années 1980. Il a été développé à l'origine pour l'industrie automobile pour remplacer le faisceau de câbles standard par un simple deux fils autobus.

Les spécifications du système permettent des tolérances EMI robustes et la capacité de auto-vérifier et corriger ses propres erreurs. Il peut être appelé un bus multi-maître reliant diverses unités de contrôle appelées nœuds.

CAN est un système où tous les nœuds connectés au bus CAN peuvent entendre chaque message envoyé en même temps simultanément. Cette méthode permet aux microcontrôleurs et d'autres processeurs pour communiquer entre eux sans avoir besoin d'un hôte. Dans un réseau CAN, de nombreux petits messages sont envoyés sur tout le bus, ce qui permet la cohérence des messages, contrairement à d'autres méthodes réseau comme USB qui envoient de gros messages blocs d'un point à un autre sous un superviseur hôte.

II-1.Présentation générale du Bus CAN

II-1-1. 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. [6]

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. [6]

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 Système Under en Schnittstellenfür die Elektronikim Kraf tfahrzeug»,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 Distribute de Xecutive), 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. [6]

II-1-2. 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 Semi-conducteurs.

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 13e conférence internationale (iCC).

II-1-3. 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). [7]

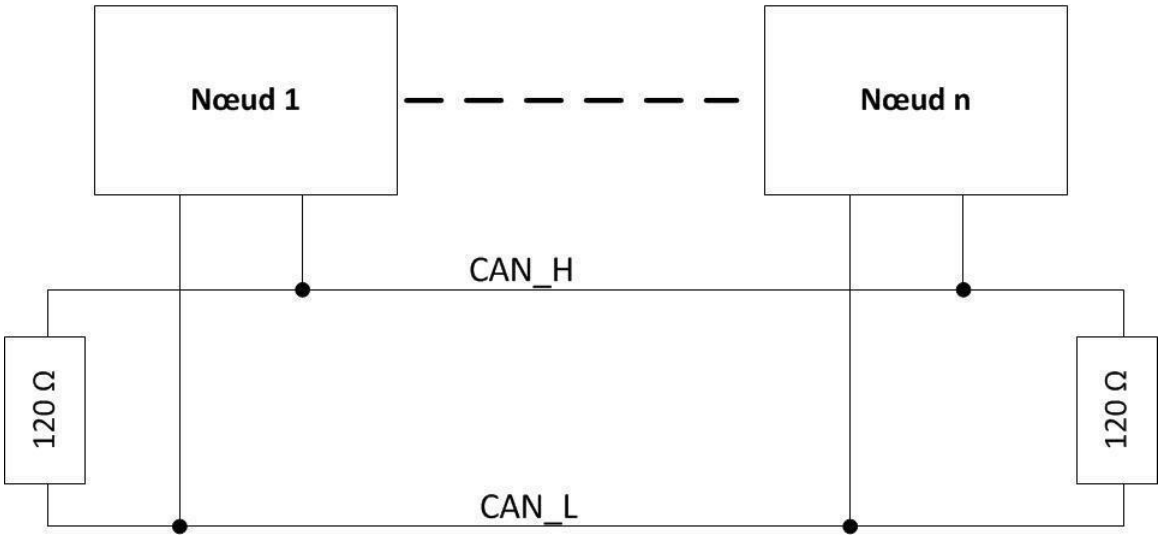


Figure II-1 : Fonctionnement du Bus CAN

II-1-4. 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.

II-1-4.1 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. [7]
- **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. [7]

- **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, Partie6: 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.

II-1-4.2 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 CAN open spécifique au réseau

CAN dans les véhicules ferroviaires, les locomotives que dans les autocars,

- IEC 61800-7-201/301 : spécifique pour le CIA 402 CAN open sur le dispositif qui établit le profil du contrôle des mouvements et des énergies.

II-1-4.3 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. [7]

II-1-5.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. [8]

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. [8]

II-1-6. 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. [8]

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.

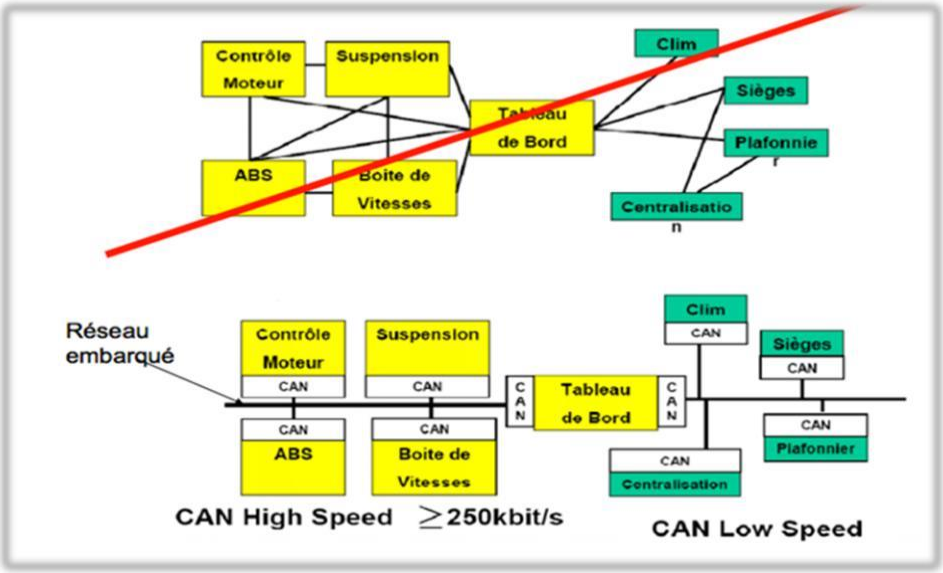


Figure II-2 : Intérêts du Bus CAN dans l'automobile

II-1-7. 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) [7]

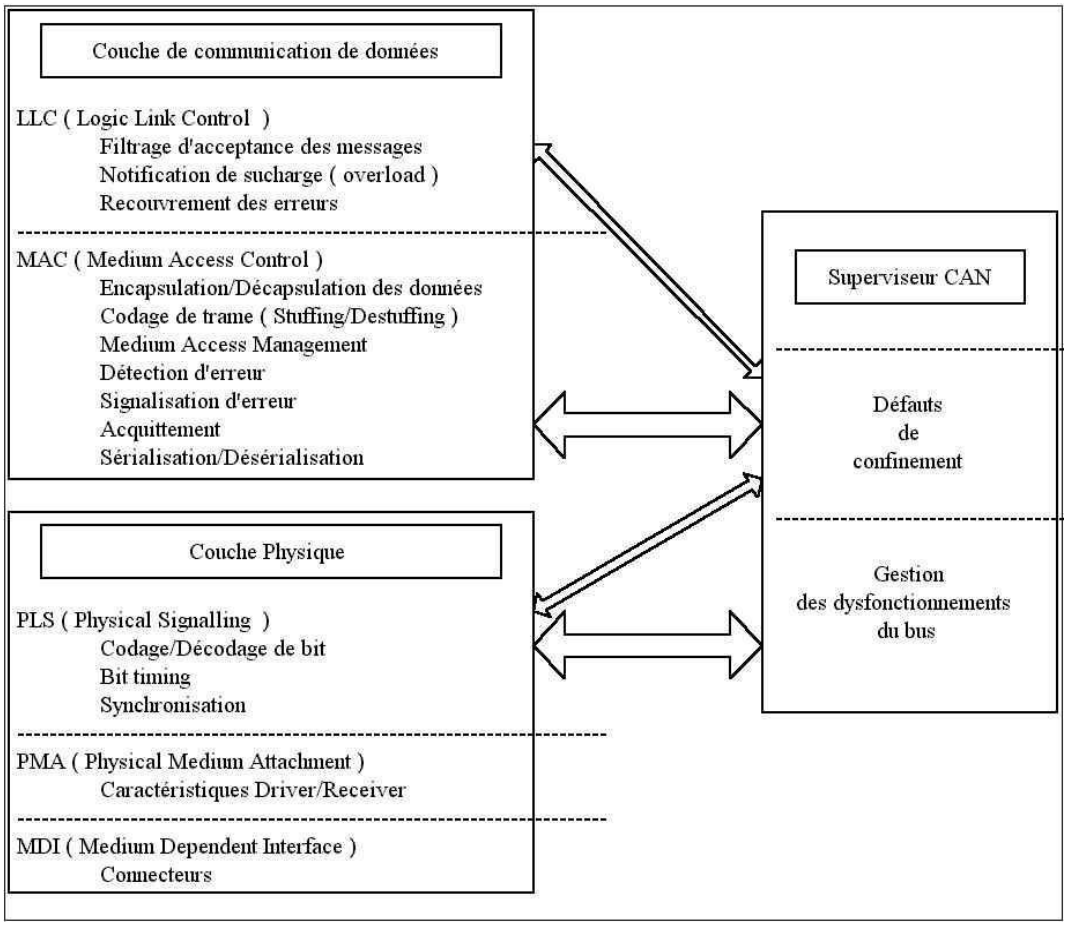


Figure II-3 : Application du CAN au modèle OSI

II-1-7.1 Couche applicative

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

II-1-7.2 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. [7]

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.

II-1-7.3 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. [7]

II-2.Analyse technique

II-2.1. Caractéristiques électriques

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

II-2-1.1 Support de transmission

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

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

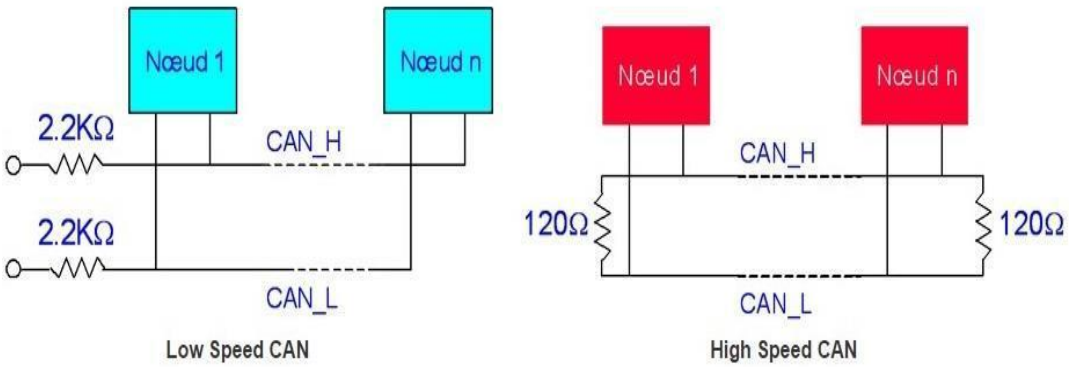


Figure II-4 : Schéma de câblage Bus CAN

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	2à20	2à30	2à30
Courant de sortie	> 1 mA sur 2,2kΩ	25 à 50 mA sur 60Ω	25 à 50 mA sur 60Ω
Niveau dominant	CANH=4V CANL=1V	CAN H = 3.5V CAN L = 1.5V	CAN H = 3.5V CAN L = 1.5V
Niveaurécessif	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 Deuxextrémités du bus entre CAN H et CAN L (=60Ω)	Résistance de 120Ω aux deuxextrémités du bus entre CAN H et CAN L (=60Ω)
Tensions d'alimentation	5V	5V	5V

Tableau II-1 : Types de transmission en CAN

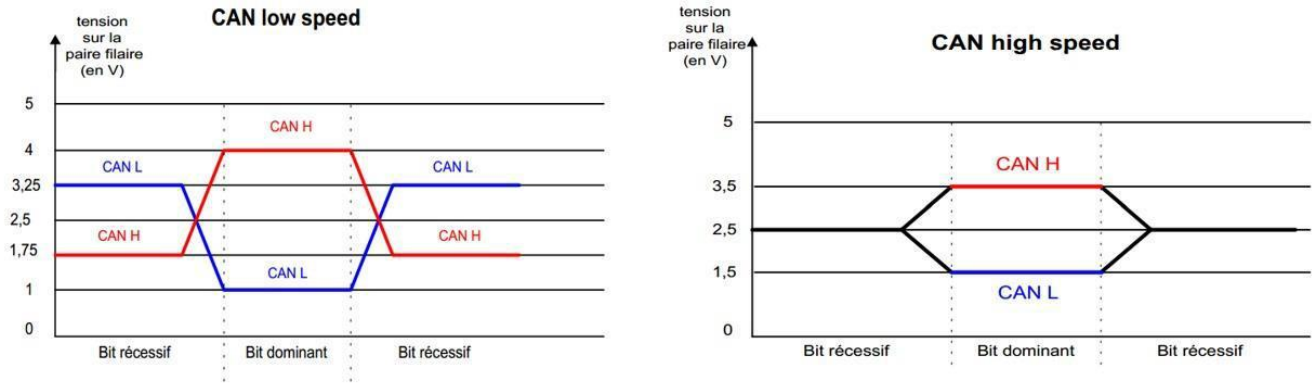


Figure II-5 : Niveaux de tension en Low Speed & High speed

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. [7]

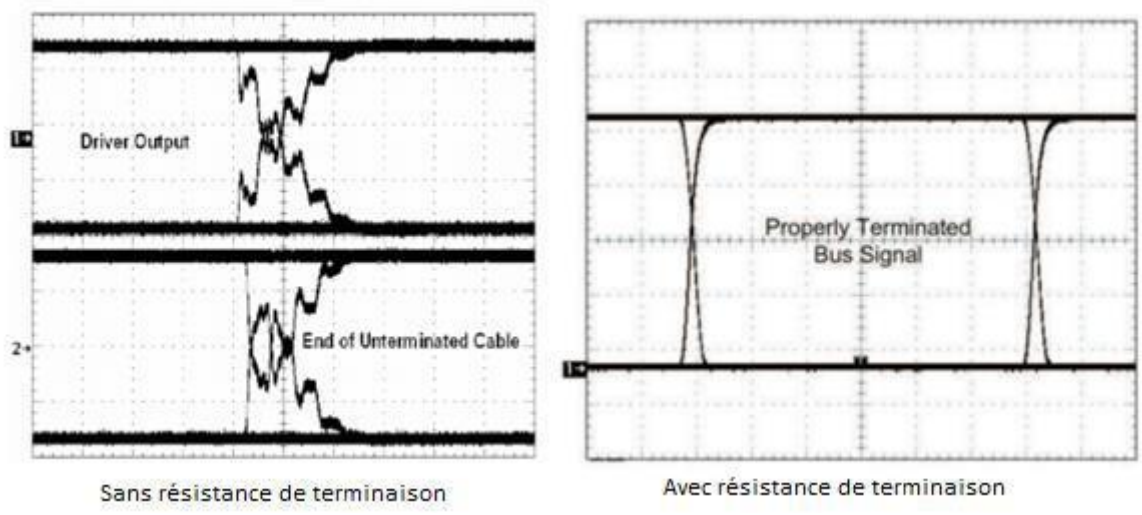


Figure II-6 : Bus CAN sans résistance de terminaison et avec résistance de terminaison

A 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.

II-2-1-2 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. [7]

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

Tableau II-2 : 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.

$tnbit > 2 * tbus$ or $tbus = longueur\_du\_bus/200000$

Sachant que le débit du réseau (Débit-du-réseau) =1/$tnbit$, on obtient la relation suivante :

$$\text{Débit-du-réseau} = \frac{1}{tnbit} = \frac{1}{2*tbus} = \frac{1}{\frac{2*longueur\_du\_bus}{200000}} = \frac{200000}{2*longueur\_du\_bus}$$

II-2-1-3 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[7]

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 ».

II-2-1.4.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.

