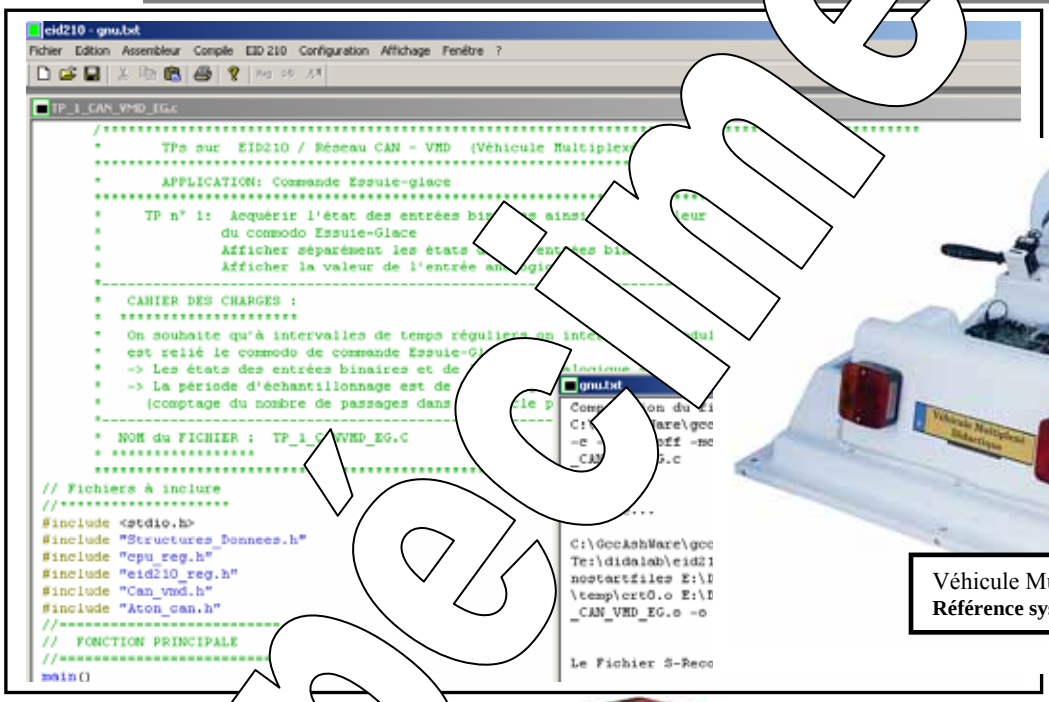
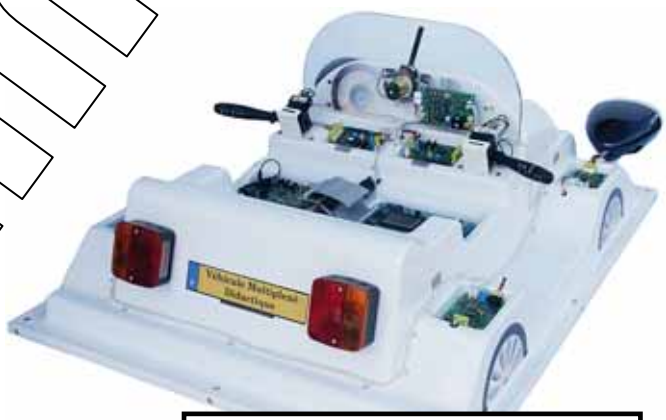
MANUEL
de
TRAVAUX PRATIQUES

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





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

Logiciels associés
- Environnement de développement intégré (Editeur, assembleur, loadeur) Réf: EID210
- Compilateur 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 Réef: EID210040
-Sur la carte simulateur E/S EID210000 seule Réef: EID211040
- avec bus CAN-VMD et noyau temps réel MTR86 Réf: EID050 050
- avec carte réseau ethernet Réf: EID213 040

didalab
ENSEIGNEMENT SUPERIEUR

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 Web : www.didalab.fr

Version du : 20/07/2010 Référence document : EID050040

SOMMAIRE

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

1.1 Sujet

Objectifs :	- Comprendre et utiliser les structures de données spécifiques proposées, - Comprendre et utiliser les fonctions spécifiques proposées. - Définir puis envoyer une trame de données à un module CAN destinataire, accessible une adresse donnée. - Tester si une trame a été reçue. - Visualiser sur l'écran les trames reçues ainsi que les trame envoyées.
Cahier des charges :	Au départ toutes les lampes "Feux droit" sont éteintes. On souhaite réaliser la fonction "clignotant" avec les 4 lampes du bloc (à intervalles de temps réguliers éteindre la lampe précédemment allumée et on allume la suivante). → Les trames reçues ou envoyées sur le bus CAN sont affichées. → La temporisation est de type "logiciel" (comptage du nombre de passages dans la boucle principale) 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".

