Dossier ressource pour les activités de la maquette didactique :

« BVR »

Le CAN

Controller Area Network

DIDAC BDH

Contenu

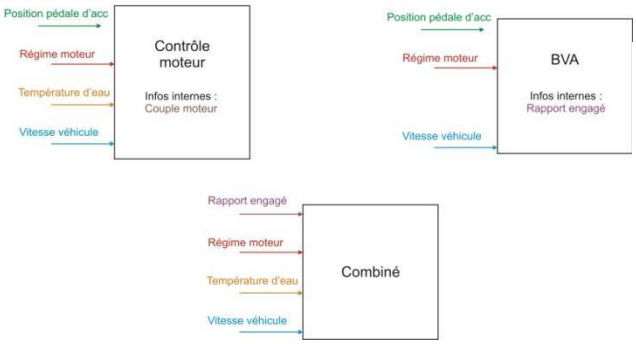
Principe des réseaux multiplexés 3
Différents types de réseaux multiplexés 4
Le CAN (Controller Area Network) 4
LE CAN dans le MODELE OSI 5
LA COUCHE PHYSIQUE 5
LA COUCHE LIAISON 6
LA COUCHE APPLICATION 6
Processus d'échange de données 7
Etude de la couche physique du CAN 8
Schéma de principe d'un module émetteur/récepteur : 8
Topologie du réseau : 10
Principe de fonctionnement du transceiver pour un réseau CAN low speed : 11
Processus de transmission des données 12
Niveaux de tensions pour le CAN low speed : 13
Etude de la couche liaison du CAN 14
Notions de protocole CAN 14
Composition d'une trame CAN : 15
Décodage d'une trame CAN : 16
Notion de priorité 18
Bit de stuffing 19
Etude de la couche application du CAN 20
Exemple de trame: 21
Principe de conversion des données : 22

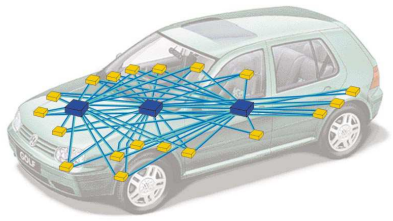
Principe des réseaux multiplexés

Les systèmes utilisent des informations identiques provenant des capteurs ou de données internes à chaque calculateur. Ainsi chaque calculateur peut obtenir ces informations :

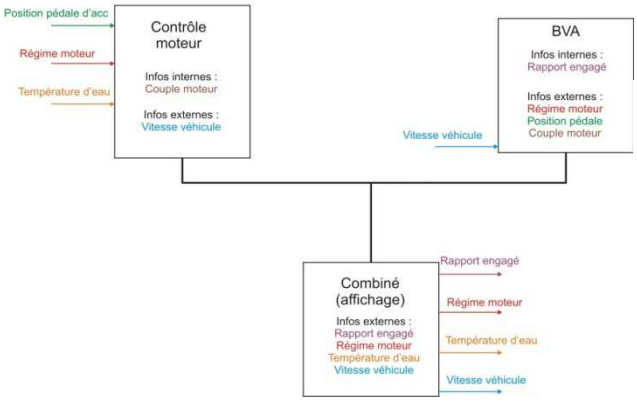
- en ayant un capteur pour chaque donnée nécessaire connecté à ses propres entrées, (Architecture ancienne)
- en échangeant ces données avec les calculateurs les ayant mesurées (ou calculées) : le multiplexage. (Architecture actuelle)

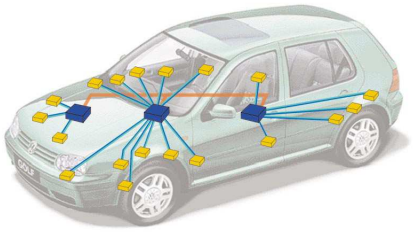
Architecture ancienne





Architecture actuelle





NB : l'architecture actuelle a amélioré le système car elle permet l'échange de données internes (calculées) qui sont ainsi traitées de la même façon qu'une donnée issue d'un capteur.

Il va donc falloir faire « circuler » ces données sur **un bus de communication** reliant les calculateurs entre eux.
Pour cela les données vont être échangées sous forme **numériques** (bits), à des **vitesses élevées**, **identifiées**, et **classées** par priorité.
Le support utilisé pour ces échanges peut être :

- filaire (fils de cuivre)
- fibre optique
- ondes (électromagnétique, infra rouge)

Sur les véhicules, les **bus** auront un support physique filaire (rapport coût/performance satisfaisant).

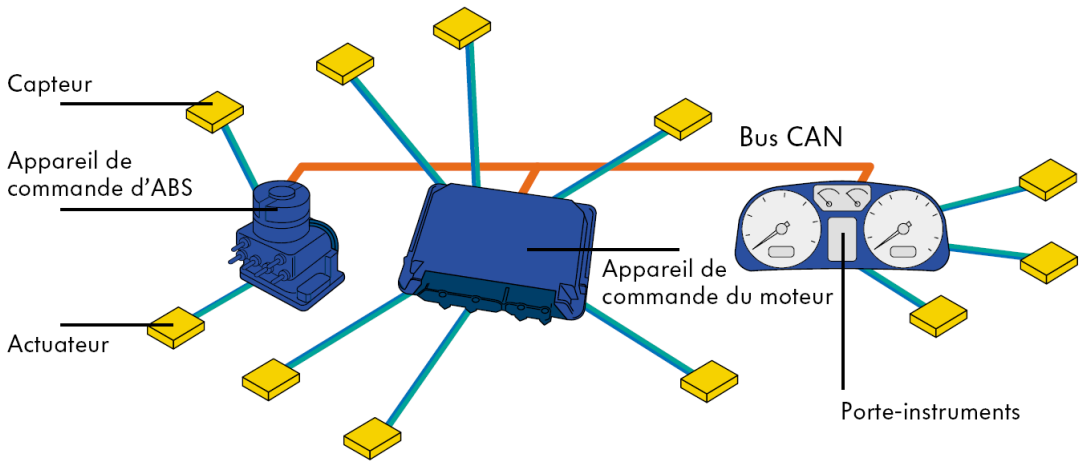
Différents types de réseaux multiplexés

Un grand nombre de bus existe sur le marché, parmi lesquels l'on peut citer :

- I²C (Inter Integrated Circuits de chez Philips)
- Ethernet
- **CAN**, VAN
- **LIN**
- FLEXRAY
- Etc..