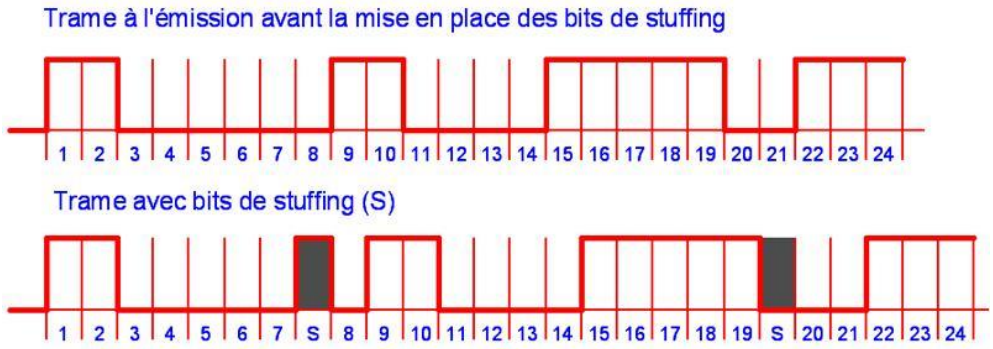


Figure II-7 : Démonstration du "bit stuffing"

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 « de stuffing » 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. [7]

II-2-2. 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. [9]

II-2-2.1 Décomposition d'une trame

II-2-2.1.1 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,

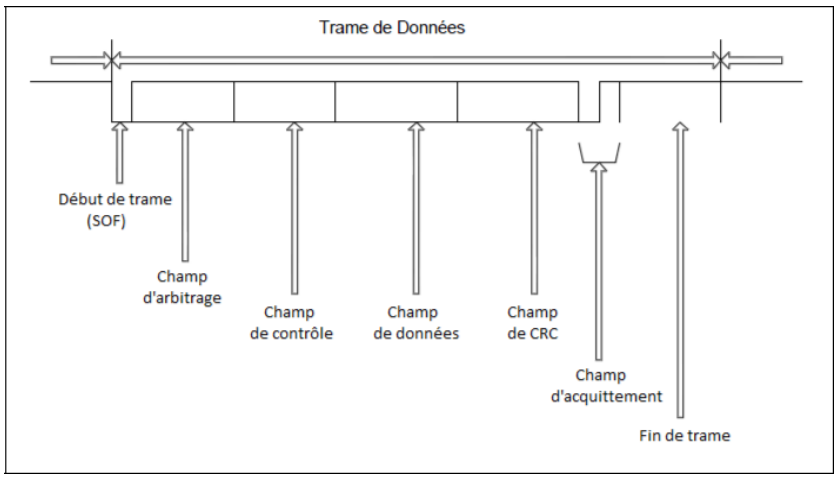


Figure II-8 : Décomposition d'une trame de données

II-2-2.1.2 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. [9]

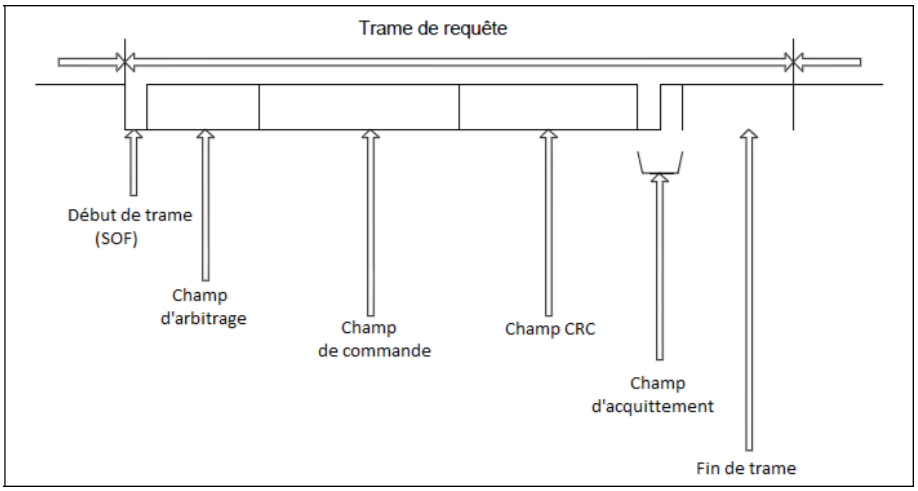


Figure II-9 : Décomposition d'une trame de requête

II-2-2.1.3 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. [9]

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

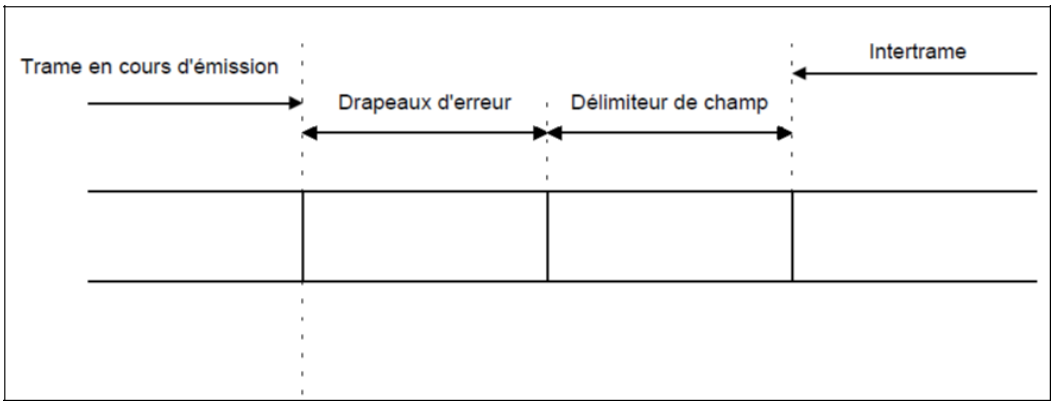


Figure II-10 : Constitution de la trame d'erreur

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). [9]

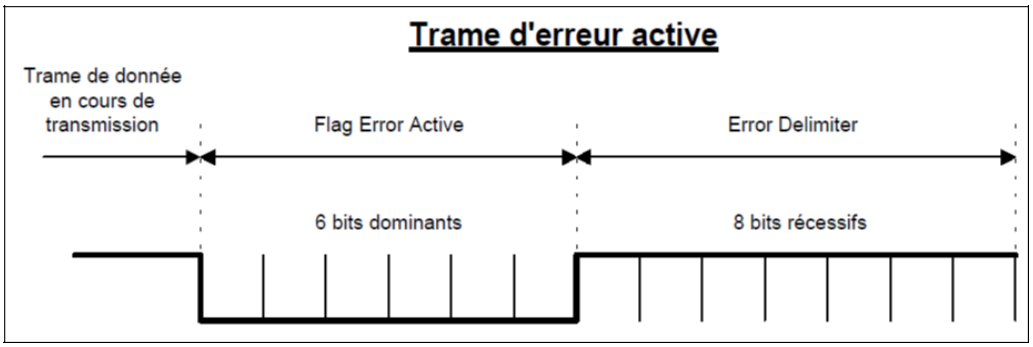


Figure II-11 : Trame d'erreur active

- 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 (ErrorDelimiter).

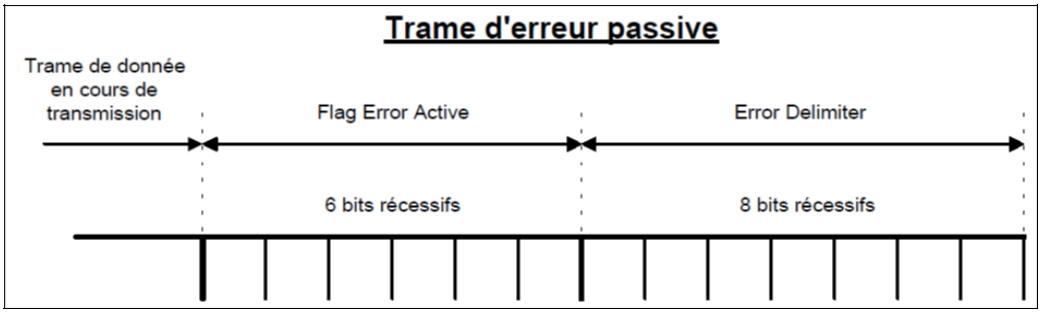


Figure II-12 : Trame d'erreur passive

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

II-2-2.1.4 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. [9]

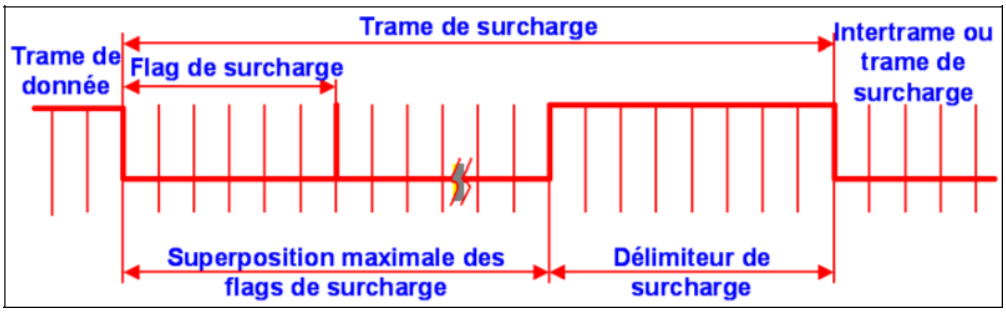


Figure II-13 : Décomposition de la trame de surcharge

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. [9]

II-2-2.2 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.

II-2-2-2-1 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.

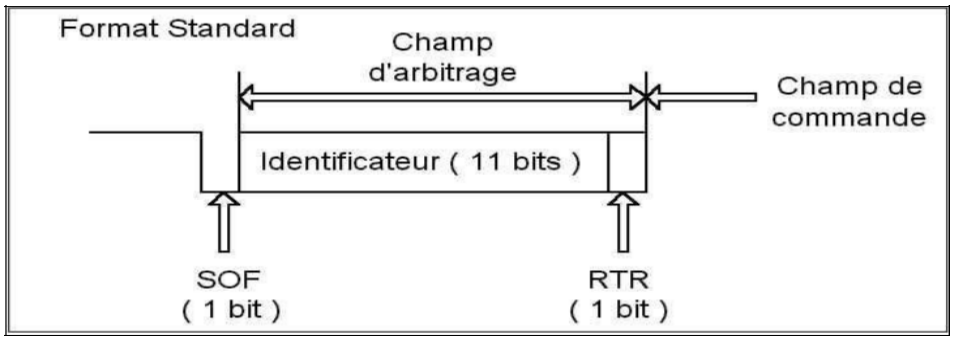


Figure II-14 : Champ d'arbitrage

II-2-2-2-2 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.

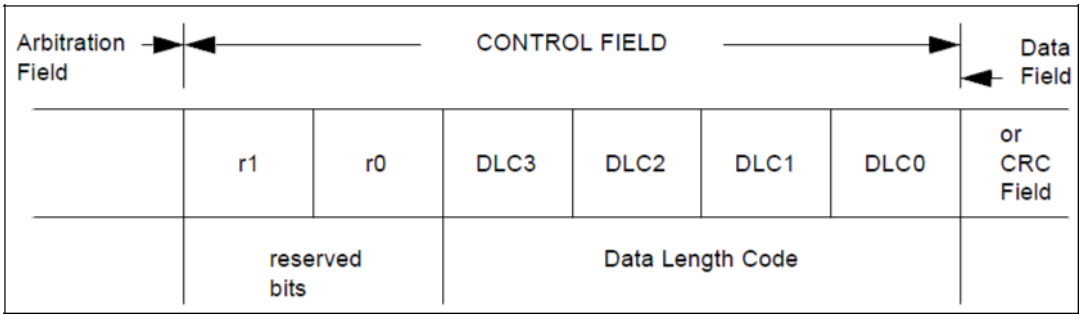


Figure II-15 : Champ de contrôle

II-2-2-2-.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. [9]

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

d :rbit:bitdominantrécessif

r : bit récessif

Figure II-16 : Codage des bits DLC suivant la taille des données en octets

II-2-2.2.4 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. [9]

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.

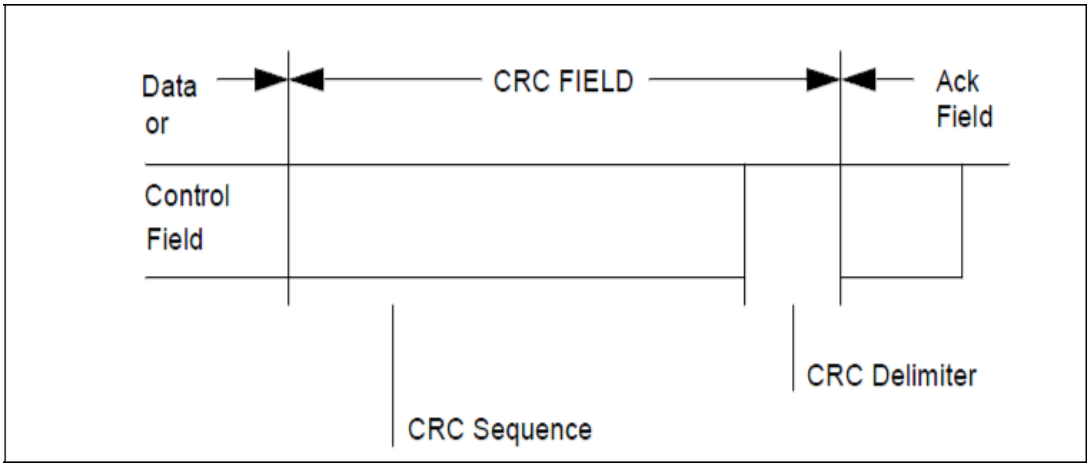


Figure II-17 : Champ CRC

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 II-18 : Algorithme CRC Bosch

II-2-2.2.5 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. [9]

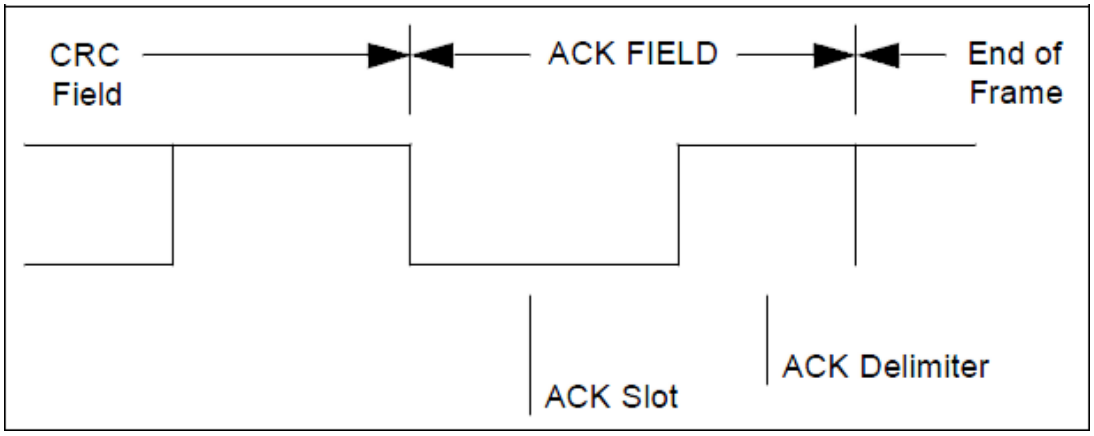


Figure II-19 : Champ d'acquittement

II-2-2.2.6 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.

II-2-2.3 Période d'inter trame

Les trames de données et de requêtes sont séparées des autres trames par une inter trame, alors que les trames d'erreur et de surcharges ne sont pas séparées par cette inter trame. 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.

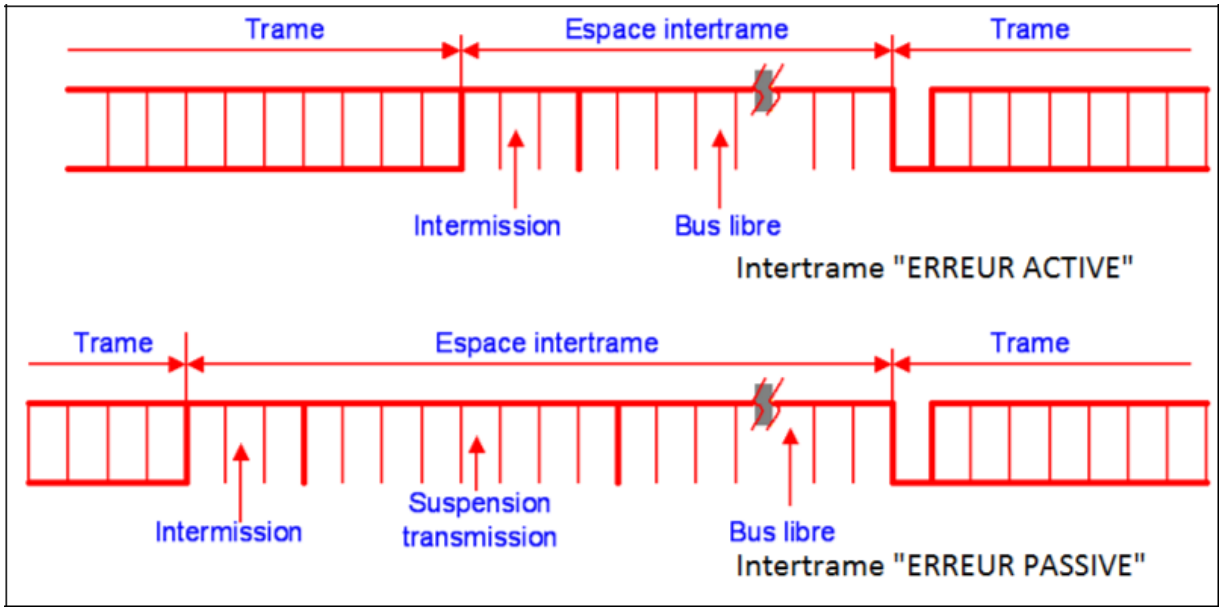


Figure II-20 : Composition période d'inter trame

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. [9]

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. [9]

II-2-3. 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.