Matériels et logiciels nécessaires

Micro ordinateur de type PC sous Windows 95 ou ultérieur
Logiciel Editeur-Compilateur-Debogueur
Si programmation en C , compilateur GNU C/C++ Réf : EID210100
Carte processeur 32 bits à microcontrôleur 68332 et son environnement logiciel (CrossAssembleur-Debogueur) Réf : EID210000
Interface réseau CAN PC/ISA chez ATON SYSTEMES Réf : EID004000
Modules interfaces de puissance destiné aux feux Réf : EID051000
Câble de liaison USB ou à défaut câble RS232, Réf : EGD000003
Alimentation AC 220V, 5V, 1A Réf : EGD000001,
Alimentation 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 identificateurs 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) définition des différents champs de la trame destinée à cet exemple.

Remarques

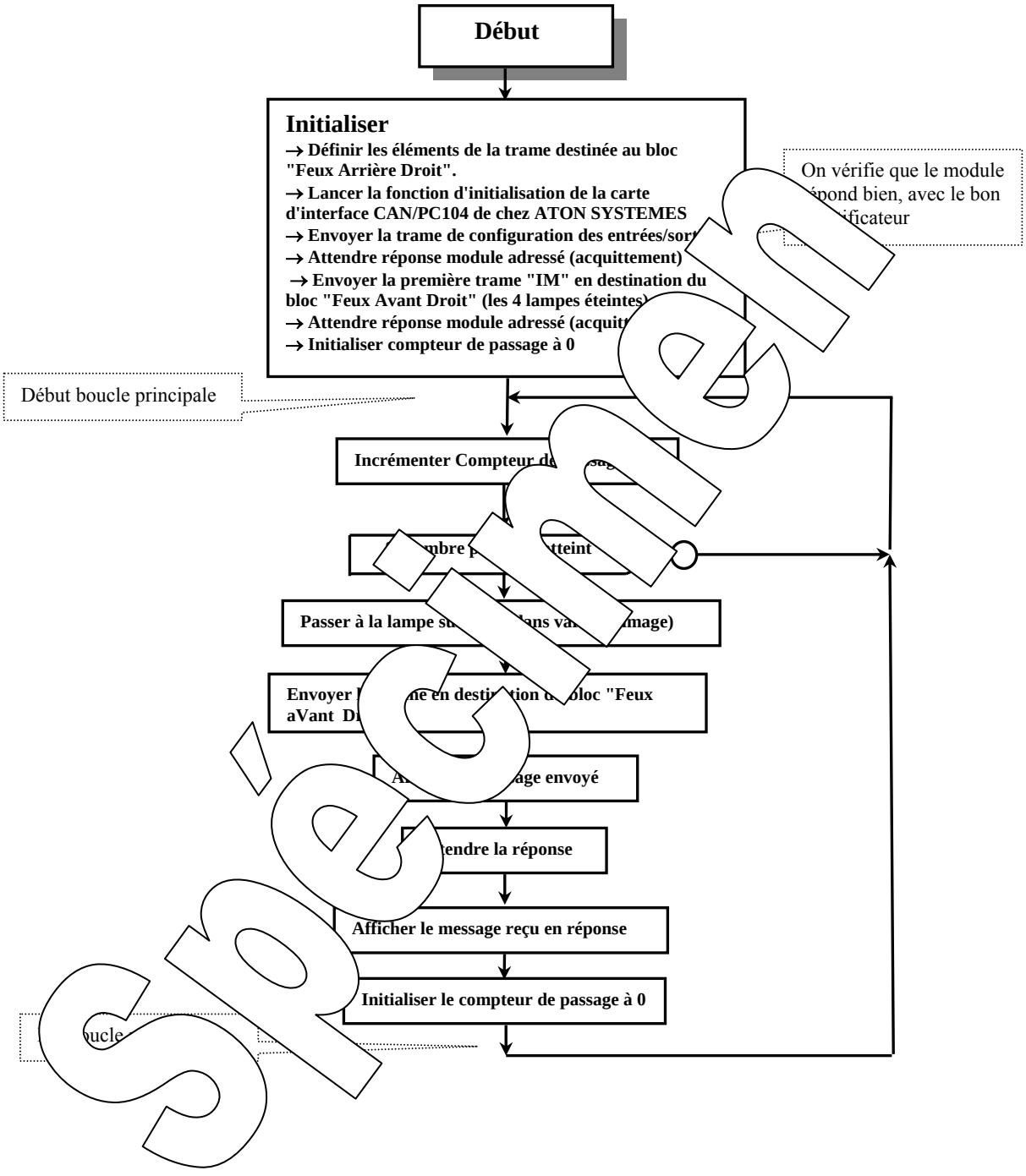
- Dans la trame de commande, il y a trois paramètres à définir dans l'octet "data" de la trame:
  - → Paramètre "addr" (au rang 0 des "data") qui définit le registre concerné par l'écriture.
    Dans le cas présent, c'est GPLAT et son adresse est 09h (page 15 notice MCP25050).
    02h+décalage (note1) soit 0Bh ou 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 enfin 08.
- Suite à un IM "Input Message" si celui-ci a été bien reçu par son destinataire, celui-ci doit renvoyer une trame d'acquittement dont l'identificateur est défini par TXID1. Dans le cas du module « feux avant droit », l'identificateur du message d'acquittement sera 0E A0 00 00 dont le label "T_Ident_AIM_FVD a été définui dans le fichier CAN_VMD.h (d'après tableau chapitre 1).

