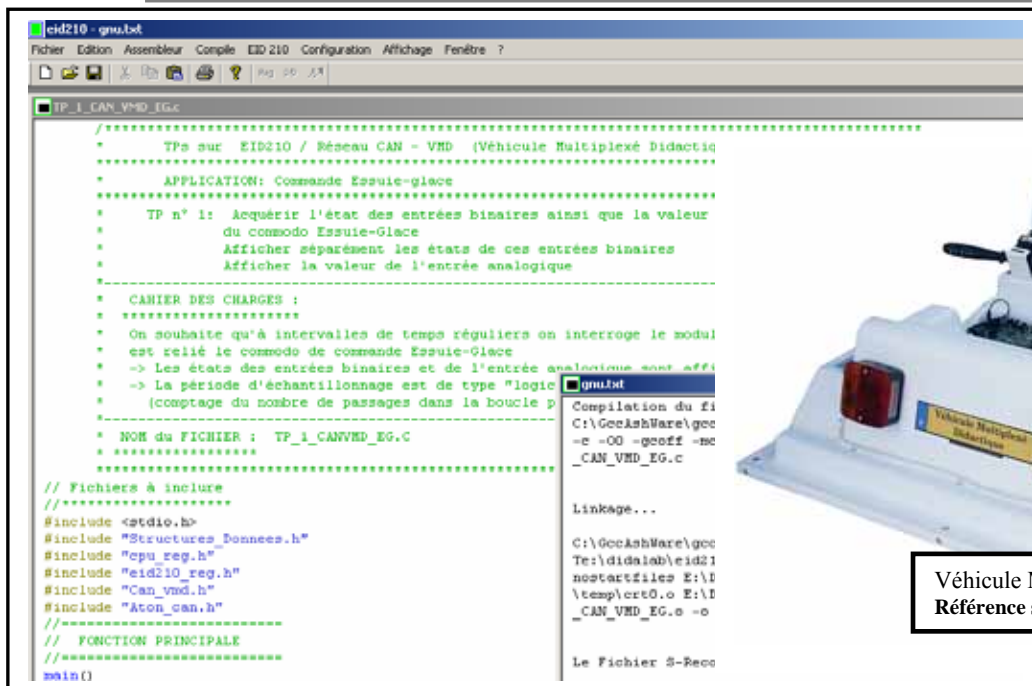


# MANUEL de TRAVAUX PRATIQUES

## Système CAN - V.M.D. Véhicule Multiplexé Didactique



```
eid210 - gnu.bat
Fichier Edition Assembleur Compile EID 210 Configuration Affichage Fenêtre ?
TP_1_CAN_VMD_EG.C
//*****
//      TPS sur EID210 / Réseau CAN - VMD (Véhicule Multiplexé Didactique)
//*****
//      APPLICATION: Commande Essuie-glace
//*****
//      TP n° 1: Acquérir l'état des entrées binaires ainsi que la valeur
//               du commodo Essuie-Glace
//               Afficher séparément les états de ces entrées binaires
//               Afficher la valeur de l'entrée analogique
//*****
//      CARNIER DES CHARGES :
//      *****
//      On souhaite qu'à intervalles de temps réguliers on interroge le module
//      est relié le commodo de commande Essuie-Glace
//      -> Les états des entrées binaires et de l'entrée analogique sont affichés
//      -> La période d'échantillonnage est de type "logique"
//      (comptage du nombre de passages dans la boucle principale)
//*****
//      NOM du FICHIER : TP_1_CANVMD_EG.C
//*****
// Fichiers à inclure
//*****
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"
#include "Aton_can.h"
//*****
//      FONCTION PRINCIPALE
//*****
main()
{
    //*****
    //      Compilation du fichier
    //      C:\GecashWare\gcc
    //      -c -OO -gcoff -mc
    //      _CAN_VMD_EG.C
    //*****
    //      Linkage...
    //      C:\GecashWare\gcc
    //      Te:\didalab\eid21
    //      nostartfiles E:\I
    //      \temp\crt0.o E:\I
    //      _CAN_VMD_EG.o -o
    //*****
    //      Le Fichier S-Rec...
```



Véhicule Multiplexé Didactique (V.M.D.)  
Référence système: VMD 01C



### Logiciels associés

- Environnement de développement intégré (Editeur, assembleur, linqueur, loader) Réf: EID210
- Compilateur "C" Réf: EID210 100
- Noyau temps réel MTR86 (option) Réf: EID210 200



### Notices techniques associées

- Sur la carte processeur EID210 000 seule Réf: EID210 010
- Sur le système V.M.D. Réf: EID055 010
- Sur le "CAN Expander" MCP25050 et Contrôleur CAN SJA1000

### Manuels de TP associés autres

- Sur la carte processeur EID210000 seule Réf: EID210040
- Sur la carte simulateur E/S EID210000 seule Réf: EID211040
- avec bus CAN-VMD et noyau temps réel MTR86 Réf: EID050 050
- avec carte réseau ethernet Réf: EID213 040



Z.A. La Clef St Pierre - 5, rue du Groupe Manoukian 78990 ELANCOURT France  
Tél. : 33 (0)1 30 66 08 88 - Télécopieur : 33 (0)1 30 66 72 20  
e-mail : [ge@didalab.fr](mailto:ge@didalab.fr) Web : [www.didalab.fr](http://www.didalab.fr)

Version du : 20/07/2010

Référence document : EID050040



**SOMMAIRE**

<b><u>1</u></b>	<b><u>TP N°1: FAIRE COMMUTER LES LAMPES D'UN BOC OPTIQUE</u></b>	<b><u>5</u></b>
1.1	SUJET:	5
1.2	ELEMENTS DE SOLUTION	6
1.2.1	Analyse	6
1.2.2	Organigramme	7
1.2.3	Programme en "C"	8
<b><u>2</u></b>	<b><u>TP N°2 : ACQUERIR L'ETAT DU COMMODO FEUX</u></b>	<b><u>9</u></b>
2.1	SUJET	9
2.2	ELEMENTS DE SOLUTION	10
2.2.1	Analyse	10
2.2.2	Organigramme	11
2.2.3	Programme en "C"	12
<b><u>3</u></b>	<b><u>TP N°3: VERIFIER LE FONCTIONNEMENT D'UN BLOC OPTIQUE</u></b>	<b><u>13</u></b>
3.1	SUJET	13
3.2	ELEMENTS DE SOLUTION	14
3.2.1	Analyse	14
3.2.2	Organigrammes	17
3.2.3	Programme en "C"	19
<b><u>4</u></b>	<b><u>TP N°4: COMMANDER FEUX A PARTIR DU COMMODO FEUX</u></b>	<b><u>23</u></b>
4.1	SUJET:	23
4.2	ELEMENTS DE SOLUTION	24
4.2.1	Analyse	24
4.2.2	Organigrammes	25
4.2.3	Programme en "C"	26
<b><u>5</u></b>	<b><u>TP N°5: COMMANDER LE MOTEUR ESSUIE GLACE</u></b>	<b><u>31</u></b>
5.1	SUJET	31
5.2	ELEMENTS DE SOLUTION	32
5.2.1	Analyse	32
5.2.2	Organigramme:	34
5.2.3	Programme en "C"	35
<b><u>6</u></b>	<b><u>TP N°6: FAIRE BATTRE LE BALAI D'ESSUIE GLACE</u></b>	<b><u>37</u></b>
6.1	SUJET	37
6.2	ELEMENTS DE SOLUTION	38
6.2.1	Analyse	38
6.2.2	Organigramme:	39
6.2.3	Programme en "C"	40
<b><u>7</u></b>	<b><u>TP N°7: REGULER LA VITESSE DU BALAI D'ESSUIE GLACE</u></b>	<b><u>43</u></b>
7.1	SUJET	43
7.2	ELEMENTS DE SOLUTION ETAPE N°1	44
7.2.1	Analyse étape n°1	44
7.2.2	Organigramme étape n°1	45
7.2.3	Programme en "C" de l'étape n°1	46

<b>7.3</b>	<b>ELEMENTS DE SOLUTION ETAPE N°2</b>	<b>48</b>
7.3.1	Analyse étape n°2	48
7.3.2	Organigramme étape n°2	49
7.3.3	Programme en langage "C"	50
<b>7.4</b>	<b>ELEMENTS DE SOLUTION ETAPE N°3</b>	<b>52</b>
7.4.1	Analyse étape n°3	52
7.4.2	Organigramme partiel étape n°3	52
7.4.3	Programme partiel étape n°3	52
<b>8</b>	<b>TP N°8: FAIRE LA COMMANDE DU SYSTEME ESSUIE GLACE</b>	<b>53</b>
<b>8.1</b>	<b>SUJET</b>	<b>53</b>
<b>8.2</b>	<b>ELEMENTS DE SOLUTION</b>	<b>54</b>
8.2.1	Analyse	54
8.2.2	Organigramme	55
8.2.3	Programme en "C"	57
<b>9</b>	<b>TP N°9: ENSEMBLE DES COMMANDES AU VOLANT</b>	<b>61</b>
<b>9.1</b>	<b>SUJET</b>	<b>61</b>
<b>9.2</b>	<b>ELEMENTS DE SOLUTION</b>	<b>62</b>
9.2.1	Analyse	62
9.2.2	Organigramme général	63
9.2.3	Programme en "C"	64
<b>10</b>	<b>ANNEXES</b>	<b>71</b>
<b>10.1</b>	<b>FICHIER DE DEFINITION PROPRE AU SYSTEME CAN_VMD</b>	<b>71</b>
<b>10.2</b>	<b>FICHIER DE DEFINITION PROPRE A LA CARTE ATON</b>	<b>73</b>

# 1 TP N°1: FAIRE COMMUTER LES LAMPES D'UN BOC OPTIQUE

## 1.1 Sujet

<b>Objectifs :</b>	<ul style="list-style-type: none"> <li>- Comprendre et utiliser les structures de données spécifiques proposées,</li> <li>- Comprendre et utiliser les fonctions spécifiques proposées.</li> <li>- Définir puis envoyer une trame de données à un module CAN destinataire, accessible une adresse donnée.</li> <li>- Tester si une trame a été reçue.</li> <li>- Visualiser sur l'écran les trames reçues ainsi que les trame envoyées.</li> </ul>
<b>Cahier des charges :</b>	<p>Au départ toutes les lampes du bloc "Feux avant droit" sont éteintes. On souhaite réaliser la fonction "chenillard" avec les 4 lampes du bloc (à intervalles de temps réguliers, on éteint la lampe précédemment allumée et on allume la suivante).</p> <p>→ Les trames reçues ou envoyées sur le bus CAN sont affichées.</p> <p>→ La temporisation est de type "logiciel" (comptage du nombre de passages dans la boucle principale)</p> <p>Le programme devra permettre, après un minimum de modifications, de réaliser la même fonction avec les blocs , "Feux avant gauche" puis "Feux arrière droit" et enfin "Feux arrière gauche".</p>

Matériels et logiciels nécessaires :

Micro ordinateur de type PC sous Windows 95 ou ultérieur

Logiciel Editeur-Assembleur-Debogueur

Si programmation en C , Compileur GNU C/C++ Réf : EID210100

Carte processeur 16/32 bits à microcontrôleur 68332 et son environnement logiciel

(Editeur-CrossAssembleur-Debogueur) Réf : EID210000

Carte réseau CAN PC/104 de chez ATON SYSTEMES Réf : EID004000

Réseau CAN avec modules 4 sorties de puissance destiné aux feux Réf : EID051000

Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003

Alimentation AC/AC 8V, 1A Réf : EGD000001,

Alimentation 12V pour l'alimentation des modules CAN (réseau "énergie").

Durée estimée : 3 heures

## 1.2 Eléments de solution

### 1.2.1 Analyse

#### Chenillard sur feux avant droit

Si l'on souhaite définir l'état des sorties d'un module 4sorties de puissance, il faut lui envoyer une trame de type IM "Input Message" avec la fonction "Write Register" sur son registre d'accès au port de sortie "GPLAT" (d'après notice technique du circuit MCP25050 page 22).

En effet, vu du module, il reçoit une trame de commande lui demandant de changer l'état de son registre de sortie qui impose l'état des différentes sorties du circuit d'interface MCP25050.

Pour un IM "Input Message" le registre concerné est le RXF1 ce qui conduit, pour les feux avant droit, à l'identificateur de basse 0E 88 xx xx (voir tableau des identificateur chapitre 1). Les 3 bits de poids faibles doivent être obligatoirement à 0 (d'après notice technique du circuit MCP25050 page 22) et on choisit de mettre également à 0 les autres bits indifférents, ce qui conduit en définitive à l'identificateur 0E880000 (seul 29 bits utiles) dont le label défini dans le fichier "CAN\_VMD.h" est "T\_Ident\_IM\_FVD".

Définition de la trame de commande (voir dans chapitre 1 de ce document la définition des différents champs de la trame destinée à cet exemple.

#### Remarques

- Dans la trame de commande, il y a trois paramètres, définis dans la zone "data" de la trame:
  - Paramètre "addr" (au rang 0 des "data") définit l'adresse registre concerné par l'écriture.  
Dans le cas présent, c'est GPLAT et son adresse est 1E h (Page 15 notice MCP25050).  
 $02h + \text{décalage} (\text{note1}) = 02h + 1Ch = 1Eh$
  - Paramètre "mask" (au rang 1 des "data") permet de laisser inchangés certains bits du registre si ceux-ci ne doivent pas être affectés par l'opération d'écriture.  
Dans notre cas les sorties de puissance sont connectées sur les 4 bits de poids faibles du port. Les 4 bits de poids forts devront donc être masqués, soit la valeur du masque: 0Fh
  - Paramètre "value" (au rang 2 des "data") permet de définir l'état des sorties non masquées.  
Dans notre cas, pour réaliser la fonction "chenillard", il faudra donner les valeurs successives 00 , puis 01, puis 02, puis 04 et enfin 08.
- Suite à un IM "Input Message", si le message a été bien reçu par son destinataire, celui-ci doit renvoyer une trame d'acquiescement module dont l'identificateur est défini par TXID1. Dans le cas du module « feux avant droit », l'identificateur de message d'acquiescement sera 0E A0 00 00 dont le label "T\_Ident\_AIM\_FVD" a été défini dans le fichier CAN\_VMD.h (d'après tableau chapitre 1).

