Banc de développement et de validation d'un système embarqué automobile

Éric Lepoutre

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PARIS

_____

MEMOIRE

Présenté en vue d'obtenir

le DIPLOME d'INGENIEUR CNAM

SPECIALITE : Electronique

OPTION : Electronique

par

(Eric Lepoutre)

_____

Banc de développement et de validation d'un système embarqué automobile

Soutenu le 02 Juillet 2013

JURY

PRESIDENT : -Didier Le Ruyet(Professeur)

MEMBRES : -Pierre Provent(Maître de conférence)

-Christian Pautot (Maître de conférences)

-Henri Belda (Société Vector)

-Wasilios Souflis (Société Vector)

Remerciements

Je tiens à remercier la société Vector-France pour m'avoir donné la possibilité de réaliser ce projet, Mr Belda et Mr Souflis, qui ont mis à ma disposition les moyens de développer mon étude.

Je souhaite également remercier la société Sojadis, pour nous avoir confié leur matériel et partager leurs connaissances, en particulier Mr Tulasne pour son aide et ses renseignements.

Enfin, je souhaite remercier Mr Provent pour le soutien et les conseils qu'il m'a apportés.

Merci à mes collègues et amis qui m'ont aidé au cours de mes recherches.

GLOSSAIRE

CAN	*Control Area Network*
CPU	*Central Processing Unit*
Noyau	*kernel* en anglais partie du système d'exploitation, permet aux différents composants matériels ou logiciels de communiquer entre eux.
Processus	ensemble d'instructions ou de tâches à exécuter.
ECU	*Electronic Control Unit,* terme générique pour les systèmes embarqués contrôlant un ou plusieurs sous systèmes d'un véhicule.
BSI	*Boîtier de Servitude Intelligent*. Calculateur PSA
CMB	Combiné/Tableau de bord véhicule PSA
IHM	*Interface Homme Machine.*
CAPL	*CAN Access Programming Langage*
VT-System	*Virtual Test-System*
XML	*Extensible Markup Langage* ou « langage de balisage extensible » en Français.
DBC	*Database For Communication.* Standard de fichier décrivant la messagerie d'un ou plusieurs réseaux
ROI	*Region Of Interest*. Sous-ensemble sélectionné d'échantillons au sein d'un ensemble de données identifié à un usage particulier
Firmware	microprogramme ou logiciel de contrôle d'un périphérique programmé directement en mémoire des puces de celui-ci
OEM	*Original Equipment Manufacturer.* Fabricant d'équipement d'origine, grand constructeur type Renault PSA
PIE	*plate-forme d'intégration électronique*

Transceiver	Emetteur récepteur, ici chargé de la réception et l'émission des données CAN sur le bus.
FIFO	*First In First Out.* Méthode de dépilement informatique des données
IL	*Interaction Layer.* Couche de communication lisant les paramètres de la base de données pour l'envoi des signaux sur le bus
DLL	*Dynamic Link Library.* Format de bibliothèque logicielle utilisant des liens dynamiques pour les associer aux programmes qui s'en servent
SYSVAR	Appellation CANoe de la variable système
TAE	*Test Automation Editor.* Logiciel de création de tests xml
CEM	compatibilité électromagnétique.

INTRODUCTION

De nos jours, l'ingénieur est chargé de participer à la conception de technologies innovantes, est amené à gérer des problèmes techniques dans un temps imparti, et aussi d'apporter des solutions en s'appuyant sur ses connaissances techniques et son savoir-faire. En outre, l'embarqué est un secteur d'activité où le nombre de calculateurs ne cesse de croître d'années en années, augmentant considérablement les problématiques de conception, de développement et de validation des organes. Le nombre de fournisseurs et la diversité des technologies de l'électronique automobile induisent des phases de développement transversales où la conception d'un organe est étroitement liée à celle des autres, et où l'ingénieur peut avoir à remettre en cause tout ou partie du projet lors de l'intégration de ces organes au sein du système. Ces technologies s'articulent autour d'un réseau de communication, sur lequel repose une architecture véhicule comportant une multitude d'ECU (Electronic Control Unit) gérant les fonctions principales du véhicule. Chaque ECU possède sa messagerie de communication avec le réseau et une quantité d'entrées et sorties analogiques interfaçant l'ECU avec son environnement physique.

Au sein de la société Vector-France et en tant qu'ingénieur d'application, le but de ce projet sera de répondre aux problématiques soulevées en adoptant le rôle de l'équipementier et en recréant à travers les différentes phases de développement d'un ECU, un banc de simulation complet d'un réseau véhicule, afin d'y intégrer réellement un module de commande auxiliaire d'aide à la conduite. Ensuite, nous serons amenés à tester cet ECU dans différentes phases de fonctionnement, aussi bien nominal que dégradé grâce à la stimulation des entrés analogiques de l'organe. En recréant l'univers physique du calculateur, il sera nécessaire de configurer des cartes d'interfaçage I/O afin de simuler et de générer les signaux attendus par l'ECU. La validation des résultats obtenus s'appuiera sur les spécifications fournisseur.

Nous apporterons à l'entreprise un démonstrateur des méthodes de conception d'un organe véhicule à travers toutes les étapes du cycle en V. Par ailleurs, le fournisseur attend de nous un banc de simulation et de tests de son calculateur, pouvant s'intégrer dans les étapes de développement et de validation de son produit. Le mémoire de l'ingénieur doit poser le cadre du métier de l'ingénieur, et à travers les défis techniques imposés par le milieu industriel, répondre aux problématiques évoquées.

Chapitre I Un banc de développement, de simulation et de test d'un calculateur

Avant toute chose, la mise ne place d'un banc de développement d'un calculateur nécessite de resituer les exigences fournisseur, déterminant la qualité du produit sur lequel le projet va aboutir. A travers les spécifications du produit, nous devrons prendre en compte les contraintes de temps et de moyens techniques inhérents à tout projet. Resituer les notions techniques fondamentales des systèmes embarqués nous amènera à mieux comprendre les tenants de notre projet.

1. Vector

Le groupe Vector est un fournisseur de solutions logicielles et matériel pour le développement de systèmes embarqués [3]. Les secteurs d'activité sont principalement l'automobile ainsi que le transport au sens large (véhicules lourds, aéronautique, ferroviaire…).

1.a Présentation du Groupe Vector

LE SIEGE DE VECTOR INFORMATIK A STUTTGART

Vector France est une filiale du groupe Vector Informatik qui a été créée en 2002 par le Président actuel Monsieur Henri Belda. Vector est l'un des principaux éditeurs d'outils et composants logiciels pour la mise en réseaux de systèmes électroniques basés sur les

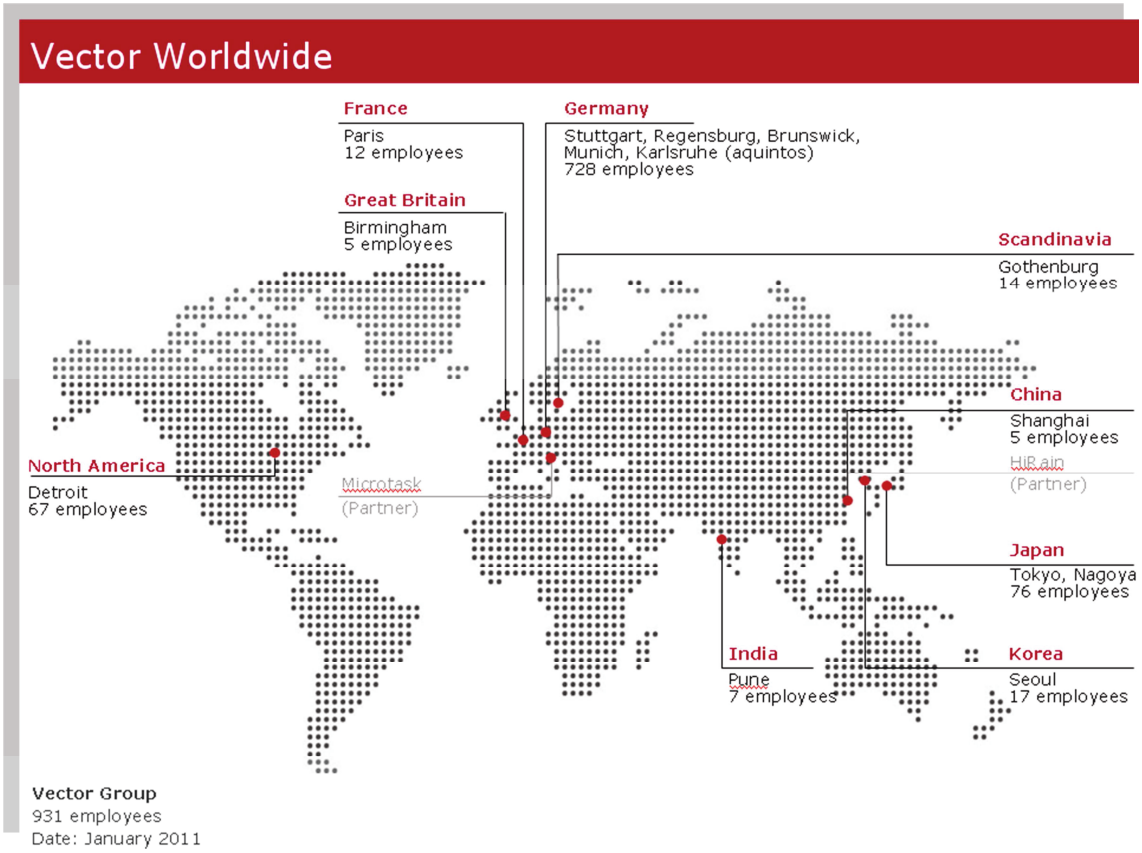
protocoles de communications CAN, LIN, MOST et FlexRay ainsi que sur les protocoles dérivés du CAN. La société diffuse son expertise sous la forme de produits ou de solutions de conseil, incluant l'ingénierie des systèmes et des logiciels. Des journées techniques et séminaires viennent compléter son programme de formation. A l'heure actuelle, le chiffre d'affaires dépasse les 200 millions d'euros et on recense plus de 1000 employés.

Lieu du Projet	Vector France
Adresse	168, Boulevard Camélinat 92 240 MALAKOFF
Téléphone	+33 1 42 31 40 00
Fax	+33 1 42 31 40 09
Site Internet	http://www.vector.com/vf_index_fr.html
Président	Henri BELDA
Activité	Conseil en systèmes et logiciels informatique
Chiffre d'affaires (2011)	5 920 000 €
Capital Social (2012)	200 000 €
Effectif moyen (2012)	14 personnes
Immatriculation	Mars 2002 au RCS de Nanterre sous le numéro 441 295 375
Siret	441 295 375 000 14
Code APE	6202A
Forme juridique	Société par Actions Simplifiée

1.b Vector Dans le Monde

Le groupe Vector se démarque de par sa dimension mondiale, traduit par une implantation dans des régions où l'automobile constitue un marché porteur ou est sur le point de le devenir. La maison mère, basée à Stuttgart est un point stratégique puisqu'elle constitue un centre international pour l'industrie automobile, à proximité de nombreux constructeurs et équipementiers susceptibles d'utiliser les produits Vector. Vector est représenté par l'intermédiaire de ses filiales et distributeurs dans les autres secteurs géographiques où l'activité automobile est importante.

Le document ci-dessous (Figure 2) présente la localisation des différents établissements Vector sur le globe. Parmi les filiales importantes, il faut citer l'Amérique du nord et le Japon où le nombre d'employés est plus important puisque proche des constructeurs, on y retrouve une partie de développement (*ex : Vector CANtech, Inc. à Detroid et Ford et General Motor*).



VECTOR DANS LE MONDE

- Vector est indépendant.
    - Depuis 2011, Le groupe Vector s'est transformé en fondation afin de garder une totale indépendance et ainsi être totalement orientée vers les attentes des clients
- Vector va de l'avant.

    La Société innove et développe aujourd'hui les technologies de demain.
- Vector est un partenaire.

    Son travail est défini par un échange bilatéral entre les clients et les employés.
- Vector est International.

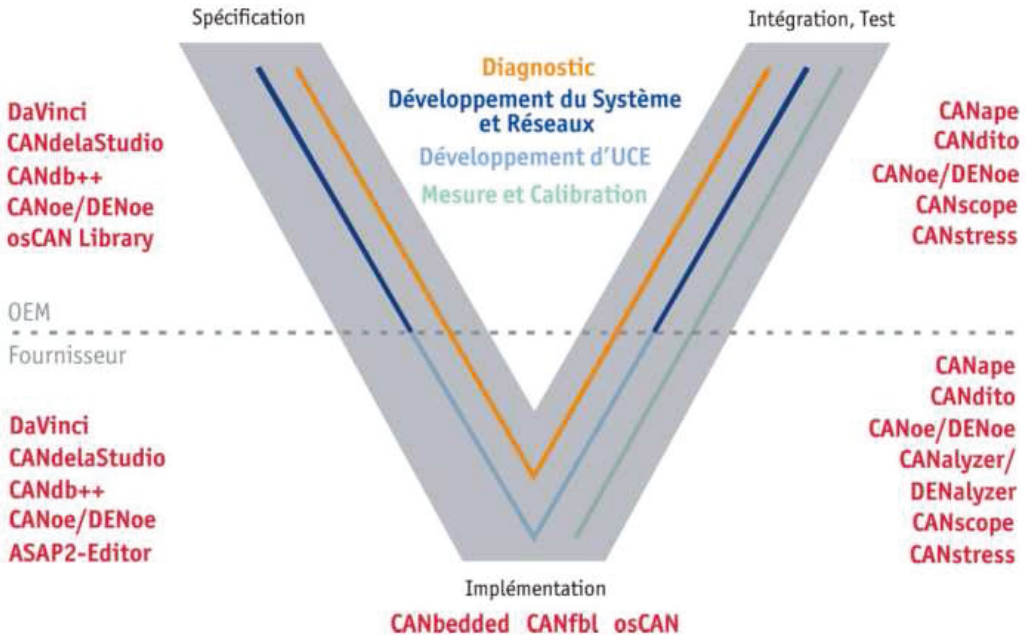
    L'entreprise est présente sur tous les marchés importants du monde au travers de filiales et de revendeurs.

1.c Historique

1er avril 1988	**Création de Vector Software GmbH (équivalent de SARL)**
1992	- Changement de nom en Vector Informatik. - Première licence CANalyzer déposée - Le chiffre d'affaire dépasse le million d'euro
1996	Livraison de la première licence CANoe et CANape
1997	Création de Vector CANtech aux USA
1998	- Création de Vector Japon - Vector est certifiée ISO 9001 : 1994
1999	Le 100ème employé rejoint le groupe Vector
2000	Le chiffre d'affaire du groupe Vector dépasse les 30 millions d'euros
2002	Création de Vector France et de VecScan en Suède
2003	Le chiffre d'affaire du groupe Vector atteint les 60 millions d'euros et compte plus de 400 employées
2004	Création des antennes clients au Nord de l'Allemagne (Braunschweig) et sud (Munich)
2005	Le groupe Vector dépasse les 500 employés dans le monde
2006	- Vector rachète la division 4m Software de Micron Electronics. - le chiffre d'affaire dépasse les 100 millions d'euros
2007	- Vector compte plus de 750 employés - Création de la filiale en Corée
2008	Construction d'un troisième bâtiment à Stuttgart
2009	- Le groupe compte plus de 900 employés - Création de Vector Grande-Bretagne, Inde et Chine
2010	- Aquintos devient une filiale du Groupe Vector

1.d L'offre

A chaque étape du cycle de développement en V d'un calculateur embarqué, depuis la phase de spécifications jusqu'à la phase de tests du système, correspond au moins un produit Vector[3].



LES PRODUITS VECTOR DANS LE CYCLE EN V

Ces produits concernent le développement de systèmes embarqués basés sur les technologies CAN, LIN, MOST, FlexRay, Autosar, IP ainsi que sur de multiples protocoles basés sur le CAN. Ils touchent les domaines suivants :

- Développement de systèmes distribués
- Logiciels pour ECU
- Test d'ECU
- Calibration d'ECU
- Diagnostic véhicule
- Management de processus

L'offre Vector est déclinée en dix lignes de produits présentés ci-dessous :

PND - Product line tools for Networks and Distributed systems

Outils logiciels pour le développement des systèmes distribués (Simulation, test analyse réseaux).

PMC - Product line tools for Measurement and Calibration

Outils de mesure et de calibration des modules ECU.

PDG - Product line Diagnostics

Outils de diagnostics pour la conception d'architectures organisés en trois types :

Spécification et description des besoins diagnostics

Implémentation de fonctionnalités diagnostics sur ECU

Configuration des paramètres de test diagnostic.

PES - Product line Embedded Software

Composants logiciels embarqués pour les réseaux de l'industrie Automobile.

PON - Product line tools & components for Open Networks

Solutions pour les réseaux ouverts basés sur le protocole CAN.

PNI - Product line tools Networks Interfaces

Outils d'interface réseaux pour les protocoles CAN, LIN, MOST et FlexRay.

PTR - Product line Training

Séminaires et formations sur les produits, les protocoles et les standards.

PCO - Product line Consulting

Optimisation des processus de développement pour les systèmes électroniques et en particulier dans le secteur automobile.

PPT - Product line Process Tools

Outils de gestion des processus liés à l'ingénierie.

1.e Les clients

Le secteur d'activité de Vector est concentré autour des bus dits de terrain. De ce fait, les principaux clients du groupe sont les constructeurs automobiles, les équipementiers de différents niveaux, laboratoires de recherche, enseignement ainsi que les fondeurs de microcontrôleurs. Par expansion, les solutions du groupe sont aussi présentes dans bien d'autres secteurs : aéronautique, aérospatial, ferroviaire, machines agricoles, même si ces domaines ne constituent pas des marchés cibles pour le moment. Dans le cadre de

partenariat, Vector fournit aussi du matériel à des fins pédagogiques aux écoles et universités. A titre indicatif, la base de données des clients Français recense près de 7000 contacts, c'est-à-dire ayant formulé au moins une demande auprès de Vector France.

Face à l'étendue du portefeuille de solutions proposées par le groupe Vector, la concurrence est très diverse et s'entend souvent uniquement par lignes de produits.

2. Le projet

2.a Contexte

L'entreprise où se déroulera le projet est Vector-France qui est une entité du groupe Vector.

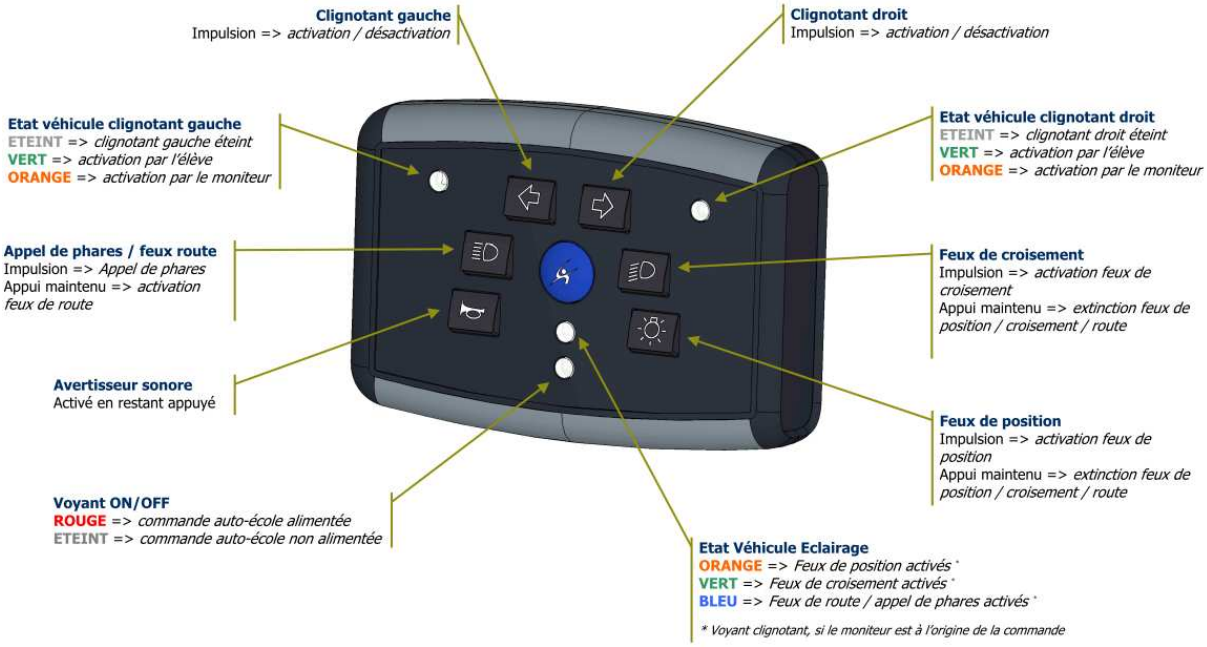
Le projet se déroulera au sein de l'équipe technique de Vector-France, en relation avec l'équipementier Sojadis. Un calculateur de commande électrique sera à ma disposition. L'équipe technique comprend aussi Mr Wasilios Souflis, ingénieur d'application responsable solution de simulation et expert en diagnostic. Cette équipe est liée à notre département développement en Allemagne avec lequel un nombre important d'ingénieurs peut être sollicité.

Le fournisseur du calculateur à l'étude est l'équipementier Sojadis, qui développe son activité dans le domaine de l'automobile, par la réalisation de commandes annexes aux véhicules multiplexés. Ces commandes sont réalisées avec l'accord des grands groupes

automobiles (PSA, RENAULT, VW,…) et destinées aussi bien à l'aide à la conduite pour les personnes handicapées que pour les auto-écoles en tant que commandes auxiliaires.

2.b Spécifications et attentes fournisseur

Sojadis a fourni à Vector un calculateur, le MONIDIS, dont les fonctions principales regroupent les commandes déportées d'éclairage du véhicule, ainsi que la signalisation par les clignotants et le klaxon.



Ce module, destiné en seconde monte, devra pouvoir être implanté sur le réseau CAN du véhicule afin de stimuler les organes de commande d'éclairage grâce à une adaptabilité à la messagerie véhicule configurée en amont, selon le modèle et la marque du véhicule auquel il est destiné.

Notre fournisseur attend une étude de la capacité du MONIDIS à s'implanter sur un réseau PSA(Peugeot-Citroën), et à délivrer au passager une commande complète des signalisations véhicules. Ayant pour but d'équiper les véhicules auto-écoles, le moniteur aura la capacité de prendre la main sur ces commandes de manière aussi réactive que si celles-ci étaient activées par le conducteur.

Le MONIDIS a une stratégie de contrôle et de signalétique précise, que notre modèle simulé devra reproduire avec exactitude.

Ce comportement doit s'appuyer sur la messagerie CAN suivante, étendue à partir de la messagerie PSA du véhicule afin d'apporter les fonctions supplémentaires du calculateur :

MESSAGERIE CAN (125kb/s, ID standards)

ID PERIODE Nb OCTETS

MESSAGES RECUS

0x094 100ms 7 ETAT HDC

0x0F6 500ms 3 ETAT VEHICULE

OCTET 0		
Bit 3	0	Clé sur position ARRET
	1	Clé sur position CONTACT

0x036 100ms 8 ETAT RESEAU

OCTET 4		
Bit 0	0	Envoi périodique de la trame d'état du MONIDIS désactivé
	1	Envoi périodique de la trame d'état du MONIDIS activé

0x046 100ms 8 COMMANDE_CONDUCTEUR

OCTET 0		
Bits 7-6	00	Commande feux de position désactivée
	11	Commande feux de position activée
Bits 5-4	00	Commande feux de croisement désactivée
	11	Commande feux de croisement activée
Bit 3	0	Commande feux de route désactivée
	1	Commande feux de route activée
OCTET 3		
Bit 7	0	Commande clignotant droit désactivée
	1	Commande clignotant droit activée
Bit 6	0	Commande clignotant gauche désactivée
	1	Commande clignotant gauche activée

MESSAGES EMIS

0x086 200ms 3 ETAT MONIDIS

OCTET 0		
Bit 7	0	Commande feux de route désactivée
	1	Commande feux de route activée
Bit 6	0	Commande feux de croisement désactivée
	1	Commande feux de croisement activée
Bit 5	0	Commande feux de position désactivée
	1	Commande feux de position activée
Bit 4	0	Commande extinction des feux désactivée
	1	Commande extinction des feux activée
Bit 3	0	Commande clignotant droit désactivée
	1	Commande clignotant droit activée
Bit 2	0	Commande clignotant gauche désactivée
	1	Commande clignotant gauche activée
Bit 1	0	Commande appel de phares désactivée
	1	Commande appel de phares activée
Bit 0	0	Commande klaxon désactivée
	1	Commande klaxon activée

TABLEAU I : SPECIFICATIONS DE BASE DE DONNEES

Le tableau suivant présente le principe de fonctionnement du calculateur :

	Effet Appui court	Voyant	Signal CAN	Effet Appui long(>1sec)	Temps de montée(sec)
Appui bouton feux de position	Allumage feux de position	Voyant orange clignotant	Commande feux de position	Extinction tout feux	0.000<S<0.200
Appui bouton feux de croisement	Allumage feux de croisement	Voyant vert clignotant	Commande feux de croisement	Extinction tout feux	0.000<S<0.200
Appui bouton feux de route	Appel de phares	Voyant bleu clignotant	Commande feux de route	Allumage feux de route	0.000<S<0.350
Appui bouton clignotant G ou D	Allumage clignotant G ou D	Voyant Orange fixe	Commande Clignotant G ou D	.	0.000<S<0.500
Appui bouton klaxon	Enclenchement klaxon	.	Commande klaxon	.	0.000<S<0.300

TABLEAU I I : STRATEGIE DE FONCTIONNEMENT DU CALCULATEUR

Le MONIDIS rentre en état fonctionnel uniquement sur certaines conditions, outre une alimentation 12V, le MONIDIS scrute le réseau CAN et attend que le signal ' Envoi périodique de la trame d'état du MONIDIS' de la trame 0X036 codé sur l'octet 4 en position de bit 0 soit à 1 pour activer sa communication sur le bus.

Le MONIDIS est spécifié pour une consommation maximum de 80mA, tous services enclenchés et en fonctionnement standard, de 40mA en moyenne. Ces consommations devront être validées par nos tests.

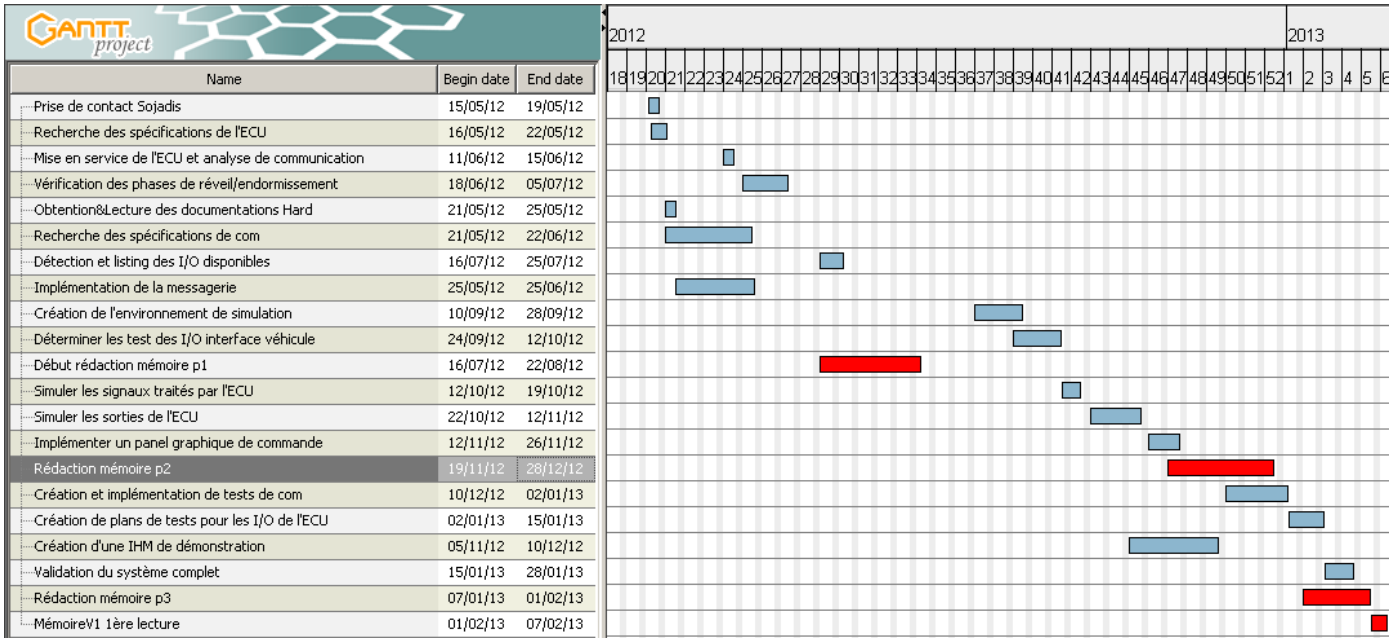
2.c Exigences de l'entreprise

En tant qu'ingénieur d'application au sein du département Vector-France, le but de ce projet est d'abord de démontrer un banc complet à notre fournisseur et client, SOJADIS, et de démontrer les capacités de nos outils à s'intégrer dans la chaîne de développement d'un calculateur.

Par ailleurs, ce banc sera le principal démonstrateur de la société pour nos journées techniques, nos interventions sur site client. Il devra être compatible aux autres produits Vector.

2.d Planning

Afin de respecter les attentes précédemment décrites, le planning suivant donnera les ordres de grandeurs des différentes étapes de la réalisation de ce projet.



Name	Begin date	End date
Prise de contact Sojadis	15/05/12	19/05/12
Recherche des spécifications de l'ECU	16/05/12	22/05/12
Mise en service de l'ECU et analyse de communication	11/06/12	15/06/12
Vérification des phases de réveil/endormissement	18/06/12	05/07/12
Obtention&Lecture des documentations Hard	21/05/12	25/05/12
Recherche des spécifications de com	21/05/12	22/06/12
Détection et listing des I/O disponibles	16/07/12	25/07/12
Implémentation de la messagerie	25/05/12	25/06/12
Création de l'environnement de simulation	10/09/12	28/09/12
Déterminer les test des I/O interface véhicule	24/09/12	12/10/12
Début rédaction mémoire p1	16/07/12	22/08/12
Simuler les signaux traités par l'ECU	12/10/12	19/10/12
Simuler les sorties de l'ECU	22/10/12	12/11/12
Implémenter un panel graphique de commande	12/11/12	26/11/12
Rédaction mémoire p2	19/11/12	28/12/12
Création et implémentation de tests de com	10/12/12	02/01/13
Création de plans de tests pour les I/O de l'ECU	02/01/13	15/01/13
Création d'une IHM de démonstration	05/11/12	10/12/12
Validation du système complet	15/01/13	28/01/13
Rédaction mémoire p3	07/01/13	01/02/13
MémoireV1 1ère lecture	01/02/13	07/02/13

Chapitre II Les outils de l'ingénieur en systèmes embarqués

Afin de comprendre dans quel milieu le projet sera déployé, il s'agit de resituer le CAN et les systèmes embarqués dans les technologies actuelles. Nous pourrons alors comprendre quelles sont les différents moyens de développement et d'aide sur lesquels l'ingénieur d'aujourd'hui peut s'appuyer. Enfin, nous présenterons les outils couramment utilisés pour simuler et analyser les réseaux des véhicules multiplexés.

1. Le bus CAN

1.a Origine

Au milieu des années 80, le nombre grandissant d'organes électroniques mis en perspective avec le prix et le poids du cuivre poussa les acteurs de l'industrie automobile à réduire les lignes dédiées entre organes. L'idée de multiplexer les réseaux véhicules fut rendu possible via la collaboration entre l'Université de Karlsruhe et Bosch, qui donna naissance au bus CAN.