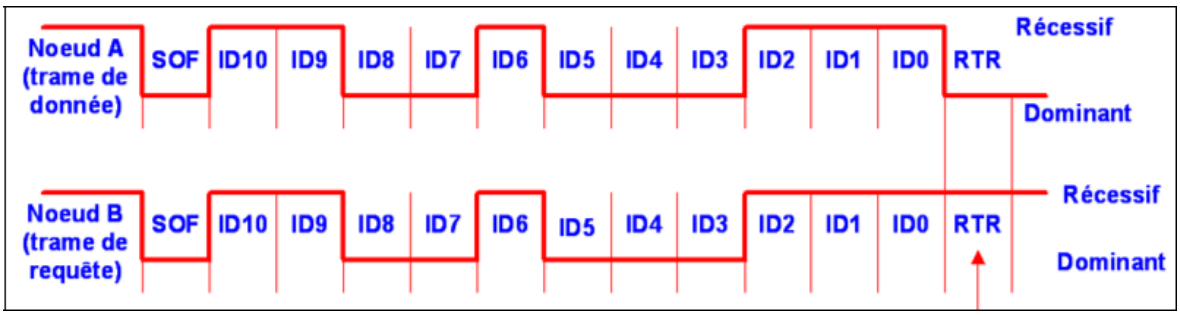


Figure II-21: Comparaison de deux niveaux de priorité

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. [9]

II-2-4.1 Les différents types d'erreurs

II-2-4.1.1 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

II-2-4.1.2 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.

II-2-4.1.3 L'erreur de CyclicRedundancy 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).

II-2-4.1.4 L'erreur d'Acknowledge Delimiter

Lorsque dans le champ d'AcknowledgeDelimiter, 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.

II-2-4.1.5 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.

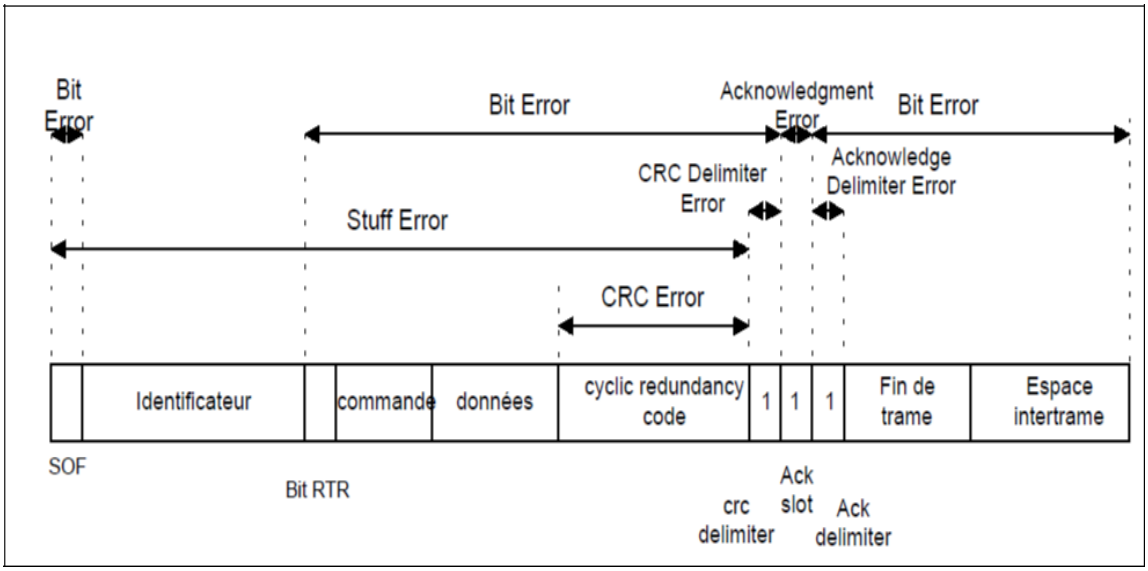


Figure II-22 : Les sources d'erreur dans la trame CAN

II-2-4.2 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 ErrorCounter), pour les erreurs d'émission
- Un compteur nommé REC (ReceiveErrorCounter), 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). [9]

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 **II-23**):

- 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

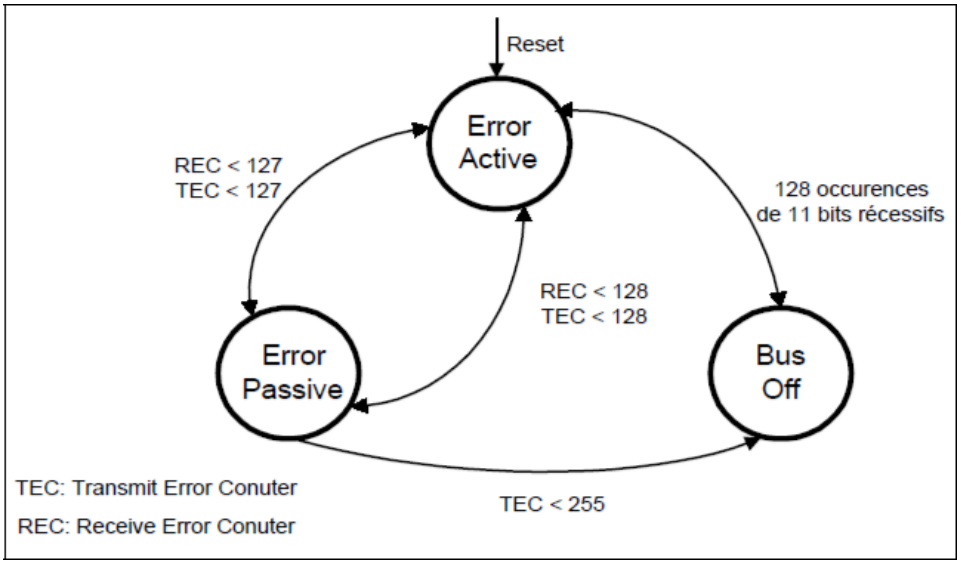


Figure II-23 : Compteur d'erreur et état d'un nœud

II-2-5. Modes de fonctionnement

II-2-5.1 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.

II-2-5.2 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.

II. Conclusion

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.

Chapitre III

Applications Sur Bus CAN

III.1.Introduction

Lorsque deux périphériques neoud1 et neuod2 se partagent le même bus, il convient de définir la façon comment accéder à ce bus. Le protocole CAN décrit avec précision et en détail la connexion de plusieurs dispositifs à un bus, c'est un bus très répandu dans l'industrie. Le protocole définit principalement la préséance d'accès au bus et aficher les données des capteurs sur neoud2et envoie les valeurs aussi pour l'affichage dans un ordinateur utilisant l'ogiciel LABVIEW [10]

II.2 Présentation des composants utilisés

III.2.1 L'ArduinoUno

Une carte Arduino est une carte électronique équipée d'un microcontrôleur. Le microcontrôleur permet, à partir d'événements détectés par des capteurs, de programmer et commander des actionneurs ; la carte Arduino est donc une interface programmable. La carte Arduino la plus utilisée est la carte Arduino Uno..

Figure III.1 : Carte arduino Uno

III.2.1.1Caractéristique :

Tension de fonctionnement	5V
Tension d'alimentation (recommandée)	7-12V
Tension d'alimentation (limites)	6-20V
Broches E/S numériques	54 (dont 14 disposent d'une sortie PWM)
Broches d'entrées analogiques	16 (utilisables en broches E/S numériques)
Intensité maxi disponible par broche E/S (5V)	40 mA (ATTENTION : 200mA cumulé pour l'ensemble des broches E/S)
Intensité maxi disponible pour la sortie 3.3V	50 mA
Intensité maxi disponible pour la sortie 5V	Fonction de l'alimentation utilisée - 500 mA max si port USB utilisé seul
Mémoire Programme Flash	256 KB dont 8 KB sont utilisés par le bootloader
Mémoire SRAM (mémoire volatile)	8 KB
Mémoire EEPROM (mémoire non volatile)	4 KB
Vitesse d'horloge	16 MHz

Tableau III-1. Circuit du schéma interne de la carte arduino Uno

III.2.2 Le capteur de température LM35

III.2.2.1 Définition

Le LM335 est un capteur de température qui produit une tension qui varie en fonction de la température. La variation de la tension est très faible, c'est pourquoi nous utilisons un amplificateur opérationnel. Sa representation graphique est une droite linéaire[11]

Figure III.2 : capteur de température LM35

III.2.2.2Fonctionnement

Le LM35 est un capteur de température de précision qui peut être facilement étalonnée. Il fonctionne comme un Zener 2-terminale et la tension de claquage est directement proportionnelle à la température absolue à l'10mV / ° K. Il fournit 0 V pour 0 degré, 250 mV pour 25 degrés, etc. et un maximum de 1 volt pour 100 degrés

Le circuit a une impédance dynamique inférieure à 1Ω et fonctionne dans une plage de courant de 450µA à 5mA sans altération de ses caractéristiques. [11]

III.2.2.3 Connexion avec ARDUINO

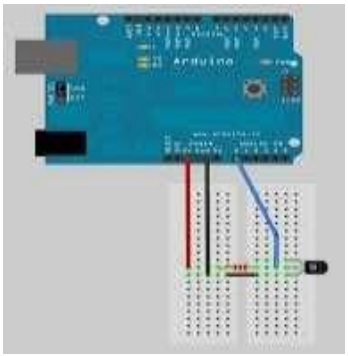


Figure III-3 :connexion de LM35

III.2.3 Potentiometr

Le potentiomètre est une résistance variable qui permet de faire varier la valeur de tension à ses bornes. Il est utilisé dans plusieurs applications notamment pour régler une valeur: régler la luminosité d'une lumière, régler le volume d'un haut-parleur, modifier la position d'un servomoteur, etc. [12]

Figure III-4 : potentiomètre

III.2.3.1 Fonctionnement

Le potentiomètre est un composant passif. Pour mesurer un changement de résistance, nous envoyons un courant entre les bornes extrêmes du potentiomètre et nous pouvons lire la valeur de la tension résultante du pont diviseur ainsi créé sur sa borne du milieu. [12]

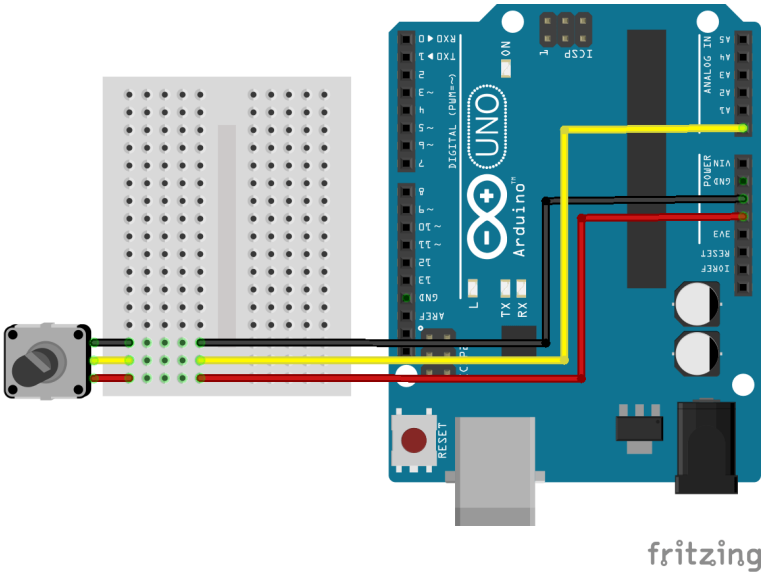


Figure III-5 : connexion arduino potentiomètre

III.2.4 Écran à cristaux liquides 16 * 2 avec système de connexion I2C

Il est caractérisé par l'utilisation du système I2C opération et contrôle à travers deux fils seulement, ce qui est une énorme provision pour les prises du panneau de contrôle utilisé dans notre projet et facilite le processus de connexion et de fonctionnement. [13]

Figure III-6 **:** Ecran LCD avec système de connexion I2C

III.2.5 Module de contrôleur de bus CAN MCP2515

Le contrôleur de bus CAN MCP2515 est un module simple qui prend en charge le protocole CAN version 2.0B et peut être utilisé pour la communication à 1 Mbps. Afin de configurer un système de communication complet, vous aurez besoin de deux modules de bus CAN. Le module utilisé dans le projet est illustré dans l'image ci-dessous. [14]

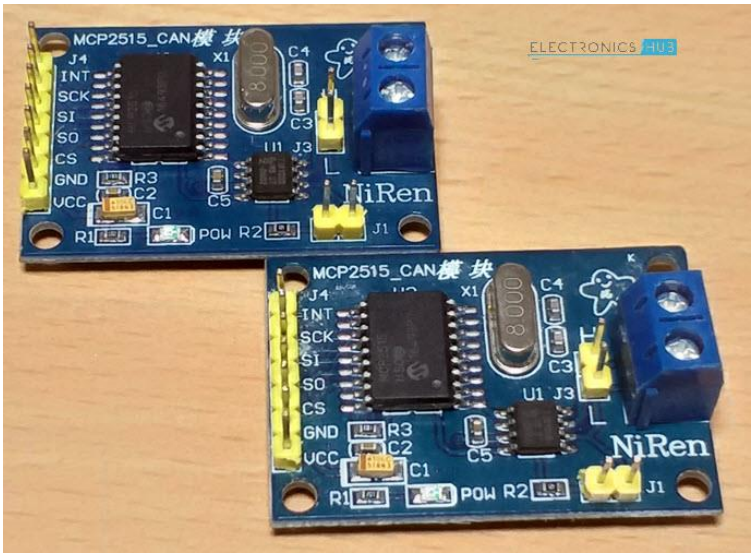


Figure III-7: module MCP2515

Ce module particulier est basé sur le contrôleur IC MCP2515 CAN et l'IC émetteur-récepteur CAN TJA1050. Le MCP2515 IC est un contrôleur CAN autonome et possède une interface SPI intégrée pour la communication avec les microcontrôleurs.

En ce qui concerne le CI TJA1050, il agit comme une interface entre le CI contrôleur CAN MCP2515 et le bus CAN physique.

L'image suivante montre les composants et les broches d'un module MCP2515 typique.

Interface de bus CAN Arduino MCP2515 Composants du module CAN MCP2515

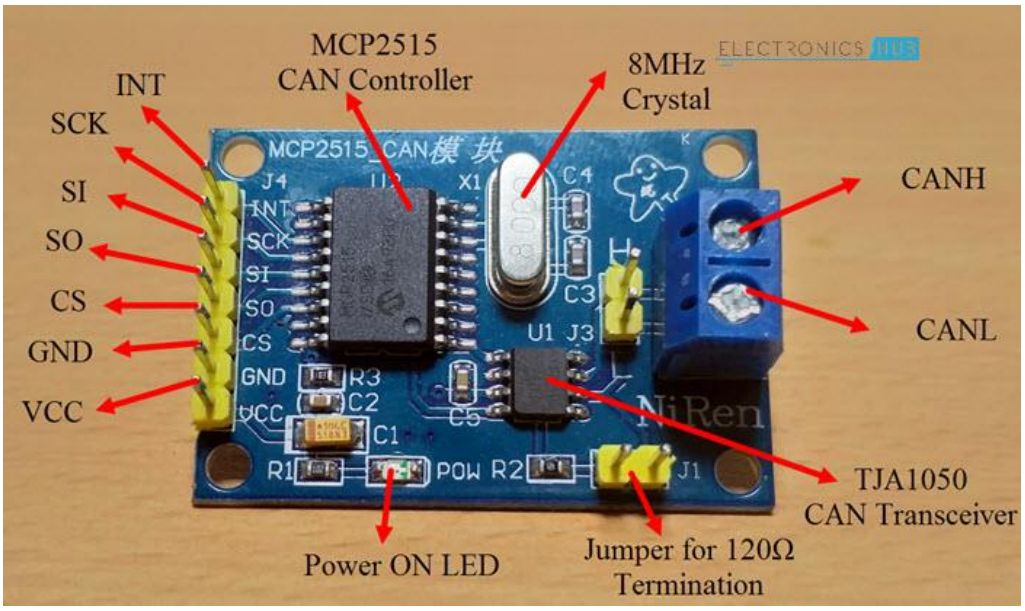


Figure III-8: Schematic of MCP2515 CAN Bus Module

Avant de voir le schéma du module, vous devez comprendre quelques choses sur les deux circuits intégrés, c'est-à-dire MCP2515 et TJA1050. [14]

MCP2515 IC est le contrôleur principal qui se compose en interne de trois sous-composants principaux: le module CAN, la logique de contrôle et le bloc SPI.