**Il faut veiller à n'envoyer une deuxième trame de commande à un nœud que s'il a répondu par une trame d'acquiescement "AIM" à la première.**

#### Chenillard sur les autres feux

Les seules choses à changer ce sont les identificateurs :

Pour Feux aVant Gauche

Identificateur message écriture sur port sortie (registre RXF1): 0E080000 -> T\_Ident\_IM\_FVG

Identificateur message d'acquiescement (registre TXD1): 0E200000 -> T\_Ident\_AIM\_FVG

Pour Feux aRrière Droit

Identificateur message écriture sur port sortie (registre RXF1): 0F880000 -> T\_Ident\_IM\_FRD

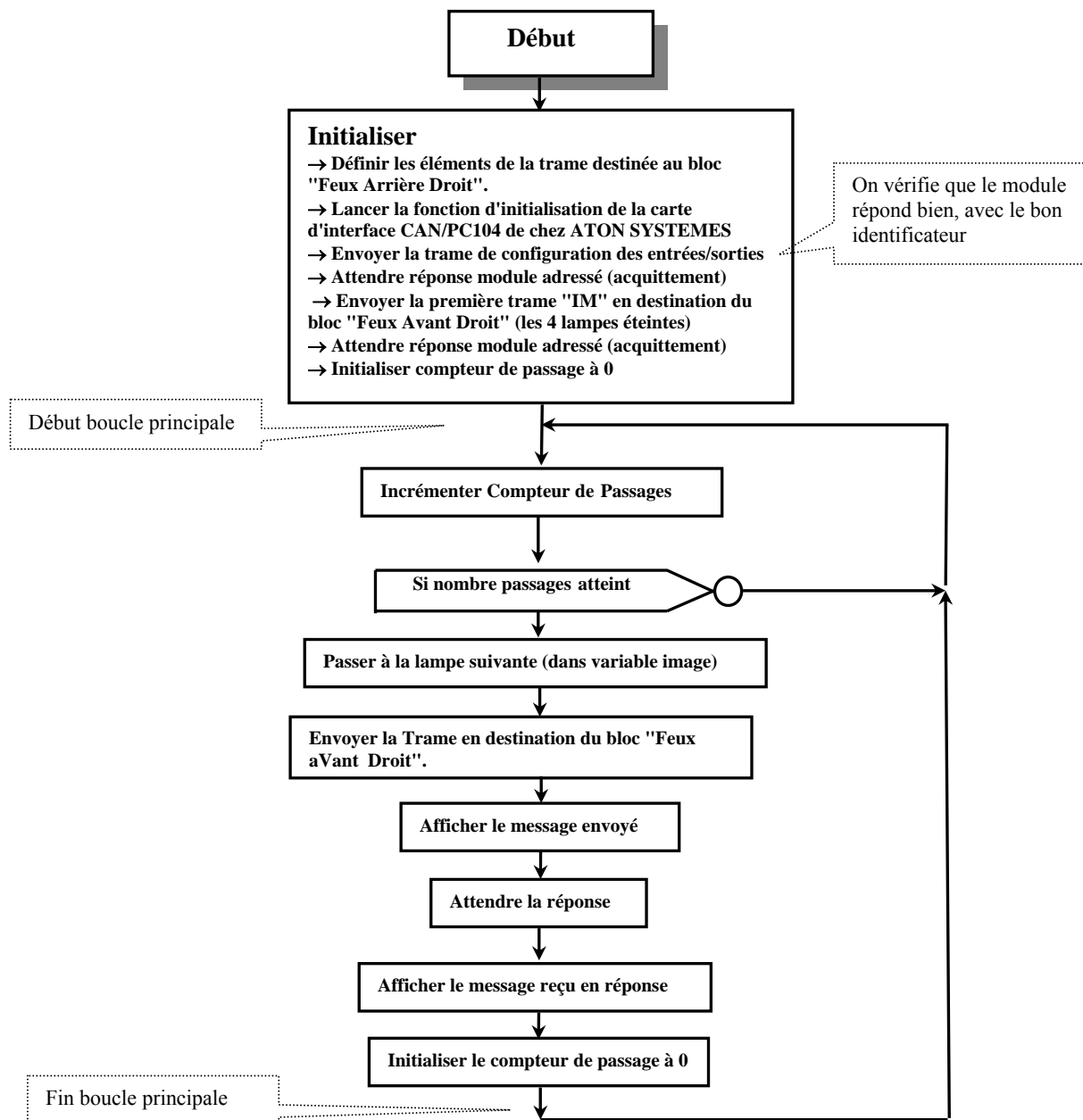
Identificateur message d'acquiescement (registre TXD1): 0FA00000 -> T\_Ident\_AIM\_FRD

Pour Feux aRrière Gauche

Identificateur message écriture sur port sortie (registre RXF1): 0F080000 -> T\_Ident\_IM\_FRG

Identificateur message d'acquiescement (registre TXD1): 0F200000 -> T\_Ident\_AIM\_FRG

## 1.2.2 Organigramme



### 1.2.3 Programme en "C"

```

/*****
* TP sur EID210 / Réseau CAN - VMD (Véhicule Multiplexé Didactique)
* *****/
* TP n 1: Faire commuter les feux d'un bloc optique
* -----
* CAHIER DES CHARGES :
* *****/
* On souhaite qu'à intervalles de temps réguliers on déactive la sortie précédemment activée
* pour activer la suivante (fonction chenillard)
* -> Les trames reçues et envoyées sur le bus CAN sont affichées
* -> La temporisation est de type "logiciel"
* (comptage du nombre de passages dans la boucle principale)
* -----
* NOM du FICHIER : CAN_VMD_TP1.C
* *****/
*****/

// Fichiers à inclure
//*****
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"
#include "Aton_can.h"
//=====
// FONCTION PRINCIPALE
//=====
main()
{
    // Déclaration des variables locales
    Trame Trame_Recue;
    Trame T_IM_Feux; // Trame de type "Input Message" pour commande module 4 Sorties de puissance
    int Compteur_Passage, Cptr_TimeOut;
    char I_Message_Pb_Affiche;
    // Initialisations
    //*****
    clrscr();
    /* Initialisation DU SJA1000 de la carte ATON-Systemes sur bus PC104 */
    Init_Aton_CAN();
    // Définition trame pour activer bloc optique arrière droit
    // D'après doc SJA1000 et doc MCP25050 pages 22 (fonction "Write Register") et 15 (Adresse GPLAT)
    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=0x0F880000; // 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) adresse = 1Fh
    page 16
    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
    // Envoi trame pour définir de la direction des entrées sorties
    I_Message_Pb_Affiche=0;
    do {Ecrire_Traine(T_IM_Feux); // Envoyer trame sur réseau CAN
        Cptr_TimeOut=0;
        do{Cptr_TimeOut++;}while((Lire_Traine(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
        if(Cptr_TimeOut==200)
        {
            if(I_Message_Pb_Affiche==0)
            {
                I_Message_Pb_Affiche=1;
                gotoxy(2,10);
                printf(" Pas de reponse a la trame de commande en initialisation \n");
                printf(" Verifier si alimentation 12 V est OK \n");
            }
        }
    }while(Cptr_TimeOut==200);
    // Initialiser les sorties à 0
    T_IM_Feux.data[0]=0x1E; // première donnée -> "Adresse" du registre concernée (GPLAT définit l'état des sorties) 02h+1Ch = 1Eh
    T_IM_Feux.data[1]=0x0F; // deuxième donnée -> "Masque" -> les sorties sont sur les 4 bits de poids faibles
    T_IM_Feux.data[2]=0x00; // troisième donnée -> "Valeur" -> au départ toutes les sorties sont à 0 (lampes éteintes)
    // Envoi trame pour définir les sorties à 0
    Ecrire_Traine(T_IM_Feux); // Envoyer trame sur réseau CAN
    do{while(Lire_Traine(&Trame_Recue)==0); // Attendre réponse "Acquittement"
    // Initialisations des variables diverses
    clrscr(); Compteur_Passage=0;
    // Pour afficher titre
    gotoxy(1,2);
    printf(" TP n: 1 COMMANDER FEUX D'UN BLOC OPTIQUE \n");
    printf(" *****/ \n");
    // Boucle principale
    //*****
    while(1) {Compteur_Passage++;
        if (Compteur_Passage==400000)
        {
            // C'est la fin temporisation
            // On passe à la lampe suivante en modifiant le paramètre "Valeur" du message
            switch(T_IM_Feux.data[2])
            {
                case 0 : T_IM_Feux.data[2]=0x01;
                    break;
                case 1 : T_IM_Feux.data[2]=0x02;
                    break;
                case 2 : T_IM_Feux.data[2]=0x04;
                    break;
                case 4 : T_IM_Feux.data[2]=0x08;
                    break;
                case 8 : T_IM_Feux.data[2]=0;
                    break;
                default : T_IM_Feux.data[2]=0;
            }
            gotoxy(1,5),printf("Ecrire sur feux arrieres droit:\n");
            Affiche_Traine(T_IM_Feux); // Afficher trame "IM" envoyée à l'écran
            Ecrire_Traine(T_IM_Feux); // Envoyer trame sur réseau CAN
            do{while(Lire_Traine(&Trame_Recue)==0); // Attendre la réponse "Acquittement"
            gotoxy(1,10),printf("Trame recue en reponse\n");
            Affiche_Traine(Trame_Recue); // Afficher trame "AIM" reçue, envoyée à l'écran
            Compteur_Passage=0;
            }
        }
    } // FIN de la boucle principale
} // FIN de la fonction principale

```



## 2 TP N°2 : ACQUERIR L'ETAT DU COMMODO FEUX

### 2.1 Sujet

<b>Objectifs :</b>	<ul style="list-style-type: none"> <li>- Définir, puis envoyer une trame interrogative à un module d'entrées, accessible à une adresse définie.</li> <li>- Tester si une trame a été reçue.</li> <li>- Extraire d'une trame réponse les informations attendues.</li> <li>- Visualiser sur l'écran les trames reçues ainsi que les trames envoyées.</li> <li>- Visualiser sur l'écran les données attendues.</li> </ul>
<b>Cahier des charges :</b>	<p>A intervalles de temps réguliers, on interroge le module sur lequel est connecté le commodo lumières afin de connaître son état.</p> <p>→ Les trames reçues ou envoyées sur le bus CAN sont affichées.</p> <p>→ La temporisation est de type "logiciel" (comptage du nombre de passages dans la boucle principale)</p> <p>→ Les différentes commandes imposées par la position de la manette commodo seront affichées individuellement.</p>

Matériels et logiciels nécessaires :

Micro ordinateur de type PC sous Windows 95 ou ultérieur

Logiciel Editeur-Assembleur-Debugueur

Si programmation en C , Compileur GNU C/C++ Réf : EID210100

Carte processeur 16/32 bits à microcontrôleur 68332 et son environnement logiciel  
(Editeur-CrossAssembleur-Debugueur) Réf : EID210000

Carte réseau CAN PC/104 de chez ATON SYSTEMES Réf : EID004000

Réseau CAN avec un module 8 Entrées logiques destiné au commodo Réf : EID050000

Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003

Alimentation AC/AC 8V, 1A pour l'alimentation de l'unité centrale Réf : EGD000001,

Alimentation 12V pour l'alimentation des modules CAN (réseau "énergie").

Durée : 3 heures

## 2.2 Eléments de solution

### 2.2.1 Analyse

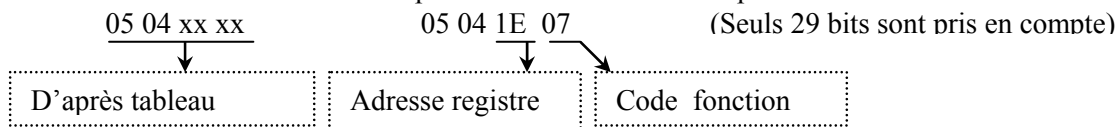
A intervalles de temps réguliers, on interroge le module 8 entrées sur lequel est connecté le commodo lumière.

#### Définition de la trame image de la trame interrogative qui sera envoyée

Dans ce cas, la trame envoyée par le contrôleur CAN (Circuit SJA1000 sur carte CAN\_PC104) sera vue par le récepteur (circuit MCP25050 sur module) comme un "Information Request message", avec la fonction "Read register" (voir documentation technique du MPC25025 pages 22).

D'après le tableau donné page 22 de la notice du MCP25050, l'identificateur lui-même contiendra l'adresse du registre lu. Cette adresse est placée sur les bits ID15 à ID8 de l'identificateur en mode étendu (bits qui seront réceptionnés et placés dans le registre RXBEID8). Le registre concerné est GPPIN d'adresse 1Eh " (voir documentation technique du MPC25025 pages 37)..

D'autre part, les trois bits de poids faibles de l'identificateur en mode étendu devront être positionnés à 1. L'identificateur défini dans le chapitre 1 devra donc être complété comme suit:



→ Définition de variables structurées sous le modèle "Trame":

```
Trame T_IRM_Commodo_Feux;
```

```
// Trame destinée à l'interrogation du module 8 entrées sur lequel est connecté le commodo lumière
```

Rem: La variable structurée T\_IRM\_Commodo comportera 5 octets utiles seulement, 1 octet pour trame\_info et 4 octets pour l'identificateur en mode étendu (qui comprendra l'adresse du registre concerné par la lecture.

