TP CAN Arduino

Sakun

December 2025

1 Introduction

Dans ce TP, un réseau CAN simple est mis en œuvre à l'aide de cartes Arduino Uno associées au contrôleur CAN MCP2515, communiquant avec l'Arduino via le bus SPI. Un transceiver CAN assure l'interface physique avec le bus différentiel CANH/CANL. L'objectif est de configurer le système, puis de transmettre et recevoir des trames CAN afin de comprendre le fonctionnement général du bus CAN et son intégration dans un système embarqué.

2 Le MCP2515

Le MCP2515 est un contrôleur CAN autonome utilisés pour des microcontrôleurs ne disposant pas nativement d'une interface CAN. Il communique avec le microcontrôleur hôte via le bus SPI et prend en charge l'ensemble des fonctionnalités du protocole CAN, notamment la gestion des trames, l'arbitrage et la détection d'erreurs. Associé à un transceiver CAN externe, il permet de mettre en œuvre simplement un nœud CAN conforme à la norme CAN 2.0.

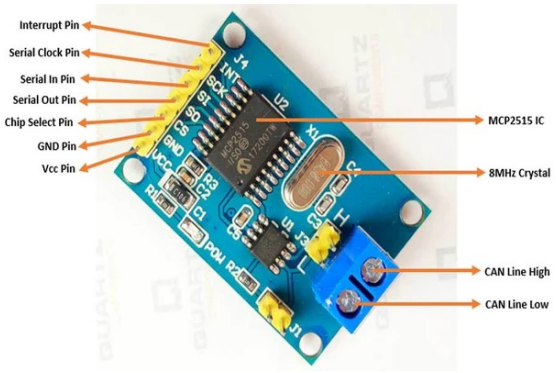

Figure 1: Module CAN basé sur le contrôleur MCP2515

2.0.1 VCC — Alimentation

- **Rôle :** Alimentation du module
- **Valeur typique :**
  - 5 V (le plus courant sur ces modules)
  - parfois 3,3 V selon la version (à vérifier sur la carte)
- **Connexion :** Sortie 5 V (ou 3,3 V) du microcontrôleur

Cette broche alimente à la fois le **MCP2515** et le **transceiver CAN**.

2.0.2 GND — Masse

- **Rôle :** Référence électrique commune
- **Connexion :** Masse (GND) du microcontrôleur

Une masse commune est indispensable au bon fonctionnement de la communication SPI.

2.0.3 CS — Chip Select (ou SS)

- **Rôle :** Sélection du périphérique SPI
- **Direction :** Entrée du MCP2515
- **État actif :** LOW

Le microcontrôleur place cette broche à l'état bas pour activer le MCP2515. Elle permet également de partager le bus SPI entre plusieurs périphériques.

2.0.4 SCK — Serial Clock

- **Rôle :** Horloge SPI
- **Direction :** Entrée du MCP2515
- **Générée par :** le microcontrôleur (maître SPI)

Cette broche synchronise l'envoi et la réception des données SPI.

2.0.5 SI — Serial Input (MOSI)

- **Autre nom :** MOSI (Master Out Slave In)
- **Rôle :** Données envoyées du microcontrôleur vers le MCP2515
- **Direction :** Entrée du MCP2515

Elle est utilisée pour transmettre les commandes SPI, les identifiants CAN et les données à envoyer sur le bus CAN.

2.0.6 SO — Serial Output (MISO)

- **Autre nom :** MISO (Master In Slave Out)
- **Rôle :** Données envoyées du MCP2515 vers le microcontrôleur
- **Direction :** Sortie du MCP2515

Cette broche permet de lire les messages CAN reçus, les registres internes et les états d'erreur.

2.0.7 INT — Interruption

- **Rôle :** Signal d'interruption
- **Direction :** Sortie du MCP2515
- **État actif :** généralement LOW

Le MCP2515 active cette broche lors de la réception d'un message CAN, d'une erreur ou de la fin d'une transmission, permettant une réaction rapide du microcontrôleur sans scrutation continue du bus SPI.

