

6. Analyse et implémentation du protocole CAN pour le contrôle des modules via MCP25050 :

6.1. Introduction :

Pour contrôler les cartes du **Module CAN01A ou le VMD** via le bus CAN, il faut envoyer des trames CAN à le MCP25050.

Une première trame qui configure les broches en **mode sortie ou en mode entrée** en écrivant dans le registre **GDDR (registre est uniquement responsable de définir l'état des broches en mode entrée ou sortie)**.

Et une deuxième trame qui définit l'état logique des sorties (haut ou bas) en écrivant dans le registre **GPLAT (registre gère l'état des entrées et des sorties)**.

La première étape consiste à configurer les broches en **mode sortie** via GDDR, une fois les broches configurées, en envoi une **deuxième trame** ciblant GPLAT pour définir l'état des sorties (allumer/éteindre les lampes). Pour effectuer cela, il est nécessaire de définir les paramètres de la trame CAN.

6.2. Remplissage des principaux champs des trames CAN :

Pour définir les paramètres de la trame CAN, nous nous basons sur les documentations fournies par **Didalab** concernant le matériel utilisé.

Nous commencerons par **définir l'ID de la trame**, qui permet d'identifier le message et sa priorité sur le bus. Cet **identifiant (ID)** peut être en **format standard (11 bits)** ou **étendu (29 bits)**, selon la configuration du réseau CAN utilisé dans notre projet.

Une fois l'ID déterminé, nous préciserons les autres champs essentiels de la trame, tels que :

- **DLC (Data Length Code)** : indique le nombre d'octets de données dans la trame (de 0 à 8)
- **Données (Data Field)** : contient les informations à transmettre aux modules du système

Les autres paramètres, tels que le **CRC (Cyclic Redundancy Check)**, le **bit ACK** et les champs de contrôle, sont gérés automatiquement par le microcontrôleur.

6.2.1. Remplissage du champ ID :

Chaque carte du modèle **CAN01A** possède un **ID unique** pour chaque type de message transmis ou reçu sur le bus CAN.

Dans notre projet, les messages échangés suivent une structure bien définie, avec des identifiants spécifiques en fonction du type de communication. On distingue plusieurs types de messages, notamment :

- **IM (Input Message)** : Utilisée pour transmettre des données.
- **IRM (Input Request Message)** : Permet à demander des données.
- **OM (Output Message)** : C'est une repense du message IRM.

Chaque ID est propre à un type de message et à une carte spécifique, ce qui permet d'assurer une communication structurée et d'éviter les conflits sur le bus.

Pour trouver les **ID des cartes** du modèle **CAN01A**, nous pouvons nous référer directement sur le modèle CAN01A ou dans le tableau suivant **d'identification des différents modules CAN** (fourni par Didalab) : Ce tableau répertorie les **ID attribués à chaque carte** en fonction du **type de message** échangé.

Registre MCP25025 Et fonction	- SIDH (en bin)	- SIDL (en bin)	SIDH (Hex)	SIDL (Hex)	Identificateur (Hex) (! sur 29 bits)	Labels définis dans fichier CAN_VMD.h
-------------------------------	-----------------	-----------------	------------	------------	--------------------------------------	---------------------------------------

Nœud "Commodo feux"

RXF0 -> IRM et OM	001 0 10 00	001 - 1-xx	28	28	0504 xx xx	T_Ident_IRM_Commodo_Feux
RXF1 -> IM	001 0 10 00	010 - 1-xx	28	48	0508 xx xx	T_Ident_IM_Commodo_Feux
TXD0 -> On Bus	001 0 10 00	100 - 1-xx	28	88	0510 xx xx	
TXD1 -> Acq IM	001 0 10 01	000 - 1-xx	29	08	0520 xx xx	T_Ident_AIM_Commodo_Feux
TXD2 -> Mes. Auto.	001 0 10 10	000 - 1-xx	2A	08	0540 xx xx	

Nœud "Feux avant gauche"

RXF0 -> IRM et OM	011 1 00 00	001 - 1-xx	70	28	0E04 xx xx	T_Ident_IRM_FVG
RXF1 -> IM	011 1 00 00	010 - 1-xx	70	48	0E08 xx xx	T_Ident_IM_FVG
TXD0 -> On Bus	011 1 00 00	100 - 1-xx	70	88	0E10 xx xx	
TXD1 -> Acq IM	011 1 00 01	000 - 1-xx	71	08	0E20 xx xx	T_Ident_AIM_FVG
TXD2 -> Mes. Auto.	011 1 00 10	000 - 1-xx	72	08	0E40 xx xx	

Nœud "Feux avant droit"

RXF0 -> IRM et OM	011 1 01 00	001 - 1-xx	74	28	0E84 xx xx	T_Ident_IRM_FVD
RXF1 -> IM	011 1 01 00	010 - 1-xx	74	48	0E88 xx xx	T_Ident_IM_FVD
TXD0 -> On Bus	011 1 01 00	100 - 1-xx	74	88	0E90 xx xx	
TXD1 -> Acq IM	011 1 01 01	000 - 1-xx	75	08	0EA0 xx xx	T_Ident_AIM_FVD
TXD2 -> Mes. Auto.	011 1 01 10	000 - 1-xx	76	08	0EC0 xx xx	

Nœud "Feux arrière gauche"

RXF0 -> IRM et OM	011 1 10 00	001 - 1-xx	78	28	0F04 xx xx	T_Ident_IRM_FRG
RXF1 -> IM	011 1 10 00	010 - 1-xx	78	48	0F08 xx xx	T_Ident_IM_FRG
TXD0 -> On Bus	011 1 10 00	100 - 1-xx	78	88	0F10 xx xx	
TXD1 -> Acq IM	011 1 10 01	000 - 1-xx	79	08	0F20 xx xx	T_Ident_AIM_FRG
TXD2 -> Mes. Auto.	011 1 10 10	000 - 1-xx	7A	08	0F40 xx xx	

Nœud "Feux arrière droit"

RXF0 -> IRM et OM	011 1 11 00	001 - 1-xx	7C	28	0F84 xx xx	T_Ident_IRM_FRD
RXF1 -> IM	011 1 11 00	010 - 1-xx	7C	48	0F88 xx xx	T_Ident_IM_FRD
TXD0 -> On Bus	011 1 11 00	100 - 1-xx	7C	88	0F90 xx xx	
TXD1 -> Acq IM	011 1 11 01	000 - 1-xx	7B	08	0FA0 xx xx	T_Ident_AIM_FRD
TXD2 -> Mes. Auto.	011 1 11 10	000 - 1-xx	7E	08	0FC0 xx xx	

On remarque que les identifiants (ID) dans le tableau contiennent des valeurs incomplètes représentées par "xx xx". Ces valeurs doivent être complétées en fonction du type de message utilisé (IM, IRM, OM).

Pour déterminer les valeurs exactes à insérer, nous nous basons sur un second tableau **de command messages** (Le tableau est en binaire) qui fournit les informations nécessaires au remplissage des parties manquantes des **identifiants** et de champ **DATA**. Ce tableau associe chaque type de message à une structure spécifique d'identifiant, nous permettant ainsi de finaliser les ID de manière précise et cohérente.

Information Request Messages (to MCP2502X/5X)																										
	Standard ID				Extended ID				Data Bytes																	
0	1	9	8	7	6	5	4	3	2	1	R/I	DLC	1	1	RXBElD8 (8 bits)	RXBElD0 (8 bits)										
Read A/D Regs	x	x	x	x	x	x	x	x	x	x	1	0	0	0	8*	x	x	xxxx xxxx	xxxx *000	n/a						
Read Control Regs	x	x	x	x	x	x	x	x	x	x	1	0	1	1	7*	x	x	xxxx xxxx	xxxx *001	n/a						
Read Config Regs	x	x	x	x	x	x	x	x	x	x	1	0	1	0	5*	x	x	xxxx xxxx	xxxx *010	n/a						
Read CAN Error	x	x	x	x	x	x	x	x	x	x	1	0	0	1	3*	x	x	xxxx xxxx	xxxx *011	n/a						
Read PWM Config	x	x	x	x	x	x	x	x	x	x	1	0	1	0	6*	x	x	xxxx xxxx	xxxx *100	n/a						
Read User Mem	x	x	x	x	x	x	x	x	x	x	1	1	0	0	8*	x	x	xxxx xxxx	xxxx *110	n/a						
Read User Mem (bank)	x	x	x	x	x	x	x	x	x	x	1	1	0	0	8*	x	x	xxxx xxxx	xxxx *110	n/a						
Read Register	x	x	x	x	x	x	x	x	x	x	1	0	0	0	1*	x	x	addr	xxxx *111	n/a						