Il faut veiller à n'envoyer une trame de commande à un nœud que s'il a répondu par une trame d'acquittement IM à la première.

Chenillard sur autres feux

Il faut quatre choses à définir ce sont les identificateurs :
Feux avant gauche
Identificateur message écriture sur port sortie (registre RXF1): 0E080000 -> T_Ident_IM_FVG
Identificateur message d'acquittement (registre TXD1): 0E200000 -> T_Ident_AIM_FVG
Pour Feux avant droit
Identificateur message écriture sur port sortie (registre RXF1): 0F880000 -> T_Ident_IM_FRD
Identificateur message d'acquittement (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'acquittement (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                    issance
int Compteur_Passage,Cptr_TimeOut;
char I_Message_Pb_Affiche;
// Initialisations
//*****************
clsscr();
/* Initialisation DU SJA1000 de la carte ATON-Systemes" sur  bu
Init_Aton_CAN();
//  Définition trame pour activer bloc optique arrière droit
// D'aprés doc SJA1000 et  doc MCP25050 pages 22 (fonction                  egiste              (Adresse GPLAT)
T_IM_Feux.trame_info.registre=0x00;
T_IM_Feux.trame_info.champ.extend=1; // On travaille en mo
T_IM_Feux.trame_info.champ.dlc=0x03; // Il y aura 3 don                  octets envoyés)
T_IM_Feux.ident.extend.identificateur.ident=  0F880000                  r du bloc optique arrière droit
T_IM_Feux.data[0]=0x1F;       // première d          > "Ad                  registe                  rnée (GPDDR donne la direction des I/O) adresse = 1Fh
page 16
T_IM_Feux.data[1]=0x7F;       // deuxième dom          sque"                  orties sont sur les 4 bits de poids faibles
T_IM_Feux.data[2]=0xF0;       // troisième donnée          leur"                  ties sont sur les 4 bits lsb
// Envoi trame pour définir de la direction des en                  ties
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Feux);// Envoyer trame sur réseau
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((Lire                  ame_Recue)                  (Cptr_TimeOut<200));
          if(Cptr_TimeOut==200)
                  {if(I_Message_Pb_Aff                  0)
                        {I_Message                  fiche=1;
                         gotoxy(2,1
                         printf(" Pa                  nse                  rame de commande en initialisation \n");
                         pri                  Ver                  ation 12 V  est OK \n");}}
    }while(Cptr_TimeOut==2                  ;
// Initialiser les sorties à 0
T_IM_Feux.data[0]=0x1E;                  emier                  née -> "Adresse" du registre concernée (GPLAT définit l'état des sorties) 02h+1Ch = 1Eh
T_IM_Feux.data[1]=0x0F;                  uxiè                  nné                  > "Masque" -> les sorties sont sur les 4 bits de poids faibles
T_IM_Feux.data[2]=0x00;                  i                  do                  > "Valeur" -> au départ toutes les sorties sont à 0 (lampes éteintes)
// Envoi trame pour définir le                  à 0
Ecrire_Trame(T_IM_Feux);  // Env                  e s                  eau CAN
do{}while(Lire_Trame(&                  cue)                  re réponse "Acquittement"
// Initialisations d                  dive
clsscr(),Compteur_Pa
// Pour afficher
gotoxy(1,2);
printf("     TP  n                  MMANDER                  'UN BLOC OPTIQUE         \n");
pri                  ********                  ********                  ********************** \n");
//                  ncipale
//                  *********
                  1)  {Co                  sage
                       /                  ssage
                       /                  st la                  cisation
                       //                  sse à la                  suivante en modifiant le paramétre "Valeur" du message
                       swi                  IM_Feux.d                  a[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_Trame(T_IM_Feux);     // Afficher trame "IM" envoyée à l'écran
                       Ecrire_Trame(T_IM_Feux);      // Envoyer trame sur réseau CAN
                       do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre la réponse "Acquittement"
                       gotoxy(1,10),printf("Trame recue en reponse\n");
                       Affiche_Trame(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