Le module CAN est responsable de la transmission et de la réception des messages sur le bus CAN. Control Logic gère la configuration et le fonctionnement du MCP2515 en interfaçant tous les blocs. Le bloc SPI est responsable de l'interface de communication SPI.

En ce qui concerne le CI TJA1050, puisqu'il agit comme une interface entre le contrôleur CAN MCP2515 et le bus CAN physique, ce CI est chargé de prendre les données du contrôleur et de les relayer sur le bus.

L'image suivante montre le schéma du module CAN MCP2515 et montre comment les IC MCP2515 et TJA1050 sont connectés sur le module.

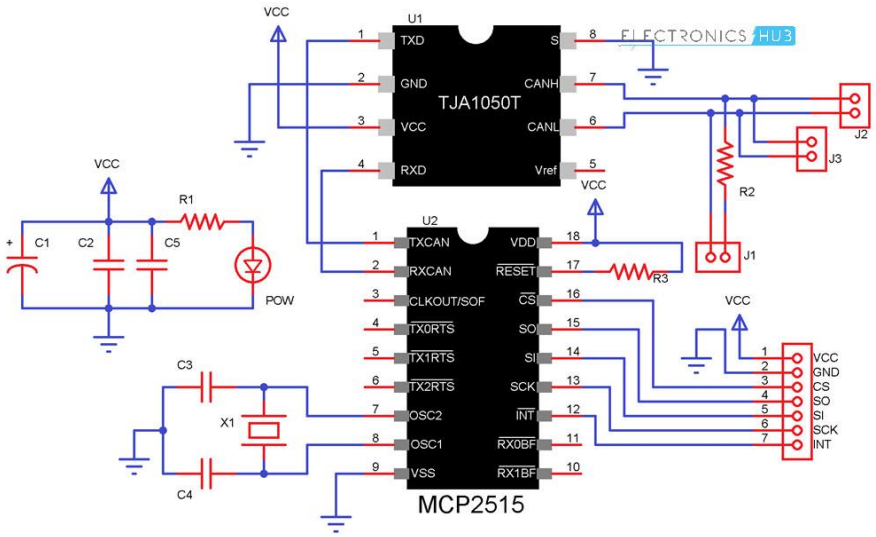


Figure III-9 : Schéma de circuit pour l'interfaçage du MCP2515 avec Arduino

L'image suivante montre le schéma de circuit de l'interfaçage du module CAN MCP2515 avec Arduino et la communication possible entre deux Arduino via le protocole CAN. [14]

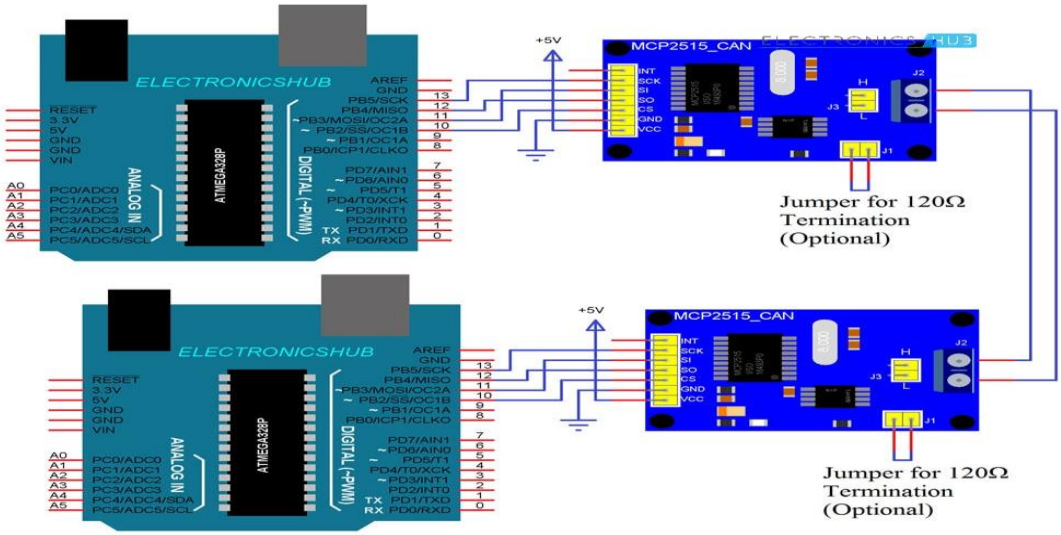


Figure III-10 : Schéma du circuit de l'interface du bus CAN Arduino MCP2515

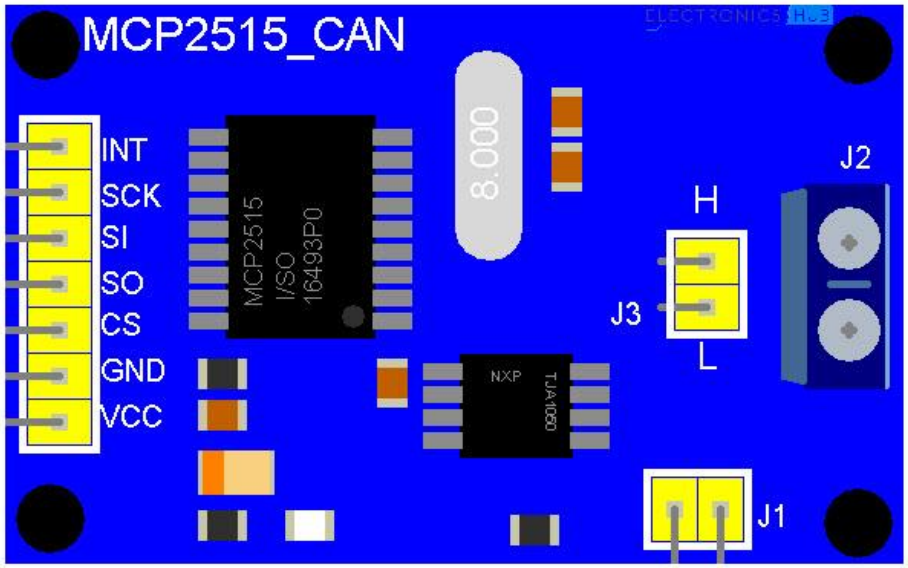


Figure III-11 :broches du module MCP2515

Avant de commencer, nous avons besoin d'une bonne bibliothèque avec laquelle CAN peut travailler. [15]

SEEED Studio a une bibliothèque qui fonctionne très bien avec leur bouclier.

Je préfère en utiliser un d'AUTOWP arduino-mcp2515 qui utilise STRUCT pour l'envoi de messages. C'est très simple et facile à utiliser.

Remarque

La librairie prenant en charge le bus CAN et ses circuits connexes comme le MCP2551 et le MCP2515 de Microchip n'est pas disponible dans ISIS-PROTEUS donc ce n'est pas possible de simuler cette application par ce logiciel.

III.3 Présentation des Logiciels utilisés

III.3.1 Le logiciel Arduino

Le logiciel est un environnement de développement intégré (IDE) écrit en langage Java, totalement gratuit et facilement téléchargeable sur le site officiel de l'Arduino à savoir www.arduino.cc. Le logiciel étant multiplateforme, télécharger la version correspondant à votre système d'exploitation. Une fois, le logiciel et ses pilotes, correctement installés, on peut donc programmer nos cartes Arduino. L'interface utilisateur du logiciel est présentée[15]

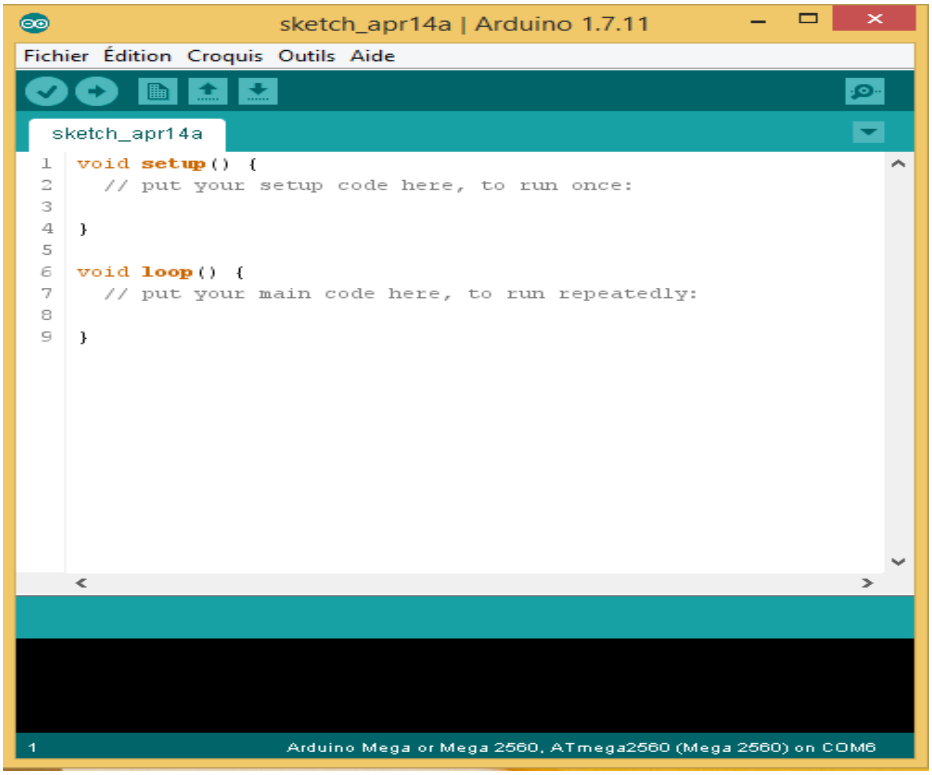


Figure III.12 : interface utilisateur du langage arduino

On peut voir deux fonctions qui se présentent automatiquement dès qu'on ouvre le logiciel : il s'agit de la fonction **setup ( )** et de la fonction **loop ( )**. En effet, tout programme Arduino, appelé sketch, doit contenir obligatoirement ces deux fonctions spécifiques. Même si, elles sont vides, le programme ne marche pas si au moins l'une d'entre elles est absente. La fonction **setup ( )** n'est exécutée qu'une seule fois juste après le lancement du programme. Elle contient généralement les instructions d'initialisation de certaines ressources de la carte telles que, par exemple, définition des lignes des ports parallèles soit en entrées ou en sorties, définition de la vitesse de fonctionnement du port série, etc…

Quant à la fonction **loop ( )**, elle est la boucle principale et se répète indéfiniment tant que l'Arduino restera sous tension. En d'autres termes, suite à un reset au moyen de son poussoir ou suite à une mise sous tension qui a pour effet de provoquer un reset automatique, l'Arduino exécute une seule fois les instructions contenues dans la fonction **setup ( )** puis exécute ensuite indéfiniment les instructions contenues dans la fonction **loop ( )** de ce même programme. Ces fonctions ne retourne pas de résultat, elles sont donc déclarées avec le mot cl «**void**».

Selon le cas, une troisième partie peut ou non être présente dans un programme Arduino mais ne contient pas d'instructions exécutables. Il s'agit de la zone de définition de constantes au

moyen du mot clé « **define**» ou de « **const**», d'inclusion d'éventuelles bibliothèques utilisés par le programme au moyen du mot clé « **include**», définition de variables, etc... Cette partie, si elle est présente, se place avant la fonction **setup ( )**, qui elle-même se place avant la fonction **loop ( )**. Tout comme le langage C, le langage arduino est composé de fonctions arithmétiques et mathématiques, des opérateurs de comparaison et opérateurs logiques, les structures de contrôle (prise de décision, boucles et sauts), des fonctions de gestion du temps et des entrées/sorties et beaucoup d'autres fonctions spécifiques à l'Arduino (gestion du port série, gestion d'interruption, générateur de nombre aléatoire, jouer des notes de musique, etc…).[15]

III.3.2 Logiciel LABVIEW

III.3.2 .1Introduction à la programmation graphique

LABVIEW (Laboratory Virtual Instrument Engineering Workbench) est un langage de programmation dédié au contrôle d'instruments et l'analyse de données. Contrairement à la nature séquentielle des langages textuels, LABVIEW est basé sur un environnement de programmation graphique utilisant la notion flot de données pour ordonnancer les opérations. Enfaite il existe deux formes de programmation graphiques:

Une programmation dite flot de contrôle et une autre dite flot de donnés. [16]

III.3.2 .1.1 programmation flot de contrôle

Ils ont longtemps été utilisés pour décrire les algorithmes, ces représentations décrivent les programmes comme étant de nœud de calcul connecté par des arcs spécifiant quel calcul doit être effectué ensuite

Ex : Grafcet et Réseau de petri.

III.3.2 .1.2Programmation flot de donné

C'est une fonction analogue à la propagation du signal à travers un circuit électrique. Le digramme flot de données est un graphe acyclique qui peut être composé de 3 éléments suivants :

• Des terminaux : qui sont les liens avec l'extérieur qui représente la production où la consommation de données.

• Des Nœud qui sont les traitements à effectué et qui sont représenté par une figure géométrique pouvant contenir une image illustrant leur fonctionnalité [16]

• Les arcs orienté : qui relie les nœuds et les terminaux et permettent d'indiqué le passage de données d'un nœud vers un autre.

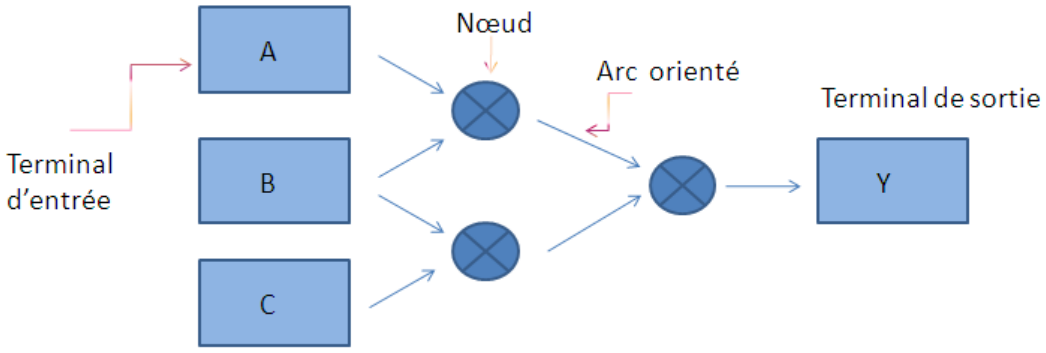


Figure III.13 : Exemple de diagramme de flot de donnés

III.3.2 .2Domaine d'utilisation

LAB VIEW est un outil d'acquisition, d'analyse et de présentation de données.

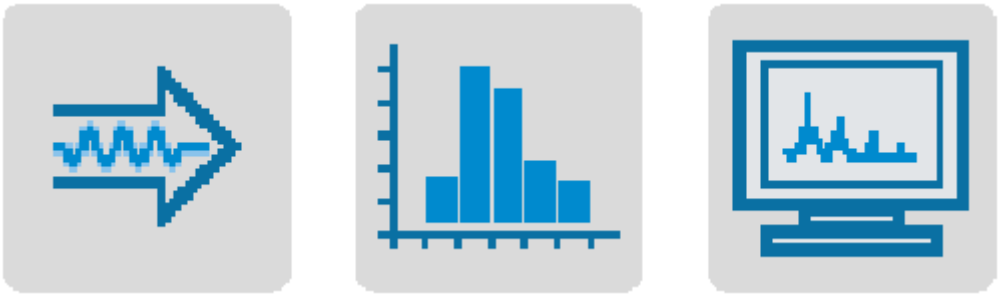


Figure III-14 : domaine d'utilisation

III.3.2 .3Acquisition

LABVIEW permet l'acquisition de données par l'intermédiaire de diverses connectiques :

- PCI (Peripheral Component Interconnect)
- Compact Flash
- LAN (Local Area Network)
- etc,

III.3.2 .4Analyse

LABVIEW inclut des outils pour l'analyse des données :

▪Traitement du signal : Convolution, analyse spectrale, transformées de Fourier,…

▪Traitement d'images : Masque, détection de contours, profils, manipulations de pixels,…

▪Mathématiques : Interpolation, statistiques (moyennes, écart-type,…), équations

▪différentielles, etc.