Le CAN est par ailleurs actuellement utilisé dans la plupart des industries comme l'aéronautique via des protocoles standardisés basés sur le CAN.

Il fut présenté avec Intel en 1985, mais fut standardisé par l'ISO qu'au début des années 1990.

En 1992 plusieurs entreprises se sont réunies pour créer le CAN in Automation, une association qui a pour but de promouvoir le CAN[2].

1.b Structuration du protocole.

Chaque équipement connecté, appelé « nœud », peut communiquer avec tous les autres. L'échange de données entre nœuds est intégralement transmis par messages ou trames CAN dont la structure est la suivante.

La trame CAN se compose de 7 champs différents[3] :

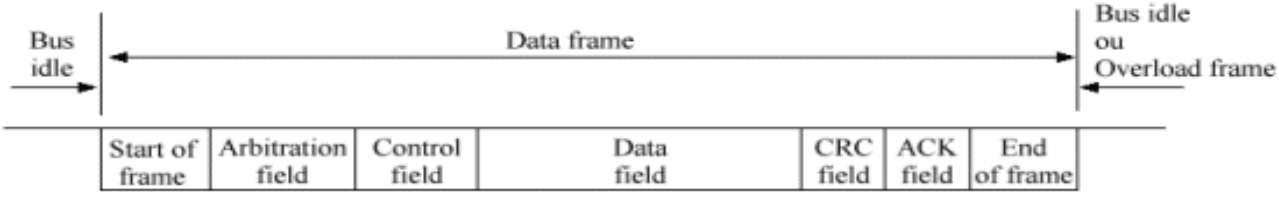


FIGURE 1 : LA TRAME CAN

Le début de trame ou SOF (Start Of Frame),

Le champ d'arbitrage (identificateur),

Le champ de contrôle,

Le champ de données composé de 0 à 8 octets,

Le champ de CRC,

Le champ d'acquittement,

La fin de trame ou EOF (End of Frame).

Pour un bus CAN « low-speed », le nombre de nœuds est limité à 20 alors que pour un bus CAN « high-speed », il est limité à 30.

1.c Stratégie de fonctionnement et robustesse.

Le bus CAN se base sur une transmission de données sur paire torsadée qui permet une interprétation différentielle du signal. Le protocole est par conséquent très robuste aux changements de milieux électromagnétiques.

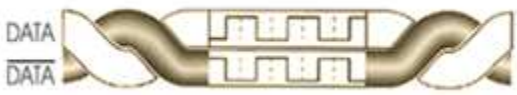


FIGURE 2 : LA PAIRE DIFFERENTIELLE

En addition de cette robustesse reconnue par les acteurs automobiles, le bus CAN profite d'une fiabilité exceptionnelle quant à l'interprétation des données. En effet, la gestion des priorités s'appuie sur la dominance des bits à 0. La trame CAN possédant un identifiant faible, sera prioritaire devant une trame à identifiant plus élevé. Il n'y ainsi aucune relation maître-esclave entre les différents nœuds.

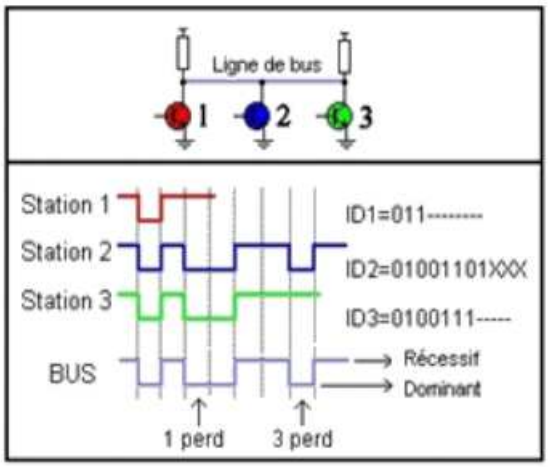


FIGURE 3 : LA GESTION DES PRIORITES

L'interprétation des erreurs est différente selon la typologie de l'erreur, erreur de réception ou de transmission.

Seul un nombre défini d'erreur de transmission peut amener au bus off, c'est-à-dire la cessation de toute activité des drivers CAN du calculateur sur le bus. Cette stratégie rend donc possible l'exclusion d'un nœud du réseau lorsque celui-ci transmet un nombre important d'informations erronées pouvant conduire à une instabilité de la communication inter-organes. Bien entendu, les erreurs de réception ne conduiront pas à un bus off des organes écoutant ces informations, le bus CAN autorisant aux nœuds une écoute de tous les organes, une coupure de tout le réseau serait alors inévitable.

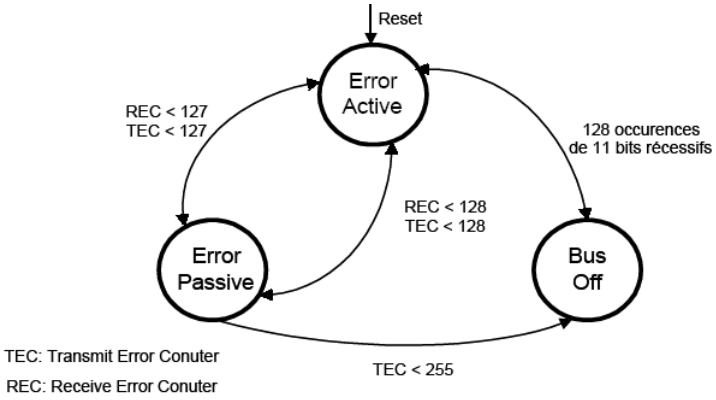


FIGURE 4 : LA GESTION DES ERREURS

1. d Niveaux de tensions

La largeur en débit du canal de transmission n'est pas l'unique différence entre le CAN High speed et le CAN Low speed.

La stratégie des niveaux de tension est complètement différente entre les deux protocoles, un 0 reste toujours dominant mais les niveaux de tension se superposent et un 1 voit une différence de tension conséquente entre le CAN-H et le CAN_L. La validation des couches basses du protocole doit tenir compte de ces différences.

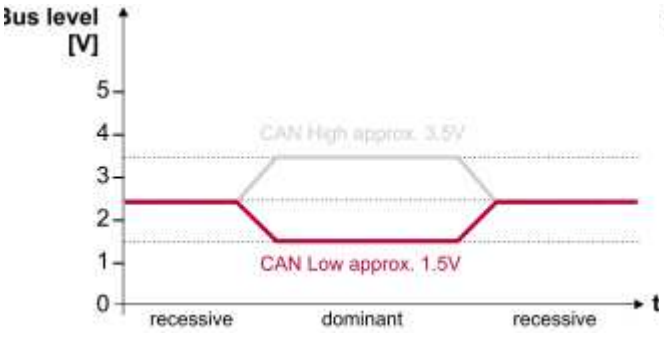


FIGURE 5 : LA COUCHE BASSE HIGH SPEED

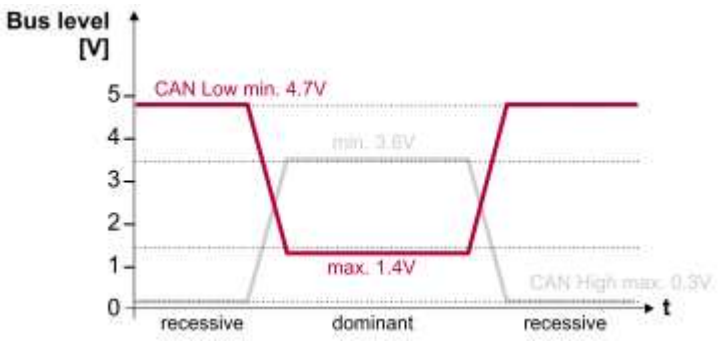


FIGURE 6 : LA COUCHE BASSE LOW SPEED

2. Les systèmes embarqués

De par sa définition[1] un système embarqué est décrit comme :

« Hardware and software which forms a component of some larger system and which is expected to function without human intervention. An embedded system may include some kind of operating system but often it will be simple enough to be written as a single program. It will not usually have any of the normal peripherals such as a keyboard, monitor, serial connections, mass storage, etc. or any kind of user interface software unless these are required by the overall system of which it is a part. Often it must provide real-time response. »

« Un ensemble matériel et logiciel partie intégrante d'un plus large système et qui peut fonctionner de manière autonome. Un système embarqué peut inclure des types d'Operating System mais est le plus souvent l'addition de quelques lignes de programmes. Sauf si ceux-ci sont requis par l'ensemble dans lequel il fait partie, le système embarqué n'a pas de clavier, d'écran, de connections séries, de système de stockage ou d'interface utilisateur. Il fournit la plupart du temps une réponse « temps-réel ».

Les systèmes embarqués sont aujourd'hui présents dans de nombreuses technologies. Nous les utilisons au quotidien, ils sont présents dans nos consoles de jeux, nos téléphones, nos box ADSL, les bornes d'achats de billets… Les technologies disponibles pour ces systèmes ont beaucoup évolué ces dernières années. Malgré les idées reçues, nous ne sommes pas limités aux langages historiques comme le C pour le développement de logiciels embarqués. Il existe une multitude de langage comme le C# ou le .NET qui sont capables de décrire l'applicatif d'un calculateur. La précision sera d'avantage liée au matériel et à l'optimisation de la programmation qu'au langage utilisé.

En dehors des contraintes techniques de base, il n'y a donc pas un langage ou une technologie capable de faire quelque chose qu'une autre ne pourrait pas faire, le choix du matériel résidera dans la vocation du calculateur.

Pour les petits systèmes autonomes, devant tenir dans la paume de la main, l'orientation se fera plus naturellement vers des processeurs à faible cadence (quelques dizaines de MHz), sans systèmes d'exploitation, les programmes principalement du langage C sont implémentés dans le « *firmware* ». En revanche, pour des systèmes sécuritaires critiques, un système « temps réel » sera nécessaire, et enfin pour des bornes de retraits dont les limites en taille et en consommation ne sont pas prépondérantes, des systèmes plus lourds comme Windows sont d'avantage employés.

Les dernières technologies citées définissent principalement l'applicatif du calculateur, le comportement fonctionnel. L'applicatif décrit le rôle du calculateur à travers son utilité propre, afficher des informations tarifaires pour une borne de retrait, déclencher un Airbag ou allumer les feux de croisement pour un ECU véhicule. Ce dernier exemple est typique des systèmes d'aujourd'hui où l'applicatif, est étroitement dépendant de l'environnement du

calculateur le contenant. En effet, dans le cas d'un ECU véhicule type BSI (*Boîtier de Servitude Intelligent*), l'allumage des feux de croisement se fera soit sur action utilisateur, soit dans la plupart des cas aujourd'hui sur une information provenant d'un capteur, ou tout simplement d'un autre calculateur.

Il faudra donc que l'applicatif qui gère ses variables et signaux internes comme la valeur d'avance de phase pour l'injection moteur, allure du PWM gérant le balayage des essuies glaces, ou encore la vitesse de rotation d'une des 4 roues véhicules pour l'ESP le fasse en fonction de paramètres totalement externes. Ainsi, respectivement, il devra prendre en compte un paramètre directement lisible par ses entrées analogiques comme l'enfoncement de la pédale d'accélérateur ou par le réseau CAN comme le signal d'activation de la climatisation. Par ailleurs le balayage de l'essuie-glace se fera en fonction des informations reçues par le signal CAN du capteur de luminosité qui gère aussi la pluviométrie sur le pare-brise, le bloc ESP devra encore prendre en compte l'angle au volant déterminant ainsi sur les directions électroniques l'anti dérapage du véhicule.

Par ce type de réseau, certains ECU deviennent ainsi totalement liés au CAN, sans capteurs externes ni entrées/sorties analogiques ou numériques, l'applicatif du calculateur repose intégralement sur le réseau du véhicule. Ce type de calculateur, souvent au centre d'un réseau dont il est la passerelle pour de nombreuses fonctions devient alors système dans un ensemble embarqué.

3. Les besoins d'analyse et de simulation des réseaux

Le calculateur CAN est un système embarqué et il fait partie d'un ensemble de calculateurs eux aussi composants un système embarqué. En s'appuyant sur les exemples précédemment cités, il est clair que la communication prend de plus en plus le pas sur les interfaces physiques, que de plus en plus d'informations provenant des capteurs véhicules sont très rapidement traduits sur le réseau CAN afin, entre autre, de ne pas prolonger les câblages jusqu'à l'ECU.

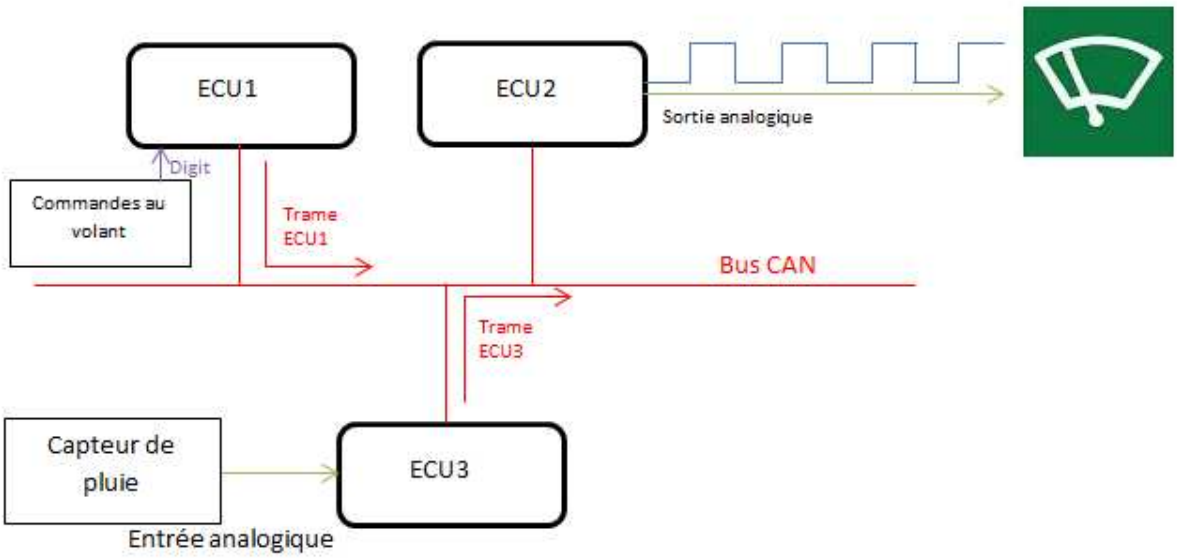


FIGURE 7 : LES INTERFACES D'UN RESEAU CAN

La place de notre projet se situe tout à fait dans ce domaine. L'écriture de l'applicatif, ainsi que la construction du calculateur sont des sciences auxquelles des cœurs de métiers répondent en amont, avant l'implantation de l'ECU dans son environnement.

4. CANoe

CANoe est un logiciel ouvert pour le développement, le test et l'analyse des ECU ou des réseaux complets d'ECU. Il permet de faciliter le travail de l'ingénieur tout au long du processus de développement, en effet ses fonctions polyvalentes et les options de configuration sont utilisées dans le monde entier par les équipementiers et les fournisseurs.

La conception ouverte permet à CANoe d'être le premier choix pour le développement d'ECU pour moteurs à combustion et des projets liés à l'électrification du groupe motopropulseur.

4.a La simulation

Un modèle de simulation peut être généré manuellement ou automatiquement à partir de la de base de données sous-jacente. Cette simulation du bus et du comportement de communication de tout ou partie du réseau d'ECU est la base de l'analyse ultérieure et les phases de test. Via les packages OEM (*Original Equipment Manufacturer*) spécifiques, la simulation CANoe peut être adaptée aux exigences du constructeur concerné. Pour le développement de fonctions liées à l'applicatif de l'ECU, il est aussi possible d'inclure des modèles MATLAB dans la simulation.

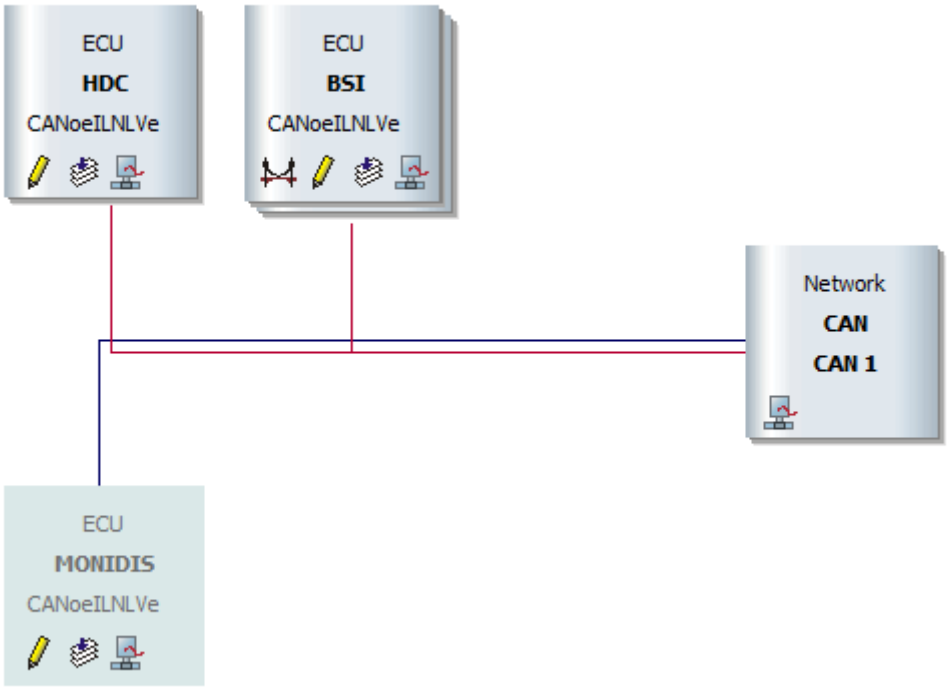
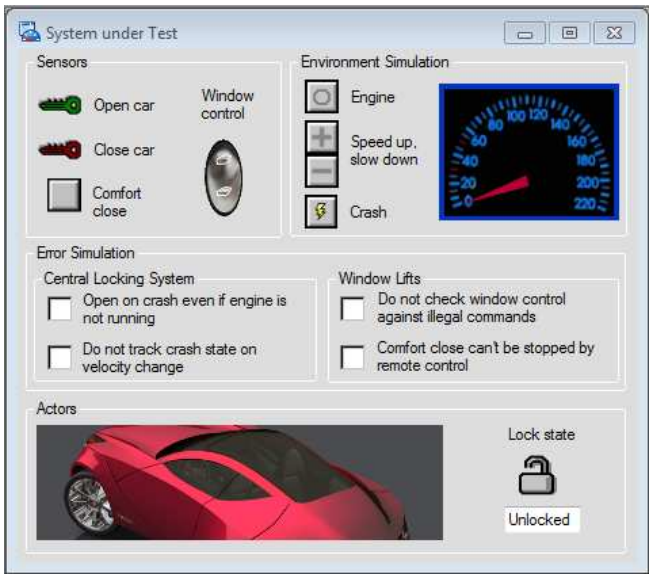


FIGURE 8 : LE SIMULATION SETUP SOUS CANOE

La possibilité de développer des IHM fonctionnelles permet d'accéder en temps réel aux champs de données des trames circulant sur le réseau. Grâce à cette fonctionnalité, les données reçues par le système sous test pourront être traitées et affichées, mais ce sont surtout les données simulées par CANoe et envoyées sur le bus auxquelles les panels peuvent accéder, permettant alors de créer des conditions de stimuli pour l'organe sous tests.



4.b L'analyse

CANoe permet également d'analyser la communication multi-bus des systèmes entiers sur véhicule ou, durant le développement sur les modèles de systèmes éclatés tels que les PIE (*plate-forme d'intégration électronique*). Le point d'entrée principal de l'analyse sur bus existant est la disponibilité du CAN_H et du CAN_L.

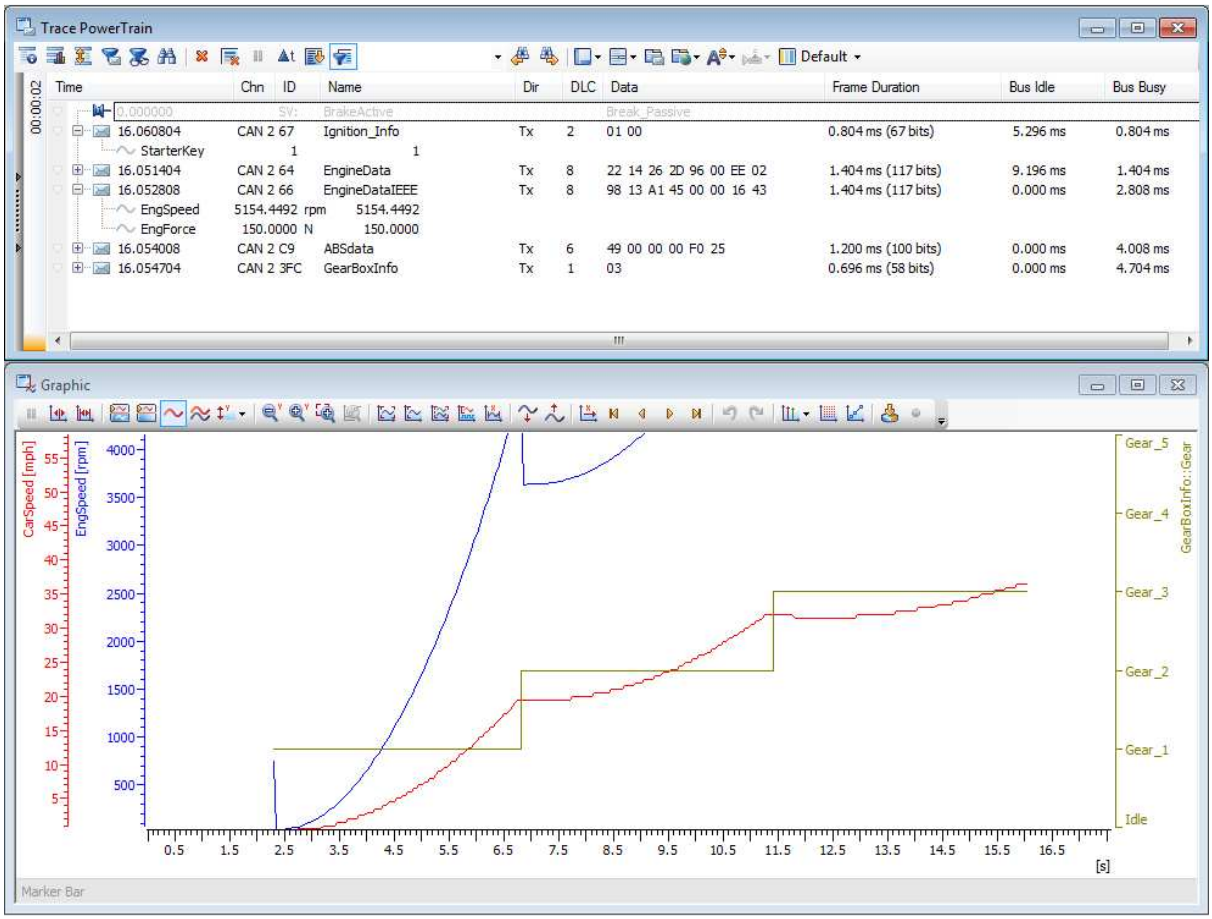


FIGURE 9 : LA FENETRE D'ANALYSE SOUS CANOE

L'analyse du bus repose sur différentes fenêtres paramétrables, donnant accès à un niveau d'observation signal ou trame. Une fenêtre tournée graphique permet de « monitorer » l'évolution de la valeur des signaux dans le temps en synchronisant la base de temps sur d'autres signaux à comparer. Une base de données est indispensable à la traduction du flux de données hexadécimales entrant dans le système. Sans base de données, le décodage des données entrantes doit se faire manuellement.

4.c Les tests

CANoe est aussi un outil de test pour l'ECU et le réseau à développer. Un environnement de test est disponible afin d'ajouter des séquences de tests sous forme de cas de test (*test cases*). Ces séquences de tests valident ou non des événements décrits suite à des stimuli programmés en début de séquence.

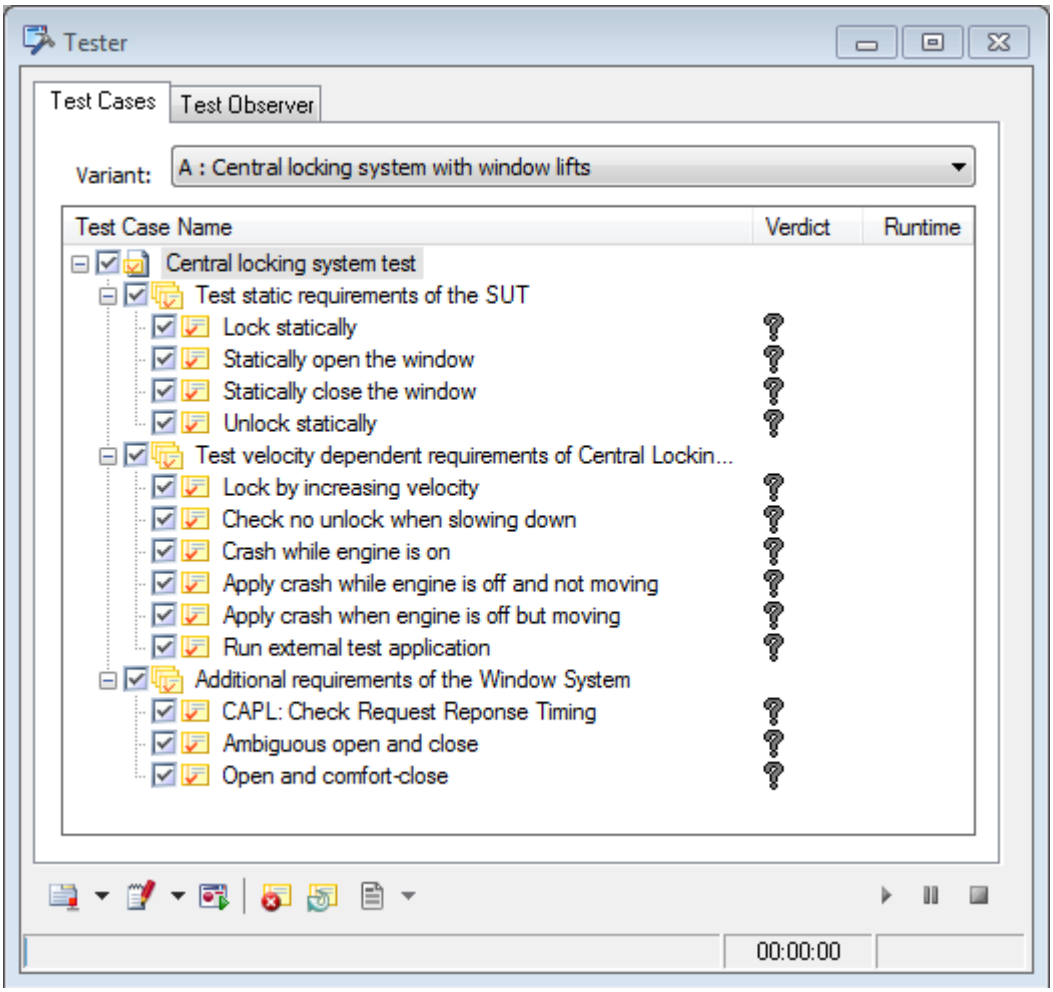


FIGURE 10 : LA FENETRE DE TEST SOUS CANOE

Toujours dans l'optique d'être ouvert aux autres plates-formes, CANoe autorise l'écriture des « *test cases* » à travers différents formats comme le .NET, le xml ou le langage de programmation de CANoe, le CAPL.

5. Le VT-System

Le VT-System est un ensemble de modules d'interface pour le test de calculateurs.

Pour tester un ou plusieurs ECU dans leur globalité, il est non seulement nécessaire comme nous l'avons vu précédemment d'interfacer les réseaux de communication tels le CAN, mais il est également nécessaire de relier les interfaces d'E/S.

Ce banc de test intègre tous les composants de circuit nécessaires pour configurer sur une carte l'analyse, une sortie de commande ou la simulation un capteur de luminosité afin de tester les spécifications de l'ECU.

FIGURE 11 : LE BANC VT-SYSTEM

Les modules sont conçus pour les plages de tension généralement utilisés dans le domaine de l'automobile, et ils peuvent également supporter des courants élevés tels que ceux que peuvent délivrer les feux de route et les moteurs.

Durant le développement du calculateur, les phases d'alimentation de l'ECU sont étroitement contrôlées, afin de maîtriser les consommations de l'ECU dans ses différentes phases de fonctionnement. Par ailleurs, les seuils de réveil de l'ECU au regard des fluctuations de tension régulièrement rencontrées dans une automobile sont soumis aux spécifications constructeur.

6. Le « *Transceiver* » CAN

Afin d'interfacer les solutions logicielles précédemment présentées avec le bus CAN de notre réseau, une carte électronique est nécessaire. Le choix de cette interface repose bien entendu sur les particularités du bus spécifiés par le fournisseur. Majoritairement, les « *transceivers »* CAN sont typés en fonction de la vitesse du bus à étudier, par ailleurs, le choix du support informatique dans lequel sera implémenté le transceiver sera déterminant si nous prenons en compte les différentes connexions disponibles sur les ordinateurs actuels. Enfin l'isolation du transceiver au regard des retours de courant est un critère de plus dans notre recherche.

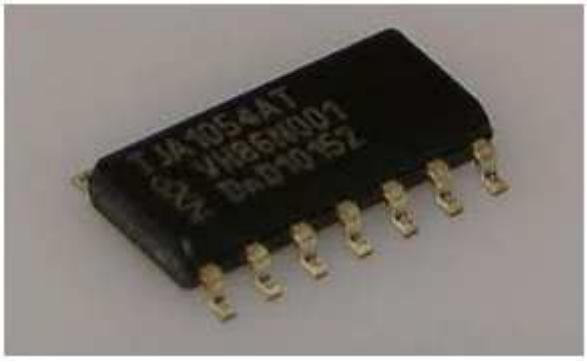


FIGURE 12 : LE TRANSCEIVER PHILIPS TJA1054

Le transceiver sera un Philips TJA1054, qui répond aux critères énoncés, et s'adapte aisément aux cartes d'interface Vector pour l'analyse et la simulation du bus CAN. Les caractéristiques techniques de ce module sont disponibles en Annexe 4. Le transceiver s'occupe uniquement de l'envoi des données via les pins disponibles Tx et Rx pour les données montantes et descendantes. La sortie du transceiver sur le bus CAN sera réalisé via les pins CAN_H et CAN_L.

La gestion des données à envoyer doit se faire par le logiciel en amont, qui cadencera l'envoi des données émises par le transceiver. Deux buffers, devront être disponibles sur la carte mère embarquant le transceiver, afin d'empiler et de dépiler les messages via une méthode FIFO (*First In First Out*) et surtout réaliser la datation des trames CAN.

Cette datation sera sujette à une résolution de l'ordre de la microseconde, alors que la précision de l'affichage des données, qui est aussi lié au périphérique (USB, PCI, etc.) et à la charge CPU du PC peut varier jusqu'à 5ms.

Chapitre III Développement et mise en place du banc

Après avoir étudié les différents outils permettant de mener à bien le projet décrit, la mise en place du banc se reposera principalement sur ces solutions. De nos jours, l'ingénieur est de plus en plus amené à assembler des solutions techniques dans le but d'établir un environnement complet.

1. Messagerie CAN

La messagerie CAN d'un véhicule est un préambule à tout projet de développement de réseau embarqué. C'est sur sa définition que reposera la définition du langage calculateur. Ce « dictionnaire » permet de segmenter et de hiérarchiser les paquets de données binaires envoyées sur le canal de transmission qu'est le bus CAN.