Output Messages (from MCP2502X/5X)																											
	Standard ID				Extended ID				Data Bytes																		
0	1	9	8	7	6	5	4	3	2	1	R/I	DLC	1	1	RXBElD8 (8 bits)	RXBElD0 (8 bits)											
Read A/D Regs	x	x	x	x	x	x	x	x	x	x	0	1	0	0	8	x	x	xxxx xxxx	xxxx *000	I0INTFL	GPIO	AN0H	AN1H	AN10L	AN2H	AN3H	AN23L
Read Control Regs	x	x	x	x	x	x	x	x	x	x	0	1	0	1	7	x	x	xxxx xxxx	xxxx *001	ADCON0	ADCON1	OPTREG1	OPTREG2	STCON	I0INTEN	I0INTPO	n/a
Read Config Regs	x	x	x	x	x	x	x	x	x	x	0	1	0	0	5	x	x	xxxx xxxx	xxxx *010	DDR	GPIO	CNF1	CNF2	CNF3	n/a	n/a	n/a
Read CAN Error	x	x	x	x	x	x	x	x	x	x	0	1	0	1	3	x	x	xxxx xxxx	xxxx *011	EFLG	TEC	REC	n/a	n/a	n/a	n/a	
Read PWM Config	x	x	x	x	x	x	x	x	x	x	0	1	0	1	6	x	x	xxxx xxxx	xxxx *100	PR1	PR2	T1CON	T2CON	PWM1DCH	PWM2DCH	n/a	n/a
Read User Mem (bank1)	x	x	x	x	x	x	x	x	x	x	0	1	0	0	8	x	x	xxxx xxxx	xxxx *101	USERID0	USERID1	USERID2	USERID3	USERID4	USERID5	USERID6	USERID7
Read User Mem (bank)	x	x	x	x	x	x	x	x	x	x	0	1	0	0	8	x	x	xxxx xxxx	xxxx *110	USERID8	USERID9	USERID10	USERID11	USERID12	USERID13	USERID14	USERID15
Read Register	x	x	x	x	x	x	x	x	x	x	0	1	0	0	1	x	x	addr	xxxx *111	value	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Input Messages (to MCP2502X/5X)																											
	Standard ID				Extended ID				Data Bytes																		
0	1	9	8	7	6	5	4	3	2	1	R/I	DLC	1	1	RXBElD8 (8 bits)	RXBElD0 (8 bits)											
Write Register	x	x	x	x	x	x	x	x	x	x	0	1	0	1	3	x	x	xxxx xxxx	xxxx x000	addr	mask	value	n/a	n/a	n/a	n/a	
Write TX Message ID 0	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x	x	xxxx xxxx	xxxx x001	TX0SIDH	TX0SDL	TX0EID0	n/a	n/a	n/a	n/a	
Write TX Message ID 1	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x	x	xxxx xxxx	xxxx x010	TX1SIDH	TX1SDL	TX1EID0	n/a	n/a	n/a	n/a	
Write TX Message ID 2	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x	x	xxxx xxxx	xxxx x011	TX2SIDH	TX2SDL	TX2EID0	n/a	n/a	n/a	n/a	
Write I/O Configuration	x	x	x	x	x	x	x	x	x	x	0	1	0	0	5	x	x	xxxx xxxx	xxxx x100	I0INTEN	I0INTPO	DDR	OPTREG1	ADCON1	n/a	n/a	
Write RX Mask	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x	x	xxxx xxxx	xxxx x101	RXMSIDH	RXMSIDL	RXMEID0	n/a	n/a	n/a	n/a	
Write RX Filter0	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x	x	xxxx xxxx	xxxx x110	RXF0SIDH	RXF0SDL	RXF0EID0	n/a	n/a	n/a	n/a	
Write RX Filter1	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x	x	xxxx xxxx	xxxx x111	RXF1SIDH	RXF1SDL	RXF1EID0	RXF1EID0	n/a	n/a	n/a	n/a

Prenons un exemple pour bien comprendre comment compléter l'identifiant (ID) d'une trame CAN :

Récapitulons étape par étape pour l'envoi d'un **IRM (Information Request Message)** à la carte 8 entrées de Commodo Feux :

1. Début de l'ID :

- Le tableau d'identification des différents modules CAN nous donne **05 04 xx xx** pour un message IRM envoyé à ce nœud.
- On garde **05 04** et on complète les **xx xx** avec le tableau **de command messages**.

2. Choix de la bonne ligne dans le tableau de command messages :

- Un message **IRM** est une requête d'information, donc on se réfère à la ligne "**Read Register**".
- Cette ligne nous donne la structure de l'Extended ID : **addr / xxxx x111**.
- Les **deux premiers xx** doivent être remplacés par l'adresse du registre contenant l'état des entrées (**GPLAT, registre gère l'état des entrées et des sorties**).

3. Complétion des derniers octets :

- Les **deux derniers xx** correspondent à **xxxx x111**, on remplace **xxxx** par 0 → **07**.

ID final à envoyer : 05 04 [Adresse_Registre] 07

6.2.2 Remplissage du champ DLC :

Le champ DLC indique le nombre d'octets présents dans le champ DATA de la trame CAN. Il peut prendre une valeur de 0 à 8, selon la quantité de données envoyées.

Si on veut envoyer 3 octets dans le champ DATA, alors DLC = 0x03. Si on envoie 8 octets, alors DLC = 0x08.

6.2.3 Remplissage du champ DATA :

Si nous voulons envoyer un message de type IM, on est en mode **écriture dans un registre (on se réfère à la ligne "Write Register")** à la carte **Commodo Feux**, nous devons configurer les broches en mode **entrée** ou en mode **sortie**.

Pour cela, nous devons compléter l'**ID du message** et envoyer **3 octets** dans le champ **DATA** :

- L'**ID du message IM** pour le commodo feux est **05 08 xx xx**.
- D'après la structure du tableau IM, la suite de l'**ID** suit le format **xxxx xxxx / xxxx x000**.
- En remplaçant les **xxxx** par **0**, nous obtenons **00 00**.

ID final du message IM : 05 08 00 00

Ensuite, nous devons compléter le champ **DATA**, qui contient :

1. **Octet d'adresse** : L'adresse du registre responsable de la configuration des broches en entrée ou sortie (**GPDDR**).
2. **Octet de masque (Mask)** : Permet de sélectionner les bits à modifier.
3. **Octet de valeur (Value)** : Définit si une broche est en **entrée (1)** ou en **sortie (0)**.

6.2.3.1. Gestion des données avec le masque :

Lorsque la valeur d'un bit de **masque** est à **0**, nous ignorons la donnée correspondante dans **l'octet de valeur (Value)**. En d'autres termes :

- Si un bit de **masque** vaut **0**, la donnée associée dans **(Value)** n'est **pas prise en compte**.
- Si un bit de **masque** vaut **1**, la donnée associée est **appliquée**.

Exemple d'utilisation d'un masque :

Avec un **octet de valeur** de **0xFF** et un masque de **0x0F** :

- **Masque 0x0F :**

- Les bits **GP3 à GP0** sont pris en compte.
- Les bits **GP7 à GP4** sont ignorés.
- **Donnée : 0xFF :**
 - Les bits **GP7 à GP4** (qui valent 1) sont ignorés grâce au masque.
 - Les bits **GP3 à GP0** sont configurés comme suit :
 - GP3 : 1 (entrée).
 - GP2 : 1 (entrée).
 - GP1 : 1 (entrée).
 - GP0 : 1 (entrée).

Pour trouver les adresses des registres **GPDDR** (registre de direction) et **GPLAT** (registre de sortie/entrée), il faut consulter les tableaux d'affectation des registres dans la documentation du MCP25050 fournie par **didalab**.

Addr*	Name	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Value on POR	Value on RST
1Fh**	GPDDR	—	DDR6	DDR5	DDR4	DDR3	DDR2	DDR1	DDR0	-111 1111	-111 1111
18h	EFLG	ESCF	RBO	TXBO	TXEP	RXEP	TXWAR	RXWAR	EWARN	0000 0000	0000 0000
19h	TEC	Transmit Error Counters								0000 0000	0000 0000
1Ah	REC	Receive Error Counters								0000 0000	0000 0000
50h	ADRES3H	AN3.9	AN3.8	AN3.7	AN3.6	AN3.5	AN3.4	AN3.3	AN3.2	xxxx xxxx	uuuu uuuu
51h	ADRES3L	AN3.1	AN3.0	—	—	—	—	—	—	xx-- ----	uu-- ----
52h	ADRES2H	AN2.9	AN2.8	AN2.7	AN2.6	AN2.5	AN2.4	AN2.3	AN2.2	xxxx xxxx	uuuu uuuu
53h	ADRES2L	AN2.1	AN2.0	—	—	—	—	—	—	xx-- ----	uu-- ----
54h	ADRES1H	AN1.9	AN1.8	AN1.7	AN1.6	AN1.5	AN1.4	AN1.3	AN1.2	xxxx xxxx	uuuu uuuu
55h	ADRES1L	AN1.1	AN1.0	—	—	—	—	—	—	xx-- ----	uu-- ----
56h	ADRES0H	AN0.9	AN0.8	AN0.7	AN0.6	AN0.5	AN0.4	AN0.3	AN0.2	xxxx xxxx	uuuu uuuu
57h	ADRES0L	AN0.1	AN0.0	—	—	—	—	—	—	xx-- ----	uu-- ----

Addr	Name	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Value on POR	Value on RST
1Eh	GPPIN	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000	0000 0000
34h	GPDDR *	—	DDR6	DDR5	DDR4	DDR3	DDR2	DDR1	DDR0	-111 1111	-111 1111
00h	IOINTEN	GP7TXC	GP6TXC	GP5TXC	GP4TXC	GP3TXC	GP2TXC	GP1TXC	GP0TXC	0000 0000	0000 0000
01h	IOINTPO	GP7POL	GP6POL	GP5POL	GP4POL	GP3POL	GP2POL	GP1POL	GP0POL	0000 0000	0000 0000
0Eh	ADCON0	ADON	T0PS2	T0PS1	T0PS0	GO/DONE	—	CHS1	CHS0	0000 0-00	0000 0-00
0Fh	ADCON1	ADCS1	ADCS0	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0	0000 0000	0000 0000
2Ch	ADCMP3H	AN3CMP.	AN3CMP.8	AN3CMP.7	AN3CMP.6	AN3CMP.5	AN3CMP.4	AN3CMP.3	AN3CMP.2	xxxx xxxx	uuuu uuuu
2Dh	ADCMP3L	AN3CMP.	AN3CMP.0	—	—	Reserved			ADPOL	xx-- ----x	uu-- ----u
2Eh	ADCMP2H	AN2CMP.	AN2CMP.8	AN2CMP.7	AN2CMP.6	AN2CMP.5	AN2CMP.4	AN2CMP.3	AN2CMP.2	xxxx xxxx	uuuu uuuu
2Fh	ADCMP2L	AN2CMP.	AN2CMP.0	—	—	Reserved			ADPOL	xx-- ----x	uu-- ----u
30h	ADCMP1H	AN1CMP.	AN1CMP.8	AN1CMP.7	AN1CMP.6	AN1CMP.5	AN1CMP.4	AN1CMP.3	AN1CMP.2	xxxx xxxx	uuuu uuuu
31h	ADCMP1L	AN1CMP.	AN1CMP.0	—	—	Reserved			ADPOL	xx-- ----x	uu-- ----u
32h	ADCMP0H	AN0CMP.	AN0CMP.8	AN0CMP.7	AN0CMP.6	AN0CMP.5	AN0CMP.4	AN0CMP.3	AN0CMP.2	xxxx xxxx	uuuu uuuu
33h	ADCMP0L	AN0CMP.	AN0CMP.0	—	—	Reserved			ADPOL	xx-- ----x	uu-- ----u
10h	STCON	STEM	STMS	STBF1	STBF0	STM3	STM2	STM1	STM0	0xxx xxxx	0uuuu uuuu

Remarque : le registre **GPPIN** est équivalent au **GPLAT**.