III.3.2 .5Présentation des données

LABVIEW inclut des outils d'aide à la présentation (communication)des données :

▪Graphiques, tableaux, images, génération de rapport,…

▪Par l'intermédiaire d'Internet : outils de publication web, serveur data socket, TCP/IP, envoie d'alertes par email,… [16]

III.3.2 .6L'environnement LABVIEW

LABVIEW est centré autour du principe d'instrument virtuel (Virtual Instrument ou encore VI). IL se décomposer en deux parties :

• la première partie (partie cachée ou interne) : elle contient l'algorithme du programme décrit sous la forme d'un diagramme flot de données en langage graphique.

• la seconde partie (partie visible) est constituée de l'interface utilisateur.

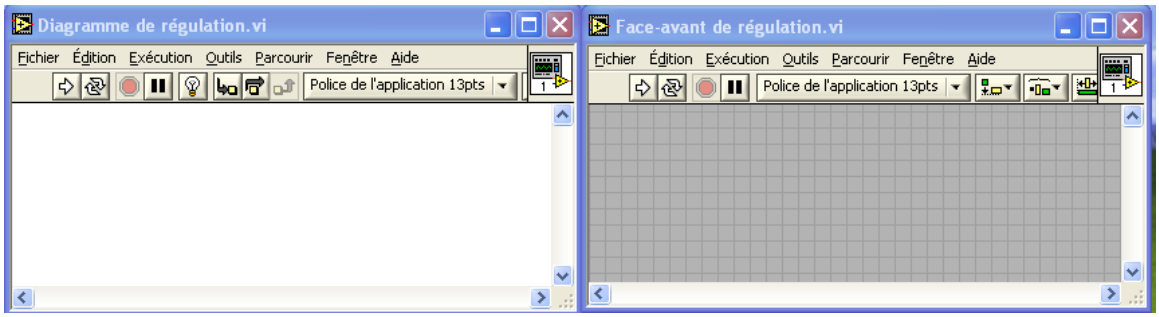


Figure III.15 : Fenêtre de l'environnement de développement sur LABVIEW
Face avant (à droite) et Diagramme (à gauche)

Pour écrire un programme sur LABVIEW, on a besoins des « Palettes » qui nous offre la possibilité de modifier la face avant et le digramme de LABVIEW, on trouve trois palettes : [16]

• **Palette d'outils**

Elle est disponible sur la face-avant et sur le diagramme, elle contient les outils nécessaires pour faire fonctionner et modifier la face avant et les objets du diagramme.

Figure III-16:palette d'outils

• Palette de commandes

Elle est disponible uniquement sur la face-avant, elle contient les commandes et les indicateurs de la face-avant nécessaire pour créer l'interface utilisateur.

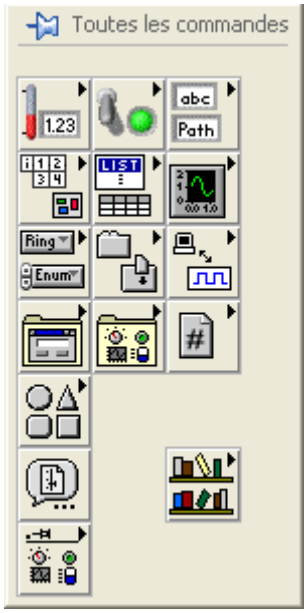


Figure III-17**:** palette de commandes

• **Palette de fonctions**

Elle est disponible uniquement sur le diagramme. Elle contient les objets nécessaire pour la programmation graphique comme les opérations d'arithmétique, d'E/S d'instrument, d'E/S de fichier et d'acquisition de données.

Figure III-18: palette de fonctions

III.3.2 .6.1Structure de données dans LABVIEW

LABVIEW utilise un langage fortement typé et toutes données ou structure de données ne peuvent être manipulées qu'avec des fonctions admettant ce type, en faite dans LABVIEW on trouve les types de base scalaire, les types entiers (signés ou non, codés sur 8, 16 ou 32 bits), le type réel (codé sur 16, 32 ou 64 bits), le type booléen et le type chaîne de caractères (figure III.19**)**. Il est important de noter que les éléments représentant ces données, ainsi que les liaisons issues de ces éléments, sont de forme et de couleur différente. [16]

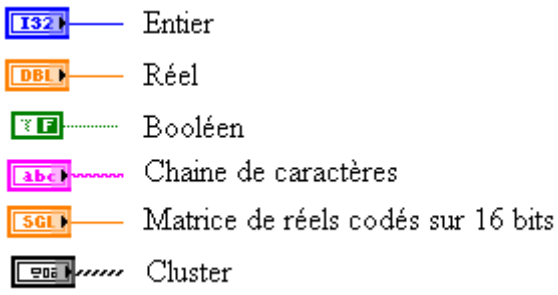


Figure III. 19 :différents types de structures de données dans LABVIEW

Le langage permet aussi de créer des structures de données plus élaborées.

III.3.2 .6.2- le type tableau (structure de données homogènes)

Comme tous les langages de programmations la présence de notion de tableau est obligatoire, puisqu'elle définit un outil de base pour le stockage et les traitements des informations.

La bibliothèque tableau dans LABVIEW est riche en fonctions prédéfinit, tel que, l'initialisation, l'indexation, l'inversement d'un tableau, la concaténation de deux tableau, et plusieurs autre fonctions, bien sur, la notion des tableaux peut être élargie pour contenir les matrices. Et comme toutes les autres bibliothèques, on peut l'enrichir avec des fonctions ou des procédures créent à partir des langages de bas niveau comme le langage C. [16]

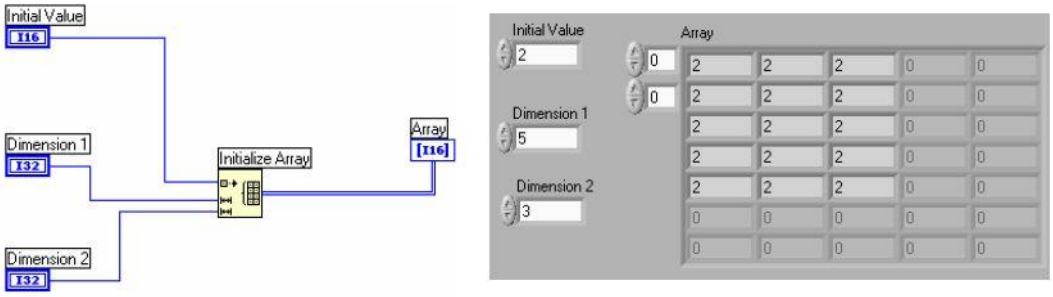


Figure III-20: Exemple d'initialisation d'un tableau a 2 dimensions (matrice)

III.3.2 .6.3-le type « cluster » (structure de données hétérogènes)

Ce second constructeur de type est l'équivalent du « record » en Ada ou du « struct » en C. Les différents composants ou champs de ce groupe de données (cluster) peuvent être assemblés ou récupérés par des fonctions spécifiques.

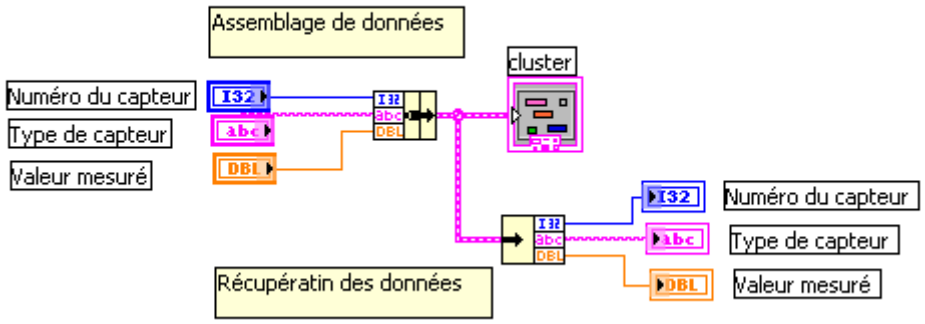


Figure III-21 : Exemple de manipulation des données avec le type cluster

III.3.2 .6.4 Structures de programmation

LABVIEW utilise un langage flot de données pur qui a été enrichi de quatre types de structures : la séquence, deux structures d'itération (la boucle« Pour » avec un nombre d'itérations fixé et la boucle « Tant Que » avec un nombre d'itérations soumis à condition) et la structure de choix. [16]

III.4.1- structure de séquence

La structure de « séquence » permet de spécifier l'ordre d'exécution de flots de données. Cette structure se présente sous la forme d'un cadre et a le statut d'un nœud

Exemple :

Sur la figure (III-22), l'utilisation de la séquence s'impose : il s'agit d'initialiser un port série puis d'envoyer une chaîne de caractères. Le premier flot de données risque de provoquer des erreurs car rien n'empêche le nœud d'écriture sur le port série de s'exécuter avant le nœud d'initialisation de ce port : l'utilisation de la séquence permet d'ordonner correctement les opérations.

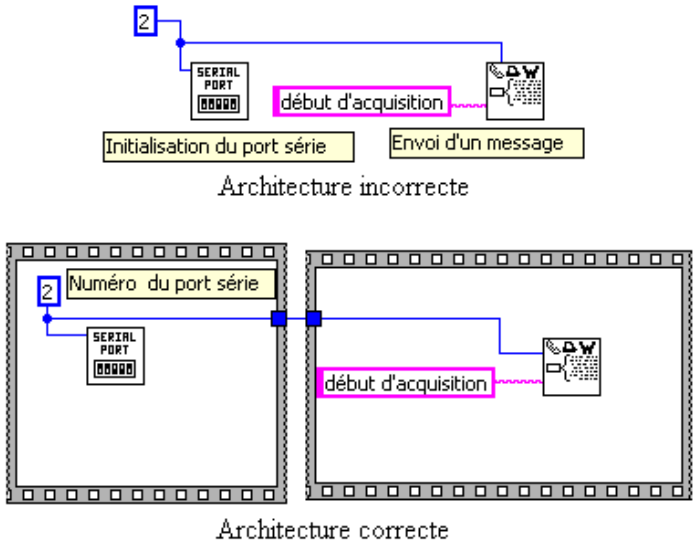


Figure III-22 : Exemple d'utilisation de la structure de séquence

III.3.2 .6.5- les structures itératives

Les deux structures itératives, la boucle « Pour » et la boucle « Tant que », ont aussi le statut d'un nœud ordinaire. La boucle « Pour » permet d'exprimer la répétition (ou itération) pour un nombre de fois prédéterminé dé fini par une connexion d'entrée obligatoire: le nombre

d'itérations à effectuer N. À l'intérieur de la boucle « Pour » se trouve un terminal d'entrée local générant l'entier indiquant l'indice d'itération de la boucle (i varie de 0 à N–1).

Exemple :

L'utilisation de cette structure, présentée sur la figure (III.23), concerne l'acquisition de N mesures avec un temps fixe entre chacune, défini par un nœud de temporisation Les N données de sortie sont assemblées pour former un vecteur avec une auto-indexation**.**

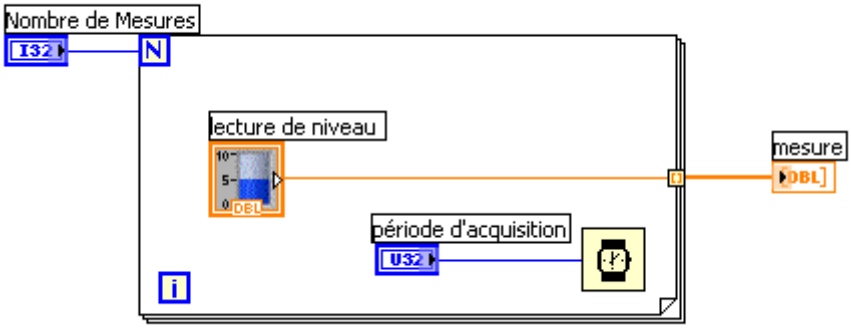


Figure III-23**: Exemple d'utilisation de la structure itérative "pour"**

La boucle « Tant Que » permet d'exprimer la répétition pour un nombre de fois non connu à l'avance. À l'intérieur de la boucle « Tant Que » se trouve un terminal d'entrée local générant l'entier indiquant l'indice d'itération de la boucle. Un terminal de sortie de type booléen permet d'arrêter la boucle lorsque la valeur « False » lui est envoyée. [16]

Exemple :

L'utilisation précédente décrite sur la figure III-24, est reprise avec cette structure « Tant Que » en arrêtant l'acquisition pour une valeur trop grande de la mesure (Figure III-24).

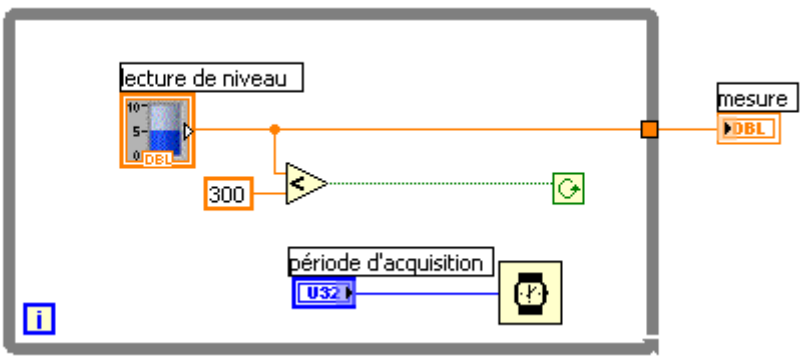


Figure III-24 Exemple d'utilisation de la structure itérative "tant que"

Dans les deux cas de structure itérative lorsque des tableaux de données arrivent ou partent des tunnels, ceux-ci ont la possibilité d'être automatiquement indexés : récupération un par un des éléments du tableau ou concaténation pour créer un tableau. Dans le cas des deux structures, il est possible de mémoriser des résultats produits lors d'itérations antérieures et de les récupérer ensuite. Ces registres à décalage sont des variables locales à une boucle. [16]

Exemple :

La figure (III.25) présente une utilisation de ces registres à décalage dans le cas de la boucle « Pour ». Il est important de remarquer que ces registres à décalage constituent un des effets de bord du langage. Il est de plus nécessaire d'initialiser ces mémoires locales pour réaliser une utilisation contrôlée et cohérente.

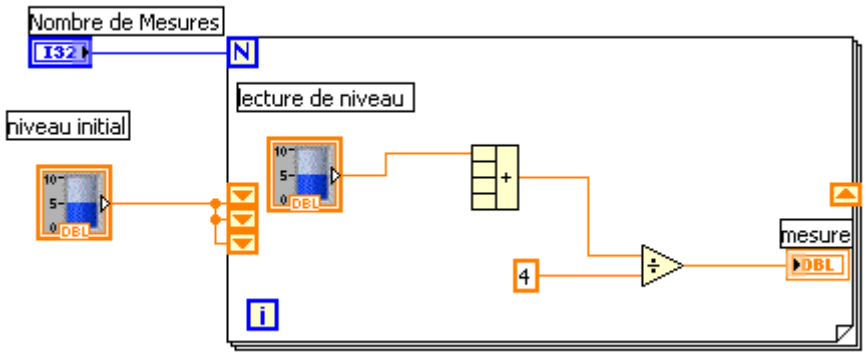


Figure III-25: Exemple d'utilisation des registres à décalage dans une boucle "pour".

III.3.2 .6.6- la structure de choix

La dernière structure nécessaire est celle qui va permettre d'exprimer l'alternative (le choix). La sélection du cas exécuté est faite par la valeur de la variable connectée à l'entrée représentée par un point d'interrogation (figure III.26). L'identifiant du cas représenté est indiqué en haut de la structure.

Exemple :

La figure (III.26) présente une utilisation de la structure de choix. Si nous avons une action « vrais » alors il' y a une acquisition de donné sur le port série, et si l'action est « faux », alors il' y aura pas d'acquisition.