→ Accès et définition des différents éléments de la variable structurée "T\_IRM\_Commodo"

```
T_IRM_Commodo.trame_info.registre=0x00; //On initialise tous les bits à 0
T_IRM_Commodo.trame_info.champ.extend=1; //On travaille en mode étendu
T_IRM_Commodo.trame_info.champ.rtr=0x01; // Type trame
T_IRM_Commodo.trame_info.champ.dlc=0x01; //Il y aura 1 octet de données
T_IRM_Commodo_ad.ident.extend.identificateur.ident=0x05041E07;
///! c'est sur 29 bits                      Pour définir l'adresse du commodo
```

Des labels définissant les différents identificateurs ont été déclarés dans le fichier CAN\_VMD.h

#### Définition de la trame image de la trame qui sera reçue en réponse

D'après la définition des identificateurs donnée en chapitre 1, une trame de réponse à une IRM a le même identificateur que la trame interrogative qui en a été à l'origine.

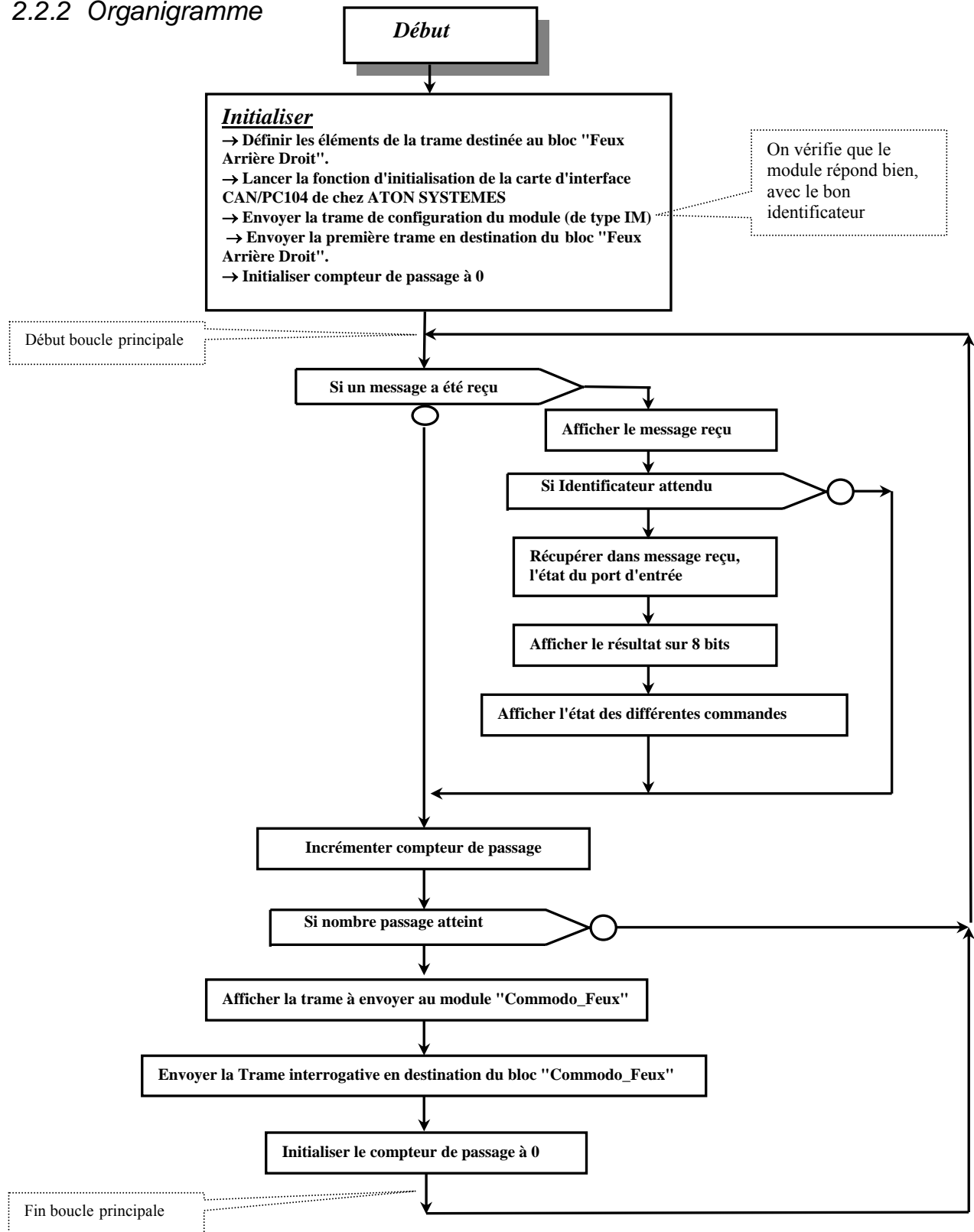
Vu du module (du MCP25050), la réponse à un IRM (Information Request Message) est un OM (Output Message).

La différence avec la trame interrogative origine est que cette trame réponse comporte le paramètre "value" (au rang 0 de la partie "data" de la trame). Ce paramètre est l'image du port d'entrée. On récupère donc l'état des différentes commandes.

#### Accès aux différents états binaires des commandes

Le paramètre "value" de la trame réponse, récupéré dans la donnée de rang 0 est un octet image des entrées. Les différents bits de cet octet sont extraits individuellement grâce à une structure de données définie dans le fichier CAN\_VMD.

## 2.2.2 Organigramme



## 2.2.3 Programme en "C"

```

/*****
* TP sur EID210 / Réseau CAN - VMD (Véhicule Multiplexé Didactique)
*****/
* TP n 2: Acquérir l'état du commodo de commande feux et afficher les états des entrées
*-----
* CAHIER DES CHARGES :
* *****
* On souhaite qu'à intervalles de temps réguliers on interroge le module 8 entrées sur lequel
* est relié le commodo de commande des phares
* -> Les trames reçues et envoyées sur le bus CAN sont affichées
* -> Les états des entrées sont affichés
* -> La temporisation est de type "logiciel" (comptage du nombre de passages dans la boucle principale)
*-----
* NOM du FICHIER : CAN_VMD_TP2.C
* *****
*****/

// Fichiers à inclure
//*****
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"
#include "Aton_can.h"
//=====
// FONCTION PRINCIPALE
//=====
main()
{int Compteur_Passage; // Définition de variables locales
Trame Trame_Recue;
Trame T_IRM_Commodo_Feux; // Trame pour interroger Module 8E sur Commodo Feux IRM -> Information Request Message Trame
T_IM_Commodo_Feux; // Trame pour interroger Module 8E sur Commodo Feux IM -> Information Message -> Trame de commande
unsigned char Cptr_TimeOut,I_Message_Pb_Affiche;
// Initialisations
//*****
clrscr();
/* Initialisation DU SJA1000 de la carte industrielle Aton-CAN au format PC104 */
Init_Aton_CAN();
// Pour initialiser les liaisons en entrées
T_IM_Commodo_Feux.trame_info.registre=0x00;
T_IM_Commodo_Feux.trame_info.champ.extend=1; // On travaille en mode étendu
T_IM_Commodo_Feux.trame_info.champ.dlc=0x03; // Il y aura 3 données de 8 bits (3 octets envoyés)
T_IM_Commodo_Feux.ident.extend.identificateur.ident=Ident_T_IM_Commodo_Feux;
T_IM_Commodo_Feux.data[0]=0x1F; // première donnée -> "Adresse" du registre concerné:GPDDR donne la direction des I/O page 16
T_IM_Commodo_Feux.data[1]=0x7F; // deuxième donnée -> "Masque" -> Tous les bits concernés sauf GP0 (voir doc page 16)
T_IM_Commodo_Feux.data[2]=0x7F; // troisième donnée -> "Valeur" -> Tous les bits en entrée
// Envoi trame pour définir de la direction des entrées sorties
I_Message_Pb_Affiche=0;
do {Ecrire_Traine(T_IM_Commodo_Feux); // Envoyer trame sur réseau CAN
Cptr_TimeOut=0;
do{Cptr_TimeOut++;}while((Lire_Traine(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
if(Cptr_TimeOut==200)
{if(I_Message_Pb_Affiche==0)
{I_Message_Pb_Affiche=1;
gotoxy(2,10);
printf(" Pas de reponse a la trame de commande en initialisation \n");
printf(" Verifier si alimentation 12 V est OK \n");}}
}while(Cptr_TimeOut==200);
clrscr();
// Pour trame interrogative envoyée au commodo lumiere -> 'IRM' (Information Request Message)
// Définir données d'identification
T_IRM_Commodo_Feux.trame_info.registre=0x00;
T_IRM_Commodo_Feux.trame_info.champ.extend=1;
T_IRM_Commodo_Feux.trame_info.champ.dlc=0x01;
T_IRM_Commodo_Feux.trame_info.champ.rtr=1;
T_IRM_Commodo_Feux.ident.extend.identificateur.ident=Ident_T_IRM_Commodo_Feux; // Voir définitions dans fichier CAN_VMD.h
Ecrire_Traine(T_IRM_Commodo_Feux); // On envoi une première trame
// Initialiser les variables diverses
Compteur_Passage=0;
// Pour afficher titre
gotoxy(1,2);
printf(" TP n: 2 ACQUERIR ET AFFICHER ETAT COMMODO FEUX \n");
printf(" ***** \n");
// Boucle principale
//*****
while(1) { if (I==Lire_Traine(&Trame_Recue)) // On teste si une trame a été reçue; La fonction renvoie 1 dans ce cas
{gotoxy(4,10),printf("Trame recue en reponse a la demande: c'est une 'OM' (Output Message) \n");
Affiche_Traine(Trame_Recue);
if (Trame_Recue.ident.extend.identificateur.ident==Ident_T_IRM_Commodo_Feux)
{ // On a reçu l'état du commodo donc on affiche les nouveaux états
Etat_Commodo_Feux.valeur==Trame_Recue.data[0];
gotoxy(4,16);
printf("Etat des differentes entrees imposees par le commodo:\n");
printf(" Octet recupere et complemente (en Hexa) =%2.2x\n",Etat_Commodo_Feux.valeur);
printf(" Veilleuse= %d , Code= %d , Phare= %d\n",Cde_Veilleuse,Cde_Code,Cde_Phare);
printf(" clignotant gauche= %d , clignotant droit= %d\n",Cde_Clign_Gauche,Cde_Clign_Droit);
printf(" Klaxon= %d\n",Cde_Klaxon);
printf(" Feux de stop= %d\n",Cde_Stop);
printf(" Commande Warning= %d\n",Cde_Warning);}
}
Compteur_Passage++;
if (Compteur_Passage==5000)
{Compteur_Passage=0; // C'est la fin de temporisation
gotoxy(4,6);
printf("Trame de demande etat commodo: c'est une 'IRM' Input Request Mesage\n");
Affiche_Traine(T_IRM_Commodo_Feux);
Ecrire_Traine(T_IRM_Commodo_Feux);
}
}
} // FIN de la boucle principale
} // FIN fonction principale

```

### 3 TP N°3: VERIFIER LE FONCTIONNEMENT D'UN BLOC OPTIQUE

#### 3.1 Sujet

<p><b>Objectifs :</b></p>	<ul style="list-style-type: none"> <li>- Analyser un schéma structurel afin de définir la mise œuvre d'une fonction matérielle par une fonction logicielle.</li> <li>- Enchaîner des trames interrogatives et des trames de commande pour satisfaire un cahier des charges imposé.</li> <li>- Tester les trames réponse reçues.</li> <li>- Extraire d'une trame réponse les informations attendues.</li> <li>- Analyser les informations reçues et faire un diagnostic.</li> <li>- Représenter par un diagramme des états un cahier des charges imposé.</li> <li>- Coder et programmer un diagramme des états.</li> <li>- Réaliser des actions cycliques, de périodes précises.</li> </ul>
<p><b>Cahier des charges :</b></p>	<p>On souhaite en enchaînement cyclique des différents états d'un bloc optique (Rien, Veilleuse, Veilleuse + Code, Veilleuse + Phare etc...) et un fonctionnement du clignoteur.</p> <p>On réalisera en plus des fonctions de contrôle:</p> <ul style="list-style-type: none"> <li>- On vérifiera que le module commandé renvoi bien une trame d'acquiescement.</li> <li>- On vérifiera (par fonction logicielle) que les lampes commandées sont effectivement allumées (consommant du courant).</li> </ul> <p>→ On affichera quelles sont les lampes concernées ainsi que le résultat du test.</p> <p>→ Le clignotement (du clignotant) sera indépendant de la permutation des autres lampes ainsi que de la période de contrôle.</p> <p>On impose les périodes suivantes, dans l'ordre de priorité décroissante:</p> <ul style="list-style-type: none"> <li>- période de permutation des lampes 3,5 S,</li> <li>- période de commutation du clignoteur 1,6 S.</li> </ul> <p>Ces périodes devront pouvoir être modifiées facilement.</p> <p>→ Le programme devra permettre un changement aisé de bloc cible.</p>

Matériels et logiciels nécessaires :

Micro ordinateur de type PC sous Windows 95 ou ultérieur

Logiciel Editeur-Assembleur-Debogueur

Si programmation en C , Compilateur GNU C/C++ Réf : EID210100

Carte processeur 16/32 bits à microcontrôleur 68332 et son environnement logiciel (Editeur-CrossAssembleur-Debogueur) Réf : EID210000

Carte réseau CAN PC/104 de chez ATON SYSTEMES Réf : EID004000

Réseau CAN avec modules 4 Sorties de puissance destinés aux feux Réf : EID051000 et le bloc optique associé

Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003

Alimentation AC/AC 8V, 1A pour l'alimentation de l'unité centrale Réf : EGD000001,

Alimentation 12V pour l'alimentation des modules CAN (réseau "énergie").

Durée : 4 heures

## 3.2 Eléments de solution

### 3.2.1 Analyse

#### Principe de détection de charge électrique coupée (rupture filament)

Le circuit de puissance de référence "VN05", qui équipe les modules 4 sorties de puissance, génère un signal de diagnostic repérée "STAT" -> STATus (se référer à la data sheet du circuit VN05).

