



UNIVERSITÉ BRETAGNE SUD

RAPPORT DE TRAVAUX PRATIQUES

## Réseaux Industriels

*Élèves :*

Alexandre PEREZ  
Benjamin MAHOUDEAU  
Leonardo MONTOYA OBESO

*Encadrants :*  
Julien DELORME

7 février 2024

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>TP1 : Prise en main de la maquette</b>	<b>2</b>
2.1	Bus CAN de la maquette . . . . .	2
2.2	Port OBDII . . . . .	3
<b>3</b>	<b>TP1 : Utilisation du logiciel MuxTrace</b>	<b>4</b>
3.1	Etude pratique : régime moteur . . . . .	6
3.2	Etude pratique : vitesse véhicule . . . . .	6
3.3	Etude pratique : Comodo de phares . . . . .	7
3.4	Etude pratique : Rétroviseurs . . . . .	8
3.5	Etude pratique : Faites votre choix . . . . .	11
<b>4</b>	<b>TP2 : Utilisation du logiciel MuxTrace et DBEdit</b>	<b>11</b>
<b>5</b>	<b>TP3 : Utilisation du mode programmation sous MuxTrace</b>	<b>13</b>
<b>6</b>	<b>Conclusion</b>	<b>16</b>
<b>A</b>	<b>Annexe A : Illustration complémentaire</b>	<b>17</b>
A.1	TP1 . . . . .	17
A.2	TP3 . . . . .	20
<b>B</b>	<b>Annexe B : Code Snippets</b>	<b>22</b>

# 1 Introduction

Ce document résulte des travaux de groupe menés durant les TP du cours de Réseaux Industriels du programme de Master SESI pour l'année académique 2023-2024 à l'Université Bretagne Sud.

Le but de ce TP est de mettre en pratique nos connaissances théoriques du bus CAN par l'utilisation d'une maquette automobile pédagogique EXXOTEST (voir annexe 11). Cette maquette est composée d'équipements réels tels que des feux xénon, des clignotants, des essuie-glaces, des rétroviseurs et des vitres électriques. De plus, la maquette dispose d'une partie simulée pour le moteur avec des potentiomètres (voir annexe 12) permettant de changer les rapports, d'augmenter le régime moteur, de définir le niveau d'essence, entre autres.

## 2 TP1 : Prise en main de la maquette

### 2.1 Bus CAN de la maquette

**Identifier le nombre de bus CAN présents sur la laquette ainsi que leur vitesse.**

Comme indiqué en haut à gauche du schéma de câblage de la maquette (voir annexe 13), il y a en tout 4 bus CAN : le bus CAN DIAG, le bus CAN I/S, le bus CAN CONF, ainsi que le bus CAN CAR. Les CAN CONF et CAR sont tous deux des CAN LS (low speed) de 125 kbit/s, alors que le CAN I/S est un CAN HS (high speed) avec une vitesse de 500 kbit/s.

**Quelles fonctions remplissent chacun de ces bus sur la maquette.**

Les fonctions remplies par chacun des bus CAN de la maquette sont les suivantes :

- Le bus CAN DIAG, pour diagnostic, permet de récupérer des informations importantes sur les équipements afin de les diagnostiquer.
- Le bus CAN I/S, pour inter-système, s'occupe des liaisons importantes de commandes et de sécurité.
- Le bus CAN CONF, pour confort, s'occupe de la liaison des équipements d'indication et d'aide à la conduite.
- Le bus CAN CAR, pour carrosserie, s'occupe de la liaison des informations moteurs et des équipements extérieurs.

**Décrire la structure d'un paquet CAN.**

La Figure 1 montre que chaque trame est séparée par 3 bits au minimum. Une trame CAN est découpée en 7 morceaux, ayant chacun une fonction différente :

- Un champ de Début de 1 bit marque le début de la trame.
- Un champ ID de 11 bits sert à identifier l'émetteur de l'information. Avec 11 bits, il est possible de coder 2047 IDs différents, c'est donc le maximum d'équipements qui peuvent être connectés à un bus CAN.

- Un champ Commande contient le DLC (data length code) de 4 bits qui spécifie la taille du champ d'information.
- Le champ information contient les données de la trame et peut avoir une taille entre 0 et 8 octets.
- Un champ CRC de 15 bits est calculé à partir de l'ensemble des champs transmis jusqu'alors et permet la détection d'erreur.
- Un champ ACK de 2 bits permet l'acquittement d'un message.
- Un champ EOF de 7 bits indique la fin de la trame.

Comme on peut le voir, le champ d'information d'une trame CAN a une longueur maximale de 64 bits, longueur courte pour une trame de 107 bits au total. Cependant, le principal avantage de l'utilisation du protocole CAN réside dans sa robustesse et sa résilience aux pannes et au bruit, outre un câblage réduit. C'est pourquoi le bus CAN est largement utilisé dans le domaine automobile

Structure des trames CAN Standard sur le bus (automobile) :

IFS	Début	Identificateur	Com.	Informations	CRC	ACK	EOF
IFS Inter trame				trame libre 3 bits mini			
Début ou SOF				Début de trame 1 bit			
Identificateur				Champ d'identification de la trame 11 bits			
Com.				DLC 4 bits et champ de commande 3 bits			
Informations				données transmises par un équipement ou lues dans un équipement jusqu'à 8 octets (8 x 8 bits).			
CRC Contrôle				champ de contrôle 15 bits			
ACK				champ accusé de réception 2 bits.			
Fin ou EOF				symbole indiquant la fin de la trame 7 bits			

FIGURE 1 – Structure des trames CAN, source : [1]

**Donner le nombre d'ID différents présents sur chaque bus identifiés (exports de données depuis le logiciel MuxTrace).**

Pour les 3 bus étudiés (I/S, CONF, et CAR), le nombre d'ID différents présents sur chaque bus est respectivement de 22, 70 et 23.

## 2.2 Port OBDII

### A quoi sert le port OBDII ?

Comme décrit sur le site [2], la norme OBDII a été instaurée à l'origine par la CARB, “Californian Air Resources Board” pour son acronyme en anglais, afin de surveiller les émissions polluantes des véhicules. La norme requiert que le véhicule surveille de manière continue le bon fonctionnement du moteur tout au long de sa durée de vie.

Le protocole OBD permet le diagnostic embarqué d'un véhicule. C'est un système électronique responsable de l'auto-diagnostic et de la production de rapports destinés aux professionnels de la réparation. Grâce à ce système, les techniciens ont accès à des informations sur les sous-systèmes du véhicule, permettant ainsi le suivi de sa performance et l'analyse des besoins en réparations.

Certaines versions de la norme OBDII sont énumérées ci-dessous.

- L'OBD ou OBDI standardise le connecteur pour assurer son uniformité sur tous les véhicules. En revanche, le protocole de communication demeure plus ou moins spécifique en fonction des marques.
- L'OBDII est arrivé aux États-Unis en 1996 pour spécifier des protocoles communs. En plus de cela, contrairement à l'OBDI qui est connecté à l'extérieur de la console d'une voiture, l'OBDII est intégré au véhicule.
- L'EOBD, pour European OBD, reprenant l'OBDII, est spécifique pour les véhicules européens.

#### **Recherche bibliographique sur le web : câblage de connecteur, mode d'échange entre les outils et le véhicule.**

Comme on peut le voir à l'annexe 14, le port OBDII est composé de 16 broches, chacune ayant un rôle spécifique. Parmi elles, il y a :

- 2 broches pour le positif et le négatif du J1850 (SAE) utilisant le protocole de modulation de durée d'impulsion (VPW).
- 2 broches pour récupérer la masse du châssis et la masse du signal.
- 2 broches pour la ligne K et la ligne L utilisant un protocole de communication série asynchrone.
- 2 broches pour le LOW et le HIGH du protocole CAN.
- 7 broches non standard réservées au constructeur.
- 1 broche d'alimentation connectée à la batterie du véhicule pour alimenter les outils de scan. Elle est soit de type A avec une tension de 12V pour les voitures, ou de type B avec une tension de 24V pour les poids lourds.