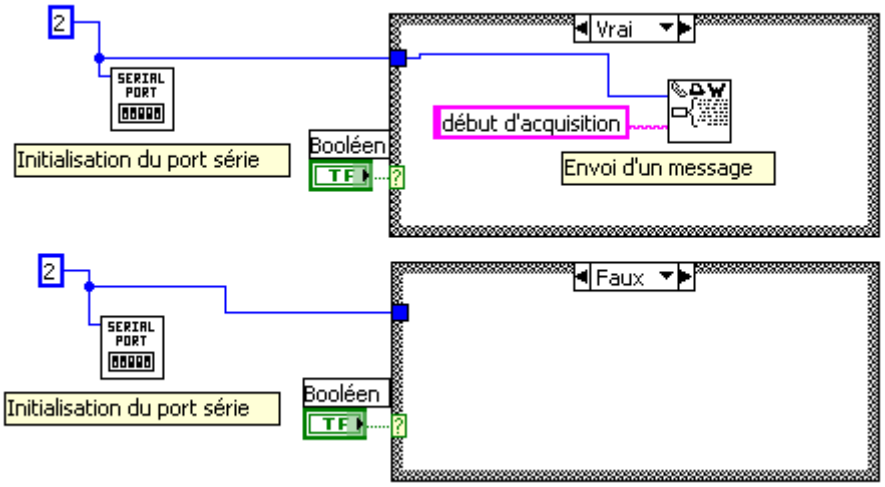


Figure III-26 : Exemple d'utilisation de structure de choix.

III.3.2 .7-Traitement numérique

III.3.2 .7.1: les fonctions prédéfinies

LABVIEW possède les instructions de base d'un langage de programmation permettant de traiter les différents types de données. Ainsi, nous avons des fonctions liées aux variables numériques (entiers, réels et complexes), aux variables booléennes, aux chaines de caractères, et aux tableaux. [16]

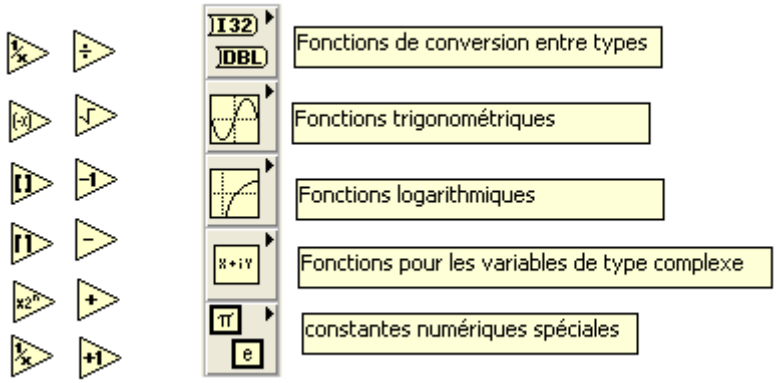


Figure III-27: Quelques instructions de traitement de données numérique dans LABVIEW

Nous trouvons aussi les opérateurs de test et de comparaisons liées à ces différents types de données. À partir des structures de contrôle, des fonctions et des opérateurs, de base, il est alors possible de traduire un algorithme quelconque et d'enrichir la bibliothèque des fonctions

en utilisant le mécanisme d'encapsulation. Un diagramme complet est alors réduit à un nœud qui peut être ensuite réutilisé.

III.3.2 .6.7.2: Boites à outils mathématique

LABVIEW contient aussi des boites de calcul mathématiques qui servent à introduire des commandes complexes tel que (linspace , ode45 ,sin, cos , etc..)

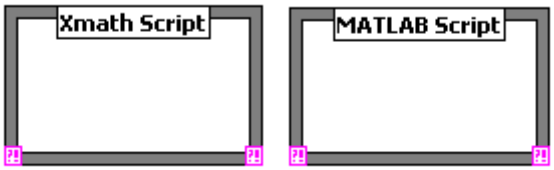


Figure III-28: Exemple de boite de calcul mathématique dans LABVIEW[16]

III.3.2 .8 Interfaçage LABVIEW-Arduino

Utilisez VISA **(NI Serial Communication)**

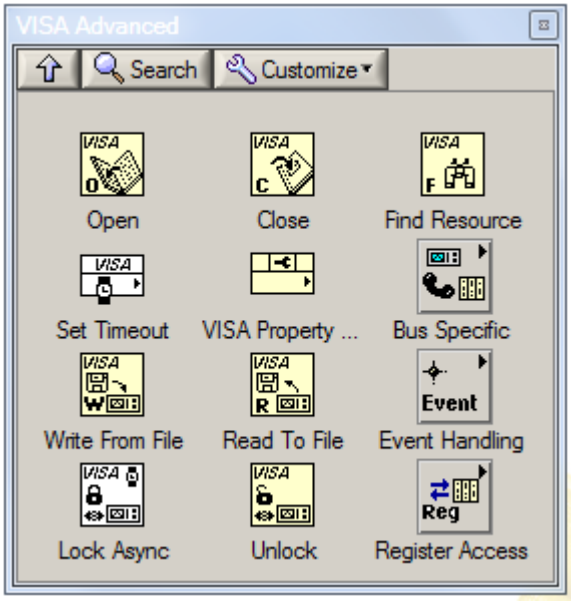


Figure III-29: palette VISA advenced

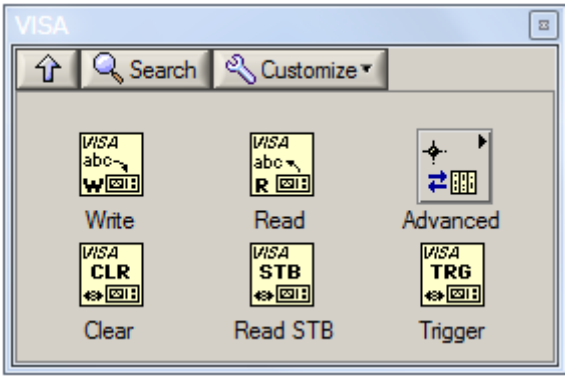


Figure III-30: palette VISA

III.3.2 .8.1NI-VISA

L'architecture logicielle des instruments virtuels (VISA) est une norme pour la configuration, la programmation et le dépannage des systèmes d'instrumentation comprenant des interfaces GPIB, VXI, PXI, série, Ethernet et / ou USB.

VISA fournit l'interface de programmation entre le matériel et les environnements de développement tels que LABVIEW, LabWindows / CVI et Measurement Studio pour Microsoft Visual Studio [17]

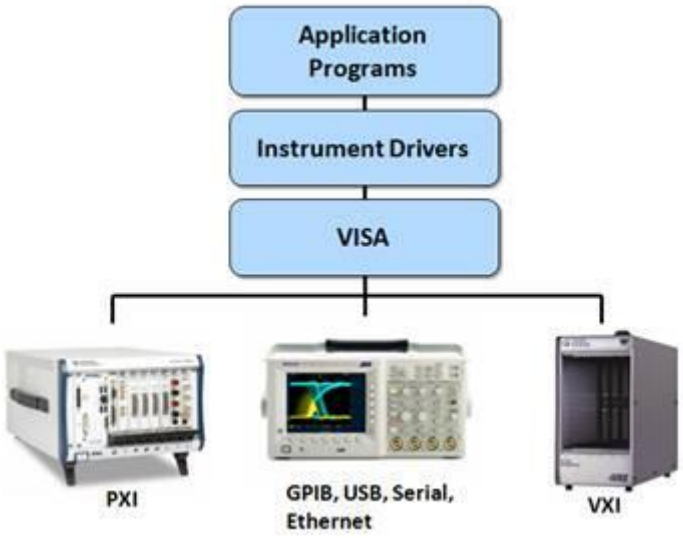


Figure III-31 : l'interface de programmation entre le matériel

III.3.2 .8.2 Quelques schémas fonctionnels importants de la palette VISA

▪VISA Open

Ouvre une session sur le périphérique spécifié par le nom de la ressource VISA et renvoie un identificateur de session qui peut être utilisé pour appeler toute autre opération de ce périphérique.

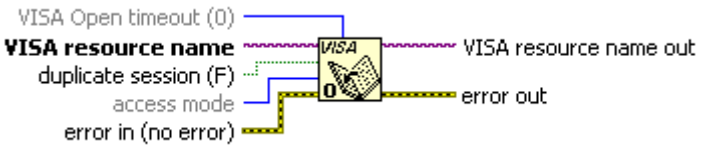


Figure III-32 : VISA Open.vi

▪VISA Read

Lit le nombre d'octets spécifié à partir du périphérique ou de l'interface spécifié par le nom de la ressource VISA et renvoie les données dans la mémoire tampon de lecture.

Figure III-33 : VISA Read.vi

▪VISA Write

Écrit les données du tampon d'écriture sur le périphérique ou l'interface spécifié par le nom de ressource VISA.

Figure III-34: VISA Write.vi

▪VISA Close

Ferme une session de périphérique ou un objet événement spécifié par le nom de ressource VISA.

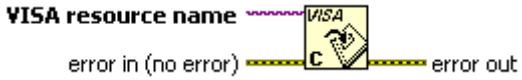


Figure III-35: VISA Close.vi

III.3.2 .8.3 Example NI-VISA

```
void setup() {
// initialise la communication série à 9600 bits par seconde:
Serial.begin(9600);
}
// la routine de boucle s'exécute encore et encore pour toujours:
voidloop() {
// lire l'entrée sur la broche analogique 0:
intsensorValue = analogRead(A0);
// affiche la valeur que vous lisez:
Serial.println(sensorValue);
delay(100);   // délai entre les lectures pour la stabilité}
```

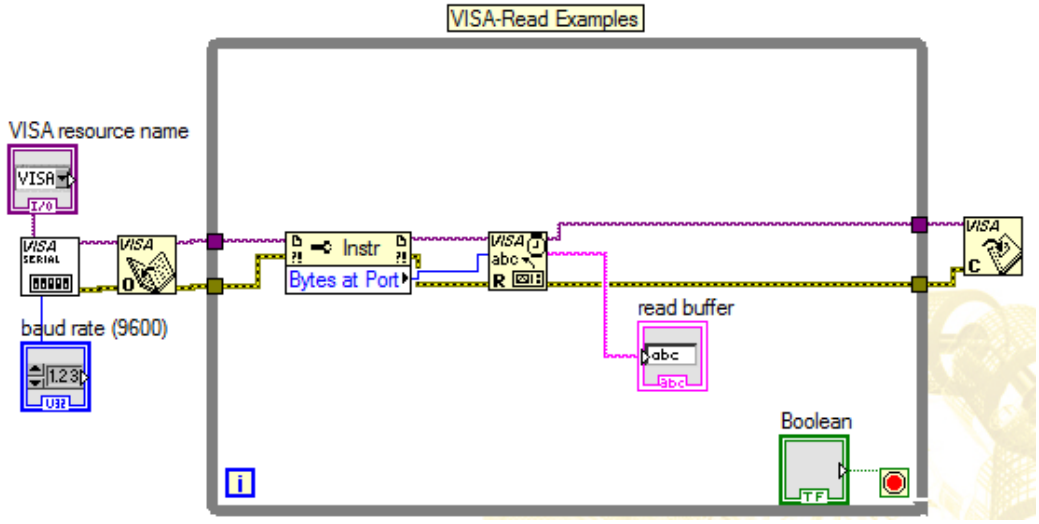


Figure III-36 :exemple d'un VISA Read[17]

III.4 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 2 nœuds qui démontrent l'intérêt de ce bus de terrain et un autre nœud (troisième nœud) relier avec deuxième nœud par un port série :

-Le premier 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 vitesse de rotation de moteur.

-Le deuxième nœud affiche les données des différents capteurs dans un afficheur lcd

-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. [18]

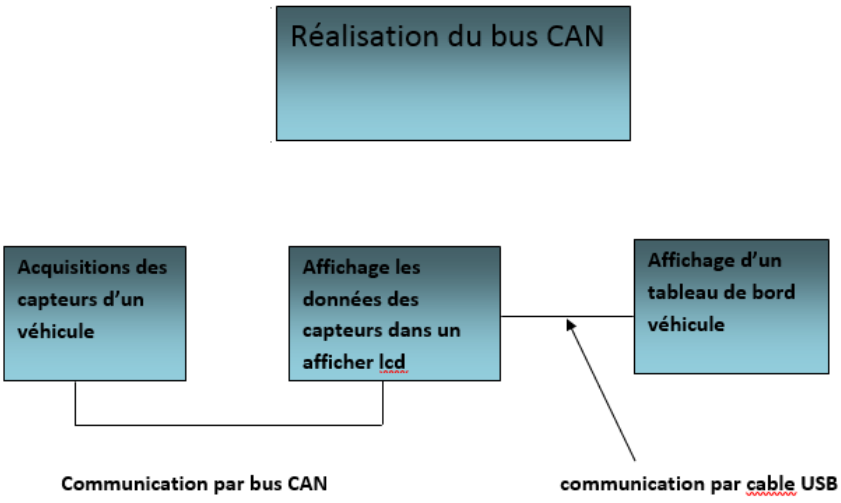


Figure III-37: présentation de la réalisation

Nous établissons des étapes 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 :

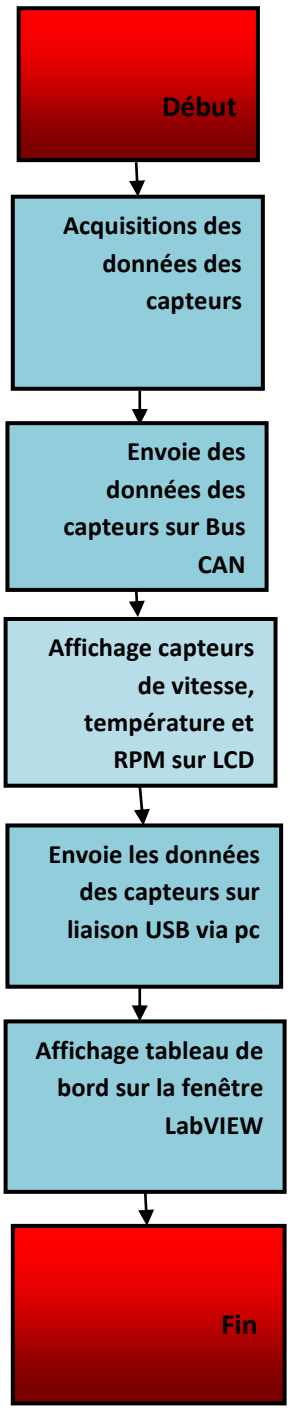


Figure III-38 : organigramme de la réalisation

III.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 d'affichage des données capteurs dans un afficheur lcd,
- La partie Tableau de bord réalisée à partir du logiciel N.I LABVIEW. [18]

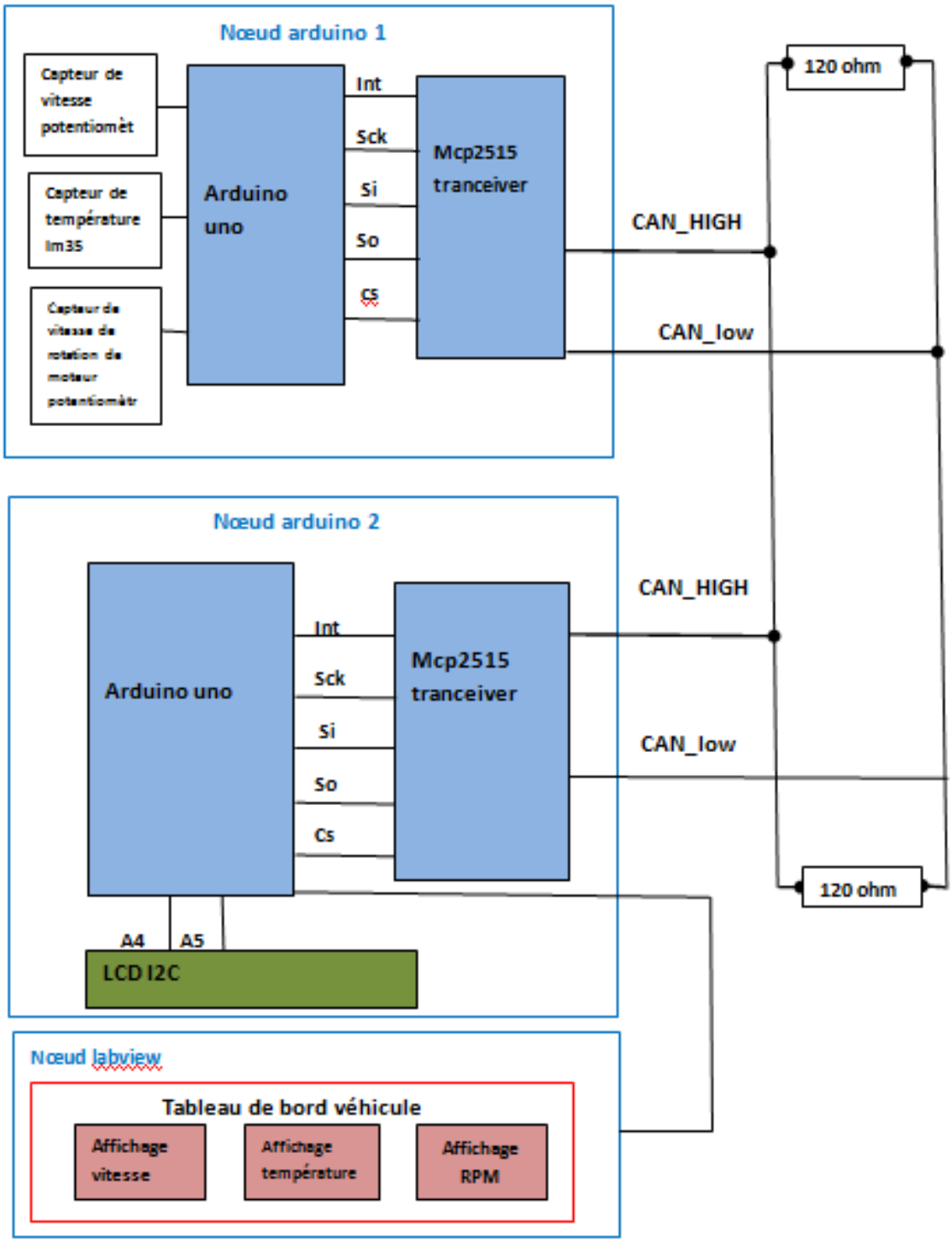


Figure III-39: Schématique de communication de la réalisation

III.6 Circuit de la réalisation

Pour commencer, et afin de prendre en main la communication CAN via ArduinoUno, la première étape c‟est de réussir une communication entre deux arduino de même type.