6.3. Exemple de trame du TP1 de VMD :

Cette trame configure les broches en mode **sortie (feux arrière droit)**.

```
T_IM_Feux.trame_info.registre=0x00;
T_IM_Feux.trame_info.champ.extend=1; // On travaille en mode étendu
T_IM_Feux.trame_info.champ.dlc=0x03; // Il y aura 3 données de 8 bits (3 octets envoyés)
T_IM_Feux.ident.extend.identificateur.ident=0xF880000; // C'est l'identificateur du bloc optique arrière droit
T_IM_Feux.data[0]=0x1F; // première donnée -> "Adresse" du registre concernée (GPDDR donne la direction des I/O)
T_IM_Feux.data[1]=0x7F; // deuxième donnée -> "Masque" -> les sorties sont sur les 4 bits de poids faibles
T_IM_Feux.data[2]=0xF0; // troisième donnée -> "Valeur" -> Les sorties sont sur les 4 bits lsb
```

Le nom de la trame : **T_IM_Feux**

- **T_IM_Feux.trame_info.registre=0x00** : initialise tous les bits de registre à zéro.
- **T_IM_Feux.trame_info.champ.extend=1** : Mode étendu activé.
- **T_IM_Feux.trame_info.champ.dlc=0x03** : Envoi de 3 données (car il s'agit d'un message **write register** en mode IM).
- **T_IM_Feux.ident.extend.identificateur.ident=0xF880000** : Identificateur spécifique au modèle arrière droit pour un message IM.

Contenu du champ champ data :

- **T_IM_Feux.data[0]=0x1F** : l'adresse de registre **GPDDR**.
- **T_IM_Feux.data[1]=0x7F** : Masque appliqué, seuls les 7 bits de poids faibles sont modifiés (le bit 7 reste inchangé car il est en mode entrée et non reprogrammable).
- **T_IM_Feux.data[2]=0xF0** : Les bits **GP7 à GP4** sont configurés en **entrée** et les bits **GP3 à GP0** sont configurés en **sortie**

Remarque : Les bits GP7 à GP4 du circuit MCP25050 sont déjà programmés en mode entrée et ne peuvent pas être reprogrammés, car le circuit n'est pas reprogrammable.

En conséquence, de notre travail sur le projet nous ne sommes pas intéressés par un masque comme de 0x7F, qui inclut des bits inutilisables (**GP7 à GP4**). À la place, nous pouvons utiliser un masque de 0x0F, qui cible uniquement les bits **GP3 à GP0**, configurables en mode sortie.

4. Analyse et implémentation du protocole CAN pour le contrôle des modules via MCP25050 :

4.1. Introduction :

Pour contrôler une carte **Module 4 sorties de puissance** via le bus CAN, il faut envoyer des trames CAN à le MCP25050.

Une première trame qui configure les broches en **mode sortie ou en mode entrée** en écrivant dans le registre **GPDDR** (**registre est uniquement responsable de définir l'état des broches en mode entrée ou sortie**).

Et une deuxième trame qui définit l'état logique des sorties (haut ou bas) en écrivant dans le registre **GPLAT** (**registre gère l'état des sorties**).

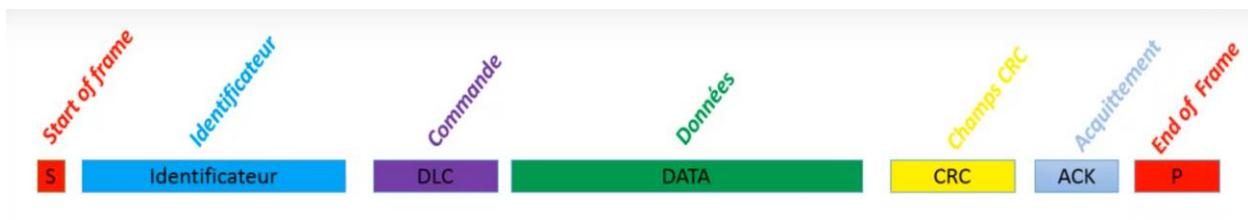
La première étape consiste à configurer les broches en **mode sortie** via GPDDR, une fois les broches configurées, en envoi une **deuxième trame** ciblant GPLAT pour définir l'état des sorties (allumer/éteindre les lampes).

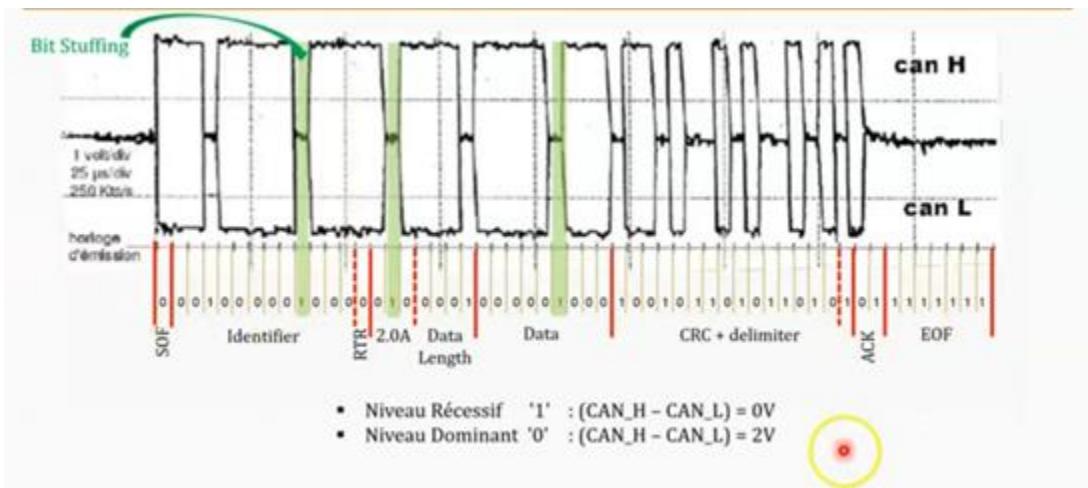
Pour effectuer cela, il est nécessaire de définir les paramètres de la trame CAN. Avant de les détailler, il est important de comprendre ce qu'est une trame CAN et quels éléments elle contient.

4.2. Définition d'une trame :

Une **trame CAN** est un message structuré qui permet la communication entre différents modules ou dispositifs connectés à un bus CAN. Elle est composée des éléments suivants :

1. **Start of frame** : début de la trame.
2. **Identifiant (ID)** : Un numéro unique qui identifie le message et son destinataire.
3. **DLC (Data Length Code)** : Spécifie le nombre d'octets de données que contient la trame.
4. **Données (Data)** : Contient les informations nécessaires, comme l'adresse d'un registre, un masque, et une valeur à appliquer.
5. **Contrôle** : Comprend des bits de gestion (CRC, ACK) pour assurer la fiabilité de la communication. Ces (CRC, ACK) champs sont générés et gérés automatiquement par le microcontrôleur.
6. **End of frame** : fin de la trame.





Les trames CAN se divisent en deux formats, **standard** et **étendu**, en fonction de la longueur de leur identifiant.

Trame standard :

- L'identifiant est codé sur **11 bits**.
- C'est le format original de la norme CAN 2.0A.

Trame étendue :

- L'identifiant est codé sur **29 bits**.
- Utilisé dans la norme CAN 2.0B.

4.3. Types de messages CAN :

- IM (Input Message)** : Ce type de trame est utilisé pour **envoyer des commandes** ou des informations vers les modules (par exemple, allumer un feu).
- IRM (Information Request Message)** : Ce type de trame est utilisé pour **demander une information** à un module (par exemple, l'état d'un feu).
- OM (Output Message)** : C'est la **réponse** d'un module à une trame IRM, contenant les informations demandées.
- Acp IM (Message d'acquittement)** : Le message d'acquittement est une réponse envoyée par un dispositif CAN pour confirmer qu'une requête ou une commande a été reçue

4.4. Identification des différents modules CAN :

Registre MCP25025 Et fonction	- SIDH (en bin)	- SIDL (en bin)	SIDH (Hex)	SIDL (Hex)	Identificateur (Hex) (! sur 29 bits)	Labels définis dans fichier CAN_VMD.h
-------------------------------------	--------------------	--------------------	---------------	---------------	---	---

Nœud "Commodo feux"

RXF0 -> IRM et OM	001 0 10 00	001 - 1-xx	28	28	0504 xx xx	T_Ident_IRM_Commodo_Feux
RXF1 -> IM	001 0 10 00	010 - 1-xx	28	48	0508 xx xx	T_Ident_IM_Commodo_Feux
TXD0 -> On Bus	001 0 10 00	100 - 1-xx	28	88	0510 xx xx	
TXD1 -> Acq IM	001 0 10 01	000 - 1-xx	29	08	0520 xx xx	T_Ident_AIM_Commodo_Feux
TXD2 -> Mes. Auto.	001 0 10 10	000 - 1-xx	2A	08	0540 xx xx	

Nœud "Feux avant gauche"

RXF0 -> IRM et OM	0111 00 00	001 - 1-xx	70	28	0E04 xx xx	T_Ident_IRM_FVG
RXF1 -> IM	0111 00 00	010 - 1-xx	70	48	0E08 xx xx	T_Ident_IM_FVG
TXD0 -> On Bus	0111 00 00	100 - 1-xx	70	88	0E10 xx xx	
TXD1 -> Acq IM	0111 00 01	000 - 1-xx	71	08	0E20 xx xx	T_Ident_AIM_FVG
TXD2 -> Mes. Auto.	0111 00 10	000 - 1-xx	72	08	0E40 xx xx	

Nœud "Feux avant droit"

RXF0 -> IRM et OM	0111 01 00	001 - 1-xx	74	28	0E84 xx xx	T_Ident_IRM_FVD
RXF1 -> IM	0111 01 00	010 - 1-xx	74	48	0E88 xx xx	T_Ident_IM_FVD
TXD0 -> On Bus	0111 01 00	100 - 1-xx	74	88	0E90 xx xx	
TXD1 -> Acq IM	0111 01 01	000 - 1-xx	75	08	0EA0 xx xx	T_Ident_AIM_FVD
TXD2 -> Mes. Auto.	0111 01 10	000 - 1-xx	76	08	0EC0 xx xx	

Nœud "Feux arrière gauche"

RXF0 -> IRM et OM	0111 10 00	001 - 1-xx	78	28	0F04 xx xx	T_Ident_IRM_FRG
RXF1 -> IM	0111 10 00	010 - 1-xx	78	48	0F08 xx xx	T_Ident_IM_FRG
TXD0 -> On Bus	0111 10 00	100 - 1-xx	78	88	0F10 xx xx	
TXD1 -> Acq IM	0111 10 01	000 - 1-xx	79	08	0F20 xx xx	T_Ident_AIM_FRG
TXD2 -> Mes. Auto.	0111 10 10	000 - 1-xx	7A	08	0F40 xx xx	

Identifiants CAN uniques pour chaque module :

- Chaque module (ex: feux avant gauche, feux avant droit) possède un **identifiant unique** pour chaque type de message (IM, IRM, OM.etc).
- Ces identifiants sont définis dans le tableau sous la colonne "Identificateur (Hex)".