Objectifs :	- Définir, puis envoyer une trame interrogative à un module d'entrées, accessible à une adresse définie. - Tester si une trame a été reçue. - Extraire d'une trame réponse les informations attendues. - Visualiser sur l'écran les trames reçues ainsi que les trames envoyées. - Visualiser sur l'écran les données attendues.
Cahier des charges :	A intervalles de temps réguliers, on in[illegible] le module sur lequel est connecté le commodo lumières afin d[illegible] so[illegible] → Les trames reçues ou en[illegible] sur le[illegible] CAN sont affichées. → La temporisation [illegible] typ[illegible] (comptage [illegible] de passages dans la boucle principale) → Les différen[illegible] mande[illegible] sées par la position de la manette commodo ser[illegible] individuellement.

Matériels et logiciels nécessaires :

Micro ordinateur de type [illegible] Windows 95 ou ultérieur
Logiciel Editeur-Assembleur-Débogueur
Si programmation en C, compilateur GNU C/C++ Réf : EID210100
Carte processeur 16/32 bits [illegible] contrôleur 68332 et son environnement logiciel (Editeur-Cross-Assembleur-Débogueur) Réf : EID210000
Carte réseau CAN PC/104 chez ATON SYSTEMES Réf : EID004000
Réseau CAN avec module Entrées logiques destiné au commodo Réf : EID050000
Câble liaison [illegible] sans câble RS232, Réf : EGD000003
Alimentation [illegible] pour l'alimentation de l'unité centrale Réf : EGD000001,
Alimentation [illegible] 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'ident[illegible] en mode étendu (bits qui seront réceptionnés et placés dans le registre RXBEID8). Le registre conc[illegible] d'adresse 1Eh " (voir documentation technique du MPC25025 pages 37)..
D'autre part, les trois bits de poids faibles de l'identificateur en mode [illegible] devront être positionnés à 1.
L'identificateur défini dans le chapitre 1 devra donc être com[illegible]mme

05 04 xx xx 05 04 1E 07 [illegible]is 2[illegible]s sont [illegible]is en compte)

D'après tableau | Adresse registre | Code fonc[illegible]

→ Définition de variables structurées sous le modèl[illegible]

```
Trame T_IRM_Commodo_Feux;
   // Trame destinée à l'interrogation du module 8 entr[illegible]est co[illegible]e le commodo lumière
```

Rem: La variable structurée T_IRM_Comp[illegible]Comp[illegible]5 octets utiles seulement, 1octet pour trame_info et 4 octets pour l'identificateur en[illegible]e étend[illegible]compr[illegible]à l'adresse du registre concerné par la lecture.