L‟un des deux arduinos fonctionne émetteur et l‟autre en récepteur

Dans l‟application schématisée par la Figure ci-dessous, nous avons un réseau CAN constitué de deux nœuds, où la communication sur le bus CAN est configurée de sorte que le premier dispositif envoie des messages contient les données des capteurs et le deuxième reçoit les données [18]

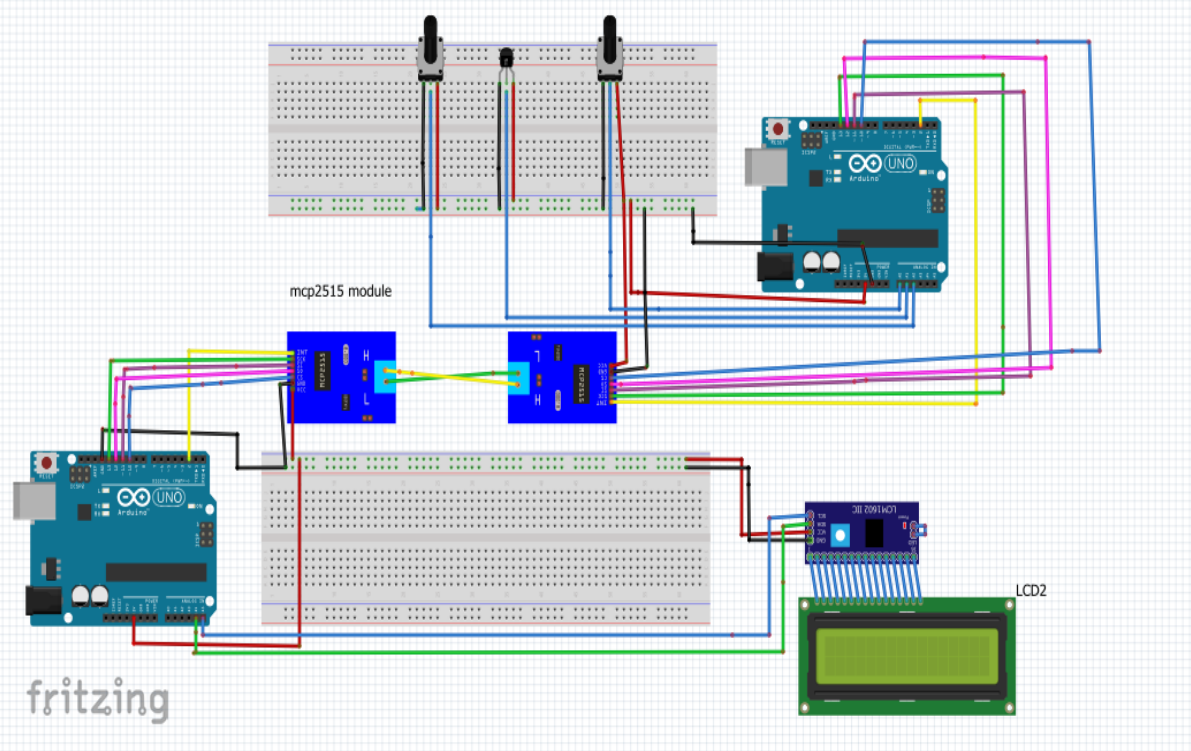


Figure III-40 : Circuit de la réalisation

III.7 Code Programme émetteur

ADXL3xx | Arduino 1.8.1

Fichier Édition Croquis Outils Aide

ADXL3xx

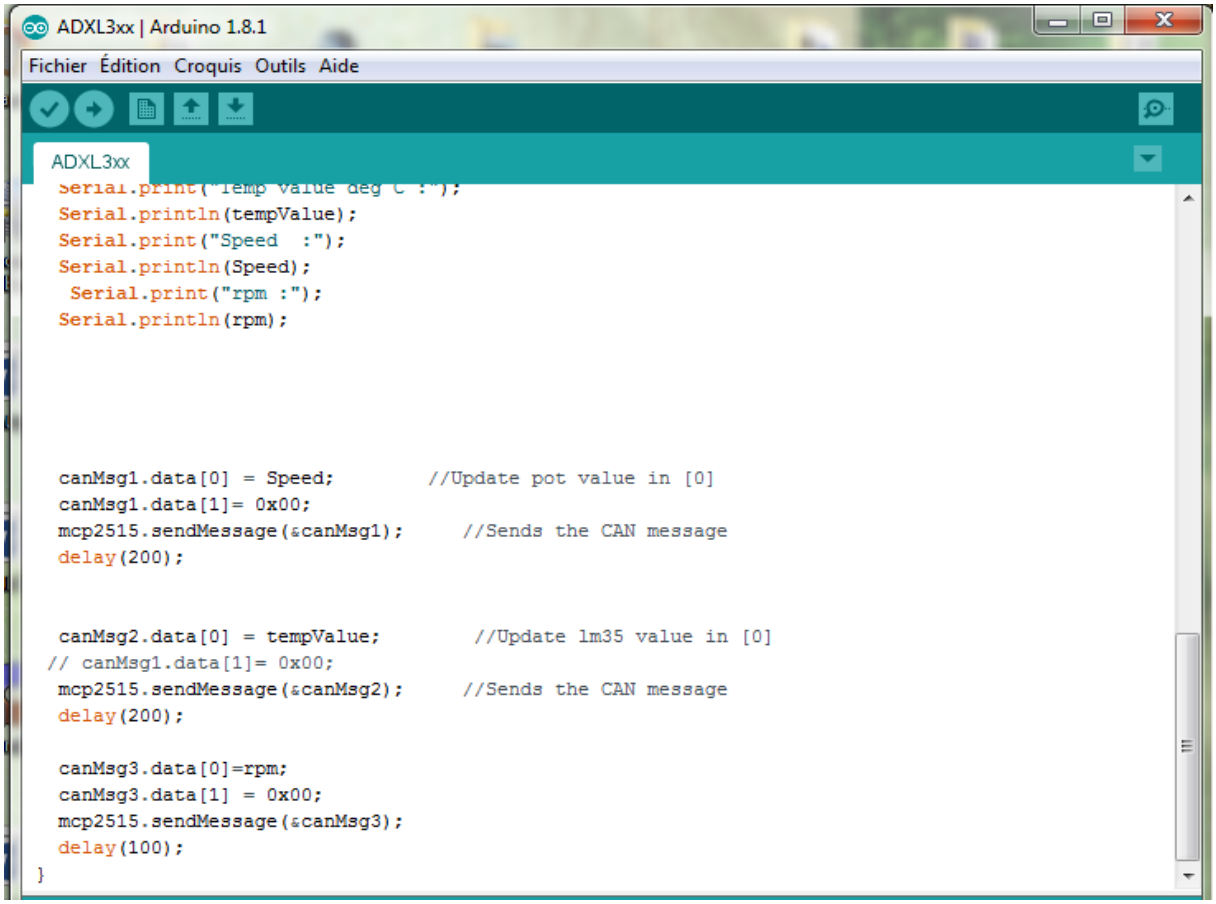
```
#include <SPI.h>          //Library for using SPI Communication
#include <mcp2515.h>      //Library for using CAN Communication

#define speedPin A0
#define lm35Pin A1
#define rpmPin A2

int Speed=0;
float tempValue; // variable to store celcius
int rpm;
struct can_frame canMsg1; //Speed message
struct can_frame canMsg2;  // lm35 message
struct can_frame canMsg3;  // rpm message
MCP2515 mcp2515(10); // chip select pin 10


void setup()
{

  Serial.begin(9600);
  SPI.begin();               //Begins SPI communication

  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ); //Sets CAN at speed 500KBPS and Clock 8MHz
  mcp2515.setNormalMode();
```