Relations entre les registres et les messages :

- Les registres comme RXF0, RXF1, TXD0, TXD1, etc., sont associés aux différents types de messages (IM, IRM, OM).
- Par exemple, pour le "Feux avant droit" :
 - RXF0 correspond à un message IRM.
 - TXD0 correspond à un message IM (utilisé pour envoyer des ordres).
 - TXD1 correspond à un message Acp IM.

Le tableau suivant détaille comment compléter les valeurs des identifiants pour former les trames CAN en fonction du type de message (IM, IRM, OM) et de l'action (écriture ou lecture).

TABLE 4-3: COMMAND MESSAGES (EXTENDED IDENTIFIER)

Information Request Messages (to MCP2502X/5X)																											
	Standard ID				Extended ID				Data Bytes																		
0	1	9	8	7	6	5	4	3	2	1	0	R	I	DLC	1	1	RXBElD8 (8 bits)	RXBElD0 (8 bits)									
Read A/D Regs	x	x	x	x	x	x	x	x	x	x	1	0	0	0	8*	x x	xxxx xxxx	xxxx *000	n/a	n/a	n/a	n/a	n/a	n/a	n/a		
Read Control Regs	x	x	x	x	x	x	x	x	x	x	1	0	1	1	7*	x x	xxxx xxxx	xxxx *001	n/a	n/a	n/a	n/a	n/a	n/a	n/a		
Read Config Regs	x	x	x	x	x	x	x	x	x	x	1	0	1	1	5*	x x	xxxx xxxx	xxxx *010	n/a	n/a	n/a	n/a	n/a	n/a	n/a		
Read CAN Error	x	x	x	x	x	x	x	x	x	x	1	0	0	1	3*	x x	xxxx xxxx	xxxx *011	n/a	n/a	n/a	n/a	n/a	n/a	n/a		
Read PWM Config	x	x	x	x	x	x	x	x	x	x	1	0	1	0	6*	x x	xxxx xxxx	xxxx *100	n/a	n/a	n/a	n/a	n/a	n/a	n/a		
Read User Mem	x	x	x	x	x	x	x	x	x	x	1	1	0	0	8*	x x	xxxx xxxx	xxxx *101	n/a	n/a	n/a	n/a	n/a	n/a	n/a		
Read User Mem (bank)	x	x	x	x	x	x	x	x	x	x	1	1	0	0	8*	x x	xxxx xxxx	xxxx *110	n/a	n/a	n/a	n/a	n/a	n/a	n/a		
Read Register	x	x	x	x	x	x	x	x	x	x	1	0	0	0	1*	0	addr	xxxx *111	n/a	n/a	n/a	n/a	n/a	n/a	n/a		
Output Messages (from MCP2502X/5X)																											
	Standard ID				Extended ID				Data Bytes																		
0	1	9	8	7	6	5	4	3	2	1	0	R	I	DLC	1	1	RXBElD8 (8 bits)	RXBElD0 (8 bits)									
Read A/D Regs	x	x	x	x	x	x	x	x	x	x	0	1	0	0	0	8	x x	xxxx xxxx	xxxx *000	IOINTFL	GPIO	AN0H	AN1H	AN10L	AN2H	AN3H	AN23L
Read Control Regs	x	x	x	x	x	x	x	x	x	x	0	1	1	1	7	x x	xxxx xxxx	xxxx *001	ADCON0	ADCON1	OPTREG1	OPTREG2	STCON	IOINTEN	IOINTPO	n/a	
Read Config Regs	x	x	x	x	x	x	x	x	x	x	0	1	0	1	5	x x	xxxx xxxx	xxxx *010	DDR	GPIO	CNF1	CNF2	CNF3	n/a	n/a	n/a	
Read CAN Error	x	x	x	x	x	x	x	x	x	x	0	1	1	1	3	x x	xxxx xxxx	xxxx *011	EFLG	TEC	REC	n/a	n/a	n/a	n/a		
Read PWM Config	x	x	x	x	x	x	x	x	x	x	0	1	1	0	6	x x	xxxx xxxx	xxxx *100	PR1	PR2	T1CON	T2CON	PWM1DCH	PWM2DCH	n/a	n/a	
Read User Mem (bank1)	x	x	x	x	x	x	x	x	x	x	0	1	0	0	8	x x	xxxx xxxx	xxxx *101	USERID0	USERID1	USERID2	USERID3	USERID4	USERID5	USERID6	USERID7	
Read User Mem (bank)	x	x	x	x	x	x	x	x	x	x	0	1	0	0	8	x x	xxxx xxxx	xxxx *110	USERID8	USERID9	USERID10	USERID11	USERID12	USERID13	USERID14	USERID15	
Read Register	x	x	x	x	x	x	x	x	x	x	0	1	0	0	1	1	x x	addr	xxxx *111	value	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Input Messages (to MCP2502X/5X)																											
	Standard ID				Extended ID				Data Bytes																		
0	1	9	8	7	6	5	4	3	2	1	0	R	I	DLC	1	1	RXBElD8 (8 bits)	RXBElD0 (8 bits)									
Write Register	x	x	x	x	x	x	x	x	x	x	0	1	0	0	1	3	x x	xxxx xxxx	xxxx x000	addr	mask	value	n/a	n/a	n/a	n/a	
Write TX Message ID 0	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x x	xxxx xxxx	xxxx x001	TX0SIDH	TX0IDL	TX0EID8	TX0EID0	n/a	n/a	n/a	n/a	
Write TX Message ID 1	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x x	xxxx xxxx	xxxx x010	TX1SIDH	TX1IDL	TX1EID8	TX1EID0	n/a	n/a	n/a	n/a	
Write TX Message ID 2	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x x	xxxx xxxx	xxxx x011	TX2SIDH	TX2IDL	TX2EID8	TX2EID0	n/a	n/a	n/a	n/a	
Write I/O Configuration	x	x	x	x	x	x	x	x	x	x	0	1	0	1	5	x x	xxxx xxxx	xxxx x100	IOINTEN	IOINTPO	DDR	OPTREG1	ADCON1	n/a	n/a	n/a	
Write RX Mask	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x x	xxxx xxxx	xxxx x101	RXMSIDH	RXMSIDL	RXMEID8	RXMEID0	n/a	n/a	n/a	n/a	
Write RX Filter0	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x x	xxxx xxxx	xxxx x110	RXF0SIDH	RXF0IDL	RXF0EID8	RXF0EID0	n/a	n/a	n/a	n/a	
Write RX Filter1	x	x	x	x	x	x	x	x	x	x	0	1	0	0	4	x x	xxxx xxxx	xxxx x111	RXF1SIDH	RXF1IDL	RXF1EID8	RXF1EID0	n/a	n/a	n/a	n/a	

Identifiant CAN :

- L'identifiant est constitué de plusieurs parties, dont certaines sont fixes et d'autres variables.
- Par exemple, pour un modèle **feux arrière droit**, pour un message IM (Input Message) de type "Write Register", l'identifiant de base est :
 - **0xF88** suivi de **xx xx**, où xx xx est une valeur dynamique déterminée par le tableau.

Complément des identifiants (xx xx) :

- En fonction du type de message et de l'action, on remplit les **xx xx** avec les valeurs spécifiques décrites dans le tableau.
- Par exemples :

Input Messages (to MCP2502X/5X)																										
	Standard ID				Extended ID				Data Bytes																	
0	1	9	8	7	6	5	4	3	2	1	0	R	I	DLC	1	1	RXBElD8 (8 bits)	RXBElD0 (8 bits)								
Write Register	x	x	x	x	x	x	x	x	x	x	0	1	0	0	1	3	x x	xxxx xxxx	xxxx x000	addr	mask	value	n/a	n/a	n/a	n/a

- **1** : Pour "Write Register" dans IM dans le champ **Extended ID**: xxxx xxxx xxxx x000. Cela donne :

- Complément des xx xx = 0x00 00
- Identifiant final = 0x F 88 00 00.

Ou on a que des x les en remplace par des 0.

Et sur le champ Data en voit qu'on a trois donnée à envoyées adresse de registre cible (par exemple GPDDR ou GPLAT), masque et une valeur.

- **2** : Pour "Read Register" dans IRM dans le champ **Extended ID** : Addr xxxx x111.
 - Complément des xx xx = 0xAA 07 (AA adresse de registre utilisé "**c'est une adresse par défaut**")
 - Identifiant final = 0x F 84 AA 07.

4.5. Gestion des données avec le masque :

Lorsque la valeur d'un bit de **masque** est à **0**, nous ignorons la donnée correspondante dans T_IM_Feux.data[2]. En d'autres termes :

- Si un bit de **masque** vaut **0**, la donnée associée dans T_IM_Feux.data[2] n'est **pas prise en compte**.
- Si un bit de **masque** vaut **1**, la donnée associée est **appliquée**.

4.6. Exemple de trame pour le TP1 de VMD :

Cette trame configure les broches en mode **sortie (feux arrière droit)**.

```
T_IM_Feux.trame_info.registre=0x00;
T_IM_Feux.trame_info.champ.extend=1; // On travaille en mode étendu
T_IM_Feux.trame_info.champ.dlc=0x03; // Il y aura 3 données de 8 bits (3 octets envoyés)
T_IM_Feux.ident.extend.identificateur.ident=0xF880000; // C'est l'identificateur du bloc optique arrière droit
T_IM_Feux.data[0]=0x1F; // première donnée -> "Adresse" du registre concernée (GPDDR donne la direction des I/O)
T_IM_Feux.data[1]=0x7F; // deuxième donnée -> "Masque" -> Les sorties sont sur les 4 bits de poids faibles
T_IM_Feux.data[2]=0xF0; // troisième donnée -> "Valeur" -> Les sorties sont sur les 4 bits lsb
```

Le nom de la trame : **T_IM_Feux**

- **T_IM_Feux.trame_info.registre=0x00** : initialise tous les bits de registre à zéro.
- **T_IM_Feux.trame_info.champ.extend=1** : Mode étendu activé.
- **T_IM_Feux.trame_info.champ.dlc=0x03** : Envoi de 3 données (car il s'agit d'un message **write register** en mode IM).
- **T_IM_Feux.ident.extend.identificateur.ident=0xF880000** : Identificateur spécifique au modèle arrière droit pour un message IM.