Les informations sont générées par les unités de contrôle du moteur puis sont analysées par les outils connectés au port OBDII présent dans le véhicule.

## **3 TP1 : Utilisation du logiciel MuxTrace**

Dans cette partie, nous allons utiliser le logiciel MuxTrace afin d'analyser les trames circulant dans les bus CAN. Pour cela, nous allons nous interfacer avec les bus I/S, CONF, et CAR grâce à la plaque de branchement de la maquette (voir illustration du branchement, annexe 15). Les signaux des bus CAN sont ensuite envoyés sur un boîtier USB-MUX-6C6L, avec le CAN CONF sur le port 1, le CAN I/S sur le port 2, et le CAN CAR sur le port 3 (voir illustration annexe 16). Cela permet, au final, de récupérer les informations des trames sur le logiciel MuxTrace.

Sur le logiciel MuxTrace, nous configurons chaque bus avec le bon débit en utilisant la détection automatique, comme illustré dans la figure 2.

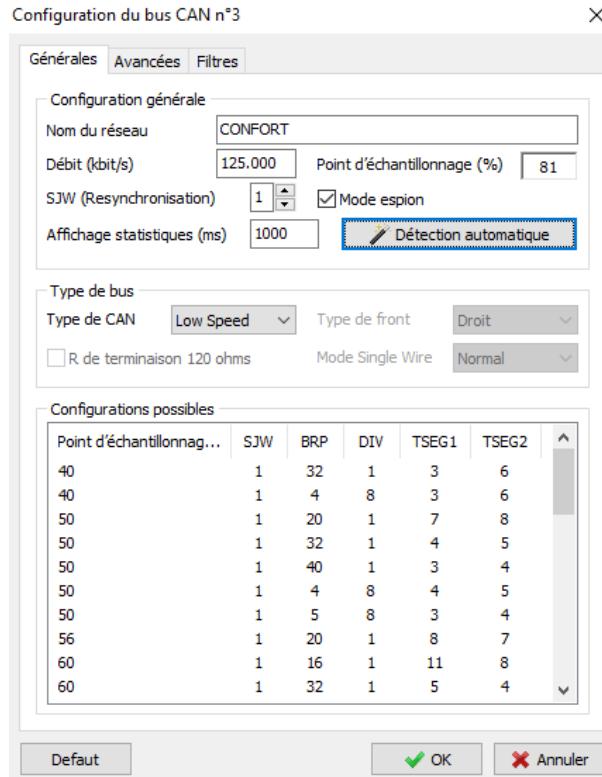


FIGURE 2 – Outil de détection automatique du débit sur le logiciel MuxTrace.

Nous pouvons ensuite observer les trames pour les 3 bus CAN sur le logiciel MuxTrace, comme on peut le voir dans la figure 3.

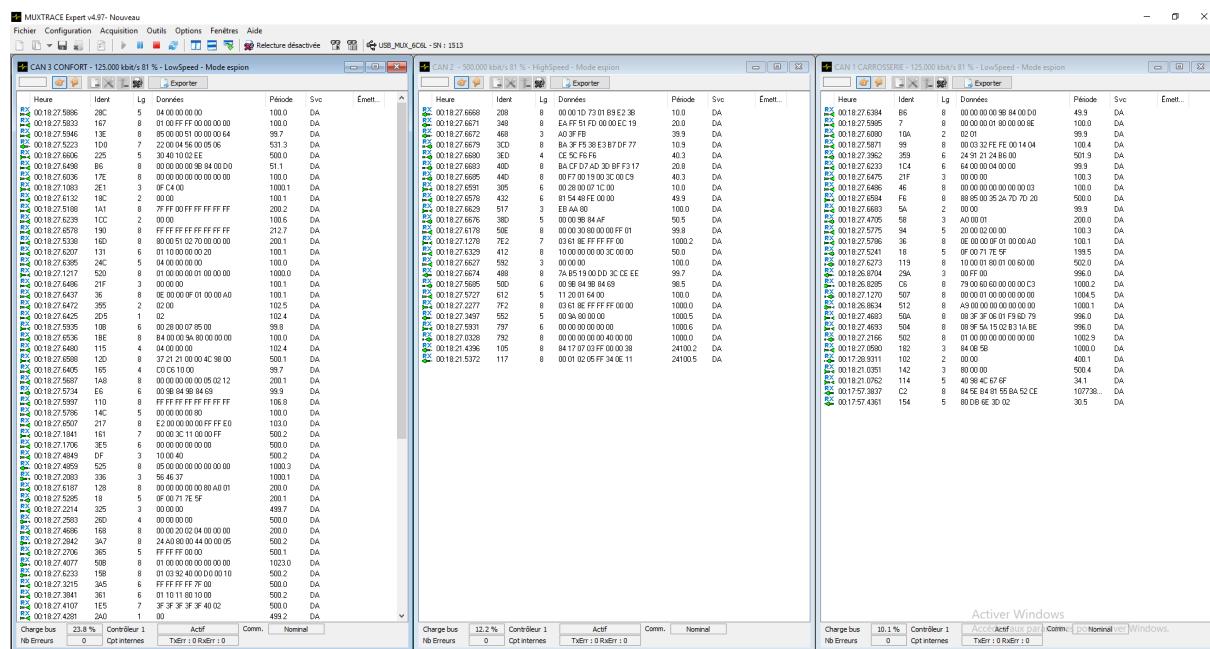


FIGURE 3 – Observation des trames sur le logiciel MuxTrace.

### 3.1 Etude pratique : régime moteur

**Observer sur le bus IS la trame 208.**

Sur MuxTrace nous pouvons voir que la trame d'ID 208 du bus IS contient des données de 8 octets et émet avec une période de 10ms.

**Faites varier le régime moteur et donnez vos observations sur le champs de données. Par exemple, vous pouvez faire varier la vitesse par pas de  $1000\text{tr}.\text{min}^{-1}$ .**

Lorsque nous faisons varier le régime moteur, nous constatons que le premier octet de la trame 208 varie. Nous avons reporté dans le tableau 1 la valeur du premier octet obtenue en faisant varier la vitesse par pas de  $1000\text{tr}.\text{min}^{-1}$  de  $0\text{tr}.\text{min}^{-1}$  à  $5000\text{tr}.\text{min}^{-1}$ .

Régime moteur ( $\text{tr}.\text{min}^{-1}$ )	Premier octet (hexa)	Premier octet (base 10)
0	00	00
1000	21	33
2000	3F	63
3000	5E	94
4000	7D	125
5000	9B	155

TABLE 1 – Premier octet de la trame 208 en fonction du régime moteur.

**Pouvez-vous en déduire une suite logique sur la variation du régime moteur ? Dans tous les cas, donnez vos arguments.**

En observant les valeurs en base 10, on constate la suite suivante :

- de 0 à 1000 : +33
- de 1000 à 2000 : +30
- de 2000 à 3000 : +31
- de 3000 à 4000 : +31
- de 4000 à 5000 : +30

On a donc une suite logique d'environ 31 par pas de  $1000\text{tr}.\text{min}^{-1}$ .

### 3.2 Etude pratique : vitesse véhicule

**Sur le bus IS, identifiez la ou les trames qui évoluent en fonction de la vitesse du véhicule.**

Lorsque l'on modifie la vitesse, nous constatons un changement sur le premier, troisième et cinquième octet de la trame 44D ainsi que le premier octet de la trame 38D.