3 La carte Arduino Uno

L'Arduino Uno est une carte de prototypage basée sur le microcontrôleur ATmega328P. Elle est utilisée dans ce TP comme microcontrôleur hôte pour piloter le contrôleur CAN MCP2515 via le bus SPI. Bien que l'Arduino Uno ne dispose pas d'interface CAN native, il permet de comprendre concrètement l'architecture d'un nœud CAN en séparant le contrôle applicatif et la gestion du protocole.

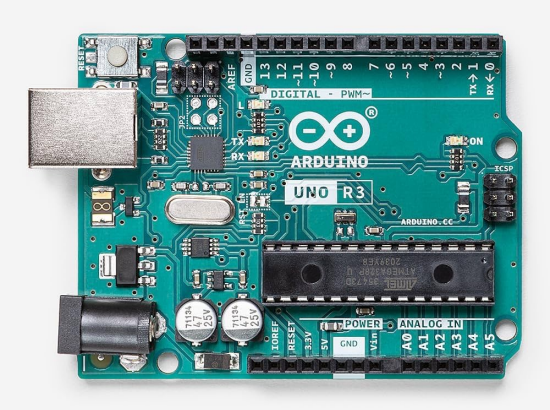

Figure 2: Arduino Uno

L'Arduino Uno dispose de plusieurs types de broches permettant l'alimentation de la carte, la communication avec des périphériques externes et l'acquisition de signaux. Ces broches peuvent être regroupées selon leur fonction.

Broches d'alimentation La carte met à disposition des broches d'alimentation permettant d'alimenter l'Arduino et des modules externes :

- **5V** : sortie d'alimentation régulée 5 V
- **3.3V** : sortie d'alimentation 3,3 V
- **GND** : masse
- **Vin** : entrée d'alimentation externe (7 à 12 V recommandé)

Broches numériques L'Arduino Uno possède **14 broches numériques** (D0 à D13), configurables en entrée ou en sortie, avec des niveaux logiques de 0 à 5 V. Certaines broches numériques ont des fonctions spécifiques :

- **D0 / D1** : communication série UART
- **D2 / D3** : interruptions externes
- **D10 à D13** : interface SPI

Interface SPI L'interface SPI matérielle est utilisée pour la communication avec le contrôleur CAN MCP2515 :

- **D10** : CS (Chip Select)
- **D11** : MOSI
- **D12** : MISO
- **D13** : SCK

Broches analogiques La carte dispose de **6 entrées analogiques** (A0 à A5), avec une résolution de 10 bits, permettant la lecture de tensions comprises entre 0 et 5 V.

4 Les bus CAN

4.1 Câblage

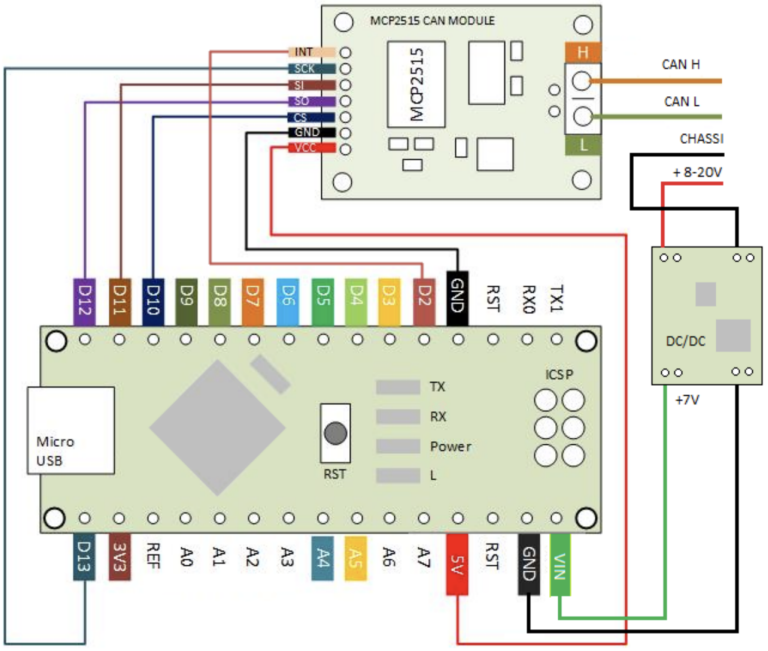

Figure 3: Câblage : connexion Arduino / MCP2515

4.2 Code

Dans ce TP, la librairie `mcp2515.h` est utilisée afin de faciliter la communication entre l'Arduino Uno et le contrôleur CAN MCP2515.