Contenu du champ champ data :

Addr*	Name	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Value on POR	Value on RST
1Fh**	GPDDR	—	DDR6	DDR5	DDR4	DDR3	DDR2	DDR1	DDR0	-111 1111	-111 1111
18h	EFLG	ESCF	RBO	TXBO	TXEP	RXEP	TXWAR	RXWAR	EWARN	0000 0000	0000 0000
19h	TEC	Transmit Error Counters								0000 0000	0000 0000
1Ah	REC	Receive Error Counters								0000 0000	0000 0000
50h	ADRES3H	AN3.9	AN3.8	AN3.7	AN3.6	AN3.5	AN3.4	AN3.3	AN3.2	xxxx xxxx	uuuu uuuu
51h	ADRES3L	AN3.1	AN3.0	—	—	—	—	—	—	xx-- ----	uu-- ----
52h	ADRES2H	AN2.9	AN2.8	AN2.7	AN2.6	AN2.5	AN2.4	AN2.3	AN2.2	xxxx xxxx	uuuu uuuu
53h	ADRES2L	AN2.1	AN2.0	—	—	—	—	—	—	xx-- ----	uu-- ----
54h	ADRES1H	AN1.9	AN1.8	AN1.7	AN1.6	AN1.5	AN1.4	AN1.3	AN1.2	xxxx xxxx	uuuu uuuu
55h	ADRES1L	AN1.1	AN1.0	—	—	—	—	—	—	xx-- ----	uu-- ----
56h	ADRES0H	AN0.9	AN0.8	AN0.7	AN0.6	AN0.5	AN0.4	AN0.3	AN0.2	xxxx xxxx	uuuu uuuu
57h	ADRES0L	AN0.1	AN0.0	—	—	—	—	—	—	xx-- ----	uu-- ----

T_IM_Feux.data[0]=0x1F : l'adresse de registre PGDDR (voir tableau pour détails).

T_IM_Feux.data[1]=0x7F : Masque appliqué : seuls les 7 bits de poids faibles sont modifiés (le bit 8 reste inchangé car il est en mode entrée et non reprogrammable).

6.2.2 Carte 4 sorties TOR

Le port 8 bit du can expander MCP25050 est configurer :

GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
E	E	E	E	S	S	S	S

Avec : E: entrée TOR,
S : sortie TOR

6.2.2.1 Feux avant

L'affectation des entrées sur la carte entrée est la suivante :

GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
Etat clignotant	Etat phare	Etat code	Etat veilleuse	clignotant	phare	code	Veilleuse

6.2.2.2 Feux arrières

L'affectation des entrées sur la carte entrée est la suivante :

GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
Etat GP3	Etat clignotant	Etat code	Etat veilleuse	(klaxon)	clignotant	code	Veilleuse

Remarques : la commande klaxon est active sur le module feux arrières gauche.

La photo montre que les **sorties** se trouvent sur les **4 bits LSB (GP0 à GP3)**. Pour configurer un bit en **mode sortie** ou en **mode entrée** :

- **Mode sortie** : mettre le bit à 0.
- **Mode entrée** : mettre le bit à 1.

T_IM_Feuux.data[2]=0xF0 : Les bits **GP7 à GP4** sont configurés en **entrée**.

Les bits **GP3 à GP0** sont configurés en **sortie**.

Remarque : Les bits GP7 à GP4 du circuit MCP25050 sont déjà programmés en mode entrée et ne peuvent pas être reprogrammés, car le circuit n'est pas reprogrammable.

En conséquence, de notre travail sur le projet nous ne sommes pas intéressés par un masque comme 0x7F, qui inclut des bits inutilisables (**GP7 à GP4**). À la place, nous pouvons utiliser un masque 0x0F, qui cible uniquement les bits **GP3 à GP0**, configurables en mode sortie.

Exemple d'utilisation d'un masque :

Avec T_IM_Feuux.data[2] = 0xF0 et un masque 0x0F :

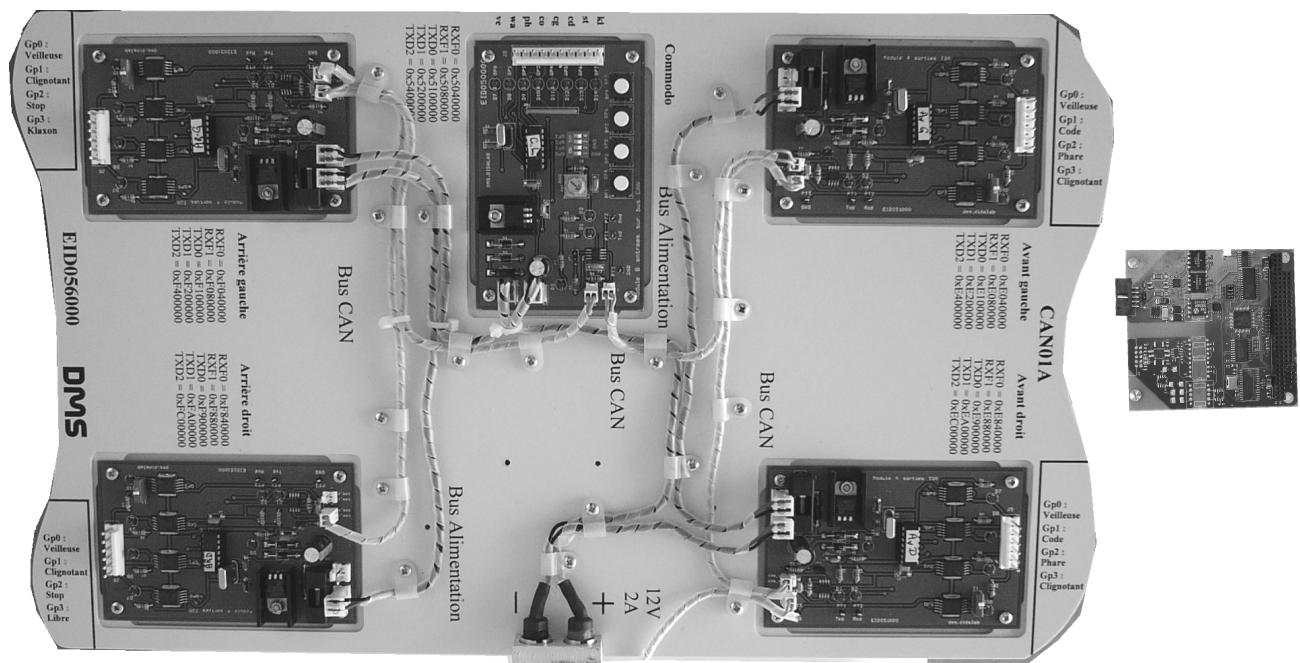
- **Masque : 0x0F** :
 - Les bits **GP3 à GP0** sont pris en compte.
 - Les bits **GP7 à GP4** sont ignorés.
- **Donnée : T_IM_Feuux.data[2] = 0xF0** :
 - Les bits **GP7 à GP4** (qui valent 1) sont ignorés grâce au masque.
 - Les bits **GP3 à GP0** sont configurés comme suit :
 - GP3 : 0 (sortie).
 - GP2 : 0 (sortie).
 - GP1 : 0 (sortie).
 - GP0 : 0 (sortie).

1 PRESENTATION

Le VMD (Véhicule Multiplexé Didactique) est un système didactique sur les RLI (Réseaux locaux industriels).

Le VMD utilise le bus CAN pour communiquer avec ses différents modules d'entrées et de sorties. Il a été développé en s'appuyant sur ce qui existe dans l'automobile. Nous avons reconstitué le bus signalisation d'un véhicule.

Le module CAN01A est un sous-système du VMD avec uniquement les cartes CAN du bus signalisation et la carte contrôleur CAN ATON_CAN sur bus PC104.



Dans le bus CAN, il n'y a que 2 couches normalisées dans le modèle OSI :

Couche physique (couche 1 OSI)

Couche liaison de donnée (couche 2 OSI).

Les autres couches ne font pas l'objet de normalisation dans le bus CAN.

Le VMD se compose :

D'un support thermoformé représentant un véhicule avec ses organes de signalisation,

D'une carte processeur EID210, représentant l'ordinateur de bord,

D'une carte 8 entrées TOR sur bus CAN gérant le commodo lumière,

De 4 cartes de sortie TOR, gérant :

Les feux avant gauche,

Les feux avant droit,

Les feux arrière gauche,

Les feux arrière droit ;

D'une carte clavier afficheur réalisant le tableau de bord.

La carte EID210 accède au bus CAN par l'intermédiaire d'une carte PC104 CAN de ATON-SYSTEME à travers un contrôleur CAN SJA1000 de philips.

Les cartes CAN possèdent un contrôleur CAN MCP25050 de microchip

3 ORGANISATION LOGICIEL

3.1 Fichier de définition

Pour utiliser le bus can, il faut utiliser le fichier de définition « aton_can.h ». Celui-ci définit les structures de données permettant la gestion des trame CAN.

L'union "ident_standard" permet de définir la partie identification en mode standard

```
typedef union
{
    struct
    {
        // Eléments constitutif de l'identificateur dans une trame
        unsigned short ident:11; // les 11 bits d'identification en mode standard
        unsigned short nul:5; // 5 bits inutilisés
    } identificateur;
    struct
    {
        // Les mêmes éléments mais dans les registres du circuit SJA1000
        unsigned char ident1; // premier registre 8 bits de définition de l'identificateur
        unsigned char ident2; // deuxième registre 8 bits de définition de l'identificateur
    } registre;
    unsigned short valeur; // La taille globale est de 16 bits
} ident_standard;
```

L'union "ident_extend" permet de définir la partie identification en mode étendu

```
typedef union
{
    struct
    {
        // Eléments constitutif de l'identificateur dans une trame
        unsigned long ident:29; // les 29 bits d'identification en mode étendu
        unsigned long x:3; // 3 bits inutilisés
    } identificateur;
    struct
    {
        // Les mêmes éléments mais dans les registres du circuit SJA1000
        unsigned char ident1; // premier registre 8 bits de définition de l'identificateur
        unsigned char ident2; // deuxième registre 8 bits de définition de l'identificateur
        unsigned char ident3; // troisième registre 8 bits de définition de l'identificateur
        unsigned char ident4; // quatrième registre 8 bits de définition de l'identificateur
    } registre;
    unsigned long valeur; // La taille globale est de 32 bits
} ident_extend;
```

La structure "Trame" permet de définir une trame complète

```
typedef struct
{
    tr_info trame_info; // Taille 8 bits
    union
    {
        ident_standard standard; // Identificateur en mode standard (2*8bits LSB)
        ident_extend extend; // Identificateur en mode étendu (4*8bits)
    } ident;
    unsigned char data[8]; // les 8 octets de données (au maximum)
} Trame; // Taille globale: 13 octets
```

3.2 Bibliothèque de fonctions

Pour gérer le VMD, la bibliothèque « aton_can.o » permet la gestion du bus can. Elle est composée des fonctions suivantes :

Void Init_Aton_Can()

Initialise le contrôleur de bus CAN SJA1000 pour la gestion du VMD :

Configure la vitesse à 100 Kbits/S,

Met en place les FIFOs d'émission et de réception en interruption (autovecteur 29)

Void Stop_Aton_Can()

Stop le module de gestion du bus can (libère le vecteur d'interruption)

void Ecrire_Trame(Trame trame)

Envoi une trame sur le bus CAN.