Dans le cas où on active le circuit de puissance, si le courant de charge est proche de 0 (filament ampoule coupé par exemple), le signal "STATus" passe à 0. Le seuil du courant de sortie qui entraîne la mise à 0 de la sortie "STATus" est de 5 mA (valeur mini) à 180 mA (valeur maxi).

Dans le cas des modules 4 sorties de puissance, les LEDs qui sont destinées à indiquer si une sortie de puissance est activée ne consomment pas un courant suffisant pour inhiber cette fonction de diagnostic.

D'après la notice technique du système CAN-VMD et le schéma structurel du module 4sorties de puissance, ces 4 signaux "STATus" sont reliés au circuit d'interface CAN MCP25050 et constituent donc des entrées de diagnostic que l'on peut lire via le réseau CAN:

- la sortie "STATus" du circuit de puissance pilotée par GP0 est reliée à GP4,
- la sortie "STATus" du circuit de puissance pilotée par GP1 est reliée à GP5,
- la sortie "STATus" du circuit de puissance pilotée par GP2 est reliée à GP6,
- la sortie "STATus" du circuit de puissance pilotée par GP3 est reliée à GP7.

#### Activation des lampes et diagnostic du bloc optique avant droit

Pour allumer les lampes, il faut envoyer une trame de type IM "Input message" avec la fonction "Write Register" sur son registre d'accès au port de sortie "GPPIN" (d'après notice technique du circuit MCP25050 page 22).

Dans ce cas, d'après tableau donné chapitre 1, pour le bloc avant droit, l'identificateur sera 0E880000, le paramètre "addr" sera 1E (adresse du registre GPLAT – page 37 de la "Data sheet" du MCP25050), le paramètre "mask" sera 0F (les 4 sorties sont sur les 4 bits de poids faibles du port), le paramètre "value" sera défini par l'état souhaité.

Pour récupérer les états logiques imposés sur un port d'entrée, il faut envoyer au module considéré une trame IRM "Information Request message" avec la fonction "Read register". Dans ce cas, l'identificateur contient l'adresse du registre considéré (GPPIN d'adresse 1E - d'après notice technique du circuit MCP25050 page 37) ainsi que le code fonction 07.

Par conséquent, pour le feu avant droit (d'après le tableau donné au chapitre 1 de ce document) l'identificateur à donner sera en définitive 0E 841E 07 et la trame ne comportera pas de paramètre en zone "data".

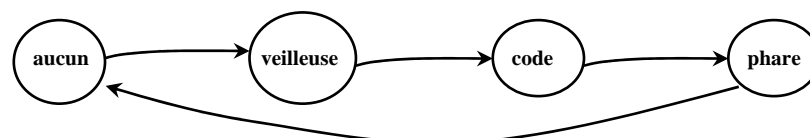
Le module récepteur de cette trame répondra avec le même identificateur, mais avec au rang 0 de la zone "Data", l'état du port d'entrée.

#### Permutation des lampes

On souhaite la suite des états des lampes suivante:

aucune puis, veilleuse seule puis, veilleuse + code puis, veilleuse + phare.

Cet enchaînement séquentiel peut être représenté par le diagramme de états représenté ci-après



On passera d'un état à un autre en fin de "temporisation feux".

De même pour la lampe "Clignotant", le changement d'état se fera à chaque fin de "temporisation clignotant".

## Réalisation des temporisations

On envisage de mettre en œuvre le "timer programmable" interne au microprocesseur. On configure celui-ci pour qu'il génère une interruption toutes les 10 mS. Un dispositif de comptage permet de positionner des indicateurs binaires informant de la fin de telle ou telle temporisation.

### Initialisations à réaliser:

Les deux registres internes au microcontrôleur "**PICR**" et "**PITR**" ont été définis dans le fichier Cpu\_reg.h

```
#define PITR    *(short *) (0xFFFFA24)
#define PICR    *(short *) (0xFFFFA22)
```

Il suffira d'effectuer les initialisations suivantes:

```
SetVect(96,&irq_bt);           // mise en place de l'autovecteur
PITR = 0x0048;                 // Une interruption toutes les 10 millisecondes
PICR = 0x0760;                 // 96 = 60H
```

où "**irq\_bt**" n'est autre que le nom de la fonction d'interruption (voir chapitre suivant l'ordinogramme de cette fonction d'interruption)

## Structure de données

Il est utile d'avoir en mémoire, une image de l'état des lampes du bloc optique (image de l'état du port de sortie du module). Quand on souhaite changer l'état d'une lampe, on agit sur l'un des bits de cette image. Ensuite cette image fait partie de la trame de commande.

La donnée envoyée dans une trame de commande (ou récupérée suite à une trame interrogative) étant sur 8 bits on utilise la structure de données "**byte\_bits**" définie dans le fichier "Structures-Donnees.h".

```
/* Union pour accéder à un octet (BYTE) soit en direct, soit en individualisant les 8 bits
//*****
union byte_bits
{
    struct
    {
        unsigned char b7:1;
        unsigned char b6:1;
        unsigned char b5:1;
        unsigned char b4:1;
        unsigned char b3:1;
        unsigned char b2:1;
        unsigned char b1:1;
        unsigned char b0:1;
    }bit;
    BYTE valeur;
};
```

### Pour la commande des feux

On définit la variable image

```
byte_bits Image_Feux;
```

On définit ensuite des variables binaires, images de l'état des lampes:

- pour un bloc optique avant

```
#define Veilleuse Image_Feux.bit.b0
#define Code Image_Feux.bit.b1
#define Phare Image_Feux.bit.b2
#define Clignot Image_Feux.bit.b3
```

- pour un bloc optique arrière

```
#define Lumiere Image_Feux.bit.b0
#define Stop Image_Feux.bit.b1
#define Clignot Image_Feux.bit.b2
#define Autre Image_Feux.bit.b3 // sur le V.M.D. c'est le claxon qui est piloté
```

Pour allumer une lampe, il suffira de:

- changer l'état dans la mémoire image,

```
Phare=1; // pour exemple
```

- transférer la mémoire image dans le paramètre "value" de trame de commande (trame IM),

```
T_IM_Feux.data[2]= Image_Feux.valeur;
```

- envoyer la trame de commande,

```
Ecrire_Trame(T_IM_Feux); // IM -> pour se rappeler que c'est une trame de commande
```

*Pour le contrôle du bon fonctionnement des ampoules*

De même pour le contrôle de l'état des ampoules:

- On définit la variable image `byte_bits Image_Etat_Feux;`

On définit ensuite des variables binaires, images de l'état des lampes:

```
#define Etat_Veilleuse Image_Etat_Feux.bit.b0
#define Etat_Code Image_Etat_Feux.bit.b1
#define Etat_Phare Image_Etat_Feux.bit.b2
#define Clignot Image_Etat_Feux.bit.b3
```

Pour lire l'état, on envoie une trame interrogative

`Ecrire_Trame(T_IRM_Feux);` // IRM -> pour se rappeler que c'est une trame interrogative

Dans la trame de réponse on récupère dans le paramètre de rang 0, le résultat de lecture du port

`Image_Etat_Feux.valeur = Trame_recue.data[0];`

On peut alors comparer les états logique des bits "status" on fonction des sorties pilotées, soit par exemple:

- si `Phare = 1` et `Etat_Phare = 0` c'est que ampoule grillée ou rien n'est connecté,
- si `Phare = 1` et `Etat_Phare = 1` c'est OK.

*Pour le codage du diagramme des états*

On envisage un codage de type décimal de chacun des états et une variable qui prendra successivement les valeurs correspondante:

```
#define Etat_aucune 0
#define Etat_veilleuse 1
#define Etat_code 2
#define Etat_phare 3
```

### Structure du programme

La fonction principale est composée de deux parties:

- une partie "Initialisation" qui n'est exécutée qu'une seule fois,
- une boucle principale qui est parcourue tant que l'on ne fait pas un "Reset".

Dans la boucle principale, on n'envoie une trame que dans la mesure où le module récepteur de la trame précédemment envoyée a répondu:

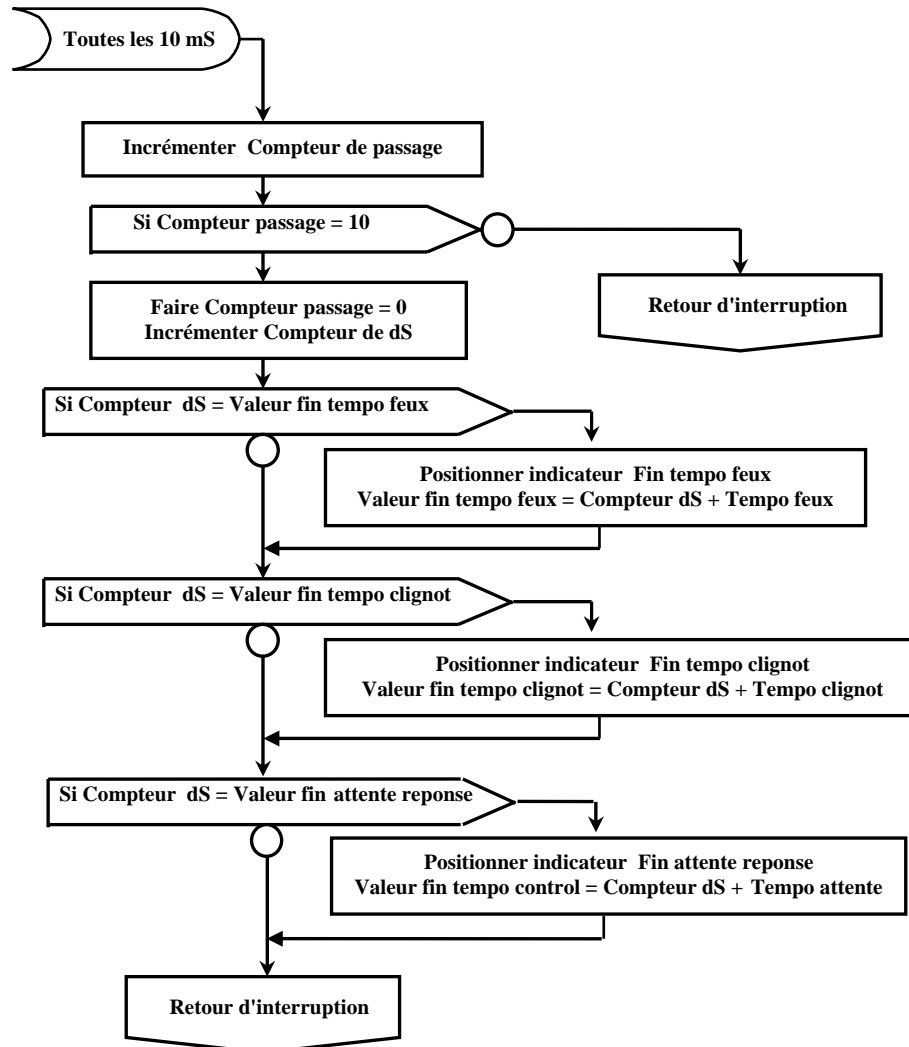
- par une trame d'acquittement dans le cas d'une réponse à une "IM",
- soit par une trame réponse avec paramètre dans le cas d'une réponse à une "IRM".

On peut envisager la mise en place d'un "Time out" temps maxi d'attente réponse après envoi d'une trame.



### 3.2.2 Organigrammes

Organigramme décrivant la génération des indicateurs de fin de temporisation:

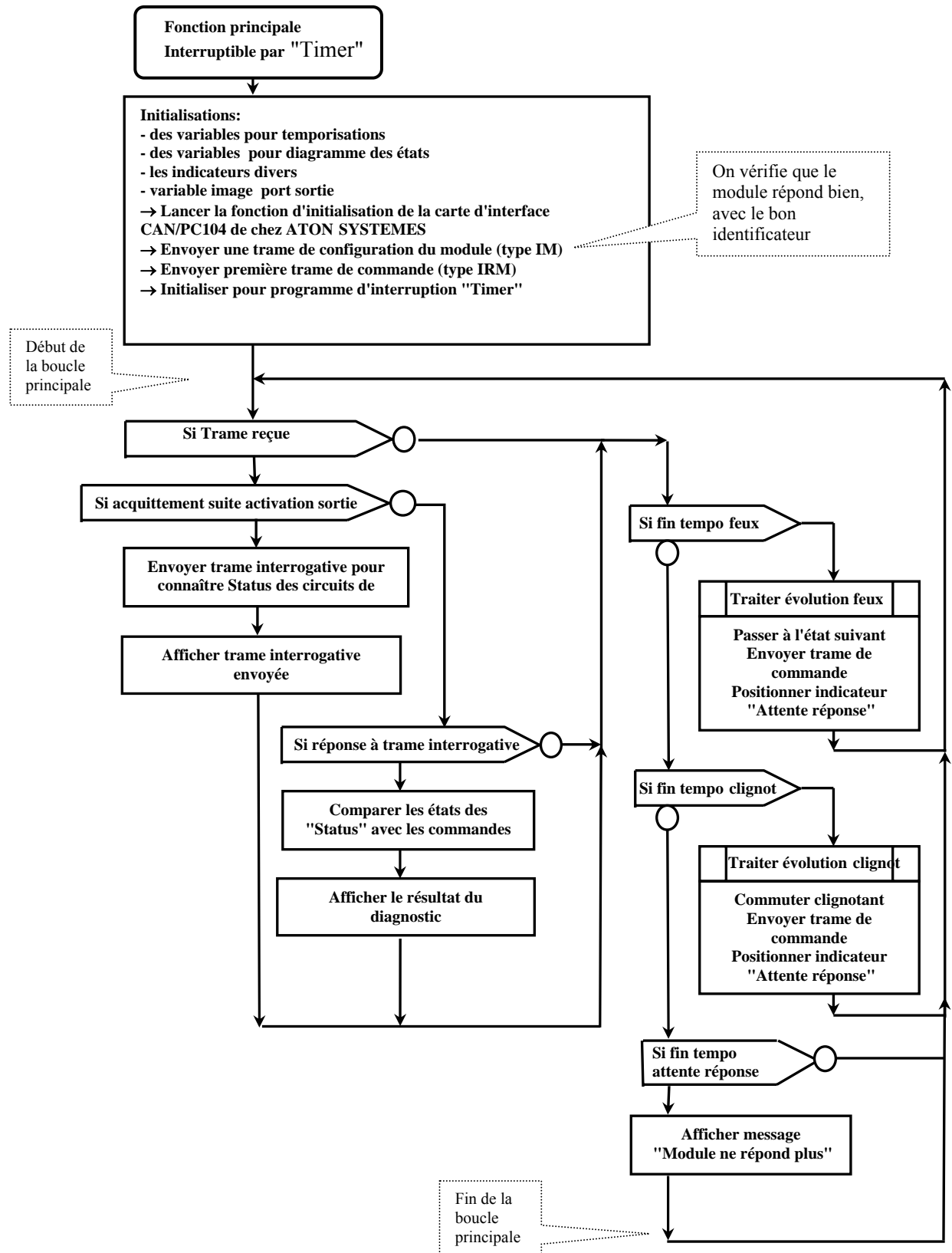


#### Remarque

Dans la partie "Initialisation" il faudra procéder aux affectations suivantes:

- Compteur ds = 0,
- Valeur fin tempo feu = Tempo feu ,
- Valeur fin tempo clignot = Tempo clignot,
- Valeur fin attente reponse = Tempo attente reponse.