1.a Le dbc

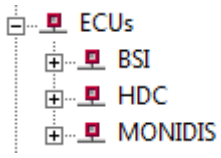
Le format dbc (*DataBase for Communication*) est devenu au fil des années un standard de définition des messageries CAN. Les acteurs de l'industrie automobile utilisent ce format afin de standardiser les définitions de messagerie véhicule entre sous-traitants, équipementiers et constructeurs. En plus d'être supporté par la majorité des outils Vector, le dbc est exploité par les softwares propres aux nombreux autres éditeurs de logiciels et de solutions embarquées. Plus qu'un protocole de traduction du bus CAN, d'un dictionnaire de la communication propre à l'architecture du véhicule, le dbc est aussi le fichier source qui permettra de générer les couches de communications embarquées. Via la définition des attributs de messagerie, le dbc permettra donc d'activer un modèle de communication d'un réseau et d'automatiser l'envoi des trames sur le bus. C'est sur cette fonctionnalité, que CANoe nous permettra de générer notre modèle de simulation.

1.b « *Layout* » et construction des trames

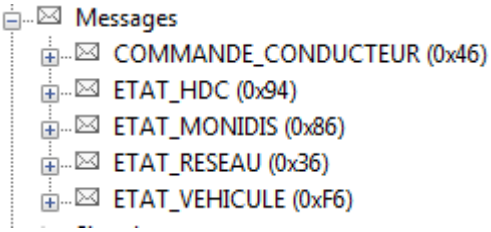
L'écriture de la messagerie se réalisera sur le logiciel CANdb++, qui est un composant de la solution CANoe, la création du réseau véhicule s'appuiera sur les spécifications fournisseur énoncées dans le premier chapitre.

Dans un premier temps, il est nécessaire de définir le nombre de nœuds CAN, en l'occurrence le nombre d'ECU que va contenir notre réseau.

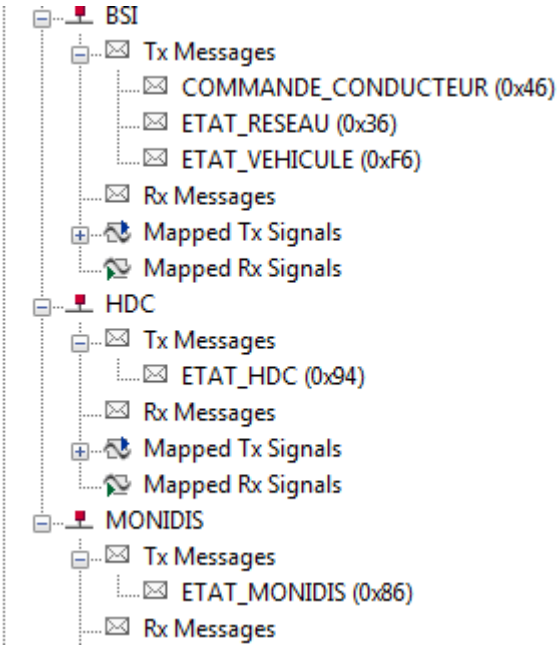
Définition des nœuds :



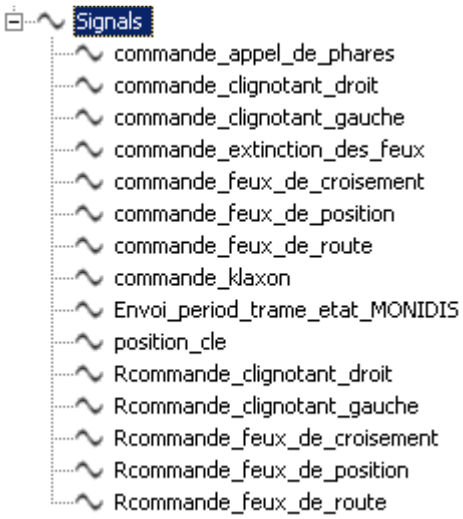
Ensuite, une définition globale des trames véhicules avec leurs propres identifiants :



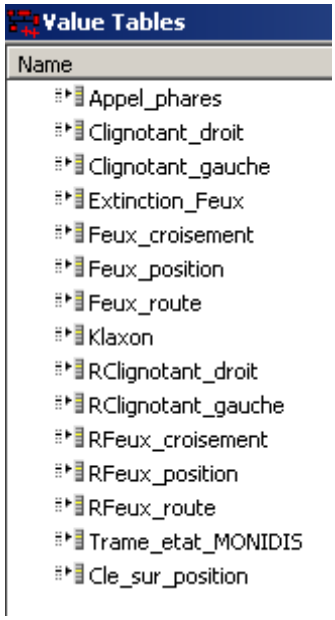
Les trames doivent être « mappées » aux ECU :



Chaque signal devra être crée selon la spécification existante, en terme de longueur bit et d'appellation réseau :

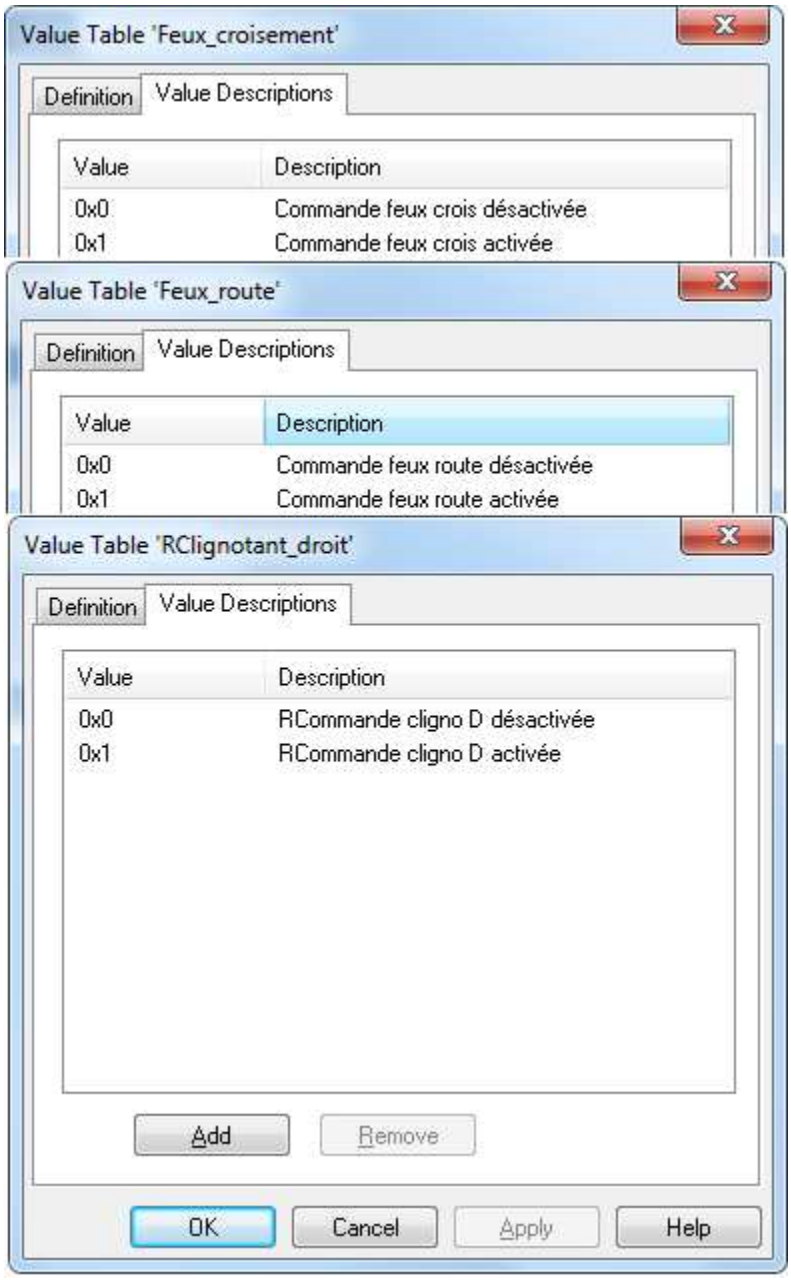


Création des tables de valeurs :

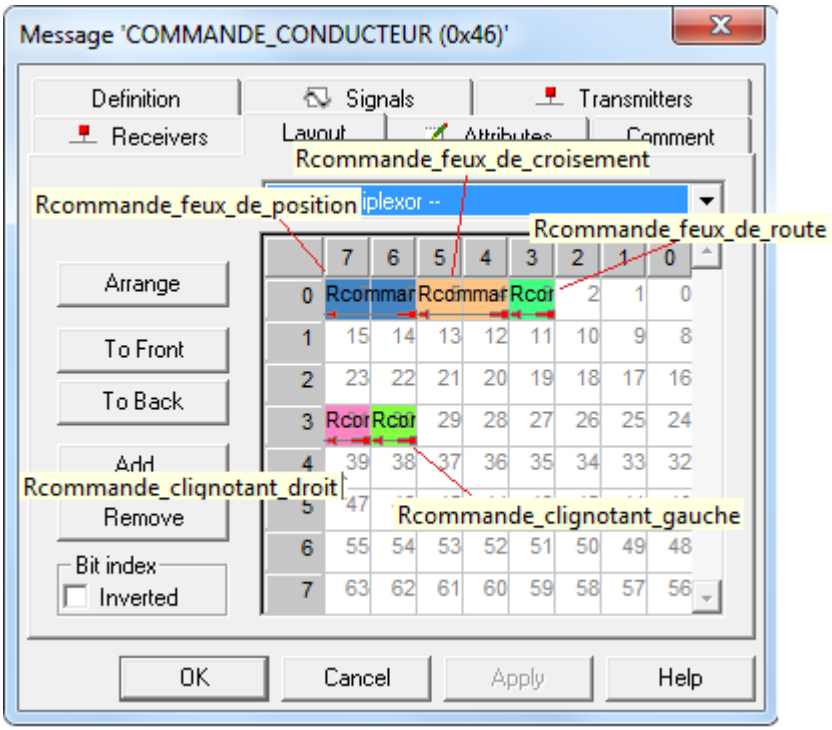


Exemple de table de valeur :

La table de valeur décrit de manière littérale la valeur du signal en fonction de son état hexadécimal. Cette table de valeur est fortement utilisée pour décoder les champs de données afin que l'analyseur de réseau CAN affiche les données directement au testeur.

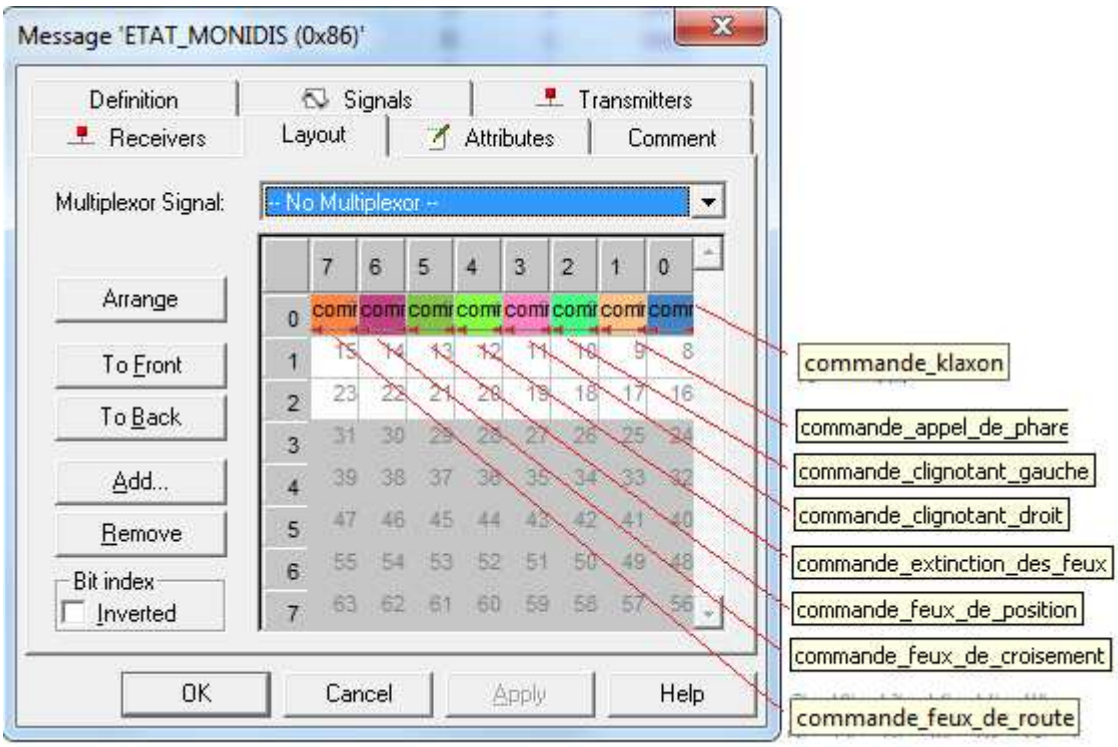


Vérification des « layout (agencements) » par trames :

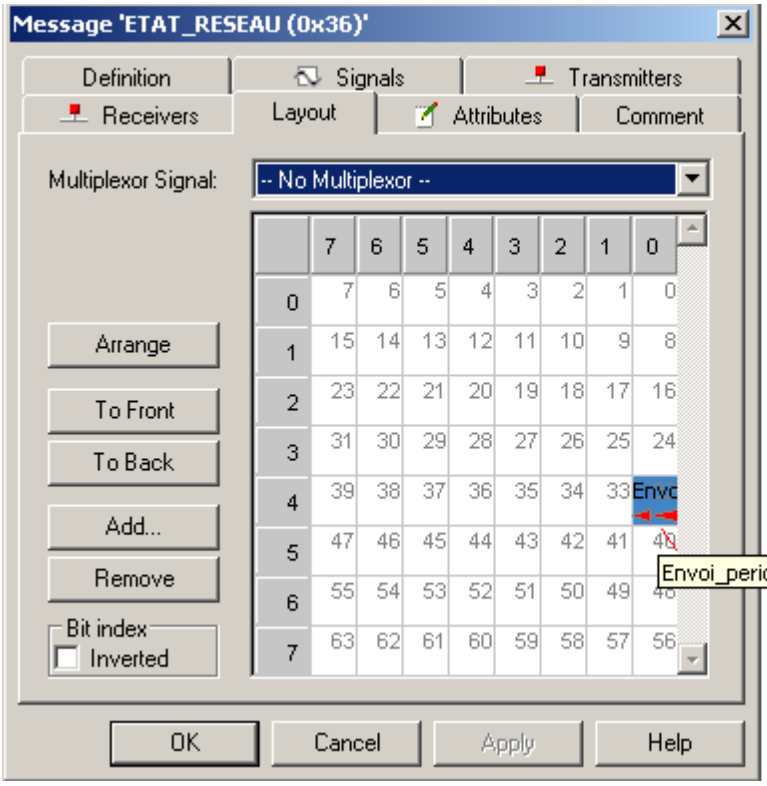


Afin d'être reconnus sur le réseau les signaux doivent être rigoureusement agencés selon la spécification fournisseur. En effet, un signal CAN est carctérisé par la trame qui le transporte, mais aussi par sa position au sein du champ de data de celle-ci. En outre, le signal CAN a une longueur en bits et une position sur un ou plusieurs octets précis du champ de data.

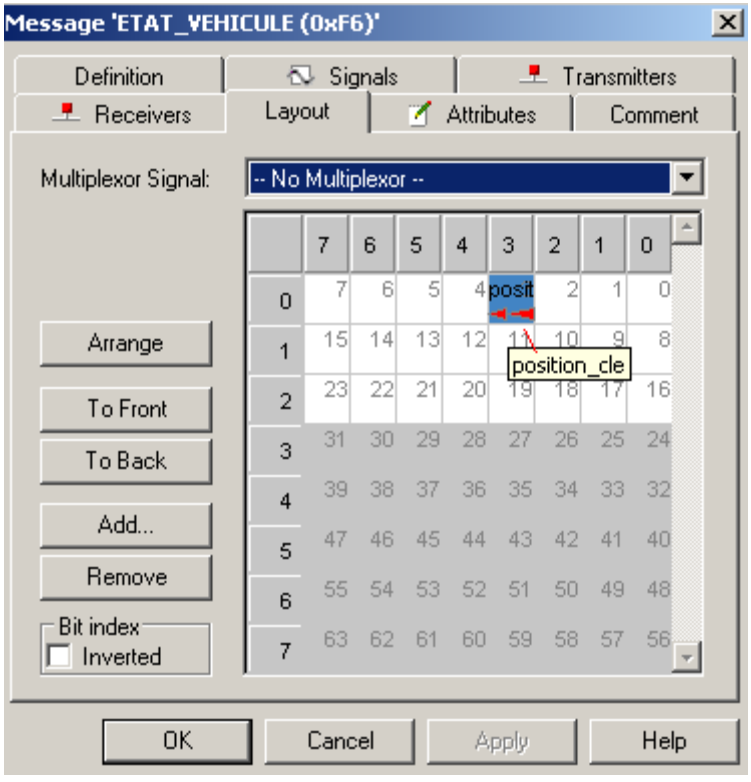
La trame 0x46 possède 5 signaux dont 3 sur l'octet 0 et 2 sur l'octer 3



La trame 0x86 du MONIDIS possède ses 8 signaux binaires sur l'octet 0 du champe de data de la trame, occupant tout l'espace de celui-çi. Les octets restants sont libres en vue de possibles évolutions des spécifications.

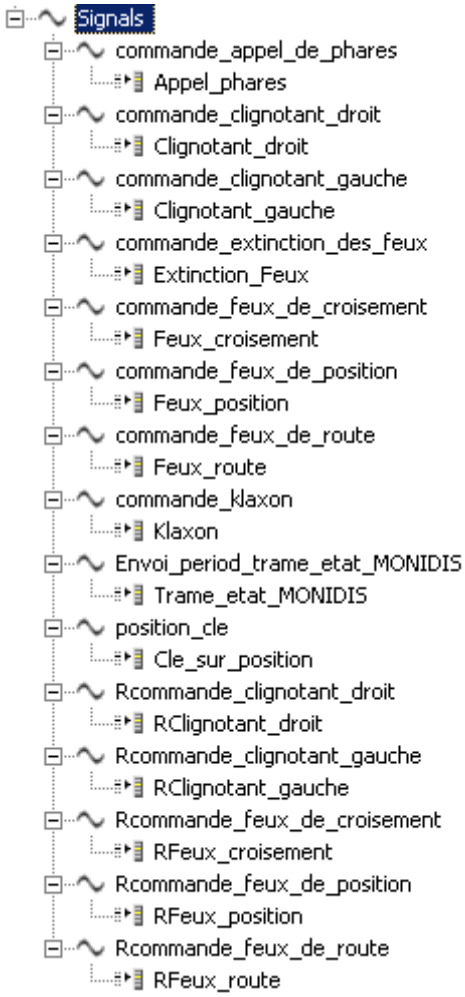


L'unique signal de la trame 0x36 et positionné en début d'octet 4, et de longueur 1 bit.



L'unique signal de la trame 0xF6 et positionné en octet 0, sur le bit 3.

Attribution des tables aux signaux :



1.c Attributs de communication

Les attributs de communication d'une base de données sont le point d'entrée à toute analyse active du bus, orientée signal. Ils regroupent en effet les paramètres de calibration des valeurs de transmission des trames.

Ces attributs définiront donc la qualité de la trame (cyclique, événementielle, sporadique), sa périodicité primaire, et secondaire si exigée par la spécification, ainsi que la dll (*Dynamic Link Library*) sur laquelle reposera l'Interaction Layer, la couche de communication pour l'envoi des trames. D'autres attributs basiques et liés à CANoe peuvent être importés via les *« templates* (modèles) » de dbc proposés par l'outil.

Configuration des attributs :

Type Of Object	Name	Value Type	Mini...	Maxi...	Default
Network	Baudrate	Integer	0	0	125
Network	BusType	String	-	-	CAN
Message	GenMsgCycleTime	Integer	0	50000	0
Message	GenMsgCycleTimeFast	Integer	0	50000	0
Message	GenMsgDelayTime	Integer	0	1000	0
Message	GenMsgILSupport	Enumeration	-	-	Yes
Message	GenMsgNrOfRepetition	Integer	0	999999	0
Message	GenMsgSendType	Enumeration	-	-	Cyclic
Message	GenMsgStartDelayTime	Integer	0	65535	0
Signal	GenSigInactiveValue	Integer	0	100000	0
Signal	GenSigSendType	Enumeration	-	-	Cyclic
Signal	GenSigStartValue	Float	0	1000...	0
Network	NmBaseAddress	Hex	0x400	0x43F	0x400
Message	NmMessage	Enumeration	-	-	no
Node	NmNode	Enumeration	-	-	no
Node	NmStationAddress	Integer	0	63	0
Node	NodeLayerModules	String	-	-	CANoeILNLVector.dll
Signal	NWM-WakeupAllowed	Enumeration	-	-	<n.a.>

Les attributs les plus importants seront définis comme suit :

Bustype correspond au protocole utilisé

Baudrate définit la vitesse du bus paramétré

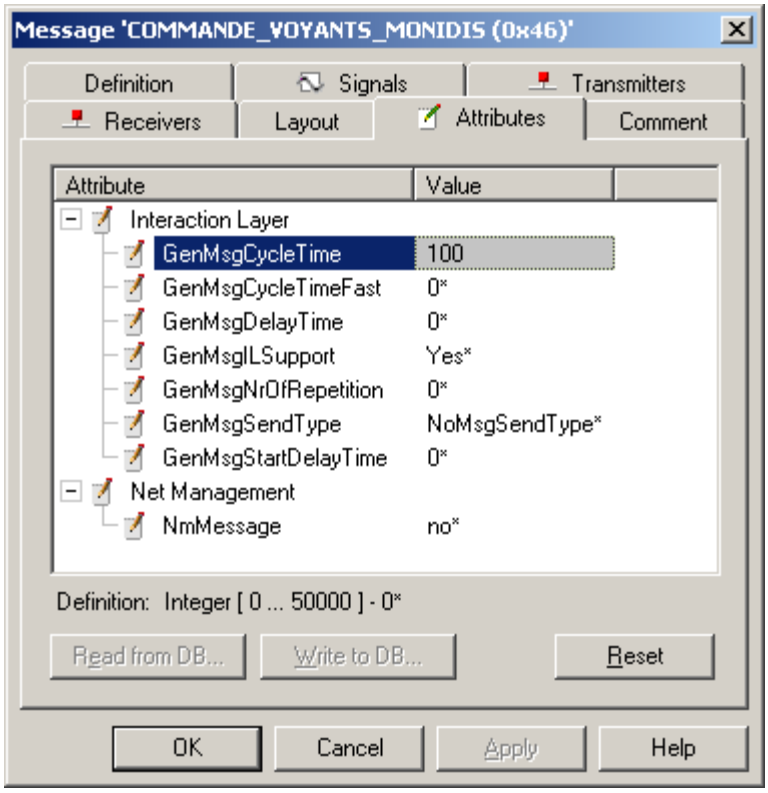
GenMsgCycleTime calibre le temps de cycle de la trame et aura une valeur définie par défaut, mais sera configuré indépendamment pour chacune des trames.

Gen(Msg/Sig)SendType correspond au type d'envoi de la trame/du signal et sera par défaut à « *Cyclic* ».

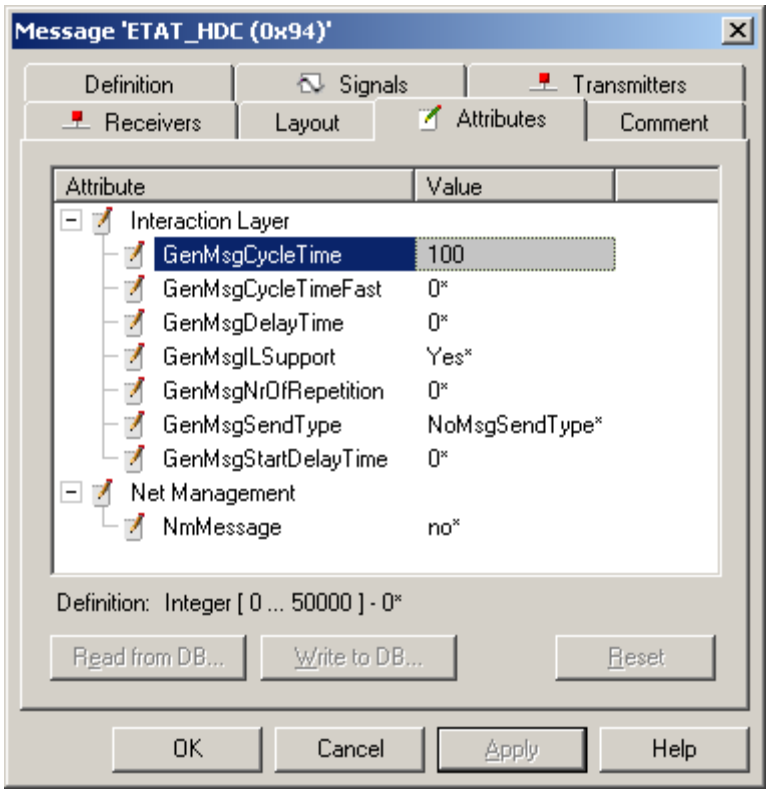
GenMsgILSupport et *NodeLayerModule* font appel à la déclaration de l'IL (*Interaction Layer*) utilisée et la dll correspondante.

GenMsgStartDelayTime permet, dans le cas de réseau chargé, d'ordonnancer l'envoi des trames au démarrrage pour éviter une surcharge du réseau si toutes les trames sont envoyées dans le mêm temps.

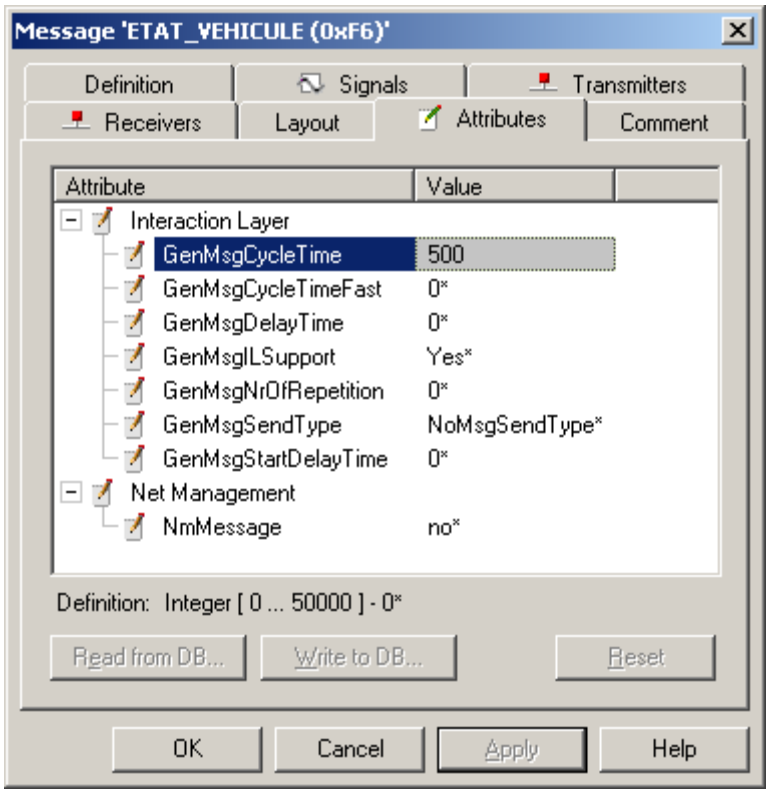
Attributs par trames :



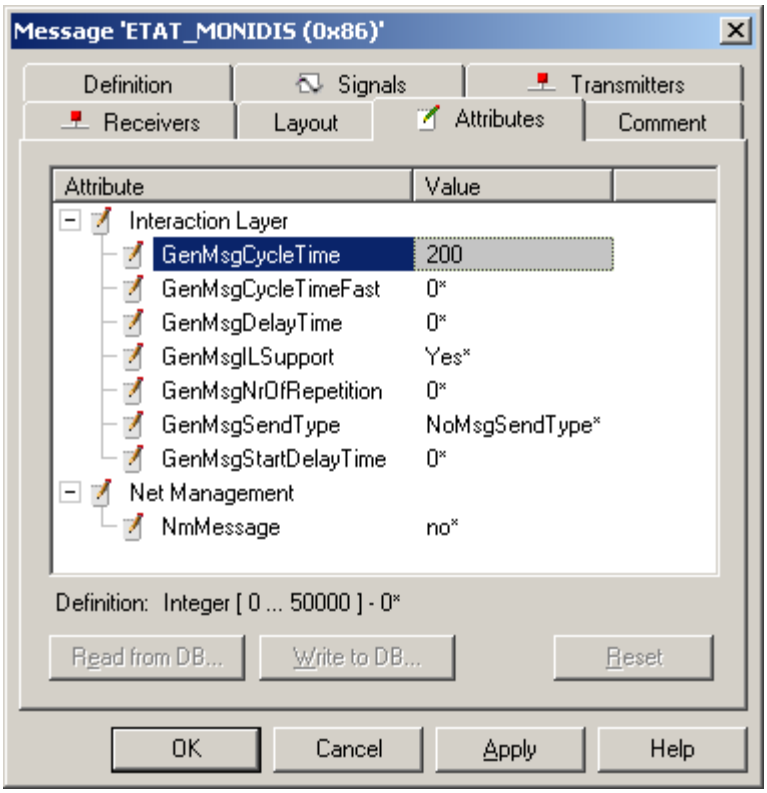
La trame 0x46 possède un temps de cycle de 100ms, n'a pas de délai de démarrage et sera supportée par l'IL.



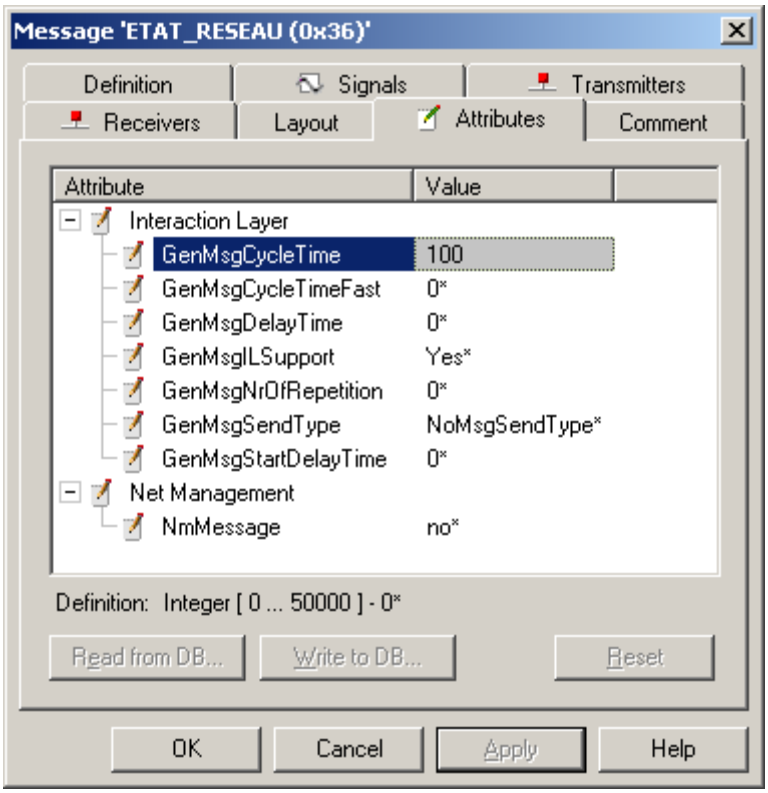
La trame 0x94 possède un temps de cycle de 100ms, n'a pas de délai de démarrage et sera supportée par l'IL.



La trame 0xF6 possède un temps de cycle de 500ms, n'a pas de délai de démarrage et sera supportée par l'IL.



La trame 0x86 possède un temps de cycle de 200ms, n'a pas de délai de démarrage et sera supportée par l'IL.



