MATRIX | EBLOCKS 2

CAN Bus Communications

CP2793

MATRIX

www.matrixtsl.com

CP2793

Formation sur le bus CAN

Notes de cours

Contenu

Introduction

Ces notes sont conçues pour vous présenter les concepts nécessaires à la compréhension des réseaux CAN et pour vous fournir des exercices pratiques qui vous permettront de développer vos compétences et celles de vos étudiants.

Le cours est structuré en plusieurs sections qui vous présentent d'abord les bases du CAN, puis des sujets intermédiaires, tels que les messages et l'envoi de données. Le cours aborde également des sujets plus avancés, notamment l'utilisation de masques et de filtres. Des exemples et des suggestions de travail sont fournis pour servir de base à l'élaboration de démonstrations et d'activités pratiques pour vos étudiants.

Ces notes fournissent un cadre pour l'enseignement du RCA aux étudiants. L'utilisation que vous en ferez est laissée à votre discrétion. Si vous enseignez à des étudiants en automobile qui n'ont pas besoin de savoir programmer, vous pouvez simplement utiliser les exemples de programmes téléchargeables.

Ce cours est réalisé à l'aide de Flowcode, un langage de programmation graphique. Le composant CAN de Flowcode est conçu pour permettre aux étudiants de se familiariser avec le CAN sans s'embarrasser des problèmes de programmation en C ou dans un langage de niveau inférieur.

Lorsque nous enseignons le CAN aux étudiants en automobile, nous n'envisageons pas de faire beaucoup de programmation en Flowcode. Cependant, nous suggérons que le superviseur ait une certaine expérience de Flowcode à des fins de débogage. Cette expérience peut être acquise rapidement et facilement.

Les étudiants plus avancés voudront utiliser Flowcode de manière intensive. Le site web de Matrix TSL Flowcode contient un certain nombre de fichiers tutoriels et de ressources que les étudiants peuvent consulter pour les aider à comprendre le fonctionnement de Flowcode.
Les étudiants constateront qu'ils peuvent faire des progrès rapides en utilisant l'interface

graphique de Flowcode. Ce cours est conçu pour être utilisé avec deux niveaux d'étudiants :

1. D'abord pour les techniciens automobiles de niveau 3, afin qu'ils puissent se familiariser avec la technologie CAN et l'équipement utilisé pour rechercher les pannes dans les systèmes CAN, ainsi qu'avec la manière dont cette recherche de pannes est effectuée. Ces techniciens sont censés télécharger et examiner les programmes présentés dans les organigrammes, mais ne sont pas censés effectuer des tâches de programmation.

2. Deuxièmement, pour les étudiants plus avancés du niveau 4, afin qu'ils acquièrent une compréhension de la technologie CAN et qu'ils puissent construire des réseaux qui communiquent avec CAN et des protocoles de niveau supérieur. Ces étudiants sont censés développer leurs propres réseaux CAN à l'aide d'organigrammes et de macros CAN. L'utilisation intensive d'organigrammes permettra aux étudiants de comprendre rapidement et facilement les protocoles et la communication CAN, en évitant d'avoir à s'impliquer dans les processus de construction de logiciels de bus CAN de niveau inférieur.

1 Démonstrations, exemples pratiques et exercices

Les notes contiennent trois types d'exercices : des *démonstrations*, des *exemples travaillés* et des *exemples*. Nous allons décrire brièvement ce que nous entendons par chacun d'entre eux.

Dans le cas des solutions basées sur les E-blocks2, tous les programmes fournis à partir du CP2793 - CAN Bus Resource Files (PIC - https://www.matrixtsl.com/webshop/e-blocks2-can-bus-training-course-pic.html#resources). (Arduino - https://www.matrixtsl.com/webshop/e-blocks2-can-bus-training-course-arduino.html#resources) nécessitent l'installation de Flowcode V10 ou d'une version plus récente sur le PC hôte.

1.1 Démonstrations

Des démonstrations sont fournies qui peuvent être utilisées avec les techniciens pour voir un système CAN en action. Tous les programmes seront fournis afin qu'ils puissent être programmés dans les nœuds. Aucune programmation n'est requise, mais les organigrammes Flowcode sont à la disposition des étudiants pour leur montrer comment ils fonctionnent. Les démonstrations sont utilisées de préférence en conjonction avec CANKing pour permettre aux étudiants de voir la messagerie en action et de noter l'effet des différents programmes sur le trafic du réseau.

Les démonstrations peuvent être utilisées pour enseigner les principes fondamentaux du RCA aux étudiants qui ne sont pas tenus de comprendre et d'appliquer les principes du RCA.
vérifier les systèmes CAN, mais n'auront pas besoin de compétences en programmation.

1.2 Exemples concrets

Un exemple travaillé est fourni pour le premier exemple de base afin de permettre aux étudiants de se familiariser à la fois avec CAN et Flowcode. L'accent est mis ici sur la mise en route d'un système CAN simple en Flowcode qui peut ensuite être utilisé comme base d'expérience pour des exemples ultérieurs.

1.3 Exercices

D'autres exercices sont fournis, ainsi qu'un ensemble de notes d'accompagnement. Ces notes donnent des informations sur la mise en place des différents aspects de la solution et peuvent servir de base à des documents à distribuer.

1.4 Autres travaux

Chaque exercice est accompagné de questions à méditer et de suggestions pour un travail plus approfondi. Ce travail complémentaire peut servir de base à des activités différenciées pour les étudiants, répondant ainsi aux exigences des organismes de certification dans les cas où la solution CAN est utilisée dans le cadre d'un cours formellement évalué.

2 Réseau CAN de base

2.1 Vue d'ensemble

Cette section est conçue pour vous permettre d'utiliser un réseau CAN le plus rapidement possible. Vous serez initié à la messagerie et à la manière d'envoyer et de recevoir un signal simple qui peut être pris en compte. Les exemples d'applications vous présenteront plusieurs fonctions CAN de base et serviront de point de départ à des études plus approfondies.

2.2 Qu'est-ce que le RCA ?

CAN - Controller Area Network (réseau de contrôleurs) est un protocole de réseau sériel. Il s'agit d'une méthode prédéfinie de communication entre les différentes parties d'un système. Chaque partie doit parler le même langage et utiliser un ensemble commun de signaux et de structures de messages afin de pouvoir comprendre les messages et d'être compris à son tour. Le CAN est l'un de ces systèmes.

D'autres systèmes, tels que RS-232, sont souvent des systèmes point à point dans lesquels un appareil communique directement avec un autre. L'une des limites de ce système est qu'il peut être nécessaire d'utiliser plusieurs connexions différentes pour communiquer avec différentes parties du système, ou qu'une partie peut avoir besoin de communiquer avec une autre, mais ne peut le faire que par le biais d'un intermédiaire.

Le CAN offre une solution simple à ce problème. Il envoie le message à toutes les parties du système et laisse chaque partie (ou *nœud*) décider elle-même si le message lui est destiné ou non. Le *contrôle d'erreur* intégré et les réponses aux messages permettent d'éviter les messages perdus ou les systèmes *bloqués* qui *attendent* une réponse. En outre, les messages peuvent être traités par plusieurs dispositifs, voire aucun, ce qui facilite la construction d'un système qui ne réagit qu'à l'endroit et au moment où il en a besoin.

CAN dispose de plusieurs systèmes intégrés de *détection des erreurs* et de moyens d'empêcher tous les nœuds d'essayer de parler en même temps. Mais vous ne voyez jamais cela, c'est la puce CAN qui s'en charge dans les coulisses. Tout ce que vous avez à faire, c'est de décider des messages à envoyer et à recevoir.

Un autre avantage du système CAN est que si vous souhaitez ajouter un autre élément au système, il suffit souvent de le programmer pour qu'il réponde aux messages appropriés et de le connecter au réseau. Il n'est pas nécessaire de le connecter à un endroit ou dans un ordre précis, de sorte que vous pouvez insérer le nouveau nœud là où vous le souhaitez, ou là où la conception physique du système l'exige.

Le CAN a été conçu à l'origine par Bosch pour l'industrie automobile, à partir d'un besoin de communication entre les différentes unités de contrôle électronique (ECU) des voitures de luxe. Depuis, le CAN est devenu un système de réseau populaire, en particulier dans les systèmes embarqués. Le CAN est utilisé sur des véhicules tels que les voitures, les bateaux, les avions, les camions et dans de nombreux autres domaines de l'industrie. La vitesse élevée et la robustesse du CAN le rendent particulièrement adapté aux applications industrielles ou à grande vitesse.

2.3 Nœuds et réseaux

Les systèmes CAN sont constitués de deux ou plusieurs nœuds connectés en *réseau*, comme le montre la figure 2.1.

Le *réseau* est l'autoroute de données à laquelle les nœuds sont connectés. Contrairement à de nombreux autres systèmes où les connexions vont d'un appareil à l'autre, le réseau CAN est autonome. Chaque nœud se nourrit du réseau, mais ne le bloque pas et n'empêche pas les autres nœuds de recevoir le message.

Les *nœuds* peuvent envoyer des messages sur le réseau et écouter le réseau pour voir s'il y a des messages. Le nœud peut alors vérifier le message pour voir s'il doit y répondre ou l'ignorer.

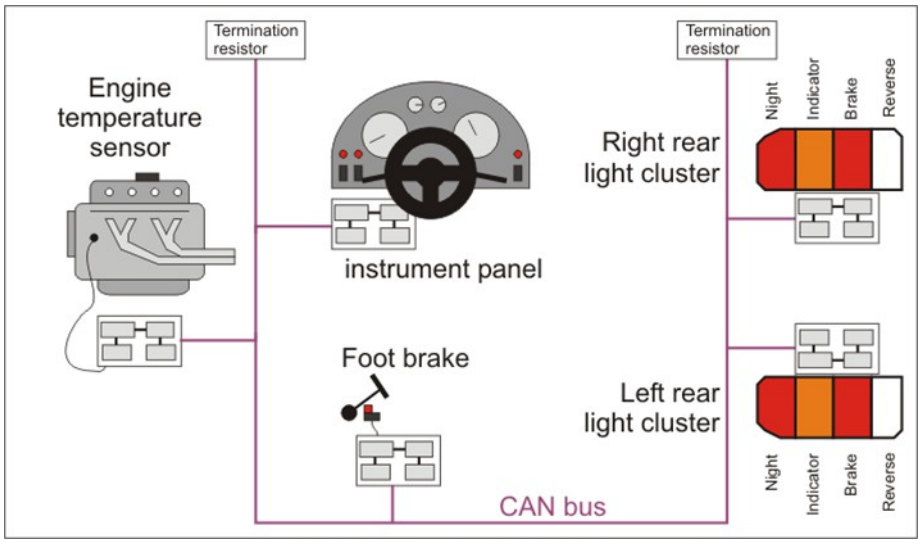


Figure 2.1 *Disposition typique d'un bus CAN automobile*

Les nœuds sont indépendants les uns des autres et peuvent être aussi simples ou complexes que le concepteur du système le souhaite. Un nœud peut être une simple lampe ou un tableau de bord entier. Les nœuds étant indépendants, ils peuvent être ajoutés, supprimés ou modifiés sans qu'il soit nécessaire de changer une autre partie du réseau CAN.

Par exemple, deux modèles de voiture peuvent avoir des tableaux de bord différents, l'un étant un modèle de luxe doté de fonctions supplémentaires que l'on ne trouve pas sur le modèle de base. Les tableaux de bord reçoivent tous les deux exactement les mêmes messages du réseau CAN ; ce sont les messages qu'ils acceptent et ce qu'ils en font qui les différencient. Le réseau CAN ne se préoccupe pas de savoir si un signal devient un simple voyant ou une bande de diodes électroluminescentes. Il ne se préoccupe même pas du fait que le message "Fancy RPM display" soit ignoré lorsque le tableau de bord de base est installé. Le réseau CAN transmet simplement les messages sur le réseau et laisse les nœuds décider de les utiliser ou non. Comme les messages envoyés ne changent pas, le réseau CAN n'a pas besoin d'être modifié pour accepter les différents tableaux de bord, de sorte que l'un ou l'autre peut être intégré dans le réseau.

2.4 La couche physique

La spécification CAN ne précise pas la couche physique de transport des signaux, mais uniquement le format des messages. Ce faisant, le CAN permet aux concepteurs de systèmes de mettre en œuvre une *couche physique* appropriée au système plutôt que d'avoir à adapter le système pour qu'il corresponde à une couche physique prédéfinie.

Il s'agit d'une question très importante, car cela signifie que la manière dont le réseau CAN est mis en œuvre dans la couche physique peut différer et différera souvent d'un système à l'autre. La couche physique d'un système CAN destiné à une usine de l'industrie lourde est susceptible de différer à bien des égards de celle d'un système intégré à une voiture de luxe. Un système CAN tel que la carte Matrix CAN est essentiellement notre implémentation de CAN utilisant notre propre composant Flowcode et notre propre carte CAN pour piloter la couche physique. Certaines parties du système global, telles que le format du message envoyé, font partie intégrante du CAN tel qu'il est défini par la spécification CAN. D'autres, tels que les tampons RX et TX (discutés plus loin) font partie de notre implémentation d'une couche physique pour CAN.

2.5 Messages

Les messages sont le cœur battant du système CAN. Sans eux, le système n'est qu'un amas de fils et de circuits imprimés redondants. Chaque message se compose d'un identifiant (l'ID du message, qui deviendra important plus tard) et d'un flux de données. En fait, il y a plus : des bits d'accusé de réception, des sommes de contrôle, des détails de transmission, etc. mais ces éléments sont traités automatiquement par le composant CAN. Tout ce dont vous devez vous préoccuper, c'est de l'identifiant et des données.

Les ID de message sont utilisés pour différencier les messages. Un nœud peut accepter certains messages et en ignorer d'autres en fonction de leur valeur d'identification. Cela permet de concevoir facilement des systèmes interdépendants complexes dans lesquels plusieurs nœuds peuvent répondre au même message aussi facilement qu'un seul nœud peut choisir un message auquel il est le seul à répondre.

Les messages peuvent contenir des données ou être complètement vides. Pour de nombreux signaux simples, tels qu'un feu de freinage, l'envoi d'un message peut suffire : un signal est envoyé et on y réagit. Pour d'autres signaux, tels que le régime ou la température, des données sont nécessaires et peuvent être transmises avec le message. Si des données sont envoyées, il n'est même pas nécessaire de les examiner. La lecture d'un capteur de vitesse de rotation d'un avion pourrait être utilisée par un nœud pour afficher la vitesse de rotation réelle, mais par un autre nœud pour simplement activer un voyant d'avertissement "moteur en marche" sans même regarder les données.

2.6 Protocoles de niveau supérieur

Le CAN est un *système de messagerie*. Il n'est pas responsable du contenu du message. CAN ne se préoccupe pas de savoir si des données sont attendues ou si une quantité incorrecte de données est envoyée. Il n'est pas chargé de s'assurer que le message est envoyé au nœud approprié dans le système, mais seulement qu'il est correctement envoyé à un nœud. CAN se préoccupe uniquement du fait que le message CAN est correctement formé.

Cependant, nous nous intéressons aux données. Nous nous soucions du *nœud auquel elles sont envoyées*. Nous nous soucions suffisamment de ces choses pour créer des *protocoles de niveau supérieur* afin de traiter ce type de problèmes. Ces protocoles se superposent au CAN et permettent de contrôler le flux de messages et de données sur le réseau. Les protocoles de niveau supérieur, ou HLP, sont utilisés dans les systèmes CAN pour exécuter des fonctions telles que les procédures de démarrage du système, le contrôle des erreurs, la surveillance des connexions et de l'état, ainsi que d'autres tâches administratives.

L'utilisation d'un HLP peut impliquer qu'un nœud CAN doive envoyer des messages pour demander à pouvoir parler à un autre nœud, et que des messages soient envoyés en retour pour accepter la communication avant que la communication réelle ne puisse commencer. De tels systèmes peuvent sembler représenter une surcharge importante lorsque vous apprenez à envoyer des messages CAN, mais une fois que vous aurez compris les messages CAN, votre esprit commencera automatiquement à chercher des moyens de contrôler les erreurs et de surveiller le système. Les HLP sont le résultat de cette progression naturelle.

Les HLP sont la colle qui permet au système de fonctionner correctement. C'est pourquoi les systèmes à grande échelle utiliseront très probablement un HLP. L'un des problèmes que posent les HLP est leur nombre - plus de quarante déjà. Le choix du HLP que vous utiliserez dépendra probablement de la société pour laquelle vous travaillez ou des produits que vous traitez.

Les PLH dépassent le cadre de ce cours. Leur diversité fait qu'il est difficile de les traiter en détail. De plus, la taille et la complexité du code nécessaire pour un HLP sont trop importantes pour être gérées par la plupart des systèmes de microcontrôleurs de base. Toutefois, si vous souhaitez vous intéresser aux HLP et à leur utilisation, vous trouverez de la documentation sur les différents HLP CAN, tels que CANOpen, sur le site de Kvaser www.kvaser.com.

3 Solution de formation CAN

Ce cours est basé sur une solution de formation CAN qui est configurée comme un réseau à quatre nœuds. Cela permet d'obtenir *un*
Un *nœud d'entrée numérique* (interrupteurs), un *nœud de sortie numérique* (feux) et un nœud d'entrée analogique (capteurs), ainsi qu'un nœud de surveillance/contrôle (le tableau de bord). Ces quatre nœuds devraient vous aider à comprendre les tâches du réseau CAN dans un contexte automobile, mais pas seulement. Tous les nœuds ne sont pas nécessaires pour chaque tâche, et pour certaines tâches, vous devrez peut-être reconfigurer certains nœuds. Cependant, pour une formation générale et pour enseigner les principes, le réseau à quatre nœuds est idéal. Un cinquième point de connexion est disponible et est utilisé avec le Kvaser CAN Analyzer pour surveiller le trafic sur le réseau.

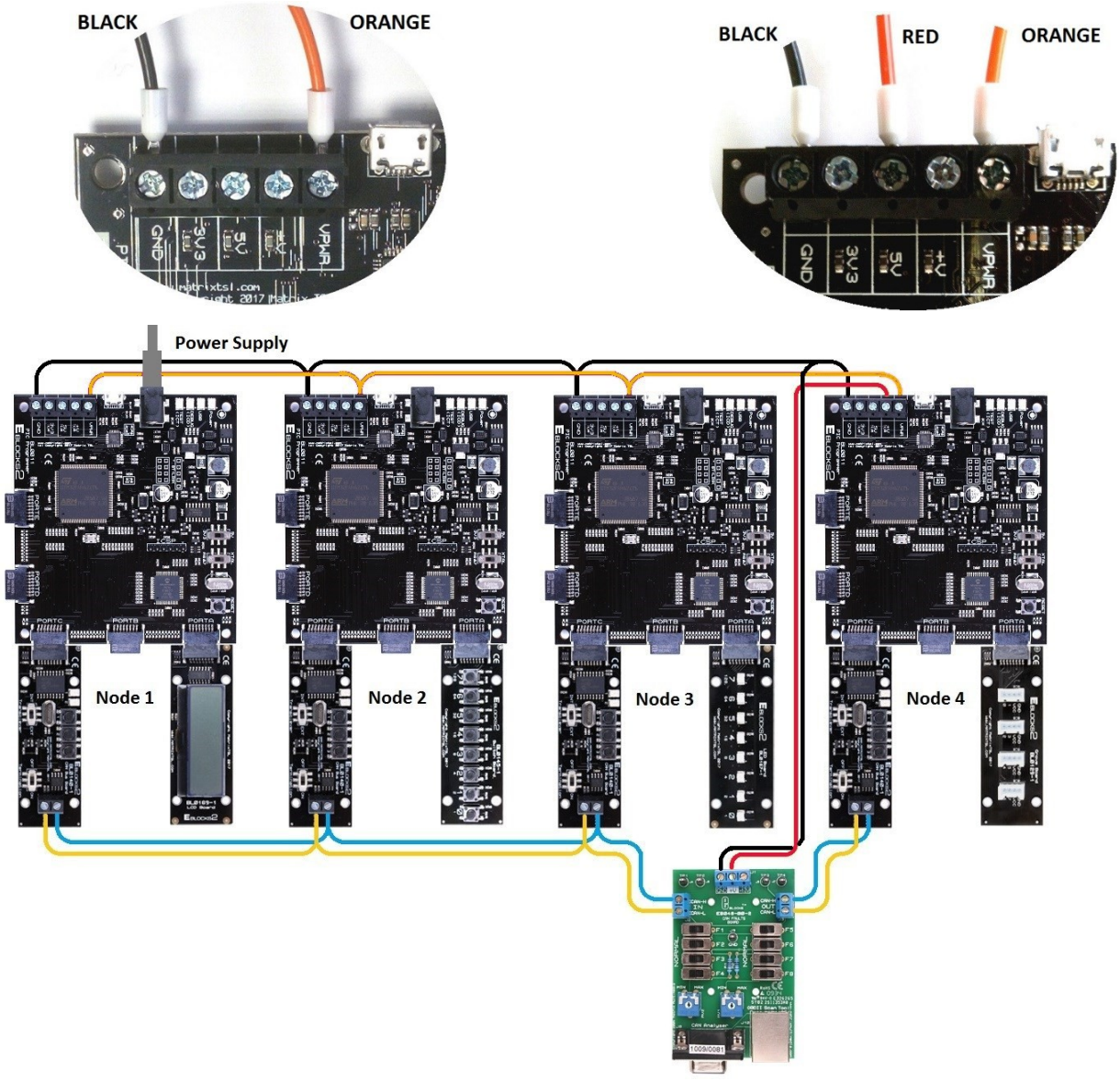


Figure 3.1 *La solution de formation CAN*

La solution de formation CAN consiste en des panneaux de fond de panier avec un nœud sur chaque panneau. L'alimentation des panneaux peut provenir d'un seul bloc d'alimentation relié à l'une des cartes de programmation, puis les signaux VPWR et GND sont bouclés comme indiqué ci-dessus. Un seul câble USB est nécessaire car chaque nœud doit être programmé séparément. Le câble USB peut être connecté à n'importe quel nœud pour la programmation. Cependant, un second port USB sur le PC est nécessaire pour l'analyseur CAN.

Note importante : les informations présentées ici sont correctes au moment où ce document a été produit. Veuillez consulter le site web de Matrix www.matrixtsl.com pour obtenir la dernière version de la documentation E-blocks2.