Char Lire_Trame(Trame *trame)

Permet de lire une trame reçue par le contrôleur CAN.

La fonction renvoi 1 si le contrôleur a reçu une trame.

Void Affiche_Trame(Trame trame)

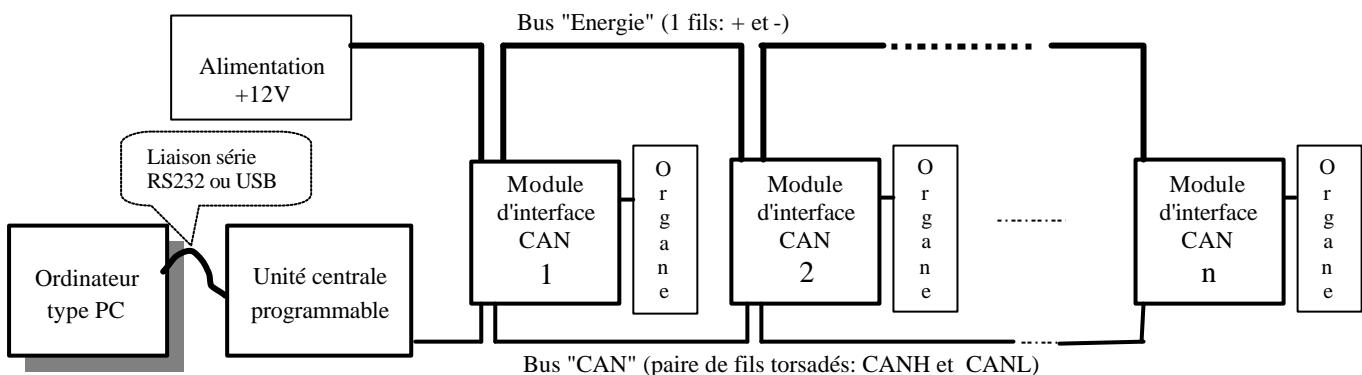
Affiche sur le port terminal la trame CAN avec son identifiant et ses données éventuelles.

4 ANALYSE FONCTIONNELLE DESCENDANTE

4.1 Schéma organisationnel d'ensemble

Le système comporte les éléments suivants :

- un ensemble "unité centrale" programmable en liaison avec un ordinateur de type PC,
- une source d'énergie (batterie 12V ou alimentation non autonome générant du 12V sous 20 A,
- un certain nombre d'ensembles, chacun pouvant être constitué :
 - * d'un module d'interface CAN configurable,
 - * d'un organe de type capteur, pré-actionneur voir actionneur, compatible avec le module associé,
- câbles de liaison.



Remarques :

- Des éléments de simulation d'entrées/sorties (LEDS, boutons poussoirs, commutateurs) ont été intégrés sur les modules et permettent éventuellement de s'affranchir des organes réels.
- Dans sa version "VMD" (Véhicule Multiplexé Didactique) les organes reliés au modules d'interface CAN peuvent être un commodo, un bloc optique avant, un bloc optique arrière, un moteur d'essuie glace ou delève vitre, plafonnier ... etc.

4.2 L'unité centrale programmable

Elle comprend un certain nombre de cartes électroniques reliées entre elles par BUS parallèles :

- cartes indispensables

- Carte processeur EID210 conçue autour du microprocesseur 32 bits Motorola 68332 (carte spécifique DMS),
- Carte industrielle d'interface parallèle / CAN conçue autour du circuit SJA1000, au format normalisé "PC104" (carte produite par la société ATON SYSTEMES),

- cartes optionnelles

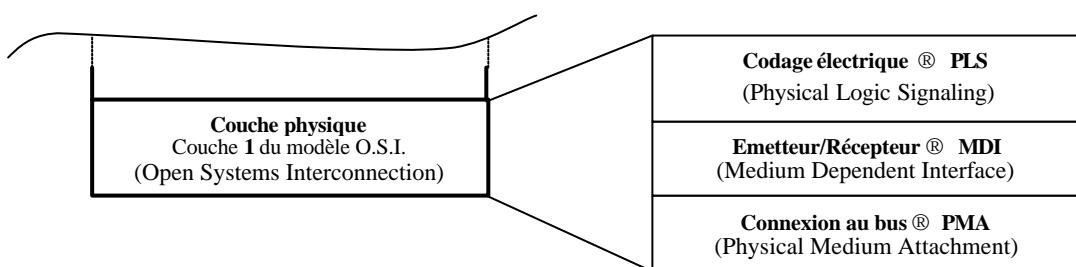
- Carte clavier 16 touches matricé et afficheur graphique (carte spécifique DMS),
- Carte d'interface réseau "Ethernet" (carte spécifique DMS),
- Toute carte au format PC1014.

4.3 Les modules d'interface CAN configurables

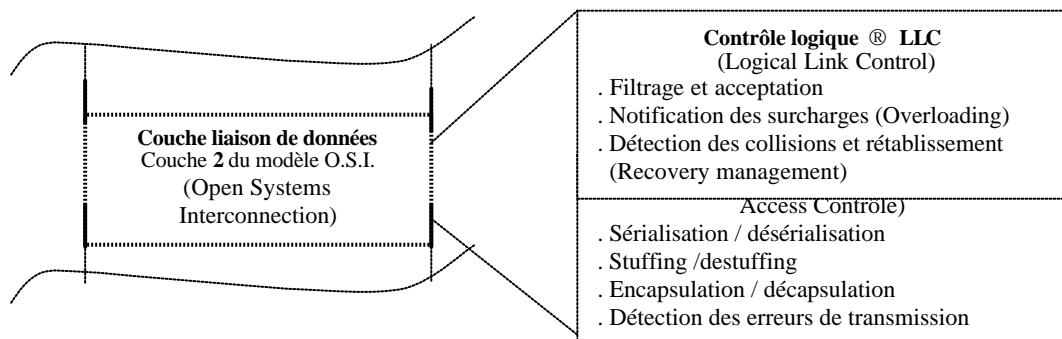
4.3.1 Les fonctions communes

Tous ces modules comportent les éléments suivants :

- deux connecteurs de liaison au médium (fils de liaisons bus CAN) permettant de connecter les modules en série,
- deux connecteurs d'alimentation (bus "alimentation énergie"),
- un régulateur de tension générant la tension +5V nécessaire aux circuits intégrés inclus sur le module ainsi que les protections d'usage et une LED de visualisation de présence tension,
- une résistance de terminaison de bus connectable ou non par "jumper",
- un circuit intégré "émetteur récepteur de ligne" (réf:82CA251) réalisant l'interface électrique (mode différentiel côté bus CAN et mode référencé côté application), c'est à dire la couche 1 du modèle O.S.I.



- un circuit intégré d'extension d'entrées sorties (Réf:MPC25050) permettant la communication et l'interprétation des messages, c'est à dire la couche 2 du modèle O.S.I.



- un oscillateur à quartz nécessaire au circuit MPC25050,
- une LED de visualisation de communication en réception,
- une LED de visualisation de communication en émission.

4.3.2 Module 8 entrées logiques

Le schéma de principe complet de ce module est donné en annexe.

Ce module permet d'interfacer 8 entrées/sorties logiques.

En plus des éléments communs, ce module comprend :

- 4 boutons poussoir,
- 4 commutateurs à 2 positions (fermé ou ouvert),
- 1 connecteur 10 points permettant de relier les capteurs externes de type fin de course,
- 8 LEDs de visualisation des états,
- 1 potentiomètre analogique.

Remarques :

- Si un capteur externe de type fin de course est relié au connecteur, le commutateur (ou bouton poussoir) correspondant doit rester en position "ouvert" (sinon il courtcircuite le fin de course).
 - Une LED s'allume si l'entrée correspondante est forcée à 0 (commutateur en position "fermé" ou bouton poussoir "appuyé").
 - Ce module peut être utilisé comme module comportant des sorties logiques compatible "TTL". En effet le circuit MPC25050 est configurable. On peut donc, via le bus CAN, lui envoyer une trame qui définira si telle ou telle liaison est une entrée ou une sortie. On pourra donc envisager toute combinaison pourvu que la somme des entrées et des sorties ne dépasse pas 8.
- Dans le cas d'une liaison configurée en sortie, il est impératif que le commutateur correspondant soit à l'état ouvert.**
- Le potentiomètre n'est actif que si le commutateur "0" est fermé.

4.3.3 Module 4 sorties de puissance

Le schéma de principe de ce module est donné en annexe.

Ce module comporte:

- 4 interfaces de puissance permettant de piloter 4 charges électriques en "tout ou rien", sous 12V
- 4 Leds de visualisation des états des sorties puissance,
- 4 entrées de contrôle des charges,
- 1 entrée de simulation de coupure de charge.

Remarques :

- Le circuit intégré réalisant l'interface de puissance (Réf: VN05) génère un signal logique indiquant si une charge est connectée (contrôle du courant absorbé). Ces signaux logiques sont considérés comme entrée du système et permettent, dans le cas du VMD et pour la commande d'une ampoule, de contrôler le bon état de fonctionnement de celle-ci.
- Les sorties de puissance sont de type source, c'est à dire que les 4 charges auront pour point commun, la référence de potentiel (dans le cas d'un véhicule, le "-" de la batterie relié à la carcasse).
- Les circuits de puissance "VN05" peuvent accepter des charges électriques consommant jusqu'à 12 A en continu. Ils sont protégés contre les court-circuits ainsi que contre les dépassages de température.