**Observez la trame 0x44D. Quelle est sa taille et sa fréquence ?**

La trame 0x44D a une taille de 8 octets et une fréquence de 40ms

**Faites varier la vitesse du véhicule et donnez vos observations sur les champs de données. Par exemple, vous pouvez faire varier la vitesse par pas de 10 de 0 à 100Km/h.**

Lorsque nous faisons varier la vitesse du véhicule, nous constatons que le premier, troisième et cinquième octet de la trame 44D varient avec la même valeur. Nous avons reporté dans le tableau 2 la valeur obtenue en faisant varier la vitesse par pas de 10km/h de 0km/h à 100km/h.

Vitesse véhicule (km/h)	Octets 1,3 et 5 (hexa)	Octets 1,3 et 5 (base 10)
0	00	00
10	03	03
20	08	08
30	0B	11
40	0F	15
50	13	19
60	17	23
70	1B	27
80	1E	30
90	22	34
100	26	38

TABLE 2 – Octets 1, 3 et 5 de la trame 44D en fonction de la vitesse du véhicule.

**Pouvez-vous en déduire une suite logique sur la variation de la vitesse du véhicule ? Dans tous les cas, donnez vos arguments.**

En observant les valeurs en base 10, on constate une suite logique de 3.8, soit environ 4 par pas de 10km/h.

### 3.3 Etude pratique : Comodo de phares

**Quels identificateurs de trames ont un rapport avec les commandes d'éclairage, de signalisation et d'essuyage ?**

Lorsque nous actionnons les commandes d'éclairage, de signalisation ou d'essuyage, nous observons des changements sur la trame 94, avec le premier octet commandant l'éclairage, le deuxième les essuie-glaces et le troisième la signalisation.

**Sur quels bus avez-vous observé ces identificateurs ?**

Nous observons que la trame 94 circule sur le bus carrosserie.

**Quelle est la taille de la trame concernée ?**

La taille de la trame 94 est de 5 octets.

**Actionnez la commande d'éclairage et établissez un tableau des fonctions commandées.**

Après avoir fait varier la commande d'éclairage entre feux de position, feux de croisement, feux de route et anti-brouillard, nous avons reporté dans le tableau 3 les valeurs prises par le premier octet de la trame 94.

fonction comandée	Premier octet (hexa)
Feux éteints	20
Feux de position	40
Feux de croisement	80
Feux de route	98 (au moment de l'activation)
Anti-brouillard	84 (au moment de l'activation)
Appel de phare	28 (au moment de l'activation)

TABLE 3 – Premier octet de la trame 94 en fonction de la commande d'éclairage.

### 3.4 Etude pratique : Rétroviseurs

**Faites fonctionner les rétroviseurs du côté droit et du côté gauche, et soyez attentif aux trames. Quel est l'identificateur de trame correspond aux rétroviseurs gauche et droit ? Sur quel bus avez-vous fait l'observation ?**

Lorsque nous faisons fonctionner les rétroviseurs avec le sélecteur et les boutons directionnels, nous constatons des changements sur la trame 115. Cette trame est présente sur le bus confort.

**Quelle est la taille des données de cette trame ?**

La trame 115 a une taille de 4 octets.

**A quoi sert le premier octet de donnée ?**

En observant la trame 115, nous constatons que les 4 premiers bits du premier octet contrôlent la direction et les 4 derniers bits du deuxième octet contrôlent le choix entre le rétroviseur gauche et droit.

**Observer le premier octet de donnés et établissez un tableau des actions réalisées en fonction de la valeur de l'octet.**

Nous avons reporté les actions réalisées en fonction des 4 premiers bits du premier octet dans le tableau 4 et ceux des 4 derniers bits du premier octet dans le tableau 5.

Actions réalisées	4 premier bits (hexa)
Aucune direction	0
Bas	1
Haut	2
Gauche	4
Droite	8

TABLE 4 – Direction du rétroviseur en fonction de la valeur des 4 premiers bits du premier octet.

Actions réalisées	4 derniers bits (hexa)
Aucun selectionné	1
rétroviseur droit	2
rétroviseur gauche	4

TABLE 5 – Choix du rétroviseur en fonction de la valeur des 4 derniers bits

Utiliser le générateur interactif afin d'émettre la trame de commande des rétroviseurs observée précédemment. Vérifiez les trames de commandes observées précédemment en associant des touches de votre clavier pour piloter les rétroviseurs. Vous pouvez faire de même avec les lève-vitres ?

Pour pouvoir utiliser le générateur interactif sur le logiciel MuxTrace, nous devons tout d'abord l'activer pour le bus CAN, comme montré dans la figure 4.

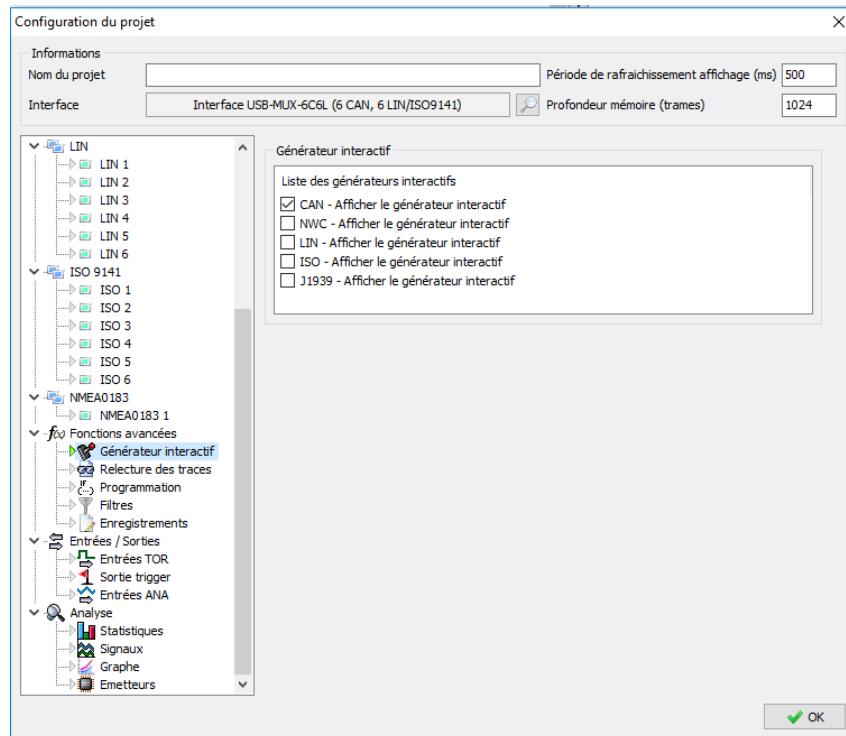


FIGURE 4 – Activation du générateur interactif pour le bus CAN sur MuxTrace

Nous pouvons ensuite ouvrir le générateur interactif de trames CAN via le menu déroulant "Fenêtres", comme illustré dans la figure 5.

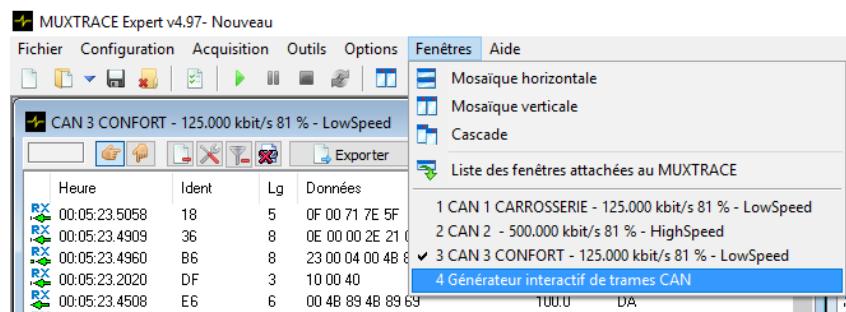


FIGURE 5 – Ouvrir le générateur interactif de trames CAN sur MuxTrace

Nous avons ensuite créé une trame d'ID 115 avec comme données 82 00 00 00, ce qui correspond à actionner le rétroviseur droit vers la droite. Cette trame est émise sur le bus confort, qui correspond au CAN 3 dans notre cas. De plus, nous avons configuré la touche "U" du clavier pour émettre la trame. La trame créée est illustrée à la figure 6.