Le CAN (Controller Area Network)

Le **CAN** a été adopté par l'automobile pour des raisons de ratio performance/coût particulièrement élevé.



LE CAN dans le MODELE OSI

Pour le protocole CAN, le document de référence développé par la société Bosch définie les deux premières couches (dites couches basses).

N° de la couche	Modèle ISO/OSI	Protocole CAN
7	Application	Spécifié par l'utilisateur (c'est son application)
6	Présentation	Vide
5	Session	Vide
4	Transport	Vide
3	Réseau	Vide
2	Liaison	Protocole CAN
1	Physique	Protocole CAN

LA COUCHE PHYSIQUE

La première couche du modèle a pour but de conduire les éléments binaires jusqu'à leur destination sur le support physique. Elle fournit les moyens matériels nécessaires à l'activation, au maintien et à la désactivation de ces connections physiques.
Cette couche gère la représentation du bit (codage, timing, synchronisation), et définit les niveaux électriques, optiques,... des signaux ainsi que le support de transmission.
Le protocole CAN ne décrit que la représentation détaillée du bit (Physical Signalling), mais pas le moyen de transport et les niveaux des signaux de telles sortes qu'ils puissent être optimisés selon l'application.

LA COUCHE LIAISON

Elle fournit les moyens fonctionnels nécessaires à l'établissement, au maintien et à la libération des connexions entres les entités du réseau. Cette couche (aussi appelée couche de communication de données, Data Link Layer) devra notamment corriger les erreurs qui ont pu se produire au premier niveau (même s'il est impossible de corriger toutes les erreurs).
Le protocole CAN décrit entièrement cette couche.
La couche liaison est subdivisée en deux sous-couches :

- La sous-couche LLC (Logical Link Control) effectue :
  - le filtrage des messages,
  - la notification des surcharges (overload),
  - la procédure de recouvrement des erreurs.
- La sous-couche MAC (Medium Access Control), qui est le coeur du protocole CAN, effectue :
  - la mise en trame du message,
  - l'arbitrage,
  - l'acquittement,
  - la détection des erreurs,
  - la signalisation des erreurs.

LA COUCHE APPLICATION

C'est la dernière couche du modèle OSI. Elle donne aux applications le moyen d'accéder aux couches inférieures.
Cette couche n'est bien sûr pas vide pour le protocole CAN, mais sa spécification est laissée à l'utilisateur.

Processus d'échange de données

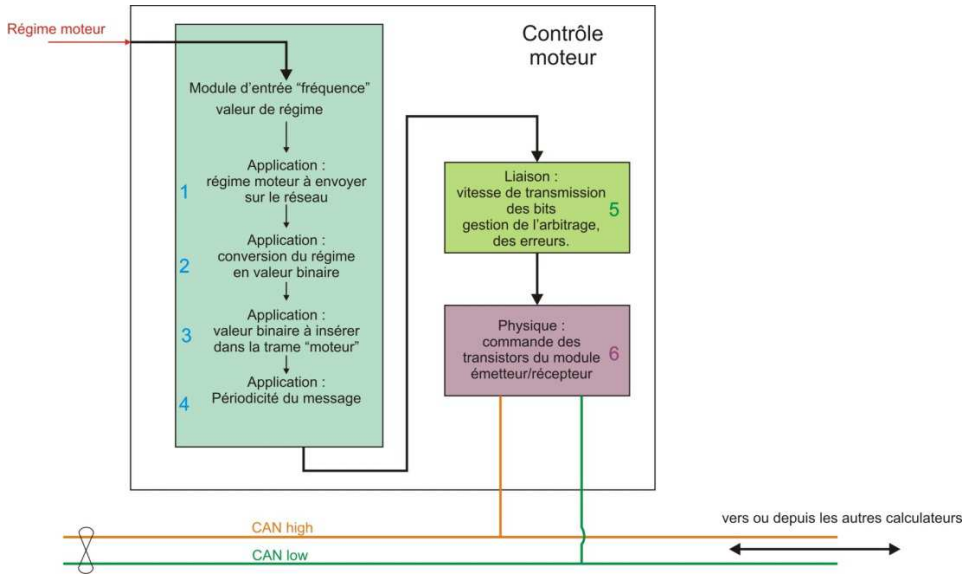
Le processus d'échange de données fait intervenir les différentes « couches » citées ci-dessus. Prenons l'exemple de la transmission du régime moteur :
Le calculateur de contrôle moteur acquiert la valeur du régime depuis le capteur de régime connecté à ces bornes.

1. Cette valeur peut être exprimée en tr/min, donc en valeur **décimale**.
2. Le régime moteur est ensuite converti en valeur **binaire** dans un format choisi par le concepteur du protocole utilisé.
3. Cette valeur binaire est positionnée dans un message défini par le concepteur du protocole qui peut contenir d'autres informations (température d'eau, position de la pédale d'accélérateur, …). On l'appelle la **trame** de données moteur 1 (par exemple ou trame dynamique moteur, ou trame EEC1-E, …).
4. Une **fréquence** de transmission de ce message est définie par le concepteur (dans ce cas la trame est **périodique** : envoyée régulièrement toutes les 10 ms).

5. Ce message est alors transféré au module de commande CAN (interne au calculateur). Ce module de commande gère les **arbitrages**, les défauts de transmission, la déconnexion du bus.

6. Le module de commande transfère alors le message (toujours binaire) au module émetteur/récepteur qui le transforme en **tension** électrique sur les fils **CAN high et CAN low**.

NB : les points **1 à 4** sont définis dans le cadre de la couche **application**.
Le point **5** est défini par le contrôleur CAN et est intégré dans la puce. Il est défini par la couche **liaison**.
Le point **6** est également intégré dans une puce et est défini en partie par la couche **physique**.



Etude de la couche physique du CAN

Un bus de communication a donc souvent pour support physique deux fils torsadés. Ces fils sont nommés **CAN high** et **CAN low**.

Nous avons donc affaire à une transmission **série asynchrone** (il n'y a pas d'horloge commune aux différents calculateurs) par **paires différentielles (**il y a deux fils dont on soustrait les tensions pour obtenir la valeur du bit transmis).

