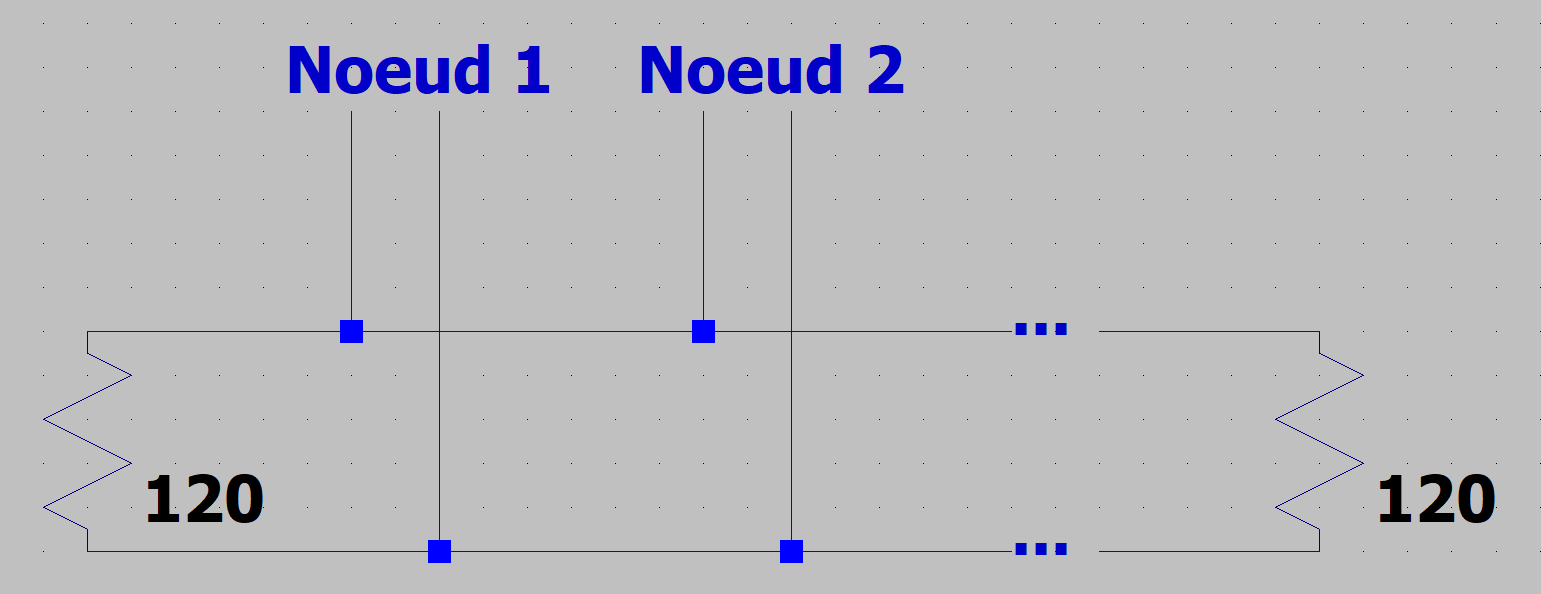
**Formation : Bus CAN**

**Introduction**

Le **bus de données CAN** (*Controller Area Network*) est un bus système série qui met en application une approche connue sous le nom de multiplexage.

Il est normalisé avec la norme ISO 11898.

**Représentation schématique**



**Hardware**

Afin de s’affranchir des problèmes d’impédance (et donc des valeurs de résistances sur le bus), on se place systématiquement dans le cas ou la longueur des fils des nœuds (fils rouges) est très inférieur à la longueur les fils du bus (fils bleus).

**Les messages**

Les messages envoyés sont tous du format suivant :

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SOF | Identifier | RTR | IDE | R0 | DLC | Data | CRC | ACK | EOF | IFS |

* SOF : Start Of Frame.
* Identifier et RTR (Remote Transmission Request) : distinction entre une trame de donnée et une trame de demande de données (les Remote Frame ne sont pas utilisées à l’EPSA pour le moment).
* IDE (IDentifier Extension) : distinction entre une trame de CAN standard et une trame de CAN étendue.
* DLC (Data Length Code) : indique la quantité d’octets de données.
* CRC (Cyclic Redundant Check) : verification de l’íntégrité du message.
* ACK (ACKnowledge).
* EOF (End Of Frame).
* IFS (Intermission Frame Space) : nombre minimal de bits séparant deux messages consécutifs.