3.1 La carte CAN

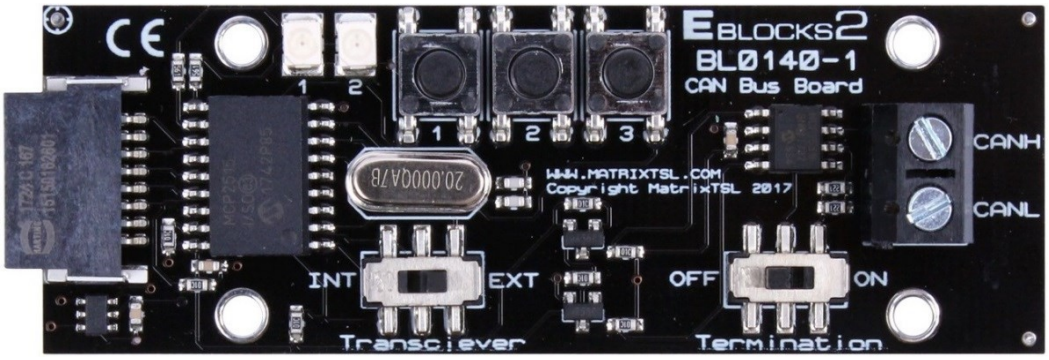


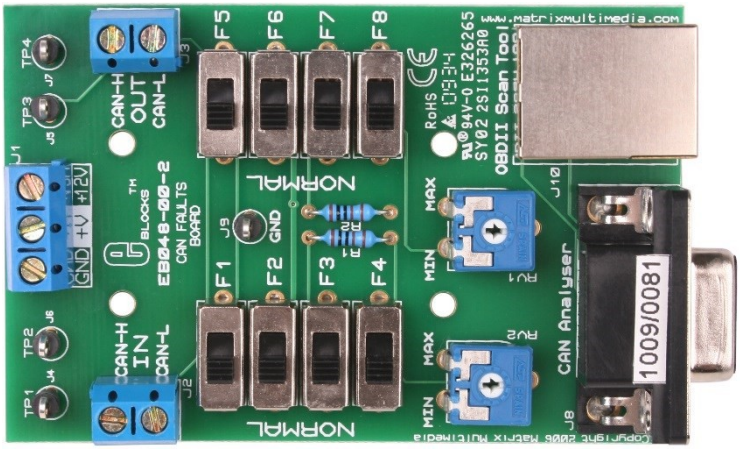
Figure 3.2 *La carte CAN E-blocks2 BL0140*

Un connecteur multivoie est utilisé ici pour connecter le contrôleur CAN au microcontrôleur en amont. Le commutateur Transceiver doit être réglé sur EXT et le commutateur Termination sur ON pour les nœuds 1 et 4, et sur OFF pour les nœuds 2 et 3. Le double bornier à vis est utilisé pour connecter le nœud au réseau CAN.

Les deux caractéristiques spécifiques à cette carte, qui peuvent ne pas être disponibles sur d'autres systèmes, sont les deux DEL et les trois interrupteurs. Les DEL peuvent être configurées comme des sorties générales ou comme des indicateurs d'état de la mémoire tampon. Les interrupteurs peuvent être configurés comme des entrées générales ou comme des interrupteurs d'activation de tampon CAN.
Ce point est abordé plus loin dans le cours.

3.1.1 Carte de défauts CAN

La carte de défauts CAN remplit trois fonctions : premièrement, elle sert de point de connexion de l'analyseur Kvaser au bus CAN à l'aide du connecteur de type D de la carte de défauts CAN, deuxièmement, elle permet d'insérer un certain nombre de défauts sur le bus CAN et, troisièmement, elle permet de connecter facilement des sondes d'oscilloscope aux lignes CAN high et CAN low à l'aide des broches de test TP 1 à TP 4. Les commutateurs marqués F1 à F8 permettent de placer chacune des lignes CAN dans quatre conditions de défaut distinctes : court-circuit à 5V, court-circuit à la masse, circuit ouvert et circuit partiellement ouvert. Les potentiomètres RV1 et RV2 permettent de faire varier la résistance du circuit partiellement ouvert. Le circuit de la carte de défauts CAN est disponible dans la fiche technique de la carte de défauts CAN.



3.1.2 Installation

1. Installer Flowcode
2. Vérifiez les mises à jour de Flowcode à l'aide de l'option du menu d'aide "Vérifier les mises à jour".
3. L'analyseur CANLeaf fourni dans le cadre de la solution CAN nécessite l'installation d'un pilote USB pour fonctionner correctement. Voir le dossier Kvaser dans les fichiers de ressources (lien à la page 8) pour les instructions et les fichiers de pilotes. IMPORTANT - Ne branchez pas l'analyseur CANLeaf USB avant que la routine d'installation ne vous le demande.
4. L'analyseur Kvaser nécessite l'installation du logiciel CANKing. Une copie de ce logiciel se trouve dans les fichiers de ressources (lien à la page 8) avec la solution de formation CAN. IMPORTANT - CANKing doit être installé avant l'analyseur Kvaser, sinon l'analyseur CAN risque de ne pas être reconnu correctement.

3.2 Configuration du système CAN

Le nœud CAN de base que nous utilisons ici est constitué d'un processeur PIC ou Arduino, auquel est attachée une carte CAN BL0140.

3.2.1 Configuration des nœuds CAN

La solution CAN se compose de 4 nœuds, plus un nœud d'attachement pour l'analyseur CAN de Kvaser :

Nœud 1) Nœud de contrôle de surveillance et d'affichage Nœud 2) Nœud de commutation d'entrée
Nœud 3) Nœud d'affichage de sortie Nœud 4) Nœud de capteur analogique.

Équipé de capteurs : Lumière (prise 0,1), Rotation (prise 2,3) et Température (prise 4,5)

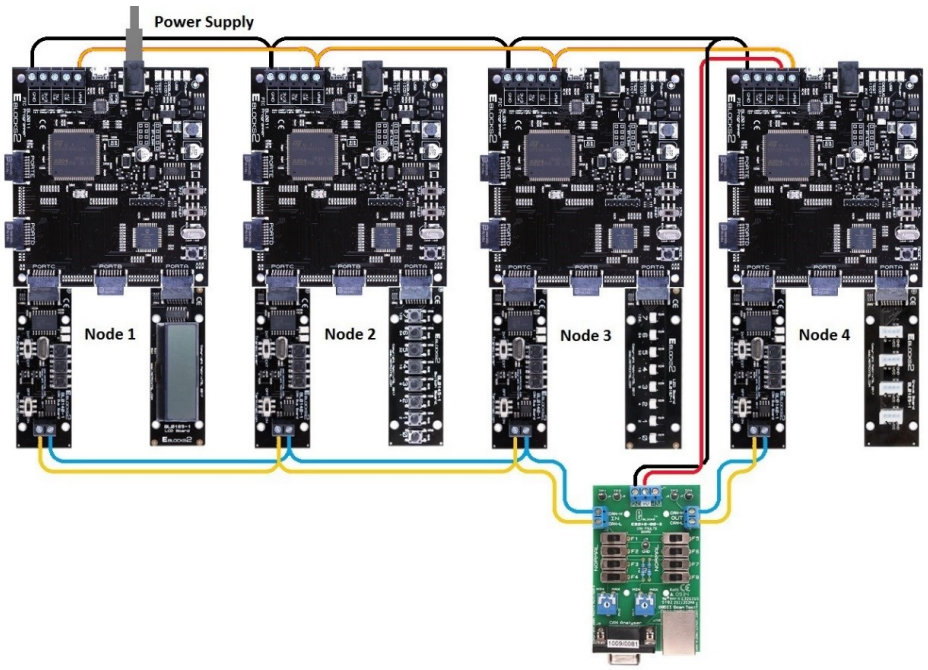


Figure 3.3 *Système CAN - câblage des connexions d'alimentation*

3.2.2 Configurations de la carte E-Blocks2

	PIC BL0011			Arduino BL0055		
	Port A	Port B	Port C	A0-5 (Port C)	D0-7 (Port D)	D8-13 (Port B)
Nœud 1	BL0169		BL0140	BL0169		BL0140
Nœud 2	BL0145		BL0140	BL0145		BL0140
Nœud 3	BL0167		BL0140	BL0167		BL0140
Nœud 4	BL0129		BL0140	BL0129		BL0140

3.2.3 Test du système CAN

Configurez le système CAN avec les panneaux sous tension et l'analyseur Kvaser branché et connecté au PC. La ligne CANH (CAN High) est équipée d'un fil bleu et la ligne CANL (CAN Low) d'un fil jaune (mais il est toujours préférable de vérifier par sécurité).
Les nœuds 1 et 4 sont des nœuds d'extrémité. Assurez-vous que le commutateur de terminaison est réglé sur ON. Pour les nœuds 2 et 3, l'interrupteur de terminaison est réglé sur OFF.

Les nœuds 1 et 4 sont des nœuds d'extrémité. Assurez-vous que le commutateur de terminaison est réglé sur ON. Pour les nœuds 2 et 3, l'interrupteur
L'interrupteur de terminaison est en position OFF.

Les programmes de test sont disponibles au format Flowcode et peuvent être compilés et téléchargés vers les nœuds CAN appropriés.

- Nœud 1 - Affichage LCD et alerte de bas niveau de carburant
- Nœud 2 - interrupteur de pédale de frein
- Nœud 3 - feu de freinage
- Nœud 4 - carte du capteur de carburant

Une fois les quatre nœuds programmés, vous pouvez tester les nœuds 1 et 4 en déplaçant le capteur rotatif et en observant un changement correspondant sur l'écran du nœud 1. Les nœuds 2 et 3 peuvent être vérifiés en appuyant sur le bouton-poussoir 0 du nœud 2 et en observant un signal sur la LED 0 du nœud 3.

3.3 Tester vos programmes

Comme CAN nécessite deux nœuds pour être utile, et donc deux programmes distincts fonctionnant en même temps, nous ne pouvons pas les simuler dans Flowcode. Pour contourner ce problème, nous devons utiliser une sorte d'analyseur qui se branche sur le réseau CAN et surveille les messages envoyés.
Afin de vous permettre de surveiller et de tester vos programmes, nous avons inclus l'analyseur CANLeaf de Kvaser dans nos packs de solutions. L'analyseur CAN de Kvaser peut être branché sur notre réseau CAN au niveau du nœud d'analyse et se connecte à un PC via USB pour vous permettre de surveiller le réseau.

Lorsque vous testez votre système CAN, assurez-vous que les interrupteurs de la carte EB048 CAN faults sont en position normale.

3.4 Configuration de l'analyseur CANLeaf de Kvaser

L'analyseur CAN Kvaser est livré avec des instructions d'installation, une documentation et un logiciel sur le Kvaser d'accompagnement dans les fichiers de ressources (lien à la page 8). Veuillez vous référer à la documentation fournie pour plus de détails sur l'installation et la configuration du logiciel.

3.4.1 Utilisation de l'analyseur CANLeaf de Kvaser

Connecter l'analyseur CANLeaf au nœud de l'analyseur Kvaser avec le connecteur de type D du canal
1. Connecter l'analyseur au PC avec le connecteur du câble USB.
Le programme d'analyseur CAN s'appelle CANKing et devrait se trouver dans votre menu Programmes. Ouvrez CANKing.

Les deux parties principales qui nous intéressent ici sont les boutons Démarrer et Pause et l'écran Message. Les boutons Démarrer et Pause font ce qu'ils disent, ils vous permettent de démarrer, de mettre en pause et de redémarrer l'analyse.

La fenêtre de sortie affiche tous les messages sur le réseau CAN.

L'identifiant du message, la longueur des données, les éléments de données, l'heure d'envoi et d'autres informations sont notés. Cela peut vous aider à repérer des problèmes dans votre code en raison de données manquantes, d'ID erronés, etc., ou à vérifier qu'un nœud reçoit bien les bonnes informations s'il ne répond pas correctement. Vous pouvez également insérer des messages personnalisés sur le réseau à des fins de test et de débogage. L'analyseur rend la vie beaucoup plus facile et devrait être utilisé comme une évidence lors de la programmation.

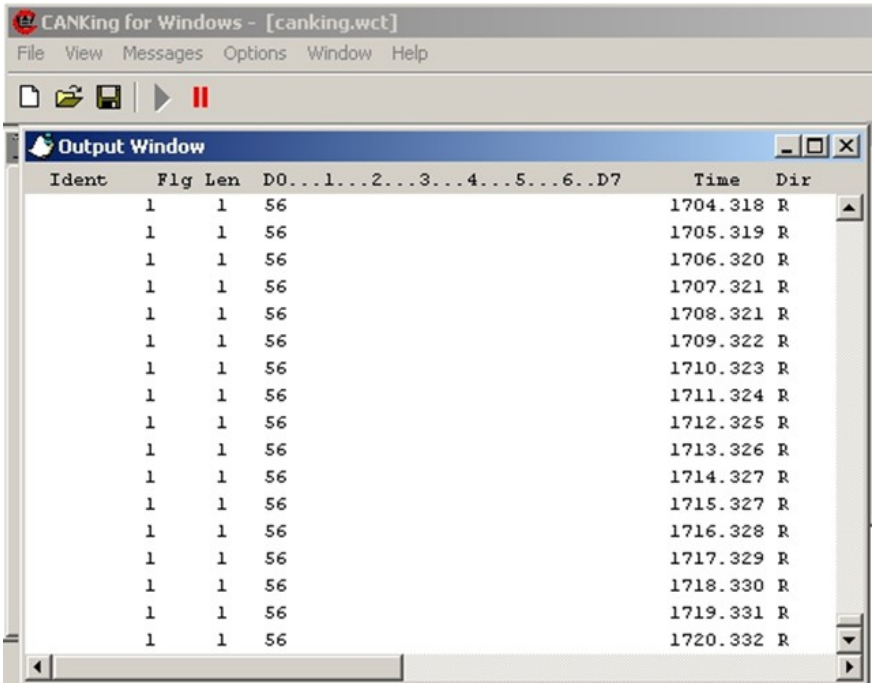


Figure 3.4 *La fenêtre de sortie de CANKing*

3.4.2 Paramètres du réseau de l'analyseur

Pour que l'analyseur fonctionne correctement sur le réseau, il faut régler les paramètres du bus. Si vous utilisez les paramètres suggérés pour le composant CAN, ceux-ci devraient déjà correspondre. Si toutefois vous devez les modifier pour une raison quelconque, ils se trouvent dans l'onglet Bus Parameters de la fenêtre CAN controller.

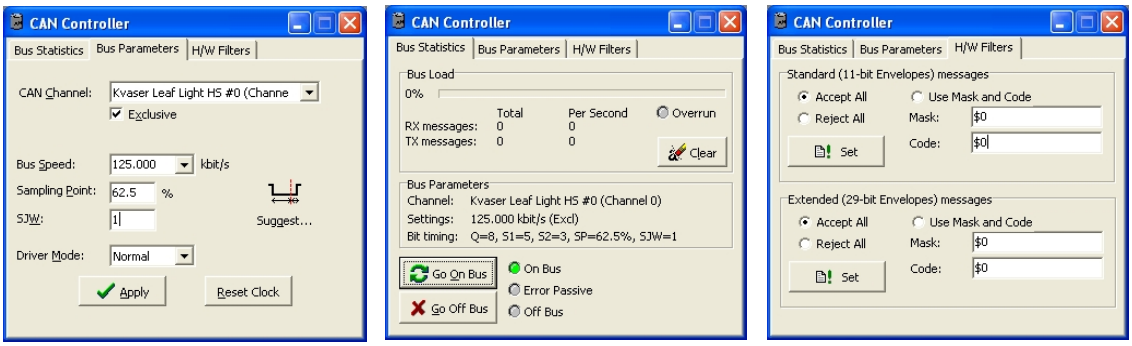


Figure 3.5 *Dialogue de configuration du contrôleur CAN*

Les détails statistiques et l'état actuel du bus sont contenus dans la page Statistiques du bus, ainsi que les boutons de connexion On/Off du bus (voir Fig. 3.5). Les paramètres standard utilisés par CANKing avec l'option de paramètres suggérés pour la carte CAN Matrix sont les suivants :

CAN Channel : Sélectionner les options USB CANLeaf
Exclusif : Sur
Vitesse du bus : 125 kbps
Point d'échantillonnage : 62.5%
SJW : 1
Mode de conduite : Normal

4 L'implémentation du CAN de Matrix

4.1 La couche physique

La couche physique utilisée dans le système CAN de Matrix se présente sous la forme d'une *paire de fils torsadés* se terminant par des résistances de terminaison. Les résistances sont ajoutées aux extrémités pour éviter les pertes de signal ou les interférences. La carte CAN dispose d'une prise CANH (CAN High Line) et d'une prise CANL (CAN Low Line) qui sont utilisées pour connecter le nœud au réseau. Un fil bleu est utilisé pour CAN High et un fil jaune est utilisé pour CAN Low. Ce code de couleur n'est pas obligatoire, vous pouvez utiliser votre propre code de couleur si vous en avez déjà un, mais vous devez relier CANH à CANH et CANL à CANL lors des connexions.

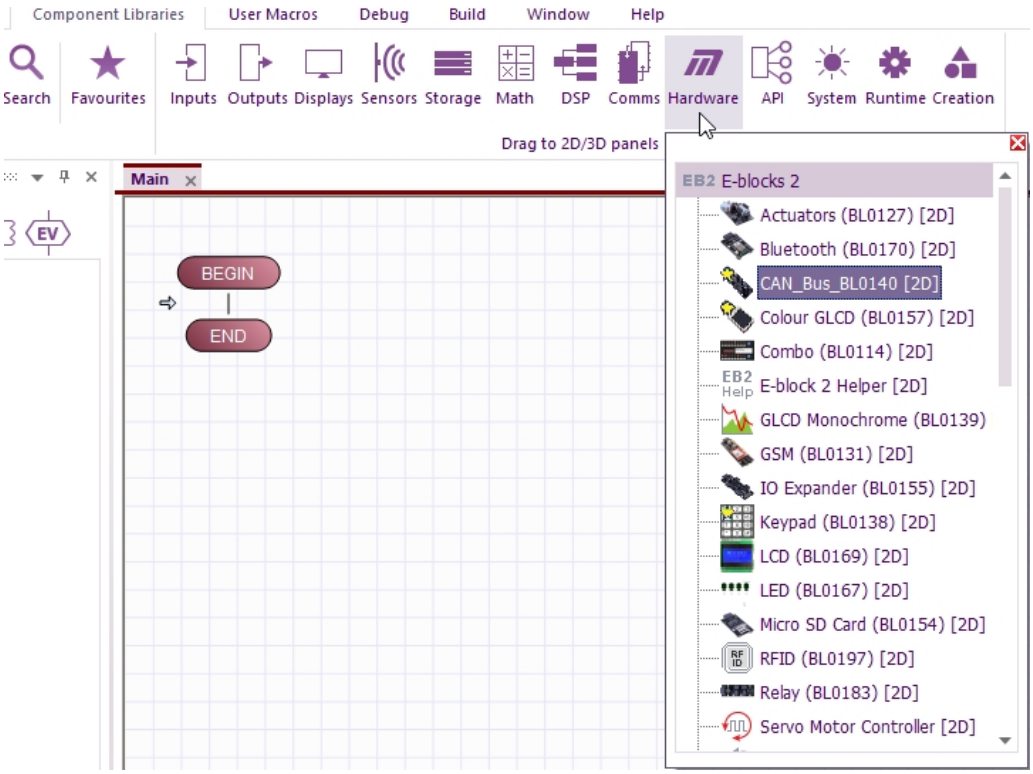
Les résistances de terminaison sont réglées au moyen d'un cavalier - J. Déplacez le cavalier sur la position END NODE pour régler la résistance de terminaison pour les nœuds situés aux extrémités du réseau.

La carte utilise à la fois un contrôleur CAN (MCP2515) et un émetteur-récepteur CAN (MCP2551). Lecontrôleur CAN utilise lebusSPI pour configurer le contrôleur CAN afin de transmettre et de recevoir des informations CAN. Les informations envoyées et reçues sont stockées dans une série de tampons. Trois tampons d'émission et deux tampons de réception sont utilisés pour stocker les données. Notez que ces tampons font partie de notre implémentation d'une couche physique pour CAN et ne font pas partie de la spécification CAN elle-même.

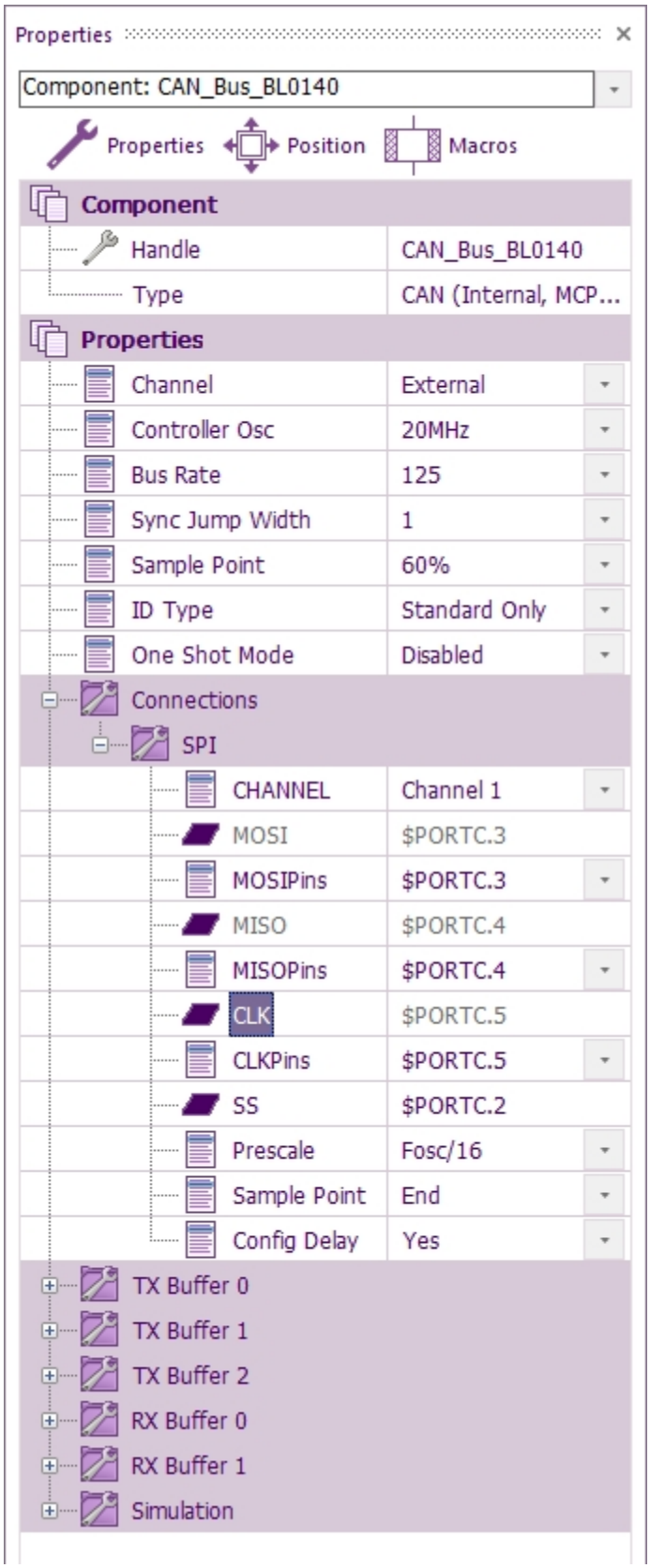
4.2 Le composant Flowcode CAN

Le composant CAN de Flowcode utilise une série de propriétés et de macros pour fournir une messagerie CAN dans Flowcode. Les propriétés sont utilisées pour définir des valeurs par défaut générales, telles que le débit en bauds et le point d'échantillonnage, et des données de message CAN, telles que les ID de message et les données par défaut. Les macros permettent à l'utilisateur d'initialiser le système CAN et d'envoyer et de recevoir des données. Les macros permettent à l'utilisateur de modifier les principales parties du message CAN, telles que l'ID du message et les données envoyées. D'autres parties du message CAN non liées à la transmission de données, telles que les bits ACK et le remplissage de bits, sont gérées automatiquement dans les coulisses.