5 IDENTIFICATION DES DIFFERENTS MODULES CAN

Registre MCP25025 Et fonction	- SIDH (en bin)	- SIDL (en bin)	SIDH (Hex)	SIDL (Hex)	Identificateur (Hex) (! sur 29 bits)	Labels définis dans fichier CAN_VMD.h
-------------------------------------	--------------------	--------------------	---------------	---------------	---	---

Noeud "Commodo feux"

RXF0-> IRM et OM	001 0 10 00	001 - 1-xx	28	28	0504 xx xx	T_Ident_IRM_Commodo_Feux
RXF1-> IM	001 0 10 00	010 - 1-xx	28	48	0508 xx xx	T_Ident_IM_Commodo_Feux
TXD0-> On Bus	001 0 10 00	100 - 1-xx	28	88	0510 xx xx	
TXD1-> Acq IM	001 0 10 01	000 - 1-xx	29	08	0520 xx xx	T_Ident_AIM_Commodo_Feux
TXD2-> Mes. Auto.	001 0 10 10	000 - 1-xx	2A	08	0540 xx xx	

Noeud "Feux avant gauche"

RXF0-> IRM et OM	011 1 00 00	001 - 1-xx	70	28	0E04 xx xx	T_Ident_IRM_FVG
RXF1-> IM	011 1 00 00	010 - 1-xx	70	48	0E08 xx xx	T_Ident_IM_FVG
TXD0-> On Bus	011 1 00 00	100 - 1-xx	70	88	0E10 xx xx	
TXD1-> Acq IM	011 1 00 01	000 - 1-xx	71	08	0E20 xx xx	T_Ident_AIM_FVG
TXD2-> Mes. Auto.	011 1 00 10	000 - 1-xx	72	08	0E40 xx xx	

Noeud "Feux avant droit"

RXF0-> IRM et OM	011 1 01 00	001 - 1-xx	74	28	0E84 xx xx	T_Ident_IRM_FVD
RXF1-> IM	011 1 01 00	010 - 1-xx	74	48	0E88 xx xx	T_Ident_IM_FVD
TXD0-> On Bus	011 1 01 00	100 - 1-xx	74	88	0E90 xx xx	
TXD1-> Acq IM	011 1 01 01	000 - 1-xx	75	08	0EA0 xx xx	T_Ident_AIM_FVD
TXD2-> Mes. Auto.	011 1 01 10	000 - 1-xx	76	08	0EC0 xx xx	

Noeud "Feux arrière gauche"

RXF0-> IRM et OM	011 1 10 00	001 - 1-xx	78	28	0F04 xx xx	T_Ident_IRM_FRG
RXF1-> IM	011 1 10 00	010 - 1-xx	78	48	0F08 xx xx	T_Ident_IM_FRG
TXD0-> On Bus	011 1 10 00	100 - 1-xx	78	88	0F10 xx xx	
TXD1-> Acq IM	011 1 10 01	000 - 1-xx	79	08	0F20 xx xx	T_Ident_AIM_FRG
TXD2-> Mes. Auto.	011 1 10 10	000 - 1-xx	7A	08	0F40 xx xx	

Noeud "Feux arrière droit"

RXF0-> IRM et OM	011 1 11 00	001 - 1-xx	7C	28	0F84 xx xx	T_Ident_IRM_FRD
RXF1-> IM	011 1 11 00	010 - 1-xx	7C	48	0F88 xx xx	T_Ident_IM_FRD
TXD0-> On Bus	011 1 11 00	100 - 1-xx	7C	88	0F90 xx xx	
TXD1-> Acq IM	011 1 11 01	000 - 1-xx	7B	08	0FA0 xx xx	T_Ident_AIM_FRD
TXD2-> Mes. Auto.	011 1 11 10	000 - 1-xx	7E	08	0FC0 xx xx	

6 DESCRIPTION DU MCP25050

Pour plus de renseignement relatif au mcp25050, se reporter à la datasheet du MCP25050 de MICROCHIP.

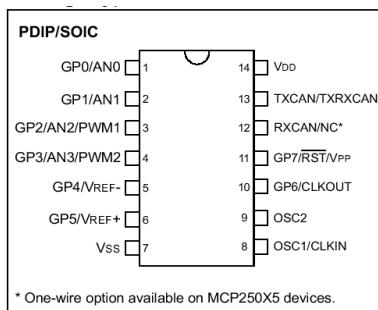
C'est un circuit intégré 14 broches disponible en version PDIP ou SOIC.

Côté interface CAN il satisfait la norme CAN V2.0B c'est à dire qu'il peut communiquer avec une vitesse de transmission qui peut atteindre 1Mbit/s. Certaines versions du circuit permettent une communication sur 1 fil (One-wire).

Côté application, il possède 8 lignes d'entrées/sorties (GP0 à GP7) configurables individuellement en entrée ou en sortie. Seule la ligne GP7 ne peut être utilisée en sortie.

Si on le souhaite, il est capable d'envoyer un message sans qu'il soit interrogé si l'une de ses entrées change d'état.

Deux liaisons (GP2 et GP3) peuvent être configurées en sorties modulées (PWM). Ces deux sorties peuvent



Device	A/D	One-wire CAN
MCP25020	No	No
MCP25025	No	Yes
MCP25050	Yes	No
MCP25055	Yes	Yes

être commandées indépendamment l'une de l'autre, fréquences et rapports cycliques sur 10 bits.

Certaines versions du circuit intègrent un convertisseur analogique → numérique (4 voies) sur 10 bits. Si on le souhaite, il est capable d'envoyer un message sans qu'il soit interrogé si l'une de ses entrées analogique dépasse des seuils de tension que l'on peut choisir.

Il possède un "schéduleur" qui lui permet d'envoyer un message à intervalles de temps réguliers sans qu'on le lui demande (par exemple l'état de ses entrées ou la valeur convertie d'une des entrées analogiques).

C'est un circuit configurable grâce à une banque de registres qui sont gravés dans le circuit lors d'une phase de programmation.

Remarque :

- Ces circuits ne sont pas reprogrammables.
- Le constructeur (MICROCHIP) commercialise des outils logiciels et matériel de configuration et de programmation.
- Un circuit déjà programmé peut être lu par le logiciel de configuration et de programmation.

6.1 La configuration de la vitesse de transmission

Cette vitesse dépend de la fréquence du signal d'horloge interne (imposée par le quartz) notée t_{osc} .
 Cette fréquence est divisée (passage dans un « perscaler ») pour obtenir la période t_Q (Time Quantum).
 Le coefficient de division BRP est choisi grâce à un mot de 6 bits $BRP_5 \dots BRP_0$ (Ces bits font partie du registre de configuration n°1 CNF1 du circuit).

On obtient la valeur de t_Q grâce à l'expression :

$$t_Q = 2^* \text{tosc}^*(\text{BRP}+1)$$

La durée de transmission d'un bit (Bit Time) est fonction de 3 paramètres :

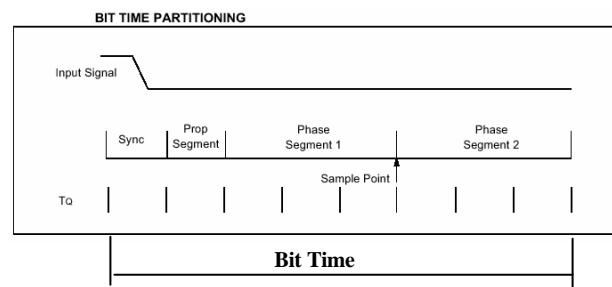
- un paramètre de synchronisation « SYN_Seg »
-> sur deux bits (SJW1 et SJW0) inclus dans le registre de configuration n°1
 - un paramètre de propagation « Prop Segment»
-> sur 3 bits PRSEG2, PRSEG1 et PRSEG0 dans le registre de configuration n° 2
 - un premier paramètre de phase «Phase Segment 1 »
-> sur 3 bits PHSEG12, PHSEG11 et PHSEG10 dans le registre de configuration n° 2
 - un deuxième paramètre de phase «Phase Segment »
-> sur 3 bits PHSEG22, PHSEG21 et PHSEG20 dans le registre de configuration n° 2

Les paramètres de phase permettent de définir l'instant d'échantillonnage (« Sample time), instant où l'on prend la valeur logique du bus ce qui va donner l'état du bit.

On obtient la valeur de t_{BIT} (durée de transmission d'un bit) grâce à l'expression :

$$t_{BIT} = t_Q * (S_Seg + P_Seg + PH_Seg1 + PH_Seg2)$$

La vitesse de transmission est l'inverse de t_{BIT}



La répartition des paramètres dans les registres est donnée ci-contre.

(Voir « DATA SHEET » du circuit MCP25050 page 8 et 9)

Exemple de calcul de la vitesse de transmission:

La fréquence du quartz sur la carte est 16 Mhz

On a choisi $BRP = 4$

$$\rightarrow t_0 = 2 * (4+1) * t_{osc} = 10 / 16 \mu s = 0.625 \mu s$$

On a choisi :

SJW = 0 , PRSEG = 0

PHSEG1 = 8 et PHSEG2 = 8

D'après « DATA SHEET »

En définitive :

$$t_{\text{d}} = 16 * t_{\text{c}} = 16 * 0.625 \mu\text{s} = 10 \mu\text{s}$$

Soit une vitesse de $1/t_{\text{v}} = 100$ Kbit/s

CNE1 - CAN CONFIGURATION REGISTER 1

| R/W-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| SJW1 | SJW0 | BRP5 | BRP4 | BRP3 | BRP2 | BRP1 | BRP0 |
| bit7 | | | | | | | bit0 |

CNF2 - CAN CONFIGURATION REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BTLMODE	SAM	PHSEG12	PHSEG11	PHSEG10	PRSEG2	PRSEG1	PRSEG0
bit7							bit0

CNF3 - CAN CONFIGURATION REGISTER 3

The diagram illustrates a memory map with several regions. The region containing **WAKFIL** is located at address **U-0**, which is highlighted in yellow. This region spans from **U-0** to **U-1**. Other regions shown include **PHSEG22** at **U-0** to **U-1**, **PHSEG21** at **U-0** to **U-1**, and **PHSEG20** at **U-0** to **U-1**. The bit range for **bit7** is indicated by a bracket from **U-0** to **U-1**, and the bit range for **bit0** is indicated by a bracket from **U-0** to **U-1**.