Pour diminuer le risque de défaut de transmission, des dispositions ont été prises au niveau la couche physique :

- la transmission est réalisée sur **deux fils** (nommés CAN high et CAN low) ;

- la **somme des tensions** est toujours **constante** quel que soit le niveau logique (le bit) transmis.

Il existe deux types de protocole CAN :

- CAN Low Speed : pour des vitesses de 40kbits/s à 125kbits/s
- CAN High Speed : pour des vitesses allant jusqu'à 1Mbits/s

NB : c'est le type de transceiver utilisé qui va différencier le type de protocole CAN low speed ou high speed (détail plus loin).

Tous les calculateurs présents sur un réseau sont reliés par les modules émetteur/récepteur intégrés dans les calculateurs. Chaque calculateur connecté est appelé un **nœud**.

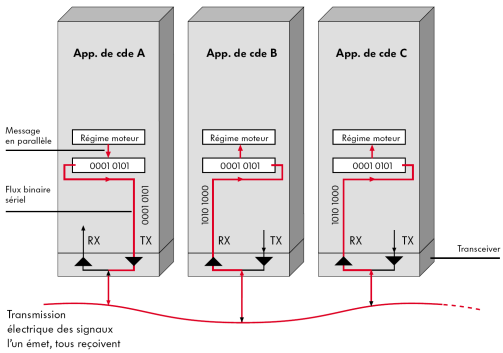
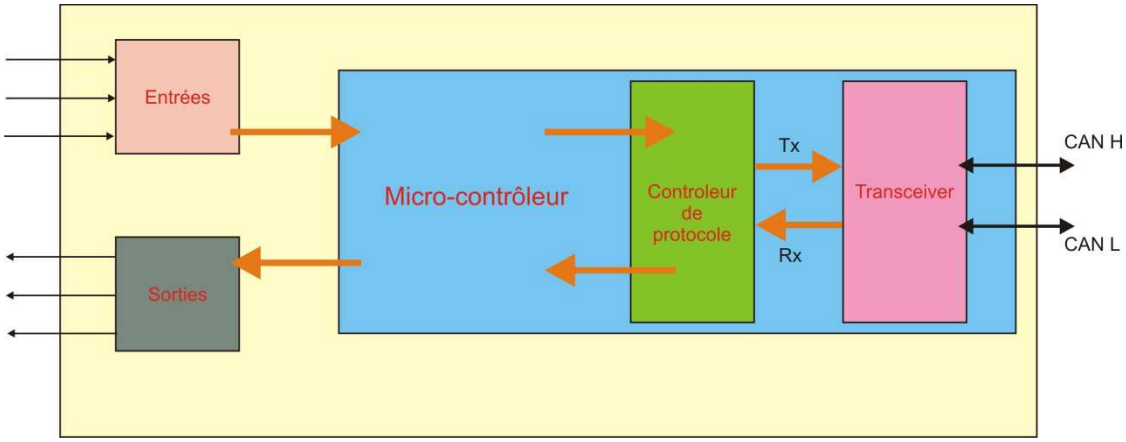


Schéma de principe d'un module émetteur/récepteur :

(Le module émetteur/récepteur est appelé *transceiver* en anglais).

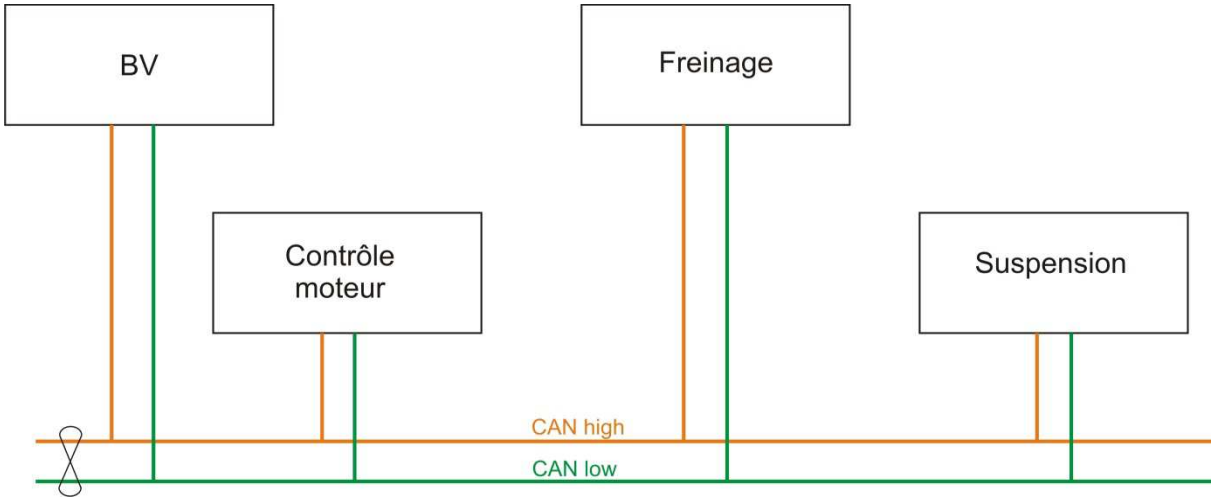

Entrées
Micro-contrôleur
Controleur
de
protocole
Tx
Rx
Transceiver
CAN H
CAN L
Sorties

Topologie du réseau :

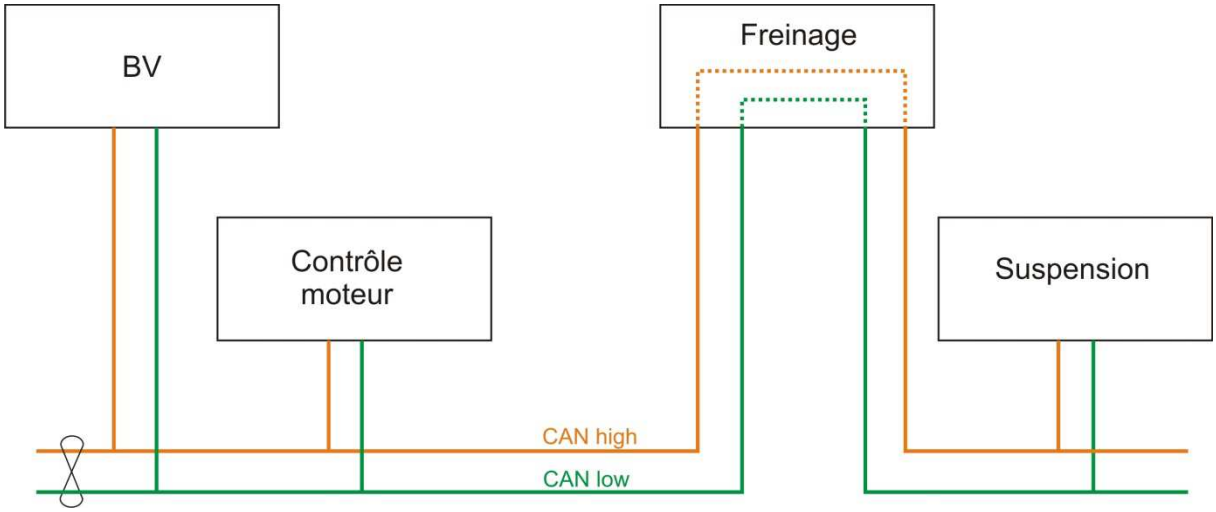
Les différents **nœuds** sont donc en liaison par les fils selon un câblage que l'on nomme **topologie** du réseau.