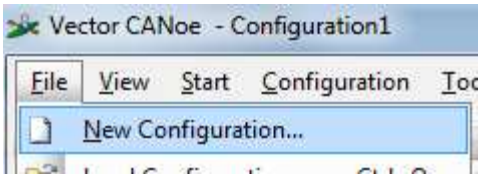
La trame 0x36 possède un temps de cycle de 100ms, n'a pas de délai de démarrage et sera supportée par l'IL.

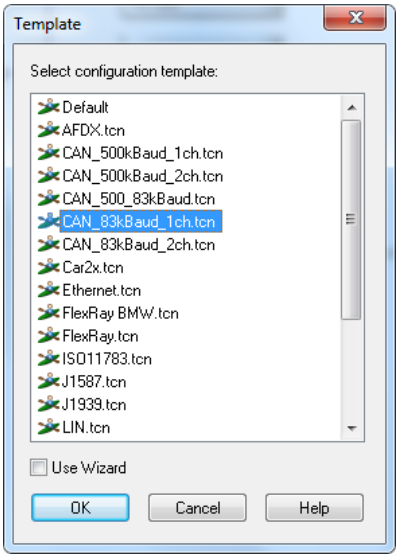
2. Configuration du réseau MONIDIS

En ayant établi notre messagerie CAN, nous pouvons créer le premier modèle simulé de notre projet.

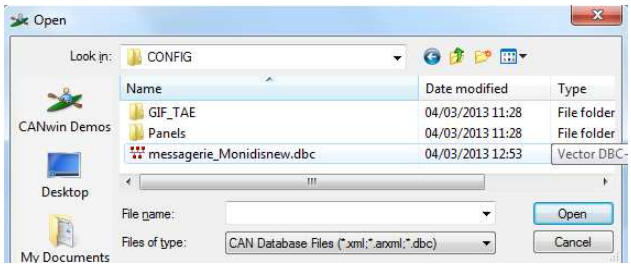
2.a Importation de la messagerie CAN et communication

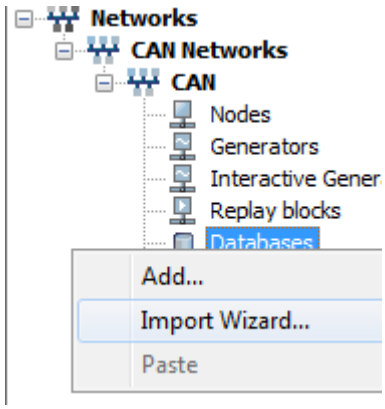
La création d'une nouvelle configuration avec l'outil CANoe débutera par le choix du modèle de notre projet, ici un réseau protocole CAN Low speed.



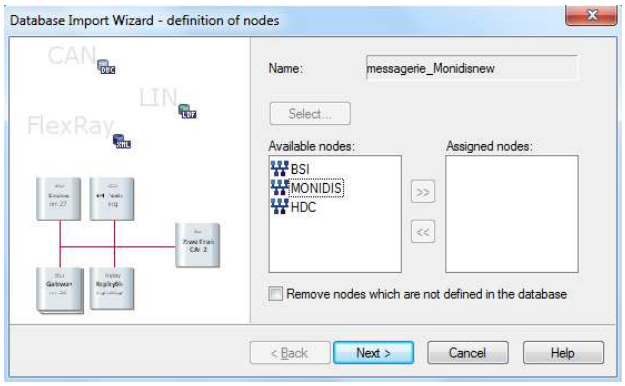


Notre espace de travail sera régulièrement ramené au simulation setup de CANoe. Cette partie du logiciel nous permettra de superviser notre réseau CAN et les accès aux variables simulées. L'importation de la base de données créée précédemment s'exécute par le menu « import wizard » du simulation setup.

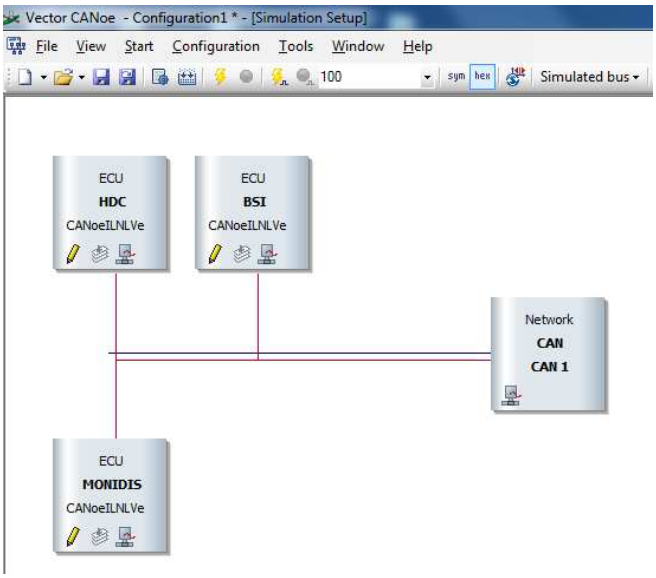




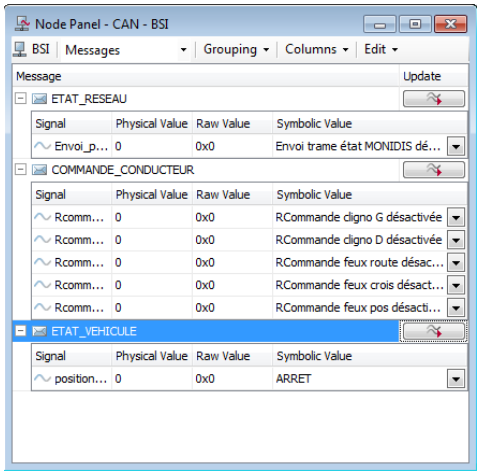
En sélectionnant la base de données de notre réseau MONIDIS, l'assignation des nœuds CAN à notre projet est proposée. Cette fenêtre nous permet de filtrer les informations de la base de données non nécessaires à la simulation de notre ECU à étudier en désactivant les ECU environnants.



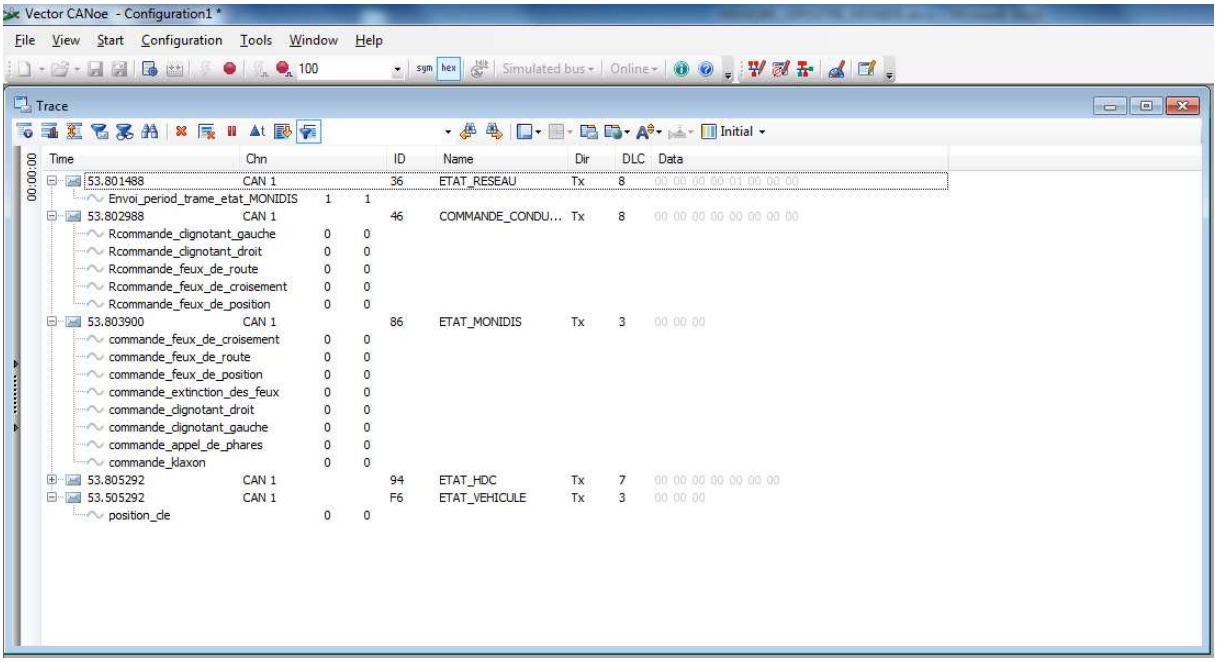
Après avoir validé l'assignation des ECU, le réseau CAN est chargé dans la fenêtre de simulation :



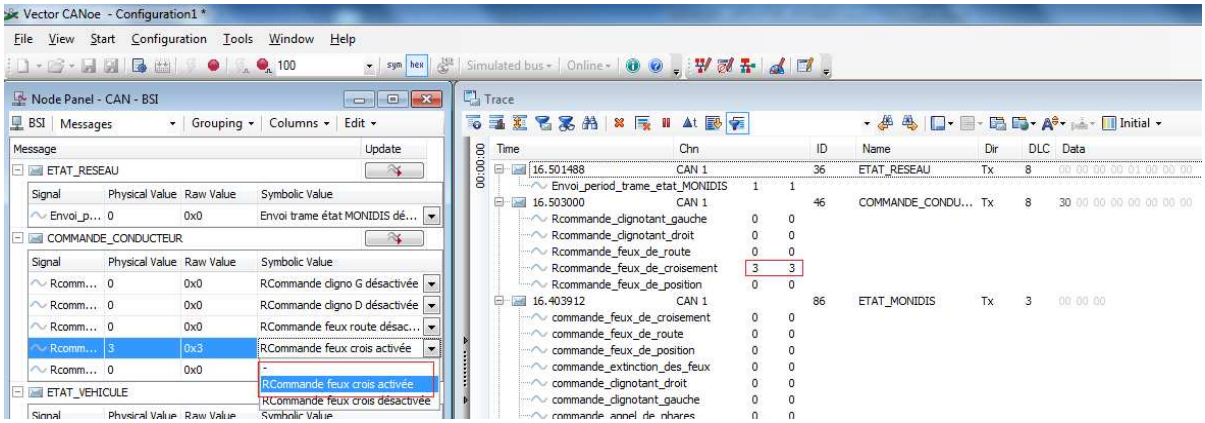
Nous remarquons la déclaration de l'Interaction Layer de CANoe pour chaque ECU. Par ailleurs, il est possible d'accéder aux signaux générés par ces ECU via les panels de chaque bloc. Pour le BSI :



Désormais, nous pouvons lancer la simulation cyclique de notre environnement CAN.



La modification en temps réel des données du réseau CAN est validée par la fenêtre trace :



Cette étape du projet marque un avancement en termes de simulation de la communication du MONIDIS. Le logiciel CANoe ainsi que la dll associée a permis d'automatiser le séquencement d'envoi des trames CAN sur le bus. Cette automatisation s'appuie intégralement sur l'écriture du dbc, le respect des structures de trames et des champs de données déterminera la qualité de la communication lorsque le modèle simulé sera placé en réseau réel.

Par ailleurs, il s'agit désormais de décrire le comportement du calculateur en tant que nœud du réseau crée, c'est-à-dire instancier point par point l'applicatif du calculateur comme celui-ci est décrit par les spécifications fournisseur présentées dans le premier chapitre.

2.b Création du panel et applicatif

Afin de simuler le comportement du calculateur, une interface de programmation est disponible à travers le CAPL (*Can Access Programming Langage*), langage de programmation développé par Vector et qui est devenu un standard de l'industrie de CANoe. Bien qu'il soit possible d'intégrer des dll écrites en Langage C dans l'environnement de CANoe, le langage CAPL offre la souplesse et l'adaptabilité d'un langage événementiel. En effet, le CAPL est tourné événement CAN, les routines ou sous programmes réagissent en fonction des changements de valeurs de signaux sur le bus. Cette particularité offre l'avantage de ne pas dérouler une séquence programmée et figée, mais d'apporter l'interactivité exigée au modèle de l'applicatif du calculateur. Ce type de langage réduit fortement l'utilisation des boucles d'itération type for (i=x; i<y; i++); ou while(z) ; . Ces fonctions pouvant potentiellement bloquer l'exécution du programme alors qu'une interruption, traduit par exemple par un événement CAN, ne pourra être lu par le programme principal.

Pour s'interfacer avec ce programme et fournir un modèle simulé complet pour l'utilisateur, la création d'une IHM, d'un panel est donc requise. Ce panel peut être réalisé grâce au composant logiciel Panel Designer en liant les actions du programme CAPL à des variables systèmes. Ces variables systèmes, sont elles-mêmes raccordées aux boutons d'actionnement de notre panel, permettant ainsi à l'utilisateur, et de manière totalement transparente, de lancer les différentes parties du programme exécutant les fonctions de base du MONIDIS, comme l'allumage des feux de croisement ou des clignotants.

Notre panel s'appuiera sur le graphique fourni par la société Sojadis :

L'annexe 1 reprend la plaquette de présentation du produit qui regroupe les informations relatives aux fonctionnalités principales du Monidis.

Pour chacune des touches de notre panel, une fonction spécifique va être réalisée à l'aide de variables systèmes. Les variables systèmes de CANoe peuvent être créées de différentes natures, string, float ou integer et varient selon les informations auxquelles elles sont reliées lors de leur manipulation dans le programme ou la configuration en général. En effet, ces variables systèmes sont globales à notre configuration bien qu'elles puissent être

utilisées dans le CAPL, en analyse via la fenêtre graphic et trace ou alors en lecture de modules I/O externes au logiciel CANoe.

Le logiciel Panel Designer se présente comme suit :

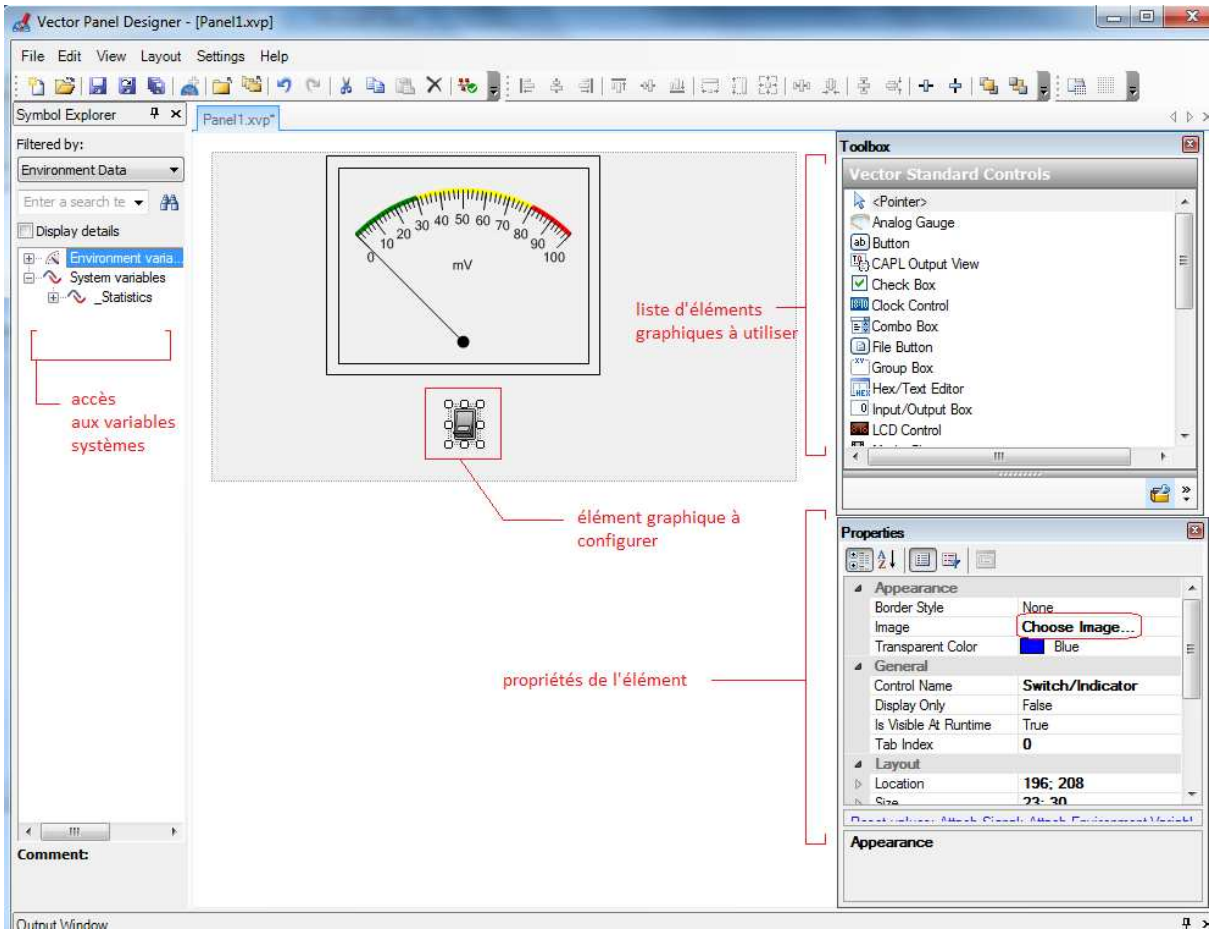


FIGURE 13 : LE LOGICIEL PANEL DESIGNER

La fonction Choose Image… nous permettra de choisir l'image qui correspondra à notre bouton. Il sera alors possible de configurer le graphique des boutons du MONIDIS à partir du bmp fourni par notre client.

Une *sysvar* (variable système) dédiée sera alors liée à chacun de ces boutons afin de déclencher les routines de programme correspondantes.

Il faudra définir les « *switch/indicators* » (élément du Panel Designer pouvant être interfacé avec une variable système) de type « lecture » et ceux de type « control ». En outre, là où les voyants feront office d'affichage des états du MONIDIS, les boutons de contrôle seront l'interface principale de l'homme avec le programme gérant la stratégie du MONIDIS. Ces boutons paramétreront selon leurs appuis des *sysvar* spécifiques listées dans la colonne de gauche ci-dessous. Ces *sysvar* déclenchent les sous-programmes liées à la levée des signaux CAN correspondants. Les *sysvar* en lecture traduiront ces états via les couleurs de voyants préconisées par le fournisseur.

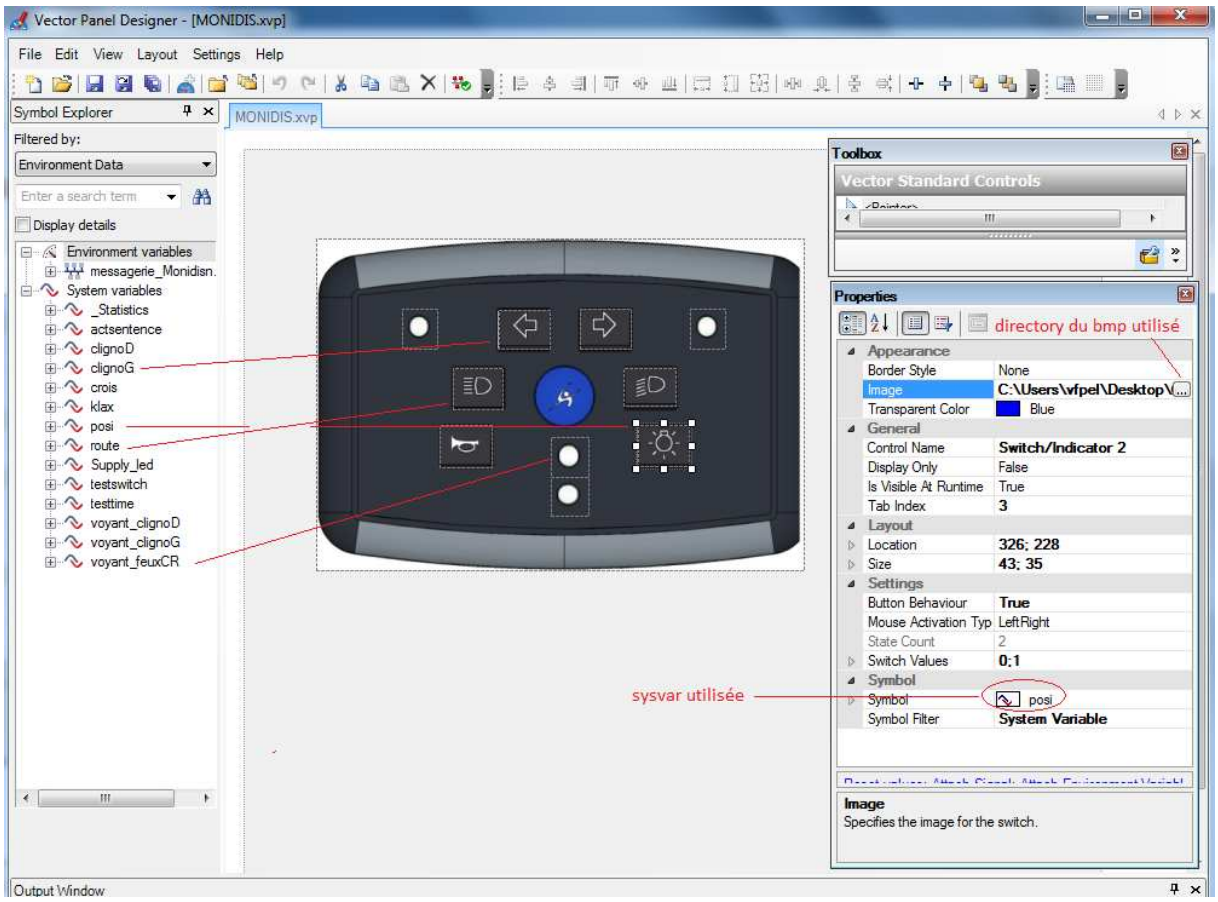


FIGURE 14 : LE DEVELOPPEMENT DE L'IHM

En raison de la lourdeur des programmes liés à cette IHM, nous ne rappelons pas la totalité des sources dans ce chapitre, mais uniquement les exemples les plus probants. L'intégralité des programmes CAPL du panel est consultable en Annexe 2.

Le nœud BSI gérera quant à lui le traitement des signaux de la trame 0x086(ETAT_MONIDIS), afin de mapper la valeur des signaux activés par les boutons aux signaux de commande conducteur de la trame 0x046(COMMANDE_CONDUCTEUR), et de pouvoir dans la deuxième partie du projet, activer de manière transparente les allumages feux véhicules. Le code correspondant à ce traitement se trouve en Annexe 3.

La programmation des boutons du panel repose sur la méthode utilisée dans le sous-programme suivant :

```
on sysvar sysvar::crois::crois      /* rentrée sur la condition bouton utilisé*/
{
 if(@sysvar::crois::crois==1)     /* rentrée sur la condition bouton enfoncé, position 1*/
```

```
  {
    eFuncButtonFeuxCroisement=feuxCroisement;   /* passage de la fonction associée en feuxCroisement*/
    setTimer(tButtonFeuxCroisement,1);              /* paramétrage du Timer de durée d'appui bouton*/
  }

/* SI lorsque le bouton est relaché, position 0, la fonction associée est toujours feuxCroisement, la routine est lancée*/
 if(@sysvar::crois::crois==0 && eFuncButtonFeuxCroisement==feuxCroisement)
  {
    setsignal(commande_extinction_des_feux,0);
    setsignal(commande_feux_de_croisement,1); /* montée du signal CAN feux de croisement à 1*/
    setsignal(commande_feux_de_position,0); /* extinction sécuritaire des autres feux*/
    setsignal(commande_feux_de_route,0);
  }

on timer tButtonFeuxCroisement    /* rentrée sur la routine à la fin du Timer de 1sec*/
{
 if(@sysvar::crois::crois==1)  /* SI le bouton est toujours enfoncé à 1*/
 {
 eFuncButtonFeuxCroisement=extinctionToutFeux;   /* passage de la fonction à extinction*/
 setsignal(commande_extinction_des_feux,1); /* montée du signal CAN correspondant*/
 canceltimer(tButtonFeuxCroisement); /* annulation du Timer*/
}
```

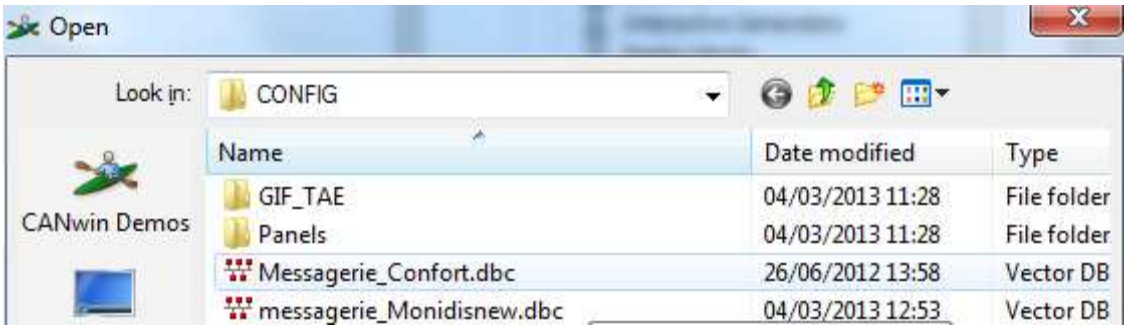
3. Configuration du réseau PSA

Dans le but de tester l'adaptabilité de notre modèle simulé du MONIDIS avec un réseau véhicule réel, il sera nécessaire d'étendre notre réseau CAN actuel via l'ajout d'un nouvel environnement véhicule et d'un calculateur dédié aux fonctions que le MONIDIS doit affecter : le tableau de bord PSA nommé CMB(pour combiné).

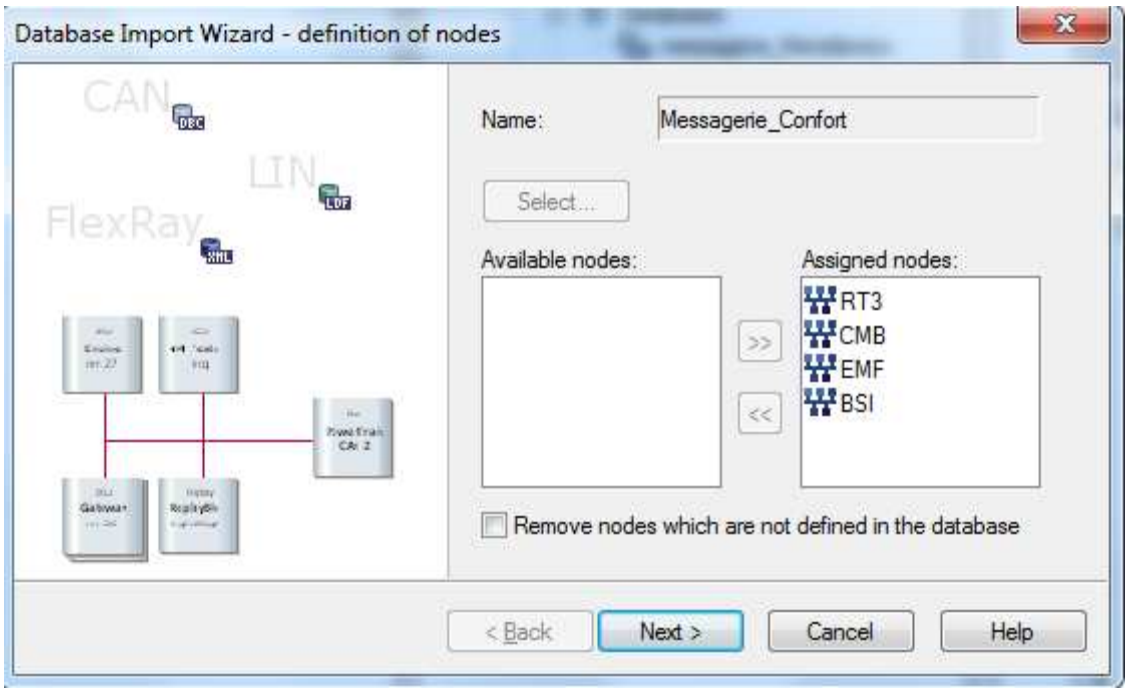
3.a Importation de la messagerie CAN

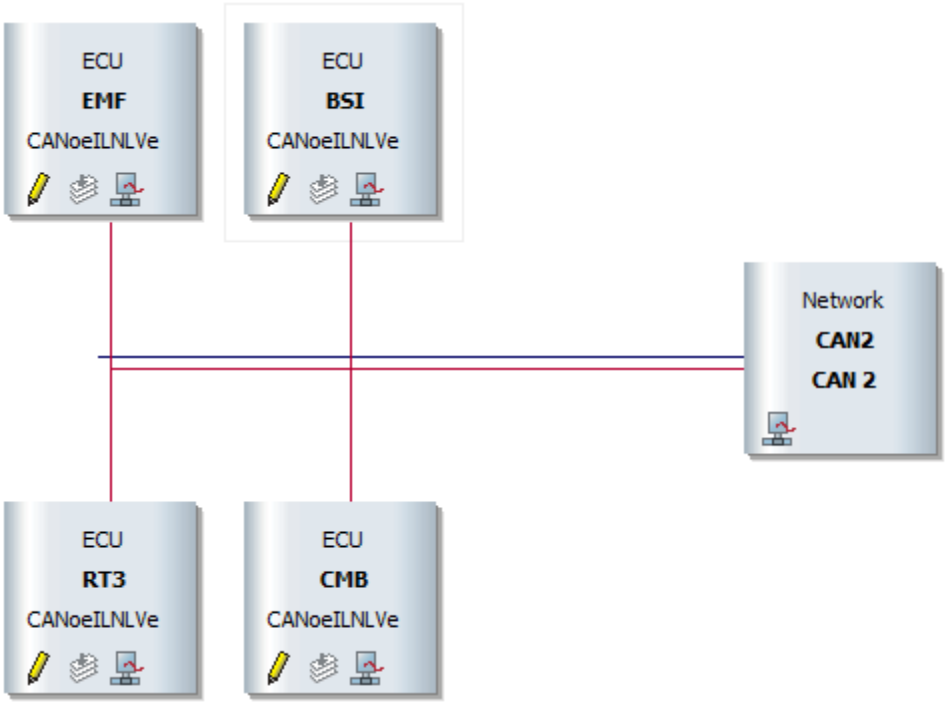
La structure du réseau PSA s'appuie également sur une messagerie CAN format *.dbc. Dans notre cas de projet, notre client Sojadis est équipementier pour les véhicules

PSA entre autre, et peut avoir accès aux messageries de leur client. Cette base de données représente le réseau Confort du véhicule, il est également cadencé à une vitesse bus de 125kbauds et regroupent les fonctions de gestion de l'habitacle du conducteur, parmi lesquelles nous retrouvons l'autoradio, les systèmes de navigation et le tableau de bord.



La méthode pour importer et paramétrer le réseau PSA est la même que lors de l'importation du réseau MONIDIS :

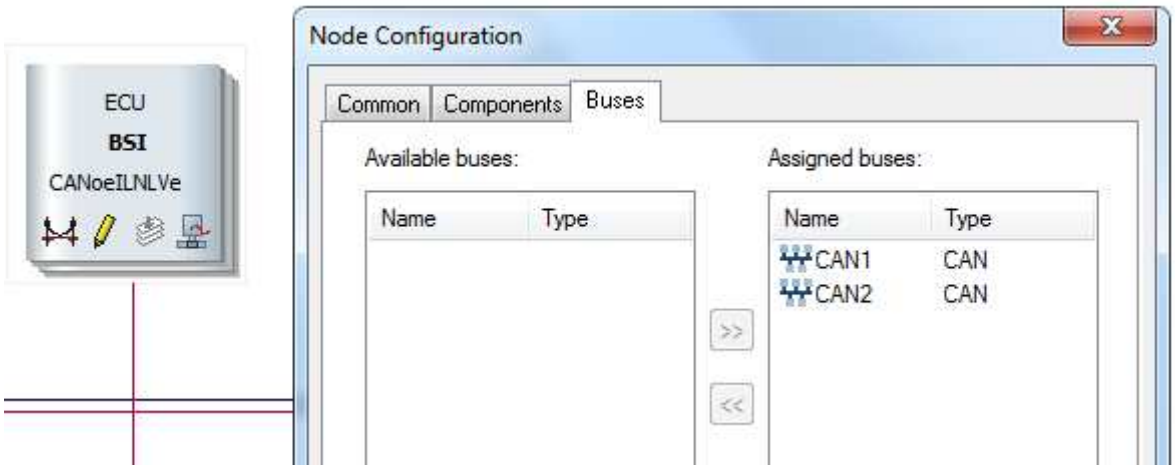




Afin de communiquer avec le réseau PSA, le MONIDIS transmet ses requêtes à la BSI. Le module BSI est essentiellement dédié à la communication inter-réseaux au sein des véhicules PSA, elle est donc une passerelle, ou plus communément nommée gateway dans les documentations techniques constructeur.