→ Accès et définition des différents élé[illegible]de la v[illegible] structurée "T_IRM_Commodo"

```
T_IRM_Commodo.trame_info.registre=0;          //On initialise tous les bits à 0
T_IRM_Commodo.trame_info.ch[illegible]xten[illegible]=1;//On travaille en mode étendu
T_IRM_Commodo.trame_info.[illegible]tr=0x01;  Type trame
T_IRM_Commodo.trame_info[illegible]mp.dlc=[illegible]01; //Il y aura 1 octet de données
T_IRM_Commodo_ar[illegible].ident.ex[illegible].iden[illegible]cateur.ident=0x05041E07;
  //! c'est su[illegible]29 bits [illegible]our [illegible]inir l'adresse du commodo
```

Des labels définissan[illegible]ès d[illegible]ts l[illegible]srs ont été déclarés dans le fichier CAN_VMD.h

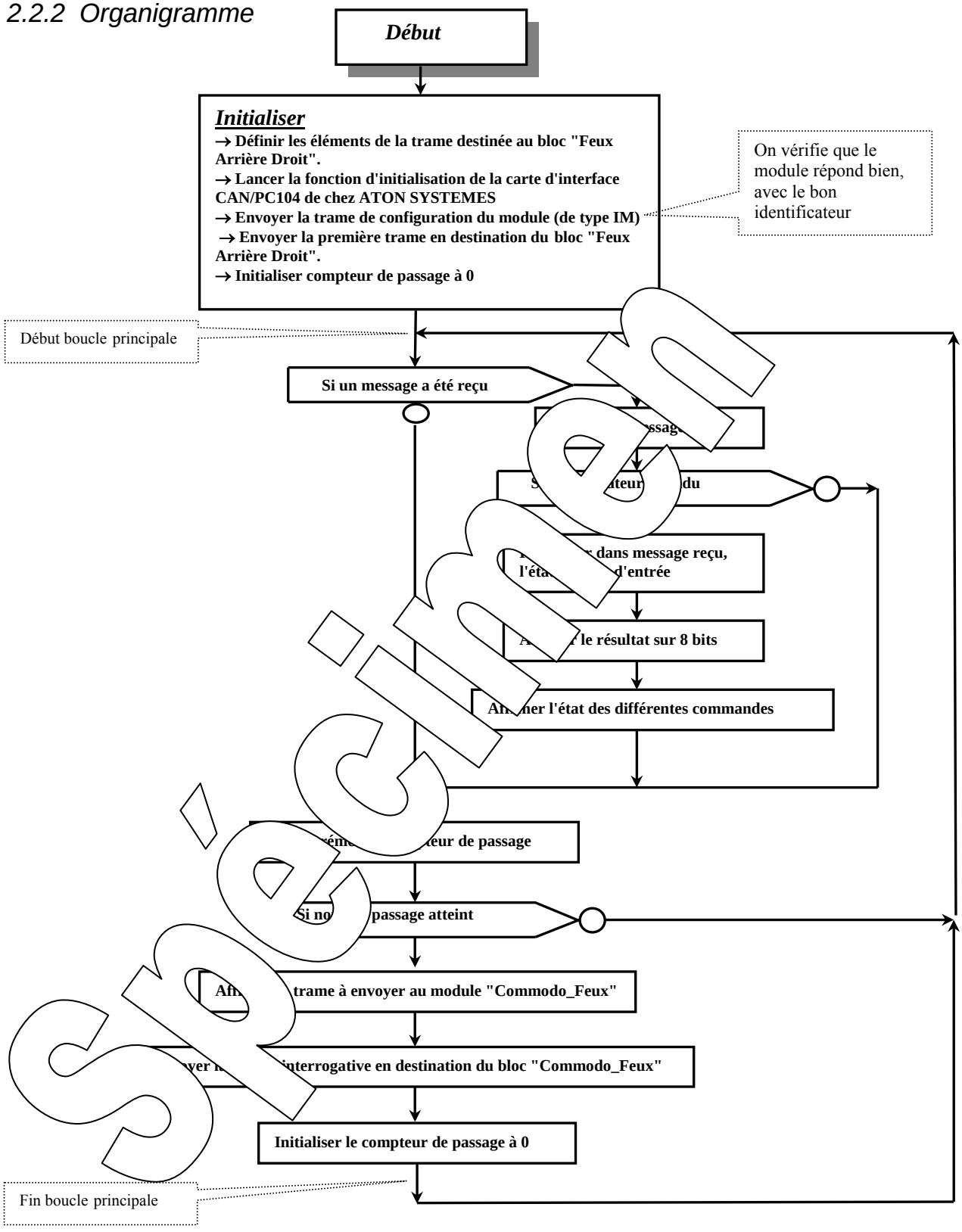
Définition de la trame[illegible]ge [illegible]a t[illegible]me qui sera reçue en réponse