Exemple de réseau :

Réseau sous forme de bus

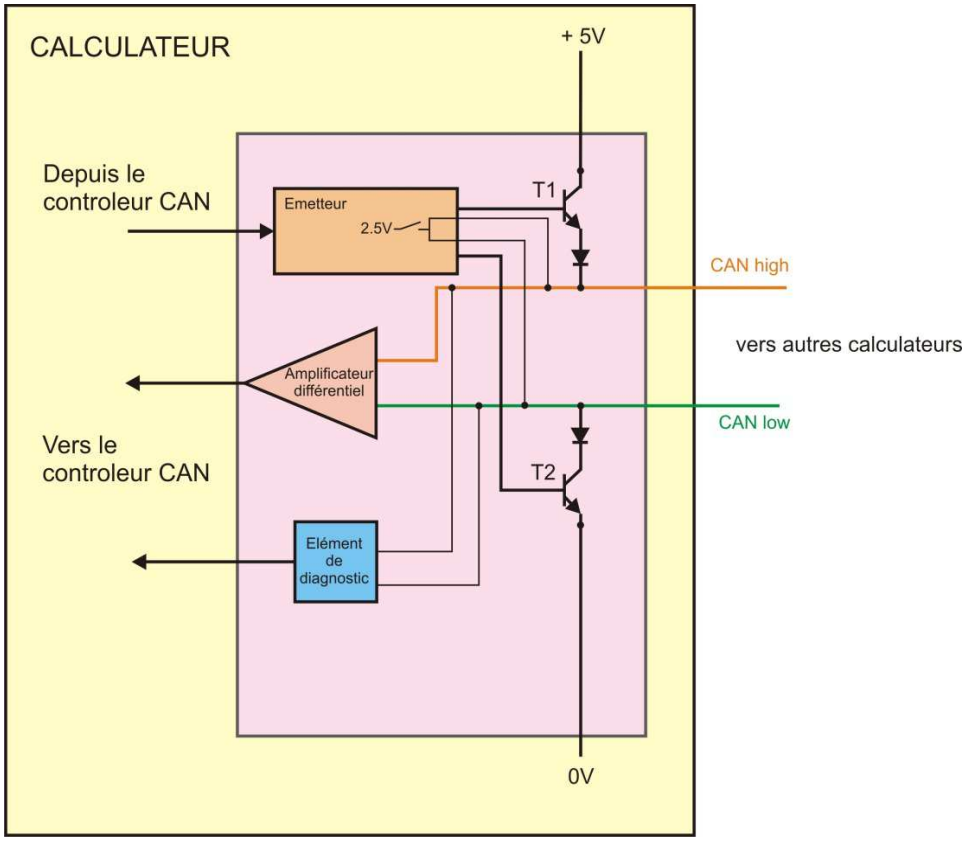


Réseau avec « passerelle » dans un calculateur (*Daisy Chain*) permettant le lien entre deux réseaux CAN.



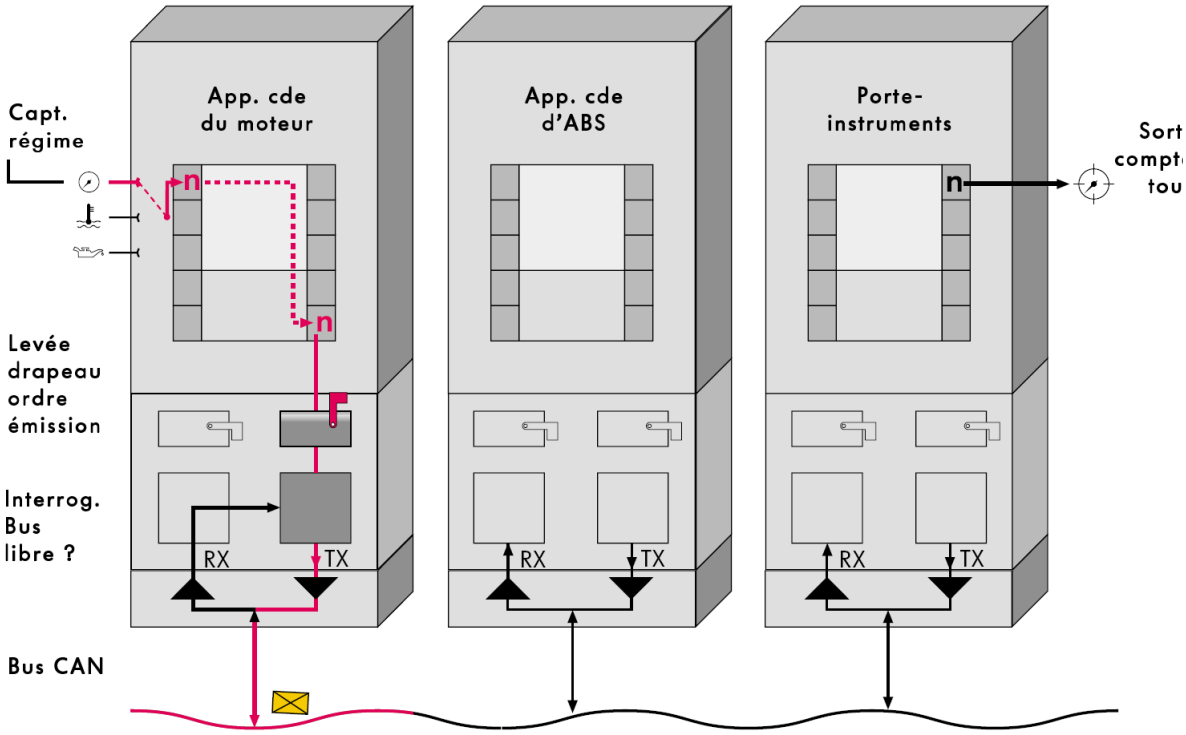
Principe de fonctionnement du transceiver pour un réseau CAN low speed :

Le *transceiver* possède un élément de détection des anomalies des tensions sur le bus. Cette configuration permettant de diagnostiquer les défauts et ainsi de continuer la transmission (ou la réception).
De par la présence de ces éléments intégrés de diagnostic, ce réseau tolère certains défauts, d'où le nom de **CAN FAULT TOLERANT** attribué au CAN low speed.

