Département de Génie Electrotechnique et Automatique
Université de 8 Mai 1945 Guelma

Licence 3ème année Automatique
Module : Bus de communication et réseaux locaux industriels

TP4: Utilisation du bus CAN

1. Introduction

Dans ce TP, nous abordons un des bus utilisés dans le milieu industriel et qui est tolérant aux perturbations dues aux bruits. Plus particulièrement, nous utilisons le bus Can qui est à l'origine un bus destiné pour communiquer les équipements de l'automobile. Dans ce TP, nous utilisons la carte Arduino avec un module CAN pour faire communiquer deux Arduino via le bus CAN, dans l'absence d'équipements industriels pour les interfacer avec l'Arduino.

En réalisant le TP, l'étudiant aura la capacité de communiquer (lire et écrire des données) avec un tableau de bord d'un véhicule, automate ou autre dispositif utilisant le bus CAN.

2. Principe de fonctionnement du bus CAN

Le bus CAN est un bus système qui permet une transmission sérielle, et il est fréquemment utilisé dans l'industrie, notamment le domaine de l'automobile. Il permet de raccorder plusieurs équipements et unités de contrôle sur un seul bus ayant seulement deux lignes, où la communication se fait par tour de rôle [1]. L'autre avantage de ce bus est de permettre une communication de longue distance qui se diffère selon le débit de transmission.

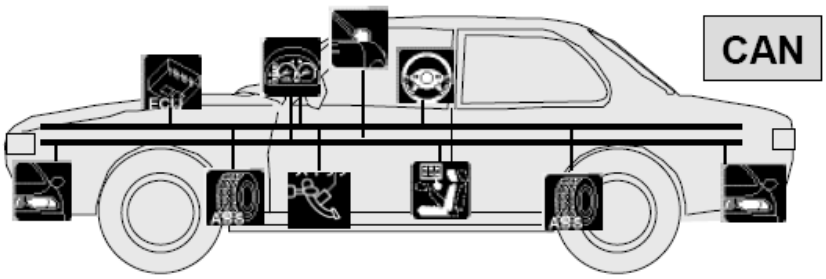

Figure 4.1. Interconnexion de plusieurs capteurs et actionneurs via le bus CAN [2]

Comme mentionné précédemment, la communication via le bus CAN est basée sur deux fils (CAN High/CanH et CAN Low/CanL), et ceux-ci engendre un signal différentiel. En d'autres termes, la lecture des bits transmis se fait par la mesure de la différence de tensions entre les deux fils. Vu la perturbation électromagnétique altère les deux fils, la différence de

tension reste identique [1], et c'est pour cette raison les véhicules (beaucoup de bruitages) utilisent le bus CAN.

Par défaut, en repos la tension des deux fils (CanH et CanL) est mise à 2.5v, et elle se change dans le cas de transmission. Plus spécifiquement, lors de transmission du bit 0 (un bit dominant), la tension de CanH augmente de 1v et la tension de CanL diminue avec 1v, et donc, pour transmettre 0, CanH devient 3.5v et CanL devient 1.5v. Tandis que pour transmettre un bit 1 (bit récessif), les tensions gardent la valeur initiale (2.5v).

Il se peut qu'il y ait une perturbation dans les fils, ce qui engendra une différence de tensions au moment du repos. A cet égard, le bus CAN contient un dispositif **transceiver** qui vérifie les signaux de CanH et CanL s'ils sont déclenchés ou non. En outre, il y a une résistance de 120 Ohm (deux résistances de 60 Ohm de chaque côté) entre les deux fils CanH et CanL pour éliminer les parasites aux réfléchissements des signaux sur les extrémités.

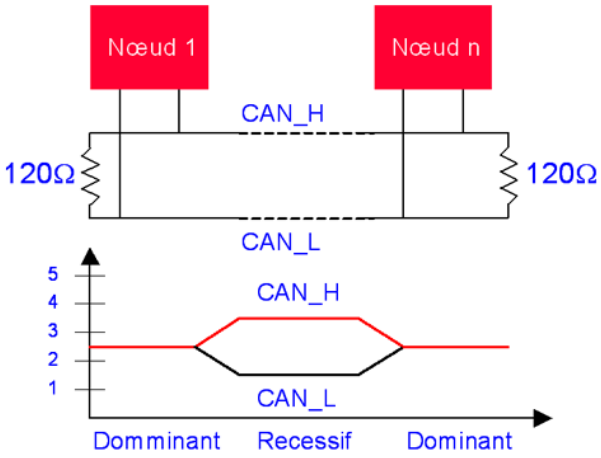

Figure 4.2. Illustration de différence de tension dans la communication CAN [3]