3.b Communication et passerelle avec le réseau MONIDIS

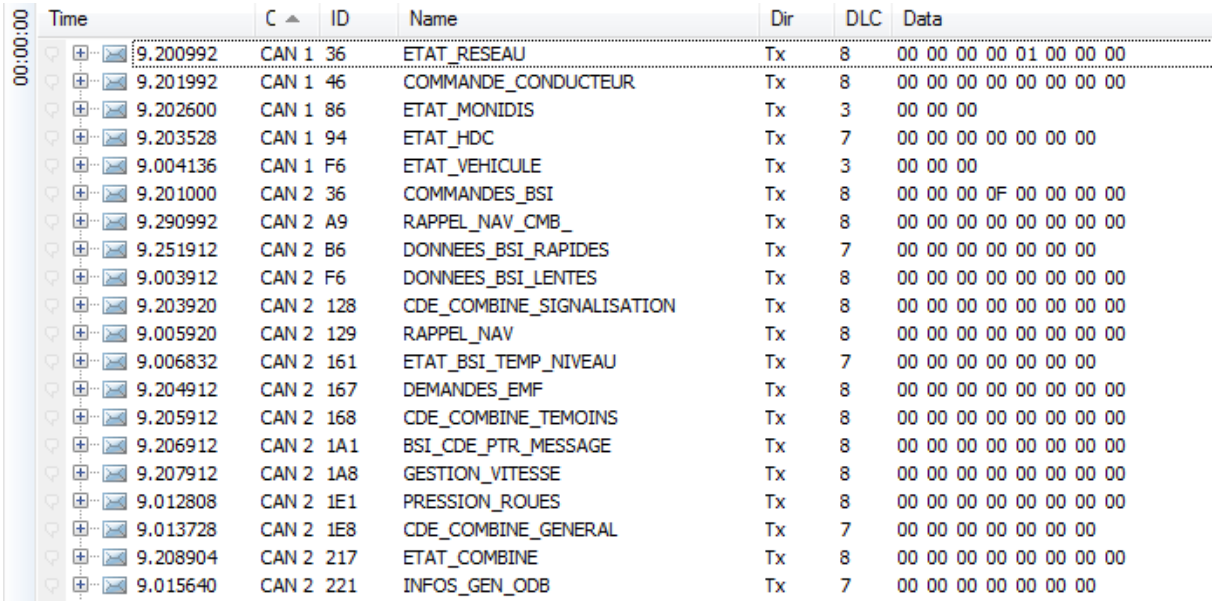
Etant commun aux deux messageries importées, le BSI est paramétrable en tant que passerelle dans les options du nœud simulé, afin de communiquer sur chaque canal les informations respectives de chaque réseau. Le but d'une passerelle est aussi d'avoir accès en lecture aux deux réseaux, et donc de pouvoir mettre à jour les signaux du réseau PSA, en fonction des signaux du réseau MONIDIS :



Le but du MONIDIS est de contrôler l'allumage des feux du véhicule. Cette fonction se réalisera sur le principe de la passerelle BSI, n'ayant pas ce calculateur réel pour notre projet, nous en simulerons l'applicatif qui est disponible en annexe 3.

Cet applicatif reposera sur le schéma fonctionnel déjà codé dans le chapitre dédié au développement du MONIDIS simulé. Le but étant désormais de faire le routage des commandes contenues dans la trame 0x046(*COMMANDE_CONDUCTEUR*) vers la trame de contrôle du BSI consommée par le CMB, 0x128(*CDE_COMBINE_SIGNALISATION*). Cette trame, circulant sur le réseau confort du véhicule gère les voyants du tableau de bord en fonction des informationss principales venant des commandes conducteur, du moteur (sonde de température) ou du châssis (voyant ESP).

Le réseau PSA étant paramétré sur le canal CAN2 du logiciel, il apparaîtra sous cette appellation dans la fenêtre trace du projet :



00:00:00

Time	C	ID	Name	Dir	DLC	Data
9.200992	CAN 1	36	ETAT_RESEAU	Tx	8	00 00 00 00 01 00 00 00
9.201992	CAN 1	46	COMMANDE_CONDUCTEUR	Tx	8	00 00 00 00 00 00 00 00
9.202600	CAN 1	86	ETAT_MONIDIS	Tx	3	00 00 00
9.203528	CAN 1	94	ETAT_HDC	Tx	7	00 00 00 00 00 00 00
9.004136	CAN 1	F6	ETAT_VEHICULE	Tx	3	00 00 00
9.201000	CAN 2	36	COMMANDES_BSI	Tx	8	00 00 00 0F 00 00 00 00
9.290992	CAN 2	A9	RAPPEL_NAV_CMB_	Tx	8	00 00 00 00 00 00 00 00
9.251912	CAN 2	B6	DONNEES_BSI_RAPIDES	Tx	7	00 00 00 00 00 00 00
9.003912	CAN 2	F6	DONNEES_BSI_LENTES	Tx	8	00 00 00 00 00 00 00 00
9.203920	CAN 2	128	CDE_COMBINE_SIGNALISATION	Tx	8	00 00 00 00 00 00 00 00
9.005920	CAN 2	129	RAPPEL_NAV	Tx	8	00 00 00 00 00 00 00 00
9.006832	CAN 2	161	ETAT_BSI_TEMP_NIVEAU	Tx	7	00 00 00 00 00 00 00
9.204912	CAN 2	167	DEMANDES_EMF	Tx	8	00 00 00 00 00 00 00 00
9.205912	CAN 2	168	CDE_COMBINE_TEMOINS	Tx	8	00 00 00 00 00 00 00 00
9.206912	CAN 2	1A1	BSI_CDE_PTR_MESSAGE	Tx	8	00 00 00 00 00 00 00 00
9.207912	CAN 2	1A8	GESTION_VITESSE	Tx	8	00 00 00 00 00 00 00 00
9.012808	CAN 2	1E1	PRESSION_ROUES	Tx	8	00 00 00 00 00 00 00 00
9.013728	CAN 2	1E8	CDE_COMBINE_GENERAL	Tx	7	00 00 00 00 00 00 00
9.208904	CAN 2	217	ETAT_COMBINE	Tx	8	00 00 00 00 00 00 00 00
9.015640	CAN 2	221	INFOS_GEN_ODB	Tx	7	00 00 00 00 00 00 00

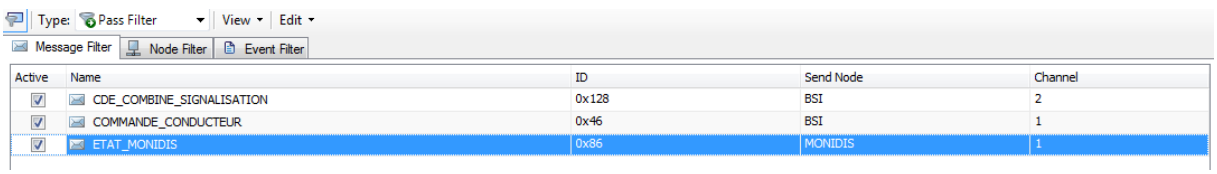
4. Simulation du projet

La partie de simulation de tout projet embarqué est prépondérante dans le développement du réseau réel et complet du véhicule. Les calculateurs étant conçus au sein d'entités différentes, la réunion de l'ensemble des calculateurs constituants un réseau véhicule arrive très tard dans le cycle en V. En général, ces tests se déroulent avant la mise en vie série, sur PIE ou durant les essais sur « mulets », lorsque les réseaux sont intégrés aux environnements mécatroniques qu'ils pilotent. L'intérêt de pouvoir simuler les ECU non disponibles est évident, tant par leurs informations qu'ils communiquent que par leurs applicatifs dont les ECU comme le MONIDIS ou le CMB dépendent.

4.a Vérification des applicatifs programmés

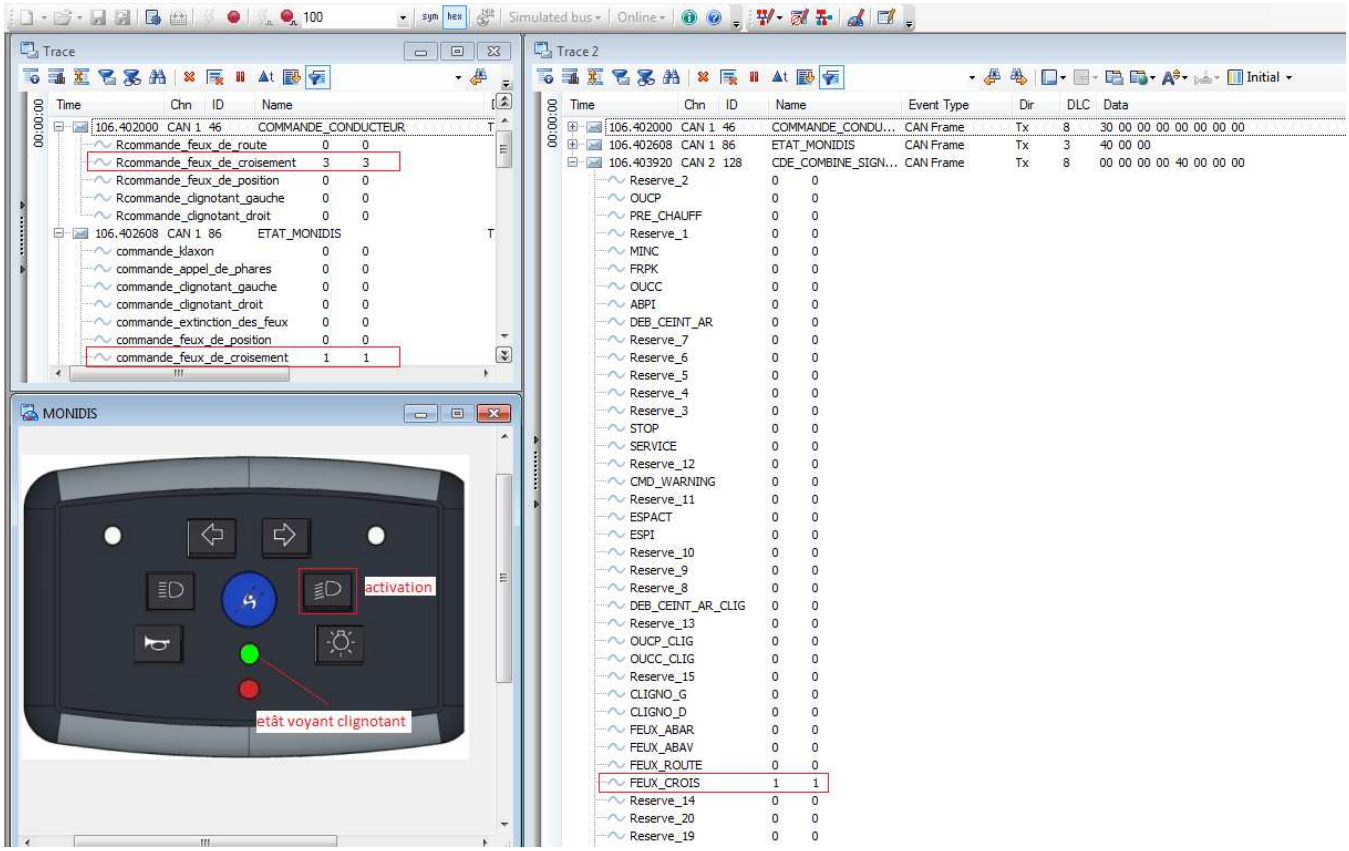
Grâce à la gestion des différentes fenêtres de la plate-forme développée, nous allons pouvoir vérifier le contrôle des interfaces développées sur les signaux CAN en temps réel. Cette vérification sera effectuée de manière plus précise lors de l'élaboration des plans de tests durant la phase de validation du système réel.

Afin de visualiser concrètement la montée des signaux du CMB, un filtre sera placée en amont du bloc trace. Celui-ci ne laissera passer que les trames à visualiser :



Lors de ce test, il est inutile de mesurer les temps de montée des signaux CAN, la vérification n'aurait que peu de sens, étant sous un modèle de bus CAN simulé. La mesure des temps de montée des signaux et des périodicités des trames fera l'objet d'un contrôle lorsqu'un bus physique et réel sera utilisé. En effet, des contraintes de charge bus, des ratés d'acquittement et des possibles trames d'erreur ne sont pas probantes dans un modèle simulé, mais font partie intégrante des plans de test lors de la validation d'un réseau CAN.

A l'aide du panel développé, nous activons les feux de croisement. Le signal 'commande_feux_de_croisement' de la trame 0x086(*ETAT_MONIDIS*) est alors levé à 1. L'applicatif codé du BSI et dédié au CAN1 gère alors la levée du signal Rcommande_feux_de_croisement de la trame 0x046(*COMMANDE_CONDUCTEUR*). La partie d'applicatif du BSI dédié au CAN2 envoie alors l'information d'activation des feux de croisement via le signal FEUX_CROIS de la trame 0X128(*CDE_COMBINE_SIGNALISATION*).



Les mêmes essais sont donc réalisés pour l'ensemble des prestations du MONIDIS, dans le but de vérifier que notre modèle simulé pourra s'adapter de manière transparente au CMB(combiné PSA) PSA lors de l'implantation de celui-ci. Notre modèle simulé pourra alors se compléter aux configurations de différentes architectures véhicule de manière totalement transparente, en mappant les signaux du MONIDIS au tableau de bord du nouveau véhicule. C'est sur ce principe que SOJADIS rend ses produits multi-plate-formes, en adaptant le MONIDIS aux bases de données constructeurs.

4.b Bilan du développement de la simulation

Lors du développement de notre modèle simulé, nous avons vu l'importance évidente de la création de la messagerie CAN. Le format *.dbc nous offre la possibilité d'une part de structurer nos trames et de détailler l'agencement de nos signaux au sein du champ de data de la trame.

Le modèle simulé du calculateur MONIDIS répond aux exigences fonctionnelles de la spécification fournisseur. Les tests de l'ensemble des prestations n'ont pas été détaillés ici mais pourront être validées lorsque le CMB réel sera placé dans le réseau. La validation du MONIDIS réel permettra les tests des entrées du calculateur.

5. Mise en service du calculateur et prise de communication à l'aide d'outil d'analyse du bus CAN

A l'instar des projets de développement des réseaux embarqués, l'intégration du calculateur réel s'aidera majoritairement de l'environnement précédemment développé, afin de l'intégrer dans notre modèle jusqu'alors entièrement simulé. La proportion de temps de projet accordée à la création de notre simulation trouve toute son importance dans la simplicité à mettre en œuvre les tests de notre calculateur, but premier de ce projet.

5.a Intégration du CMB et validation du modèle simulé

Le CMB est un nœud CAN Low-speed alimenté en tension nominale standard 12V.

Pour lancer le mode nominal du CMB, plusieurs signaux CAN provenant des nœuds environnants dont la BSI sont nécessaires.

Le signal PHASE_VIE de la trame 0x36(*COMMANDES_BSI*) rend état de l'activité réseau et de la communication du BSI. Son état doit être à 1.

Le signal ETAT_PRINCIP_SEV de la trame 0xF6(*DONNES_BSI_LENTES*) indique l'état du moteur arrêté, en démarrage, contact ou en fonction. Son état doit être à 1.

Le signal ON_CMB de la trame 0x128(*CDE_COMBINE_SIGNALISATION*) définit l'état actif ou non du Combiné. Le signal doit lui aussi être monté à 1.

Le signal LUMINOSITE de la trame 0x36(*COMMANDES_BSI*) amène une plage de valeur de la luminosité de l'affichage du tableau de bord. Nous fixerons sa valeur à 15.

Le script CAPL permettant cette activation du Combiné est définit dans l'annexe 3 correspondant à l'applicatif BSI.

5.b Intégration du MONIDIS et mise en oeuvre du modèle réel

Le MONIDIS est un nœud CAN Low-speed alimenté en tension nominale standard 12V. Son brochage est décrit ci-dessous ou en Annexe7.

CONNECTEUR 10v	SIGNAL
1	+CAN
2	MASSE
3	/
4	CAN-H (1)
5	CAN-L (1)
6	BUZZER (2)
7	RETROECLAIRAGE (2)
8	/
9	/
10	/

(1) CAN-LS 125kb/s

(2) BUZZER et RETROECLAIRAGE actifs sur +12v

Pour lancer le mode nominal du MONIDIS, le signal CAN 'Envoi périodique de la trame d'état du MONIDIS' de la trame 0x36(ETAT_RESEAU) doit être levé à 1.

Ce signal n'est pas choisi au hasard, en effet, il correspond au signal PHASE_VIE de la trame 0x36(*COMMANDES_BSI*) qui rend état de l'activité réseau et de la communication du BSI. Cette trame est en effet envoyée par la BSI, sur les deux réseaux CAN via la fonction de passerelle de celle-ci. Ainsi, l'adaptation au réseau PSA est réalisée et les spécifications de réveil des organes PSA est respectée par le MONIDIS. Celui-ci ne pourra donc pas communiquer sur le réseau PSA, et ainsi générer de possibles erreurs tant que le bus et les organes ne seront pas en activité.

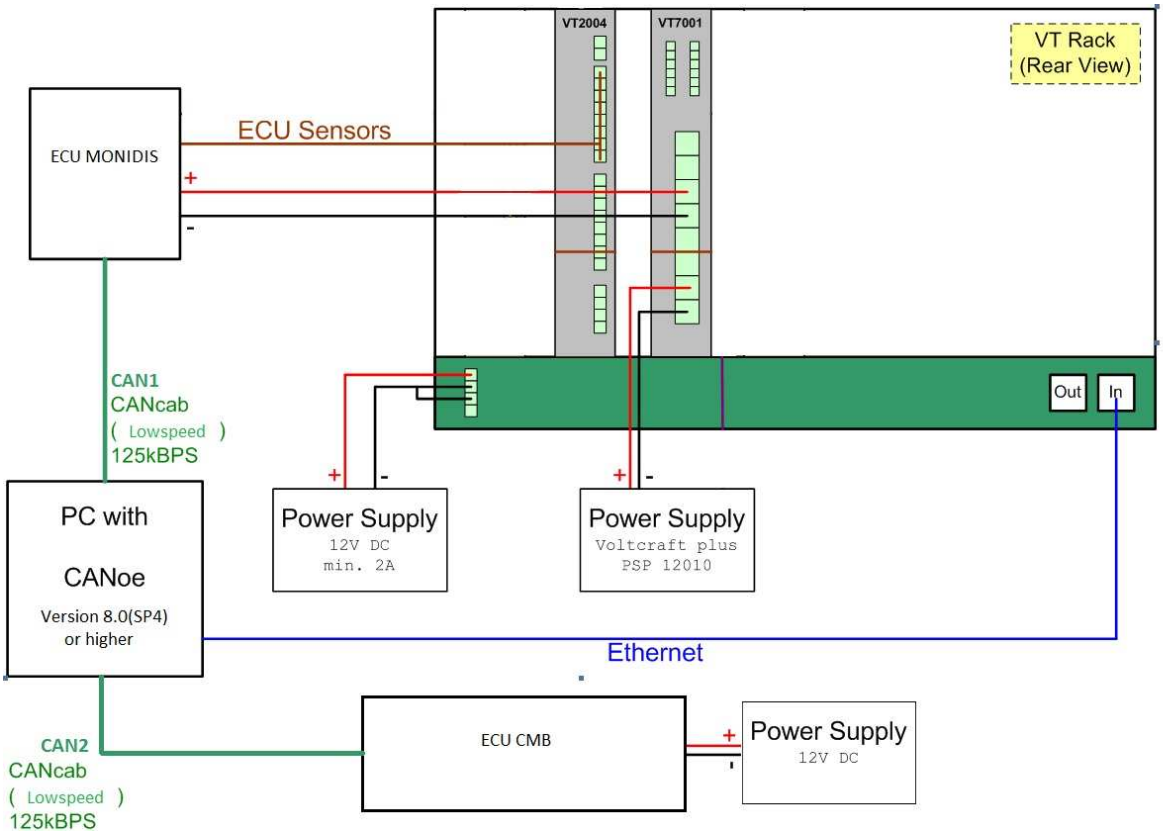


FIGURE 15 : LE SYSTEME COMPLET REEL

Chapitre IV Validation et utilisation

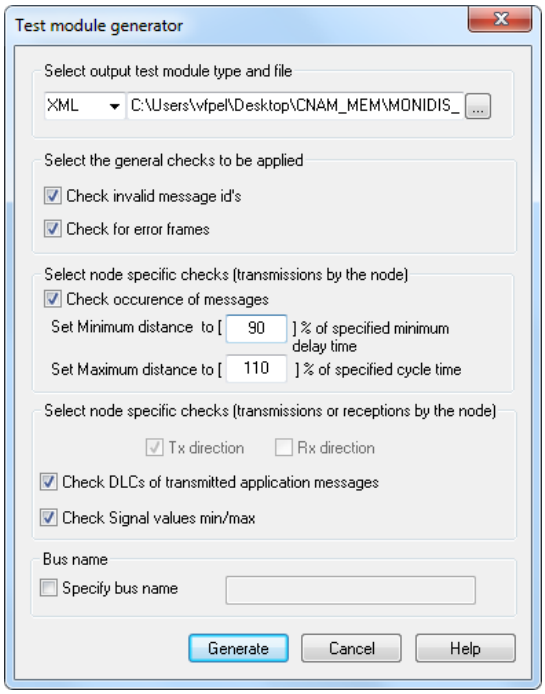
La phase de validation de notre projet représente l'aboutissement de l'intégration réalisée précédemment. La qualité du travail demandé par le fournisseur pourra être démontrée uniquement si les étapes de validation du produit correspondent aux spécifications d'entrée du projet. Il sera nécessaire d'adapter ces étapes de validation pour qu'une équipe de test puisse facilement prendre en main le banc afin de réaliser les tests validant le produit. Ces tests seront segmentés en 3 domaines distincts, les tests de communication du réseau CAN contrôleront l'activité du MONIDIS sur le réseau alors que les tests analogiques valideront les I/O du calculateur et sa gestion d'alimentation. Enfin, les tests comportementaux auront pour but de confondre l'applicatif du calculateur avec les exigences spécifiées, au sein d'un réseau véhicule constructeur.

1. Tests de communication du réseau CAN

1.a Test de communication générale

La phase de test de communication s'exécute sur les spécifications de la base de données *.dbc. En sélectionnant les points de structure du bus à contrôler, comme l'occurrence des messages, le respect des valeurs min/max des signaux, les identifiants inconnus et les « error frames », un fichier *.xml est généré et intégré à notre plan de test sous CANoe. Le test pourra être exécuté en parallèle de la simulation de l'organe afin de contrôler durant toute la phase d'essai du MONIDIS la cohérence de la communication du bus au regard des paramètres définis dans la base de données. Les informations sortant du champ défini seront enregistrées et intégrées au rapport de test sous format html.

Pour le nœud MONIDIS :



1.b Validation des temps de montée des signaux du MONIDIS

Afin de valider le temps de montée des signaux de la trame de commande du MONIDIS, les périodicités des trames sont contrôlées au regard de la tolérance paramétrée dans le générateur du test. Afin d'accepter des écarts minimes cette tolérance a été fixée à 90-110% du temps de cycle déclaré dans la base de données. Si la périodicité relevée entre chaque trame est comprise entre 180ms et 220ms, le comportement de l'organe est validé.

Check/NodeMsgsOccurrence (1#3): Check Passed

Parameters	
Type of check	NodeMsgsOccurrence
Node	Node 'MONIDIS' on bus CAN
MinRel-time	0.9
MaxRel-time	1.1

Statistics	
Runtime of the statistic	2455.777672 ms
Number of samples	12
Message ID	134 (0x86)
Cycle time	200 ms
Delay time	-
Failure ratio (in %)	0

Distribution of measured times		
<= 200 ms	200 ms - 220 ms	> 220 ms
11	0	0

FIGURE 16 : LE RAPPORT DE TEST

Les signaux CAN étant mis à jour via l'envoi de la trame sur le bus CAN, le contrôle de la périodicité est une étape indispensable à la validation de la communication du bus. La qualité de l'applicatif est directement liée au respect du cadencement des trames de signaux.

1.c Injections de trames d'erreur

Le réseau CAN est un réseau robuste et fiable, par ailleurs il est autonome à travers sa capacité à détecter si une erreur de transmission ou de réception doit exclure un nœud du bus. Pour valider ce comportement, nous disposons de la possibilité d'injecter des « *errors frames* » (trames d'erreur) dans le réseau et d'en détecter le nombre et le type grâce à notre test de communication.

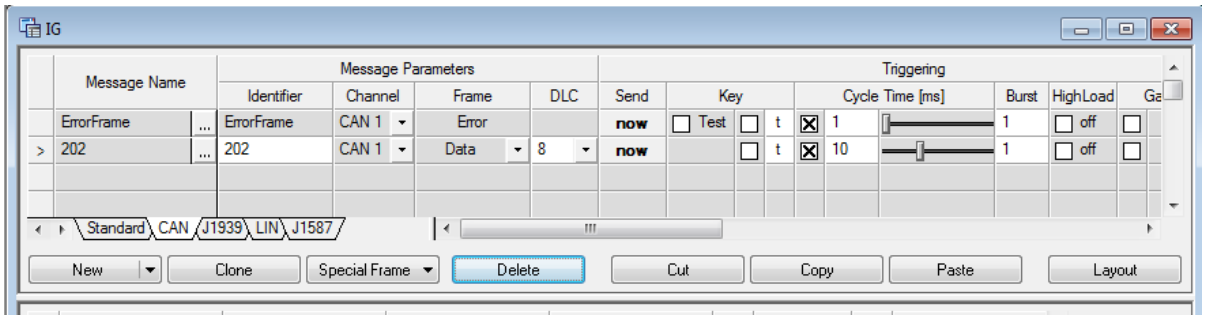
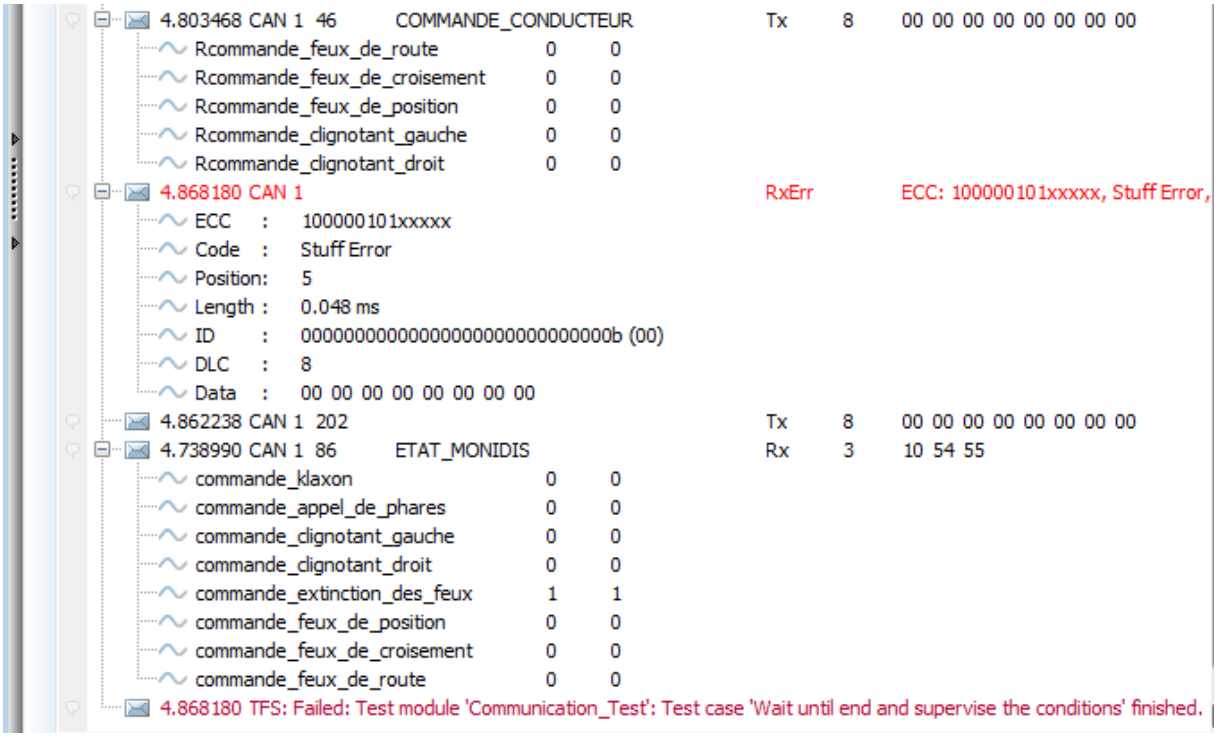


FIGURE 17 : LE GENERATEUR DE TRAMES D'ERREUR

Les trames d'erreur détectées sont enregistrées et ajoutées au rapport de test généré en fin d'essai. Le générateur d'« *errorframes* » nous permet de générer par ailleurs des trames à identifiant inconnu. Le but est de vérifier l'intégrité du bus au regard de trames non prévues dans les spécifications. Grâce à cette manipulation, nous pouvons surcharger le bus et augmenter les contraintes appliquées au réseau lors du contrôle de la redondance des trames CAN.

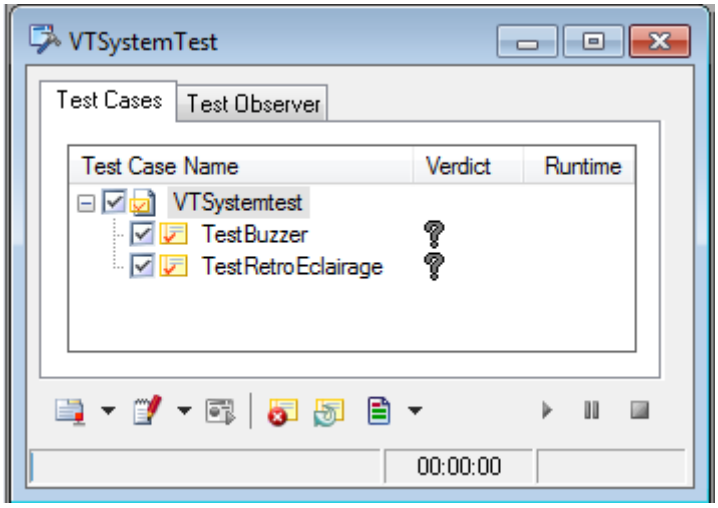


2. Tests analogiques à l'aide de modules VTSystem

En parallèle des validations purement réseau du bus CAN, le calculateur possédant des interfaces entrées/sorties analogiques doit aussi répondre aux spécifications de fonctionnement sur activation de ces entrées. Le module VT-System répond à nos besoins en termes de simulation de signaux analogiques. Les exigences explicitées dans le premier chapitre donnent l'allure du plan de test qui sera déroulé.