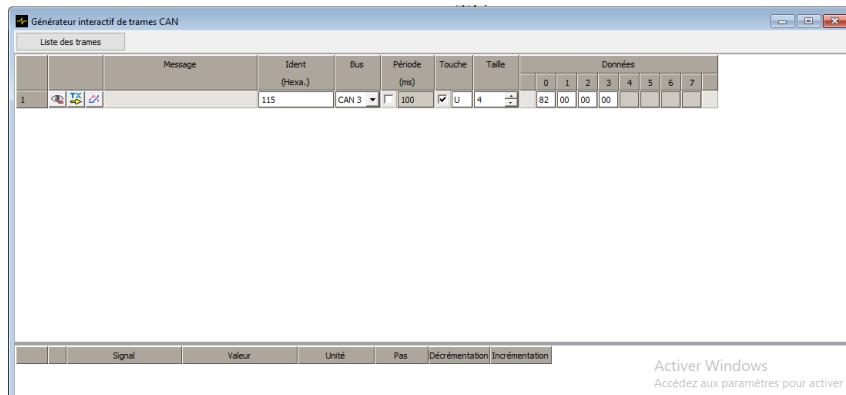


FIGURE 6 – Trame de contrôle du rétroviseur droit crée dans le générateur interactif

### 3.5 Etude pratique : Faites votre choix

En fonction de votre avancement dans la séance de TP, vous pouvez exposer une ou deux trames supplémentaires de votre choix sur une ou plusieurs bus de la maquette. De la même manière que précédemment, donnez la taille de données, vos observations sous forme de tableau et justifiez vos analyses.

- Commande radio :

Nous allons dans cette partie déterminer la trame responsable des commandes radio. Pour cela, nous avons actionné les commandes radio à partir des boutons sous le volant et nous avons observé un changement sur la trame 21F du bus CAN carrosserie, notamment sur le premier octet. Nous avons reporté les commandes radio en fonction du premier octet de la trame 21F dans le tableau 6.

Actions réalisées	Premier octet (hexa)
SRC	02
Augmentation volume	08
Diminution volume	04
Station suivante	80
Station précédente	40

TABLE 6 – Commande radio en fonction de la valeur du premier octet de la trame 21F

## 4 TP2 : Utilisation du logiciel MuxTrace et DBEdit

Basé sur les observations faites dans la section précédente, créer une base de donnée pour chaque bus et chaque identifiant de paquets analysées.

À partir de ce qui était obtenu précédemment, une base de données est créé, qui comportera toutes les trames importantes étudiées. Voici ci-dessous un exemple 7 de création de trame, dans cet exemple ce sera la trame liée au régime :

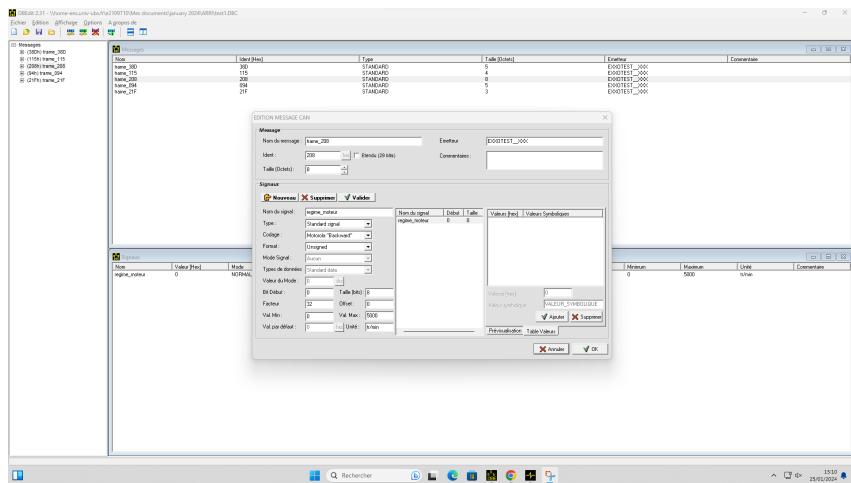


FIGURE 7 – Création de trame DBEdit.

Il y a plusieurs choses que nous pouvons observer sur cette liste :

- Tout d'abord on doit donner l'identité de la trame qui correspond dans ce cas-ci à celle du régime moteur (208).
- Ensuite on donne le nombre d'octets que contient la trame (8, dans cet exemple).
- Après cela on ajoute une valeur maximale et minimale (de 0 à 5000, dans cet exemple).
- On peut aussi ajouter un offset si on en a besoin (0, dans cet exemple)
- La dernière chose qui a changé est le "Facteur", qui représente le pas moyen entre chaque valeur (1000 tr/min dans ce cas).

**Lorsque les fichiers de base de données sont prêts, chargez-les depuis votre projet MuxTrace. Vérifiez que vos observations sont cohérentes lorsque vous actionnez les différentes commandes concernées de la maquette.**

Une fois toutes les trames ajoutées dans la base de données sur DB Edit, on va utiliser MuxTrace dans le but d'observer que nos choix lors de la création de cette dernière sont cohérents. Après avoir ajouté notre base de données on viendra changer quelques actionneurs sur la maquette, qui sont bien sûrs en lien avec les trames ajouté dans notre base de données, comme par exemple la trame liée au régime moteur.

Grâce à ça, on a pu vérifier que notre base de données correspondait avec ce qui est fait via la maquette.

### Qu'en déduisez-vous de l'intérêt de ces bases de données ?

L'intérêt des bases de données, est de repérer les trames que nous voulons étudier pour ensuite agir dessus et vérifier que nous obtenons bien le comportement souhaité.

**Utilisez maintenant le générateur interactif de MuxTrace pour faire varier les champs de données d'un ID de trame connu dans la base de données. Quel est l'intérêt de ce lien avec la base de donnée par rapport aux signaux d'une trame ?**

Pour cette question dans MuxTrace, on ajoute un générateur interactif, avec ce dernier, on viendra faire varier certaines trames pour ensuite vérifier si on bien le bon rendu sur la maquette. Donc par exemple si on augmente la trame correspondant au régime moteur alors sur la maquette on pourra aussi observer une augmentation. Une fois cela fait on vérifie si les valeurs que nous avons définies lors de la création de la base de données son juste ou non, en comparant tout simplement la valeur espérée valeur réelle lire sur la maquette. En fonction du résultat on ajuste ou non notre "facteur", par exemple. Voici si un exemple 8 de DBEdit dans MuxTrace :