Le composant CAN de ce cours se trouve dans la section Hardware > Eblocks2 de la barre d'outils Components Libraries.



Les détails des macros et des propriétés du composant CAN de Flowcode sont contenus dans le fichier d'aide du composant CAN. Les paramètres illustrés dans les figures 4.1 à 4.4 et énumérés dans les sections 4.3 à 4.5 sont utilisés pour tous les projets de ce cours, sauf indication spécifique dans les instructions relatives à cette tâche.



Propriétés des composants CAN
Les propriétés des composants CAN se trouvent dans le "panneau des propriétés" lorsque le composant CAN est sélectionné / mis en surbrillance.

Propriétés
La section *Propriétés* permet de définir les paramètres de con- figuration CAN.

Connexions
La section *Connexions* vous permet de définir les paramètres *SPI* pour le CAN. Pour le PIC, il s'agit du port C, pour Arduino/ AVR, du port B.

Tampon TX
Les sections 3 *TX Buffer* vous permettent de définir les détails par défaut pour les trois tampons d'émission (*TX Buffer 0* à *TX Buffer 2*) utilisés dans le composant CAN. Sauf modification dans le programme par des macros, ces paramètres par défaut seront les valeurs d'*ID de message* et les valeurs de données (*D0* à *D7*) envoyées.

Tampon RX
Les 2 sections du *tampon RX* vous permettent de définir des *masques* et des
Filtres pour traiter et sélectionner les ID de messages à recevoir (voir plus loin). Il est également possible de configurer les *paramètres de* chaque tampon individuellement.

4.3 Dispositifs de microcontrôleurs ciblés

Ce cours a été rédigé à l'aide de microcontrôleurs PIC et AVR. Si vous souhaitez utiliser d'autres microcontrôleurs, vous devrez adapter les paramètres, les programmes et les instructions au nouveau dispositif.

4.4 Réglages des composants CAN

Les paramètres requis sont les suivants :

Canal : Externe
Contrôleur Osc : 20MHz
Taux de bus: 125 (kbps)
Point d'échantillonnage : 60%
SJW : 1
Type d'identification : Standard uniquement

Note : Assurez-vous que la ***vitesse du bus*** est réglée sur 125, car elle est réglée sur 1000 par défaut.

	BL0011	BL0055
Canal :	Canal 1	Channel 1
Sélection de puce (SS):	Port C broche 2	Port B broche 2
MOSI:broche	du port C 3	broche du port B 3
MISO :	Port C broche 4	Port B broche 4
Pré-échelle :	Fosc/16	Fosc/16

.

4.5 Paramètres de configuration de Flowcode

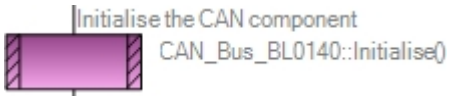
Cible PIC : BL0011
Cible Arduino : BL0055

4.6 Paramètres de la carte de développement PIC/E-blocks2

Sélecteur de tension : 5V

4.7 Macro d'initialisation CAN

Le composant CAN doit être initialisé avant de pouvoir être utilisé dans votre programme.
Pour initialiser le composant CAN, placez la macro CAN "Initialise" dans votre programme avant d'utiliser toute autre macro CAN. L'idéal est de la placer au début du programme.



5 Signaux CAN de base

Le CAN permet à la fois d'envoyer des signaux des nœuds vers le réseau et de recevoir des signaux du réseau et d'agir sur ces signaux s'ils sont destinés à ce nœud. Ces deux tâches sont des processus distincts. Un nœud peut envoyer un signal ou en recevoir un ; il n'est pas nécessaire de faire les deux. Cependant, il peut faire les deux, ce qui ajoute à la flexibilité des systèmes CAN.

Un système CAN de base nécessite un nœud qui peut envoyer des signaux et un nœud qui peut en recevoir. Il peut y avoir plus de nœuds, qu'ils envoient ou reçoivent des signaux, mais un minimum d'un nœud émetteur et d'un nœud récepteur sont nécessaires pour la communication.

Comme indiqué précédemment, les signaux CAN sont envoyés et reçus en utilisant notre implémentation de la couche physique - CAN spécifie le message en laissant la couche physique inférieure à notre charge. Dans notre système, nous avons des tampons d'émission TX et des tampons de réception RX qui sont utilisés pour stocker les données en vue de leur envoi ou pour les examiner lorsqu'elles sont reçues. Chaque message est envoyé avec une valeur d'identification de message qui peut être vérifiée par d'autres nœuds et prise en compte si elle correspond à une liste d'identifications à accepter.

5.1 Implémentation des signaux CAN de base dans Flowcode

5.1.1 Initialisation du composant CAN

Quelle que soit la fonction d'un nœud - envoi, réception ou un mélange des deux -, il faut que le composant CAN soit ini... pour que le composant fonctionne.
Ajoutez une macro *Initialise* à votre programme, de préférence au début, là où elle peut être vérifiée rapidement. Les signaux CAN de base nécessitent la mise en place de deux nœuds distincts - un nœud d'émission et un nœud de réception.

5.1.2 Nœuds d'envoi

Les paramètres par défaut du composant CAN pour les trois tampons d'émission TX sont tous réglés sur 0, ce qui signifie qu'ils doivent être configurés pour fonctionner. Une fois configurés, les messages peuvent être envoyés à l'aide de la macro *SendBuffer*. Le paramètre *Buffer* fait référence au tampon TX0-TX2 à envoyer, et peut donc être réglé entre 0 et 2 respectivement. Les propriétés de la mémoire tampon TX utilisées dans les fichiers d'exemple sont indiquées ci-dessous. Pour le tampon TX 0, l'ID du message est fixé à 688 et la longueur est fixée à 2, ce qui détermine le nombre d'octets de données qui seront envoyés - 85, 170, 0, 0, 0. Nous en apprendrons plus sur les ID de message et les données plus tard. Pour l'instant, il suffit de se rappeler que ces données seront envoyées avec cet ID de message lors de l'envoi du tampon TX 0.

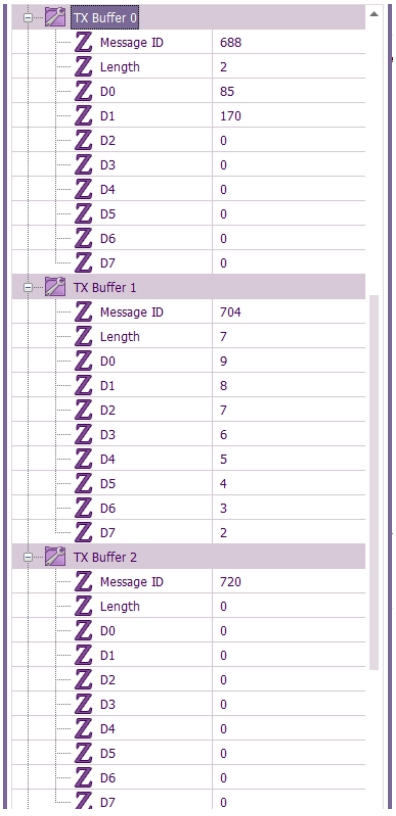


Figure 5.1 *Paramètres du nœud d'envoi*

5.1.3 Nœuds de réception

Le nœud de réception doit être configuré pour accepter les messages entrants afin de fonctionner. Par défaut, les paramètres de réception RX sont définis pour rejeter tous les messages. Pour modifier cette configuration, allez dans la section Tampon RX du panneau "Propriétés" et réglez les paramètres du tampon RX 0 sur "Utiliser le masque et le filtre", ce qui permet d'accepter un ensemble de messages spécifiques. Remarque : pour les paramètres spécifiés dans les fichiers d'exemple, réglez la propriété Filter 0 sur "100". Lorsque vous utilisez les paramètres de l'exemple, veillez à ce que les paramètres du tampon RX 1 soient réglés sur "Accepter tout".

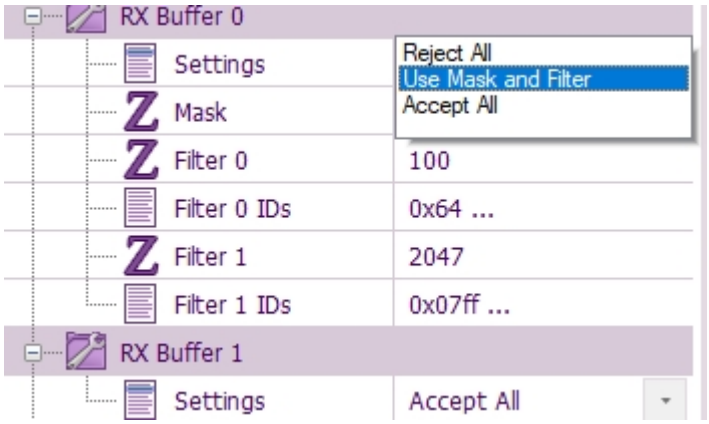


Figure 5.2 *Paramètres du nœud de réception*

Le nœud acceptera désormais toutes les transmissions sur ce tampon particulier, quel que soit l'expéditeur.

La macro *CheckRx* permet de vérifier l'arrivée des messages. Le paramètre buffer sélectionne le buffer de réception à vérifier. Dans ce cas, nous devons vérifier le tampon 0 pour le tampon RX 0. Si un message est arrivé, *CheckRx* renvoie une valeur non nulle. Dans ce cas, le nœud peut répondre au message en exécutant les fonctions pour lesquelles il a été programmé.

5.2 Utilisation des signaux CAN de base

Un système simple peut être mis en place en utilisant des signaux CAN de base. Toutefois, le système est limité à un seul signal d'activation et à une seule réponse. Plusieurs nœuds d'activation peuvent être présents et envoyer un signal d'activation, mais comme le nœud récepteur accepte tous les signaux, l'expéditeur n'a pas d'importance. De la même manière, tout nœud configuré pour recevoir tous les signaux réagira au message envoyé. Peu importe qui a envoyé le signal, ou à qui, seul compte le fait qu'un signal a été envoyé. Cela nous permet d'avoir plus d'un nœud répondant au même signal, mais comme ils acceptent tous les messages, la réponse sera la même pour tous les signaux envoyés.

Cette méthode de communication est très basique. Tous les signaux sont traités de la même manière et il n'existe aucun moyen d'empêcher que des signaux non destinés au nœud récepteur soient également acceptés, générant ainsi de faux événements. Bien que ce système puisse fonctionner pour un petit réseau à usage unique, par exemple pour un système de freins/feux de freinage, il est peu probable qu'un tel système soit utilisé en dehors d'un environnement d'apprentissage. Le système CAN tel qu'il est décrit ici n'est tout simplement pas assez robuste pour une utilisation pratique. Une méthode doit être utilisée pour rendre les nœuds plus sélectifs quant aux messages qu'ils envoient ou reçoivent. Ce point sera abordé dans la section suivante, qui traite des ID de message.

6 Démonstrations du RCA

Les trois démonstrations suivantes illustrent le CAN en action et certaines de ses principales utilisations. Ces démonstrations peuvent être utilisées pour donner aux étudiants une compréhension de base des systèmes CAN avant les exemples de programmation CAN.

Les démonstrations peuvent également servir à illustrer des concepts tels que la surveillance des messages, les analyses de démarrage et les tests de diagnostic pour les étudiants qui peuvent travailler avec des systèmes CAN, mais qui ne sont pas tenus de créer ou de programmer des systèmes CAN.

- DM01 (6.1) illustre une analyse de démarrage avec vérification des erreurs système de base. La vérification de la présence d'un nœud sur le système est la première étape de l'examen du problème.
- DM02 (6.2) est un exemple de base de contrôle des messages. La surveillance des messages est une méthode standard de diagnostic des défaillances. La surveillance du trafic de messages peut aider à identifier le ou les nœuds présentant des défauts.
- DM03 (6.3) est un programme de contrôle et de test spécifique à un nœud. Cette démonstration illustre l'utilisation de tests de diagnostic pour examiner un nœud particulier afin de diagnostiquer le défaut.

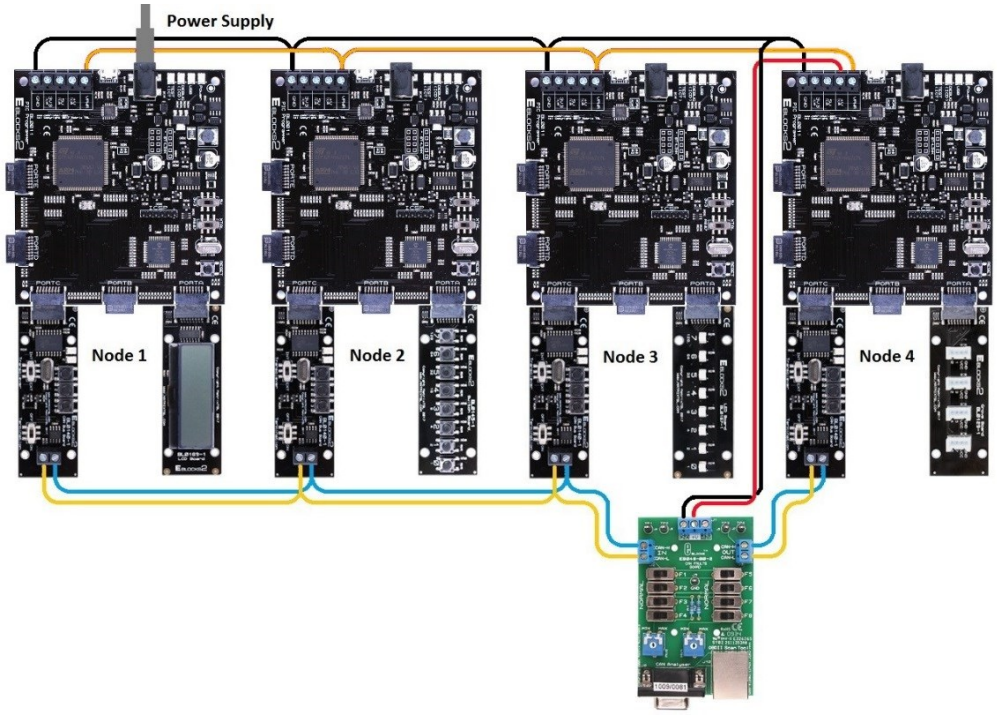


Figure 6.1 *Système CAN - Vue d'ensemble du système*

Nœud CAN 1 - Panneau de commande principal
Nœud CAN 2 - Panneau de commutateurs Nœud CAN 3 - Panneau de DEL Nœud CAN 4 - Panneau de capteurs

Les fichiers Flowcode FCFX sont fournis afin que les utilisateurs avancés puissent voir et travailler le code si nécessaire. Pour des démonstrations simples, les nœuds peuvent être préprogrammés pour permettre aux élèves de les examiner immédiatement.

Remarque :
Pour ces démonstrations, seul l'octet HI de l'ID du message est utilisé afin de simplifier les mathématiques.
On suppose que tous les octets LO de l'ID du message sont identiques et qu'ils peuvent donc être ignorés dans le cadre de ces démonstrations.

6.1 DM01 - Balayage de démarrage

6.1.1 Objectif

Démonstration d'un balayage initial de démarrage qui vérifie que les composants du système sont tous connectés et fonctionnent, et démonstration de l'utilité de ce balayage pour le diagnostic automobile.

6.1.2 Ressources :

Cette tâche utilise 5 programmes :
DM01_N1 - Programme du nœud 1 (programme du panneau de commande principal) DM01_N2 - Programme du nœud 2 (interrupteur de frein)
DM01_N3 - Programme du nœud 3 (feux de freinage) DM01_N4 - Programme du nœud 4 (capteur de carburant) DM01_NX - Programme pour le nœud ne fonctionnant pas

Les programmes sont fournis au format FCFX afin que les utilisateurs puissent visualiser et modifier le

programme de l'organigramme. (Remarque : pour les besoins de cet exercice, les programmes ne

contiennent que le code de la procédure de balayage de démarrage).

6.1.3 Partie 1 : Exécution de l'analyse

1. Charger les programmes DM01_N1 - DM01_N4 dans les nœuds 1-4 respectivement.
2. Dès la réinitialisation du nœud du panneau de commande du nœud 1, le programme de balayage commence.
3. L'écran LCD affiche "Starting scan" (début du balayage)
4. Le nœud N2 suivant (interrupteurs de frein, etc.) sera vérifié.
5. Une fois le nœud N2 détecté, les mentions "N2 - Brake Sw" et "Present" s'affichent.
6. Le nœud suivant, N3 (feux de freinage), sera vérifié.
7. Le nœud N3 suivant (capteur de carburant) sera vérifié.
8. Enfin, l'écran LCD affiche le résultat global de l'analyse.
   Dans ce cas, "Tous les systèmes sont activés" pour montrer que tous les systèmes ont répondu correctement.

6.1.4 Que se passe-t-il ?

L'analyse fonctionne sur la base d'un système simple de messages et de réponses.

L'ensemble du système est divisé en quatre unités distinctes, appelées nœuds, qui fonctionnent indépendamment les unes des autres. Ils sont cependant tous connectés au bus CAN, ce qui leur permet d'écouter et d'envoyer des signaux sur le bus CAN.

Le nœud de la centrale - N1 - envoie une séquence de messages avec un numéro d'identification prédéfini aux autres nœuds du système.

Les messages ne sont pas envoyés à un nœud spécifique. Ils sont simplement placés sur le bus CAN pour que tous les nœuds les écoutent.

Les autres nœuds du système sont à l'écoute des messages. Ils ont été configurés pour regarder l'ID du message et ne répondre qu'à un message d'ID spécifique. Lorsqu'un nœud repère un message auquel il peut répondre, il envoie son propre message, c'est-à-dire qu'il répond au message initial.

Le nœud de contrôle N1 attend alors cette réponse. S'il n'en reçoit pas dans un délai déterminé, il enregistre le nœud comme étant absent et passe à l'élément suivant à vérifier. Si le nœud N1 reçoit un message, il sait alors que ce nœud par- ticulier est présent sur le système.

6.1.5 Partie 2 : Relever les erreurs.

1. Envoyer le programme DM01_NX à un ou plusieurs des nœuds 2-4 (par exemple le nœud 2).
2. Appuyez sur reset sur le nœud N1 pour réexécuter le balayage.
3. Notez ce qui se passe lorsque l'analyse atteint le(s) nœud(s) contenant DM01_NX.
4. Le programme marque une pause pendant qu'il recherche le nœud. Au bout d'un moment, il supposera que l'absence de réponse signifie que le nœud n'est pas connecté ou qu'il ne fonctionne pas et indiquera que le nœud est "non présent".
5. Lorsque tous les nœuds ont été recherchés, l'écran LCD affiche un message d'avertissement et énumère les nœuds qu'il n'a pas trouvés, par exemple "N2".
6. Ensuite, envoyez le programme DM01_NX aux nœuds N2-N4.
7. Appuyez sur reset pour relancer l'analyse.
8. Une fois l'opération terminée, un message d'erreur et le message "Aucun système trouvé" s'affichent pour vous informer du problème.

6.1.6 Que se passe-t-il ?

Le programme DM01_NX désactive effectivement le nœud en imitant un calculateur qui ne fonctionne pas et qui ne répond pas au "scan" du nœud N1.

6.1.7 Limites

L'analyse de démarrage vérifie simplement la présence d'un nœud : elle ne prouve pas que toutes les parties du circuit du nœud fonctionnent correctement. Par exemple, un nœud de carburant dont le capteur est cassé sera présent, mais ne fonctionnera pas. Cependant, le simple fait de savoir qu'un nœud est ou n'est pas présent peut aider à résoudre certains problèmes dans les réseaux automobiles.

Notez que nous avons créé des programmes distincts pour illustrer la manière dont le balayage de démarrage d'une voiture peut aider à vérifier le fonctionnement de la voiture avant le début d'un voyage. Dans la pratique, chaque capteur devrait avoir une routine de ce type incorporée dans son programme principal.

6.1.8 En quoi cela m'aide-t-il ?

Si un système (nœud) n'est pas connecté ou ne fonctionne pas, il ne peut pas être utilisé. Une analyse de démarrage est utile pour ce niveau de diagnostic de base. Tous les systèmes sont-ils présents et pris en compte ? Si vous découvrez que le nœud N2 ne répond pas à votre balayage, vous pouvez alors commencer à vérifier si le nœud N2 présente des défauts.
Alors que nous n'avons ici que quatre nœuds simples, imaginez un système plus complexe avec une centaine de nœuds ou plus. Un simple programme de diagnostic au démarrage peut vous épargner des heures de vérification avec un multimètre.

6.2 DM02 - Moniteur CAN

6.2.1 Objectif

- Pour surveiller les messages et les données envoyés sur le bus CAN.
- Pour afficher le nœud et la fonction, l'ID du message et les données.

6.2.2 Ressources :

Cette tâche contient 5 programmes :
DM02_N1 - Programme du nœud 1 (programme du panneau de commande principal) DM02_N2 - Programme du nœud 2 (nœud de commutation)
DM02_N3 - Programme du nœud 3 (nœud de lumière) DM02_N4 - Programme du nœud 4 (nœud de capteur) DM02_NX - Programme pour les messages inconnus

Les programmes sont fournis au format FCFX afin que les utilisateurs puissent visualiser et modifier le code.