ADXL3xx | Arduino 1.8.1

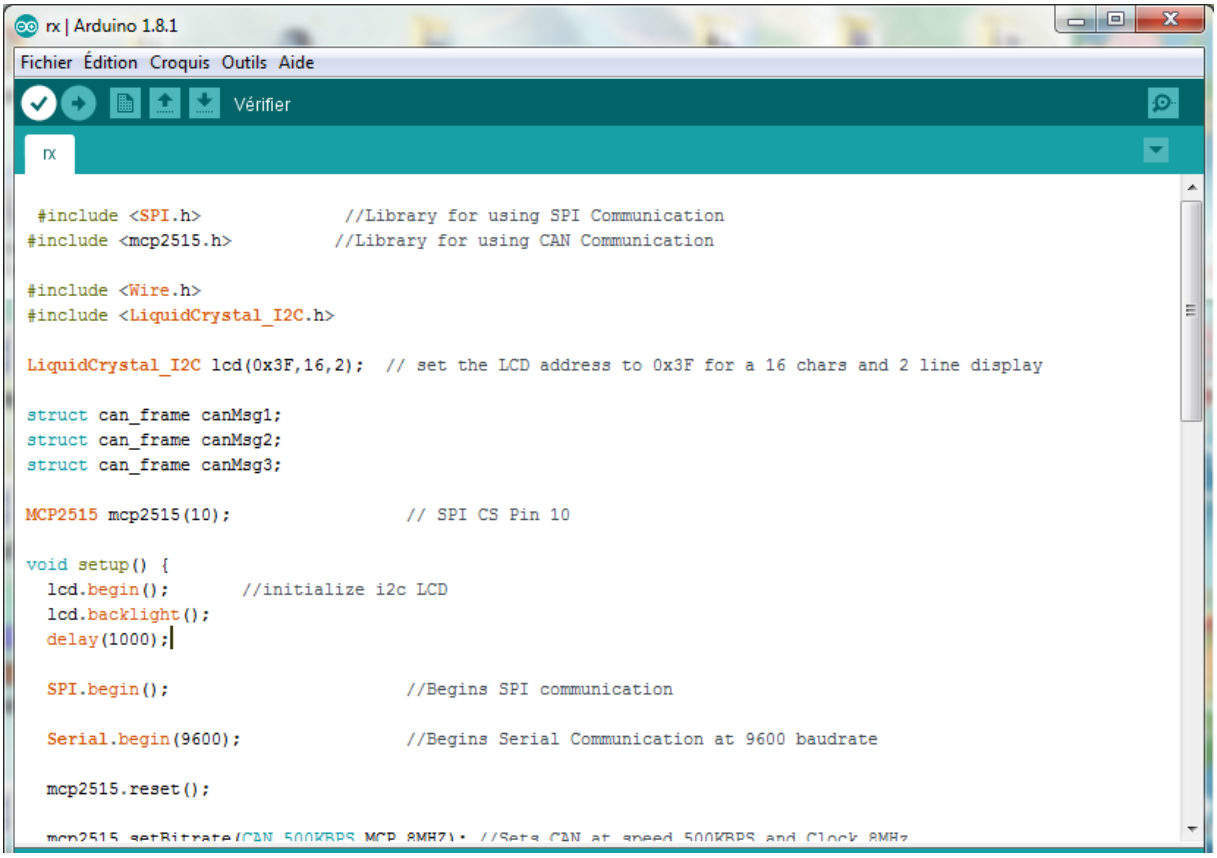
Fichier Édition Croquis Outils Aide

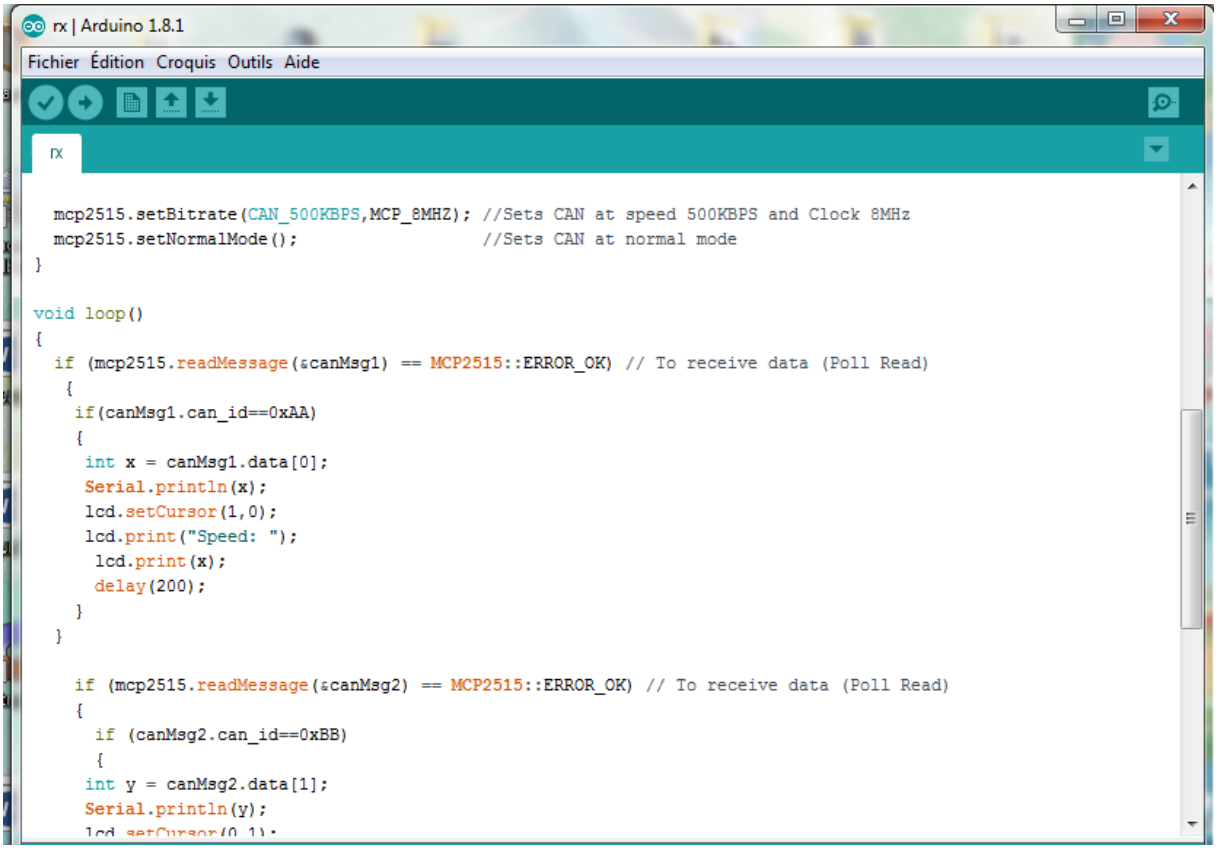
ADXL3xx

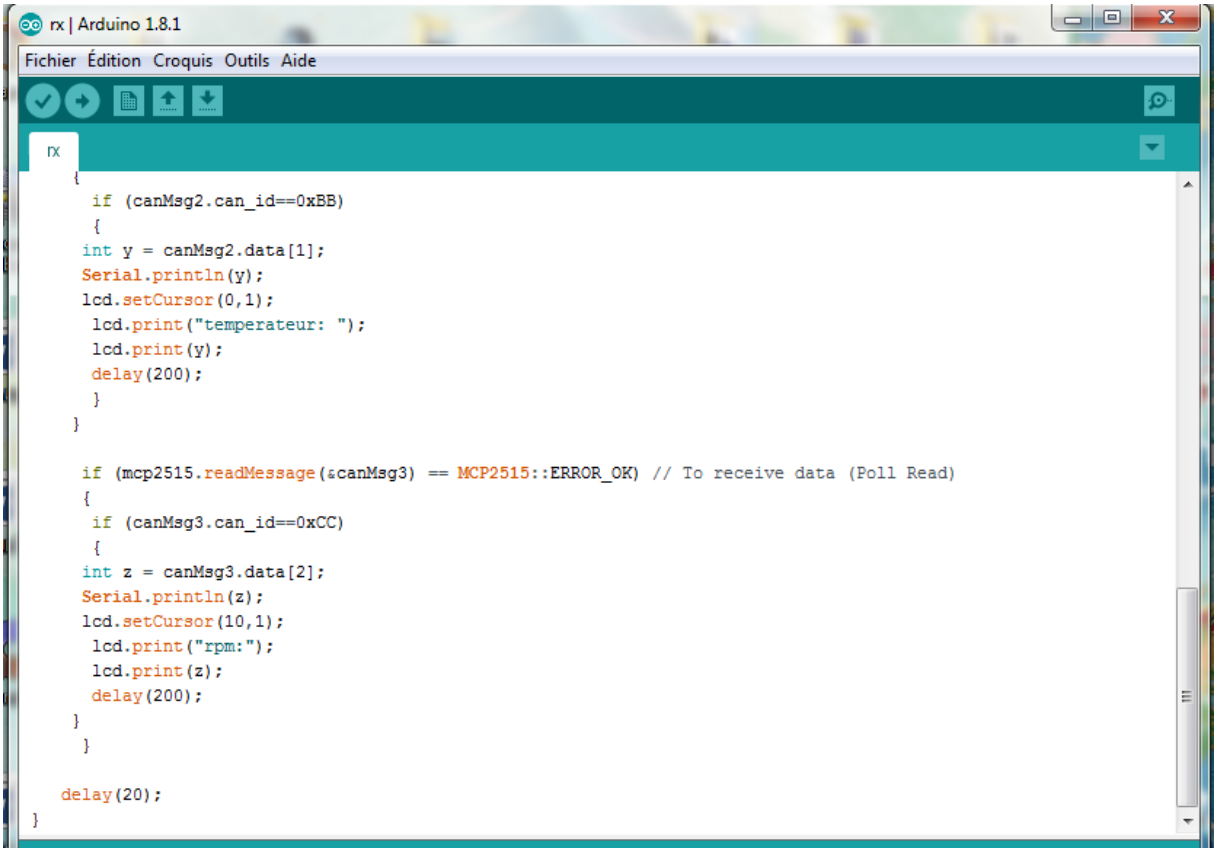
```
  canMsg1.can_id  = 0xAA;           //CAN id as 0xAA for potentiometer
  canMsg1.can_dlc = 2;               //CAN data length as 2

  canMsg2.can_id  = 0xBB;           //CAN id as 0xBB for LM35
  canMsg2.can_dlc = 1;               //CAN data length as 1

  canMsg3.can_id  = 0xCC;           //for rpm
  canMsg3.can_dlc = 2;


}

void loop()
{
  Serial.print(canMsg1.can_id);
  Speed = analogRead(speedPin);
  tempValue=analogRead(lm35Pin);
  Speed=map(Speed,0,1023,0,255);
  tempValue=(tempValue/1023)*5000;
   rpm=analogRead(rpmPin);
  rpm=map(rpm,0,1023,0,255);

  Serial.print("Temp value deg C :");
  Serial.println(tempValue);
  Serial.print("Speed  :");
  Serial.println(Speed);
```



III.8 Code Programme récepteur


rx | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
Vérifier
rx
#include <SPI.h> //Library for using SPI Communication
#include <mcp2515.h> //Library for using CAN Communication
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x3F,16,2); // set the LCD address to 0x3F for a 16 chars and 2 line display
struct can_frame canMsg1;
struct can_frame canMsg2;
struct can_frame canMsg3;
MCP2515 mcp2515(10); // SPI CS Pin 10
void setup() {
lcd.begin(); //initialize i2c LCD
lcd.backlight();
delay(1000);
SPI.begin(); //Begins SPI communication
Serial.begin(9600); //Begins Serial Communication at 9600 baudrate
mcp2515.reset();


rx | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
rx
mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ); //Sets CAN at speed 500KBPS and Clock 8MHz
mcp2515.setNormalMode(); //Sets CAN at normal mode
}
void loop()
{
if (mcp2515.readMessage(&canMsg1) == MCP2515::ERROR_OK) // To receive data (Poll Read)
{
if(canMsg1.can_id==0xAA)
{
int x = canMsg1.data[0];
Serial.println(x);
lcd.setCursor(1,0);
lcd.print("Speed: ");
lcd.print(x);
delay(200);
}
}
if (mcp2515.readMessage(&canMsg2) == MCP2515::ERROR_OK) // To receive data (Poll Read)
{
if (canMsg2.can_id==0xBB)
{
int y = canMsg2.data[1];
Serial.println(y);



III.9 Réalisation avec LABVIEW

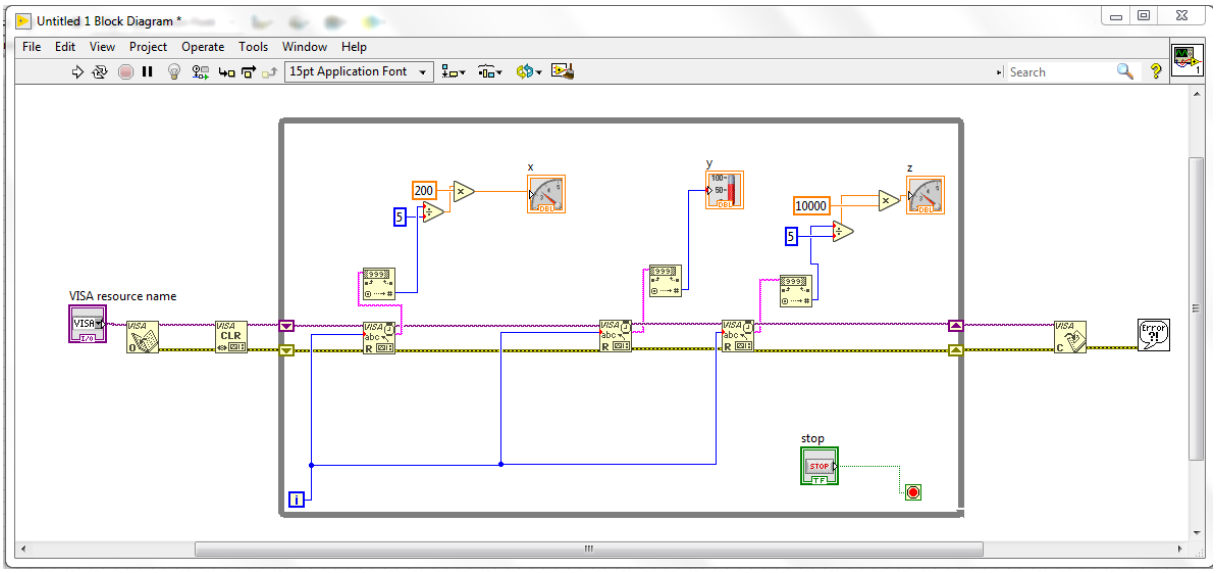


Figure III-41 : block diagram

Ces différentes informations qui arrivent par le port USB 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. Et affiche les valeurs des capteurs sur le tableau de bord :

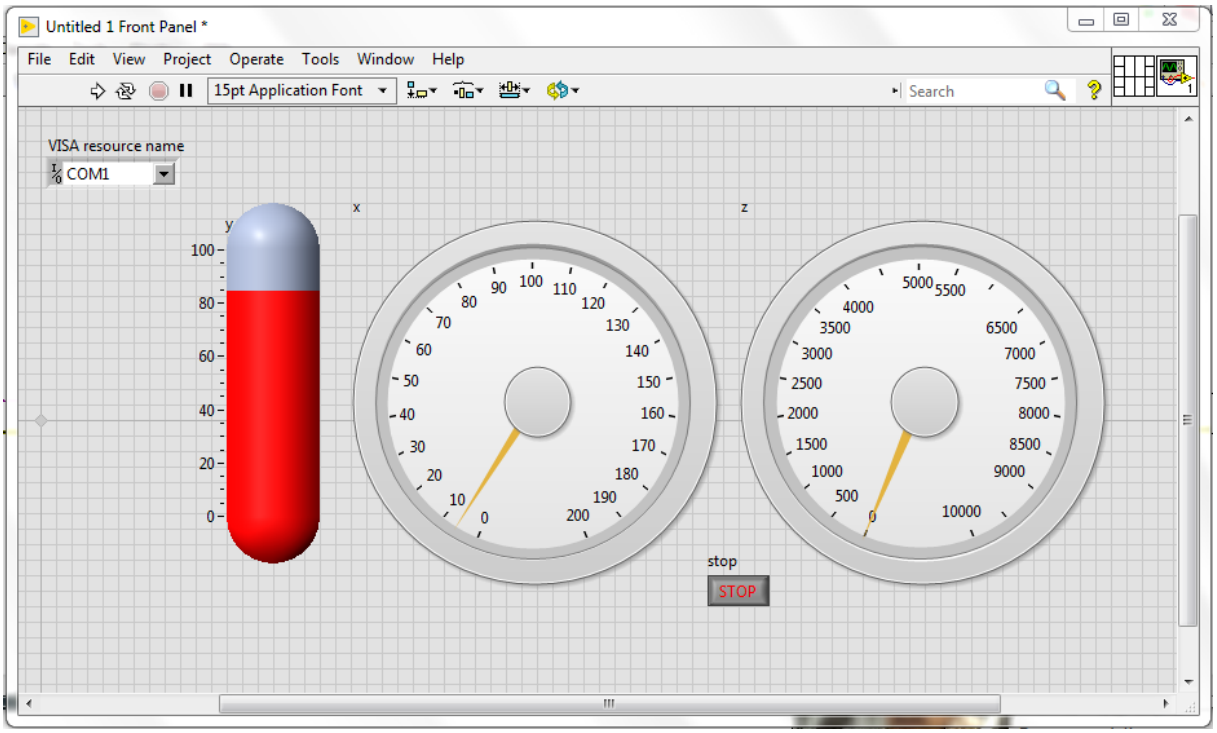


Figure III-42: Face avant LABVIEW (Tableau de bord véhicule)

Y=température (celiciuce)

X= vitesse (km/h)

Z=RPM(vitesse de rotation)(tr/min)

III.10CONCLSION

Dans ce chapitre, nous avons réussi de faire la conception et la réalisation d'une communication entre DEUX ARDUINO via bus CAN. nous avons mesuré la température a partir d'un lm35 et la vitesse de véhicule et la vitesse de rotation de moteur branchés sur un nœud1, ce dernier envoie les valeurs mesurée pour l'affichage dans un LCD sur un deuxième nœud2,ce dernier envoie les valeurs aussi pour l'affichage dans un ordinateur utilisant l'ogiciel LABVIEW

L'absence de la simulation et des cartes de développement MCP 2515 sur proteus, ont rendu difficile le débogage des d' application. Mais malgré ces difficultés rencontrées ; nous avons réussi d'établir une communication CAN et voire son fonctionnement en pratique.

Conclusion Générale

Conclusion général

Les systèmes embarqués sont donc des outils très puissants pouvant proposer une multitude de fonctionnalités et réaliser une quantité incalculable de tâches. Cependant, ces systèmes ne peuvent garantir une fiabilité de fonctionnement et une sécurité maximale à ses utilisateurs due à sa complexité de conception et son environnement constamment instable.

Ce travail démontre l'appartenance et le rattachement des systèmes embarqués dans le domaine de l'automobile avec une flagrante augmentation de leurs utilisations dans différents domaines, que ce soit la sécurité, le confort ou directement le fonctionnement du véhicule.

L'objectif de ce projet était d'étudier le principe, la conception et la réalisation de quelques applications utilisant le bus CAN. Dans ce sens, il a été judicieux de diviser le travail en trois chapitres; où le premier était une introduction aux réseaux informatiques et industriels.
Le deuxième était une présentation détaillée du réseaux CAN et ses différentes caractéristiques.
Dans le troisième chapitre, quelque applications du bus CAN ont étaient présentées en utilisant quelques informations issues des chapitres précédents, pour réaliser la communication entre des arduino uno via bus CAN.

Suite à l'étude et la manipulation des différents aspects et composants des applications CAN vues, nous avons pu comprendre quelques notions techniques, à savoir :

- La réalisation d'une application embarquée ; le choix des composants Hardware et Software (la programmation d'un arduinouno avec un logiciel Arduino IDE).

- La réalisation d'une communication CAN entre des microcontrôleurs.

- TRAITEMENT DES DON2ES DES CAPTEUR avec des conditions sous Labview.

Ce projet de fin d'étude nous a permis de découvrir le bus CAN, et de nous rendre compte de l'importance de ces réseaux dans quelques applications de la vie pratique. En effet, de nos jours les équipements dans un véhicule ou dans l'industrie se développent, et une communication entre les différents systèmes s'avère nécessaire.

Le prototype que nous avons réalisé préfigure un bus CAN dans des applications réelles, mais il est évident qu'un projet de cette nature ne se termine jamais.

Comme perspective de ce travail, on envisage d'étudier le bus CAN en utilisant plus d'outils de développement qui permettent la visualisation des déférentes trames et le débogage de l'application

. Donc, comme travail future, nous pouvons étendre ce travail dans une application automobile pour le diagnostique par exemple (OBD : On Board Diagnostic), ou réaliser une application temps réel pour la synchronisation de plusieurs machines.

Références

Références

[1]Benkhelifa, A. (2018). Les systèmes embarqués dans l'automobile (Doctoral dissertation, Haute école de gestion de Genève).

[2]Oubrahim, M. A. M. H., Tahiry, K., &Farchi, A. (2019, June). Architecture réseaux et électroniques embarqués automobile.

[3]https://www.fichier-pdf.fr/2012/12/20/technique-de-diagnostic-dans-le-domaine-automobile

[4]Jugurtha, F. (2010). Etude d'un système électronique embarqué et réalisation d'une simulation d'injection electronique de carburant (Doctoral dissertation, Université Mouloud Mammeri).

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

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

[7]Le Bus CAN – Patrice Kadionik – [En Ligne]

[8]Seminaire Industriel CAN – CFAI Languedoc Roussillon – [En Ligne] : http://www.cfai languedocroussillon.com/telechargement/Seminaire_Bus_Industriel_CAN.pdf

[9]Tavernier, Christian. Arduino Maîtrisez sa programmation et ses cartes d'interface (shields). Dunod, Paris, 2011**.**

[10]https://www.carnetdumaker.net/articles/mesurer-une-temperature-avec-uncapteur-lm35-et-une-carte-arduino-genuino/

[11]https://www.aranacorp.com/fr/utilisation-dun-potentiometre-avec-arduino/

[12]Manuel d'utilisation de l'afficheur I2C LCD 16x2

[13]https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/

[14]Arduino-https://www.arduino.cc – site officiel de l'Arduino.

[15]Guide d'utilisateur du logiciel LABVIEW

[16]Sugiarto, T., & PENERBANGAN, P. T. (2016). LABVIEW-Arduino Interfacing for Data Acquisition and Telemetry System. LAPAN UAV Development, Aeronautics Technology Center, Indonesian Institute of Aeronautics and Space.

[17]http://nationalinstrument.com