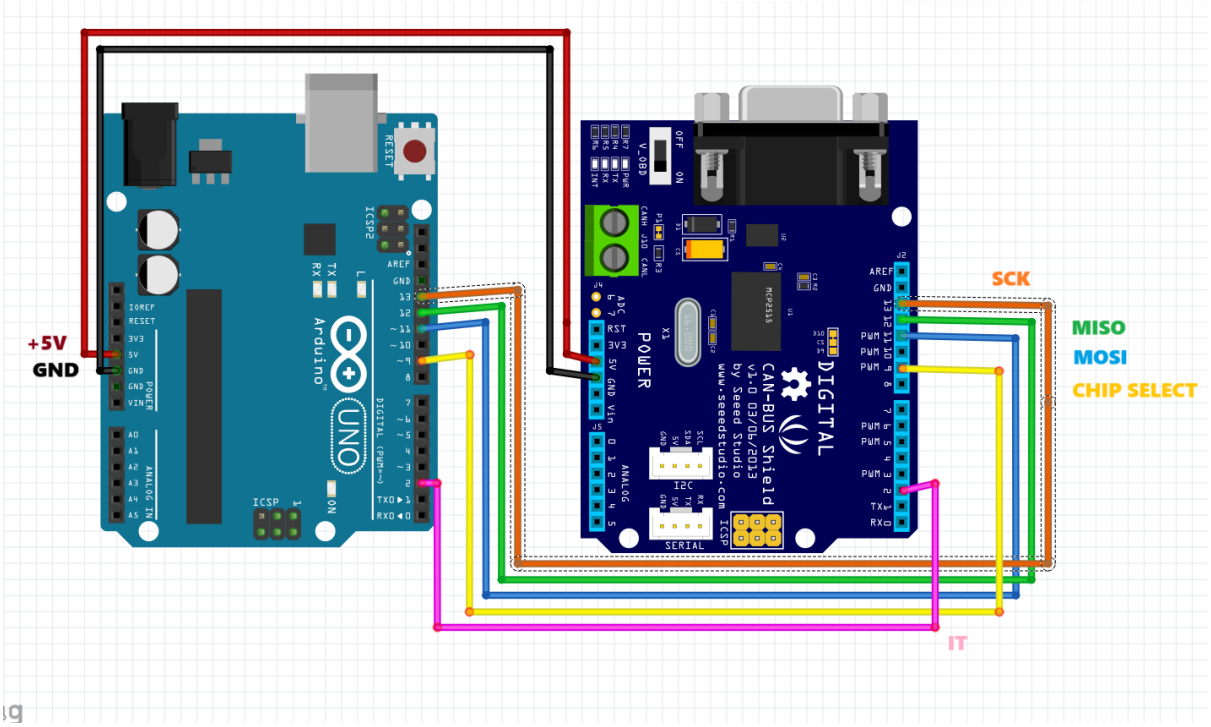
a-Assemblage matériel — Arduino Uno R3 + CAN-BUS Shield V1.2

Lorsqu'on téléverse le code depuis l'IDE Arduino vers la carte Arduino Uno R3, le fichier binaire est transmis via le port USB en utilisant le convertisseur USB-série vers le microcontrôleur principal, qui exécute ensuite le bootloader avant de lancer le programme utilisateur, initialisant la communication SPI grâce à la fonction SPI.begin(). Cela configure automatiquement les broches D11 (MOSI), D12 (MISO), D13 (SCK) en mode maître SPI, et la broche D9 est utilisée comme CS (Chip Select) pour sélectionner le MCP2515 lors des transmissions.

Schéma de câblage:



Placement électrique : le shield s'en fiche directement sur l'Arduino.

Étape SPI :

L'Arduino transmet les octets ci-dessus à travers le **bus SPI** vers le MCP2515, selon le protocole SPI standard :

1. Le signal **CS (D9)** est mis à LOW
2. Les données sont envoyées bit par bit via MOSI (D11)
3. Le SCK (D13) cadence la transmission
4. Le MCP2515 répond éventuellement via MISO (D12)

Séquence complète : Arduino met CS à LOW → transmet les octets → MCP2515 répond → CS à HIGH & termine.