6.2.3 Partie 1 : Surveillance des messages

1. Charger les programmes DM02_N1 - DM02_N4 dans les nœuds 1-4 selon le cas.
2. Dès la réinitialisation du nœud du panneau de contrôle du nœud 1, le programme de surveillance commence.
3. Lorsqu'un message est reçu, l'identifiant est comparé à une liste d'identifiants connus et le nœud et la fonction sont vérifiés.
   s'affiche sur la ligne supérieure de l'écran LCD.
4. Sur la ligne suivante de l'écran LCD, l'ID du message et les données s'affichent.

6.2.4 Signaux envoyés

- Le capteur de carburant du nœud 4 envoie automatiquement des signaux de niveau de carburant toutes les quelques secondes.
- Les interrupteurs du nœud 2 envoient des messages lorsqu'ils sont enfoncés : valeur de données 255 lorsqu'ils sont enfoncés et valeur de données 0 lorsqu'ils sont relâchés.
- Le nœud 3 allume les DEL le cas échéant, mais n'envoie aucun signal.

6.2.5 Que se passe-t-il ?

Le panneau de contrôle est à l'écoute de tout message et affiche l'ID du message et la valeur des données sur l'écran LCD. L'ID du message est également comparé à une liste d'ID de messages connus pour obtenir les informations sur le nœud et la fonction, qui sont également affichées. Les messages qui ne figurent pas sur la liste des nœuds et fonctions connus sont répertoriés comme "NX - Inconnu".

Certains messages sont envoyés automatiquement, comme la lecture du niveau de carburant - l'ECU contient un programme qui envoie les messages à intervalles réguliers. D'autres sont envoyés en réponse à une action, comme le fait d'appuyer ou de relâcher le commutateur de frein. Certains nœuds, comme le nœud 3, n'envoient jamais de messages : ils se contentent d'écouter les messages et de les traiter. Notez qu'il s'agit d'un exemple de nœud fonctionnant parfaitement, qui ne crée aucun message sur le bus CAN.

Notez que lorsqu'un interrupteur est enfoncé, un message CAN est généré, et lorsque l'interrupteur est relâché, un message différent est généré. Nous pourrions envoyer un signal continu jusqu'à ce que l'interrupteur soit relâché, mais cela inonderait le bus CAN de messages. Si un signal était envoyé uniquement pour appuyer sur un interrupteur, nous ne saurions jamais quand il a été relâché. Si la même valeur de données était envoyée, nous pourrions confondre le moment où l'interrupteur a été allumé et celui où il a été éteint. C'est pourquoi nous envoyons des valeurs de données différentes pour distinguer les transitions "allumer" et "éteindre". Vous pouvez également utiliser des messages différents avec des ID de message différents. L'important ici est de pouvoir différencier les différents états du signal.

6.2.6 Partie 2 : Messages inconnus.

1. Envoyez le programme DM02_NX à l'un des nœuds 2-4 (par exemple, le nœud 2).
2. Lorsqu'un message est reçu de ce nœud, il ne correspond à aucun des nœuds et fonctions connus.
3. Dans ce cas, le nœud est signalé comme "NX - Inconnu"
4. L'ID du message et les données éventuelles s'affichent sur la deuxième ligne de l'écran LCD.

Les messages inconnus sont très rares dans un nouveau système automobile, mais ils peuvent se produire dans certaines circonstances. Par exemple, lorsqu'un véhicule a été endommagé, un nouveau calculateur peut être installé, dont la version du logiciel est plus récente que celle du véhicule d'origine. Ce nouveau calculateur CAN peut avoir des messages légèrement différents ou générer de nouveaux messages. Pour cette raison, les techniciens automobiles peuvent être amenés à télécharger de nouveaux logiciels sur certaines parties du véhicule afin de tenir compte des modifications techniques survenues lors de la modification de la conception.

6.2.7 Limites

Les ID des messages sont comparés à une liste de nœuds connus et à leurs fonctions contenues dans le nœud N1. Chaque message ne contient qu'un identifiant et quelques données - un programmeur a créé une sorte de table de recherche dans le nœud N1 qui relie ces données à la fonction du nœud d'où provient le message. Un message ne nous dit pas d'où il vient. Il contient uniquement l'ID du message et les données. Si un message arrive avec un ID de message figurant dans la liste, il sera signalé comme provenant de ce nœud et de cette fonction, qu'il le soit ou non. Il s'agit d'un point très important à comprendre. Par exemple, si vous avez programmé un calculateur de capteur de température avec le programme d'un capteur de carburant, il est possible que vous vous retrouviez avec un indicateur de niveau de capteur de carburant qui est en fait régi par la température du bloc moteur !

En raison des contraintes de taille de l'écran LCD, nous ne pouvons afficher que le premier élément de données du message. Cependant, les nœuds CAN peuvent envoyer jusqu'à huit données par message.

6.2.8 En quoi cela m'aide-t-il ?

Le bus CAN est un système de messagerie. Il transporte les messages produits par les nœuds le long du système pour que d'autres nœuds les écoutent et y réagissent si nécessaire. Le programme CAN Monitor nous permet de visualiser ces informations.

La nature de ces messages et les données qu'ils contiennent sont d'une importance capitale pour nous. Des données et des messages erronés ou manquants sont les principaux indicateurs de problèmes. Si le nœud de freinage n'envoie pas de messages lorsqu'il est actionné, nous savons qu'il y a un problème au niveau du nœud de freinage. S'il envoie des données mais que le feu stop ne s'allume pas, nous savons qu'il y a un problème au niveau du nœud du feu stop. Le simple fait de surveiller les messages de freinage peut nous indiquer où se situe le problème : à la source des messages ou à la destination.

En surveillant les messages envoyés et les valeurs des données, ainsi que ce qui provoque des signaux et ce qui réagit, nous pouvons commencer à diagnostiquer les problèmes et réduire le nombre de nœuds individuels nécessitant des tests de diagnostic supplémentaires. Si une unité est mise à niveau ou modifiée, nous pouvons également contrôler ses messages et ses données pour nous assurer qu'elle fonctionne correctement.

6.3 DM03 - Programme de diagnostic des capteurs

6.3.1 Objectif

- Pour surveiller les messages et les données envoyés sur le bus CAN.
- Pour afficher le nœud et la fonction, l'ID du message et les données.

6.3.2 Ressources :

	PIC BL0011			Arduino BL0055		
	Port A	Port B	Port C	A0-5 (Port C)	D0-7 (Port D)	D8-13 (Port B)
Nœud 1	BL0169	BL0145	BL0140	BL0169	BL0145	BL0140
Nœud 2	BL0145		BL0140	BL0145		BL0140
Nœud 3	BL0167		BL0140	BL0167		BL0140
Nœud 4	BL0129		BL0140	BL0129		BL0140

Le nœud 4 BL0129 doit être équipé de capteurs : Lumière (prise 1, broches 0,1), Rotation (prise 2, broches 2,3) et Température
(prise 3, broches 4,5)

Les programmes sont fournis au format HEX pour une utilisation immédiate et au format FCFX pour permettre aux utilisateurs de visualiser et de modifier le code.

6.3.3 Partie 1 : Contrôle des capteurs

1. Charger les programmes DM03_N1 à DM03_N4 dans les nœuds 1-4 respectivement.
2. Dès la réinitialisation du nœud du panneau de contrôle du nœud 1, le programme de balayage de démarrage commence. Ce programme est décrit en détail dans le document DM01.
3. Le balayage de démarrage est utilisé pour vérifier que le nœud de capteur est présent.
4. Une fois l'analyse terminée, le programme contrôle et affiche les valeurs des capteurs. Vous devriez pouvoir constater un changement dans la valeur affichée en modifiant le niveau d'éclairage, la température et la résistance variable sur la carte du capteur qui est utilisée pour imiter le niveau de carburant.
5. La température, le niveau de carburant et les niveaux d'éclairage sont contrôlés, les valeurs des capteurs s'affichant sur l'écran LCD sous le titre approprié.
6. Les données ont une valeur initiale de zéro et sont mises à jour à chaque fois qu'un message arrive, contenant des données du nœud de capteur sur l'état des capteurs.

6.3.4 Que se passe-t-il ?

Le panneau de contrôle écoute les messages des nœuds de capteurs et enregistre les données qu'ils lui transmettent. Dans ce cas, les données sont des nombres compris entre 0 et 255. Dans la pratique, un programme distinct sur le tableau de bord sera utilisé pour convertir ces données en une quantité significative pour l'homme, par exemple le carburant restant.

Dans cet ensemble de programmes, les données reçues sont affichées pour permettre à un technicien de contrôler les valeurs des capteurs. Différents problèmes entraînent des lectures distinctes qui peuvent être utilisées pour prédire les défauts survenus.

6.3.5 Limites

Le programme de surveillance des capteurs n'affiche que les valeurs des données et ne fournit pas de messages d'erreur diagnostiques pour toutes les situations. Compte tenu de la petite taille du système, ce n'est pas un problème, mais des systèmes plus complexes pourraient nécessiter l'incorporation de meilleures procédures internes de contrôle des erreurs et de messages de diagnostic. Cependant, comme vous le verrez dans les prochains exercices, le système CAN peut grandement aider à trouver les défauts des nœuds.

6.3.6 Partie 2 : Problèmes potentiels

La série d'erreurs suivante montre différents tests de diagnostic en action.

6.3.7 Erreur 1 : Le nœud 4 n'est pas alimenté.

1. Mettre en place les programmes comme dans la partie 1.
2. Débrancher l'alimentation du nœud de capteur
3. Exécutez le programme de diagnostic.
4. L'analyse de démarrage indique que N4, le nœud du capteur, n'est pas présent, ce qui signifie qu'il y a un problème avec cette unité.
5. Les valeurs de surveillance seront toutes égales à zéro et ne changeront pas.

Il s'agit du premier test, qui consiste à vérifier que le nœud est bien présent. Si ce test indique que le nœud du capteur n'est pas présent, nous savons que le défaut affecte l'ensemble du nœud et que l'erreur est probablement due à une défaillance de l'ECU principal, à une perte d'alimentation ou à une mise à la terre.

6.3.8 Erreur 2 : Les capteurs ne sont pas alimentés.

1. Configurer les programmes comme dans la partie 1
2. Coupez l'alimentation de la carte du capteur (pas tout le nœud, seulement la carte du capteur) en dévissant la borne à vis d'alimentation et en retirant le fil rouge.
3. Exécutez le programme de diagnostic.
4. Le balayage de démarrage indique que le nœud N4, le nœud capteur, est présent.
5. Les valeurs de surveillance resteront toutes bloquées à leur valeur par défaut et ne changeront pas, ce qui indique que les capteurs ne sont pas mis à jour.
6. La stimulation des capteurs ne produit aucun changement dans la lecture.

Ici, le nœud de capteur N4 fonctionne correctement. Cependant, une partie du circuit du capteur est défectueuse et le nœud ne peut pas la détecter. Nous savons que le nœud fonctionne parce que nous obtenons des relevés : comme aucun des relevés des capteurs ne change lorsque nous stimulons les capteurs, nous savons que le défaut doit se situer au niveau de la carte du capteur dans son ensemble.

6.3.9 Erreur 3 : Panne en cours de route

1. Configurer les programmes comme dans la partie 1
2. Exécuter le programme de diagnostic
3. Surveillez les valeurs des capteurs pour voir si elles changent et se mettent à jour lorsque les capteurs sont stimulés.
4. Débranchez l'alimentation du nœud de capteur.
5. Plus aucun message ne sera envoyé, ce qui signifie que les valeurs des données du capteur ne seront pas mises à jour.
6. Les relevés semblent se bloquer. La stimulation des capteurs ne modifie pas leurs valeurs.

Nous simulons ici une panne, où la perte de puissance se produit après le démarrage, de sorte que nous n'obtenons pas l'avertissement "Not present" qui nous indiquerait le problème à ce moment-là. Cela pourrait être le résultat d'un défaut intermittent dans la ligne d'alimentation d'un capteur ECU : le système fonctionne au démarrage, mais tombe en panne en cours de route.

Il s'agit d'une situation délicate car, dans l'idéal, nous voudrions savoir si un système a échoué et n'est pas en train de se mettre à jour. Il y a plusieurs façons de le vérifier.
Nous pourrions, par exemple, fixer un délai après lequel les relevés du capteur passent à "-".
Nous pourrions également réexécuter l'analyse de démarrage pour l'ensemble du système toutes les quelques minutes pendant le trajet afin de vérifier que tous les nœuds sont présents.

7 Exemple pratique 1 : Brake !!!!

Pour ce premier exemple, nous nous contenterons d'envoyer un message et de voir si nous pouvons le recevoir sur l'autre nœud, en réagissant d'une manière ou d'une autre pour montrer qu'il est arrivé.

7.1 Objectif

Créez un réseau CAN qui allume le feu stop lorsque vous appuyez sur la pédale de frein. La pédale de frein est sur l'interrupteur 0 du nœud 2, et le feu arrière est sur la LED 0 du nœud 3.

7.2 Partie 1 : Les programmes de base

Pour ce programme, nous utiliserons les paramètres par défaut.

Note : Vous devrez configurer la carte de développement du microcontrôleur pour le dispositif de microcontrôleur approprié**.**

7.2.1 Le signal d'envoi

1. Sur l'organigramme, ajoutez une icône "Appeler la macro" et définissez-la sur *Initialiser*. Il s'agit d'une macro importante qui est nécessaire pour

le fonctionnement du composant CAN. Il est préférable d'ajouter cette macro au début du programme ou à proximité pour faciliter la consultation.

2. Ajoutez une boucle pour que le programme s'exécute en continu.
3. Ajoutez une icône d'entrée et récupérez la valeur de l'interrupteur D0, la pédale de frein, dans une variable appelée *BRAKE*.
4. Ajouter une icône de décision réglée sur *FREIN > 0*.
5. Sur la branche *YES*, ajoutez une autre macro. Ouvrez le panneau des propriétés de la macro et sélectionnez la macro *SendBuffer.* La macro prend un paramètre - *Buffer*. Nous aborderons les tampons plus en détail ultérieurement. Pour l'instant, attribuez la valeur '0' à *Buffer*.
6. Ajoutez un court délai (icône Delay réglée sur 100 ms) juste avant la fin de la boucle.
7. Enregistrez le programme sous CAN_Example_01_Send.fcfx.

La figure 7.1 montre l'implémentation Flowcode. Le programme est maintenant prêt à être compilé et téléchargé vers un nœud CAN. Nous avons maintenant un nœud CAN fonctionnel. Lorsque nous appuyons sur l'interrupteur D0, un message est envoyé à la mémoire tampon 0.

7.2.2 Le signal de réception

Nous avons un message - il nous faut maintenant un nœud qui puisse y réagir !

1. Lancez un deuxième programme, toujours avec le microcontrôleur approprié.
2. Ajouter une macro CAN *Initialise* et une boucle.
3. Dans la boucle, ajoutez la macro *ReadRx* et définissez le paramètre *Buffer* à "0" pour qu'il corresponde à celui du message que nous avons reçu.

envoyé dans le premier programme.

4. Créez une variable MESSAGE et utilisez-la pour la valeur de retour.
5. Lorsqu'un message est envoyé à la mémoire tampon 0, ReadRx renvoie une valeur non nulle.
6. Ainsi, si nous faisons suivre la macro Read Rx d'une icône de décision, nous pouvons alors réagir au message.
7. Ajoutez l'icône de décision et réglez-la sur *MESSAGE > 0*.
8. Dans la boucle YES, ajoutez une icône de sortie pour que nous puissions réagir au message.
9. Nous allons régler ce paramètre pour allumer la DEL 0 - le feu de freinage.
10. Sur la boucle NO, ajoutez une icône de sortie pour éteindre D0, de sorte qu'il ne soit pas allumé en permanence.
11. Ajoutez un court délai (icône Delay réglée sur 100 ms) juste avant la fin de la boucle.
12. Enregistrez le programme sous CAN_Example_01_Receive.fcfx.

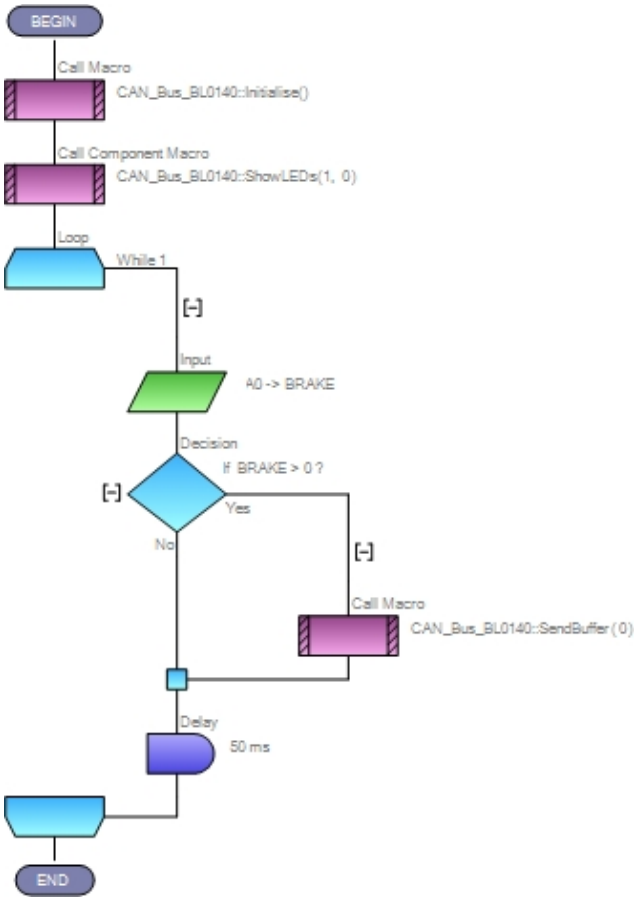


Figure 7.1 *Organigramme du signal d'envoi*

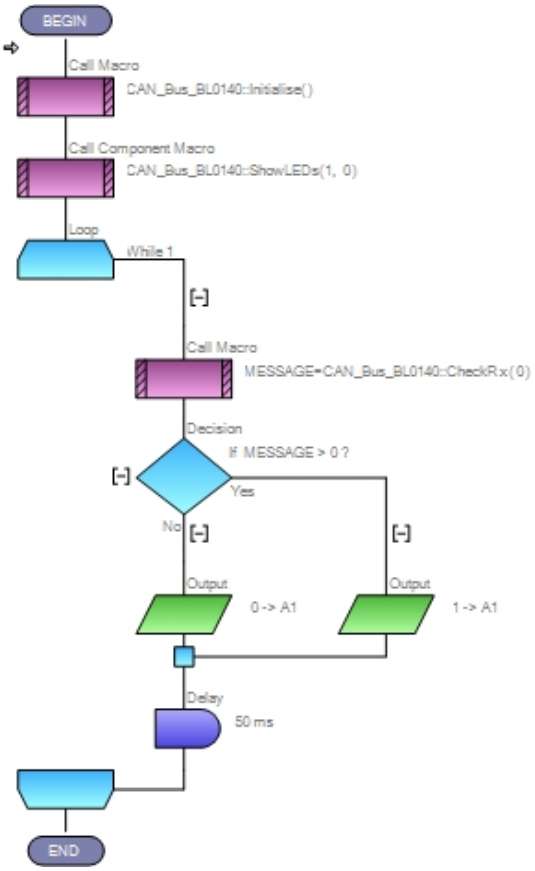


Figure 7.2 *Organigramme du signal de réception*

Après avoir saisi les organigrammes, vous êtes maintenant prêt à les compiler et à les télécharger.

1. Compilez et téléchargez CAN_Example_01_Send.fcfx sur le nœud CAN avec les commutateurs attachés.
2. Compilez et téléchargez CAN_Example_01_Receive.fcfx sur le nœud CAN avec les LEDs attachées.
3. Appuyez sur l'interrupteur 0 et, si tout s'est bien passé, la LED 0 doit s'allumer.

7.2.3 Tester les programmes

Le fait d'appuyer sur l'interrupteur 0 du nœud 2, le nœud des interrupteurs, devrait maintenant allumer

la DEL 0 du nœud 3, le nœud des DEL. Pour visualiser le trafic réseau

1. Connecter l'analyseur CANLeaf au nœud de l'analyseur sur le réseau.
2. Ouvrez CANKing et sélectionnez le périphérique USB CANLeaf.

7.3 Partie 2 : Un deuxième nœud de réception

Envoyez le programme de réception CAN_Example_01_Receive.fcfx au nœud 1. Maintenant, lorsque l'interrupteur D0 est actionné, la même LED s'allume sur le nœud 1 et le nœud 3.

Modifiez ensuite le programme pour allumer la LED 1 à la place. Sauvegardez le programme sous CAN_Example_01_Receive_A.fcfx et envoyez le programme au nœud 3. Maintenant, lorsque l'interrupteur 0 est pressé, des LED différentes s'allument sur le nœud 1 et le nœud 3.

7.4 Conclusions

Cet exemple démontre que l'envoi et la réception ne sont pas seulement des actes distincts, mais aussi qu'ils sont totalement indépendants. En fonction des programmes envoyés aux différents nœuds récepteurs, chaque nœud pourrait répondre au signal d'une manière totalement différente.

7.5 Autres travaux

- Considérez ce qui se passerait si, dans la partie 2, vous chargiez un programme qui enverrait un signal si l'interrupteur 1 était actionné. Comment les autres nœuds seraient-ils affectés ? Modifiez le programme d'envoi pour envoyer le signal lorsqu'un interrupteur du port A est actionné et envoyez le programme au nœud 1 pour voir ce qui se passe.

- Si vous utilisez CANKing pour visualiser le trafic réseau, vous remarquerez le nombre de messages et leur fréquence d'envoi. Demandez-vous s'il n'y a pas une meilleure façon de procéder pour réduire le trafic sur le réseau. Modifiez les programmes d'envoi et de réception en conséquence pour voir si vous pouvez réduire le trafic réseau généré.

8 Démonstration 1 : Freinez !!!

Cet exemple montre un simple système signal-réponse en action, le système CAN le plus basique possible.

8.1 Mise en place

	PIC BL0011			Arduino BL0055		
	Port A	Port B	Port C	A0-5 (Port C)	D0-7 (Port D)	D8-13 (Port B)
Nœud 1	BL0167		BL0140	BL0167		BL0140
Nœud 2	BL0145		BL0140	BL0145		BL0140
Nœud 3	BL0167		BL0140	BL0167		BL0140

- Interrupteur 0 sur le nœud 2 pour imiter l'action de la pédale de frein.
- LED 0 sur le nœud 3 pour imiter l'action du feu de freinage arrière.
- LED 0 sur le nœud 1 pour imiter l'action du signal du feu de freinage du tableau de bord.