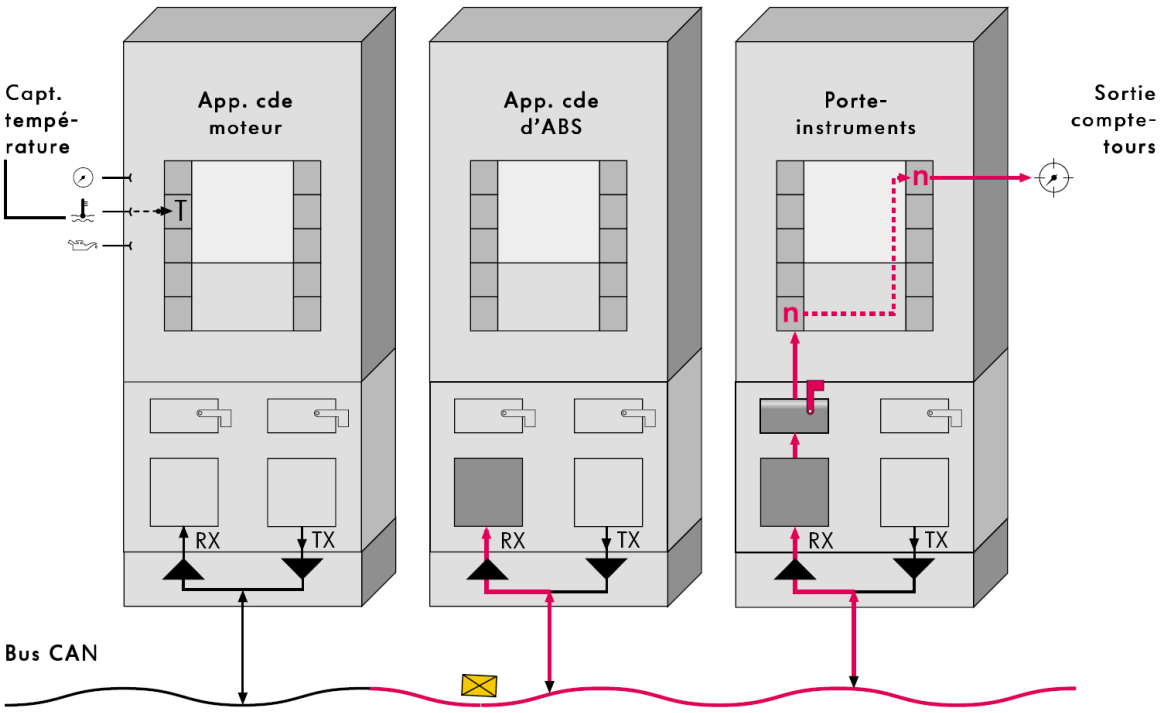


Processus de transmission des données

Emission :

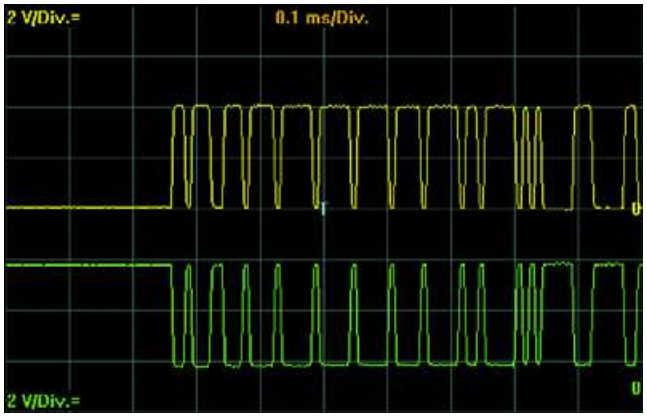


Réception :



Niveaux de tensions pour le CAN low speed :

Oscillogramme :

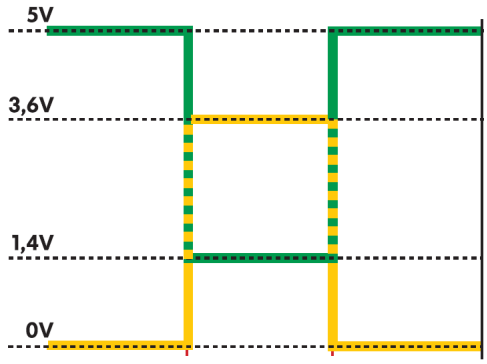


La voie 1 de l'oscilloscope entre CAN H et la masse.

La voie 2 de l'oscilloscope entre CAN L et la masse.

Sur cet oscillogramme les deux voies sont décalées verticalement.

Sur le fil **CAN H** : la tension évolue de **0V à 3.6V.**
Sur le fil **CAN L** : la tension évolue de **1.4V à 5V**.