Le MCP2515, qui est un contrôleur CAN autonome, est interfacé avec l'Arduino à travers ces lignes SPI :

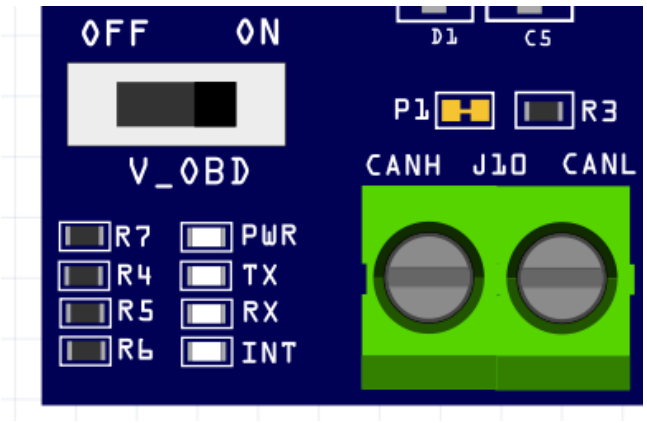
Broche Arduino	Fonction sur le shield CAN BUS v1.2	Rôle
D2	INT	Pour l'interruption en cas de message CAN entrant. (Le MCP2515 générant un interrupt sur `INT`, Arduino exécute "***mcp2515.readMessage()"*** pour récupérer l'ACK ou les données.
D9	CS (Chip Select SPI)	Sélection du MCP2515 pendant la communication SPI (active le MCP2515 lors des échanges)
D11 (MOSI)	SPI Master-Out, Slave-In	Données envoyées par l'Arduino vers le MCP2515 (données de configuration ou de trames)
D12 (MISO)	SPI Master-In, Slave-Out	Données reçues du MCP2515 vers l'Arduino
D13 SCK (via header ICSP)	SPI Clock	Horloge SPI
GND	Masse	Référence électrique commune
5V	Alimentation du shield	Alimente le MCP2515 et le TJA1050

Le programme configure ensuite le MCP2515 via SPI est initialisé pour fonctionner à 100 kbps avec un oscillateur de 16 MHz comme suit:
"mcp2515.reset();"
"mcp2515.setBitrate(CAN_100KBPS, MCP_16MHZ);"

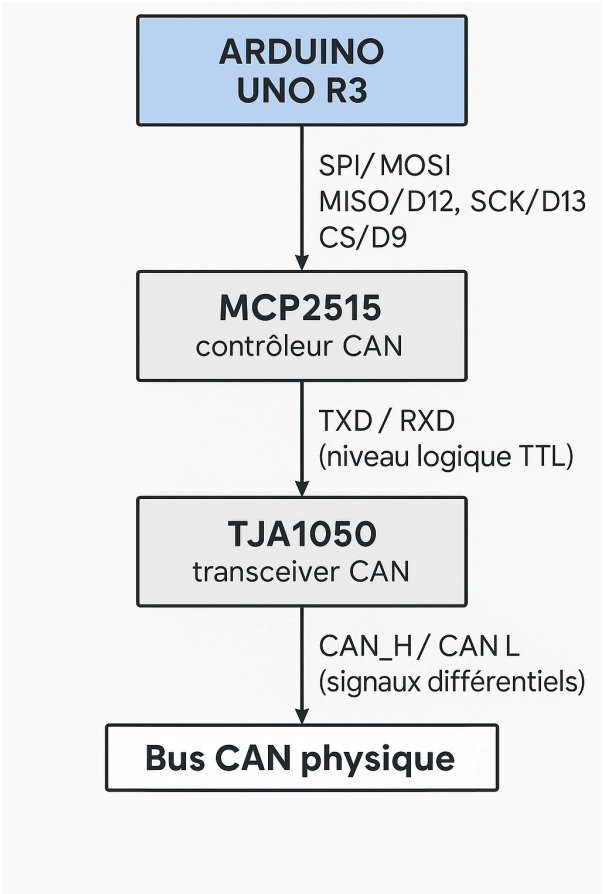
puis le passage en mode de fonctionnement normal avec
"mcp2515.setNormalMode();"

Le protocole SPI permet à l'Arduino (maître SPI) d'envoyer des commandes au MCP2515 (esclave SPI) pour écrire, lire ou envoyer des trames CAN.