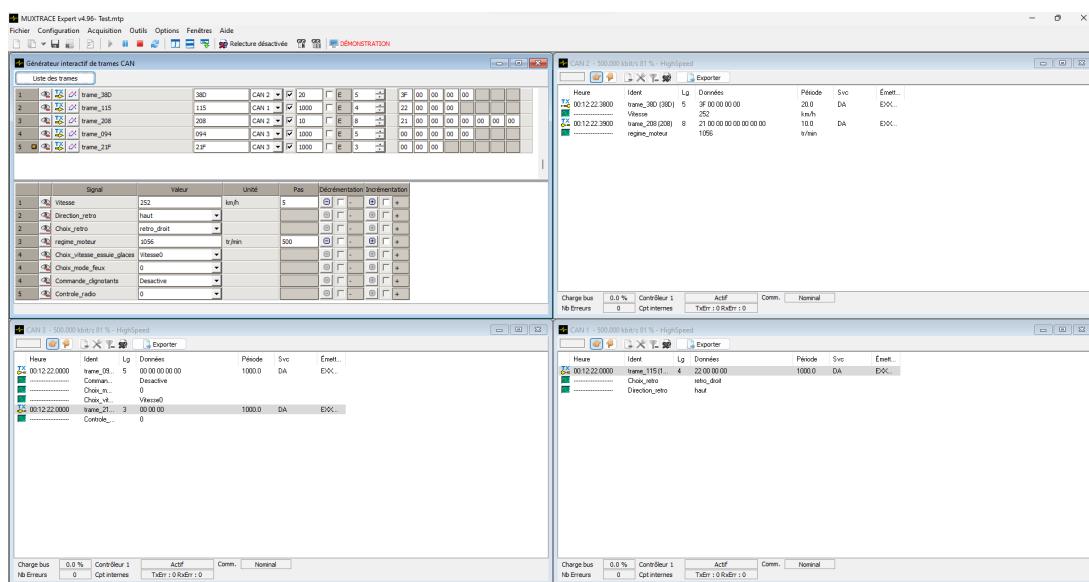


FIGURE 8 – Utilisation d'une base de donnée DBEdit sur MuxTrace.

On peut observer dans le cas ci-dessus qu'on a notre base de données avec nos trames à l'intérieur ainsi qu'une touche qui est affectée pour choisir si oui ou non ont envoyé sur le bus CAN une commande. Mais il y a aussi bien d'option, par exemple nous ne sommes pas obligés à utiliser une touche mais plutôt d'envoyer de façon périodique une commande. Une fois nos paramètres choisis, il nous reste les tests.

## 5 TP3 : Utilisation du mode programmation sous Mux-Trace

*Sous MuxTrace, il est possible d'avoir un mode programmation qui permet d'automatiser la génération interactive de trames sur des actions clavier ou sur des évènements conditionnels liés à la configuration du véhicule.*

### Etude pratique : chargement d'un exemple

Dans un premier temps, on vous demande de charger le fichier d'exemple de dll fournit en séances de TP dans l'outil MuxTrace.

Analysez et observez cet exemple et donnez vos remarques sur le comportement et les interactions de cet exemple sur la maquette.

La figure 9 montre le bon fonctionnement du programme MuxTrace après sa configuration avec l'exemple de fichier .dll sans aucune configuration supplémentaire.

Dans l'Annexe B - Code 1 se trouve le code original de la fonction OnEvent chargée de mettre à jour périodiquement les données de chaque trame sur les trois bus comme le montre la figure 9.

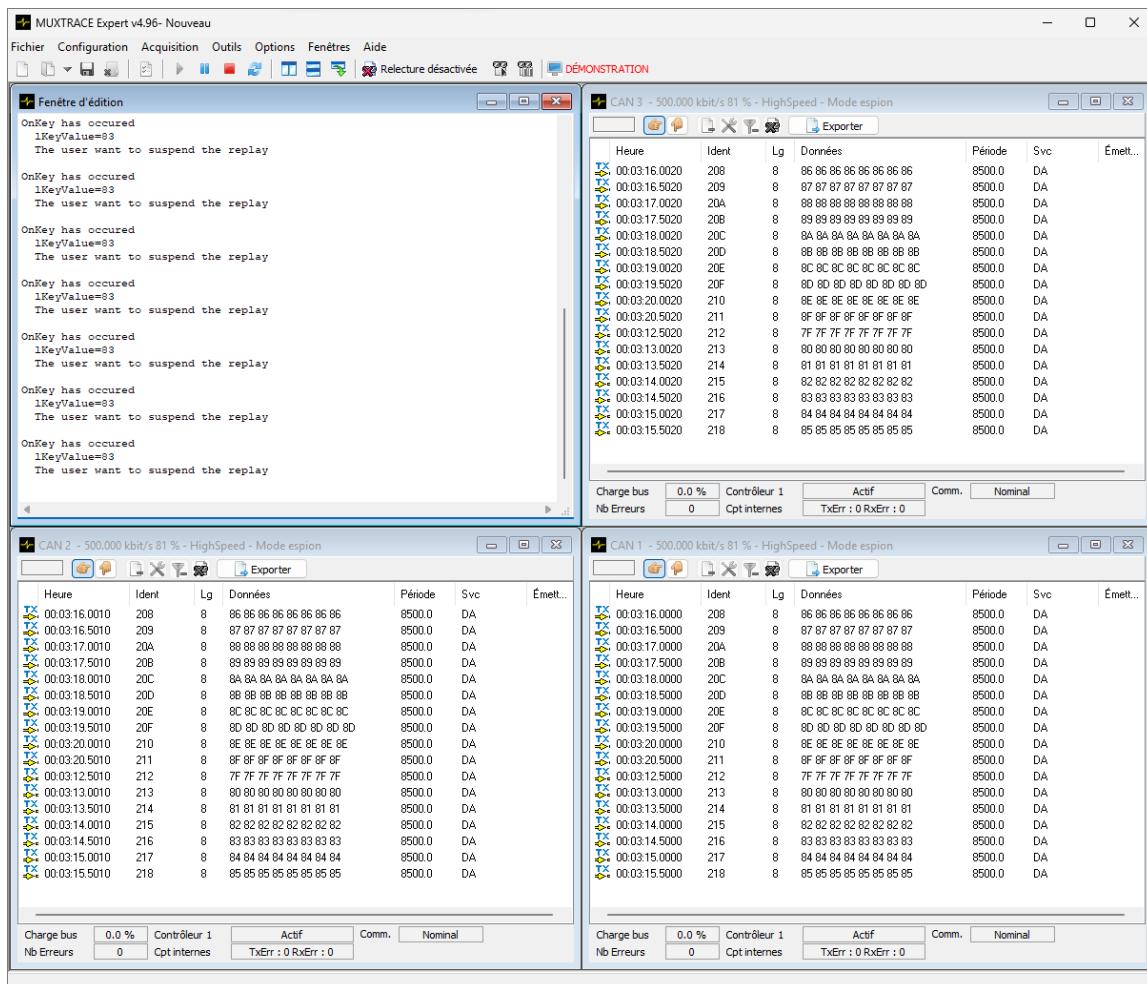


FIGURE 9 – Test local du fichier par défaut .dll.

### Etude pratique : Création de votre exemple

Proposez un scénario d'exemple que vous souhaitez réaliser et décrivez-le de manière détaillée (identifiants de paquets, événements conditionnels, ...). Ces précisions porteront une attention toute particulière pour évaluer le Compte-rendu de TP avec le code source et la dll qui sera remis en fin de séances de TPs.

Développez votre programme en C en partant de l'exemple fourni dans le répertoire d'installation de MuxTrace afin de créer votre dll d'exemple qui réalisera le scénario que vous aurez inventé.

En groupe, l'idée était de démontrer l'utilisation et la gestion correctes des châssis et des bus avec une relation entre action et réaction, c'est-à-dire qu'une action dans un système du simulateur de véhicule générera une réaction dans un autre système. De cette manière, la logique présentée dans la figure 10 a été développée.