D'après la définition des i[illegible]ate[illegible]onnée en chapitre 1, une trame de réponse à une IRM a le même identificateur qu[illegible]in[illegible]é qui en a été à l'origine.
Vu du module [illegible]0), la réponse à un IRM (Information Request Message) est un OM (Output Message).
La[illegible]ce av[illegible]rogative origine est que cette trame réponse comporte le paramètre "value" ([illegible]ng 0 de la parti[illegible]de la trame). Ce paramètre est l'image du port d'entrée. On récupère donc l'état [illegible]iffé[illegible]mand[illegible]

Accés aux d[illegible]tats binaires des commandes

Le paramètre "[illegible]ue" 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                    forme   equest Message Trame
T_IM_Commodo_Feux;  // Trame pour interroger Module 8E sur Commodo Feux      In    on Messa -> Trame de commande
unsigned char Cptr_TimeOut,I_Message_Pb_Affiche;
// Initialisations
//*****************
clsscr();
/* Initialisation DU SJA1000 de la carte industrielle Aton-CAN a      PCI
Init_Aton_CAN();
// Pour initialiser les liaison en entrées
T_IM_Commodo_Feux.trame_info.registre=0x00;
T_IM_Commodo_Feux.trame_info.champ.extend=1; // On travaill          étend
T_IM_Commodo_Feux.trame_info.champ.dlc=0x03; // Il y aura          e 8 bi          ets envoyés)
T_IM_Commodo_Feux.ident.extend.identificateur.ident=Ident_          Feux;
T_IM_Commodo_Feux.data[0]=0x1F; // première donnée ->  "          d          e concerné;GPDDR donne la direction des I/O)  page 16
T_IM_Commodo_Feux.data[1]=0x7F; // deuxième donnée ->          -> To          ts concernés sauf GP0 (voir doc page 16)
T_IM_Commodo_Feux.data[2]=0x7F; // troisième donnée ->          -> Tous          s en entrée
// Envoi trame pour définir de la direction          entrées s
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Commodo_Feux);  // Envoyer          r réseau
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((Lire_Trame(&Tram          =0)&&(          _TimeOut<200));
          if(Cptr_TimeOut==200)
                  {if(I_Message_Pb_Affich
                          {I_Message_          =1;
                           gotoxy(2,          );
                           printf("          reponse          trame de commande en initialisation \n");
                           printf("          ier si al          tion 12 V  est OK \n");}}
    }while(Cptr_TimeOut==          0);
clsscr();
// Pour trame interrogati          envoyé          commo                    RM'   (Information Request Message)
// Définir données d'identi          ca
T_IRM_Commodo_Feux.trame_info
T_IRM_Commodo_Feux.trame_info          .ext
T_IRM_Commodo_Feux.trame_info          .dl          1;
T_IRM_Commodo_Feux.trame_info          .r
T_IRM_Commodo_Feux.ident.extend          cateu          nt=Ident_T_IRM_Commodo_Feux; // Voir définitions dans fichier CAN_VMD.h
Ecrire_Trame(T_IRM_Commodo_Feux);          envoi          première trame
// Initialiser les va          iver
Compteur_Passage=0;
// Pour afficher tit
gotoxy(1,2);
printf("    TP          COUERIN          ICHER ETAT COMMODO FEUX   \n");
printf(          ********          ********** \n");
//          cipale
//          ********
w          if (1==Lire          me_Recue))  //  On teste si une trame a été reçue;  La fonction renvoie 1 dans ce cas
                  {          toxy(          ntf("Trame recue en reponse a la demande: c'est une 'OM' (Output Messsage) \n");
                          e_Tra          Recue);
                          me_Rec          xtend.identificateur.ident==Ident_T_IRM_Commodo_Feux)
                           { // O          çu l'état du commodo donc on affiche les nouveaux états
                          Etat_Comm          o_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_Trame(T_IRM_Commodo_Feux);
                  Ecrire_Trame(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