Ouvrez le fichier *CAN_Example_01_Send.fcfx* dans Flowcode et téléchargez-le vers le nœud 2 (le nœud Switches). Ouvrez le fichier *CAN_Example_01_Recieve.fcfx* dans Flowcode et téléchargez-le vers le nœud 3 (le nœud LED).

8.2 Visualisation des messages

Si vous ouvrez CANKing et consultez le trafic réseau, vous verrez qu'un message est envoyé chaque fois que vous appuyez sur la pédale de frein.

8.3 Partie 1 - Le feu stop

Lorsque la pédale de frein est enfoncée (interrupteur 0 sur le nœud 2), le voyant de frein (LED 0 sur le nœud 3) s'allume. Le signal généré par le nœud 2 (les interrupteurs) est capté par le nœud 3 (les DEL) et, comme tous les messages sont acceptés, le message est activé par l'allumage de la DEL.

8.4 Partie 2 - L'affichage du tableau de bord

Téléchargez ensuite le programme CAN_Example_01_Recieve.fcfx sur le nœud 1.
Lorsque l'on appuie sur la pédale de frein, les DEL des nœuds 1 et 3 s'allument.
Nous n'avons pas modifié le signal envoyé de quelque manière que ce soit, mais les deux nœuds récepteurs reçoivent le signal et agissent en conséquence.

Téléchargez ensuite le programme CAN_Example_01_Recieve_A.fcfx vers le nœud 3.
Cette fois, la même LED s'allume sur le nœud 1, mais une LED différente (1)
s'allume sur le nœud 3. Le même signal est envoyé, mais il est traité différemment
par les deux nœuds récepteurs.

8.5 Conclusions

Cet exemple démontre que l'envoi et la réception ne sont pas seulement des actes distincts, mais aussi qu'ils sont totalement dépendants l'un de l'autre. En fonction des programmes envoyés aux différents nœuds récepteurs, chaque nœud pourrait répondre au signal d'une manière totalement différente.

8.6 Autres travaux

- Considérez ce qui se passerait si, dans la partie 2, vous chargiez un programme qui enverrait un signal si l'interrupteur A1 était actionné. Comment les autres nœuds seraient-ils affectés ?
- Si vous utilisez CANKing pour visualiser le trafic réseau, vous remarquerez le nombre de messages et leur fréquence d'envoi. Vous pouvez vous demander s'il n'y a pas une meilleure façon de procéder pour réduire le trafic sur le réseau.

9 Recherche d'erreurs dans les systèmes CAN

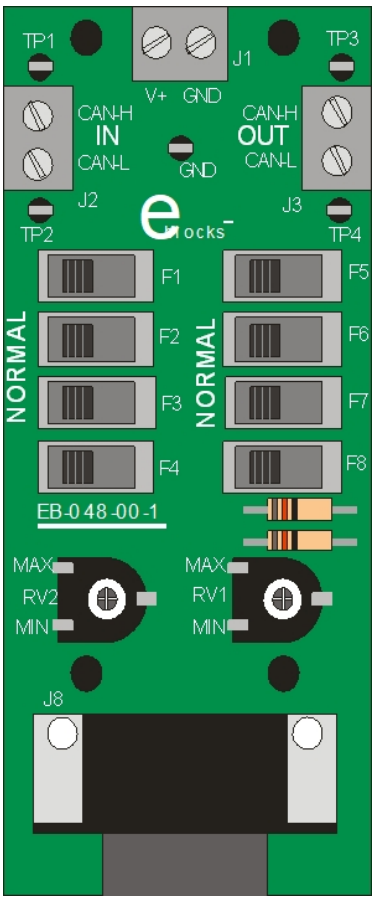
Cet exercice vous permettra de comprendre comment trouver des défauts dans les systèmes de bus CAN.

9.1 Mise en place

Connectez et mettez sous tension la solution CAN. Nous utiliserons le nœud 1, le nœud d'affichage, pour afficher les données. Nous utiliserons le nœud 4, le nœud capteur, pour envoyer les données.

Ouvrez le fichier *CAN_EXAMPLE_04_RECEIVE.FCFX* en Flowcode et téléchargez-le vers le nœud 1 (le nœud d'affichage). Ouvrez le fichier *CAN_EXAMPLE_04_SEND.FCFX* dans Flowcode et téléchargez-le vers le nœud 4 (le nœud capteur).

La carte de défauts CAN est placée entre les nœuds 3 et 4 du système. Tout défaut dans le bus CAN entraîne l'impossibilité de transmettre les données.



Voici un schéma de la carte des défauts. Les bornes à vis situées sur la partie supérieure correspondent aux marchés IN et OUT. Bien entendu, les signaux du bus CAN circulent dans les deux sens, mais nous utiliserons IN et OUT pour des raisons de commodité. Notez que la prise de l'analyseur CAN est connectée aux lignes CAN IN. Assurez-vous que tous les interrupteurs sont en position NORMALE. Dans cette position, aucun défaut n'est inséré. Vous introduirez des défauts dans le système entre les nœuds 3 et 4. Dans les programmes de l'exemple, les signaux sont envoyés du nœud 4 au nœud 1 : cela signifie que l'analyseur se trouve du côté non défectueux du bus CAN.

TP1 et TP2 sont connectés aux signaux CAN-H et CAN-L "avant" un défaut. TP3 et TP4 sont connectés aux signaux CAN-H et CAN-L "après" un défaut.

9.2 Visualisation des messages

Si vous ouvrez CANKing et visualisez le trafic réseau, vous verrez que des messages sont envoyés à intervalles réguliers. Si vous faites varier le potentiomètre sur la carte du capteur du nœud 4, vous verrez sur l'écran de l'analyseur que la partie "données" du message est liée au réglage du potentiomètre. Dans ce cas, la valeur du potentiomètre imite le capteur de carburant d'un réservoir d'essence. La variation du potentiomètre entraîne une modification de l'affichage sur l'écran LCD du nœud 1 en indiquant la quantité de carburant dans le réservoir.

- Connectez un oscilloscope entre GND et TP1. Vous devriez pouvoir voir le signal du bus CAN.
- Connectez un oscilloscope à mémoire entre GND et TP2. Quelle est la différence avec la première mesure ?
- Faites un dessin (approximatif) de chaque signal ou imprimez-le. Notez la chronologie et les niveaux de tension afin de pouvoir vous référer à ces dessins ultérieurement.

Retirez les fils CAN-L et CAN-H du connecteur à vis 'IN'. L'écran LCD doit s'éteindre. Il s'agit d'une condition de défaut : si l'écran n'affiche rien, c'est qu'il y a un défaut dans le système. Notez que dans des conditions de défaut de circuit ouvert partiel, votre analyseur de bus CAN peut toujours afficher des données sur le bus même si l'écran LCD est vide : la raison en est que l'interface CAN sur le nœud 1 a un circuit d'interface différent de celui de l'analyseur CAN, qui est plus sensible.

9.3 Partie 1 - F1

- Placez le commutateur F1 en position droite.
- Qu'advient-il de l'écran LCD ?
- S'agit-il d'une faute ?
- Utilisez l'oscilloscope pour visualiser les signaux sur TP 1 à TP4. Que voyez-vous ?
- Coupez l'alimentation de tous les nœuds. Utilisez un multimètre pour mesurer la résistance entre les nœuds et complétez le tableau ci-dessous.
- Quel type de faute la F1 représente-t-elle ?

	TP1	**TP2**	**TP3**	**TP4**
Résistance à la terre				
Résistance à +V				
TP1	-			
TP2		-		
TP3			-	
TP4				-

9.4 Partie 2 - F2, F3, F5, F6, F7

Remettez l'interrupteur F1 en position NORMALE. Répétez l'exercice de la section ci-dessus individuellement pour les interrupteurs de défaut F2, F3, F5, F6, F7. Assurez-vous alors que tous les interrupteurs sont en position NORMALE, à l'exception d'un seul. Vous devriez maintenant avoir une idée claire du défaut que chaque interrupteur introduit dans le système.

9.5 Circuits partiellement ouverts

F4 et F8 sont utilisés pour insérer des circuits partiellement ouverts dans chaque ligne du bus CAN. Placez tous les interrupteurs en position NORMALE. Placez le commutateur F4 en position de défaut à droite. À l'aide du potentiomètre, faites varier la résistance de défaut jusqu'à ce que le système s'arrête de fonctionner, c'est-à-dire jusqu'à ce que l'écran LCD du nœud 1 devienne vide. Coupez l'alimentation. Complétez à nouveau le tableau ci-dessus et notez la résistance de défaut. Refaites cet exercice pour le commutateur F8. La résistance à laquelle les lignes CAN-H et CAN-L introduisent un défaut varie-t-elle ?

10 Réseau CAN intermédiaire

Nous allons ici discuter et expliquer les principales caractéristiques du réseau CAN, et fournir des exemples et des suggestions de projets pour vous permettre de vous entraîner. À la fin de cette section, vous serez en mesure de mettre en place et de faire fonctionner un réseau CAN à nœuds multiples qui peut répondre à une variété de messages. Il y a beaucoup de choses à apprendre dans cette section et il se peut que vous ayez besoin de revenir à plusieurs reprises sur les différentes parties de cette section pour réviser ce que vous avez appris.

11 Le composant CAN

Nous nous pencherons ici sur les ID de message et sur la manière dont nous pouvons sélectionner les messages auxquels nous répondons. Nous étudierons également l'envoi et la réception de données. Nous examinerons les paramètres généraux et la manière dont ils affectent le réseau.

Pour pouvoir travailler avec les messages ID, nous devons comprendre les panneaux de propriétés du composant CAN. Les différents onglets des propriétés du composant CAN détaillent les paramètres généraux et les paramètres par défaut des tampons d'émission TX et de réception RX.

11.1 Paramètres généraux

L'onglet des propriétés des paramètres généraux contient les principales propriétés des paramètres du réseau CAN.

Taux de bus Le taux de bus est la vitesse de connexion du nœud. Le nœud s'attend à ce que tous les messages arrivent à cette vitesse. Les messages arrivant à un rythme trop rapide ou trop lent peuvent être échantillonnés de manière incorrecte, ce qui entraîne des messages erronés ou mélangés. Tous les nœuds d'un réseau doivent être réglés sur la même vitesse de bus, sans quoi des problèmes de communication se posent.

Point d'échantillonnage Le point d'échantillonnage est le point de l'impulsion du signal attendu auquel le signal est mesuré pour déterminer s'il s'agit d'un 1 ou d'un 0. Ce point est normalement fixé à 50-80% de la période d'impulsion du signal.

Largeur de saut de synchronisation - SJW La largeur de saut de synchronisation est utilisée pour aider à synchroniser les nœuds CAN. Comme CAN n'utilise pas d'horloge, il doit se synchroniser avec les nœuds émetteurs. SJW est une variable qui permet de définir la marge de manœuvre maximale autorisée pour la synchronisation. Elle est utilisée pour faciliter le transfert de données entre les nœuds de données sur des câbles CAN exceptionnellement longs et peut être laissée telle quelle pour la plupart des réseaux de nœuds.
Les paramètres par défaut du composant CAN sont les suivants : Channel = External, Bus Rate = 500, Sample Point = 60%, Sync Jump Width = 1, ID Type = Standard Only. Pour utiliser l'analyseur CAN livré avec certains de nos systèmes CAN, il faut s'assurer que les réglages sont les mêmes pour le composant CAN et l'analyseur CAN, car cela est nécessaire pour qu'ils fonctionnent correctement.

Remarque : Modifiez la "vitesse de bus" de "500" à "125" pour l'adapter aux paramètres requis pour les exercices.

11.2 Tampons de transmission

Notre implémentation de la couche physique CAN comporte trois tampons d'émission - TX Buffer 0, TX Buffer 1 et TX Buffer 2. Chaque tampon contient un ID de message et un lot de données. Les trois tampons fonctionnent de la même manière. Les tampons sont utilisés pour stocker le message CAN jusqu'à ce qu'il soit prêt à être envoyé. La mémoire tampon peut être modifiée et changée et n'est pas fixe. Les valeurs par défaut des panneaux de propriétés sont utilisées à moins que des changements programmatiques ne soient effectués, auquel cas les valeurs modifiées sont utilisées.

Utilisez le panneau des propriétés pour définir les ID de message et les données pour chacun des trois tampons, ce qui vous permet d'envoyer trois messages prédéfinis. Nous verrons plus tard comment modifier ces propriétés à la volée, mais pour l'instant, nous utiliserons le panneau des propriétés pour configurer les messages.

11.2.1 ID du message

Lorsqu'un message est placé sur le réseau, les nœuds doivent savoir s'ils doivent réagir au message ou non. Dans l'exemple de base, nous avons simplement réagi à la présence d'un message. Cependant, nous pouvons être sélectifs. Chaque message envoyé sur le réseau possède un numéro d'identification de message. Les nœuds peuvent vérifier ce numéro d'identification de message pour voir s'il correspond à la liste des numéros d'identification qu'ils doivent accepter. Si c'est le cas, le nœud peut réagir au message, sinon le message peut être ignoré.

Si vous regardez le panneau des propriétés (voir Fig. 10.1) et trouvez la section TX Buffer 0, vous verrez une propriété Message ID. Il s'agit de la valeur d'identification du message envoyée avec le tampon 0.

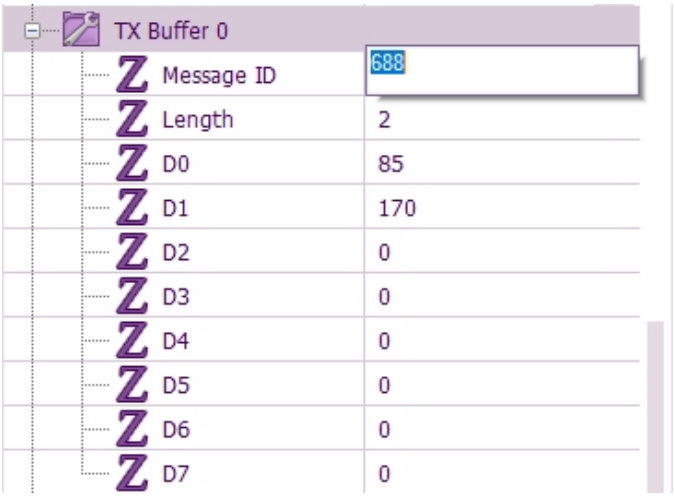


Figure 11.1 *Définition de l'identifiant du message dans le panneau des propriétés*

Les tampons TX 1 et TX 2 disposent également de cases identiques dans lesquelles vous pouvez indiquer leur ID de message. Cela signifie que nous pouvons générer des messages avec trois ID de message distincts à partir de n'importe quel nœud en utilisant uniquement les propriétés de la mémoire tampon par défaut. Cependant, il peut y avoir plusieurs nœuds sur un réseau, chacun d'entre eux pouvant transmettre des signaux avec des ID de message différents - ou même les mêmes ID de message, selon le système. Vous pouvez donc commencer à comprendre l'énorme quantité de messages potentiels, chacun avec son propre ID de message, qui peuvent être envoyés. Plus tard, nous verrons comment modifier l'ID de message du tampon de manière programmatique, ce qui nous donnera encore plus de flexibilité que les trois valeurs par défaut.

11.3 Tampons de réception

Dans notre implémentation de la couche physique CAN, il existe deux tampons de réception, RX Buffer 0 et RX Buffer 1, qui stockent les données reçues. Il est possible de vérifier s'ils ont reçu des messages.

Les tampons de réception peuvent être l'une des parties les plus complexes à utiliser et à comprendre de notre implémentation de CAN. Les propriétés des filtres sont similaires à celles des messages ID, car les tampons y réagissent. Le mode avancé sera abordé plus loin dans la section sur les réseaux CAN avancés.

Il y a 6 filtres au total répartis entre deux tampons RX (0 et 1). Le tampon RX dispose de 2 filtres (filtres 0 à 1) et le tampon RX dispose de 4 filtres (filtres 2 à 5).

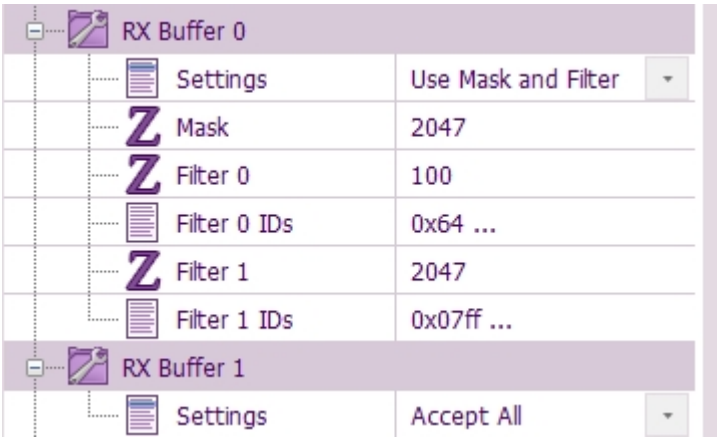


Figure 11.2 *Filtres et propriétés du tampon RX*

Vous pouvez configurer le mode simple en cliquant sur la case à cocher "Paramètres simples". Vous pouvez ensuite entrer des valeurs d'ID de message dans les cases. Les messages portant ces ID seront acceptés par le tampon RX. Une fois qu'un message est arrivé, nous pouvons le vérifier avec la macro *CheckRX(Buffer)*. Le paramètre buffer est utilisé pour indiquer quel tampon RX est vérifié (0 pour le tampon RX 0, et 1 pour le tampon RX 1). Une valeur de retour non nulle indique qu'un message est arrivé.

12 Travailler avec les ID de message

Les ID de messages sont complexes à comprendre en raison des mathématiques qu'ils impliquent, bien qu'une sélection minutieuse des ID Mes- sage puisse simplifier considérablement le problème. Il y a deux parties à considérer - comment lire et différencier les ID Mes- sage et comment changer les ID prédéfinis pour les messages sortants.

12.1 Vérification des identifiants des messages

Vous pouvez vérifier la présence de messages à l'aide de *CheckRx(Buffer)*, qui renvoie une valeur non nulle lorsqu'un message est reçu. Vous pouvez interroger un message pour connaître l'ID du message. Cependant, les ID de message peuvent aller jusqu'à une valeur de 2047 (hex 0x7FF), alors que Flowcode et le microcontrôleur n'utilisent que des valeurs allant jusqu'à 255 (0xFF). La gamme des valeurs possibles de Message ID est supérieure au plus grand nombre que le microcontrôleur peut gérer. Pour contourner ce problème, le message est divisé en deux octets, un octet *hi* et un octet *lo*. Les deux macro-fonctions *GetRxIDHi* et *GetRxIDLo* permettent d'y accéder. Ces deux fonctions prennent le paramètre *Buffer* pour sélectionner le tampon RX à partir duquel les données d'identification du message doivent être obtenues.

Le problème est encore compliqué par le fait que l'ID du message est un nombre de 11 bits dont les 8 premiers bits les plus significatifs forment l'octet hi et les 3 bits les moins significatifs forment les trois premiers bits les plus significatifs de l'octet lo (voir la figure 11.1). Cela peut rendre les mathématiques de la vérification manuelle des valeurs d'ID de message assez complexes. La méthode la plus simple consiste à comparer les valeurs extraites avec les valeurs connues pour hi et lo pour voir si elles correspondent.

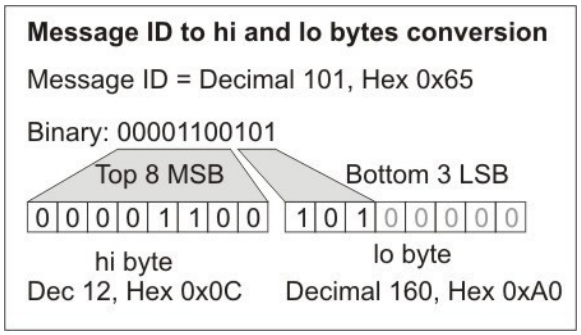


Figure 12.1 *Conversion des octets hi et lo de l'ID du message*

12.1.1 Conversion de valeurs 8 bits en valeurs 11 bits

À un moment donné, vous aurez peut-être besoin de savoir comment obtenir la valeur complète de 11 bits à partir des deux octets High et Low de 8 bits. Les mathématiques sont les suivantes :

Message ID = (hi x 0x08) + (lo / 0x20)

12.2 Identification manuelle des messages - une recommandation

En raison de la complexité accrue du travail avec les valeurs *hi* et *lo du* message ID, nous suggérons, dans le cadre d'une utilisation pédagogique, en particulier lorsque vous commencez à travailler avec CAN, d'utiliser un système dans lequel l'un des octets (probablement l'octet lo) reste le même, tandis que l'autre octet (l'octet hi) est modifié. Cela simplifie considérablement les mathématiques tout en vous permettant d'accéder à 256 messages ID différents.

Remarque : comme nous créons des nœuds d'envoi et de réception, nous pouvons choisir des ID de message qui nous facilitent la vie. Toutefois, si vous ajoutez un nœud à un système existant, vous devrez peut-être travailler avec des ID de message déjà définis. Cela peut signifier que les octets hi et lo doivent être vérifiés. Il se peut également que vous deviez tester le réseau pour vous assurer que vos ID de message n'affectent pas d'autres nœuds par inadvertance.

13 Exercice 2 : faisceau lumineux arrière

13.1 Partie A : Envoi

13.1.1 Objectif

Mettez en place une série d'interrupteurs pour activer un feu de freinage, un feu clignotant et un feu arrière.

13.1.2 Instructions

Les trois feux ont reçu les numéros d'identification suivants :

Frein=	ID 8
Feu	arrière= ID 16
	Indicateurs= ID 32

Les interrupteurs d'activation sont les suivants :

Frein=	Interrupteur 0 -
FreinFeu arrière	Interrupteur 1 - Feu arrière
	Indicateurs= Commutateur 2 - Indicateur gauche

Le nœud 2 (nœud de commutation) sera utilisé pour envoyer les signaux.

13.2 Partie B : Réception

13.2.1 Objectif

Créez un affichage de base pour un groupe de feux arrière de voiture contenant un feu stop, un clignotant et un feu arrière.