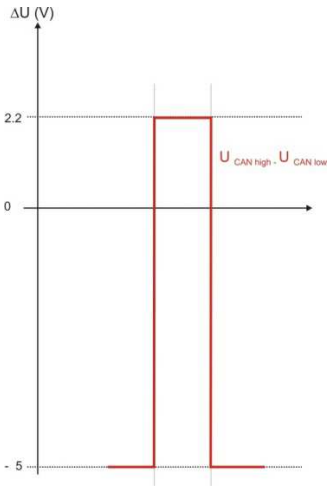


Ci-dessus : 1-0-1

A un instant :
$U_{CAN\ H}$ - $U_{CAN\ L}$ = 0 - 5 = **- 5V**
correspond **à 1 logique**

A un autre instant :
$U_{CAN\ H}$ - $U_{CAN\ L}$ = 3.6 - 1.4 = **+ 2.2V**
correspond **à 0 logique**

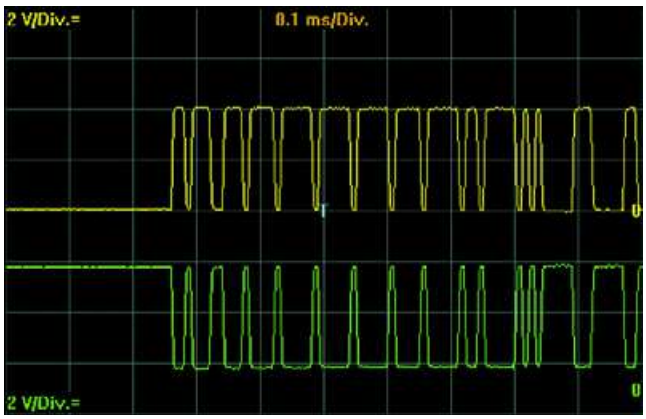
La différence des tensions est bien marquée pour distinguer les niveaux logiques : -7.2V = (-5 - (+2.2))



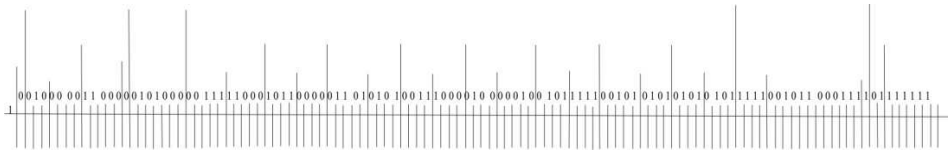
Etude de la couche liaison du CAN

A partir des tensions obtenues après le *transceiver*, le contrôleur de protocole CAN est donc en mesure de lire des 0 et 1 logiques.
Ainsi un ensemble des niveaux de tensions va constituer une **trame** et correspondre à une série de 0 et de 1 logiques.

Oscillogramme d'une trame :



Trame convertie en 0 et 1 par le controleur CAN :



En première approche, il n'est pas évident d'interpréter cette suite de 0 et de 1 lue (ou envoyée) par le contrôleur CAN.

Notions de protocole CAN

Les **trames** circulent sur le BUS, à l'intérieur nous retrouvons les données mais aussi des éléments qui servent à l'orientation des données.
En effet les données sont envoyées à tous les calculateurs présents (connectés) sur le réseau et tous les calculateurs vont les lire.
En fonction de leur application respective des données seront utiles pour certains calculateurs et seront ignorées pour d'autres.

Ainsi la trame va comporter des **champs** permettant son décodage par les calculateurs.

Composition d'une trame CAN :

Chaque champ possède une longueur (nombre de bit) et un rôle bien précis :

SOF (Start Of Frame) :
- 1 bit.
- Champ de départ de la trame toujours égal à 0.

Identificateur :
- 11 bits.
- Identifie l'émetteur du message.

RTR (Remote Transmission Request) :
- 1 bit.
- Généralement à 0 sauf dans le cas d'une trame de requête.

Commande :
- 6 bits.
- Contient le DLC (Data Length Code), la longueur des données transmises en nb d'octets. Par exemple si 4 octets de donnée : DLC = 001000.

Données :
- De 0 à 8 octets.

CRC (Contrôle de Redondance Cyclique) :
- 16 bits.
- Permet grâce à un algorithme de calcul de vérifier s'il a eu une erreur de transmission.

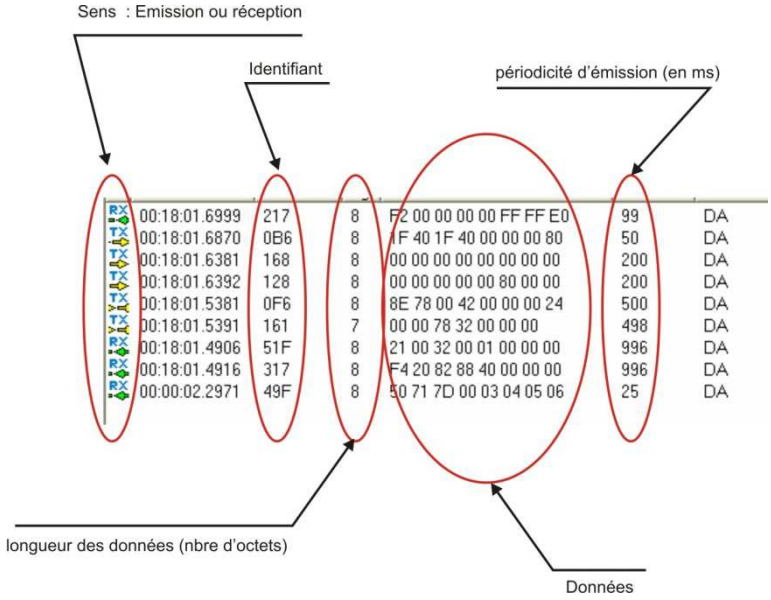
ACK (ACKnowledge) :
- 2 bits.
- Acquittement permettant de savoir si la trame a été lue par un nœud.

EOF (End Of Frame) :
- 7 bits.
- Indique la fin de la transmission du message, 7 bits à 1 : 1111111.