2.a Simulation de l'entrée « buzzer » et du rétroéclairage traités par le calculateur

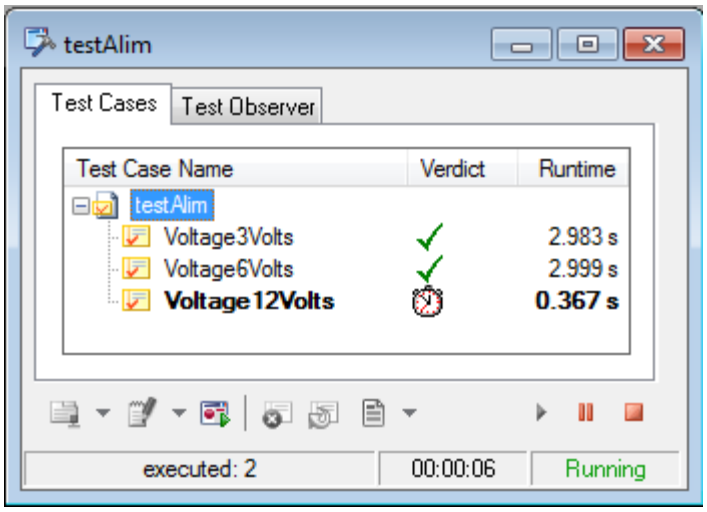
Les deux entrées analogiques principales (hors alimentation) sont les entrées du buzzer et du rétroéclairage. Ces entrées fonctionnent sur 12V, en différentiel avec la masse du calculateur. Le test de ces deux entrées sera intégré au plan de test en parallèle du fonctionnement du MONIDIS afin de valider le traitement de celle-ci sans générer d'interruptions à la routine principale de l'applicatif du calculateur.



Le codage des tests en CAPL repose sur des principes différentes des nœuds standards, dans le sens où les nœuds de test ne sont pas tournés événements mais scénarios de test. Les étapes et le déroulement sont donc préprogrammées et déroulées de manière figée. Les sources de programmation sont disponibles en Annexe 5.

2.b Tests d'alimentation du calculateur

Le calculateur MONIDIS est typiquement alimenté sous 12V continu, référence de tension couramment utilisée dans l'automobile. Or, la tension disponible dans l'habitacle véhicule n'est jamais stabilisée. En effet, le fonctionnement de l'alternateur, ainsi que les phases de démarrage véhicule ou les coupures contacts obligent les fournisseurs à proposer une gamme de tension d'alimentation pour le calculateur. A l'aide du VT-System, un profil de tension sera appliqué au calculateur afin de valider la plage s'utilisation de celui-ci.

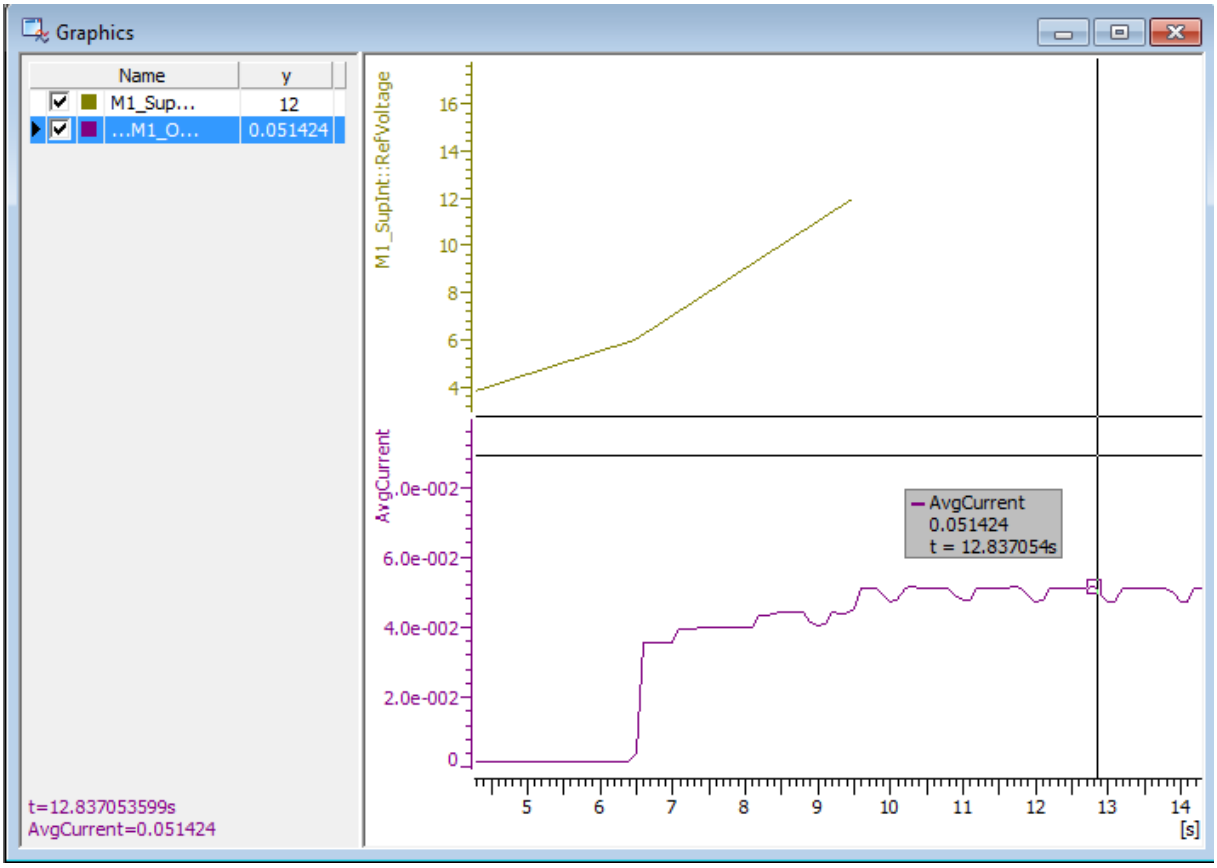


2.c Mesure du courant consommé en fonction des actionneurs

La consommation en courant du calculateur est typiquement une étape de plus en plus importante dans les validations véhicule d'aujourd'hui, d'avantage encore avec l'arrivée sur le marché des véhicules électriques. Le MONIDIS a une consommation en courant définie par les normes de développement du calculateur, et présenté au constructeur avant d'être validé par celui-ci. Ces consommations sont souvent moyennées pour un usage dit « fonctionnel ». Lors de l'activation des prestations du MONIDIS, des pics de consommation de courant dépassent régulièrement cette valeur. Nous serons donc amenés à tester les différentes consommations en courant du calculateur en fonction de ses phases d'utilisation.

	Prestation	Voyant	Consommation (mA)
Fonctionnement standard		éteint	37mA
Appui bouton feux de position	Allumage feux de position	Voyant orange clignotant	8mA
Appui bouton feux de croisement	Allumage feux de croisement	Voyant vert clignotant	8mA
Appui bouton feux de route	Allumage feux de route	Voyant bleu clignotant	10mA
Appui bouton clignotant G ou D	Allumage clignotant G ou D	Voyant Orange fixe	3mA

TABLEAU I I I : COURANT MESURE DES PRESTATIONS DU CALCULATEUR



Le rebond observé correspond à l'activation régulière du voyant d'état du MONIDIS. Celui-ci représente la majeure part de la consommation en courant du système.

En utilisation nominale, le MONIDIS ne dépasse pas, toutes prestations activées, une consommation en courant de 52mA. Les valeurs résultantes de nos tests valident les spécifications d'entrée, de 80mA maximum, et de 40mA en fonctionnement standard, sans prestations.

3. Tests d'applicatif du MONIDIS

3.a Création et implémentation de l'interface de test

Dans la mesure où les montées de signaux du MONIDIS doivent être contrôlées, une vérification automatique du réseau CAN et des champs de données des trames doivent pouvoir être maitrisées par le testeur de manière transparente. Pour se faire, une IHM de test est créée afin de guider l'utilisateur dans la séquence de test à dérouler. Celui sera alors sollicité pour intervenir sur les touches du MONIDIS et le test décrit en CAPL et en langage xml à l'aide de TAE (*Test Automation Editor*) permettra de vérifier la propagation de la commande jusqu'au CMB et à la gestion de son affichage.

Le test est essentiellement décrit grâce à l'interface de programmation xml afin de définir les étapes et les conditions de validation des tests.

L'IHM de contrôle du test est dessinée à l'aide de panel designer, les notions de temps d'intervention du testeur et la barre de progression seront codées à l'aide d'une fonction CAPL appelée par la routine xml sous la forme d'une librairie.

Le codage de la séquence de test est disponible en Annexe 5.

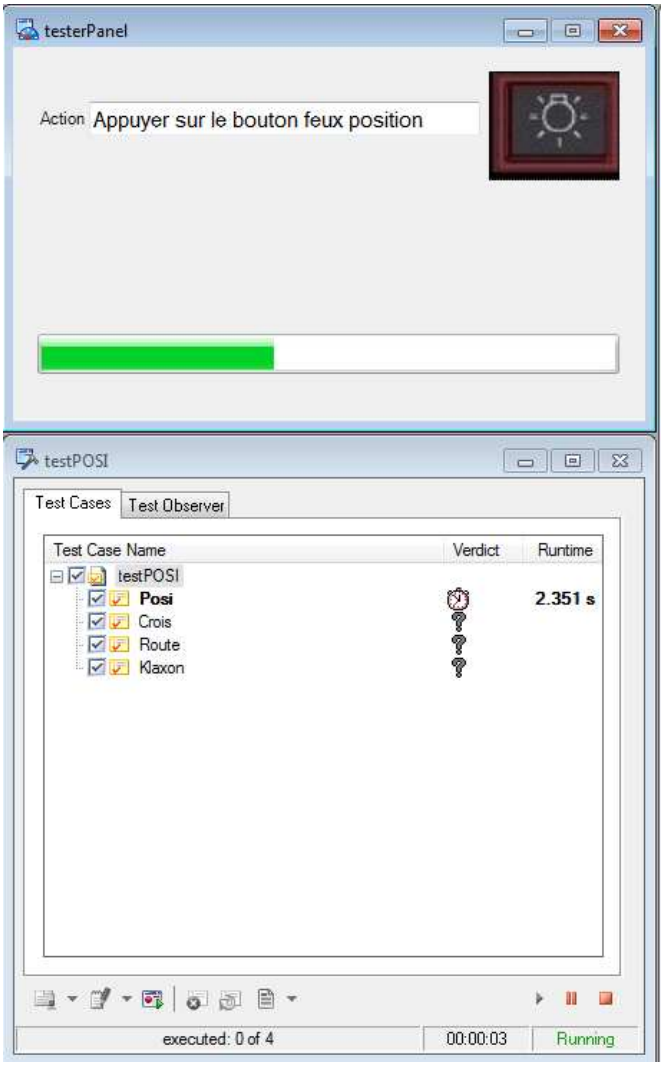


FIGURE 18 : L'IHM DE CONTROLE DE TEST

3.b Validation des fonctionnalités du MONIDIS via le CMB

En parallèle du test d'applicatif, le CMB nous permet de visualiser l'état des signaux correspondants sur le réseau véhicule constructeur. La validation des fonctionnalités du MONIDIS est donc complétée par une séquence de test visuelle au plus proche de la situation du produit dans son milieu d'utilisation.

Que l'activation des feux véhicule soit enclenchée par le conducteur ou par le MONIDIS, les voyants du tableau de bord sont mis à jour via la trame 0x128(CDE_COMBINE_SIGNALISATION).

Une possibilité de validation a été proposée par un de nos clients, Magnetti Marelli afin de remonter le positionnement des aiguilles et des signalisations du tableau de bord via une caméra à capture de mouvement. Des ROI (*Region Of Interest*) sont enregistrés via un logiciel de capture vidéo dont les valeurs vont être retournées par une interface National Instruments sous forme de forme de signaux digitaux pouvant être par conséquent intégrés à un plan de test sous CANoe via les variables systèmes[5].

Le modèle réel de notre bus est validé, il est possible d'élargir les scénarios de tests par rapport à l'ECU CMB quant à la qualité de l'affichage des données et les sorties de celui-ci sur le réseau véhicule. Toutefois, le but du banc étant principalement de valider le MONIDIS, ces types de tests peuvent faire l'objet d'une extension de la plateforme réalisée.

FIGURE 19: LE COMBINE PSA

CONCLUSION

Le déroulement de ce mémoire aura permis de comprendre la complexité d'un projet dans les systèmes embarqués. En recréant le contexte de travail et les outils disponibles au sein d'un département de validation, nous avons pu répondre aux exigences d'un client, en tant que fournisseur d'un système de développement et de validation. En travaillant avec les contraintes inhérentes au métier de l'ingénieur, les défis techniques rencontrés ont permis de faire évoluer le cahier des charges standard afin d'étendre notre démonstrateur à un réseau constructeur.

Afin de valider notre calculateur dépendant d'un réseau complet et non disponible, nous avons été amenés à recréer un réseau CAN environnant en simulant les données de stimulation de l'organe. En outre, ce réseau nous a permis de développer les scénarios de tests et les perturbations avec lesquelles le calculateur devra valider ses spécifications. C'est à travers tous les domaines d'utilisation du calculateur, aussi bien par son interface électrique que par sa stratégie de fonctionnement et enfin son modèle de communication, que le système complet a pu être testé. En outre, notre démonstrateur pourra s'adapter à d'éventuelles évolutions de spécifications, de par sa nature modulable et programmable. De nouveaux éléments de validation des couches basses du CAN peuvent être envisagés, afin de valider les niveaux électriques de notre protocole et vérifier leur stabilité en milieu CEM (compatibilité électromagnétique).

Dans l'optique de valider les compétences de l'ingénieur, ce mémoire aura été l'occasion d'estimer à sa juste valeur la difficulté de tenir des échéances, au milieu d'un environnement en changement, tant par la nature des produits à tester que les moyens en temps, et en matériel. Le projet ayant été mené dans le cadre de mon métier d'ingénieur d'application, celui-ci a développé mes compétences dans les domaines de la programmation et le développement de systèmes embarqués. Ce mémoire servira de référence pour l'élaboration de futurs projets.

ANNEXE1

SOJADIS

Notice d'utilisation

MONIDIS II

V1

02/08/2011

Table des Matières

Utilisation du produit MONIDIS II

Présentation

Le *MONIDIS II* est une commande électrique destinée aux moniteurs auto-école. Elle est composée de 7 touches fonctions, de 4 voyants d'états du véhicule et d'un rétro-éclairage du clavier.

Touches fonctions

Le *MONIDIS II* permet la prise de main sur les commandes d'origine du véhicule, grâce à un clavier composé de 7 touches.

Voyants d'état véhicule

Le *MONIDIS II* permet la prise de main sur les commandes d'origine du véhicule, et informe le moniteur sur l'état des clignotants* et des Feux.

** Suivant le véhicule, les voyants d'états clignotant peuvent clignoter ou s'allumer fixe*

Rétro-éclairage

Le *MONIDIS II* dispose d'un clavier rétro-éclairé, qui s'active automatiquement avec l'éclairage du véhicule, quelle que soit l'origine de la commande (Moniteur/Elève).

Luminosité auto-adaptative

Le *MONIDIS II* dispose d'un système de luminosité auto-adaptatif, permettant d'abaisser automatiquement la luminosité des voyants d'état véhicule, lorsque les feux du véhicule sont allumés. Ceci permet de supprimer le risque de gène, lors du roulage de nuit.

3

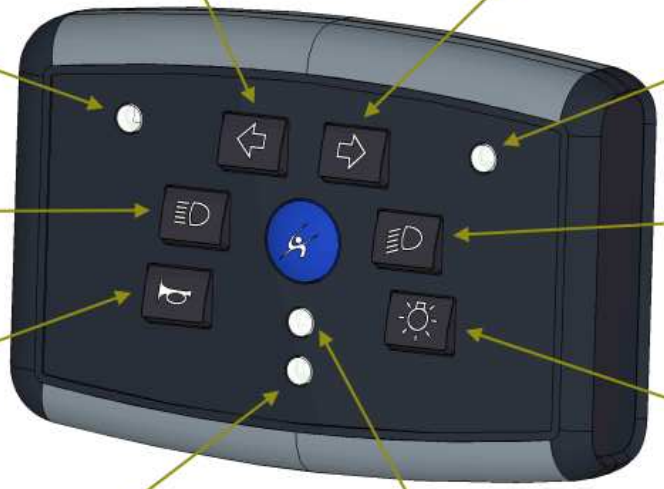
Utilisation du MONIDIS II

Clignotant gauche
Impulsion => *activation / désactivation*

Clignotant droit
Impulsion => *activation / désactivation*

Etat véhicule clignotant gauche
ETEINT => *clignotant gauche éteint*
VERT => *activation par l'élève*
ORANGE => *activation par le moniteur*

Etat véhicule clignotant droit
ETEINT => *clignotant droit éteint*
VERT => *activation par l'élève*
ORANGE => *activation par le moniteur*



Appel de phares / feux route
Impulsion => *Appel de phares*
Appui maintenu => *activation feux de route*

Feux de croisement
Impulsion => *activation feux de croisement*
Appui maintenu => *extinction feux de position / croisement / route*

Avertisseur sonore
Activé en restant appuyé

Feux de position
Impulsion => *activation feux de position*
Appui maintenu => *extinction feux de position / croisement / route*

Voyant ON/OFF
ROUGE => *commande auto-école alimentée*
ETEINT => *commande auto-école non alimentée*

Etat Véhicule Eclairage
ORANGE => *Feux de position activés **
VERT => *Feux de croisement activés **
BLEU => *Feux de route / appel de phares activés **

** Voyant clignotant, si le moniteur est à l'origine de la commande*

Utilisation des fonctions

La commande électrique ***MONIDIS II*** est équipée de 7 touches fonctions, permettant au moniteur d'activer les commandes du véhicule.

Fonction		Action	Touche
Clignotant droit	Marche	Impulsion	
	Arrêt	Impulsion	
Clignotant gauche	Marche	Impulsion	
	Arrêt	Impulsion	
Feux de position	Allumer	Impulsion	
	Eteindre	Appui maintenu*	ou
Feux de croisement	Allumer	Impulsion	
	Eteindre	Appui maintenu*	ou
Feux de route	Allumer	Appui maintenu*	
	Eteindre	Appui maintenu*	ou
Appel de phares	Allumer	Impulsion	
	Eteindre	Relâcher l'impulsion	
Avertisseur sonore	Activer	Appui maintenu	

** L'appui maintenu est signalé par un court signal sonore*

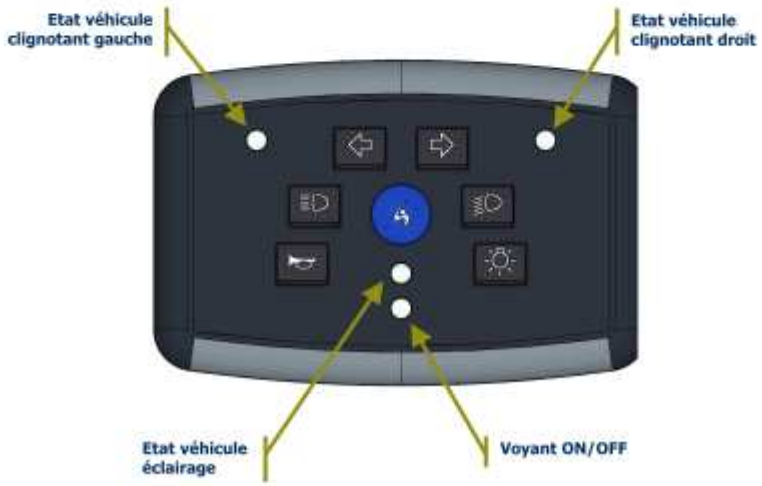
SOJADIS Notice d'utilisation MONIDIS II

Définition des voyants d'état véhicule

La commande électrique *MONIDIS II* est équipée de 4 voyants d'états, permettant au moniteur de connaître en temps réel l'état du véhicule.

Voyant d'état		Etat véhicule	Origine de la commande	
			Elève	Moniteur
Clignotant droit	ETEINT	Clignotant D éteint		
	VERT	Clignotant D allumé	X	
	ORANGE	Clignotant D allumé		X
Clignotant gauche	ETEINT	Clignotant G éteint		
	VERT	Clignotant G allumé	X	
	ORANGE	Clignotant G allumé		X
Eclairage	ETEINT	Feux éteints		
	ORANGE	Feux de position allumés	ALLUMÉ FIXE *	CLIGNOTANT *
	VERT	Feux de croisement allumés	ALLUMÉ FIXE *	CLIGNOTANT *
	BLEU	Feux de route / appel de phares allumés	ALLUMÉ FIXE *	CLIGNOTANT *
ON / OFF	ETEINT	*MONIDIS II* non alimenté		
	ROUGE	*MONIDIS II* alimenté		

** Suivant véhicule*



6

Paramétrage de la commande *MONIDIS II*

L'utilisateur peut régler des paramètres liés au fonctionnement de la commande ***MONIDIS II***.
L'entrée en mode paramétrage est indiquée par des informations sonores et visuelles :

- « BIP » sonore
- clignotement de la LED d'état ON/OFF
- activation des 2 voyants verts de clignotants

Notes :

- *L'entrée en mode paramétrage n'est autorisée seulement que pendant les 30 premières secondes après la mise du contact*
- *Les commandes **MONIDIS II** sont inhibés durant le paramétrage*
- *Tous paramétrages d'une durée supérieure à 5 min est automatiquement annulé*
- *La coupure du contact annule toute procédure de paramétrage en cours*

Mode « Belgique »

Le ***MONIDIS II*** permet de s'adapter à la législation en vigueur en Belgique. L'activation du mode « Belgique » permet, lorsque les pédales du moniteur sont actionnées :

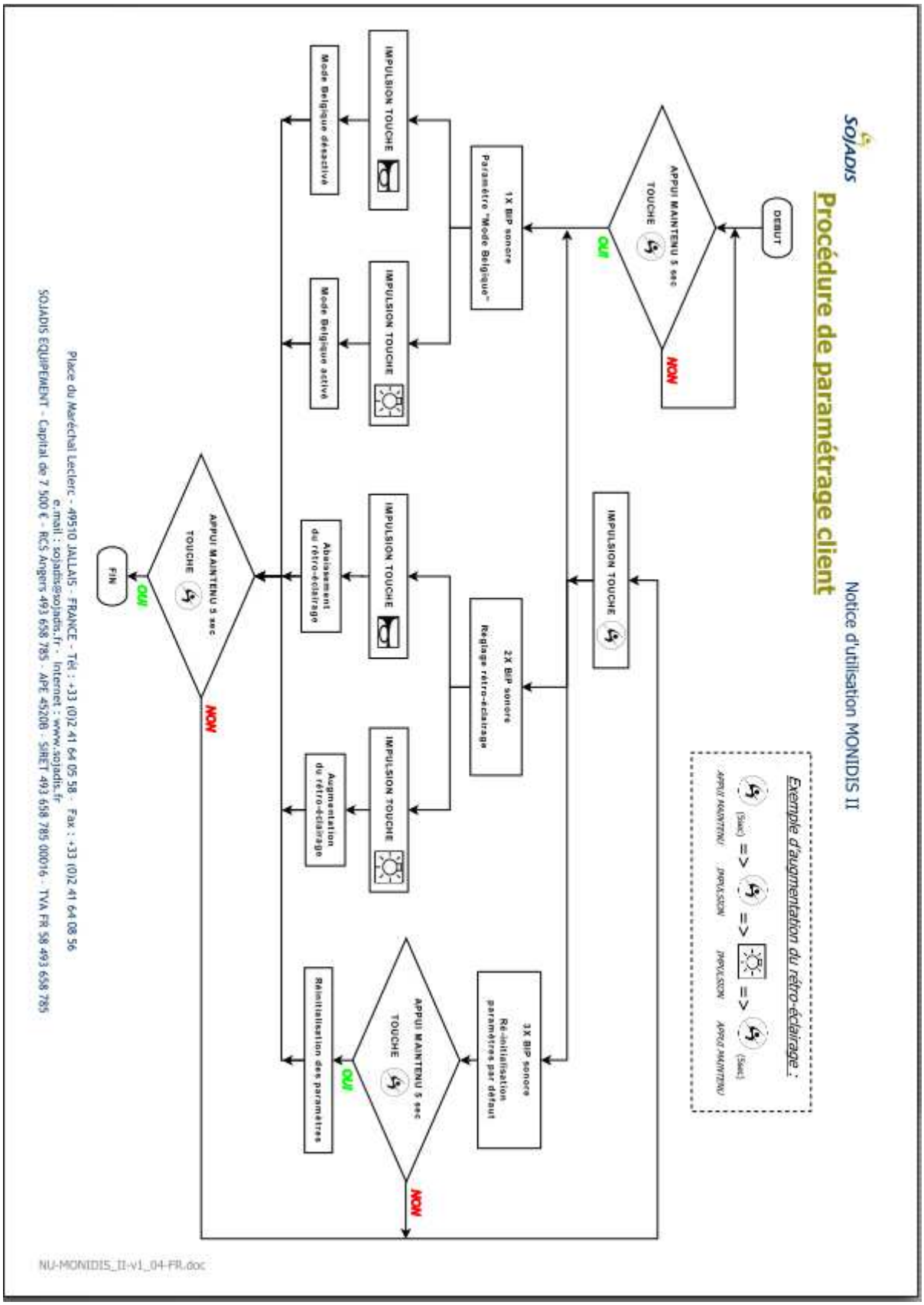
- d'avoir un retour sonore (interne au ***MONIDIS II***)
- d'éteindre le voyant d'état ON/OFF

Réglage de la luminosité du rétro-éclairage

Le rétro-éclairage du ***MONIDIS II*** peut être réglé, pour un meilleur confort de l'utilisateur.

Réinitialisation des paramètres par défaut

Les paramètres utilisateur du ***MONIDIS II*** peuvent tous être réinitialisés par défaut, afin de configurer la commande en mode « sortie d'usine ».


SOJADIS
Notice d'utilisation MONIDIS II
Procédure de paramétrage client
DEBUT
APPUI MAINTENU 5 sec
TOUCHE
OUI
NON
1X BIP sonore
Paramètre "Mode Belgique"
IMPULSION TOUCHE
Mode Belgique désactivé
Mode Belgique activé
IMPULSION TOUCHE
Mode Belgique activé
IMPULSION TOUCHE
2X BIP sonore
Réglage rétro-éclairage
Abaissement du rétro-éclairage
IMPULSION TOUCHE
Augmentation du rétro-éclairage
IMPULSION TOUCHE
3X BIP sonore
Ré-initialisation paramètres par défaut
Réinitialisation des paramètres
APPUI MAINTENU 5 sec
TOUCHE
OUI
NON
APPUI MAINTENU 5 sec
TOUCHE
OUI
NON
FIN
Exemple d'augmentation du rétro-éclairage :
(5sec) => => => (5sec)
APPUI MAINTENU
IMPULSION
IMPULSION
APPUI MAINTENU
Place du Maréchal Leclerc - 49510 JALLAIS - FRANCE - Tél : +33 (0)2 41 64 05 58 - Fax : +33 (0)2 41 64 08 56
e.mail : sojadis@sojadis.fr - Internet : www.sojadis.fr
SOJADIS EQUIPEMENT - Capital de 7 500 € - RCS Angers 493 658 785 - APE 4520B - SIRET 493 658 785 00016 - TVA FR 58 493 658 785
NU-MONIDIS_II-v1_04-FR.doc