- Les sorties de puissance sont de type source, c'est à dire que les 4 charges auront pour point commun, la référence de potentiel (dans le cas d'un véhicule, le "-" de la batterie relié à la carcasse).
- Les circuits de puissance **VN05** peuvent accepter des charges électriques consommant jusqu'à 12 A en continu. Ils sont protégés contre les courts circuits ainsi que contre les dépassemens de température.

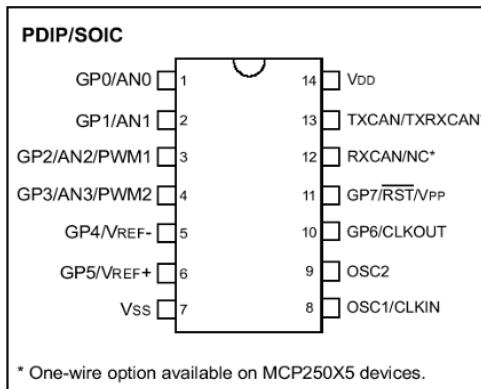
2.5 DESCRIPTION DU MCP25050 (le circuit intégré principale) :

C'est un circuit intégré 14 broches disponible en version PDIP ou SOIC.

Côté interface CAN : Peut communiquer avec une vitesse de transmission qui peut atteindre 1Mbit/s. Certaines versions du circuit permettent une communication sur 1 fil (One-wire).

Côté application : Il possède 8 lignes d'entrées/sorties (GPO à GP7) configurables individuellement en entrée ou en sortie, seule la ligne GP7 ne peut être utilisée en sortie.

Il est capable d'envoyer un message sans qu'il soit interrogé si l'une de ses entrées change d'état.



Deux liaisons (GP2 et GP3) peuvent être configurées en sorties modulées (PWM), ces deux sorties peuvent être commandées indépendamment l'une de l'autre. Certaines versions du circuit intègrent un convertisseur analogique numérique (4 voies) sur 10 bits.

Il est capable d'envoyer un message sans qu'il soit interrogé si l'une de ses entrées analogique dépasse des seuils de tension que l'on peut choisir. Il possède un "**schéduleur**" qui lui permet d'envoyer un message à intervalles de temps réguliers sans qu'on le lui demande (par exemple l'état de ses entrées ou la valeur convertie d'une des entrées analogiques).

C'est un circuit configurable grâce à une banque de registres qui sont gravés dans le circuit lors d'une phase de programmation.

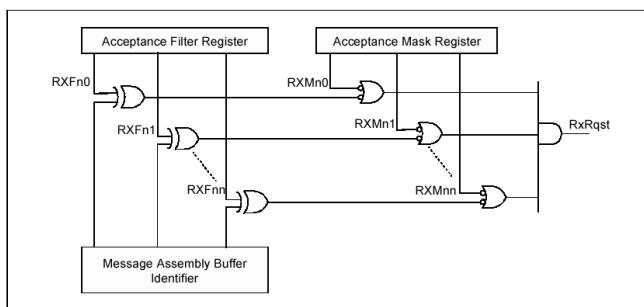
Remarque :

- Ces circuits ne sont pas reprogrammables.
- Le constructeur (MICROCHIP) commercialise des outils logiciels et matériels de configuration et de programmation.
- Un circuit déjà programmé peut être lu par le logiciel de configuration et de programmation.

La configuration des masques et des filtres "d'acceptance"

Un message circulant sur le bus ne sera pris en considération par le circuit que si l'identificateur associé au message a passé avec succès les obstacles du filtre et du masque.

Cette technique peut être déduite du schéma logique donné ci-après, ainsi que de la table de vérité.



FILTER/MASK TRUTH TABLE

Mask Bit n	Filter Bit n	Message Identifier bit n001	Accept or reject bit n
0	X	X	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Note: X = don't care

En résumé, si on donne à un bit de masque l'état 1 logique un message ne sera accepté que si le bit correspondant de l'identificateur associé est du même état que celui donné au filtre.

Par contre, si on donne à un bit de masque l'état 0 logique, le bit correspondant de l'identificateur est masqué c'est à dire que son état n'est pas vérifié.

La définition des masques et des filtres s'effectue grâce à un certain nombre de registres destiné à cet effet (voir tableau ci-après).

Rappel :

Dans la version "standard" du bus CAN (version V2.0A) l'identificateur est sur 11 bits. Dans ce cas les bits de l'identificateur sont repérés **SID10 ... SID0**.

Dans la version "étendue" du bus CAN (version V2.0B) l'identificateur est sur 29 bits. Dans ce cas les bits de l'identificateur sont repérés **EID17 ... EID0** (ce sont les poids faibles de l'identificateur, les poids forts étant l'identificateur standard).

Il y a deux séries de registres dans lesquels on charge les valeurs des masques et des filtres :

→ la série d'indice 0 destinée aux trames interrogatives ("Information Request"),

→ la série d'indice 1 destinée aux entrées de données ("Input message").

Les registres définissant les masques

RXMSIDH - ACCEPTANCE FILTER MASK STANDARD IDENTIFIER HIGH

| R/W-x |
|-------|-------|-------|-------|-------|-------|-------|-------|
| SID10 | SID9 | SID8 | SID7 | SID6 | SID5 | SID4 | SID3* |
| bit7 | | | | | | | bit0 |

R = Readable bit
W = Writable bit
U = Unimplemented, read as '0'

RXMSIDL - ACCEPTANCE FILTER MASK STANDARD IDENTIFIER LOW

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDE	—	EID17	EID16
bit7							bit0

RXMEID8 - ACCEPTANCE FILTER MASK EXTENDED IDENTIFIER MID

| R/W-x |
|-------|-------|-------|-------|-------|-------|-------|-------|
| EID15 | EID14 | EID13 | EID12 | EID11 | EID10 | EID9 | EID8 |
| bit7 | | | | | | | bit0 |

RXMEID0 - ACCEPTANCE FILTER MASK EXTENDED IDENTIFIER LOW

| R/W-x |
|-------|-------|-------|-------|-------|-------|-------|-------|
| EID7 | EID6 | EID5 | EID4 | EID3 | EID2 | EID1 | EID0 |
| bit7 | | | | | | | bit0 |

Les registres définissant les filtres

RXFNSIDH - ACCEPTANCE FILTER N STANDARD IDENTIFIER HIGH

| R/W-x |
|-------|-------|-------|-------|-------|-------|-------|-------|
| SID10 | SID9 | SID8 | SID7 | SID6 | SID5 | SID4 | SID3* |
| bit7 | | | | | | | bit0 |

R = Readable bit
W = Writable bit
U = Unimplemented, read as '0'

RXFNSIDL - ACCEPTANCE FILTER N STANDARD IDENTIFIER LOW

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDE	—	EID17	EID16
bit7							bit0

RXFNEID8 - ACCEPTANCE FILTER N EXTENDED IDENTIFIER MID

| R/W-x |
|-------|-------|-------|-------|-------|-------|-------|-------|
| EID15 | EID14 | EID13 | EID12 | EID11 | EID10 | EID9 | EID8 |
| bit7 | | | | | | | bit0 |

RXFNEID0 - ACCEPTANCE FILTER N EXTENDED IDENTIFIER LOW

| R/W-x |
|-------|-------|-------|-------|-------|-------|-------|-------|
| EID7 | EID6 | EID5 | EID4 | EID3 | EID2 | EID1 | EID0 |
| bit7 | | | | | | | bit0 |

D'après "DATA SHEET" du circuit pages 12, 13 et 14

Les différents messages

Pour communiquer avec le circuit via le bus CAN, on distingue 3 types de messages :

- messages de demande d'information (Information Request Messages) → IRM qui permettent de demander au circuit de renvoyer la valeur d'un ou plusieurs de ses registres internes (par exemple pour lire l'état de ses entrées),
- messages d'entrée (Input Messages) qui permettent de modifier la valeur d'un ou plusieurs registres internes (par exemple pour modifier l'état de ses sorties),
- messages de sortie (Output messages) qui permettent de répondre à un message d'interrogation.

Interrogations sur la valeur des registres		Pour changer la valeur des registres	
Name	Description		
Read A/D Registers	Transmits a single message containing the current state of the analog and I/O registers including the configuration	Write Register	Uses a mask to write a value to an addressed register
Read Control Registers	Transmits several control registers not included in other messages	Write TX Message ID0 (TXID0)	Writes the identifiers to a specified value
Read Configuration Registers	Transmits the contents of many of the configuration registers	Write TX Message ID1 (TXID1)	Writes the identifiers to a specified value
Read CAN error states	Transmits the error flag register and the error counts	Write TX Message ID2 (TXID2)	Writes the identifiers to a specified value
Read PWM Configuration	Transmits the registers associated with the PWM modules	Write I/O Configuration Registers	Writes specified values to the three IOCON registers
Read User Registers 1	Transmits a the values in bytes 0 - 7 of the user memory	Write RX Mask	Changes the receive mask to the specified value
Read User Registers 2	Transmits a the values in bytes 8 -15 of the user memory	Write RX Filter0	Changes the specified filter to the specified value
Read Register*	Transmits a single byte containing the value in an addressed user memory register	Write RX Filter1	Changes the specified filter to the specified value

*The Read Register command is available when using extended message format only. Not available with standard message format.

Se reporter à la "DATA SHEET" du circuit (pages 21 et 22) pour connaître la constitution des différents messages.

6.2 Configuration des entrées/sorties

6.2.1 Carte 8 entrées (Commodo lumière)

Le port 8 bits du can expander MCP25050 est configurer en entrée TOR.

GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
E	E	E	E	E	E	E	E

Avec E: entrée TOR.

L'affectation des entrées sur la carte entrée est la suivante :

GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
klaxon	stop	Clignotant droit	Clignotant gauche	code	phare	warning	Veilleuse

6.2.2 Carte 4 sorties TOR

Le port 8 bit du can expander MCP25050 est configurer :

GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
E	E	E	E	S	S	S	S

Avec : E: entrée TOR,

S : sortie TOR

6.2.2.1 Feux avant

L'affectation des entrées sur la carte entrée est la suivante :

GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
Etat clignotant	Etat phare	Etat code	Etat veilleuse	clignotant	phare	code	Veilleuse

6.2.2.2 Feux arrières

L'affectation des entrées sur la carte entrée est la suivante :

GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
Etat GP3	Etat clignotant	Etat code	Etat veilleuse	(klaxon)	clignotant	code	Veilleuse

Remarques : la commande klaxon est active sur le module feux arrières gauche.