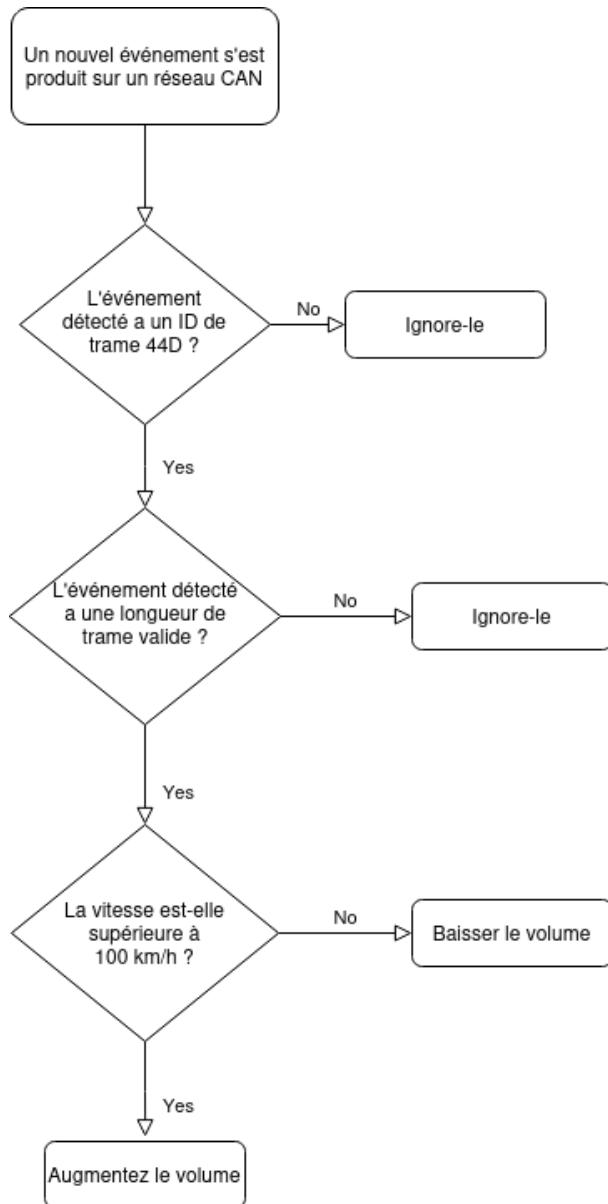


FIGURE 10 – Diagramme UML du scénario proposé pour développer le test TP3.

Pour réaliser la logique de la figure 10 la fonction *OnCanEvent* a été modifiée comme on peut le trouver dans l'Annexe B - Code 2.

En plus, dans l'annexe A, dans la section A.2, la figure 17 montre la configuration supplémentaire à effectuer dans MuxTrace pour l'implémentation du fichier '.dll'. En

outre, les figures 18 et 19 montrent un avant et un après la modification manuelle de la valeur de la vitesse dans MuxTrace et le changement qu'elle produit dans l'image 44D (volume), ce qui permet de vérifier que le code développé fonctionne correctement.

## 6 Conclusion

Lors des différents TP que nous avons faits, nous avons utilisé une maquette qui représente une voiture avec tous ses accessoires qui lui sont liés. Cela nous permet donc de travailler avec un réseau de terrain, à savoir le bus CAN. Nous avons donc pu comprendre ce dernier avec tous les outils qui nous ont été fournis et qui ont été présentés précédemment. Une fois la prise en main des différents outils effectuée, un script a été réalisé, que l'on peut voir dans le TP3.

## Références

- [1] Multiplexage can hs, can ls, lin.
- [2] L'obd c'est quoi ?

## A Annexe A : Illustration complémentaire

### A.1 TP1

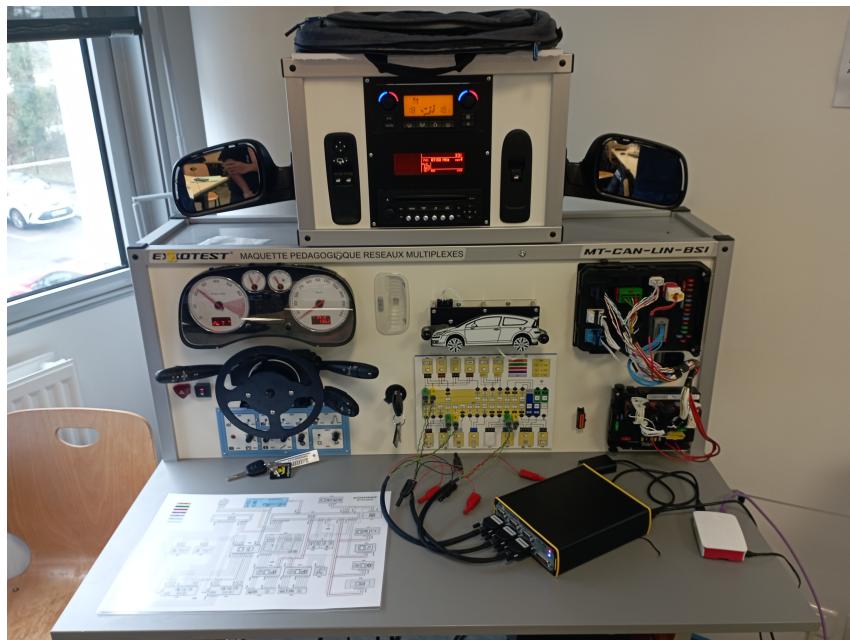


FIGURE 11 – Illustration de la maquette EXXOTEST.

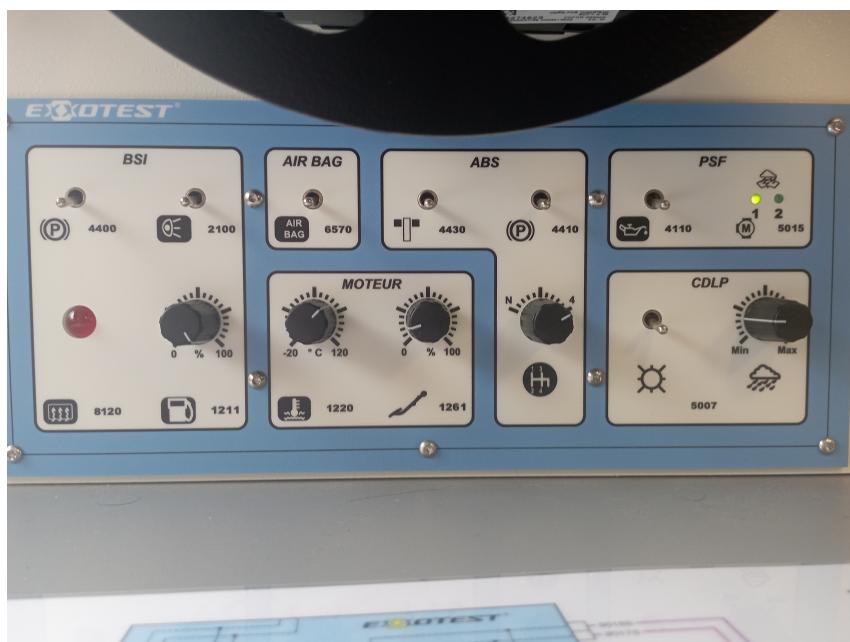


FIGURE 12 – Illustration des potentiomètres et interrupteurs de la partie simulée.