13.2.2 Instructions

Les trois feux ont reçu les numéros d'identification suivants :

Frein=	ID 8
Feu	arrière= ID 16
Indicateurs=	ID 32

Les voyants de l'écran sont les

suivants :

Frein=	LED 0 - FreinFeu
arrière	LED 1 - Feu arrière
	Indicateurs= LED 2 - Indicateur gauche

Le nœud 3 (nœud de DEL) sera utilisé pour afficher les signaux.

Les indicateurs doivent clignoter, si possible, à raison d'une seconde d'allumage et d'une seconde d'extinction.

13.3 Autres travaux

- Nous avons configuré le programme pour un faisceau de feux arrière GAUCHE. Réfléchissez aux modifications que nous devrions être en mesure d'apporter pour créer un groupe de feux arrière DROIT.
- Est-il possible de mettre en place un groupe de feux avant ? Dans l'affirmative, quels messages seraient identiques ? De quels messages supplémentaires aurions-nous besoin ?
- L'idéal serait d'avoir un groupe arrière gauche et un groupe arrière droit. Quelles modifications faudrait-il apporter au système CAN pour nous permettre de réaliser cette opération ?

14 Notes pour l'exercice 2

14.1 Partie A : Envoi

Nous avons trois signaux à envoyer et, par coïncidence, nous disposons de trois tampons pour envoyer les signaux. Si nous installons une lumière pour chaque tampon, nous pouvons envoyer les trois signaux en utilisant les trois tampons.

Le programme de base est le même que celui du feu de freinage de l'exemple 1, mais avec trois sections d'interrupteurs de contrôle/signaux d'envoi. Configurez chaque section pour qu'elle utilise un tampon différent, et configurez chaque tampon avec un ID de message différent. C'est ce dont nous allons nous occuper par la suite.

Ouvrez le panneau des propriétés et définissez l'ID du message pour les trois tampons TX comme suit :

Tampon	ID du message	Fonction
Tampon TX 0	8	Frein
Tampon TX 1	16	Feux arrière
Tampon TX 2	32	Indicateurs

14.2 Partie B : Réception

Dans la partie A, nous avons configuré un nœud pour envoyer trois messages possibles avec les ID de message 8, 16 et 32. Nous devons maintenant configurer un nœud capable d'accepter ces trois ID de message.
Si nous ouvrons les propriétés et regardons le tampon RX 0, nous verrons que nous pouvons accepter jusqu'à 2 ID de message pour ce tampon. Si nous vérifions le tampon Rx, nous verrons que le tampon RX 1 a 4 ID de message que nous pouvons configurer. Nous avons trois messages entrants possibles, nous utilisons donc le tampon RX 1 pour les recevoir tous.

Réglez les trois premières cases ID du tampon RX 1 sur Message ID 8, Message ID 16 et Message ID 32.
Désélectionnez les autres cases pour qu'elles n'acceptent pas d'autres ID de message.
Désormais, nous recevrons une réponse dans le tampon RX 1 chaque fois que l'un de ces trois messages ID sera envoyé.

Nous pouvons utiliser une simple macro *CheckRx* pour vérifier si un message est arrivé. Cependant, trois messages peuvent déclencher le RX Buffer 1, et nous devons donc les distinguer.
A ce stade, il vous faudra probablement prendre un stylo et du papier pour déterminer les valeurs des octets hi et lo pour les deux ID de message. Heureusement, nous l'avons fait pour vous.

ID du message	Bonjour octet	Octet de poids faible
8	1	0
16	2	0
32	4	0

Vous comprenez maintenant pourquoi nous avons choisi une séquence d'ID de message aussi étrange. L'octet lo est le même pour chaque message, il suffit donc de tester l'octet hi pour savoir de quel message il s'agit.
Une approche plus sophistiquée consisterait à tester d'abord l'octet lo pour s'assurer qu'il vaut 0, comme prévu, puis à tester l'octet hi pour voir de quel message il s'agit.

Si nous avions choisi des valeurs telles que 101, 149, 150, les valeurs auraient été les suivantes :

ID du message	Bonjour octet	Octet de poids faible
101	12	160
149	18	160
150	18	192

Et nous devrions vérifier les octets hi et lo pour les valeurs concernées.

Dans notre programme d'envoi, nous avons défini les ID de message 8, 16 et 32. Nous pouvons maintenant vérifier quel message a été envoyé
et la traiter en conséquence.

ID du message	Valeur de l'octet Hi	Message	Sortie
8	1	Freins	D0
16	2	Lumières	D1
32	4	Indicateurs	D2

Ainsi, si nous recevons le message ID 16, par exemple, nous activons la sortie D1.

Le fragment de code de la figure 13.1 montre comment nous pouvons vérifier la présence d'un message, puis récupérer les valeurs d'identification de l'ID du message (dans ce cas, seul l'octet Hi de l'ID du message est nécessaire) et les vérifier pour savoir quel message a été envoyé.

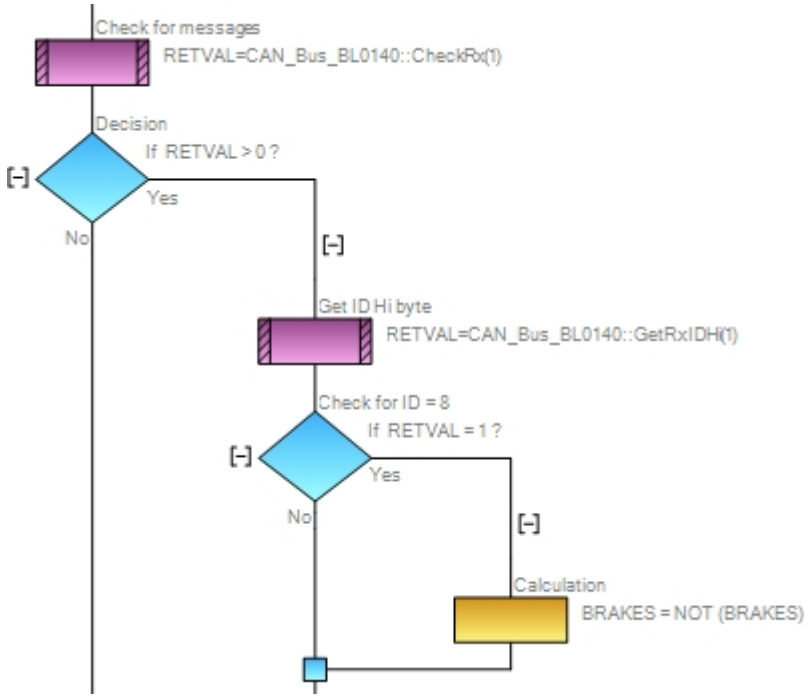


Figure 14.1 *Fragment de code montrant comment récupérer l'identifiant du message*

14.2.1 Quel tampon utiliser ?

Nous devons accepter trois valeurs d'ID de message, mais il n'y a que 2 emplacements d'ID de message dans le tampon RX 0. Cependant, le tampon RX 1 dispose de 4 emplacements pour les valeurs d'ID de message. Dans l'exemple ci-dessus, nous avons configuré les trois messages sur le tampon RX 1 et nous n'avons utilisé que ce tampon. Cependant, nous aurions pu utiliser une combinaison des deux tampons si nous l'avions souhaité.

14.3 Indicateurs

La présence d'indicateurs clignotants est facultative car elle ne concerne pas le réseau CAN, mais c'est un exercice utile qui améliore l'attrait visuel de l'affichage final.

14.4 Conclusion

Nous avons maintenant un système qui fonctionne. Il peut envoyer différents signaux et les différencier. Cela nous permet de construire des systèmes assez complexes, capables d'envoyer de nombreux messages différents à partir de nombreux nœuds différents, et d'être sélectifs et de réagir à certains des messages potentiels du système, voire à aucun d'entre eux.

14.5 ERREUR MAJEURE !!! - Le frein est-il activé ?

Il y a un problème avec le système. Nous avons un signal de freinage que nous utilisons pour allumer ou éteindre le feu stop. Mais que se passerait-il si le système manquait un signal, ou si le frein était allumé alors qu'il devrait être éteint ? Le signal serait inversé. Le feu stop s'allumerait alors que le frein est éteint et non allumé. Ce n'est pas une bonne chose.

Nous sommes coincés dans cette situation parce que nous ne pouvons envoyer que trois signaux et que les trois sont nécessaires pour des tâches différentes. De plus, le signal envoyé pour le frein ne nous indique pas si le frein est activé ou désactivé, mais simplement qu'un signal de frein a été envoyé. Ce que nous voudrions faire, c'est soit envoyer des signaux distincts pour le frein activé et le frein désactivé, ce qui nécessiterait plus d'ID de message que nous n'en avons actuellement, soit la possibilité d'envoyer des données avec le message pour dire si le frein est activé ou désactivé. Ce sont précisément les problèmes que nous allons aborder dans la suite de cet article.

Autres travaux

Les travaux ultérieurs comprennent des questions pratiques simples et une question théorique.

Pour créer un groupe d'éclairage droit, il suffirait de remplacer le clignotant gauche par un ID de message et un commutateur d'activation différents. De la même manière, le groupe d'éclairage avant devrait ignorer le message relatif au feu de freinage puisqu'il n'y a pas de feu de freinage à cet endroit. Mais il faudrait lui envoyer un signal Dip à la place.

La mise en place d'un système gauche-droite (ou d'un système avant-arrière complet si l'on pousse la réflexion jusqu'au bout) est théorique, car elle nécessite des macros qui n'ont pas encore été introduites. La mise en place d'un système gauche-droite nécessiterait soit un quatrième Message ID, soit une sorte de données indiquant quel indicateur utiliser.

15 Démonstration 2 : Groupe de feux arrière

Cet exemple montre un système de messages CAN en action.

15.1 Mise en place

	PIC BL0011			Arduino BL0055		
	Port A	Port B	Port C	A0-5 (Port C)	D0-7	D8-13 (Port B)
Nœud 1	BL0145		BL0140	BL0145		BL0140
Nœud 2	BL0145		BL0140	BL0145		BL0140
Nœud 3	BL0167		BL0140	BL0167		BL0140

Connectez et mettez sous tension la solution CAN. Nous utiliserons les interrupteurs 0-2 sur le nœud 2 pour imiter les signaux d'action d'activation. Nous utiliserons les DEL 0-2 sur le nœud 3 pour imiter le groupe de feux arrière.

Ouvrez le fichier CAN_EXAMPLE_02_SEND.FCFX dans Flowcode et téléchargez-le vers le nœud 2 (le nœud Switches). Ouvrez le fichier CAN_EXAMPLE_02_RECEIVE.FCFX dans Flowcode et téléchargez-le vers le nœud 3 (le nœud LED).

15.2 Visualisation des messages

Si vous ouvrez CANKing et visualisez le trafic réseau, vous verrez qu'un message est envoyé chaque fois qu'un des interrupteurs est actionné.

15.3 Le faisceau lumineux

Lorsque la pédale de frein est enfoncée (interrupteur 0 sur le nœud 2), le voyant de frein (LED 0 sur le nœud 3) s'allume. Le signal généré par le nœud 2 (les interrupteurs) est capté par le nœud 3 (les DEL) et, comme tous les messages sont acceptés, le message est activé par l'allumage de la DEL.
De la même manière, le commutateur des feux (commutateur 1) active le feu arrière (LED 1), et le commutateur de l'indicateur gauche (2) active l'indicateur gauche (LED 2).

15.4 Les messages

Les trois feux ont reçu les numéros d'identification suivants :

Frein= ID 8
Feu arrière= ID 16
Indicateurs= ID 32

Observez dans CANKing l'envoi des messages et la façon dont l'ID du message vous indique quelle lumière sera activée.

15.5 Autre trafic réseau

Ouvrez le fichier *CAN_EXAMPLE_02_RANDOM_SEND.FCFX* dans Flowcode et téléchargez-le vers le nœud 1 (le nœud d'affichage).
Les interrupteurs du nœud d'affichage ont été réglés pour envoyer divers messages. Appuyez sur ces boutons pour voir ce qui se passe. Si quelque chose d'important se produit, vérifiez avec CANKing pour voir si vous pouvez en trouver la raison.

15.6 Conclusions

Cette démonstration montre les ID de message en action. Les événements peuvent être liés à des ID de message spécifiques, de sorte qu'ils peuvent se trouver sur un réseau avec beaucoup de trafic et qu'ils ne répondront qu'au signal correct, et non au trafic aléatoire comme dans la démonstration 1.

16 Notes pour la démonstration 2

Les premiers fichiers de démonstration *CAN_EXAMPLE_02_SEND.FCFX* et *CAN_EXAMPLE_02_RECEIVE.FCFX* sont basés sur ceux créés pour l'exercice 2.

L e fichier final *CAN_EXAMPLE_02_RANDOM_SEND.FCFX* envoyé au nœud Display envoie différents signaux. Il faut noter le signal pour le commutateur D4 qui a l'ID de message 32. Il s'agit du même ID de message que le signal d'indicateur. Comme il a le même ID de message, il déclenchera les indicateurs sur le nœud de réception.
Ceci illustre un problème potentiel sur les réseaux CAN - celui des conflits d'ID de message. Plusieurs nœuds peuvent générer un message avec le même ID de message, ce qui déclenche par inadvertance le nœud récepteur. Cet exercice peut être utilisé à la fin de l'exercice 2 pour illustrer le même problème potentiel.

17 Modification des identifiants des messages

Jusqu'à présent, nous avons utilisé les ID des messages tels qu'ils sont définis dans les panneaux de propriétés. Cependant, vous pouvez modifier l'ID à l'aide de la macro *SetTxID*. Cela ouvre un tout nouveau champ de possibilités. En étant capable de changer les ID de message à la volée, vous pouvez réagir aux entrées en envoyant des messages différents en fonction des données reçues. Par exemple, le message qui déclenche le clignotement d'un feu d'avertissement peut être remplacé par un message qui émet également un bruit d'avertissement après un certain temps.

Il y a cependant un problème : les ID de message ont une longueur de 11 bits, mais les microcontrôleurs ne peuvent traiter que des nombres de 8 bits. La valeur de l'ID du message doit donc être divisée en deux octets de 8 bits distincts, un octet hi et un octet lo, comme le montre la figure 16.1.

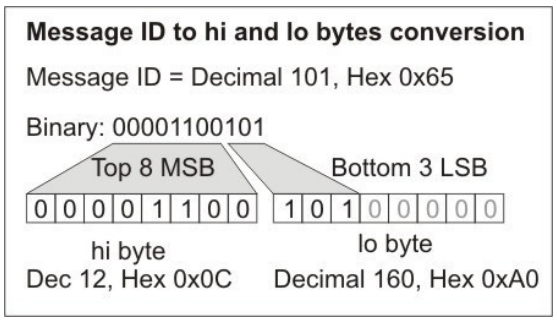


Figure 17.1 *Conversion des octets hi et lo de l'ID du message*

17.1.1 La macro SetTxID

La macro SetTxID prend trois paramètres : buffer, hi et lo. Buffer est le numéro de la mémoire tampon TX, de 0 à 2. Les deux autres paramètres, hi et lo, définissent la valeur de l'ID du message.

Le problème est que Flowcode et le microcontrôleur ne peuvent gérer que des nombres allant jusqu'à 255 (hex 0xFF), alors que la valeur du Message ID peut aller jusqu'à 2047 (hex 0x7FF). Nous devons donc diviser la valeur de l'ID du message en deux octets distincts. L'ID du message a une longueur de 11 bits et nous devons le convertir en deux octets de 8 bits.

17.1.2 Conversion de valeurs de 11 bits en valeurs de 8 bits

Les deux octets sont présentés comme suit :
Les 8 bits les plus significatifs (les 8 premiers) de l'ID sont placés dans l'octet hi.
Les 3 bits de poids faible (les 3 derniers bits) sont placés dans l'octet de poids faible - mais dans les 3 positions de bits de poids fort (c'est-à-dire les 3 premiers).

La conversion de l'ID complet du message en deux octets hi et lo se fait de la manière suivante :

hi = (Message_ID AND 0x7F8) / 8
lo = (Message_ID AND 0x007) * 0x20

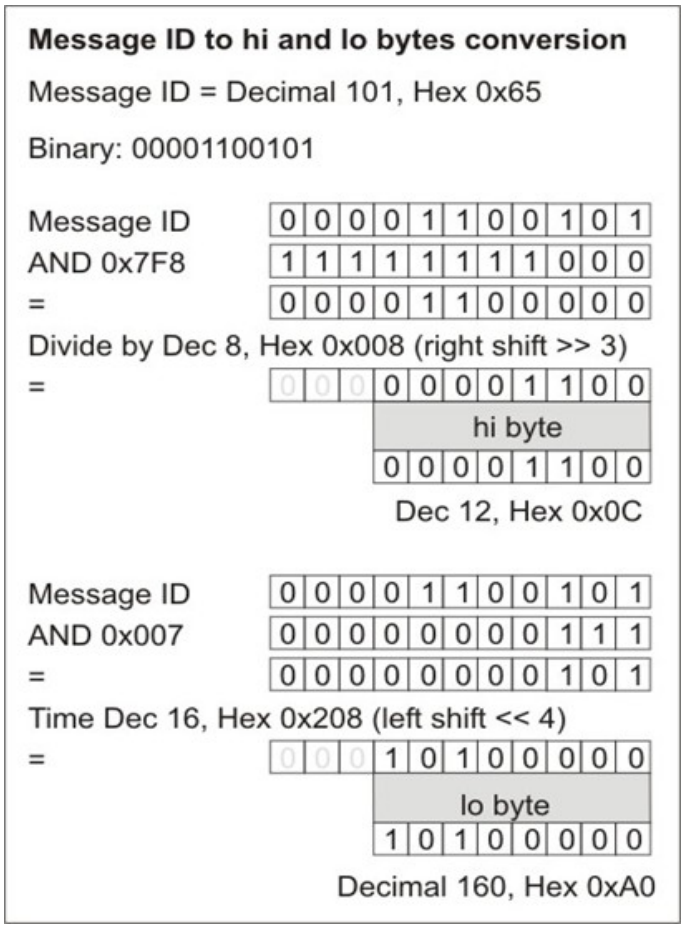


Figure 17.2 *Conversion des octets hi et lo de l'ID du message*

La méthode la plus simple consiste à fixer les valeurs hi et lo à des valeurs prédéterminées. Lorsque vous choisissez les valeurs d'ID de message, vous devez vous assurer que vous n'utilisez pas les mêmes ID de message que ceux utilisés par d'autres nœuds de transmission, sinon vous risquez de semer le chaos dans le réseau. Vous pouvez également aider le concepteur des nœuds récepteurs en choisissant des valeurs d'ID de message que les nœuds récepteurs peuvent identifier et différencier facilement.

18 Exercice 3 : système d'éclairage arrière

L'exercice 3 étend l'exercice 2 en y incluant les indicateurs gauche et droite. La solution de l'exercice 2 peut servir de point de départ à l'exercice 3.

18.1 Partie A : Envoi

18.1.1 Objectif

Installez une série d'interrupteurs pour activer un feu de freinage, un clignotant gauche et un clignotant droit, ainsi qu'un feu arrière.

18.1.2 Instructions

Les trois feux ont reçu les numéros d'identification suivants :

Frein= ID 8
Feu arrière= ID 16
Indicateur gauche ID32
Indicateur droit ID 64

Les interrupteurs d'activation sont les suivants :

Frein= Commutateur 0 - Frein
Feu arrière Commutateur 1 - Feu arrière Voyant gauche Commutateur 2- Voyant gauche Voyant droit= Commutateur 3 - Voyant droit

Le nœud 2 (nœud de commutation) sera utilisé pour envoyer les signaux.

18.2 Partie B : Réception

18.2.1 Objectif

Créez un groupe de feux arrière gauche et droit contenant un feu stop, un clignotant et un feu arrière. Notez que cela nécessitera deux nœuds distincts avec des programmes distincts mais similaires.
Créez d'abord un nœud et utilisez le programme comme base pour le programme du deuxième nœud.

18.2.2 Instructions

Les trois feux ont reçu les numéros d'identification suivants :

Frein= ID 8
Feu arrière= ID 16
Indicateur gauche = ID 32
Indicateur de droite = ID 64

Les voyants d'affichage sont les suivants :

Frein= LED 0 - Frein
Feu arrière= LED 1 - Feu arrière
Indicateur gauche LED 2 - Indicateur gauche
Indicateur droit LED 3 - Indicateur droit

Le nœud 1 (nœud d'affichage) sera utilisé pour afficher les signaux du faisceau gauche.

Frein= LED 0 - Frein
Feu arrière LED 1 - Feu arrière
Indicateur gauche LED 2 - Indicateur gauche

Le nœud 3 (nœud de DEL) sera utilisé pour afficher les signaux de la grappe de droite.

Frein= LED 0 - Frein
Feu arrière LED 1 - Feu arrièreIndicateur droit LED 3 - Indicateur droit

Les indicateurs doivent clignoter, si possible, à raison d'une seconde d'allumage et d'une seconde d'extinction.

18.3 Autres travaux

- Nous disposons à présent d'un jeu de lumière arrière fonctionnel. Quelles sont les modifications à apporter pour créer un jeu de lumière avant ?

- Est-il possible de créer un système complet d'éclairage avant et arrière ? (Le nœud 1 peut être utilisé pour le groupe de feux avant).

- Nous pouvons utiliser les mêmes signaux pour configurer l'affichage d'un tableau de bord. La seule différence avec un cluster est que le tableau de bord possède des signaux lumineux pour les deux indicateurs.

19 Notes pour l'exercice 3

19.1 Les programmes

A part le fait d'avoir quatre messages, le problème est le même que dans l'exercice 2. Nous avons besoin d'un quatrième message et nous devons donc modifier l'ID du message pour au moins un des tampons TX. Cependant, nous pouvons laisser les deux autres par défaut. La meilleure solution serait de conserver les messages de frein et de feu tels quels, et de modifier l'ID de message de l'indicateur en fonction de ce qu'il est - gauche ou droite.

Nous pouvons utiliser la macro *SetTxID* pour définir les ID de message à la volée. Pour cette macro, nous avons besoin des octets hi et lo à envoyer en tant que paramètres. Les octets hi et lo nécessaires sont listés ci-dessous pour plus de commodité. Notez que nous avons conservé le même système que précédemment en ne changeant que l'octet hi.

ID du message	Bonjour octet	Octet de poids faible
8	1	0
16	2	0
32	4	0
64	8	0

Une fois de plus, il convient de rappeler que nous n'avons pas toujours le luxe de disposer d'un ensemble aussi précis d'identifiants de messages lorsque nous travaillons avec d'autres réseaux.