ANNEXE 2 sources de l'applicatif du modèle simulé.

```
/* ------- Sendmodul --- begin generated Comment ----
| CAPL template for CANoe IL models
| ActiveX based generation
| Functionality is done in CANoe directly - you can add application code for this node
   ---------- end generated Comment---- */

variables
{
  //Please insert your code below this comment
Timer tButtonFeuxCroisement;
  enum enumFuncButtonFeuxCroisement {feuxCroisement, extinctionToutFeux} eFuncButtonFeuxCroisement;

  Timer tButtonFeuxPosition;
  enum enumFuncButtonFeuxPosition {feuxPosition, extinctionToutFeux2} eFuncButtonFeuxPosition;

  Timer tButtonFeuxRoute;
  enum enumFuncButtonFeuxRoute {AppelfeuxRoute, FeuxRoute} eFuncButtonFeuxRoute;

  enum enumFuncButtonCliG {CliGActivé, CliGDesactivé} eFuncButtonCliG;

  enum enumFuncButtonCliD {CliDActivé, CliDDesactivé} eFuncButtonCliD;
/*
  enum enumFuncActiCliG {VoyantCliGMoni, VoyantCliGConducteur} eFuncActiCliG;

  enum enumFuncActiCliD {VoyantCliDMoni, VoyantCliDConducteur} eFuncActiCliD;*/

  enum enumFuncvoyantfeux_P {VoyantFeuxP0, VoyantFeuxP1} eFuncvoyantfeux_P;
  enum enumFuncvoyantfeux_C {VoyantFeuxC0, VoyantFeuxC1} eFuncvoyantfeux_C;
  enum enumFuncvoyantfeux_R {VoyantFeuxR0, VoyantFeuxR1} eFuncvoyantfeux_R;
```

```
  int valueCR=0;
  Timer tMoniVoyant;

  int valuecliD=0;
  int valuecliG=0;
  int valueposi;
  int valuecrois;
  int valueroute;
  int valueRposi;
  int valueRcrois;
  int valueRroute;
}

on start
{
eFuncButtonCliG=CliGDesactivé;
eFuncButtonCliD=CliDDesactivé;

eFuncvoyantfeux_P=VoyantFeuxP0;
eFuncvoyantfeux_C=VoyantFeuxC0;
eFuncvoyantfeux_R=VoyantFeuxR0;
  //Please insert your code below this comment
@sysvar::voyant_feuxCR::voyant_feuxCR=0;
@sysvar::Supply_led::Supply_led=1;
}

on stopMeasurement
{
  //Please insert your code below this comment
@sysvar::Supply_led::Supply_led=0;
}
```

```
/* Gestion des appuis boutons panels*/
on sysvar sysvar::crois::crois
{
  if(@sysvar::crois::crois==1)
    {
      eFuncButtonFeuxCroisement=feuxCroisement;
      setTimer(tButtonFeuxCroisement,1);
    }


  if(@sysvar::crois::crois==0 && eFuncButtonFeuxCroisement==feuxCroisement)
    {
      setsignal(commande_extinction_des_feux,0);
      setsignal(commande_feux_de_croisement,1);
      setsignal(commande_feux_de_position,0);
      setsignal(commande_feux_de_route,0);
    }

}

on timer tButtonFeuxCroisement
{
  if(@sysvar::crois::crois==1)
  {
  eFuncButtonFeuxCroisement=extinctionToutFeux;
  setsignal(commande_extinction_des_feux,1);
  canceltimer(tButtonFeuxCroisement);
  write("generalfinish");
  if (eFuncvoyantfeux_C==VoyantFeuxC0&&eFuncvoyantfeux_P==VoyantFeuxP0&&eFuncvoyantfeux_R==VoyantFeuxR0)
  {
  @sysvar::voyant_feuxCR::voyant_feuxCR=0;
  valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
  }
  else @sysvar::voyant_feuxCR::voyant_feuxCR=valueCR;
```

```
  }
}

on sysvar sysvar::posi::posi
{
  if(@sysvar::posi::posi==1)
    {
      eFuncButtonFeuxPosition=feuxPosition;
      setTimer(tButtonFeuxPosition,1);
    }


  if(@sysvar::posi::posi==0 && eFuncButtonFeuxPosition==feuxPosition)
    {
      setsignal(commande_extinction_des_feux,0);
      setsignal(commande_feux_de_position,1);
      setsignal(commande_feux_de_croisement,0);
      setsignal(commande_feux_de_route,0);
    }

}

on timer tButtonFeuxPosition
{
 if(@sysvar::posi::posi==1)
  {
  eFuncButtonFeuxPosition=extinctionToutFeux2;
  setsignal(commande_extinction_des_feux,1);
  canceltimer(tButtonFeuxPosition);
  write("generalfinish");
  if (eFuncvoyantfeux_C==VoyantFeuxC0&&eFuncvoyantfeux_P==VoyantFeuxP0&&eFuncvoyantfeux_R==VoyantFeuxR0)
  {
  @sysvar::voyant_feuxCR::voyant_feuxCR=0;
```

```
  valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
  }
  else @sysvar::voyant_feuxCR::voyant_feuxCR=valueCR;
  }
}

on sysvar sysvar::route::route
{
 if(@sysvar::route::route==1)
    {
      eFuncButtonFeuxRoute=AppelfeuxRoute;
      setsignal(commande_appel_de_phares,1);
      setTimer(tButtonFeuxRoute,1);
    }


  if(@sysvar::route::route==0 && eFuncButtonFeuxRoute==AppelfeuxRoute)
    {
      setsignal(commande_appel_de_phares,0);
    }
}

on timer tButtonFeuxRoute
{
if(@sysvar::route::route==1)
  {
  eFuncButtonFeuxRoute=FeuxRoute;
  setsignal(commande_feux_de_route,1);
  setsignal(commande_extinction_des_feux,0);
  setsignal(commande_appel_de_phares,0);
  setsignal(commande_feux_de_position,0);
  setsignal(commande_feux_de_croisement,0);
  }
}
```

```
on sysvar sysvar::clignoD::clignoD
{
if (@this==1)
{
  if (eFuncButtonCliD==CliDDesactivé)
  {
  setsignal(commande_clignotant_droit,1);
  setsignal(commande_clignotant_gauche,0);
  eFuncButtonCliG=CliGDesactivé;
  eFuncButtonCliD=CliDActivé;
  valuecliD=1;
  }

  else if (eFuncButtonCliD==CliDActivé)
  {
  setsignal(commande_clignotant_droit,0);
  eFuncButtonCliD=CliDDesactivé;
  }
}
}

on sysvar sysvar::clignoG::clignoG
{

if (@this==1)
{
  if (eFuncButtonCliG==CliGDesactivé)
  {
  setsignal(commande_clignotant_gauche,1);
  setsignal(commande_clignotant_droit,0);
  eFuncButtonCliD=CliDDesactivé;
  eFuncButtonCliG=CliGActivé;
  }
```

```
  else if (eFuncButtonCliG==CliGActivé)
  {
  setsignal(commande_clignotant_gauche,0);
  eFuncButtonCliG=CliGDesactivé;
  }
}
}

on signal_change Rcommande_clignotant_droit

{
if (this==1&&eFuncButtonCliD==CliDActivé)
  {
  valuecliG=0;
  @sysvar::voyant_clignoG::voyant_clignoG=valuecliG;
  valuecliD=1;
  @sysvar::voyant_clignoD::voyant_clignoD=valuecliD;
  }
  if (this==1&&eFuncButtonCliD==CliDDesactivé)
  {
   valuecliD=2;
  @sysvar::voyant_clignoD::voyant_clignoD=valuecliD;
  }
if (this==0)
   {
  valuecliD=0;
  @sysvar::voyant_clignoD::voyant_clignoD=valuecliD;
   }
}

on signal_change Rcommande_clignotant_gauche

{
```

```
if (this==1&&eFuncButtonCliG==CliGActivé)
   {
   write("entercg");
   write("entercg2");
  valuecliD=0;
   write("entercg3");
  @sysvar::voyant_clignoD::voyant_clignoD=valuecliD;
   write("entercg4");
  valuecliG=1;
  @sysvar::voyant_clignoG::voyant_clignoG=valuecliG;
  write("entercg5");
   }
   if (this==1&&eFuncButtonCliG==CliGDesactivé)
  {
   valuecliG=2;
  @sysvar::voyant_clignoG::voyant_clignoG=valuecliG;
  }
if (this==0)
   {
  valuecliG=0;
  @sysvar::voyant_clignoG::voyant_clignoG=valuecliG;
   }

}

on sysvar sysvar::klax::klax
{
if (@sysvar::klax::klax==1)setsignal(commande_klaxon,1);
if (@sysvar::klax::klax==0)setsignal(commande_klaxon,0);
}

on signal_change commande_extinction_des_feux
{
if (this==1)
```

```
{
setsignal(commande_feux_de_position,0);
setsignal(commande_feux_de_croisement,0);
setsignal(commande_feux_de_route,0);
canceltimer(tMoniVoyant);
eFuncButtonFeuxCroisement=extinctionToutFeux;
eFuncButtonFeuxPosition=extinctionToutFeux2;
eFuncButtonFeuxRoute=AppelfeuxRoute;
}
}
```

```
on signal_change Rcommande_feux_de_position
{
/* Gestion du clignotement voyant posi*/

if ((this==0x3)&&((eFuncButtonFeuxPosition==feuxPosition)))
  {
  canceltimer(tMoniVoyant);
  @sysvar::voyant_feuxCR::voyant_feuxCR=3;
  valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
  SetTimer(tMoniVoyant,1);
  }
if (this==0&&eFuncvoyantfeux_R==VoyantFeuxR0&&eFuncvoyantfeux_C==VoyantFeuxC0)
{
eFuncvoyantfeux_P=VoyantFeuxP0;
canceltimer(tMoniVoyant);
write("timeroffP");
@sysvar::voyant_feuxCR::voyant_feuxCR=0;
canceltimer(tMoniVoyant);
}
/* Gestion du voyant fixe posi*/
if (this==0x0) eFuncvoyantfeux_P=VoyantFeuxP0;
```

```
if (this==0x3) eFuncvoyantfeux_P=VoyantFeuxP1;

if
(this==0x3&&eFuncvoyantfeux_R==VoyantFeuxR0&&eFuncvoyantfeux_C==VoyantFeuxC0&&(eFuncButtonFeuxPosition!=feux
Position))

{

eFuncvoyantfeux_P=VoyantFeuxP1;

@sysvar::voyant_feuxCR::voyant_feuxCR=3;

valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;

}

if
(this==0x0&&(eFuncvoyantfeux_R==VoyantFeuxR0)&&(eFuncvoyantfeux_C==VoyantFeuxC0)&&(eFuncButtonFeuxRoute!=Fe
uxRoute)&&(eFuncButtonFeuxCroisement!=feuxCroisement))

{

eFuncvoyantfeux_P=VoyantFeuxP0;

@sysvar::voyant_feuxCR::voyant_feuxCR=0;

valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;

}

}


on signal_change Rcommande_feux_de_croisement

/* Gestion du clignotement voyant crois*/

{

write("on Rcom_crois");

if ((this==0x3)&&(eFuncButtonFeuxCroisement==feuxCroisement))

 {

 write("on testcondicroiscligno");

 canceltimer(tMoniVoyant);

 @sysvar::voyant_feuxCR::voyant_feuxCR=2;

 valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;

 SetTimer(tMoniVoyant,1);

 }

if (this==0&&eFuncvoyantfeux_R==VoyantFeuxR0&&eFuncvoyantfeux_P==VoyantFeuxP0)

{

eFuncvoyantfeux_C=VoyantFeuxC0;

canceltimer(tMoniVoyant);

write("timeroffCrois");
```

```
@sysvar::voyant_feuxCR::voyant_feuxCR=0;
canceltimer(tMoniVoyant);
}
/* Gestion du voyant fixe crois*/
if (this==0x0) eFuncvoyantfeux_C=VoyantFeuxC0;
if (this==0x3) eFuncvoyantfeux_C=VoyantFeuxC1;
if ((this==0x3)&&(eFuncvoyantfeux_R==VoyantFeuxR0)&&(eFuncButtonFeuxCroisement!=feuxCroisement))
 {
 eFuncvoyantfeux_C=VoyantFeuxC1;
 canceltimer(tMoniVoyant);
 @sysvar::voyant_feuxCR::voyant_feuxCR=2;
 valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
 }
if (this==0&&eFuncvoyantfeux_R==VoyantFeuxR0&&eFuncvoyantfeux_P==VoyantFeuxP1)
{
write("enter");
eFuncvoyantfeux_C=VoyantFeuxC0;
@sysvar::voyant_feuxCR::voyant_feuxCR=3;
valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
}
if (this==0&&eFuncvoyantfeux_R==VoyantFeuxR1&&eFuncvoyantfeux_P==VoyantFeuxP0)
{
eFuncvoyantfeux_C=VoyantFeuxC0;
@sysvar::voyant_feuxCR::voyant_feuxCR=1;
valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
}

}

on signal_change Rcommande_feux_de_route
{
/* Gestion du clignotement voyant Route*/
if ((this==1)&&(eFuncButtonFeuxRoute==FeuxRoute))
 {
```

```
 canceltimer(tMoniVoyant);
 @sysvar::voyant_feuxCR::voyant_feuxCR=1;
 valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
 SetTimer(tMoniVoyant,1);
 }
if
(this==0&&eFuncvoyantfeux_P==VoyantFeuxP0&&eFuncvoyantfeux_C==VoyantFeuxC0&&(eFuncButtonFeuxCroisement!=feu
xCroisement)&&(eFuncButtonFeuxPosition!=feuxPosition))
{
eFuncButtonFeuxRoute=AppelfeuxRoute;
eFuncvoyantfeux_R=VoyantFeuxR0;
canceltimer(tMoniVoyant);
write("timeroff");
@sysvar::voyant_feuxCR::voyant_feuxCR=0;
canceltimer(tMoniVoyant);
}
/* Gestion du voyant fixe route*/
if (this==1)
{
eFuncvoyantfeux_R=VoyantFeuxR1;
@sysvar::voyant_feuxCR::voyant_feuxCR=1;
valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
}
if (this==0&&eFuncvoyantfeux_C==VoyantFeuxC0&&eFuncvoyantfeux_P==VoyantFeuxP0)
{
eFuncvoyantfeux_R=VoyantFeuxR0;
@sysvar::voyant_feuxCR::voyant_feuxCR=0;
valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
}
if (this==0&&eFuncvoyantfeux_C==VoyantFeuxC1&&eFuncvoyantfeux_P==VoyantFeuxP0)
{
eFuncvoyantfeux_R=VoyantFeuxR0;
@sysvar::voyant_feuxCR::voyant_feuxCR=2;
valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
}
```

```
if (this==0&&eFuncvoyantfeux_C==VoyantFeuxC0&&eFuncvoyantfeux_P==VoyantFeuxP1)
{
eFuncvoyantfeux_R=VoyantFeuxR0;
@sysvar::voyant_feuxCR::voyant_feuxCR=3;
valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
}
if (this==0&&eFuncvoyantfeux_C==VoyantFeuxC1&&eFuncvoyantfeux_P==VoyantFeuxP1)
{
eFuncvoyantfeux_R=VoyantFeuxR0;
@sysvar::voyant_feuxCR::voyant_feuxCR=2;
valueCR=@sysvar::voyant_feuxCR::voyant_feuxCR;
}



}
```

```
on Timer tMoniVoyant
{
write("timer go");
if (@sysvar::voyant_feuxCR::voyant_feuxCR!=0) @sysvar::voyant_feuxCR::voyant_feuxCR=0;
else if (@sysvar::voyant_feuxCR::voyant_feuxCR==0)@sysvar::voyant_feuxCR::voyant_feuxCR=valueCR;
setTimer(tMoniVoyant,1);
}
```

Annexe 3 Applicatif BSI pour gestion des feux véhicule.

```
includes
{


}

variables
{


}

/* Applicatif BSI pour réception MONIDIS */

on signal commande_feux_de_croisement
{
if (this==1) setsignal (Rcommande_feux_de_croisement,0x3);
if (this==0) setsignal (Rcommande_feux_de_croisement,0x00);
}

on signal commande_feux_de_position
{
if (this==1) setsignal (Rcommande_feux_de_position,0x3);
if (this==0) setsignal (Rcommande_feux_de_position,0x00);
}

on signal commande_feux_de_route
{
if (this==1) setsignal (Rcommande_feux_de_route,0x1);
if (this==0) setsignal (Rcommande_feux_de_route,0x0);
}
```

```
on signal commande_clignotant_droit
{
if (this==1) setsignal (Rcommande_clignotant_droit,0x1);
if (this==0) setsignal (Rcommande_clignotant_droit,0x0);
}

on signal commande_clignotant_gauche
{
if (this==1) setsignal (Rcommande_clignotant_gauche,0x1);
if (this==0) setsignal (Rcommande_clignotant_gauche,0x0);
}



/* Applicatif BSI pour réception conducteur*/

on signal Rcommande_clignotant_gauche
{
if(this==1)
        setsignal(CLIGNO_G,1);

 else if (this==0)
 setsignal(CLIGNO_G,0);
}

on signal Rcommande_clignotant_droit
{
        if(this==1)
        setsignal(CLIGNO_D,1);

 else if (this==0)
 setsignal(CLIGNO_D,0);
}
```

```
on signal Rcommande_feux_de_position
{
if(this==3)
        setsignal(ETAT_JN,1);

  else if (this==0)
  setsignal(ETAT_JN,0);

}

on signal Rcommande_feux_de_croisement
{
if(this==3)
        setsignal(FEUX_CROIS,1);

  else if (this==0)
  setsignal(FEUX_CROIS,0);

}

on signal Rcommande_feux_de_route
{
        if(this==1)
        setsignal(FEUX_ROUTE,1);

  else if (this==0)
  setsignal(FEUX_ROUTE,0);
}
on key'm'
{
  setsignal(Envoi_period_trame_etat_MONIDIS,1);

}
```

```
on key'c'
{
  setsignal(PHASE_VIE,1);
  setsignal(ETAT_PRINCIP_SEV,1);
  setsignal(ON_CMB,1);
  setsignal(LUMINOSITE,15);
 }
```

INTEGRATED CIRCUITS

DATA SHEET

TJA1054
Fault-tolerant CAN transceiver

Preliminary specification
File under Integrated Circuits, IC18

1999 Feb 11

Philips
Semiconductors

PHILIPS

Fault-tolerant CAN transceiver TJA1054

FEATURES

Optimized for in-car low-speed communication

- Baud rate up to 125 kBaud
- Up to 32 nodes can be connected
- Supports unshielded bus wires
- Very low Radio Frequency Interference (RFI) due to built-in slope control function and a very good matching of the CANL and CANH bus outputs
- Fully integrated receiver filters
- Permanent dominant monitoring of transmit data input
- Good immunity performance of ElectroMagnetic Compatibility (EMC) in normal operating mode and in low power modes.

Bus failure management

- Supports single-wire transmission modes with ground offset voltages up to 1.5 V
- Automatic switching to single-wire mode in the event of bus failures, even when the CANH bus wire is short-circuited to V_{CC}
- Automatic reset to differential mode if bus failure is removed
- Fully wake-up capability during failure modes.

Protection

- Short-circuit proof to battery and ground in 12 V powered systems
- Thermally protected
- Bus lines protected against transients in an automotive environment
- An unpowered node does not disturb the bus lines.

Support for low power modes

- Low current sleep and standby mode with wake-up via the bus lines
- Power-on reset flag on the output.

GENERAL DESCRIPTION

The TJA1054 is the interface between the protocol controller and the physical wires of the bus lines in a Control Area Network (CAN). It is primarily intended for low-speed applications, up to 125 kBaud, in passenger cars. The device provides differential transmit capability but will switch in error conditions to single-wire transmitter and/or receiver.

The TJA1054T is pin and upwards compatible with the PCA82C252T and the TJA1053T. This means that these two devices can be replaced by the TJA1054T with retention of all functions.

The most important improvements are:

- Very low RFI due to a very good matching of the CANL and CANH bus lines outputs
- Good immunity performance of EMC, especially in low power modes
- Fully wake-up capability during failure modes
- Extended bus failure management including short-circuit of the CANH bus line to V_{CC}
- Supports easy fault localization
- Two-edge sensitive wake-up input signal via pin $\overline{WAKE}$.

ORDERING INFORMATION

TYPE NUMBER	PACKAGE		
	NAME	DESCRIPTION	VERSION
TJA1054T	SO14	plastic small outline package; 14 leads; body width 3.9 mm	SOT108-1

QUICK REFERENCE DATA

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
V_{CC}	supply voltage on pin V_{CC}		4.75	–	5.25	V
V_{BAT}	battery voltage on pin BAT	no time limit	–0.3	–	+40	V
		operating mode	5.0	–	27	V
		load dump	–	–	40	V
I_{BAT}	battery current on pin BAT	Sleep mode; V_{CC} = 0 V; V_{BAT} = 12 V	–	30	50	µA
V_{CANH}	CANH bus line voltage	V_{CC} = 0 to 5.5 V; $V_{BAT} \geq 0$ V; no time limit	–40	–	+40	V
V_{CANL}	CANL bus line voltage	V_{CC} = 0 to 5.5 V; $V_{BAT} \geq 0$ V; no time limit	–40	–	+40	V
ΔV_{CANH}	CANH bus line transmitter voltage drop	I_{CANH} = –40 mA	–	–	1.4	V
ΔV_{CANL}	CANH bus line transmitter voltage drop	I_{CANL} = 40 mA	–	–	1.4	V
t_{PD}	propagation delay	TXD to RXD	–	1	–	µs
t_r	bus line output rise time	10 to 90%; C1 = 10 nF	–	0.6	–	µs
t_f	bus line output fall time	90 to 10%; C1 = 1 nF	–	0.3	–	µs
T_{amb}	operating ambient temperature		–40	–	+125	°C

BLOCK DIAGRAM

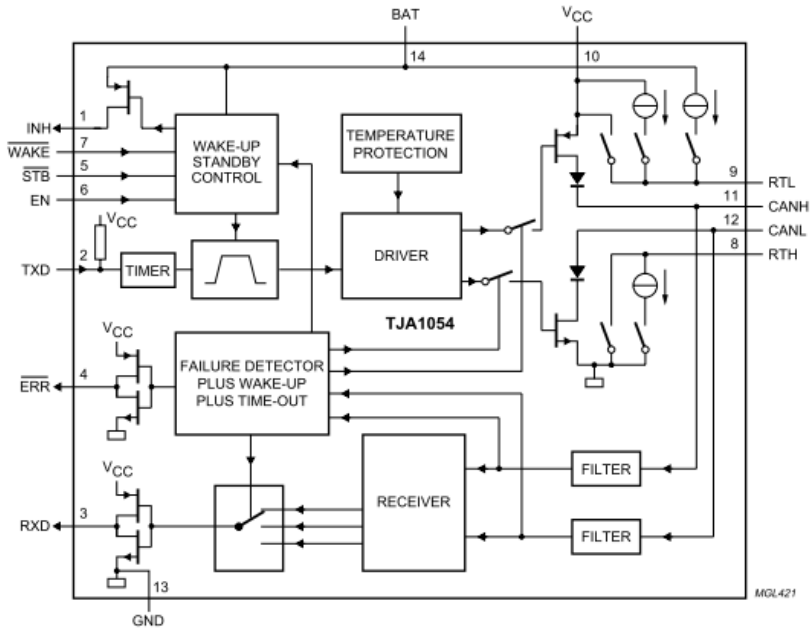


Fig.1 Block diagram.

PINNING

SYMBOL	PIN	DESCRIPTION
INH	1	inhibit output for switching an external voltage regulator if a wake-up signal occurs
TXD	2	transmit data input for activating the driver to the bus lines
RXD	3	receive data output for reading out the data from the bus lines
$\overline{ERR}$	4	error, wake-up and power-on indication output; active LOW in normal operating mode when the bus has a failure and in low power modes (wake-up signal or in power-on standby)
$\overline{STB}$	5	standby digital control signal input (active LOW); defines together with input signal on pin EN the state of the transceiver (in normal and low power modes); see Table 2 and Fig.3
EN	6	enable digital control signal input; defines together with input signal on pin $\overline{STB}$ the state of the transceiver (in normal and low power modes); see Table 2 and Fig.3
$\overline{WAKE}$	7	local wake-up signal input; falling and rising edges are both detected
RTH	8	termination resistor connection; in case of a CANH bus wire error the line is terminated with a selectable impedance
RTL	9	termination resistor connection; in case of a CANL bus wire the line is terminated with a selectable impedance
V_{CC}	10	supply voltage
CANH	11	HIGH-level voltage bus line
CANL	12	LOW-level voltage bus line
GND	13	ground
BAT	14	battery supply

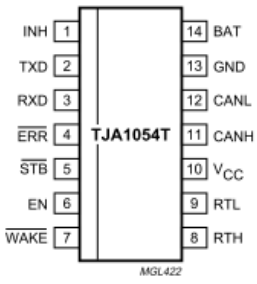


Fig.2 Pin configuration.

FUNCTIONAL DESCRIPTION

The TJA1054 is the interface between the CAN protocol controller and the physical wires of the CAN bus (see Fig.7). It is primarily intended for low speed applications, up to 125 kBaud, in passenger cars. The device provides differential transmit capability to the CAN bus and differential receive capability to the CAN controller.

To reduce RFI, the rise and fall slope are limited. This allows the use of an unshielded twisted pair or a parallel pair of wires for the bus lines. Moreover, it supports transmission capability on either bus line if one of the wires is corrupted. The failure detection logic automatically selects a suitable transmission mode.

In normal operating mode (no wiring failures) the differential receiver is output on pin RXD (see Fig.1). The differential receiver inputs are connected to pins CANH and CANL through integrated filters. The filtered input signals are also used for the single-wire receivers. The receivers connected to pins CANH and CANL have threshold voltages that ensure a maximum noise margin in single-wire mode.

A timer has been integrated at pin TXD. This timer prevents the TJA1054 from driving the bus lines to a permanent dominant state.

Failure detector

The failure detector is fully active in the normal operating mode. After the detection of a single bus failure the detector switches to the appropriate mode (see Table 1).

Table 1 Bus failures

FAILURE	DESCRIPTION
1	CANH wire interrupted
2	CANL wire interrupted
3	CANH short-circuited to battery
3a	CANH short-circuited to V_{CC}
4	CANL short-circuited to ground
5	CANH short-circuited to ground
6	CANL short-circuited to battery
6a	CANL short-circuited to V_{CC}
7	CANL mutually short-circuited to CANH

The differential receiver threshold voltage is set at −3.2 V typically (V_{CC} = 5 V). This ensures correct reception with a noise margin as high as possible in the normal operating mode and in the event of failures 1, 2, 4 and 6a. These failures, or recovery from them, do not destroy ongoing transmissions.

Failures 3 and 6 are detected by comparators connected to the CANH and CANL bus lines, respectively. If the comparator threshold is exceeded for a certain period of time, the reception is switched to the single-wire mode. This time is needed to avoid false triggering by external RF fields. Recovery from these failures is detected automatically after a certain time-out (filtering) and no transmission is lost. In the event of failure 3 the CANH driver and pin RTH are switched off. In the event of failure 6 the CANL driver and pin RTL are switched off. The pull-up current on pin RTL and the pull-down current on pin RTH will not be switched off.

Failures 3a, 4 and 7 initially result in a permanent dominant level on pin RXD. After a time-out, the CANL driver and pin RTL are switched off (failures 4 and 7) or the CANH driver and pin RTH are switched off (failure 3a). Only a weak pull-up on pin RTL or a weak pull-down on pin RTH remains. Reception continues by switching to the single-wire mode via pins CANH or CANL. When failures 3a, 4 or 7 are removed, the recessive bus levels are restored. If the differential voltage remains below the recessive threshold level for a certain period of time, reception and transmission switch back to the differential mode.

If any of the wiring failure occurs, the output signal on pin $\overline{ERR}$ will become LOW. On error recovery, the output signal on pin $\overline{ERR}$ will become HIGH again.

During all single-wire transmissions, the EMC performance (both immunity and emission) is worse than in the differential mode. The integrated receiver filters suppress any HF noise induced into the bus wires. The cut-off frequency of these filters is a compromise between propagation delay and HF suppression. In the single-wire mode, LF noise cannot be distinguished from the required signal.

Low power modes

The transceiver provides 3 low power modes which can be entered and exited via pins $\overline{STB}$ and EN (see Table 2 and Fig.3).

The Sleep mode is the mode with the lowest power consumption. Pin INH is switched to high-impedance for deactivation of the external voltage regulator. Pin CANL is biased to the battery voltage via pin RTL. If the supply voltage is provided pins RXD and $\overline{ERR}$ will signal the wake-up interrupt signal.

The standby mode will react the same as the Sleep mode but with a HIGH-level on pin INH.

The power-on standby mode is the same as the standby mode with the battery power-on flag instead of the wake-up interrupt signal on pin $\overline{ERR}$. The output on pin RXD will show the wake-up interrupt. This mode is only for reading out the power-on flag.

Wake-up requests are recognized by the transceiver when a dominant signal is detected on either bus line or if pin $\overline{WAKE}$ detects an edge (rising or falling) which stays longer HIGH or LOW respectively during a certain period of time. On a wake-up request the transceiver will set the output on pin INH which can be used to activate the external supply voltage regulator.

If V_{CC} is provided the wake-up request can be read on the $\overline{ERR}$ or RXD outputs, so the external microcontroller can wake-up the transceiver (switch to normal operating mode) via pins $\overline{STB}$ and EN.

To prevent false wake-up due to transients or RF fields, the wake-up voltage levels have to be maintained for a certain period of time. In the low power modes the failure detection circuit remains partly active to prevent an increased power consumption in the event of failures 3, 3a, 4 and 7.

Pin INH is set to floating only during the goto-sleep command and stays floating during the Sleep mode. If pin INH is set to floating, pin INH will not be set to HIGH-level again just by a mode change to normal operating mode. Pin INH will be set to HIGH-level by the following events only:

- power-on (V_{BAT} switching-on at cold start)
- rising or falling edge on pin $\overline{WAKE}$
- a message with 5 consecutive dominant bits during pin EN or pin $\overline{STB}$ is at LOW-level.

The signals on pins $\overline{STB}$ and EN will internally be set to LOW-level when V_{CC} is below a certain threshold voltage so providing fail safe functionality.

Table 2 Normal operating and low power modes

MODE	$\overline{STB}$	EN	$\overline{ERR}$		RXD		RTL SWITCHED TO
			LOW	HIGH	LOW	HIGH	
Goto-sleep command	0	1	wake-up interrupt signal; notes 2 and 3		wake-up interrupt signal; notes 2 and 3		V_{BAT}
Sleep	0	0[1]					V_{BAT}
Standby	0	0					V_{BAT}
Power-on standby	1	0	V_{BAT} power-on flag; notes 2 and 4		wake-up interrupt signal; notes 2 and 3		V_{BAT}
Normal operating	1	1	error flag	no error flag	dominant received data	recessive received data	V_{CC}

Notes

1. In case the goto-sleep command was used before. When V_{CC} drops pin EN will become LOW, but this does not effect the internal functions due to the fail safe functionality.
2. If the supply voltage V_{CC} is present.
3. Wake-up interrupts are released when entering the normal operating mode.
4. V_{BAT} power-on flag will be reset when entering the normal operating mode.

Power-on

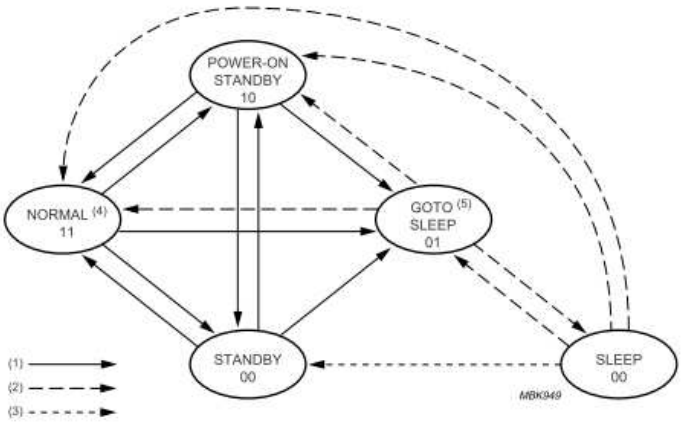
After power-on (V_{BAT} switched on) the signal on pin INH will become HIGH and an internal power-on flag will be set. This flag can be read in the power-on standby mode via pin $\overline{ERR}$ ($\overline{STB}$ = 1; EN = 0) and will be reset by entering the normal operating mode.

Protections

A current limiting circuit protects the transmitter output stages against short-circuit to positive and negative battery voltage.

If the junction temperature exceeds a maximum value, the transmitter output stages are disabled. Because the transmitter is responsible for the major part of the power dissipation, this will result in a reduced power dissipation and hence a lower chip temperature. All other parts of the IC will remain operating.

The pins CANH and CANL are protected against electrical transients which may occur in an automotive environment.



(1) Mode change via input ports $\overline{STB}$ and EN.
(2) Mode change via input ports $\overline{STB}$ and EN, but in the sleep mode INH is inactive and possibly there is no V_{CC}. Mode control is only possible if V_{CC} of the transceiver is active.
(3) INH is activated after wake-up via bus or input port $\overline{WAKE}$.
(4) Transitions to normal mode clear the internal wake-up: interrupt and battery fail flag are cleared.
(5) Transitions to sleep mode: INH is deactivated.

Fig.3 Mode control.

LIMITING VALUES

In accordance with the Absolute Maximum Rating System (IEC 134); note 1.

SYMBOL	PARAMETER	CONDITIONS	MIN.	MAX.	UNIT
V_{CC}	supply voltage on pin V_{CC}		−0.3	+6	V
V_{BAT}	battery voltage on pin BAT		−0.3	+40	V
V_n	DC voltage on pins 2 to 6		−0.3	V_{CC} + 0.3	V
V_{CANH}	DC voltage on pin CANH		−40	+40	V
V_{CANL}	DC voltage on pin CANL		−40	+40	V
$V_{trt(n)}$	transient voltage on pins CANH and CANL	see Fig.6	−150	+100	V
V_{WAKE}	DC input voltage on pin $\overline{WAKE}$		–	V_{BAT} + 0.3	V
I_{WAKE}	DC input current on pin $\overline{WAKE}$		−15	–	mA
V_{INH}	DC output voltage on pin INH		−0.3	V_{BAT} + 0.3	V
V_{RTH}	DC voltage on pin RTH		−0.3	V_{BAT} + 1.2	V
V_{RTL}	DC voltage on pin RTL		−0.3	V_{BAT} + 1.2	V
R_{RTH}	termination resistance on pin RTH		500	16000	Ω
R_{RTL}	termination resistance on pin RTL		500	16000	Ω
T_{vj}	virtual junction temperature	note 2	−40	+150	°C
T_{stg}	storage temperature		−55	+150	°C
V_{esd}	electrostatic discharge voltage	human body model; note 3	−2.0	+2.0	kV
		machine model; note 4	−200	+200	V

Notes

1. All voltages are defined with respect to pin GND. Positive current flows into the IC.
2. Junction temperature in accordance with *"IEC 747-1"*. An alternative definition is: $T_{vj} = T_{amb} + P \times R_{th(vj-a)}$ where $R_{th(vj-a)}$ is a fixed value to be used for the calculation of T_{vj}. The rating for T_{vj} limits the allowable combinations of power dissipation (P) and operating ambient temperature (T_{amb}).
3. Equivalent to discharging a 100 pF capacitor through a 1.5 kΩ resistor.
4. Equivalent to discharging a 200 pF capacitor through a 10 Ω resistor and a 0.75 µH coil.

THERMAL CHARACTERISTICS

SYMBOL	PARAMETER	CONDITIONS	VALUE	UNIT
$R_{th(vj-a)}$	thermal resistance from junction to ambient	in free air	120	K/W

QUALITY SPECIFICATION

Quality specification in accordance with *"SNW-FQ-611-Part-E"*.