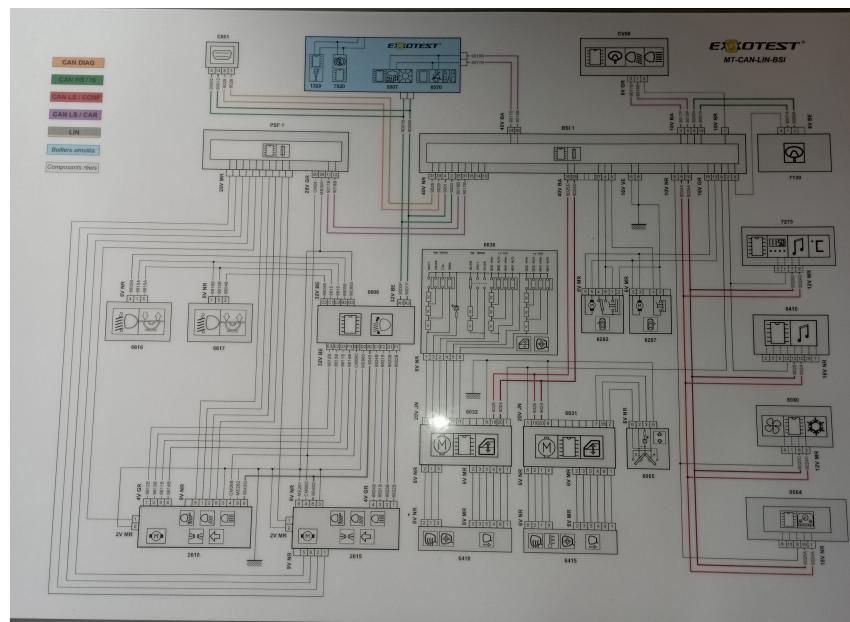
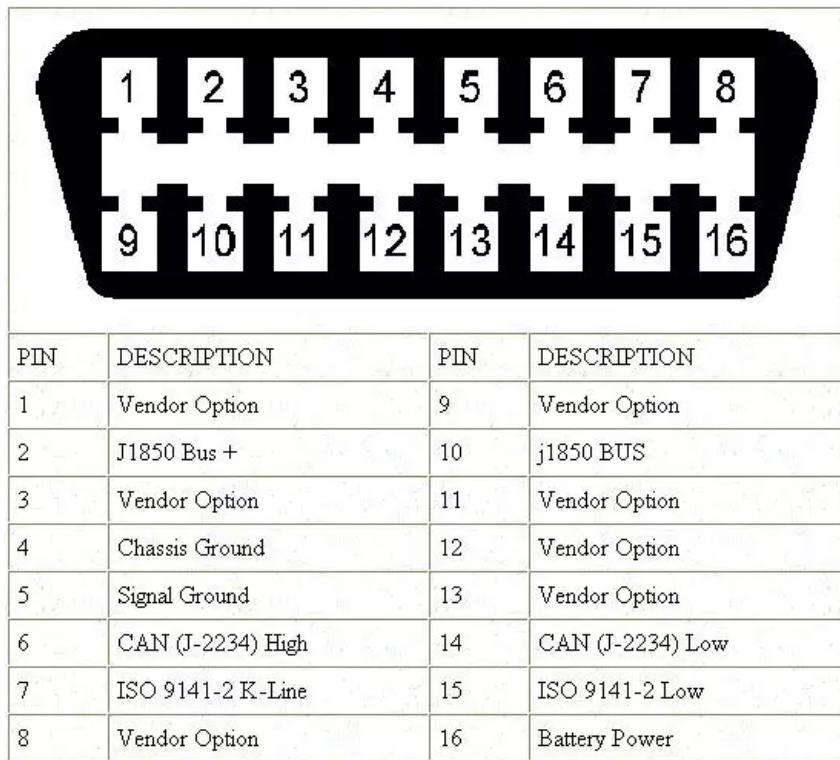


FIGURE 13 – Schéma du cablage de la maquette.



OBD-II Connector and Pinout

FIGURE 14 – Port OBDII.

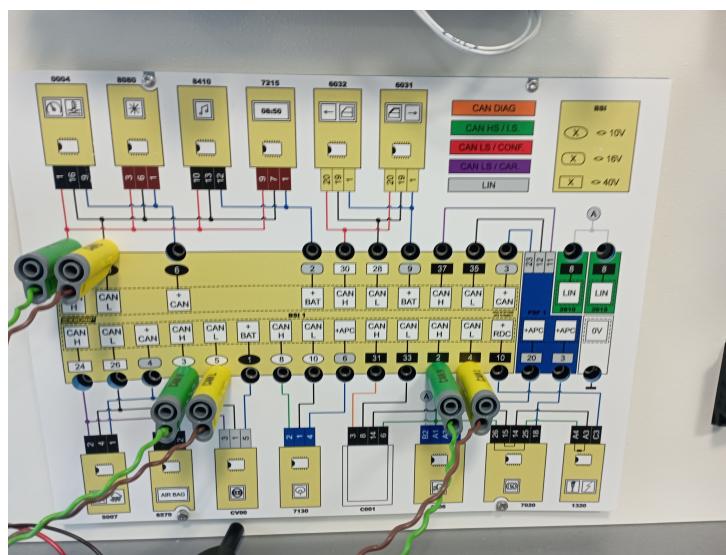


FIGURE 15 – Illustration des branchements sur la plaque de la maquette.



FIGURE 16 – Illustration du boîtier USB-MUX-6C6L.

## A.2 TP3

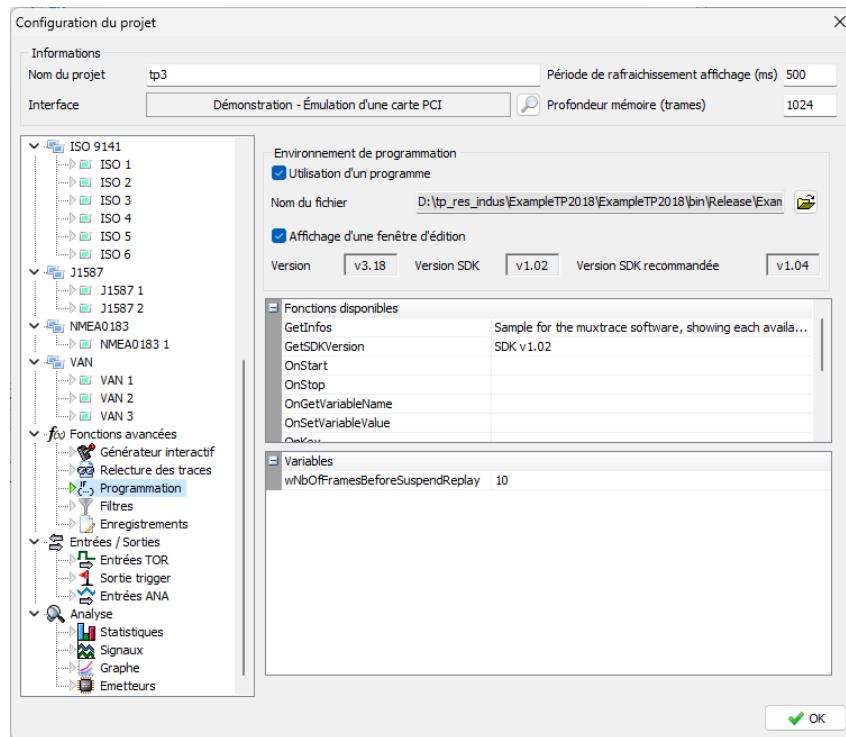


FIGURE 17 – Configuration supplémentaire du projet pour ajouter l’archive .dll afin de réaliser le test réel vitesse-volume.

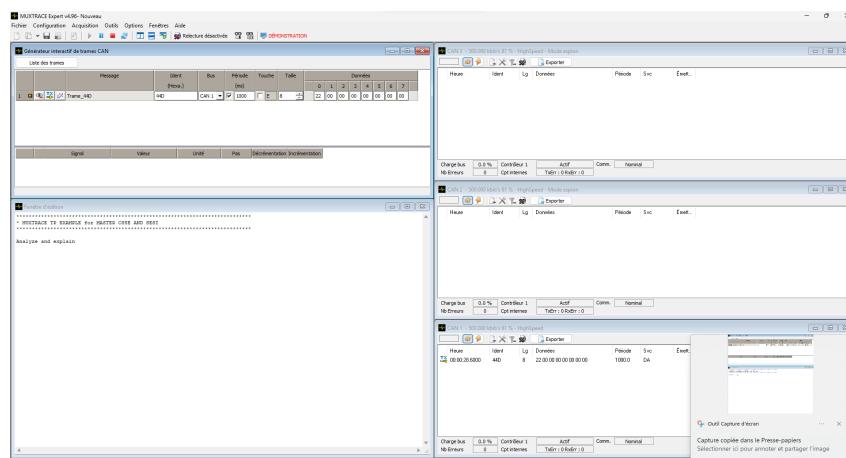


FIGURE 18 – Test local avec une vitesse inférieure à 100 km/h. Trame de volume (ID 44D) par défaut 22.