Le bus CAN est basé sur le mécanisme de diffusion (broadcasting) des paquets à tous les dispositifs connectés, et cela permet à tout le monde de lire les données circulantes. Il y a deux types de paquets tels que le paquet standard et le paquet étendu [4], où chaque paquet a une taille différente (CAN standard 11 bits et CAN étendu 29 bits) et une structure contenant des informations, i.e. ID, CRC, ACK, etc. Pour plus d'informations sur le principe de fonctionnement et le détail des paquets, l'étudiant doit consulter le chapitre '*Bus CAN*' du cours '*Bus de communication et réseaux locaux industriels*'.

3. Matériel utilisé dans ce TP

Afin de réaliser ce TP et pratiquer le protocole SPI, on utilisera le matériel suivant :

- Deux cartes Arduino UNO ou Arduino Mega ;
- Deux modules CAN Bus ;
- Des fils de prototypage male-femelle (*male-male en cas d'utiliser une plaque d'essai*) ;

Figure 4.3. Matériel utilisé pour faire communiquer deux Arduino via le bus CAN

4. Communication entre deux Arduino via le bus CAN

Dans ce TP, nous allons faire communiquer deux Arduino via le bus CAN vu que les équipements de l'automobile ne sont pas à la porter de tout le monde pour les interfacer. Et pour ce faire, on a besoin de deux Arduino Uno, deux modules CAN Bus MCP2515 et des fils de prototypages.

Le module MCP2515 met en place une interface SPI pour pouvoir interagir avec les cartes de développement qui ne disposent pas de bus CAN comme l'Arduino. Donc, la communication entre l'Arduino et le module se fait à travers le protocole SPI, et le module traduit les commandes et les données reçues en paquets CAN pour les envoyer dans le bus CAN.

Le module MCP2515 est équipé d'un circuit MCP2551 qui gère les signaux CanH et CanL (transceiver), ainsi ce circuit est alimenté par 5v et permet de connecter jusqu'à 112 nœuds sur le bus.

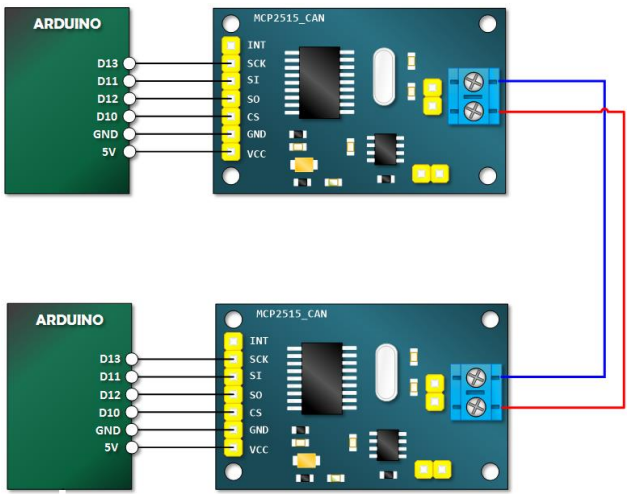

Figure 4.4. Schéma de branchement des modules MCP2515 sur les Arduino

Donc, afin de réaliser ce TP, on a besoin de deux Arduino Uno, des fils de prototypage et deux modules MCP2515. On essaie de lire un message à partir du ***Serial Monitor*** du premier Arduino, le transmettre au deuxième Arduino via le bus CAN et l'afficher dans le ***Serial***

monitor du deuxième Arduino. Par conséquent, on relit le port VCC du module à la sortie 5v de l'Arduino, le GND dans le GND, le CS dans le port 10, le SO dans le port 12, le SI dans le port 11 et le SCK dans le port 13 (Figure 4.4). Il est impératif de vérifier si les deux fils de CAN (rouge et bleu) ne sont pas inversés, sinon la communication ne se fera pas.