**CAN Standard et CAN étendu**

Le format de trame CAN dit standard, également appelé CAN 2.0 A, admet un identificateur d’une longueur de 11bis.

Le format de trame CAN dit étendu, également appelé CAN 2.0 B, admet un identificateur de longueur 29 bits.

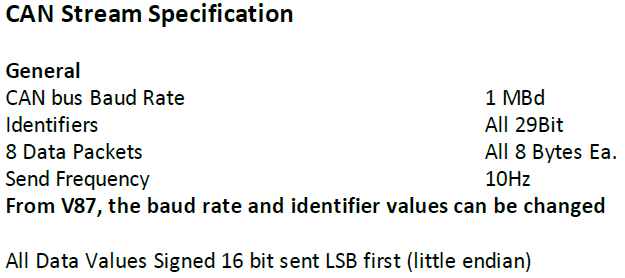
**L’endianness**

Il en existe trois types : le little-endian, le big-endian, le mixed-endian.

Voici un exemple pour en comprendre le principe. On considère le nombre 0x12345678 :

* **Little-endian :  0x78 0x56 0x34 0x12** (cas du DTA)
* Big-endian : 0x12 0x34 0x56 0x78
* Mixed-endian : 0x34 0x12 0x78 0x56

**Le DTA** (manuel p205)



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Data 0  Data 1 | Data 2  Data 3 | Data 4  Data 5 | Data 6  Data 7 |
| 0x2000 | RPM | TPS  (%) | Water Temp  (°C) | Air Temp  (°C) |
| 0x2001 | Manifold Press  (kPa) | Lambda  (\*1000) | Speed  (\*10 km/h) | Oil Press  (kPa) |
| 0x2002 | Fuel Press  (kPa) | Oil Temp  (°C) | Battery  (\*10 V) | Fuel Consumption  (\*10 L/h) |
| 0x2003 | Gear | Advance  (\*10 °) | Injection Time  (\*100 ms) | Fuel Consumption  (\*10 L/100km) |
| 0x2004 | Ana 1  (mV) | Ana 2  (mV) | Ana 3  (mV) | Cam Advance  (\*10 °) |
| 0x2005 | Cam Targ  (\*10 °) | Cam PWM  (\*10 %) | Crank Errors  (Number of errors) | |
| 0x2006 | Cam 2 Adv  (\*10 °) | Cam 2 Targ  (\*10 °) | Cam 2 PWM  (\*10 %) | External 5V  (mV) |
| 0x2007 | Inj Duty Cycle  (%) | Lambda PID Trag  (\*10 %) | Lambda PID Adj  (\*10 %) | ECU Switches  (See below) |
| 0x2008 | RD Speed  (\*10 km/h) | R UD Speed  (\*10 km/h) | LD Speed  (\*10 km/h) | R UD Speed  (\*10 km/h) |
| 0x2009 | Right lambda  (\*1000) |  |  |  |

ECU Switches

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bit | 0 | 1 | 2 | 3 | 4 | 5 |
| Value | Launch Button | Launch Active | Traction ON | Traction WET | Fuel Pump ON | Fan ON |

Addition IDs

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Data 0 | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

**La bibliothèque arduino** <mcp\_can.h> **(Optimus)**

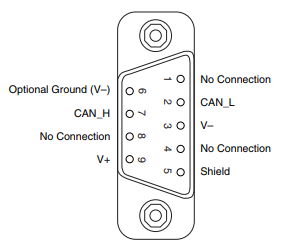
Git : <https://github.com/coryjfowler/MCP_CAN_lib>