# Organigramme général de la fonction principale



### 3.2.3 Programme en "C"

```

/*****
*
*      TP sur  EID210 / Réseau CAN - VMD  (Véhicule Multiplexé Didactique)
*****
*
*      TP n 3: Vérifier le fonctionnement d'un bloc optique
*-----
*      CAHIER DES CHARGES :
*      *****
*
*      On souhaite en enchaînement cyclique des différents états du bloc optique avant droit
*      (Rien, Veilleuse, Veilleuse+Stop, Veilleuse+Phare etc) et un fonctionnement du clignoteur.
*      On réalisera en plus des fonctions de controle:
*      - On vérifiera que le module commandé renvoi bien une trame d'acquiescement.
*      - On vérifiera (par fonction logicielle) que les lampes commandées sont effectivement
*      allumées
*      - On affichera quelles sont les lampes concernées ainsi que le résultat du test.
*      - Le clignotement (du clignotant) sera indépendant de la permutation des autres lampes.
*      On impose les périodes suivantes, dans l'ordre de priorité décroissante:
*      - période de permutation des lampes 3,5 S,
*      - période de commutation du clignoteur 1,6 S,
*      Ces périodes devront pouvoir être modifiées facilement.
*      - Le programme devra permettre un changement aisé de bloc cible.
*-----
*      NOM du FICHIER :  CAN_VMD_TP3.C
*
*      *****
*****
/

// Fichiers à inclure
//*****
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"
#include "Aton_can.h"

// Pour les temporisations
#define Tempo_Feux 30 // En dixième de seconde -> 3 S
#define Tempo_Clignot 16 // En dixième de seconde -> 1,6 S
#define Tempo_Att_Rep 40 // Attente réponse en dixième de seconde -> 4 S
// Pour le codage des états
#define Etat_aucune 0
#define Etat_veilleuse 1
#define Etat_code 2
#define Etat_phare 3
// Déclaration des variables
//-----
// Pour les variables images
union byte_bits Image_Feux, Image_Etat_Feux, Indicateurs;
#define Valeur_Feux Image_Feux.valeur // Pour un accès port complet
#define Veilleuse Image_Feux.bit.b0
#define Code Image_Feux.bit.b1
#define Phare Image_Feux.bit.b2
#define Clignot Image_Feux.bit.b3
#define S_Veilleuse Image_Etat_Feux.bit.b4 // Status veilleuse
#define S_Code Image_Etat_Feux.bit.b5
#define S_Phare Image_Etat_Feux.bit.b6
#define S_Clignot Image_Etat_Feux.bit.b7
// Pour les indicateurs divers
#define I_Att_Rep_Acquit Indicateurs.bit.b0
#define I_Fin_Tempo_Feux Indicateurs.bit.b1
#define I_Fin_Tempo_Clignot Indicateurs.bit.b2
#define I_Fin_Tempo_Control Indicateurs.bit.b3
#define I_Fin_Tempo_Att_Rep Indicateurs.bit.b4
#define I_Att_Rep_Interrog Indicateurs.bit.b5
#define I_Autorise_Emis_Mes Indicateurs.bit.b6
#define I_Message_Pb_Affiche Indicateurs.bit.b7
// Déclaration des trames
Trame Trame_Recue;
Trame Trame_Envoyee;
Trame T_IM_Feux; // Trame de type "Input Message" pour commande module 4 Sorties de puissance
Trame T_IRM_Feux; // Trame de type "Information Request Message" pour interroger les états lampes
// Pour la comparaison des identificateurs Trame envoyée <-> Trame reçue
#define Ident_Traine_Envoyee Trame_Envoyee.ident.extend.identificateur.ident
#define Ident_Traine_Recue Trame_Recue.ident.extend.identificateur.ident

// Pour les temporisations
WORD Compteur_Passage, Compteur_dS; // dS -> dixième de Seconde
WORD Valeur_Fin_Tempo_Feux, Valeur_Fin_Tempo_Clignot, Valeur_Fin_Tempo_Att_Rep;
// Pour le diagramme des états
unsigned char Etat;
// Pour controle communication
int Cptr_TimeOut, Temp;

```

```

// Fonction d'interruption "Base de Temps"
//=====
void irq_bt()
// Fonction exécutée toute les 10 mS
{Compteur_Passage++;
if(Compteur_Passage==10) // Un 1/10 Seconde s'est écoulée
{Compteur_Passage=0;
Compteur_dS++;
if(Compteur_dS==Valeur_Fin_Tempo_Feux)
{I_Fin_Tempo_Feux = 1;
Valeur_Fin_Tempo_Feux = Compteur_dS + Tempo_Feux;}
if(Compteur_dS==Valeur_Fin_Tempo_Clignot)
{I_Fin_Tempo_Clignot = 1;
Valeur_Fin_Tempo_Clignot = Compteur_dS + Tempo_Clignot;}
if(Compteur_dS==Valeur_Fin_Tempo_Att_Rep)
{I_Fin_Tempo_Att_Rep = 1;}
}
} // Fin de la fonction d'interruption

//=====
// FONCTION PRINCIPALE
//=====
main()
{
// Initialisations
//*****
clrscr();
// Définition des trames pour activer ou lire un bloc optique
// D'après doc SJA1000 et doc MCP25050 pages 22 (fonction "Write Register") et 37 (Adresse GPPIN)
// Pour trame de commande -> IM
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=Ident_T_IM_FVG; // l'identificateur Feux aVant Gauche
// Voir définitions dans CAN_VMD.h

// Pour trame interrogative -> IRM (Information Request Trame)
T_IRM_Feux.trame_info.registre=0x00;
T_IRM_Feux.trame_info.champ.extend=1;
T_IRM_Feux.trame_info.champ.dlc=0x01;
T_IRM_Feux.trame_info.champ.rtr=1;
T_IRM_Feux.ident.extend.identificateur.ident=Ident_T_IRM_FVG; // Voir définitions dans CAN_VMD.h

/* Initialisation DU SJA1000 de la carte ATON-Systemes" sur bus PC104*/
Init_Aton_CAN();
// Envoi trame pour définir de la direction des entrées sorties
T_IM_Feux.data[0]=0x1F; // première donnée -> "Adresse" du registre concernée -> GPDDR
T_IM_Feux.data[1]=0x7F; // deuxième donnée -> "Masque" -> Voir Doc page 16
T_IM_Feux.data[2]=0xF0; // troisième donnée -> "Valeur" -> les sorties sur 4 bits de poids faibles
I_Message_Pb_Affiche=0;
do {Ecrire_Traine(T_IM_Feux); // Envoyer une première trame sur réseau CAN
Cptr_TimeOut=0;
do{Cptr_TimeOut++;}while((Lire_Traine(&Traine_Recue)==0)&&(Cptr_TimeOut<200));
if(Ident_Traine_Recue!=Ident_T_AIM_FVG)Cptr_TimeOut=200; // Test si identificateur correct
if(Cptr_TimeOut==200)
{if(I_Message_Pb_Affiche==0)
{I_Message_Pb_Affiche=1;
gotoxy(2,10);
printf(" Pas de reponse a la trame de commande en initialisation \n");
printf(" Verifier si alimentation 12 V est OK \n");}
for(Temp=0;Temp<100000;Temp++;)} // Pour attendre un peu!
}while(Cptr_TimeOut==200);
clrscr();
// Pour l'ensemble des indicateurs
Indicateurs.valeur=0;
Etat = Etat_aucune;
Valeur_Feux=0x00;
// On envoie sur le bus la première trame -> état initial des sorties (qui sont initialisées à 0
// Initialiser les sorties à 0
T_IM_Feux.data[0]=0x1E; // première donnée -> "Adresse" du registre concernée (GPLAT définit l'état des sorties)
// 03h+1Ch = 1Eh
T_IM_Feux.data[1]=0x0F; // deuxième donnée -> "Masque" -> les sorties sont sur les 4 bits de poids faibles
T_IM_Feux.data[2]=0x00; // troisième donnée -> "Valeur" -> au départ toutes les sorties sont à 0 (lampes éteintes)

Ecrire_Traine(T_IM_Feux);
Traine_Envoyee = T_IM_Feux;
I_Att_Rep_Acquit=1;
// Pour base de temps et temporisations
//*****
SetVect(96,&irq_bt); // mise en place de l'autovecteur
PITR = 0x0048; // Une interruption toutes les 10 millisecondes
PICR = 0x0760; // 96 = 60H
Compteur_Passage = 0,Compteur_dS = 0;
Valeur_Fin_Tempo_Feux = Tempo_Feux;
Valeur_Fin_Tempo_Clignot = Tempo_Clignot;
Valeur_Fin_Tempo_Att_Rep = Tempo_Att_Rep;
I_Autorise_Emis_Mes=1;

```

```

// Afficher titre
gotoxy(1,1);
printf("    TP n° 3    ACTIVER FEUX ET CONTROLER ETAT AMPOULES    \n");
printf("    ***** \n");

// Boucle principale
//*****
while(1)
{
    if (l==Lire_Trame(&Trame_Recue)) //Si trame reçue, la fonction retourne 1
    { // On vient de recevoir une trame en réponse
        gotoxy(1,3); // Pour effacer message d'alerte de non reponse module adresse
        printf("                                     \n");
        if(I_Att_Rep_Acquit)
        { // On attendait une trame d'acquiescement suite à l'envoi d'une commande feux
            I_Att_Rep_Acquit=0;
            I_Autorise_Emis_Mes=1;
            gotoxy(1,8);
            printf("    Trame d'acquiescement suite à une 'IM' (Input Message)\n");
            Affiche_Trame(Trame_Recue);
            // On peut envoyer trame interrogative afin de vérifier l'état des feux
            Ecrire_Trame(T_IRM_Feux);
            Valeur_Fin_Tempo_Att_Rep = Compteur_dS+Tempo_Att_Rep;
            I_Fin_Tempo_Att_Rep=0;
            gotoxy(1,12);
            printf("    Trame interrogative envoyée -> 'IRM' (Information Request Message) \n");
            printf("    En vue de tester le bon fonctionnement des feux \n");
            Trame_Envoyee = T_IRM_Feux;
            Affiche_Trame(Trame_Envoyee);
            I_Att_Rep_Interrog=1;
        }
        else if(I_Att_Rep_Interrog)
        { // On attendait une trame de réponse à une interrogation
            I_Att_Rep_Interrog=0;
            I_Autorise_Emis_Mes=1;
            gotoxy(1,16);
            printf("    Trame réponse à une interrogation -> 'OM' (Output Message) \n");
            Affiche_Trame(Trame_Recue);

            // Analyse de l'état des ampoules et affichage résultat diagnostique
            Image_Etat_Feux.valeur=Trame_Recue.data[0];
            if(Veilleuse==1 && S_Veilleuse==0)
            {gotoxy(1,20),printf("!!    Probleme sur Veilleuse    \n");}
            if(Veilleuse==1 && S_Veilleuse==1)
            {gotoxy(1,20),printf("!!    Veilleuse    OK    \n");}
            if(Code==1 && S_Code==0)
            {gotoxy(1,21),printf("!!    Probleme sur Code    \n");}
            if(Code==1 && S_Code==1)
            {gotoxy(1,21),printf("!!    Code    OK    \n");}
            if(Phare==1 && S_Phare==0)
            {gotoxy(1,22),printf("!!    Probleme sur Phare    \n");}
            if(Phare==1 && S_Phare==1)
            {gotoxy(1,22),printf("!!    Phare    OK    \n");}
            if(Clignot==1 && S_Clignot==0)
            {gotoxy(1,23),printf("!!    Probleme sur Clignotant    \n");}
            if(Clignot==1 && S_Clignot==1)
            {gotoxy(1,23),printf("!!    Clignotant    OK    \n");}
        }
    }
    if(I_Fin_Tempo_Feux)
    {I_Fin_Tempo_Feux=0;
        // On passe à l'état suivant
        switch(Etat)
        {case Etat_aucune :      {Etat=Etat_veilleuse;
                                Veilleuse =1; }
          break;
          case Etat_veilleuse : {Etat=Etat_code;
                                Code=1;}
          break;
          case Etat_code :      {Etat=Etat_phare;
                                Code=0,Phare=1;}
          break;
          case Etat_phare :      {Etat=Etat_aucune;
                                Veilleuse=0,Phare=0;}
          break;}
        // On envoie la trame de commande avec les états mis à jour
        T_IM_Feux.data[2]=Valeur_Feux;
        if(I_Autorise_Emis_Mes)
        {Ecrire_Trame(T_IM_Feux); // Envoyer trame de commande sur réseau CAN
            gotoxy(1,5);
            printf("    Trame de commande -> 'IM' (Input Message) \n");
            Trame_Envoyee = T_IM_Feux;
            Affiche_Trame(Trame_Envoyee);
            I_Att_Rep_Acquit=1;
            I_Autorise_Emis_Mes=0;
            Valeur_Fin_Tempo_Att_Rep = Compteur_dS+Tempo_Att_Rep;
            I_Fin_Tempo_Att_Rep=0;}}
        else if(I_Fin_Tempo_Clignot)