Etant la bibliothèque[1] d'interfaçage du module MCP2515 n'est pas disponible par défaut sur l'éditeur Arduino, on doit la télécharger et l'installer. Ensuite, le premier Arduino doit être programmé en étant émetteur et l'autre en étant récepteur.

Tableau 4.1. Liste des vitesses de transmission utilisées par MCP2515

Vitesses de transmission
CAN_5KBPS, CAN_10KBPS, CAN_20KBPS, CAN_31K25BPS, CAN_33KBPS, CAN_40KBPS, CAN_50KBPS, CAN_80KBPS, CAN_83K3BPS, CAN_95KBPS, CAN_100KBPS, CAN_125KBPS, CAN_200KBPS, CAN_250KBPS, CAN_500KBPS, CAN_1000KBPS

```
1.  #include <SPI.h>
2.  #include <mcp2515.h>
3.  struct can_frame canMsg;
4.  char buffer[8] ;
5.  MCP2515 mcp2515(10);
6.
7.  void setup()  {
8.      Serial.begin(9600);
9.      SPI.begin();
10.
11.     mcp2515.reset();
12.     mcp2515.setBitrate(CAN_500KBPS , MCP_8MHZ);
13.     mcp2515.setNormalMode();
14. }
15.
16. void loop()  {
17.     while(Serial.available())   {
18.         Serial.readBytes(buffer, 8) ;
19.         canMsg.can_id  = 0x036;
20.         canMsg.can_dlc = 8;
21.
22.         memcpy(canMsg.data , buffer , 8) ;
23.
24.         mcp2515.sendMessage(&canMsg);
25.         delay(1000);
26.     }
27. }
```

Figure 4.5. Code source d'envoi un message via le bus CAN en utilisant MCP2515

La première chose à faire avant d'utiliser le module MCP2515 consiste à inclure les bibliothèques nécessaires (SPI et MCP2515), et configurer le module en choisissant le numéro du port CS du module (pratiquement 10), puis en précisant la vitesse de transmission parmi une liste prédéfinie (Tableau 4.1). En outre, on précise la vitesse de l'horloge que le module

[1] *https://github.com/autowp/arduino-mcp2515/archive/master.zip*

doit utiliser telle que 8Mhz, 16Mhz et 20Mhz (ligne 12 Figure 4.5). Consécutivement, on définit le mode de communication (ligne 13 Figure 4.5).

```
1.  #include <SPI.h>
2.  #include <mcp2515.h>
3.  struct can_frame canMsg;
4.  char buffer[8] ;
5.  MCP2515 mcp2515(10);
6.
7.  void setup()  {
8.      Serial.begin(9600);
9.      SPI.begin();
10.
11.     mcp2515.reset();
12.     mcp2515.setBitrate(CAN_500KBPS , MCP_8MHZ);
13.     mcp2515.setNormalMode();
14. }
15.
16. void loop() {
17.     if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK)  {
18.         memcpy(buffer , canMsg.data , 8) ;
19.         Serial.println(buffer) ;
20.     }
21. }
```

Figure 4.6. Code source de réception d'un message via le bus CAN en utilisant MCP2515

Après avoir configuré le module, l'émetteur prépare un paquet de données (de taille 11 octets) représenté par une structure de données (4 octets ID, 1 octet DLC et 8 octets DATA). Donc, on lit le message tapé sur le ***Serial Monitor*** et on le sauvegarde dans les 8 octets de DATA du paquet CAN pour le transmettre sur le bus en utilisant la fonction **sendMessage** de l'objet MCP2515. Contrairement dans le récepteur, on utilise la fonction **readMessage** pour lire les données reçues dans le bus et on les affiche dans le ***Serial Monitor*** du deuxième Arduino.

Références

[1] Découverte du bus CAN. *https://arduino103.blogspot.com/2018/04/canbus-la-decouverte-du-bus-can-et-du.html* [Consulté en 10/2019]

[2] Exposé sur le Bus CAN. *http://igm.univ-mlv.fr/~dr/XPOSE2009/BusCAN/intro_can.html* [Consulté en 10/2019]

[3] Cours Systèmes Embarqués: Le Bus CAN. *https://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-Bus-CAN.htm* [Consulté en 10/2019]

[4] Tutoriel sur la lecture des données d'automobiles via le bus CAN. *https://publicism.info/engineering/penetration/3.html* [Consulté en 10/2019]