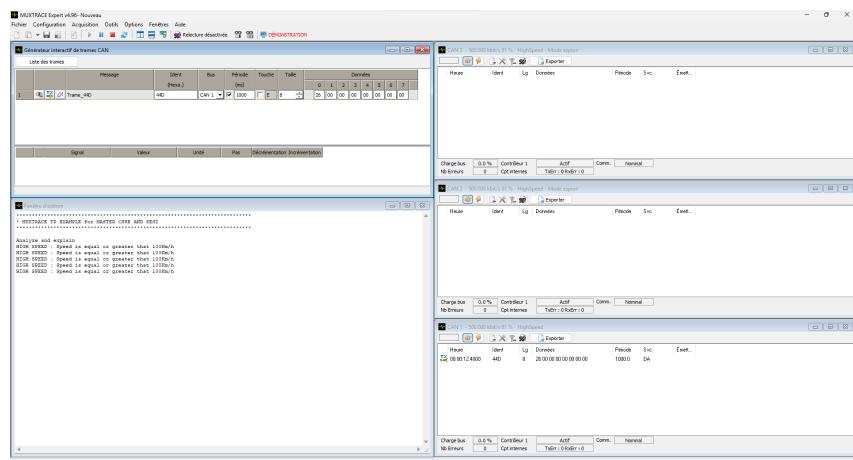


FIGURE 19 – Essai local avec une vitesse supérieure à 100 km/h, noter le message de débogage et la valeur de la trame 44D (volume) passe de 22 à 26.

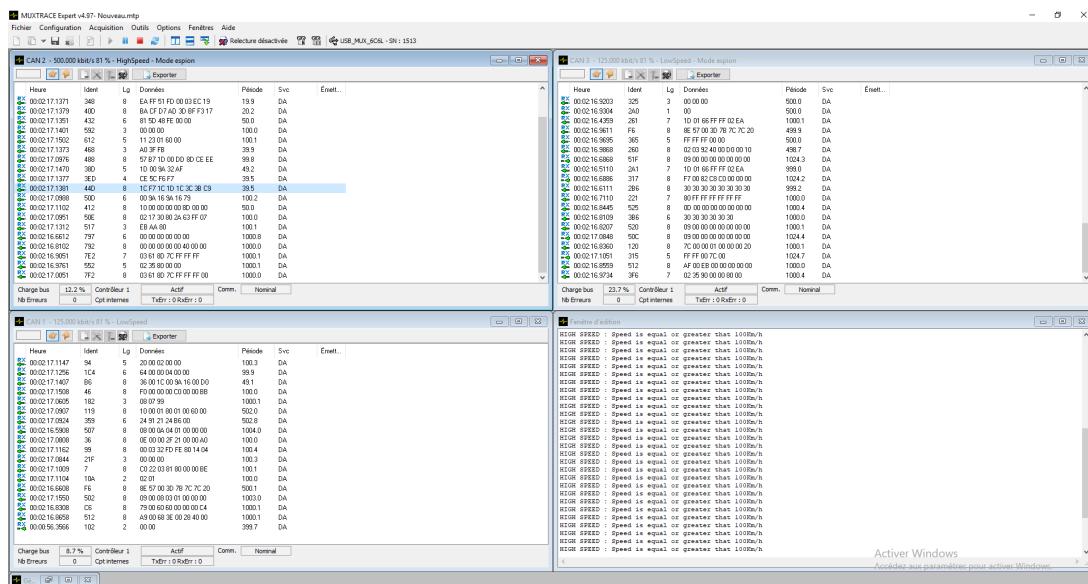


FIGURE 20 – Illustration du test vitesse-volume réel

## B Annexe B : Code Snippets

```

1  /* OnTimer
2   Called each time the timer counter has changed.
3
4   Parameters
5   dwTime > IN > New value of the timer counter , defined in milliseconds .
6
7   Return values
8   Should return 0 on error
9 */
10 _EXTERNC int _MUXPRG OnTimer(DWORD dwTime)
11 {
12     char szOut[100];
13     int i;
14     if( (dwTime%5000)==0)
15     { // Each 5 seconds only
16         DisplayMsg("OnTimer has occured");
17         sprintf(szOut, " dwTime=%d", dwTime);
18         DisplayMsg(szOut);
19         DisplayMsg("");
20         i=0;
21     }
22     if( (dwTime%500)==0) // Every 500ms we enter here
23     {
24         // Send a CAN packet on CAN 0
25         CanSendMsg( 0, 0, &g_hCanMsgTx );
26         // Send a CAN packet on CAN 1
27         CanSendMsg( 0, 1, &g_hCanMsgTx );
28         // Send a CAN packet on CAN 2
29         CanSendMsg( 0, 2, &g_hCanMsgTx );
30         g_hCanMsgTx.dwIdent++;
31         if(g_hCanMsgTx.dwIdent > 0x218)
32         {
33             g_hCanMsgTx.dwIdent = 0x208;
34         }
35         for( i=0;i<8;i++)
36         {
37             g_hCanMsgTx.bData[ i]++;
38         }
39     }
40
41     return(1);
42 }
```

Listing 1 – Fonction OnEvent original.

```

1  /* OnCanEvent
2   Called each time an event has occured on a CAN network .
3
4   Parameters
5   hCanEvent > IN > Definition of the event ( see refmux.h and documentation files ) .
6
7   Return values
8   Should return 0 on error
9 */
10 _EXTERNC int _MUXPRG OnCanEvent(tCanEvent *hCanEvent)
11 {
12     char msgDisplayed[100];
13
14     // 100km -> 0x13 -> 00100110
15     const char vitesse_limit = 0x26;
16     const char volume_up = 0x08;
17
18     const unsigned short XCAN_SI = 0; // default set in MUXTRACE
19     const unsigned short XCAN_CONF = 1;
20     const unsigned short XCAN_CAR = 2;
21
22     // Detect events with ID 0x44D (change in speed)
23     if(hCanEvent->dwIdent == 0x44D)
24     {
```

```

25     if(hCanEvent->wDataLen >= 1) // Security that data exists
26     {
27         // const char that stores the first byte of Detected Event
28         const char firstByte = hCanEvent->bData[0];
29
30         // If speed read is greater than defined limit
31         if(firstByte >= vitesse_limit)
32         {
33             DisplayMsg("HIGH SPEED !!!");
34             sprintf(msgDisplayed, "Speed is : %d Km/h", firstByte * 10/4);
35             DisplayMsg(msgDisplayed);
36
37             // Configure auxiliar 'tCanMsg' to volume frame
38             g_hCanMsgTx_aux.dwIdent = 0x21F;
39             g_hCanMsgTx_aux.wDataLen = 3;
40             g_hCanMsgTx_aux.bData[0] = 0x08;
41             // Turn volume UP !!!
42             CanSendMsg(0, XCAN_CAR, &g_hCanMsgTx_aux);
43         }
44         else
45         {
46             // Configure auxiliar 'tCanMsg' to volume frame
47             g_hCanMsgTx_aux.dwIdent = 0x21F;
48             g_hCanMsgTx_aux.wDataLen = 3;
49             g_hCanMsgTx_aux.bData[0] = 0x04;
50             // Turn volume DOWN :/
51             CanSendMsg(0, XCAN_CAR, &g_hCanMsgTx_aux);
52         }
53     }
54
55
56     return(1);
57 }
```

Listing 2 – Fonction OnCanEvent modifie.