Pour l'ajouter à l'IDE Arduino, les étapes suivantes doivent être suivies :

- Télécharger l'archive `.zip` de la librairie depuis le dépôt GitHub suivant : `https://github.com/autowp/arduino-mcp2515/archive/master.zip`
- Ouvrir l'IDE Arduino, puis sélectionner le menu `Sketch` → `Add Library` → `Add .ZIP Library`
- Sélectionner l'archive téléchargée pour installer la librairie

4.3 Exemples de code

4.3.1 Envoyer un message dans le bus

```
1 #include <SPI.h>
2 #include <mcp2515.h>
3
4 struct can_frame canMsg1;
5 struct can_frame canMsg2;
```

```
MCP2515 mcp2515(10);

void setup() {
  canMsg1.can_id  = 0x0F6;
  canMsg1.can_dlc = 8;
  canMsg1.data[0] = 0x8E;
  canMsg1.data[1] = 0x87;
  canMsg1.data[2] = 0x32;
  canMsg1.data[3] = 0xFA;
  canMsg1.data[4] = 0x26;
  canMsg1.data[5] = 0x8E;
  canMsg1.data[6] = 0xBE;
  canMsg1.data[7] = 0x86;

  canMsg2.can_id  = 0x036;
  canMsg2.can_dlc = 8;
  canMsg2.data[0] = 0x0E;
  canMsg2.data[1] = 0x00;
  canMsg2.data[2] = 0x00;
  canMsg2.data[3] = 0x08;
  canMsg2.data[4] = 0x01;
  canMsg2.data[5] = 0x00;
  canMsg2.data[6] = 0x00;
  canMsg2.data[7] = 0xA0;

  while (!Serial);
  Serial.begin(115200);

  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS);
  mcp2515.setNormalMode();

  Serial.println("Example: Write to CAN");
}

void loop() {
  mcp2515.sendMessage(&canMsg1);
  mcp2515.sendMessage(&canMsg2);

  Serial.println("Messages sent");

  delay(100);
}
```

Listing 1: Envoi de deux trames CAN avec un MCP2515

4.3.2 Lire un message

```
#include <SPI.h>
#include <mcp2515.h>

struct can_frame canMsg;
MCP2515 mcp2515(10);

void setup() {
  Serial.begin(115200);

  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS);
  mcp2515.setNormalMode();

  Serial.println("-------␣CAN␣Read␣----------");
  Serial.println("ID␣␣DLC␣␣␣DATA");
}

void loop() {
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) {
    Serial.print(canMsg.can_id, HEX);   // Affichage de l'
        identifiant
    Serial.print("␣");
    Serial.print(canMsg.can_dlc, HEX);  // Affichage du DLC
    Serial.print("␣");

    for (int i = 0; i < canMsg.can_dlc; i++) {
      Serial.print(canMsg.data[i], HEX); // Affichage des
          donn es
      Serial.print("␣");
    }

    Serial.println();
  }
}
```

Listing 2: Réception et affichage de trames CAN avec un MCP2515

4.3.3 Code fait maison

voir sur github

5 Bibliographie

- **autowp**, *Arduino MCP2515 – CAN write example*, https://github.com/autowp/arduino-mcp2515/blob/master/examples/CAN_write/CAN_write.ino

- **LucasThTrT**, *N7RT Data – CAN Bus*,
https://github.com/LucasThTrT/N7RT_Data/tree/main/Embedded_System/CAN$_BUS$
- **N7 Racing Team**, *Pôle Data – Dépôt GitHub*,
https://github.com/N7-Racing-Team/Pole_data