```

```
{
    I_Fin_Tempo_Clignot=0;
    // On change l'état du clignotant
    Clignot=~Clignot;
    // On envoie la trame de commande
    T_IM_Feux.data[2]=Valeur_Feux;
    if(I_Autorise_Emis_Mes)
        {Ecrire_Trame(T_IM_Feux); // Envoyer trame de commande sur réseau CAN
        Trame_Envoyee = T_IM_Feux;
        I_Att_Rep_Acquit=1;
        I_Autorise_Emis_Mes=0;
        Valeur_Fin_Tempo_Att_Rep = Compteur_dS+Tempo_Att_Rep;
        I_Fin_Tempo_Att_Rep=0;}}
else if(I_Fin_Tempo_Att_Rep)
    // Cela fait trop longtemps que l'on attend une reponse!
    {clrscr();
    gotoxy(1,1);
    printf("      TP n° 3      ACTIVER FEUX ET CONTROLER ETAT AMPOULES   \n");
    printf("      ***** \n");
    printf("!!          Le module adresse ne repond plus          !!\n");
    Ecrire_Trame(T_IRM_Feux); // On réitère une interrogation
    Valeur_Fin_Tempo_Att_Rep = Compteur_dS+Tempo_Att_Rep; // On réarme la temporisation
    I_Fin_Tempo_Att_Rep=0; }
} // FIN de la boucle principale
} // FIN de la fonction principale
```

## 4 TP N°4: COMMANDER FEUX A PARTIR DU COMMODO FEUX

### 4.1 Sujet

<b>Objectifs :</b>	<ul style="list-style-type: none"> <li>- Réaliser une application de contrôle commande d'un système pilotable par réseau CAN.</li> <li>- Visualiser sur l'écran l'état du système.</li> <li>- Réaliser des temporisations précises.</li> <li>- Réaliser des tâches de vérification et de contrôle.</li> </ul>
<b>Cahier des charges :</b>	<p>A intervalles de temps réguliers, on interroge le module sur lequel est connecté le commodo lumières afin de connaître son état. En fonction de l'état du commodo lumière, on active les différentes lampes des blocs optiques avant et arrière.</p> <p>→ La temporisation nécessaire au fonctionnement des clignotants est réalisée par "Timer programmable" (interne au micro-processeur).</p> <p>→ Les différentes commandes imposées par la position de la manette commodo seront affichées individuellement.</p> <p>→ On contrôle le bon fonctionnement des ampoules des différents blocs optiques.</p>

Matériels et logiciels nécessaires :

Micro ordinateur de type PC sous Windows 95 ou ultérieur

Logiciel Editeur-Assembleur-Debugueur

Si programmation en C , Compilateur GNU C/C++ Réf : EID210100

Carte processeur 16/32 bits à microcontrôleur 68332 et son environnement logiciel  
(Editeur-CrossAssembleur-Debugueur) Réf : EID210000

Carte réseau CAN PC/104 de chez ATON SYSTEMES Réf : EID004000

Réseau CAN avec:

- 1 module 8 Entrées logiques destiné au commodo Réf : EID050000

- 4 modules 4S de puissance destinés aux feux arrière gauche, droit et aux feux avant gauche et droit Réf : EID051000

Câble de liaison USB, ou à défaut câble RS232, Réf : EGD000003

Alimentation AC/AC 8V, 1A pour l'alimentation de l'unité centrale Réf : EGD000001,

Alimentation 12V pour l'alimentation des modules CAN (réseau "énergie").

Durée : 2x4 heures

## 4.2 Eléments de solution

### 4.2.1 Analyse

#### Tâches à réaliser

##### Tâche principale

On envisage un fonctionnement cyclique où l'état du commodo est demandé à intervalles de temps réguliers, imposés par une base de temps. L'état recueilli du commodo est alors comparé à l'état précédent (recueilli le coup d'avant). Si un changement de position a été détecté, une phase de changement des états feux débute alors (on commande successivement les 4 blocs optiques avec les nouvelles valeurs)

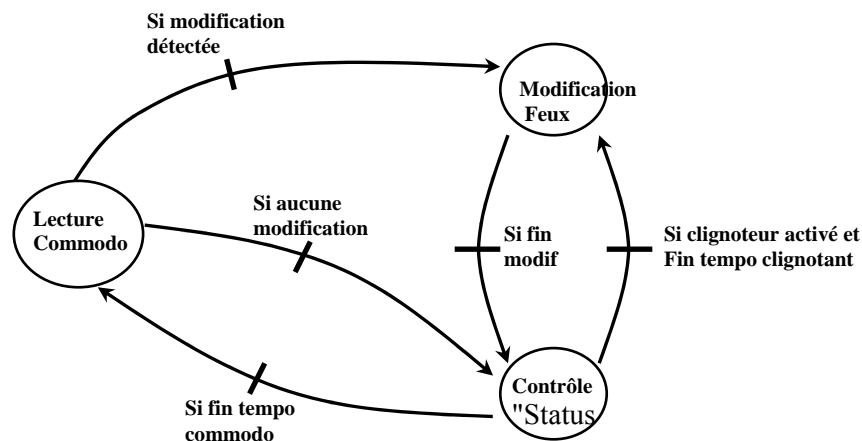
##### Tâches secondaires

→ On interroge successivement les 4 blocs optiques et on vérifie si les valeurs des "Status" corroborent les commandes envoyées. Si une différence est détectée, un message d'alerte est affiché.

→ Suite à l'envoi d'une trame, on vérifiera que la trame reçue en réponse est correcte (acquittement du module ayant reçu une trame de commande, ou réponse cohérente du module ayant reçu une trame de interrogative).

#### Diagramme des états principaux

On peut matérialiser le fonctionnement global par le diagramme des états suivant:



##### Etat "Modification feux"

Il s'agit d'envoyer une trame de commande (trame de type IM) à chacun des blocs optiques (Voir TP n°1). On décide d'envoyer les trames de commande dans l'ordre suivant:

- Feux aVant Gauche (FVG),
- puis Feux aVant Droit (FVD),
- Feux aRrière Gauche (FRG),
- et enfin Feux aRrière Droit (FRD).

Seules deux informations sont à changer lorsque l'on passe d'un bloc optique à un autre:

- l'identificateur,
- le paramètre "Value".

##### Etat "Lecture commodo"

Il s'agit d'envoyer une trame interrogative (trame de type IRM) au module 8 entrées sur lequel est connecté le commodo (Voir TP n°2). L'analyse de la trame réponse et la comparaison avec l'état mis en mémoire permet de détecter un changement éventuel.

##### Etat "Contrôle status"

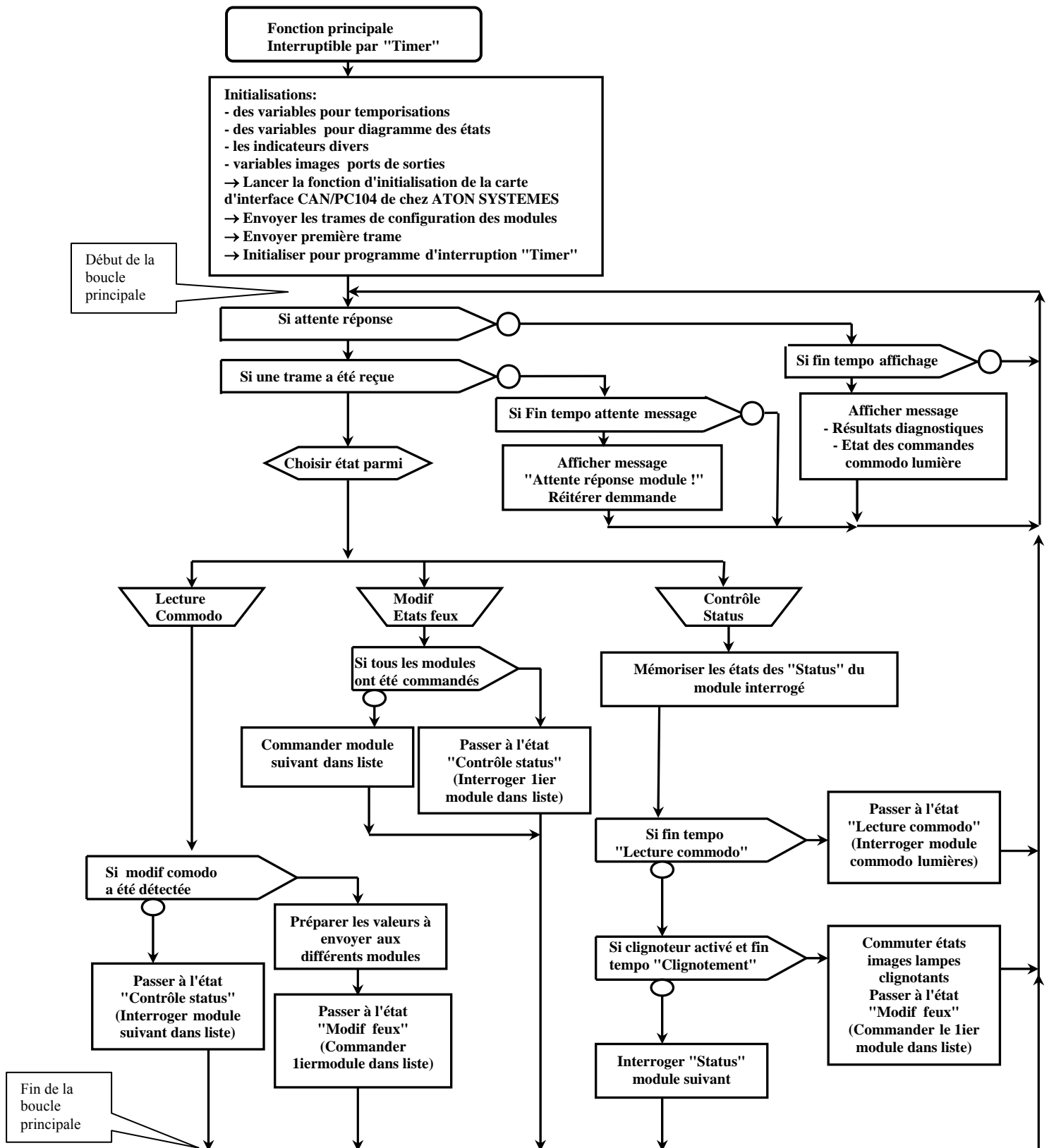
Il s'agit d'envoyer des trames interrogatives (trame de type IRM) aux différents modules 4 sorties de puissance sur lesquels sont connectés les blocs optiques (Voir TP n°3).