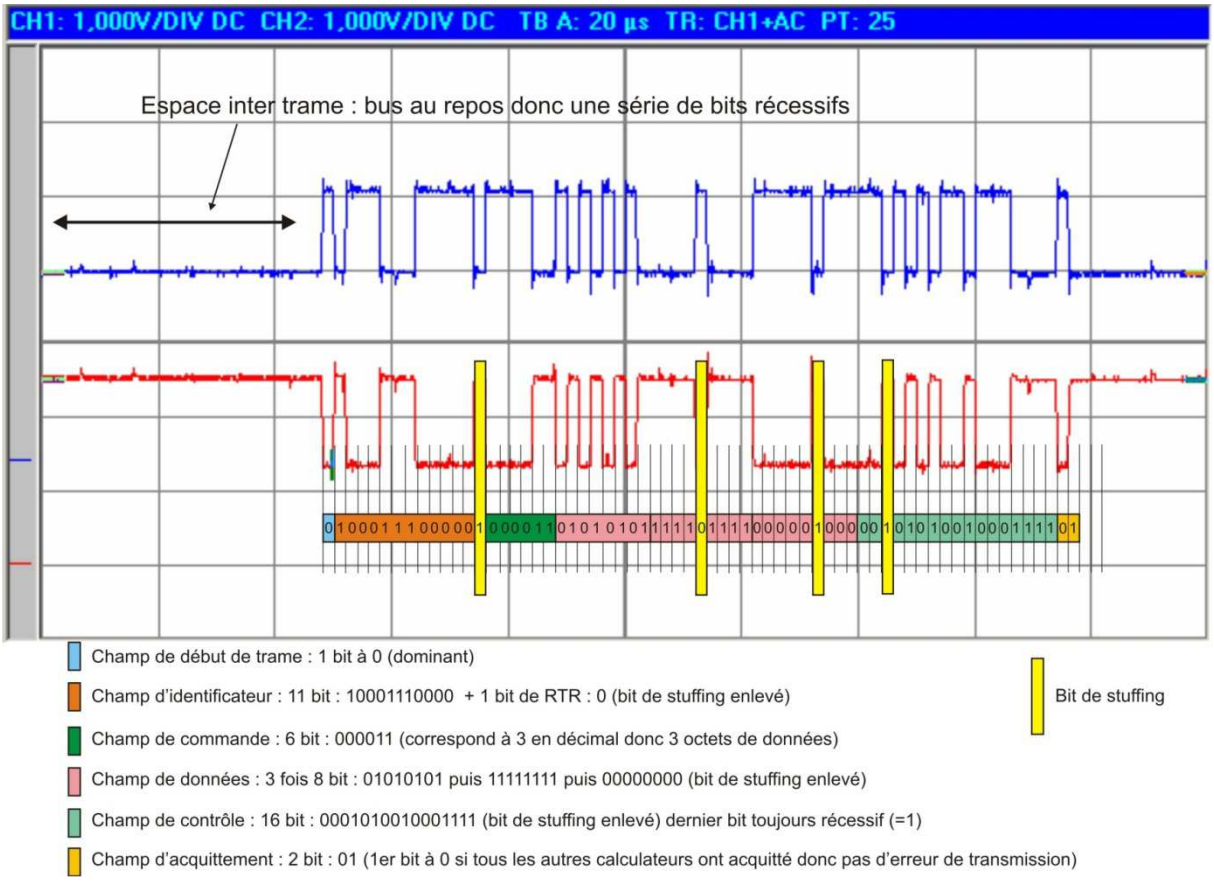
Décodage d'une trame CAN :

A l'aide d'outil d'analyse des trames CAN
Ces outils permettent de lire les trames présentes sur un réseau multiplexé.
Ils sont en liaison avec un PC pour enregistrement.
Ils permettent également de participer aux échanges en envoyant des trames spécifiées par l'utilisateur.

Une lecture de trames se présente en général sous cette forme :



Avec un oscillogramme :
Il s'agit de repérer chaque champ en définissant la longueur du bit (appelé le Time Slot) et en se référant à la définition du protocole concernant le nombre de bit affecté à chaque champ. Et ne pas oublier d'enlever les bits de *stuffing* éventuels pour avoir le message original.



- Champ de début de trame : 1 bit à 0 (dominant)
- Champ d'identificateur : 11 bit : 10001110000 + 1 bit de RTR : 0 (bit de stuffing enlevé)
- Champ de commande : 6 bit : 000011 (correspond à 3 en décimal donc 3 octets de données)
- Champ de données : 3 fois 8 bit : 01010101 puis 11111111 puis 00000000 (bit de stuffing enlevé)
- Champ de contrôle : 16 bit : 0001010010001111 (bit de stuffing enlevé) dernier bit toujours récessif (=1)
- Champ d'acquittement : 2 bit : 01 (1er bit à 0 si tous les autres calculateurs ont acquitté donc pas d'erreur de transmission)

Bit de stuffing

Notion de priorité

Que se passe-t-il si plusieurs nœuds tentent de transmettre simultanément?

Il existe une procédure d'accès au bus à laquelle chaque nœud doit se soumettre : lorsqu'il transmet un bit, le nœud doit rester à l'écoute du bus. Dit autrement, après avoir envoyé un bit, il lit le bus et vérifie que le bit lu correspond à celui transmis, si il y une différence (forcément sur un bit récessif), c'est qu'un autre nœud l'a écrasé par un bit dominant, le nœud doit alors interrompre sa transmission, monitorer le bus pour attendre la fin de la transmission, puis tenter à nouveau d'envoyer son message.
Ainsi une priorité est réalisée grâce au champ d'arbitrage.

C'est pendant la transmission des bits du champ d'identification (que l'on pourrait appeler arbitrage également) que le message de plus haute priorité (donc celui qui contient le plus de 0) gagne le bus.

C'est donc l'identifiant le plus petit qui est prioritaire.

Dans l'exemple ci-dessous, 3 calculateurs A, B et C émettent en même temps.
Identifiant de A : 0001 0001 1111 $((8F)_{16}$ $(143)_{10})$.
Identifiant de B : 0001 1011 1111 $((1BF)_{16}$ $(447)_{10})$.
Identifiant de C : 0001 0101 1111 $((15F)_{16}$ $(351)_{10})$.

Jusqu'au 4ème bit : il y a égalité des identifiants.
Au 5ème bit : le calculateur B émet un bit identifiant à 1 (récessif), le calculateur B perd la priorité et arrête d'émettre.
Au 6ème bit : le calculateur C émet un bit identifiant à 1 (récessif), le calculateur C perd la priorité et arrête d'émettre.
Ainsi le calculateur A peut continuer d'émettre sur le bus.

Ainsi le constructeur devra choisir judicieusement les identifiants affectés aux messages qui doivent être prioritaire notamment pour des raisons de sécurité.
Exemple : un message demandant la coupure de l'injection pendant une phase de régulation de freinage (ABS ou ESP) prioritaire par rapport à une demande d'accélération !!!

Bit de stuffing

Afin de sécuriser la transmission des messages, la méthode du « bit-stuffing » est utilisée.

Elle consiste, dans le cas où l'on a émis 5 bits de même polarité d'affilée, d'ajouter à la suite un bit de polarité contraire, pour casser des chaînes trop importantes de bits identiques.