Indicateurs LED sur le support CAN-BUS SHIELD indiquent l'état des transmissions :
PWR (reste ON avec l'alimentation), TX, RX, INT



b-Séquence du flux SPI:



c-Construction d'une trame CAN avant transmission:

Lorsqu'on souhaite envoyer une commande de feu vers un module VMD (feux avant ou arrière), la fonction "***envoyerCommande()"*** prépare une trame CAN. Lorsqu'on lance le code depuis l'Arduino Uno R3, celui-ci initialise le contrôleur MCP2515 via l'interface SPI, utilisant les 5 broches, le CS selon la déclaration "***MCP2515 mcp2515(9);"***. L'Arduino transmet donc une trame SPI au MCP2515 lui demandant de transmettre une trame CAN bien définie. -Cette trame est construite avec un ID étendu de 29 bits (ici *0x0E880000*), ce qui permet d'adresser de manière précise un module spécifique sur le réseau CAN (par exemple, ici, le module feux avant droit). -Vient ensuite le champ DLC (Data Length Code), qui indique ici la longueur des données (3 octets). Ensuite, les 3 octets de données utiles (Data) sont envoyés : 0x1E, 0x0F, 0x02. Ces octets ont une signification claire dans le contexte du système VMD (Véhicule Multiplexé Didactique).

- Le premier (0x1E) désigne le registre GPLAT du circuit MCP25050 du module feux, qui sert à écrire l'état des sorties (GP0 à GP7).
- Le second octet (0x0F) est un masque binaire ciblant les bits GP0 à GP3, soit les quatre sorties contrôlant les LED du bloc optique (sélectionne les bits modifiables veilleuse, code, phare, clignotant et évite de toucher aux GP4–GP7).
- Enfin, le dernier octet (0x02) signifie une valeur qui désigne l'état des sorties (LED). Dans ce cas, seulement la LED du code (GP1) qui s'allume (valeur binaire 0010 sur GP0 à GP3).

Traitement interne :

- Le **MCP2515** vérifie s'il a le droit d'émettre (arbitrage du bus)
- Si oui, il encode la trame CAN complète

Exemple de la fonction utilisé: "*envoyerCommande(REG_GPLAT, 0x0F, 0x02)"*

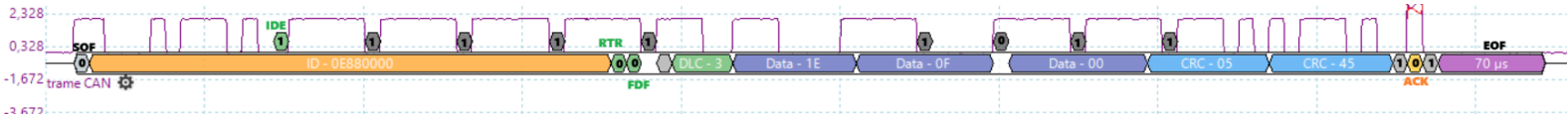
(allume le Code GP1)

La trame CAN créée à cette forme :

Champ	Valeur
can_id	0x0E880000 (module FAD) avec CAN_EFF_FLAG
can_dlc	3 (3 octets de données)
data[0]	0x1E (adresse du registre GPLAT)
data[1]	0x0F (masque : 4 bits à modifier)
data[2]	0x02 (commande à appliquer)

Une fois ces données SPI transférées, le MCP2515 génère une trame CAN à transmettre physiquement. Il ajoute un champ CRC automatiquement (`0x0545`) pour garantir l'intégrité de la trame. Cette valeur CRC est calculée à partir de tous les bits transmis précédemment, selon un polynôme standardisé (généralement `0x4599` pour CAN 2.0B).