## 4.2.2 Organigrammes

Pour les indicateurs de fin de temporisation idem TP n°3

Ordinogramme général de la fonction principale



### 4.2.3 Programme en "C"

```

/*****
*      TP sur  EID210 / Réseau CAN - VMD  (Véhicule Multiplexé Didactique)
*      *****/
*      TP n 4:  Commander les feux à partir du commodo feux
*      -----
*      CAHIER DES CHARGES :
*      *****/
*      A intervalles de temps réguliers, on interroge le module sur lequel est connecté le commodo lumières
*      afin de connaître son état.
*      En fonction de l'état du commodo lumière, on active les différentes lampes des blocs optiques

*      avant et arrière.
*      La temporisation nécessaire au fonctionnement des clignotants est réalisée par "Timer programmable"
*      (interne au micro-processeur).
*      Les différentes commandes imposées par la position de la manette commodo seront affichées
*      individuellement.
*      On réalisera en plus des fonctions de controle:
*      - On vérifiera (par fonction logicielle) que les lampes commandées sont effectivement
*      allumées
*      - On affichera quelles sont les lampes concernées ainsi que le résultat du test.
*      - Le clignotement (du clignotant) sera indépendant de la permutation des autres lampes
*      ainsi que de la période de controle,
*      - On détectera si un module ne répond pas.
*      On impose les périodes suivantes, dans l'ordre de priorité décroissante:
*      - période de commutation du clignoteur 0,8 S,
*      - période de lecture commodo 0.2 S,
*      - temporisation de détection non réponse module 1S
*      Ces périodes devront pouvoir être modifiées facilement.
*      - Le programme devra permettre un changement aisé de bloc cible.
*      -----
*      NOM du FICHIER :  CAN_VMD_TP4.C
*      *****/
*****/

// Fichiers à inclure
//*****
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"
#include "Aton_can.h"
void Passer_a_Etat_Modif_Feux(void);
void Passer_a_Etat_Control_Stat(void);
// Déclarations des constantes
//-----
// Pour les temporisations
#define Tempo_Commodo 2           // En dixième de seconde -> 0,2 S
#define Tempo_Clignot 8           // En dixième de seconde -> 0,8 S
#define Tempo_Att_Rep 20          // Attente réponse en dixième de seconde -> 2 S
#define Tempo_Affichage 10        // Attente réponse en dixième de seconde -> 0,1 S
// Pour le codage des états

#define Etat_Lect_Commodo_Feux 0
#define Etat_Modif_Feux 1
#define Etat_Control_Stat 2

// Déclaration des variables
//-----
// Pour les indicateurs divers (variables binaires)
union word_bits Indicateurs;
#define I_Attente_Reponse Indicateurs.bit.b0
#define I_Fin_Tempo_Commodo Indicateurs.bit.b1
#define I_Fin_Tempo_Clignot Indicateurs.bit.b2
#define I_Fin_Tempo_Affichage Indicateurs.bit.b3
#define I_Fin_Tempo_Att_Rep Indicateurs.bit.b4
#define I_En_Att_Rep Indicateurs.bit.b5
#define I_Clignot_Gauche Indicateurs.bit.b6
#define I_Clignot_Droit Indicateurs.bit.b7
#define I_Message_Pb_Affiche Indicateurs.bit.b8
// Déclaration des trames
Trame Trame_Recue;
Trame Trame_Envoyee;
Trame T_IM; // Trame de type "Input Message" pour commande module 4 Sorties de puissance
Trame T_IRM; // Trame de type "Information Request Message" pour interroger "Status" lampes
// Ou commodo

// Pour la comparaison des identificateurs Trame envoyée <-> Trame reçue
#define Ident_Traine_Envoyee Traine_Envoyee.ident.extend.identificateur.ident
#define Ident_Traine_Recue Traine_Recue.ident.extend.identificateur.ident
#define Ident_T_IRM T_IRM.ident.extend.identificateur.ident
#define Ident_T_IM T_IM.ident.extend.identificateur.ident
#define Valeur_T_IM T_IM.data[2]
// Pour les temporisations
WORD Compteur_Passage, Compteur_dS; // dS -> dixième de Seconde
WORD Valeur_Fin_Tempo_Commodo, Valeur_Fin_Tempo_Clignot, Valeur_Fin_Tempo_Affichage, Valeur_Fin_Tempo_Att_Rep;
// Pour le diagramme des états
unsigned char Etat, Rang_Control_Stat, Rang_Modif_Feux;
// Pour la mémoire
unsigned char Valeur_Commodo_Feux_Mem;
// Fonction d'interruption "Base de Temps"
void irq_bt()
// Fonction exécutée toute les 10 mS
{Compteur_Passage++;
if(Compteur_Passage==10) // Une 1/2 Seconde s'est écoulée
{Compteur_Passage=0;
Compteur_dS++;
if(Compteur_dS==Valeur_Fin_Tempo_Commodo)
{I_Fin_Tempo_Commodo = 1;
Valeur_Fin_Tempo_Commodo = Compteur_dS + Tempo_Commodo;}
if(Compteur_dS==Valeur_Fin_Tempo_Affichage)
{I_Fin_Tempo_Affichage = 1;

```

```

        Valeur_Fin_Tempo_Affichage = Compteur_dS + Tempo_Affichage;}
    if(Compteur_dS==Valeur_Fin_Tempo_Clignot)
    { if(I_Clignot_Gauche||I_Clignot_Droit)
        {I_Fin_Tempo_Clignot = 1;
        Valeur_Fin_Tempo_Clignot = Compteur_dS + Tempo_Clignot;}
    }
    if(Compteur_dS==Valeur_Fin_Tempo_Att_Rep)
    {I_Fin_Tempo_Att_Rep = 1;}
}
} // Fin de la fonction d'interruption

//=====
// FONCTION PRINCIPALE
//=====
main()
{
    int Cptr_TimeOut,Temp;
    // Initialisations
    //*****
    clrscr();
    /* Initialisation DU SJA1000 de la carte ATON-Systemes" sur */
    Init_Aton_CAN();
    // Définition des trames pour activer ou lire un bloc optique
    // D'après doc SJA1000 et doc MCP25050 pages 22 (fonction "Write Register") et 37 (Adresse GPPIN)
    // Pour les trames de commande -> IM
    T_IM.trame_info.registre=0x00;
    T_IM.trame_info.champ.extend=1; // On travaille en mode étendu
    T_IM.trame_info.champ.dlc=0x03; // Il y aura 3 données de 8 bits (3 octets envoyés)
    Ident_T_IM=Ident_T_IM_FRD;// C'est l'identificateur du bloc optique arrière droit
    T_IM.data[0]=0x1F; // première donnée -> "Adresse" du registre concernée -> GPDDR
    T_IM.data[1]=0x7F; // deuxième donnée -> "Masque" -> Voir Doc MCP25050 page 16
    T_IM.data[2]=0xF0; // troisième donnée -> "Valeur" -> Les sorties sont les 4 bits de poids faibles
    //Configuration du registre de direction et
    //Verification de la presence des modules
    I_Message_Pb_Affiche=0;
    do {Ecrire_Traine(T_IM);// Envoyer une première trame sur réseau CAN
        Cptr_TimeOut=0;
        do{Cptr_TimeOut++;}while((Lire_Traine(&Traine_Recue)==0)&&(Cptr_TimeOut<200));
        if(Ident_Traine_Recue!=Ident_T_AIM_FRD)Cptr_TimeOut=200; // Test si identificateur correct
        if(Cptr_TimeOut==200)
            {if(I_Message_Pb_Affiche==0)
                {I_Message_Pb_Affiche=1;
                gotoxy(2,10);
                printf(" Pas de reponse du Feux aRriere Droit a la trame de commande en initialisation \n");
                printf(" Verifier si alimentation 12 V est OK \n");}
            for(Temp=0;Temp<100000;Temp++);} // Pour attendre un peu!
        }while(Cptr_TimeOut==200);
    Ident_T_IM=Ident_T_IM_FRG;// C'est l'identificateur du bloc optique arrière gauche
    I_Message_Pb_Affiche=0;
    do {Ecrire_Traine(T_IM);// Envoyer une première trame sur réseau CAN
        Cptr_TimeOut=0;
        do{Cptr_TimeOut++;}while((Lire_Traine(&Traine_Recue)==0)&&(Cptr_TimeOut<200));
        if(Ident_Traine_Recue!=Ident_T_AIM_FRG)Cptr_TimeOut=200; // Test si identificateur correct
        if(Cptr_TimeOut==200)
            {if(I_Message_Pb_Affiche==0)
                {I_Message_Pb_Affiche=1;
                gotoxy(2,10);
                printf(" Pas de reponse Feux aRriere Gauche a la trame de commande en initialisation \n");
                printf(" Verifier si alimentation 12 V est OK \n");}
            for(Temp=0;Temp<100000;Temp++);} // Pour attendre un peu!
        }while(Cptr_TimeOut==200);
    Ident_T_IM=Ident_T_IM_FVG;// C'est l'identificateur du bloc optique aVant gauche
    I_Message_Pb_Affiche=0;
    do {Ecrire_Traine(T_IM);// Envoyer une première trame sur réseau CAN
        Cptr_TimeOut=0;
        do{Cptr_TimeOut++;}while((Lire_Traine(&Traine_Recue)==0)&&(Cptr_TimeOut<200));
        if(Ident_Traine_Recue!=Ident_T_AIM_FVG)Cptr_TimeOut=200; // Test si identificateur correct
        if(Cptr_TimeOut==200)
            {if(I_Message_Pb_Affiche==0)
                {I_Message_Pb_Affiche=1;
                gotoxy(2,10);
                printf(" Pas de reponse Feux aVant Gauche a la trame de commande en initialisation \n");
                printf(" Verifier si alimentation 12 V est OK \n");}
            for(Temp=0;Temp<100000;Temp++);} // Pour attendre un peu!
        }while(Cptr_TimeOut==200);
    Ident_T_IM=Ident_T_IM_FVD;// C'est l'identificateur du bloc optique aVant Droit
    I_Message_Pb_Affiche=0;
    do {Ecrire_Traine(T_IM);// Envoyer une première trame sur réseau CAN
        Cptr_TimeOut=0;
        do{Cptr_TimeOut++;}while((Lire_Traine(&Traine_Recue)==0)&&(Cptr_TimeOut<200));
        if(Ident_Traine_Recue!=Ident_T_AIM_FVD)Cptr_TimeOut=200; // Test si identificateur correct
        if(Cptr_TimeOut==200)
            {if(I_Message_Pb_Affiche==0)
                {I_Message_Pb_Affiche=1;
                gotoxy(2,10);
                printf(" Pas de reponse Feux aVant Droit a la trame de commande en initialisation \n");
                printf(" Verifier si alimentation 12 V est OK \n");}
            for(Temp=0;Temp<100000;Temp++);} // Pour attendre un peu!
        }while(Cptr_TimeOut==200);
    Ident_T_IM=Ident_T_IM_Commodo_Feux;// C'est l'identificateur du bloc Commodo Lumiere
    T_IM.data[2]=0xFF; // troisième donnée -> "Valeur" -> 8 bits sont des entrées
    I_Message_Pb_Affiche=0;
    do {Ecrire_Traine(T_IM);// Envoyer une première trame sur réseau CAN
        Cptr_TimeOut=0;
        do{Cptr_TimeOut++;}while((Lire_Traine(&Traine_Recue)==0)&&(Cptr_TimeOut<200));
        if(Ident_Traine_Recue!=Ident_T_AIM_Commodo_Feux)Cptr_TimeOut=200; // Test si identificateur correct
        if(Cptr_TimeOut==200)
            {if(I_Message_Pb_Affiche==0)
                {I_Message_Pb_Affiche=1;
                gotoxy(2,10);
                printf(" Pas de reponse du Commodo Lumiere a la trame de commande en initialisation \n");
                printf(" Verifier si alimentation 12 V est OK \n");}
            for(Temp=0;Temp<100000;Temp++);} // Pour attendre un peu!
        }while(Cptr_TimeOut==200);
}

```

```

classcr();

// Initialiser les sorties à 0 (registre GPLat)
T_IM.data[0]=0x1E; // première donnée -> "Adresse" du registre concernée (GPLAT définit l'état des sorties) 03h+1Ch = 1Eh
T_IM.data[1]=0x0F; // deuxième donnée -> "Masque" -> les sorties sont sur les 4 bits de poids faibles
T_IM.data[2]=0x00; // troisième donnée -> "Valeur" -> au départ toutes les sorties sont à 0 (lampes éteintes)
// Pour les trames interrogative -> IRM (Information Request Trame)
T_IRM.trame_info.registre=0x00;
T_IRM.trame_info.champ.extend=1;
T_IRM.trame_info.champ.dlc=0x01;
T_IRM.trame_info.champ.rtr=1;
Ident_T_IRM=Ident_T_IRM_Commodo_Feux; // On commence par lire le commodo
Etat = Etat_Lect_Commodo_Feux;
Valeur_Commodo_Feux_Mem=0;
// Et on envoie sur le bus la première trame
//Ecrire_Traine(T_IRM);
//Trame_Envoyee = T_IRM;
//I_Attese_Reponse=1;
// Pour l'état "Control Status"
Rang_Control_Stat=0;
// Pour l'état "Modif Feux"
Rang_Modif_Feux=0;
// Pour l'ensemble des indicateurs
Indicateurs.valeur=0;
// On envoie sur le bus la première trame
Ecrire_Traine(T_IRM);
while((Lire_Traine(&Traine_Recue)==0));
Traine_Envoyee = T_IRM;
I_Attese_Reponse=1;
// Pour base de temps et temporisations
//*****
SetVect(96,&irq_bt); // mise en place de l'autovecteur
PITR = 0x0048; // Une interruption toutes les 10 millisecondes
PICR = 0x0760; // 96 = 60H
// Pour les temporisations
Compteur_Passage = 0,Compteur_dS = 0;
Valeur_Fin_Tempo_Clignot = Tempo_Clignot;
Valeur_Fin_Tempo_Affichage = Tempo_Affichage;
Valeur_Fin_Tempo_Commodo = Tempo_Commodo;
Valeur_Fin_Tempo_Att_Rep = Tempo_Att_Rep;
// Afficher titre
gotoxy(1,2);
printf(" TP n: 4 ACTIVER FEUX AVEC COMMODO ET CONTROLER ETAT AMPOULES \n");
printf(" ***** \n");
// Boucle principale
//*****
while(1) {if(I_Attese_Reponse) // Une trame est attendue
{
if (l=Lire_Traine(&Traine_Recue)) //Si trame reçue, la fonction retourne 1
// Une Traine a été reçue
{I_Attese_Reponse=0;
I_Fin_Tempo_Att_Rep=0; // On réarme la tempo attente réponse
Valeur_Fin_Tempo_Att_Rep = Compteur_dS + Tempo_Att_Rep;
I_En_Att_Rep=0;
if(Etat==Etat_Lect_Commodo_Feux)
// On est dans l'état "Lecture Commodo"
{Valeur_Commodo_Feux=~(Traine_Recue.data[0]); //On récupère l'état du commodo
if(Valeur_Commodo_Feux!= Valeur_Commodo_Feux_Mem)
// On a détecté une modification de l'état commodo
{Valeur_Commodo_Feux_Mem = Etat_Commodo_Feux.valeur; // On met en mémoire
// On prédéfinit l'état des différentes ampoules
// Combinaisons définies dans CAN_VMD.h
Valeur_FVG=Cde_Nulle,Valeur_FVD=Cde_Nulle;
Valeur_FRG=Cde_Nulle,Valeur_FRD=Cde_Nulle;
I_Clignot_Droit=0;
I_Clignot_Gauche=0;
if(Cde_Phare) // Si Commande Phare
{Valeur_FVG=Cde_FV_P,Valeur_FVD=Cde_FV_P; // Les feux aVant
Valeur_FRG=Cde_FR_P,Valeur_FRD=Cde_FR_P; // Les feux aRrière
else if(Cde_Code) // Si Commande Code
{Valeur_FVG=Cde_FV_C,Valeur_FVD=Cde_FV_C; // Les feux aVant
Valeur_FRG=Cde_FR_C,Valeur_FRD=Cde_FR_C; // Les feux aRrière
else if(Cde_Veilleuse) // Si commande Veilleuse
{Valeur_FVG=Cde_FV_V,Valeur_FVD=Cde_FV_V; // Les feux aVant
Valeur_FRG=Cde_FR_V,Valeur_FRD=Cde_FR_V; // Les feux aRrière
if(Cde_Clign_Droit)
{Valeur_FVD|=Masque_Clign_AV;
Valeur_FRD|=Masque_Clign_AR;
I_Clignot_Droit=1;
Valeur_Fin_Tempo_Clignot = Compteur_dS + Tempo_Clignot;
I_Fin_Tempo_Clignot = 0;} // Pour Init tempo_clignot
if(Cde_Clign_Gauche)
{Valeur_FVG|=Masque_Clign_AV;
Valeur_FRG|=Masque_Clign_AR;
I_Clignot_Gauche=1;
Valeur_Fin_Tempo_Clignot = Compteur_dS + Tempo_Clignot;
I_Fin_Tempo_Clignot = 0;} // Pour Init tempo_clignot
if(Cde_Klaxon)
{Valeur_FRG|=Masque_Klaxon;
Valeur_FRD|=Masque_Klaxon;}
if(Cde_Stop)
{Valeur_FRG|=Masque_Stop;
Valeur_FRD|=Masque_Stop;}
Passer_a_Etat_Modif_Feux(); // Fin si action commodo détecté
Passer_a_Etat_Control_Stat(); // FIN bloc si "Etat lecture commodo"
}
else if(Etat==Etat_Modif_Feux)
{// On est dans Etat "Modif feux"
//On prépare la trame pour commande feux suivant
switch(Rang_Modif_Feux) // Module suivant dans la liste
{// Le Feux aVant gauche a déjà été commandé
case 1 : {Ident_T_IM=Ident_T_IM_FVD; //Feux aVant Droit
Valeur_T_IM=Valeur_FVD;
Rang_Modif_Feux++;
Ecrire_Traine(T_IM); // Envoyer traine