DC CHARACTERISTICS

V_{CC} = 4.75 to 5.25 V; V_{BAT} = 5 to 27 V; V_{STB} = V_{CC}; T_{amb} = −40 to +125 °C; unless otherwise specified. All voltages are defined with respect to ground. Positive currents flow into the IC. All parameters are guaranteed over the temperature range by design, but only 100% tested at 25 °C.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Supplies						
I_{CC}	supply current	normal operating mode; V_{TXD} = V_{CC} (recessive)	4	7	11	mA
		normal operating mode; V_{TXD} = 0 V (dominant); no load	11	17	27	mA
		low power modes; V_{TXD} = V_{CC}	0	0	10	µA
I_{BAT}	battery current on pin BAT	all modes; in low power modes at V_{RTL} = V_{BAT} or V_{RTL} < 2.5 V (>1.5 ms)				
		V_{BAT} = V_{WAKE} = V_{INH} = 12 V	10	30	50	µA
		V_{BAT} = V_{WAKE} = V_{INH} = 5 to 27 V	5	30	125	µA
		V_{BAT} = V_{WAKE} = V_{INH} = 3.5 V	5	20	30	µA
		V_{BAT} = V_{WAKE} = V_{INH} = 1 V	0	0	10	µA
I_{CC} + I_{BAT}	supply current plus battery current	low power modes; V_{CC} = 5 V; V_{BAT} = V_{WAKE} = V_{INH} = 12 V	–	35	60	µA
V_{BAT}	battery voltage on pin BAT	low power modes				
		for setting power-on flag	–	–	1	V
		for not setting power-on flag	3.5	–	–	V
Pins $\overline{\text{STB}}$, EN and TXD						
V_{IH}	HIGH-level input voltage		0.7V_{CC}	–	V_{CC} + 0.3	V
V_{IL}	LOW-level input voltage		−0.3	–	0.3V_{CC}	V
I_{IH}	HIGH-level input current	V_I = 4 V				
	pins $\overline{\text{STB}}$ and EN		–	9	20	µA
	pin TXD		−25	−80	−200	µA
I_{IL}	LOW-level input current	V_I = 1 V				
	pins $\overline{\text{STB}}$ and EN		4	8	–	µA
	pin TXD		−100	−320	−800	µA
V_{CC}	supply voltage	for forced power-on standby mode (fail safe)	2.75	–	4.5	V
Pins RXD and $\overline{\text{ERR}}$						
V_{OH}	HIGH-level output voltage					
	on pin $\overline{\text{ERR}}$	I_O = −100 µA	V_{CC} − 0.9	–	V_{CC}	V
	on pin RXD	I_O = −1 mA	V_{CC} − 0.9	–	V_{CC}	V
V_{OL}	LOW-level output voltage on pins $\overline{\text{ERR}}$ and RXD	I_O = 1.6 mA	0	–	0.4	V
		I_O = 7.5 mA	0	–	1.5	V

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Pin $\overline{WAKE}$						
I_{IL}	LOW-level input current	V_{WAKE} = 0 V; V_{BAT} = 27 V	–1	–4	–10	µA
$V_{th(WAKE)}$	wake-up threshold voltage	V_{STB} = 0 V	2.5	3.2	3.9	V
Pin INH						
ΔV_H	HIGH-level voltage drop	I_{INH} = –0.18 mA	–	–	0.8	V
$\lvert I_L \rvert$	leakage current	Sleep mode; V_{INH} = 0 V	–	–	5	µA
Pins CANH and CANL						
V_{diff}	differential receiver threshold voltage	no failures and bus failures 1, 2, 5, 6a; see Fig.4				
		V_{CC} = 5 V	–3.5	–3.2	–2.9	V
		V_{CC} = 4.75 to 5.25 V	$-0.70V_{CC}$	$-0.64V_{CC}$	$-0.58V_{CC}$	V
$V_{O(reces)}$	recessive output voltage	V_{TXD} = V_{CC}				
	on pin CANH	R_{RTH} < 4 kΩ	–	–	0.2	V
	on pin CANL	R_{RTL} < 4 kΩ	V_{CC} – 0.2	–	–	V
$V_{O(dom)}$	dominant output voltage	V_{TXD} = 0 V; V_{EN} = V_{CC}				
	on pin CANH	I_{CANH} = –40 mA	V_{CC} – 1.4	–	–	V
	on pin CANL	I_{CANL} = 40 mA	–	–	1.4	V
$I_{O(CANH)}$	output current on pin CANH	normal operating mode; V_{CANH} = 0 V; V_{TXD} = 0 V	–45	–80	–110	mA
		low power modes; V_{CANH} = 0 V; V_{CC} = 5 V	–	–0.25	–	µA
$I_{O(CANL)}$	output current on pin CANL	normal operating mode; V_{CANL} = 14 V; V_{TXD} = 0 V	45	70	100	mA
		low power modes; V_{CANL} = 12 V; V_{BAT} = 12 V	–	0	–	µA
$V_{det(CANH)}$	detection threshold voltage for short-circuit to battery voltage on pin CANH	normal operating mode	1.5	1.7	1.85	V
		low power modes	1.1	1.8	2.5	V
$V_{det(CANL)}$	detection threshold voltage for short-circuit to battery voltage on pin CANL	normal operating mode	6.5	7.3	8	V
$V_{th(wake)}$	wake-up threshold voltage					
	on pin CANL	low power modes	2.5	3.2	3.9	V
	on pin CANH	low power modes	1.1	1.8	2.5	V
$\Delta V_{th(wake)}$	difference of wake-up threshold voltages	low power modes	0.8	1.4	–	V
$V_{se(CANH)}$	single-ended receiver threshold voltage on pin CANH	normal operating mode and failures 4, 6 and 7				
		V_{CC} = 5 V	1.5	1.7	1.85	V
		V_{CC} = 4.75 to 5.25 V	$0.30V_{CC}$	$0.34V_{CC}$	$0.37V_{CC}$	V

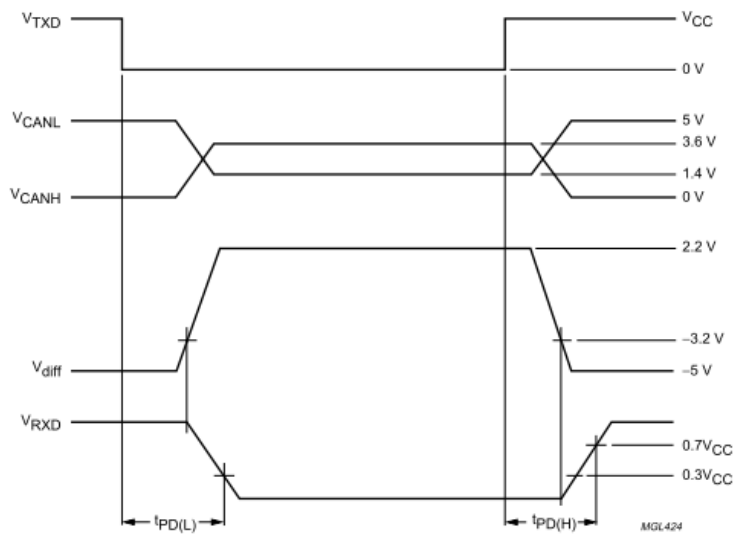
SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
$V_{se(CANL)}$	single-ended receiver threshold voltage on pin CANL	normal operating mode and failures 3 and 3a				
		V_{CC} = 5 V	3.15	3.3	3.45	V
		V_{CC} = 4.75 to 5.25 V	$0.63V_{CC}$	$0.66V_{CC}$	$0.69V_{CC}$	V
Pins RTH and RTL						
$R_{sw(RTL)}$	switch-on resistance between pin RTL and V_{CC}	normal operating mode; $\|I_O\|$ < 10 mA	–	50	100	Ω
$R_{sw(RTH)}$	switch-on resistance between pin RTH and ground	normal operating mode; $\|I_O\|$ < 10 mA	–	50	100	Ω
$V_{O(RTH)}$	output voltage on pin RTH	low power modes; I_O = 1 mA	–	0.7	1.0	V
$I_{O(RTL)}$	output current on pin RTL	low power modes; V_{RTL} = 0 V	–1.25	–0.65	–0.3	mA
$I_{pu(RTL)}$	pull-up current on pin RTL	normal operating mode and failures 4, 6 and 7	–	75	–	µA
$I_{pd(RTH)}$	pull-down current on pin RTH	normal operating mode and failures 3 and 3a	–	75	–	µA
Thermal shutdown						
T_j	junction temperature	for shutdown	155	165	180	°C

TIMING CHARACTERISTICS

V_{CC} = 4.75 to 5.25 V; V_{BAT} = 5 to 27 V; V_{STB} = V_{CC}; T_{amb} = –40 to +125 °C; unless otherwise specified. All voltages are defined with respect to ground. Positive currents flow into the IC. All parameters are guaranteed over the temperature range by design, but only 100% tested at 25 °C.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
$t_{t(r-d)}$	CANL and CANH output transition time for recessive-to-dominant	10 to 90%; C1 = 10 nF; C2 = 0; R1 = 100 Ω; see Fig.5	0.35	0.60	–	µs
$t_{t(d-r)}$	CANL and CANH output transition time for dominant-to-recessive	10 to 90%; C1 = 1 nF; C2 = 0; R1 = 100 Ω; see Fig.5	0.2	0.3	–	µs
$t_{PD(L)}$	propagation delay TXD to RXD (LOW)	no failures and failures 1, 2, 5, 6a; see Figs 4 and 5				
		C1 = 1 nF; C2 = 0; R1 = 100 Ω	–	0.75	1.35	µs
		C1 = C2 = 3.3 nF; R1 = 100 Ω	–	1	1.75	µs
		failures 3, 3a, 4, 6 and 7; see Figs 4 and 5				
		C1 = 1 nF; C2 = 0; R1 = 100 Ω	–	0.85	1.4	µs
		C1 = C2 = 3.3 nF; R1 = 100 Ω	–	1.1	1.7	µs
$t_{PD(H)}$	propagation delay TXD to RXD (HIGH)	no failures and failures 1, 2, 5, 6a; see Figs 4 and 5				
		C1 = 1 nF; C2 = 0; R1 = 100 Ω	–	1.2	1.9	µs
		C1 = C2 = 3.3 nF; R1 = 100 Ω	–	2.5	3.3	µs
		failures 3, 3a, 4, 6 and 7; see Figs 4 and 5				
		C1 = 1 nF; C2 = 0; R1 = 100 Ω	–	1.1	1.7	µs
		C1 = C2 = 3.3 nF; R1 = 100 Ω	–	1.5	2.2	µs
$t_{CANH(min)}$	minimum dominant time for wake-up on pin CANH	low power modes; V_{BAT} = 12 V	7	–	38	µs
$t_{CANL(min)}$	minimum dominant time for wake-up on pin CANL	low power modes; V_{BAT} = 12 V	7	–	38	µs
$t_{WAKE(min)}$	minimum time on pin $\overline{WAKE}$	low power modes; V_{BAT} = 12 V; for wake-up after receiving a falling or rising edge	7	–	38	µs
t_{det}	failure detection time	normal mode				
		failure 3 and 3a	1.6	–	8.0	ms
		failure 4, 6 and 7	0.3	–	1.6	ms
		low power modes; V_{BAT} = 12 V				
		failure 3 and 3a	1.6	–	8.0	ms
		failure 4 and 7	0.1	–	1.6	ms

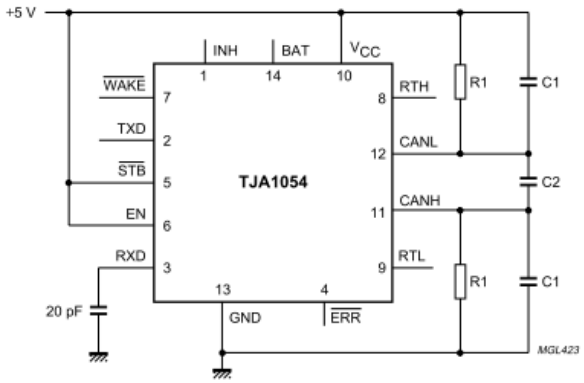
SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
t_{rec}	failure recovery time	normal mode				
		failure 3 and 3a	0.3	–	1.6	ms
		failure 4 and 7	7	–	38	µs
		failure 6	125	–	750	µs
		low power modes; V_{BAT} = 12 V				
		failures 3, 3a, 4 and 7	0.3	–	1.6	ms
$t_{h(min)}$	minimum hold time of goto-sleep command		5	–	50	µs
$t_{dis(TXD)}$	disable time of TXD permanent dominant timer	normal mode; V_{TXD} = 0 V	0.75	–	4	ms
Δpc	pulse-count difference between CANH and CANL	normal mode and failures 1, 2, 5 and 6a				
		failure detection (pin $\overline{ERR}$ becomes LOW)	–	4	–	
		failure recovery	–	4	–	



$V_{diff} = V_{CANH} - V_{CANL}$.

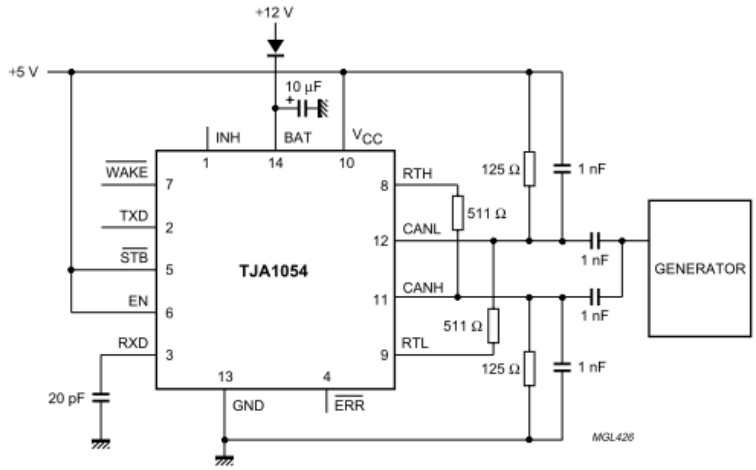
Fig.4 Timing diagram for dynamic characteristics.

TEST AND APPLICATION INFORMATION



For testing, the 100 Ω termination resistors are not connected to RTH or RTL because minimum 500 Ω per transceiver is allowed.

Fig.5 Test circuit for dynamic characteristics.



The waveforms of the applied transients will be in accordance with ISO 7637 part 1, test pulses 1, 2, 3a and 3b.

Fig.6 Test circuit for automotive transients.

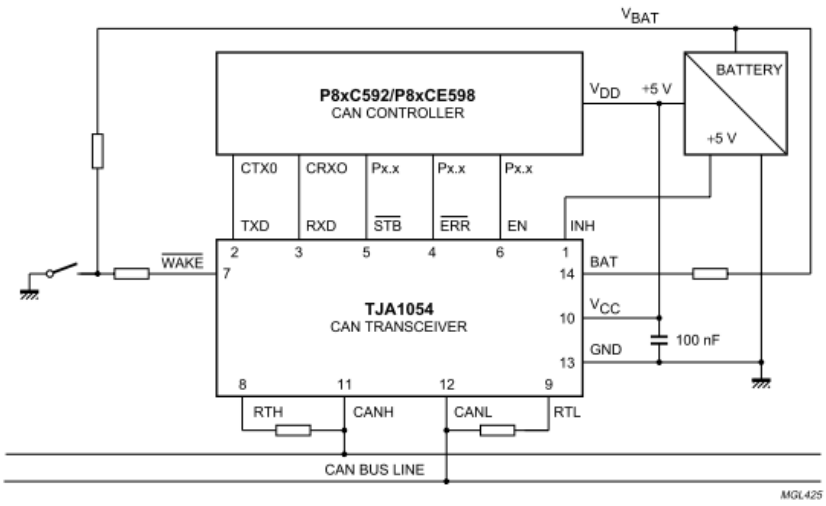
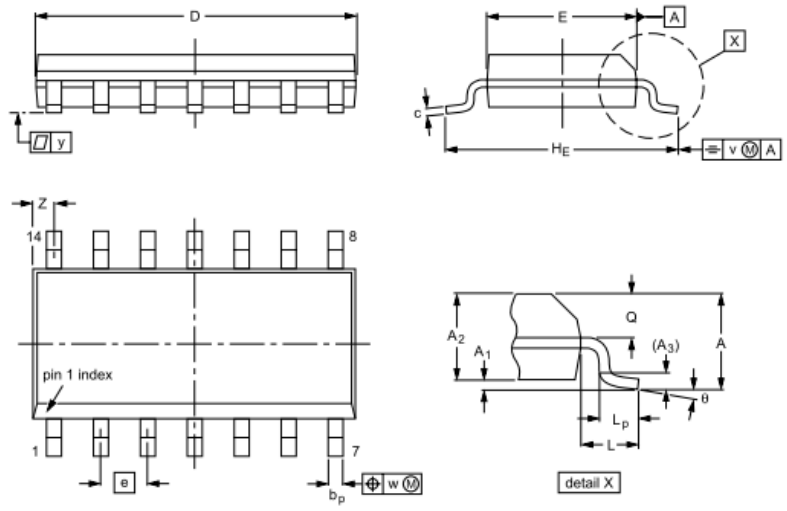
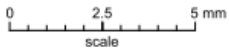


Fig.7 Application diagram.

PACKAGE OUTLINE

SO14: plastic small outline package; 14 leads; body width 3.9 mm **SOT108-1**





DIMENSIONS (inch dimensions are derived from the original mm dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(1)}$	e	H_E	L	L_p	Q	v	w	y	$Z^{(1)}$	θ
mm	1.75	0.25 0.10	1.45 1.25	0.25	0.49 0.36	0.25 0.19	8.75 8.55	4.0 3.8	1.27	6.2 5.8	1.05	1.0 0.4	0.7 0.6	0.25	0.25	0.1	0.7 0.3	8° 0°
inches	0.069	0.010 0.004	0.057 0.049	0.01	0.019 0.014	0.0100 0.0075	0.35 0.34	0.16 0.15	0.050	0.244 0.228	0.041	0.039 0.016	0.028 0.024	0.01	0.01	0.004	0.028 0.012	

Note

1. Plastic or metal protrusions of 0.15 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	EIAJ			
SOT108-1	076E06S	MS-012AB				~~95-01-23~~ 97-05-22

SOLDERING

Introduction to soldering surface mount packages

This text gives a very brief insight to a complex technology. A more in-depth account of soldering ICs can be found in our *"Data Handbook IC26; Integrated Circuit Packages"* (document order number 9398 652 90011).

There is no soldering method that is ideal for all surface mount IC packages. Wave soldering is not always suitable for surface mount ICs, or for printed-circuit boards with high population densities. In these situations reflow soldering is often used.

Reflow soldering

Reflow soldering requires solder paste (a suspension of fine solder particles, flux and binding agent) to be applied to the printed-circuit board by screen printing, stencilling or pressure-syringe dispensing before package placement.

Several methods exist for reflowing; for example, infrared/convection heating in a conveyor type oven. Throughput times (preheating, soldering and cooling) vary between 100 and 200 seconds depending on heating method.

Typical reflow peak temperatures range from 215 to 250 °C. The top-surface temperature of the packages should preferable be kept below 230 °C.

Wave soldering

Conventional single wave soldering is not recommended for surface mount devices (SMDs) or printed-circuit boards with a high component density, as solder bridging and non-wetting can present major problems.

To overcome these problems the double-wave soldering method was specifically developed.

If wave soldering is used the following conditions must be observed for optimal results:

- Use a double-wave soldering method comprising a turbulent wave with high upward pressure followed by a smooth laminar wave.
- For packages with leads on two sides and a pitch (e):
  - larger than or equal to 1.27 mm, the footprint longitudinal axis is **preferred** to be parallel to the transport direction of the printed-circuit board;
  - smaller than 1.27 mm, the footprint longitudinal axis **must** be parallel to the transport direction of the printed-circuit board.

  The footprint must incorporate solder thieves at the downstream end.
- For packages with leads on four sides, the footprint must be placed at a 45° angle to the transport direction of the printed-circuit board. The footprint must incorporate solder thieves downstream and at the side corners.

During placement and before soldering, the package must be fixed with a droplet of adhesive. The adhesive can be applied by screen printing, pin transfer or syringe dispensing. The package can be soldered after the adhesive is cured.

Typical dwell time is 4 seconds at 250 °C.
A mildly-activated flux will eliminate the need for removal of corrosive residues in most applications.

Manual soldering

Fix the component by first soldering two diagonally-opposite end leads. Use a low voltage (24 V or less) soldering iron applied to the flat part of the lead. Contact time must be limited to 10 seconds at up to 300 °C.

When using a dedicated tool, all other leads can be soldered in one operation within 2 to 5 seconds between 270 and 320 °C.

Suitability of surface mount IC packages for wave and reflow soldering methods

PACKAGE	SOLDERING METHOD	
	WAVE	REFLOW(1)
BGA, SQFP	not suitable	suitable
HLQFP, HSQFP, HSOP, HTSSOP, SMS	not suitable(2)	suitable
PLCC(3), SO, SOJ	suitable	suitable
LQFP, QFP, TQFP	not recommended(3)(4)	suitable
SSOP, TSSOP, VSO	not recommended(5)	suitable

Notes

1. All surface mount (SMD) packages are moisture sensitive. Depending upon the moisture content, the maximum temperature (with respect to time) and body size of the package, there is a risk that internal or external package cracks may occur due to vaporization of the moisture in them (the so called popcorn effect). For details, refer to the Drypack information in the *"Data Handbook IC26; Integrated Circuit Packages; Section: Packing Methods"*.
2. These packages are not suitable for wave soldering as a solder joint between the printed-circuit board and heatsink (at bottom version) can not be achieved, and as solder may stick to the heatsink (on top version).
3. If wave soldering is considered, then the package must be placed at a 45° angle to the solder wave direction. The package footprint must incorporate solder thieves downstream and at the side corners.
4. Wave soldering is only suitable for LQFP, TQFP and QFP packages with a pitch (e) equal to or larger than 0.8 mm; it is definitely not suitable for packages with a pitch (e) equal to or smaller than 0.65 mm.
5. Wave soldering is only suitable for SSOP and TSSOP packages with a pitch (e) equal to or larger than 0.65 mm; it is definitely not suitable for packages with a pitch (e) equal to or smaller than 0.5 mm.

DEFINITIONS

Data sheet status	
Objective specification	This data sheet contains target or goal specifications for product development.
Preliminary specification	This data sheet contains preliminary data; supplementary data may be published later.
Product specification	This data sheet contains final product specifications.
Limiting values	
Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.	
Application information	
Where application information is given, it is advisory and does not form part of the specification.	

LIFE SUPPORT APPLICATIONS

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips for any damages resulting from such improper use or sale.

Annexe 5 Sequences de Test CAPL

Test Alim.can :

```
includes
{

}

variables
{

}




void MainTest ()
{
sysvar::VTS::M1_VT7001.SetInterconnectionMode(0);
Voltage3Volts();
Voltage6Volts();
Voltage12Volts();
}




testcase Voltage3Volts()
{
sysvar::VTS::M1_SupInt.SetRefVoltageMode(1);
@sysvar::VTS::M1_SupInt::RefVoltage = 03;
@sysvar::VTS::M1_Out1::Active = 1;
sysvar::VTS::M1_SupInt.StartStimulation();
testwaitfortimeout(3000);
}




testcase Voltage6Volts()
{
sysvar::VTS::M1_SupInt.SetRefVoltageMode(1);
@sysvar::VTS::M1_SupInt::RefVoltage = 06;
@sysvar::VTS::M1_Out1::Active = 1;
sysvar::VTS::M1_SupInt.StartStimulation();
```

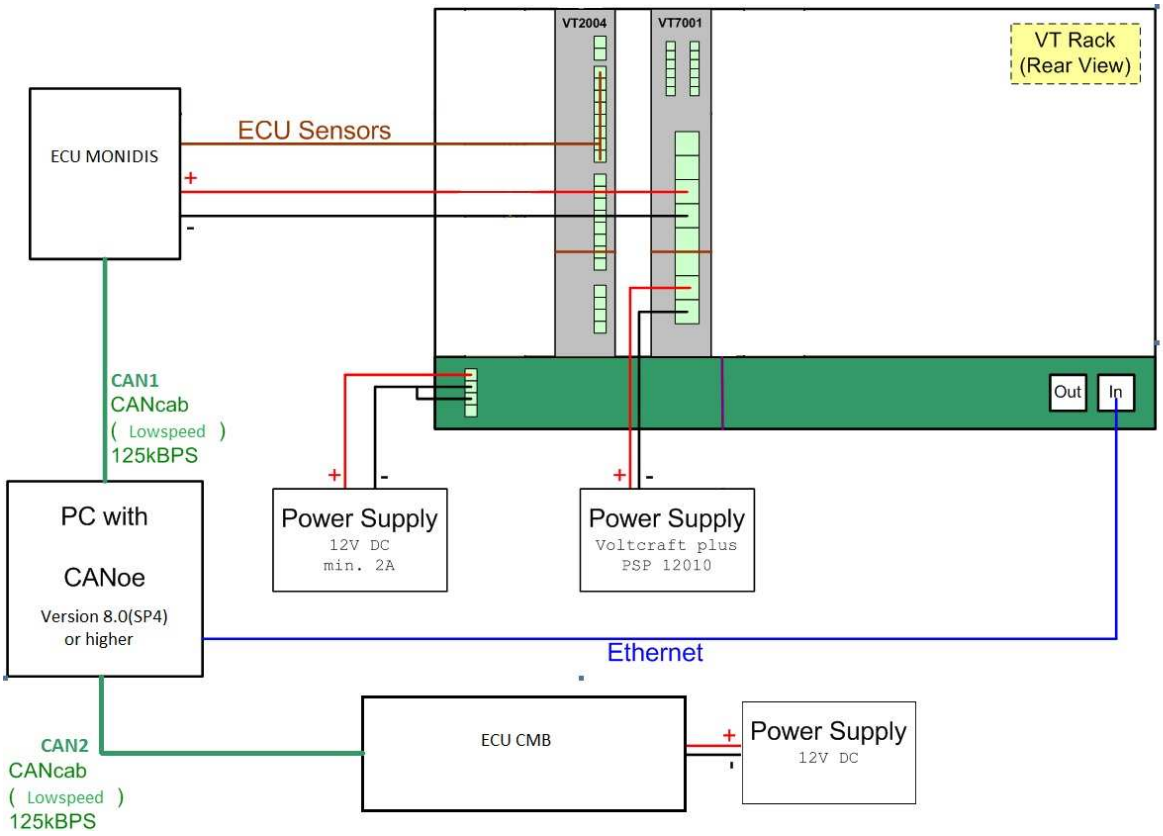
```
testwaitfortimeout(3000);
}

testcase Voltage12Volts()
{
sysvar::VTS::M1_SupInt.SetRefVoltageMode(1);
@sysvar::VTS::M1_SupInt::RefVoltage = 12;
@sysvar::VTS::M1_Out1::Active = 1;
sysvar::VTS::M1_SupInt.StartStimulation();
testwaitfortimeout(3000);
}



Test OUTWF.can

includes
{

}

variables
{

}

void MainTest ()
{
sysvar::VTS::M1_VT7001.SetInterconnectionMode(0);
Waveform();

}


testcase Waveform()
{
  // Choose voltage stimulation and waveform curve type
 sysvar::VTS::M1_SupInt.SetRefVoltageMode(2);

  // Load waveform (the contents of WaveForm.txt are listed below)

sysvar::VTS::M1_SupInt.LoadWFVoltage("C:\\Users\\vfpel\\Desktop\\CNAM_MEM\\MONIDIS_demo
\\CONFIG\\WaveForm.txt");

  // Configure waveform. Parameters:
  // TimeIncrement (time to hold each sample)      = 65ms
  // Pause (pause between two waveform repetitions) = 1s
  // NumberOfRepeats (number of repetitions)       = 0 (unlimited)
  sysvar::VTS::M1_SupInt.SetWFParams(0.065, 1.0, 0);

  // Output the configured waveform for 10 seconds
  @sysvar::VTS::M1_Out1::Active = 1;
```

```
  sysvar::VTS::M1_SupInt.StartStimulation();
  TestWaitForTimeout(40000);
  sysvar::VTS::M1_SupInt.StopStimulation();

}
```

TesterConfirm.can

```
includes
{

}

variables
{
char phrase[50];
int state;
float x=0;
mstimer test;
}

testfunction TesterCall(char phrase[],int state)
{
openPanel("testerPanel");
x=0;
SysSetVariableString(sysvar::actsentence::actsentence,phrase);
@sysvar::testswitch::testswitch=state;


}

on sysvar_update sysvar::testswitch::testswitch
{
setTimer(test,500);
@sysvar::testtime::testtime=0;
}



on timer test
{
setTimer(test,500);
x=x+0.5;
@sysvar::testtime::testtime=x;
write("%f",x);

}
```

Annexe6 Schéma synoptique du banc de test



Annexe7 Brochage du connecteur du MONIDIS

CONNECTEUR 10v	SIGNAL
1	+CAN
2	MASSE
3	/
4	CAN-H (1)
5	CAN-L (1)
6	BUZZER (2)
7	RETROECLAIRAGE (2)
8	/
9	/
10	/

(1) CAN-LS 125kb/s

(2) BUZZER et RETROECLAIRAGE actifs sur +12v

Références :

[1] Embedded system (*http://dictionary.reference.com/*)

[2] CIA http://www.can-cia.org

[3] Documents internes Vector:

AN-AND-1-110_Quick_Introduction_to_CANalyzer

AN-AND-1-112_Canoe_Tutorial

AN-AND-1-113_Quick_Introduction_to_CAPL

AN-AND-1-118_CANoe_TFS_Tutorial

AN-AND-1-901_Using_CANoe_for_vehicle_development

AN-IND-1-002_Testing_with_CANoe

AN-ANI-1-115_HS_Physical_Layer_Problems

[4] Philips TJA1054 Fault-Tolerant CAN Transceiver Datasheet

[5] CANoe : National Instruments devices with CANoe

- liste des figures :

- liste des tableaux :

Banc de développement et de validation d'un système embarqué automobile

Mémoire d'ingénieur C.N.A.M., Paris-2013

Depuis plus de quinze ans, le modèle de l'architecture automobile a évolué de manière significative. Avec l'arrivée des bus véhicule comme le CAN, la façon de concevoir, et d'organiser le réseau électrique a progressé vers une diminution des connexions filaires grâce au multiplexage, et à la multiplication des prestations électroniques et des calculateurs les gérant. Le bus CAN permet de transporter les informations de commandes sur une même ligne, là où les systèmes anciens nécessitaient une ligne dédiée par commande, réduisant le poids et le coût des fils cuivrés. A l'inverse, le nombre grandissant des calculateurs a amené, outre une augmentation du poids et de l'encombrement véhicule, des problématiques d'interfaçage entre les différents systèmes, en provenance de différents fournisseurs. L'adaptation d'un système embarqué dans un ensemble plus vaste met en évidence ces défis d'intégration, qui nécessitent le développement de standards de communication, et la création d'environnements simulés afin de recréer les conditions de validation avant l'intégration réelle. Les tests des interfaces de communication, des entrées et sorties analogiques et de l'applicatif programmé constituent le plan de validation d'un calculateur. Dans cette optique, les industriels automobiles doivent s'appuyer sur les sciences électroniques et informatiques pour le développement et la validation de ces systèmes embarqués.

Mots clés : CAN - Systèmes embarqués – Développement – Simulation – Validation – Calculateur – Automobile.

Development and validation bench of an automotive embedded system

For over fifteen years, the model of the vehicle architecture has changed significantly. With the arrival of the vehicle bus such as CAN, the way to design and organize the electrical network evolved, to a decrease of wired connections due to multiplexing, and the proliferation of electronic services and calculators which are managing them. The CAN bus allows to carry the command information on the same line where the old systems required a dedicated line by command, reducing the weight and cost regarding copper wires. Unlike, the growing number of calculators has led, in addition to increased weight and vehicle overcrowding, problems interfacing between different systems from different providers. The adaptation of an embedded system in a larger whole system highlights these integration challenges, which require the development of communication standards, and create simulated environments to recreate the conditions of validation before the real integration. Tests of communication interfaces, analog inputs and outputs and the programmed application make up the validation plan of a calculator. In this context, automotive manufacturers must rely on electronic and computer sciences for the development and validation of these embedded systems.

Key words: CAN – Embedded systems - Development – Simulation – Validation – Calculator – Automotive.