Cette méthode n'est appliquée que sur les champs SOF, d'arbitrage, de commande et de CRC (délimiteur exclu).

Par exemple, « 1111 1110 » deviendra « 1111 1011 0 ».

Etude de la couche application du CAN

C'est le constructeur qui a décidé des identifiants (priorité) et des données à transmettre. Ces éléments sont regroupés sous le terme de **messagerie**.
Ainsi chaque constructeur possède une messagerie qui lui est propre et qui est souvent confidentielle.

Parmi les éléments entrant dans l'organisation de cette messagerie, il faut définir :

- Si les messages doivent être **périodiques** (régulièrement transmis) ou bien **événementiel** (arrivée d'une situation, idem interruption en micro contrôleur).
- La **priorité** (le message le plus important aura un identifiant le plus petit, commençant par des 0, des bits dominants donc).
- Le **contenu** de chaque message : les données à transmettre, leur **longueur** (nombre de bit) et leur **position** dans les données.

Le constructeur, pour chaque **grandeur physique** ou information à transmettre va définir le **nombre de bit ou d'octet** nécessaires :

- Pour transmettre l'état d'un contacteur de pédale de frein : il suffit d'un seul bit : si la pédale n'est actionnée le bit est à 1 et si la pédale est actionné le bit est à 0 (par exemple).
- Pour transmettre une valeur de température d'eau : 1 octet est nécessaire donc 8 bit (donc 256 valeurs de température possibles pour une précision de 1°C).
- Pour transmettre une valeur de régime moteur, il faut 2 octets : 65536 valeurs possibles.
- Etc.

Ensuite il faut définir un mode de conversion entre d'une part la grandeur physique et d'autre la valeur binaire (ou hexadécimale).
Pour cela les constructeurs utilisent deux valeurs supplémentaires : le **gain** et **l'offset**.

Exemple de trame:

Message	DLC	Data	Cycle Time
38D	4	3F E7 13 88	100

Identifiant : *38D*
Longueur des données : *4 octets*
Période : 100 ms.
Données :

3F								E7								13								88							
0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0
1.7	1.6	1.5	1.4	1.3	1.2	1.1	1.0	2.7	2.6	2.5	2.4	2.3	2.2	2.1	2.0	3.7	3.6	3.5	3.4	3.3	3.2	3.1	3.0	4.7	4.6	4.5	4.4	4.3	4.2	4.1	4.0
Octet 1								Octet 2								Octet 3								Octet 4							

Messagerie :

Octet1					
Bit	Paramètre	Description	Codage	Type	Valeur Defaut
1.7 1.0	T_EAU	Température d'eau moteur	N * 1.0 - 40 Mini : -40 Maxi : 214°C Invalide : 0xFF	ENT NS	

Octet2					
Bit	Paramètre	Description	Codage	Type	Valeur Defaut
2.7	ABPI	Signalisation airbag passager inhibé	0 : Normal 1 : Inhibé		
2.6	OUCC	Signalisation d'oubli de ceinture conducteur	0 : Normal 1 : Oubli		
2.5	FRPK	Signalisation de frein parking	0 : Desserré 1 : Serré		
2.4	MINC	Signalisation de mini carburant	0 : Eteint 1 : Allumé		
2.3	CCARB_NEUT	Signalisation circuit de carburant neutralisé	0 : Normal 1 : Déclenché		
2.2	PRE_CHAUFF	Information préchauffage Diesel	0 : Préchauffage non actif 1 : Préchauffage actif		
2.1	OUCP	Signalisation d'oubli de ceinture passager	0 : Normal 1 : Oubli		
2.0	ESPI	ESP inhibé	0 : Fonctionne 1 : Inhibé		

Octet3, Octet4					
Bit	Paramètre	Description	Codage	Type	Valeur Defaut
3.7 4.0	VIT	Vitesse	N * 0,02 Invalide : 0xFF	Flottant	

Principe de conversion des données :

Message	DLC	Data	Cycle Time
38D	4	3F E7 13 88	100

Exemple octet1 la température :

Octet1					
Bit	Paramètre	Description	Codage	Type	Valeur Defaut
1.7 1.0	T_EAU	Température d'eau moteur	N * 1 - 40 Invalide : Mini : -40 Maxi : 214°C 0xFF	ENT NS	

- L'octet 1 contient la température codée : 3F en hexadécimal.
- La formule de codage est N*1-40.

Ainsi si l'on veut traduire une valeur lue sur le bus en température en °C, la conversion est la suivante :

- On convertit en décimal : $(3F)_{16} = (63)_{10}$
- On applique la formule : **Température = 63 * 1 - 40 = 23°C**

Exemple octet2 l'état de l'airbag passager :

Octet2					
Bit	Paramètre	Description	Codage	Type	Valeur Defaut
2.7	ABPI	Signalisation airbag passager inhibé	0 : Normal 1 : Ihnibé		

- L'octet 2 contient 8 bits codés en hexadécimal :
- Le 8ème bit (bit 2.7) en partant de la droite contient l'état de l'airbag passager.

Ainsi si l'on veut connaître l'état de l'airbag passager, la conversion est la suivante :

- On convertit en binaire : $(E7)_{16} = (11100111)_2$
- Le 8ème bit (bit 2.7) en partant de la droite vaut 1 : **L'airbag est inhibé.**

Si l'airbag était normal : le 8ème bit (bit 2.7) vaudrait 0.
L'octet 2 deviendrait en binaire : $(01100111)_2$
Ainsi l'octet 2 en hexadécimal : $(67)_{16}$

Exemple octet3 et octet4 la vitesse du véhicule :

Octet3, Octet4					
Bit	Paramètre	Description	Codage	Type	Valeur Defaut
3.7 4.0	VIT	Vitesse	N * 0,02 Invalide : 0xFF	Flottant	

- L'octet 3 vaut : 13 en hexadécimal
- L'octet 4 vaut : 88 en hexadécimal

La conversion est la suivante :

- On convertit en binaire l'octet 3 : $(13)_{16} = (00010011)_2$
- On convertit en binaire l'octet 4 : $(88)_{16} = (10001000)_2$
- On met bout à bout les deux octets : 0001001110001000
- On convertit le mot binaire ainsi obtenu en décimal : 5000
- On applique la formule de conversion : **Vitesse = 5000 * 0.02 = 100 km/h**