```

```

Trame_Envoyee =T_IRM;
I_Attente_Reponse=1;}

break;
case 2 : {Ident_T_IM=Ident_T_IM_FRD; //Feux aRrière Droit
Valeur_T_IM=Valeur_FRD;
Rang_Modif_Feux++;
Ecrire_Trame(T_IRM); // Envoyer trame
Trame_Envoyee =T_IRM;
I_Attente_Reponse=1;}

break;
case 3 : {Ident_T_IM=Ident_T_IM_FRG; //Feux aRrière Gauche
Valeur_T_IM=Valeur_FRG;
Rang_Modif_Feux++;
Ecrire_Trame(T_IRM); // Envoyer trame
Trame_Envoyee =T_IRM;
I_Attente_Reponse=1;}

}

break;
default : // On est arrivé à la fin de la modif feux
// On retourne au controle status
{Passer_a_Etat_Control_Stat();}

break;
} // FIN bloc "Etat modif feux"
else if(Etat==Etat_Control_Stat)
{while(I_Fin_Tempo_Affichage==0){};
//I_Fin_Tempo_Affichage=0;
switch(Rang_Control_Stat) // Module suivant dans la liste
{case 0 : // C'est le controle du Feux aVant Gauche
Valeur_Status_FVG=Trame_Recue.data[0];
// On passe (peut être) au feu suivant
Ident_T_IRM=Ident_T_IRM_FVD; //Feux aVant Droit

break;
case 1 : // C'est le controle du Feux aVant Droit
Valeur_Status_FVD=Trame_Recue.data[0];
// On passe (peut être) au feu suivant
Ident_T_IRM=Ident_T_IRM_FRD; //Feux aRrière Droit

break;
case 2 : // C'est le controle du Feux aRrière Droit
Valeur_Status_FRD=Trame_Recue.data[0];
// On passe (peut être) au feu suivant
Ident_T_IRM=Ident_T_IRM_FRG; //Feux aRrière Gauche

break;
case 3 : // C'est le controle du Feux aRrière Gauche
Valeur_Status_FRG=Trame_Recue.data[0];
// On passe (peut être) au feu suivant
Ident_T_IRM=Ident_T_IRM_FVG; //Feux aVant Gauche

break;
}
// On teste s'il est temps d'aller lire le commodo
if(I_Fin_Tempo_Commodo)
{// Passer à l'état "Lecture Commodo"
I_Fin_Tempo_Commodo=0;
Etat = Etat_Lect_Commodo_Feux;
Ident_T_IRM=Ident_T_IRM_Commodo_Feux;
// On envoie sur le bus la première trame
Ecrire_Trame(T_IRM);
Trame_Envoyee = T_IRM;
I_Attente_Reponse=1;}
// On teste si clignotants activés ET si fin tempo clignotant
else if((I_Clignot_Gauche|I_Clignot_Droit)&&I_Fin_Tempo_Clignot)
{I_Fin_Tempo_Clignot=0;
if(I_Clignot_Gauche)
{// Commuter ampoules clignotants gauche
Valeur_FVG^=Masque_Clign_AV;
Valeur_FRG^=Masque_Clign_AR;
Passer_a_Etat_Modif_Feux();}
if(I_Clignot_Droit)
{// Commuter ampoules clignotants gauche
Valeur_FVD^=Masque_Clign_AV;
Valeur_FRD^=Masque_Clign_AR;
Passer_a_Etat_Modif_Feux();}
} // Fin prise compte clignotant
// On continue le controle des Feux
else
{if(Rang_Control_Stat==3)Rang_Control_Stat=0;
else Rang_Control_Stat++; // Passage au feu suivant
// On envoie la trame interrogative sur le bus
Ecrire_Trame(T_IRM);
Trame_Envoyee =T_IRM;
I_Attente_Reponse=1;}
} // FIN bloc "Etat Control Status"

} // Fin Si trame reçue
else if (I_Fin_Tempo_Att_Rep) // On attend depuis trop longtemps une réponse !
{clrscr();
gotoxy(1,2); // On réaffiche le titre
printf(" TP n: 4 ACTIVER FEUX AVEC COMMODO FEUX ET CONTROLER ETAT AMPOULES \n");
printf(" ***** \n");
I_Fin_Tempo_Att_Rep=0; // On réarme la tempo attente réponse
Valeur_Fin_Tempo_Att_Rep = Compteur_ds + Tempo_Att_Rep;
gotoxy(1,4),printf(" !! Attente reponse modules !! \n");
gotoxy(1,9),printf(" \n");
gotoxy(1,20),printf(" \n");
gotoxy(1,16),printf(" \n");
I_En_Att_Rep=1;
// On refait une tentative d'interrogation commodo
Etat = Etat_Lect_Commodo_Feux;
Ident_T_IRM=Ident_T_IRM_Commodo_Feux;
// On envoie sur le bus la première trame
Ecrire_Trame(T_IRM);
Trame_Envoyee = T_IRM;
I_Attente_Reponse=1;
}
//On refait une tentative d'interrogation du commodo
if(I_Fin_Tempo_Affichage)

```

```

{I_Fin_Tempo_Affichage=0;
    if(I_En_Att_Rep==0) // Alors on peut afficher
    // Résultat diagnostic Feux aVant Gauche
    gotoxy(1,4),printf("Bloc optique avant gauche: \n");
    if(Veilleuse_FVG==1 && S_Veilleuse_FVG==0){gotoxy(1,5),printf("!! Probleme sur Veilleuse avant gauche \n");}
    if(Veilleuse_FVG==0 && S_Veilleuse_FVG==1){gotoxy(1,5),printf(" \n");}
    if(Code_FVG==1 && S_Code_FVG==0){gotoxy(1,6),printf("!! Probleme sur Code avant gauche \n");}
    if(Code_FVG==0 && S_Code_FVG==1){gotoxy(1,6),printf(" \n");}
    if(Phare_FVG==1 && S_Phare_FVG==0){gotoxy(1,7),printf("!! Probleme sur Phare avant gauche \n");}
    if(Phare_FVG==0 && S_Phare_FVG==1){gotoxy(1,7),printf(" \n");}
    if(Clignot_FVG==1 && S_Clignot_FVG==0){gotoxy(1,8),printf("!! Probleme sur Clignotant avant gauche \n");}
    if(Clignot_FVG==0 && S_Clignot_FVG==1){gotoxy(1,8),printf(" \n");}
    // Résultat diagnostic Feux aVant Droit
    gotoxy(1,9),printf("Bloc optique avant Droit:\n");
    if(Veilleuse_FVD==1 && S_Veilleuse_FVD==0){gotoxy(1,10),printf("!! Probleme sur Veilleuse avant droit \n");}
    if(Veilleuse_FVD==0 && S_Veilleuse_FVD==1){gotoxy(1,10),printf(" \n");}
    if(Code_FVD==1 && S_Code_FVD==0){gotoxy(1,11),printf("!! Probleme sur Code avant droit \n");}
    if(Code_FVD==0 && S_Code_FVD==1){gotoxy(1,11),printf(" \n");}
    if(Phare_FVD==1 && S_Phare_FVD==0){gotoxy(1,12),printf("!! Probleme sur Phare avant droit \n");}
    if(Phare_FVD==0 && S_Phare_FVD==1){gotoxy(1,12),printf(" \n");}
    if(Clignot_FVD==1 && S_Clignot_FVD==0){gotoxy(1,13),printf("!! Probleme sur Clignotant avant droit \n");}
    if(Clignot_FVD==0 && S_Clignot_FVD==1){gotoxy(1,13),printf(" \n");}
    // Résultat diagnostic Feux aRriere Droit
    gotoxy(1,20),printf("Feux arriere droit:\n");
    if(Veilleuse_FRD==1 && S_Veilleuse_FRD==0){gotoxy(1,21),printf("!! Probleme sur Lampe Arriere droit \n");}
    if(Veilleuse_FRD==0 && S_Veilleuse_FRD==1){gotoxy(1,21),printf(" \n");}
    if(Clignot_FRD==1 && S_Clignot_FRD==0){gotoxy(1,22),printf("!! Probleme sur Clignotant Arriere droit \n");}
    if(Clignot_FRD==0 && S_Clignot_FRD==1){gotoxy(1,22),printf(" \n");}
    if(Stop_FRD==1 && S_Stop_FRD==0){gotoxy(1,23),printf("!! Probleme sur Stop Arriere droit \n");}
    if(Stop_FRD==0 && S_Stop_FRD==1){gotoxy(1,23),printf(" \n");}
    // Résultat diagnostic Feux aVant Gauche
    gotoxy(1,16),printf("Feux arriere gauche:\n");
    if(Veilleuse_FRG==1 && S_Veilleuse_FRG==0){gotoxy(1,17),printf("!! Probleme sur Lampe Arriere Gauche \n");}
    if(Veilleuse_FRG==0 && S_Veilleuse_FRG==1){gotoxy(1,17),printf(" \n");}
    if(Clignot_FRG==1 && S_Clignot_FRG==0){gotoxy(1,18),printf("!! Probleme sur Clignotant Arriere Gauche \n");}
    if(Clignot_FRG==0 && S_Clignot_FRG==1){gotoxy(1,18),printf(" \n");}
    if(Stop_FRG==1 && S_Stop_FRG==0){gotoxy(1,19),printf("!! Probleme sur Stop Arriere Gauche \n");}
    if(Stop_FRG==0 && S_Stop_FRG==1){gotoxy(1,19),printf(" \n");}
    if(Klaxon_FRG==1 && S_Klaxon_FRG==0){gotoxy(1,19),printf("!! Probleme sur Klaxon Arriere Gauche \n");}
    if(Klaxon_FRG==0 && S_Klaxon_FRG==1){gotoxy(1,19),printf(" \n");}
    // Pour l'état commodo
    gotoxy(4,24);
    printf("Etat des differentes entrees imposees par le commodo:\n");
    printf(" Veilleuse=%d , Code=%d , Phare=%d , clign. G.=%d \n",Cde_Veilleuse,Cde_Code,Cde_Phare,Cde_Clign_Gauche);
    printf(" Klaxon=%d , Feux de stop=%d , clign. D.= %d\n",Cde_Klaxon,Cde_Stop,Cde_Clign_Droit);
    } // Fin si on n'est pas en attente de réponse après message d'alerte
    } // Fin if fin tempo affichage
}
} // FIN de la boucle principale
} // FIN de la fonction principale

// Fonction "Passer à l'état CONTROL STATUS"
void Passer_a_Etat_Control_Stat(void)
{Etat=Etat_Control_Stat; // On retourne à l'état "Control des status"
    // Pour préparer la trame interrogative pour module suivant dans la liste
    switch(Rang_Control_Stat) // Module suivant dans la liste
    {case 0 : Ident_T_IRM=Ident_T_IRM_FVG; //Feux aVant Gauche
        break;
        case 1 : Ident_T_IRM=Ident_T_IRM_FVD; //Feux aVant Droit
        break;
        case 2 : Ident_T_IRM=Ident_T_IRM_FRD; //Feux aRrière Droit
        break;
        case 3 : Ident_T_IRM=Ident_T_IRM_FRG; //Feux aRrière Gauche
        break;
        default : {Rang_Control_Stat=0;
                    Ident_T_IRM=Ident_T_IRM_FVG;} //Feux aVant Gauche
        break;
    }
    // On envoie la trame interrogative sur le bus
    Ecrire_Trame(T_IRM);
    Trame_Envoyee =T_IRM;
    I_Attente_Reponse=1;
} // FIN fonction "Paaser à Etat control Status"

// Fonction "Passer à l'Etat Modification Feux"
void Passer_a_Etat_Modif_Feux(void)
{WORD I_TEMP;
    Etat=Etat_Modif_Feux; // Passer à l'état "Modif feux"
    //On prépare la trame pour commander premier feux
    Ident_T_IM=Ident_T_IM_FVG; // Feux aVant Gauche: Premier dans la liste
    Valeur_T_IM=Valeur_FVG;
    Rang_Modif_Feux=1;
    // On envoie la trame de commande sur le bus
    Ecrire_Trame(T_IM);
    I_Attente_Reponse=1;
} // FIN Fonction "Passer à l'Etat Modification Feux"

// Fin du fichier source C

```