Les deux nœuds de réception sont simples à réaliser. L'indicateur de gauche est exactement le même que celui de l'exercice 2, et celui de droite n'a besoin que d'être modifié pour accepter l'indicateur de droite Message ID.

19.2 Conclusion

Cet exercice peut sembler anodin, mais il est d'une importance fondamentale. Nous sommes maintenant libérés de la limitation par défaut de trois tampons TX/ID de message. Nous pouvons modifier les ID de message comme bon nous semble et créer ainsi jusqu'à 65536 ID de message potentiels. Notre seul souci est maintenant de nous assurer que nos ID de message n'entrent pas en conflit avec d'autres ID de message sur le réseau, et que nos nœuds de réception sont configurés pour les recevoir. Mais il s'agit là d'une bonne planification, d'un bon codage et d'une bonne documentation.

20 Démonstration 3 : Groupe de feux arrière

Cet exemple montre un système de messages CAN en action.

20.1 Mise en place

	PIC BL0011			Arduino BL0055		
	Port A	Port B	Port C	A0-5 (Port C)	D0-7 (Port D)	D8-13 (Port B)
Nœud 1	BL0145		BL0140	BL0145		BL0140
Nœud 2	BL0145		BL0140	BL0145		BL0140
Nœud 3	BL0167		BL0140	BL0167		BL0140

Connectez et mettez sous tension la solution CAN.
Nous utiliserons les interrupteurs 0-2 sur le nœud 2 pour imiter les signaux d'action d'activation. Nous utiliserons les DEL 0-2 sur le nœud 3 pour imiter le groupe de feux arrière.

Ouvrez le fichier CAN_EXAMPLE_03_SEND.FCFX dans Flowcode et téléchargez-le vers le nœud 2 (le nœud Switches). Ouvrez le fichier CAN_EXAMPLE_03 _LEFT_INDICATOR.FCFX dans Flowcode et téléchargez-le vers le nœud 1 (le nœud Display).
Ouvrez le fichier CAN_EXAMPLE_03 _RIGHT_INDICATOR.FCFX dans Flowcode et téléchargez-le vers le nœud 3 (la LED). nœud).

20.2 Visualisation des messages

Si vous ouvrez CANKing et que vous observez le trafic réseau, vous verrez qu'un message est envoyé chaque fois qu'un des commutateurs est actionné.

20.3 Le faisceau lumineux

Lorsque la pédale de frein est enfoncée (interrupteur 0 sur le nœud 2), les feux de freinage (LED 0 sur le nœud 1 et le nœud 3) s'allument. Le signal généré par le nœud 2 (les interrupteurs) est capté par les nœuds 1 et 3 (les DEL) et, comme les deux nœuds sont réglés pour accepter tous les messages, le message est pris en compte et les DEL s'allument.

De la même manière, une pression sur l'interrupteur des feux (interrupteur 1) active les feux arrière (LED 1 sur le nœud 1 et le nœud 3). Notez que si les cartes de développement utilisent souvent des interrupteurs à poussoir, les applications réelles utiliseront probablement des interrupteurs à bascule pour des éléments tels que les interrupteurs d'éclairage.

Le commutateur du clignotant gauche (2) active le clignotant gauche (LED 2 sur le nœud 1) et le commutateur du clignotant droit (3) active le clignotant droit (LED 3 sur le nœud 3). Toutefois, contrairement aux signaux de freinage et d'éclairage, ces deux signaux de clignotants ne sont acceptés que par des nœuds spécifiques. Ainsi, le signal du clignotant gauche n'est accepté et pris en compte que par le nœud 1, le groupe de feux gauche. De même, la grappe de droite accepte le signal du clignotant droit, alors que la grappe de gauche ne l'accepte pas.

20.4 Les messages

Les quatre feux ont reçu les numéros d'identification suivants
: Frein= ID 8
Feu arrière ID16
Indicateurs de gauche ID
32 Indicateurs de droite= ID
64

Observez dans CANKing l'envoi des messages et la façon dont l'ID du message vous indique quelle lumière sera activée.

20.5 Conclusions

Cette démonstration montre les messages ID en action. Les événements peuvent être liés à des Message ID spécifiques de sorte qu'ils peuvent se trouver sur un réseau avec beaucoup de trafic et qu'ils ne répondront qu'au signal correct, et non à un trafic aléatoire comme dans la première démonstration.

21 Notes pour la démonstration 3

Cet exemple est identique à la démonstration 2, mais avec les deux groupes de feux arrière, ce qui permet aux élèves de voir les différents indicateurs s'allumer.

Vous pouvez utiliser cette démonstration comme alternative à la démonstration 2.

L'envoi du fichier CAN_EXAMPLE_02_RANDOM_SEND.FCFX de la démonstration 2 au nœud 1 vous permettra de montrent comment des messages aléatoires avec les mêmes ID de message peuvent poser des problèmes aux nœuds récepteurs.

22 Données du message

22.1 Propriétés des données par défaut

Les panneaux de propriétés contiennent un ensemble de propriétés permettant d'ajouter des données à un message. Les propriétés sont une propriété de longueur de données et jusqu'à huit octets de données. La propriété Longueur des données définit le nombre d'octets de données que contient le message et peut être comprise entre 0 et 8. Lorsque la valeur est 0, aucune donnée n'est envoyée. Si la valeur est comprise entre 1 et 8, le nombre d'octets de données sera envoyé. En fonction de la valeur définie, certaines ou toutes les cases de données peuvent être grisées. Ces cases grisées ne seront pas utilisées dans le message. Vous pouvez modifier la valeur des cases de données pour définir les valeurs de données par défaut qui seront transmises avec ce tampon. Les trois tampons de transmission fonctionnent de la même manière.

22.2 Modification des données du message

Il existe une macro que vous pouvez utiliser pour modifier les données de votre programme.
La macro *SetTxData* prend les paramètres suivants : *Buffer, Count, d0, d1, d2, d3, d4, d5, d6, d7*.
Buffer fait référence à la mémoire tampon TX à modifier (0, 1 ou 2).
Count définit le nombre d'octets de données à utiliser (0 - 8, 0 étant sans données).
d0 - d7 sont les octets individuels de données.

Notez que les valeurs des 8 octets doivent être ajoutées, car elles sont requises par la macro. Il suffit d'ajouter une valeur pour tous les octets non utilisés (traditionnellement, "0" est utilisé en programmation pour les valeurs qui doivent être fournies mais qui ne sont pas prises en compte). Cette opération doit être effectuée même si le *compte* est fixé à 0, ce qui signifie qu'aucune donnée n'est envoyée.

Exemple 1

SetTxData(0, 4, 255, 128, 32, 56, 0, 0, 0, 0) configure le tampon TX 0 pour envoyer les 4 octets de données 255, 128, 32, 56.

Exemple 2

SetTxData(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) configure le tampon TX 1 pour envoyer 0 octet de données.

22.3 Suivi des données

Lorsqu'un programme utilise *SetTxData* pour modifier les données, il incombe au programmeur de garder la trace de ce que sont maintenant les données. Il incombe également au programmeur de s'assurer que les nouvelles données correctes sont transmises à la mémoire tampon. Les valeurs par défaut du panneau des propriétés correspondent à ce qui se trouve dans la mémoire tampon lorsque le programme démarre. Si vous modifiez les données, elles resteront modifiées jusqu'à ce que vous les modifiiez à nouveau.

22.4 Envoi de données

Chaque fois qu'un tampon est envoyé, les données associées à ce tampon sont envoyées automatiquement. Aucune autre action n'est nécessaire. Si *SetTxData* n'a pas été utilisé pour modifier les données par défaut, les données par défaut spécifiées dans le panneau des propriétés de ce tampon seront envoyées. Si *SetTxData* a été utilisé pour modifier les données, les données actuelles seront envoyées. Notez que le panneau des propriétés n'est pas modifié par *SetTxData*.

Lorsqu'un message est envoyé, la longueur des données est transmise avec les octets de données correspondants. Si la longueur des données est de 5, cinq octets de données seront envoyés. Si elle est de 3, seuls trois octets de données seront envoyés. Ceci est important, car essayer de lire cinq octets de données si seulement trois ont été envoyés causera des problèmes.

22.5 Réception des données du message

La réception des données d'un message se fait en deux temps. Il faut d'abord déterminer la quantité de données envoyées, puis extraire les différents octets de données.

La macro *GetRxDataCount* est utilisée, avec le paramètre buffer, pour obtenir la longueur des données du message dans le tampon RX concerné. Une fois que vous avez la longueur des données, vous pouvez voir si le message contient des données et, le cas échéant, combien d'octets.
Vous pouvez ensuite utiliser ces informations pour extraire les données en toute sécurité à l'aide de la macro *GetRxData*. *GetRxData* prend les pa- ramètres *Buffer* (le tampon RX à utiliser) et *Index* (l'élément à extraire).

L'index est numéroté de 0 à 7, de la même manière que les éléments de données du tampon TX sont nommés de D0 à D7 sur les panneaux de propriétés. L'index 0 est le premier élément, l'index 1 le deuxième, etc. Commencer à 0 plutôt qu'à 1 est une caractéristique courante en programmation, dont il faut tenir compte si vous obtenez des retours de données erronés.

22.6 Considérations relatives à l'ordre des données

Il convient d'être prudent lorsque l'on travaille avec des données, car si l'on modifie l'ordre dans lequel les données sont stockées, il faudra modifier en conséquence la manière dont les données sont extraites. Étant donné que les nœuds sont indépendants les uns des autres, il est préférable de décider d'une stratégie pour les données à un stade précoce de la conception du système. Si davantage de données sont nécessaires, il est souvent plus facile d'ajouter les nouvelles données en tant qu'éléments supplémentaires plutôt que de modifier l'ordre, car cela aura moins d'impact sur les autres nœuds.

Il peut également être plus facile d'utiliser une structure de données préexistante et de se contenter de lire et d'ignorer les éléments qui ne sont pas nécessaires, plutôt que de reprogrammer plusieurs nœuds simplement pour réorganiser l'ordre des données.

22.6.1 Exemple : Configuration du nœud de données

Nous avons pour tâche d'envoyer deux octets d'information dans le tampon TX 0, message ID 105, lorsque l'interrupteur A0 est actionné. Les deux octets de données ont des valeurs par défaut, mais ces valeurs peuvent être mises à jour par programme.

Pour définir les valeurs par défaut, nous ouvrons tout d'abord le panneau des propriétés et allons dans l'onglet TX Buffer 0 et sélectionnons une longueur de données de 2. Notez que les cases D0 et D1 sont actives, mais que les autres sont grisées. Nous devons ensuite entrer les valeurs de données de défaut qui seront automatiquement envoyées lorsque le tampon est envoyé, par exemple 145 dans D0, et 12 dans D1. Une fois les valeurs par défaut définies, elles seront envoyées chaque fois que le tampon sera envoyé, à moins qu'elles ne soient modifiées par programmation.

Nous pouvons modifier la valeur dans le programme en utilisant la macro *SetTxData*, par exemple *SetTxData (0, 2, MyVar0, MyVar1, 0, 0, 0, 0, 0, 0, 0)* changerait les deux octets d'information pour les valeurs de MyVar0 et MyVar1.

22.6.2 Exemple : Configuration du nœud de réception

Si nous souhaitons recevoir des données d'un message, par exemple deux données envoyées dans l'exemple ci-dessus, nous pouvons
GetRxData pour récupérer les éléments de données.

Mettre en place un nœud de réception de base qui interroge la mémoire tampon RX 0 pour l'ID de message 105.
Une fois qu'un message est reçu, nous pouvons l'interroger.

N'oubliez pas que le paramètre d'index *GetRxData* est 0-7 et non 1-8, ce qui correspond aux éléments de données D0-D7.

Ajoutez deux macros *GetRxData* au programme. Définissez la première pour récupérer l'index 0 de l'élément de données du tampon RX 0, *GetRxData (0,0)*, et placez la valeur dans DATA_0 (ou dans une autre variable convenablement nommée). Réglez le second pour récupérer l'index 1 de l'élément de données et placez-le dans DATA_1, *GetRxData(0,1)*.
Nous recevons alors les données et pouvons les vérifier, les afficher ou les modifier à notre guise.

22.6.3 Exemple : Quantités variables de données

Nous savions qu'il n'y aurait que deux éléments de données dans le programme ci-dessus puisque nous avons créé les deux nœuds, mais si nous n'étions pas sûrs, nous pourrions vérifier combien de données sont arrivées avec la macro GetDataCount.
Une fois qu'un message est arrivé, nous utilisons GetRxDataCount pour vérifier combien d'éléments de données sont arrivés. Nous pouvons alors utiliser cette information pour parcourir et lire les éléments de données. Une fois les données lues à l'aide de GetRxData, nous pouvons les utiliser en fonction des besoins du programme.

23 Exemple 4 : Jauge de carburant et témoin lumineux

Installez une jauge de carburant de base avec un témoin lumineux qui s'allume lorsqu'il reste 10 % ou moins d'essence.

23.1 Partie A : Envoi

23.1.1 Objectif

Mettre en place un capteur de niveau de carburant qui transmet le niveau de carburant sous la forme d'une valeur de 0 à 255. En outre, il faut mettre en place un capteur d'alerte "manque de carburant" qui s'active à un niveau de carburant prédéfini.

23.1.2 Instructions

Créez un programme d'envoi CAN de base avec les propriétés par défaut suivantes :
TX Buffer 0 - Message ID = 160, Data Length = 1, D0 = 0.
TX Buffer 1 - Message ID = 176, Data Length = 0.

La mémoire tampon TX 0 transporte la valeur du carburant en D0 et la mémoire tampon TX 1 est utilisée pour le signal d'avertissement.

Le nœud 4 (nœud capteur) sera utilisé pour envoyer les signaux.
Note : Le potentiomètre variable peut être utilisé pour représenter le niveau de carburant.

23.2 Partie B : Réception

23.2.1 Objectif

Mettre en place un panneau d'affichage qui indique le niveau de carburant. Configurer également un indicateur d'alerte qui clignote lorsque le niveau de carburant devient insuffisant.

23.2.2 Instructions

L'écran LCD peut être utilisé pour afficher la quantité de carburant. Il peut s'agir de données brutes (0-255) ou d'une forme de conversion (par exemple, un pourcentage ou un système dans lequel le maximum 255 = X nombre de gallons).

La LED 0 sera utilisée pour le témoin de manque de carburant.

Le nœud 1 (nœud d'affichage) sera utilisé pour afficher le niveau de carburant et le témoin lumineux.

23.3 Autres travaux

Le témoin de carburant s'allume lorsque le niveau de carburant est bas. Cependant, les conducteurs ont la réputation de manquer ou d'ignorer les voyants d'avertissement. Une chose qui attire notre attention est le clignotement d'un voyant.
Modifier le programme pour qu'il produise une lumière clignotante lorsque le carburant atteint la moitié du niveau de remplissage.

24 Notes pour l'exercice 4

Pour le programme, nous devons ajouter un capteur de position analogique (Grove Rotary Angle Sensor Component) et contrôler sa lecture. La lecture du capteur analogique consiste en un octet de données (0 à 255). Ajoutez la macro *GetByte* pour lire le niveau de carburant dans une variable appropriée telle que FUEL_LEVEL.
Ajoutez une macro *SetTxData* et mettez les paramètres *Buffer* 0 (pour TX Buffer 0), *Count* = 1 (1 registre de données) et *D0* = FUEL_LEVEL (les données à envoyer). Vous devez ajouter des données pour les autres registres de données même s'ils ne sont pas utilisés, vous devrez donc ajouter un zéro pour chacun d'entre eux.

Ensuite, une icône de décision permet de vérifier si le niveau de carburant est trop bas. Ici, nous avons choisi de vérifier FUEL_LEVEL par rapport à une autre variable appelée MIN_FUEL, que nous devrons initialiser au début du programme.

Si le niveau de carburant est trop bas, nous pouvons envoyer le signal TX Buffer 1. En utilisant deux tampons, nous pouvons envoyer des mises à jour de données sur le carburant en permanence, mais n'envoyer le signal d'avertissement qu'en cas de besoin.

24.1.1 Réception des données

Tout d'abord, mettez en place un programme qui surveille le réseau CAN à la recherche d'un signal portant l'ID de message 100. Une fois ce signal détecté, nous devons extraire les données, dans ce cas les données FUEL_LEVEL qui ont été envoyées. Nous récupérons les données à l'aide de la macro *GetRxData*. Nous devons fournir les paramètres *Buffer* (0 pour RX Buffer 0), et l'index du registre de données que nous voulons, dans ce cas l'élément que nous voulons est *Index* = 0 qui correspond à l'élément D0 sur le TX Buffer (voir Fig. 23.1).

Fixez la valeur de retour à une variable pratique telle que FUEL_LEVEL. Maintenant que vous disposez des données, vous pouvez les traiter et les afficher sur l'écran LCD. Vous pouvez également utiliser diverses méthodes de sortie, telles qu'un graphique à barres qui s'estompe au fur et à mesure que le carburant est consommé.

Il existe une autre macro utile à utiliser lors de la récupération de données : *GetRxDataCount*, qui renvoie la longueur des données pour le tampon spécifié. Cela vous permet à la fois de vérifier qu'il y a des données (0
= pas de données envoyées), ou pour connaître la quantité de données si une quantité variable est possible.

24.1.2 Réception du signal d'alarme

Le signal d'avertissement est simple. Si un message avec l'ID de message 101 apparaît, nous devons allumer la LED d'avertissement. Nous pouvons le traiter directement après avoir lu et traité les données relatives au niveau de carburant. Ou nous pouvons le traiter ailleurs. Il existe peut-être un autre nœud spécifique qui gère les signaux d'avertissement. Il n'est pas nécessaire de traiter tous les messages envoyés par un nœud au même nœud de réception. Le grand avantage de CAN est qu'il n'est pas nécessaire de tout faire au même endroit. Nous pourrions créer un troisième nœud pour gérer les signaux d'avertissement.

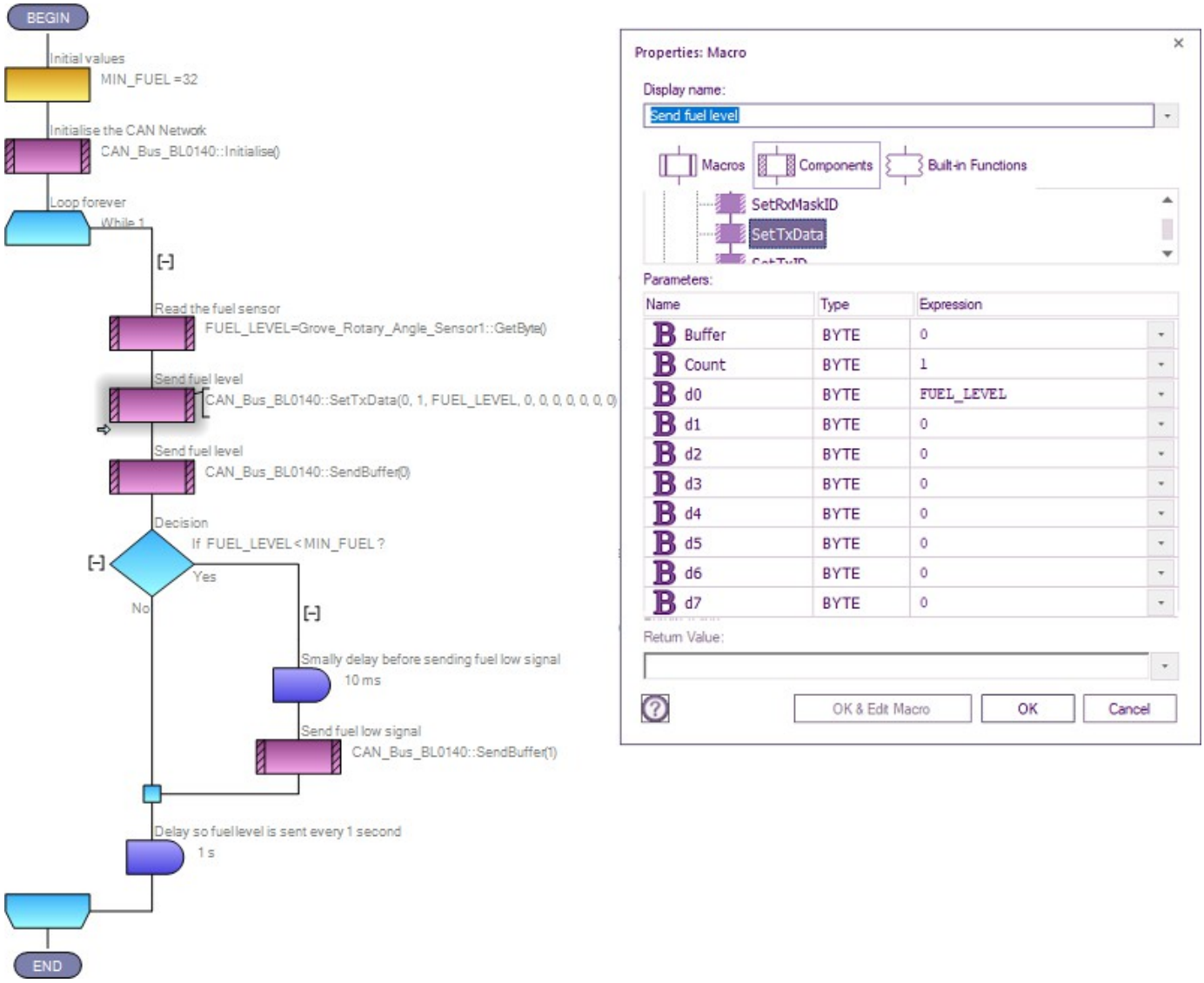


Figure 24.1 *Définition des propriétés de la macro*

25 Démonstration 4 : Jauge de carburant et témoin lumineux

Cet exemple montre un système de messages CAN avec des données en action.

25.1 Mise en place

	PIC BL0011			Arduino BL0055		
	Port A	Port B	Port C	A0-5 (Port C)	D0-7 (Port D)	D8-13 (Port B)
Nœud 1	BL0169		BL0140	BL0169		BL0140
Nœud 4	BL0129		BL0140	BL0129		BL0140

Nous utiliserons le nœud 1, le nœud d'affichage, pour afficher les données. Nous utiliserons le nœud 4, le nœud capteur, équipé du potentiomètre rotatif dans la prise 1 (broches 0,1) pour envoyer les données.

Ouvrez le fichier CAN_EXAMPLE_04_RECEIVE.FCFX dans Flowcode et téléchargez-le vers le nœud 1 (nœud d'affichage). Ouvrez le fichier CAN_EXAMPLE_04_SEND.FCFX dans Flowcode et téléchargez-le vers le nœud 4 (le nœud capteur).