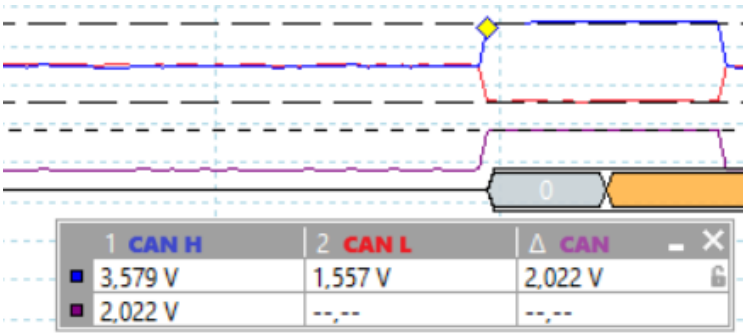
Le bit d'acquittement (ACK) suit : chaque nœud connecté au bus CAN, s'il reçoit la trame sans erreur, doit envoyer un bit dominant (`0`) dans le slot ACK. Puis, la trame se termine avec le champ EOF (End of Frame), constitué de 7 bits récessifs (`1111111`) qu'on observe comme un segment de 70 µs, ce qui est parfaitement cohérent avec un débit de **100 kbps** (chaque bit = 10 µs).



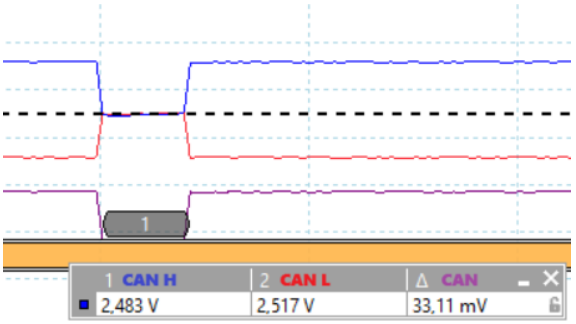
h-Conversion en signaux physiques (CAN_H et CAN_L)

Rôle du transceiver

À ce moment-là, le transceiver CAN connecté au MCP2515 généralement un MCP2551 ou TJA1050 prend le relais. Son rôle est de convertir les signaux numériques du MCP2515 TXD, RXD (niveau TTL de 0 V / 5 V) en une différence de tension différentielle sur les lignes **CAN_H** et **CAN_L**. Lorsqu'un bit dominant (`0`) est transmis, le transceiver génère typiquement CAN_H ≈ 3.5 V et CAN_L ≈ 1.5 V, soit un écart de ~2 V entre les deux lignes, comme visualisé sur le Picoscope:



Pour un bit récessif (1), les deux lignes sont ramenées à environ 2.5 V, et donc le différentiel devient nul : CAN_H - CAN_L ≈ 0 V. Cette modulation différentielle est extrêmement robuste face aux perturbations, ce qui rend le bus CAN si fiable en environnement bruyant (automobile, industrie).



État logique	CAN_H	CAN_L	Bus CAN
Récessif (1)	~2.5V	~2.5V	Aucune différence → bus au repos
Dominant (0)	~3.5V	~1.5V	Différence de potentiel = communication