**La bibliothèque arduino** FlexCAN

Git : <https://github.com/teachop/FlexCAN_Library>

**PCAN View**

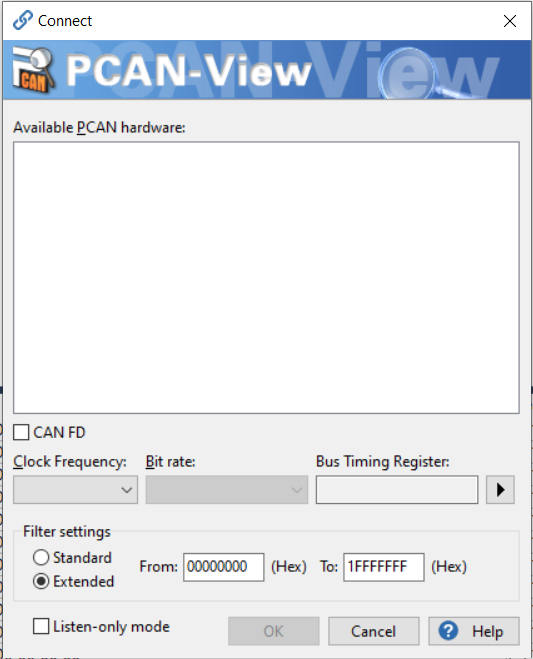
On dispose de deux adaptateurs PCAN-USB. Le câblage est le suivant :



On utilise ensuite le logiciel PCAN View. (Installer et driver dans le dossier).

Ce logiciel permet des envois cycliques de trames fixes.

**Connecter un appareil**

Il faut **toujours** sélectionner Extended comme format de trame.

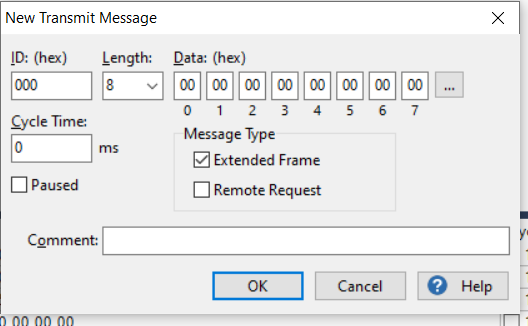
La fréquence d’horloge correspond à celle du DTA (ie 50 Hz).

En bas de page du logiciel, on peut lire le statut de l’appareil :

* Not Connected (dans ce cas, il faut connecter l’adaptater CAN au PC)
* Connected

Status : OK (dans ce cas tout va bien)

Status : Bus Heavy (dans ce cas, il faut vérifier le câblage du CAN, c’est souvent un pb de résistance…)

**Envoyer des trames**

On sélectionne ici aussi Extended comme format de trame.

On défini ensuite l’ID et les Data que l’on souhaite transmettre pour simuler le DTA.

Il s’agit d’un premier moyen simple de tester le code de réception du CAN sur l’arduino.

**Matlab**

On peut utiliser Matlab pour faire un envoie de trames variables dans le temps. Voici un exemple de script d’envoie (avec les même adaptateurs PCAN-USB)

|  |
| --- |
| % CAN Config  CANCh = canChannel('PEAK-System','PCAN\_USBBUS1');  configBusSpeed(CANCh,1000000)  % CAN Sending  start(CANCh)    for i=1:N  message = canMessage(8192,true,8);  message.Data = (CAN Message);  message.Remote = false;    transmit(CANCh,message)  pause(0.02)  end    stop(CANCh) |

**Kvaser Database Editor**

Logiciel à tester

**Vérification 'sur banc'**

L'objectif est ici de simuler le plus fidèlement possible les envois de messages du DTA. Pour ce faire on se propose d'utiliser les logs CAN d'Optimus acquis grâce au racecapture.

Voici les différentes étapes du travail :

* Extraire des logs d'Optimus les messages CAN envoyés par le DTA.
* Regénérer avec Matlab les messages envoyés par le DTA d'Optimus pendant un run.
* Envoyer sur une maquette du CAN (contenant principalement la carte avant) grâce au script fourni dans la section 'Matlab'.
* Observer le bon fonctionnement de la carte avant grâce au moniteur série.
* Corriger les éventuelles erreurs et réitérer.

|  |
| --- |
| == Vérification 'sur banc' ==  L'objectif est ici de simuler le plus fidèlement possible les envois de messages du DTA. Pour ce faire on se propose d'utiliser les logs CAN d'Optimus acquis grâce au racecapture.  Voici les différentes étapes du travail :  \* Extraire des logs d'Optimus les messages CAN envoyés par le DTA.  \* Regénérer avec Matlab les messages envoyés par le DTA d'Optimus pendant un run.  \* Envoyer sur une maquette du CAN (contenant principalement la carte avant) grâce au script fourni dans la section 'Matlab'.  \* Observer le bon fonctionnement de la carte avant grâce au moniteur série.  \* Corriger les éventuelles erreurs et réitérer. |