25.2 Visualisation des messages

Si vous ouvrez CANKing et que vous visualisez le trafic réseau, vous verrez que le message est envoyé périodiquement pour le niveau de carburant, ainsi que pour le voyant d'alerte carburant en cas de bas niveau de carburant.

25.3 Le niveau de carburant

Lorsque le programme démarre, le niveau de carburant s'affiche sur l'écran LCD.
Le déplacement du potentiomètre variable sur le nœud de capteurs modifie le niveau de carburant affiché.

25.4 Le témoin lumineux

Lorsque le niveau de carburant devient trop bas, un témoin lumineux s'allume.

25.5 Visualisation des données

Si vous vérifiez le trafic réseau envoyé dans CANKing, vous constaterez qu'un flux de messages est envoyé avec Message ID
100. Il s'agit des données envoyées. Notez que le message ne contient qu'une seule donnée. Les données se rapportent-elles directement au chiffre affiché ? Ou sont-elles modifiées d'une manière ou d'une autre - par exemple, des données brutes transformées en gallons ?
D'où vient le message du voyant d'avertissement ?

25.6 Conclusions

Cette démonstration montre que des données sont transmises. Bien qu'il ne s'agisse que d'un seul élément, cet exemple démontre les possibilités offertes par le réseau CAN. Non seulement vous pouvez envoyer des messages spécifiques qui seront captés par des nœuds récepteurs spécifiques, mais vous pouvez également leur transmettre des données - carburant, vitesse, hauteur, pression, états marche/arrêt, n'importe quoi. Tout ce qui peut être converti en données peut être envoyé.

26 Réseau CAN avancé

Cette section aborde des concepts CAN avancés tels que le réglage des masques et des filtres d'ID de message et les réglages CNF. La structure des messages est examinée, ainsi que d'autres questions telles que le câblage du réseau.

26.1 Exercices

Aucun exercice ou démonstration n'est fourni pour cette section. Les exercices existants peuvent être adaptés pour utiliser les masques et le filtrage ou des exercices peuvent être générés pour démontrer les exemples mathématiques de masques et de filtres donnés ci-dessous.

26.2 Masques et filtres

26.2.1 Masques et filtres - le concept général

Les masques et les filtres constituent une partie importante mais complexe de notre mise en œuvre du CAN. Les masques sont utilisés pour modifier les valeurs d'identification des messages.
Les identifiants des messages sont comparés aux filtres pour déterminer s'ils doivent être acceptés ou non.

26.2.2 Masques

Les masques modifient les valeurs d'ID de message reçues par le tampon. Ils modifient la valeur en supprimant les bits de masque de l'ID de message entrant. Cela peut être utilisé pour faire en sorte qu'un certain nombre d'ID de message différents aient l'air d'avoir la même valeur. Par exemple, un masque peut supprimer le chiffre des dizaines d'un message, de sorte que les messages ID 120, 123, 140 et 165 apparaîtraient respectivement comme 100, 103, 100, 105. Pour plus de détails sur le processus de masquage, voir les exemples ci-dessous.

Si les filtres sont configurés pour accepter l'ID de message 100, les ID de message 120 et 140, qui sont tous deux convertis en 100 par le masque, seront acceptés. Un tel système pourrait être utilisé pour modifier un lot de messages ID qui partagent tous une fonction connexe - par exemple, plusieurs signaux d'avertissement pourraient être envoyés à leurs nœuds de feux d'avertissement respectifs en utilisant leurs messages ID distincts et, parce qu'ils sont tous masqués par la même valeur, ils seraient captés et traités par un nœud central principal de feux d'avertissement.

26.2.3 Filtres

Les filtres sont les portiers du système CAN. Ils vérifient les ID de messages entrants en les comparant aux filtres - une liste d'ID de messages qu'ils peuvent accepter. Si votre nom figure sur la liste, vous serez autorisé à entrer. Si ce n'est pas le cas... désolé, essayez ailleurs.

Chaque filtre est accompagné d'une case à cocher dans le panneau des propriétés, qui permet d'activer ou de désactiver le filtre en question.

Il existe trois réglages de filtre généraux et 2 ou 4 filtres spécifiques en fonction du tampon RX.

- Accepter tous les messages - tous les ID de message sont acceptés. Le nœud répondra à tout message arrivant sur cette page.

tampon.

- Rejeter tous les messages - Il s'agit en fait d'un interrupteur pour la mémoire tampon. Le nœud ne répondra à aucun message sur ce tampon. Peut sembler étrange, mais peut être utilisé pour désactiver un tampon RX qui n'est pas utilisé. Pourquoi activer le tampon RX 1 si vous n'utilisez quele tampon RX 0 ( ) ?
- Utiliser les masques et les filtres - Utilise les masques et les filtres pour modifier et vérifier les ID de message afin de déterminer s'ils doivent être acceptés ou non.

26.2.4 Propriétés du tampon RX

Pour activer les masques et les filtres, sélectionnez le paramètre "Use Mask and Filter", les deux autres paramètres étant "Accept All" et "Reject All". Les valeurs des filtres peuvent être définies dans le panneau des propriétés, par exemple les propriétés du tampon RX 0 affichées ci-dessous.

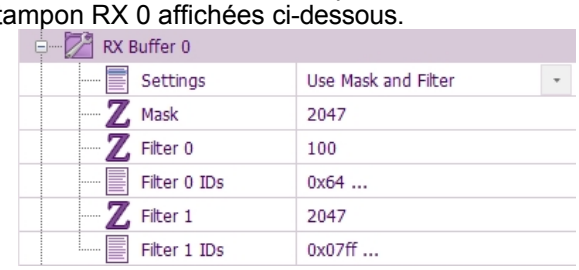


Figure 26.1 *Valeurs du masque et des filtres du tampon RX 0, et ID de message de chaque filtre.*

Les propriétés de la mémoire tampon RX1 sont similaires à celles de la mémoire tampon RX0 présentées ci-dessus, mais davantage de filtres sont disponibles.

26.3 Comment déterminer les messages qui seront piégés par une combinaison particulière de masques et de filtres ?

La meilleure façon est de travailler à partir de quelques exemples :

Notez que toutes les valeurs des ID de message, des masques et des filtres sont des nombres compris entre 0x000 et 0x7FF.

26.3.1 Utilisation de masques et de filtres : Exemple 1

Masque 0 =	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Filtre 0 =	0	0	1	0	0	0	0	0	0	0	0
Filtre 1 =	0	0	0	0	1	0	1	0	0	0	0

Masque 00x0FF Filtre 0 = 0x100 Filtre1 = 0x050

En binaire, cela se présente comme suit :

Masque 1 =	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Filtre 2 =	x	x	x	0	0	0	0	0	0	0	0
Filtre 3 =	x	x	x	0	1	0	1	0	0	0	0

Pour le masque, un "1" signifie "vérifier ce bit" et un "0" signifie "ignorer ce bit

Ces filtres accepteront donc les messages suivants ("x" = indifférent),
à savoir :
Le filtre 0 accepte 0x000, 0x100, 0x200, 0x300, 0x400, 0x500, 0x600,
0x700 Le filtre 1 accepte 0x050, 0x150, 0x250, 0x350, 0x450, 0x550,
0x650, 0x750

26.3.2 Utilisation de masques et de filtres : Exemple 2

Masque 1 = 0x350Filtre 2 = 0x200Filtre 3 = 0x123Filtre 4 = 0x3FF

Masque 1 =	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**
Filtre 2 =	0	1	0	0	0	0	0	0	0	0	0
Filtre 3 =	0	0	1	0	0	1	0	0	0	1	1
Filtre 4 =	1	1	1	1	1	1	1	1	1	1	1

Réécriture en binaire :

Ici, le masque ne vérifiera que 4 bits et ignorera les 6 autres. Voici ce que les filtres acceptent :

Masque 1 =	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**
Filtre 2 =	0	1	0	0	x	0	x	x	x	x	x
Filtre 3 =	0	0	1	0	x	1	x	x	x	x	x
Filtre 4 =	1	1	1	1	x	1	x	x	x	x	x

En fait, ils retiendront un grand nombre de messages (64 chacun !):

Filtre 2 = 0x200, 0x201, 0x202, ... 0x220, 0x221, ... 0x280, 0x281, ... 0x2A0, 0x2A1, ... 0x2AF
Filtre 3 = 0x100, 0x101, 0x102, ... 0x120, 0x121, ... 0x180, 0x181, ... 0x1A0, 0x1A1, ... 0x1AF
Filtre 4 = 0x750, 0x751, 0x752, ... 0x770, 0x771, ... 0x7D0, 0x7D1, ... 0x7F0, 0x7F1, ... 0x7FF

Ce deuxième exemple n'est pas très pratique. En général, il est plus logique de définir le masque de manière à ce que chaque filtre accepte une série de messages consécutifs.

Comme vous pouvez le voir, le masque détermine quels bits des filtres sont effectivement examinés. En fixant le masque à 0x000, le filtre acceptera tout message entrant. En outre, la valeur du masque est directement liée au nombre de messages que chaque filtre piègera - c'est-à-dire 2^(nombre de bits '0' dans le masque).
Une façon utile d'utiliser le masque serait d'ignorer les bits les moins significatifs. Supposons que vous souhaitiez que les filtres acceptent 16 messages chacun - en réglant le masque 0 sur 0x7F0, vous y parviendrez. Ensuite, en réglant les filtres comme suit... :

Filtre 0 = 0x100
Filtre 1 = 0x110

...signifie que les messages suivants sont acceptés :

Filtre 0 = 0x100, 0x101, 0x102, 0x103, 0x104, ... 0x10D, 0x10E, 0x10F
Filtre 1 = 0x110, 0x111, 0x112, 0x113, 0x114, ... 0x11D, 0x11E, 0x11F

Bien entendu, pour les applications CAN simples, vous pouvez souhaiter n'accepter qu'un ou deux messages. Dans ce cas, le fait de fixer le masque à 0x7FF signifie que seul l'ID du message spécifié par chaque filtre sera accepté, par exemple

Masque 1 =
0x7FF Filtre 2 =
0x100 Filtre 5 =
0x200

Cela signifie que seuls les messages 0x100 et 0x200 seraient acceptés dans le tampon RX 1.

26.4 Paramètres CNF

La section "Propriétés" du "Panneau des propriétés" comprend des options qui déterminent les paramètres du CNF. Ces propriétés modifient automatiquement les valeurs de la CNF et devraient suffire dans la plupart des cas. Toutefois, il peut arriver que vous deviez définir manuellement ces valeurs. Dans ce cas, il y a de fortes chances que l'on vous ait donné les valeurs à définir. Si c'est le cas, vous pouvez simplement saisir ces valeurs directement.
Si ce n'est pas le cas, vous devrez peut-être consulter la documentation et les tableaux CAN pour déterminer les valeurs CNF à définir. Des liens vers la documentation CAN se trouvent à la fin de ce document.
N'oubliez pas qu'en règle générale, il est préférable de s'assurer que ces paramètres sont les mêmes pour chaque nœud CAN sur le bus, bien que seule la valeur du taux de bus doive être cohérente - les ajustements du point d'échantillonnage et du SJW ne sont jamais nécessaires qu'en cas d'utilisation de câbles anormalement longs entre les nœuds.

26.5 Détails du message

Lorsqu'un message est envoyé, non seulement l'ID du message et toutes les données associées sont envoyées, mais un certain nombre de marqueurs et d'éléments d'enveloppe sont également ajoutés. La figure 25.2 montre à quoi ressemble un message.

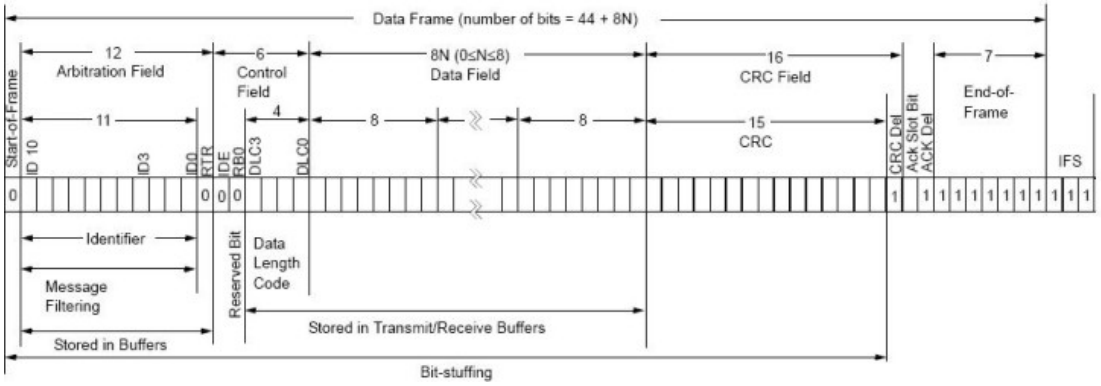


Figure 26.2 *Format du message*

Notez que les 1 et les 0 se réfèrent à des niveaux de tension dominants et récessifs et non à des niveaux logiques CMOS ou TTL. En principe
CAN est capable d'utiliser différents niveaux de tension afin de s'adapter à différents environnements électriques.

26.6 Détection des erreurs

CAN dispose d'un certain nombre de systèmes de détection automatique d'erreurs qui signalent les erreurs lorsqu'elles se produisent. Si une erreur est détectée, un signal d'erreur est envoyé pour détruire le trafic du réseau et les nœuds du réseau prennent les mesures appropriées, par exemple en rejetant le message contenant l'erreur.

Le suivi des erreurs est complexe et, pour en connaître tous les détails, il faut consulter les spécifications CAN, mais il peut être résumé brièvement comme suit : Les nœuds suivent le nombre d'erreurs de transmission et d'erreurs de réception signalées sur deux compteurs. Les erreurs de transmission incrémentent le compteur d'erreurs de transmission de 8 par erreur, et les erreurs de réception incrémentent le compteur d'erreurs de réception de
1. Les transmissions et les réceptions réussies décrémentent les valeurs. Cela signifie que le compteur d'erreurs de transmission augmentera plus rapidement, ce qui est approprié car les émetteurs défectueux sont plus susceptibles de poser problème. Cela permet de s'assurer qu'un émetteur défectueux atteindra le point de désactivation avant que les nœuds ne reçoivent son signal et n'obtiennent des erreurs de réception.

Si l'une de ces valeurs atteint 127, le nœud devient passif à l'égard des erreurs. En mode passif, le nœud transmet toujours des erreurs, mais celles-ci ne détruisent plus le trafic réseau, c'est-à-dire que le nœud se déclare sujet aux erreurs et potentiellement responsable du problème, de sorte qu'il cesse de détruire le trafic réseau, car les messages ne sont peut-être pas à l'origine du problème. Cependant, le nombre d'erreurs continue d'augmenter normalement.

Si l'un des compteurs d'erreurs atteint 255, le nœud passe en mode "Bus Off" et cesse de transmettre. Le nœud a identifié un problème et s'est retiré du réseau.

En résumé, le nœud passe du cri "erreur majeure - tout s'arrête", au cri d'erreur ignoré, puis à la désactivation.

26.6.1 Contrôle des bits

Lorsque les signaux sont transmis, leur niveau est également vérifié par la partie réceptrice du système CAN. Si le niveau détecté n'est pas ce qu'il devrait être, une *erreur de bit* est signalée.

26.6.2 Farce de mors.

CAN a besoin de savoir si un long signal est une erreur plutôt qu'une partie du message. Pour ce faire, le CAN procède à un bourrage de bits (Bit Stuffing). Si 5 bits de la même valeur sont envoyés (0 ou 1), un bit 6th est inséré avec la valeur opposée pour indiquer au réseau qu'il ne s'agit pas d'une erreur. Ce bit supplémentaire est automatiquement supprimé par les nœuds. Si ce bourrage de bits ne se produit pas, CAN se rendra compte qu'il y a eu un problème quelque part et émettra une *erreur de bourrage*.

Ceci est nécessaire car il n'y a pas de signal d'horloge séparé dans CAN. Au lieu de cela, le débit de données est synchronisé en utilisant le débit de données. Le remplissage de bits aide le système CAN à synchroniser cette fréquence d'horloge de données.

26.6.3 Bit ACK

Lorsqu'un nœud envoie un message, le bit ACK ou Accusé de réception est mis à 0 (récessif). Lorsqu'un nœud reçoit un message, il en accuse réception en renvoyant le signal avec le bit ACK à 1 (dominant). Cela ne signifie pas que le message a atteint sa destination prévue, mais simplement qu'il a été reconnu par le nœud récepteur en question comme un message légitime. Toutefois, en vérifiant que le bit ACK est bien présent dans le signal renvoyé, le nœud d'envoi peut signaler une *erreur d'accusé de réception* si ce n'est pas le cas.

26.6.4 Vérification du cadre

Certaines parties du message CAN ont des formats et des signaux définis. Le message est surveillé pour détecter les erreurs dans ces parties. Si une erreur est détectée, une *erreur de forme* est générée.

26.6.5 Contrôle de redondance cyclique (CRC)

Les messages contiennent une somme de contrôle de 15 bits qui peut être vérifiée par les nœuds récepteurs. Si les valeurs ne correspondent pas, un message
Le signal d'*erreur CRC* peut être déclenché.

26.7 Câblage et autres questions pratiques

Une implémentation courante de l'interface physique CAN utilise une paire de fils torsadés, ce qui permet de minimiser les erreurs dues aux pics de tension et aux interférences CEM. Les réseaux sont terminés par une résistance entre les fils afin de lutter contre les interférences électriques. Les nœuds sont ajoutés en les connectant à la paire de fils torsadés. Tous les nœuds sont interconnectés via le réseau ; aucun nœud n'est isolé d'un autre sur le réseau.

Les tensions sont soit *dominantes* (signifiées par un 1 dans le format écrit), soit *récessives* (signifiées par un 0 dans le format écrit). Il n'existe pas de tensions absolues. La seule exigence est que le système soit capable de faire *la distinction entre* les signaux dominants et récessifs. Cela nous libère considérablement car nous pouvons alors travailler avec des signaux appropriés à l'interface physique qui convient le mieux au système, plutôt que de devoir concevoir le système avec des tensions spécifiques à l'esprit. Cependant, les circuits intégrés et autres matériels utilisés dans votre système CAN peuvent avoir leurs propres tolérances et niveaux attendus qui doivent être pris en considération.

CAN ne spécifie que le format des messages, pas la couche physique. Bien que cela nous donne une plus grande flexibilité dans le câblage d'un système CAN, cela signifie également que les détails du câblage peuvent varier considérablement entre des systèmes similaires. D'autres interfaces physiques, telles que les fibres optiques à ligne unique, sont des solutions de câblage parfaitement acceptables.

27 Données de référence

Ce document a été conçu pour vous enseigner les bases du CAN, la théorie et les concepts qui le sous-tendent, ainsi que des exercices pratiques destinés à accroître vos connaissances et vos compétences en matière de travail avec le CAN. Il devrait également vous permettre de développer votre propre programme d'enseignement sur le CAN, avec de nombreuses possibilités de démonstrations et de travaux pratiques.

Mais ce n'est pas tout. Si vous souhaitez approfondir votre étude du RCA ou vous orienter vers des secteurs industriels qui utilisent le RCA, il y a beaucoup plus à savoir et beaucoup plus à étudier !

27.1 Normes CAN

CAN est une spécification évolutive. Il a déjà progressé par rapport à la spécification originale de Bosch et est actuellement disponible en versions standard (version 2.0A) et étendue (version 2.0B).

Le système représenté ici est un système CAN standard. Le CAN étendu présente un certain nombre de différences, notamment en ce qui concerne le format des messages. La principale différence pratique est que le CAN étendu utilise des ID de message de 29 bits au lieu des valeurs de 11 bits utilisées par le CAN standard. Cela permet à Extended CAN de s'adresser à quelque 500 millions de nœuds. (Compte tenu de la pénibilité des conversions 11-8 bits, n'êtes-vous pas heureux que nous n'ayons pas opté pour 29 bits !)

Les spécifications CAN peuvent être obtenues sur le site web de Bosch à l'adresse suivante : www.semiconductors.bosch.de

Deux normes ISO sont également utilisées pour les transmissions CAN. La norme ISO1159 est utilisée pour les réseaux à faible vitesse (jusqu'à 125kbit/seconde). La norme ISO11898 est utilisée pour les réseaux à grande vitesse (jusqu'à 1Mbit/seconde).
Il existe des différences entre les deux normes en ce qui concerne le câblage, les tolérances de tension, etc. Si vous en avez besoin, les normes ISO peuvent être achetées sur le site web de l'ISO à l'adresse suivante : www.iso.org.

27.2 Protocoles de niveau supérieur

Les protocoles de plus haut niveau (HLP) sortent du cadre de ce cours, mais peuvent intéresser ceux qui poursuivront l'étude du RCA. Les protocoles de niveau supérieur font référence aux langages système qui transmettent les données et les présentent dans un format que les applications utilisant ce protocole peuvent comprendre.

CAN ne spécifie que le format du message et laisse le protocole de niveau supérieur ouvert. Cela permet à différents secteurs industriels ou à différentes entreprises de développer ou d'adopter les protocoles de niveau supérieur qui répondent le mieux à leurs besoins, tels que CANopen de Kvaser (www.kvaser.com).

Pour nous, cela signifie qu'il n'y a pas un seul ensemble de protocoles de niveau supérieur à étudier. Il existe un certain nombre de protocoles CAN de niveau supérieur sur le marché. Le choix du protocole à apprendre et à utiliser peut simplement dépendre de l'entreprise pour laquelle vous allez travailler. Les concepteurs de systèmes CAN peuvent avoir besoin d'être suffisamment flexibles pour s'adapter à un travail de conception dans un ou plusieurs de ces protocoles de niveau supérieur.

27.3 Acronymes et abréviations

ACK Accusé de réception
ADCAConvertisseur analogique-numérique CANRéseau de zone de contrôle CANH CAN high
CANL CAN low
CMOS Semi-conducteur complémentaire à base d'oxyde métallique CRCContrôle de redondance cyclique
DACConvertisseur numérique-analogique ECUEUnité de commande électronique
EMCE Compatibilité électromagnétique HLPH Protocole de haut niveau
ICIcircuit intégré IDIdentifiant (ou identification)
ISOOrganisation internationale de normalisation LCDÉcran à cristaux liquides
LEDDiode électroluminescente
RPMRévolutions par minute RX Réception
SJWS Largeur de saut de synchronisation TTLT Logique à transistor TX Transmettre
USBBus série universel