Décomposition complète et continue de ta trame CAN

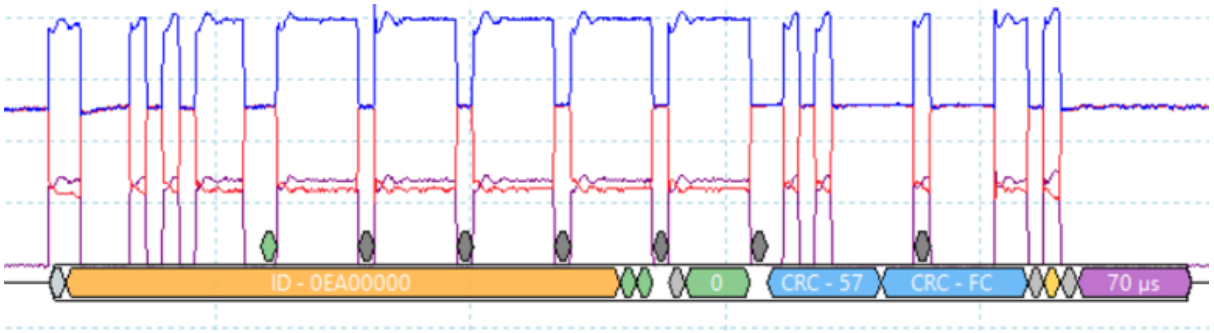
Champ	Longueur (bits)	Description
SOF	1	Start of Frame : début de la trame (toujours 0 = dominant)
ID (11 bits)	11	Bits d'identifiant prioritaire (partie haute)
SRR	1	Substitute Remote Request : 1 pour trames étendues
IDE	1	Identifier Extension : 1 → format étendu
ID (18 bits)	18	Bits d'identifiant secondaires (partie basse)
RTR	1	Remote Transmission Request : 0 si trame de données
FDF	1	Flexible Data Format, 1 sila trame est au format CANFD sinon CAN classique 2.0B
r1 (reserved)	1	Réservé (dominant : 0)
DLC	4	Data Length Code : nombre d'octets de données (0 à 8)
Data	0 à 64	Données utiles (jusqu'à 8 octets en CAN 2.0B)
CRC (15 bits)	15	Code de redondance cyclique
CRC Delimiter	1	Toujours récessif (1)
ACK Slot	1	Bit d'acquittement attendu à 0 si reçu correctement

ACK Delimiter	1	Toujours récessif (1)
EOF	7	End of Frame : toujours 1 (récessif)

Packet	Start Time	End Time	ID	RTR	FDF	DLC	Data	CRC	ACK	CRC Valid	Bit Stuffing Valid	Valid	IDE
1	120,2 ns	960 µs	0E880000	0	0	3	1E 0F 02	4B EE	0	✔	✔	✔	1
2	1,01 ms	1,71 ms	0EA00000	0	0	0		57 FC	0	✔	✔	✔	1

d-Réception & Acquittement

module cible reçoit une trame avec un ID correspondant à sa configuration, son propre transceiver re-convertit les signaux CAN_H/L en niveaux logiques TTL, que son contrôleur interne lit et interprète effectuant alors la commande sur les sorties GP0 à GP3 du module (représentant veilleuse, phare, code, clignotant, etc.). Il peut aussi générer une trame d'acquittement (ACK) qu'il renvoie sur le bus, ce que le MCP2515 détecte via le bit ACK dans la trame CAN, ce retour est ensuite signalé à l'Arduino via une interruption déclenchée sur la broche INT



Ce type de trame AIM (Acknowledge Input Message) est envoyé automatiquement par le module VMD pour confirmer la bonne réception d'une commande de type IM (Input Message) provenant de l'Arduino.

C'est un acquittement explicite qui complète le mécanisme d'ACK implicite du protocole CAN, ce qui permet à l'Arduino :

- de s'assurer que le message a bien été traité par le bon module (grâce à l'ID unique)
- de détecter si un module ne répond pas, et donc de gérer les erreurs

Méthodes pour récupérer la trame AIM dans le MCP2515

- **Par interruption (INT)**

Le MCP2515 dispose d'une sortie d'interruption (broche INT), qui passe à l'état bas (LOW) dès qu'une nouvelle trame est reçue.

Le MCP2515 déclenche une interruption sur la broche INT (souvent reliée à D2 de l'Arduino Uno) dès qu'un message est reçu dans ses buffers RX.

- **Par sondage (polling)**

L'Arduino vérifie régulièrement dans la boucle principale (loop()) si une trame CAN est disponible. Appels répétés à "***mcp2515.readMessage(&frame)"*** pour tester la présence d'une trame, Si une trame est présente, elle est lue et traitée.

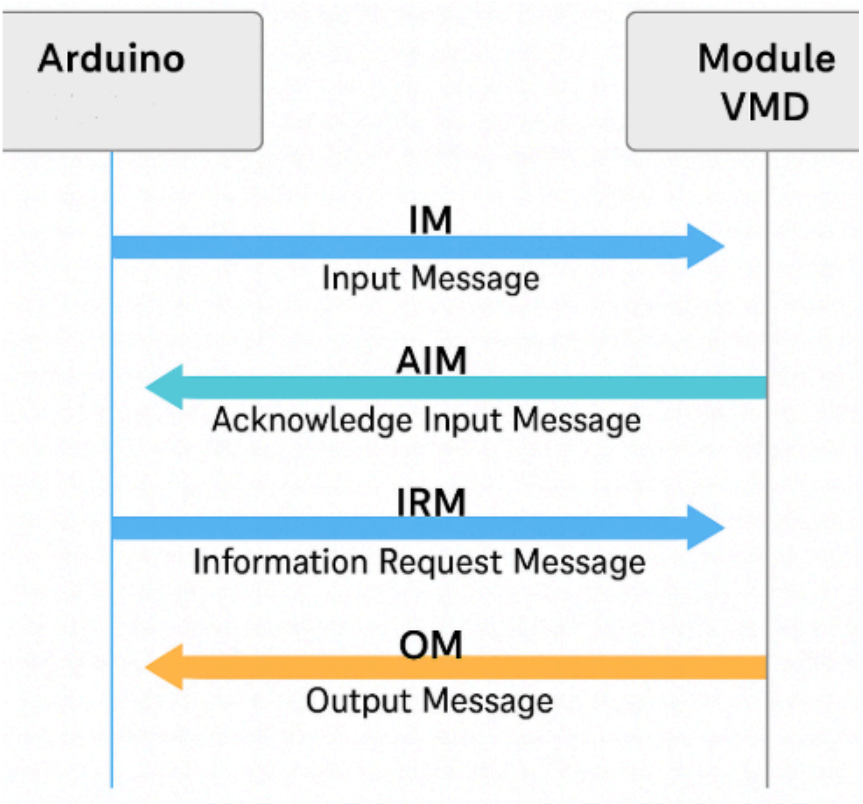
Ainsi, le flux de données suit une chaîne complète depuis le microcontrôleur (exécution du code), passant par la communication SPI, la construction d'une trame CAN via le MCP2515, la conversion électrique vers le bus CAN par le MCP2551, la réception par les modules de feux, et enfin la validation via trame d'ACK.

Autres types de messages utilisées:
IRM (Information Request Message)

- **Émetteur :** Arduino
- **Récepteur :** Module VMD
- **Contenu :** Demande de lecture d'un registre (ex. GPLAT)
- **Utilisé pour :** Lire l'état d'un bouton ou d'une entrée du commodo
- **Exemple :** L'Arduino envoie une IRM toutes les 200 ms dans le TP4 pour lire l'état des BP du commodo.

OM (Output Message):

- **Émetteur :** Module VMD
- **Récepteur :** Arduino (ou tout noeud CAN)
- **Contenu :** Message spontané contenant l'état des entrées
- **Utilisé pour :** Notification d'événement (ex : changement d'état d'un bouton)
- **Remarques :**
    - Générés automatiquement par certains modules (ex : le commodo VMD)
    - Utilisé dans TP2 et TP4 : si un BP est pressé, un OM est émis sans requête IRM



Message	Déclenché par	Reçu automatiquement ?	Interruption utile ?	Contenu transmis
IM	Arduino	/	Oui (envoie)	Commande (ex : allumer feu)
AIM	Module VMD	OUI (après IM)	Oui	Acquittement : "commande bien reçue"
IRM	Arduino	/	Oui (envoie)	Requête d'état
OM	Module VMD	OUI (auto ou IRM)	Oui	État actuel du module (boutons, entrées)