Objectifs :	- Analyser un schéma structurel afin de définir la mise œuvre d'une fonction matérielle par une fonction logicielle. - Enchaîner des trames interrogatives et des trames de commande pour satisfaire un cahier des charges imposé. - Tester les trames réponse reçues. - Extraire d'une trame réponse les informations attendues. - Analyser les informations reçues et [illegible] diagnostique. - Représenter par un diagramme d[illegible] des charges imposé. - Coder et programmer un diagr[illegible]s états. - Réaliser des actions cycliques, de p[illegible] précis.
Cahier des charges :	On souhaite en enchaînemen[illegible]que différents états d'un bloc optique (Rien, Veilleuse,Ve[illegible] + Co[illegible]se + Phare etc…) et un fonctionnement du [illegible] On réalisera en[illegible]s fon[illegible]e de contrôle: - On vérifiera q[illegible]le co[illegible]andé renvoi bien une trame d'acquittement. - On vérifie[illegible] fonc[illegible]icielle) que les lampes commandées sont ef[illegible]ement [illegible]es (co[illegible]nment du courant). → On a[illegible]que[illegible]t les lampes concernées ainsi que le résultat du test. → L[illegible]notem[illegible]u clignotant) sera indépendant de la permutation des a[illegible]mpes ains[illegible]e de la période de contrôle. [illegible]mpose l[illegible]ériodes suivantes, dans l'ordre de priorité décroissante: - péri[illegible]de permutation des lampes 3,5 S, [illegible] de commutation du clignoteur 1,6 S. [illegible]périodes devront pouvoir être modifiées facilement. [illegible] Le [illegible]rogramme devra permettre un changement aisé de bloc cible.

Matériels et logi[illegible]san

Micro ordina[illegible] type [illegible]us Windows 95 ou ultérieur
Lo[illegible]diteur [illegible]ble[illegible]bogueur
S[illegible]nmation e[illegible]pilateur GNU C/C++ Réf : EID210100
[illegible] pr[illegible]/32 [illegible]icrocontrôleur 68332 et son environnement logiciel
[illegible]bleur[illegible]gueur) Réf : EID210000
Carte[illegible]u CAN [illegible]104 de chez ATON SYSTEMES Réf : EID004000
Réseau CA[illegible]dules 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 ou 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 entrîne 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 consomme pas un courant suffisant pour inhi[illegible] fonction de diagnostic.

D'après la notice technique du système CAN-VMD et le schéma s[illegible]el du[illegible]e 4sorties de puissance, ces 4 signaux "STATus" sont reliés au circuit d'interface CAN MCP[illegible] et con[illegible]ent donc des entrées de diagnostic que l'on peut lire via le réseau CAN:

- la sortie "STATus" du circuit de puissance pilotée [illegible]rem[illegible]GP4,
- la sortie "STATus" du circuit de puissance piloté[illegible] GP[illegible]st r[illegible]iée à GP5,
- la sortie "STATus" du circuit de puissance pilotée [illegible], est[illegible]ée à GP6,
- la sortie "STATus" du circuit de puissance [illegible]ilotée pa[illegible]iée à GP7.

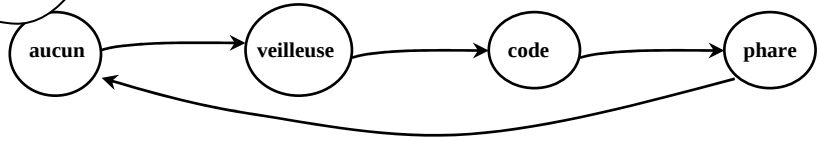
Activation des lampes et diagnostique du blo[illegible] avan[illegible]t

Pour allumer les lampes, il faut env[illegible]er une[illegible]de ty[illegible]"Input message" avec la fonction "Write Register" sur son registre d'accès a[illegible]t de so[illegible]PPIN [illegible]après notice technique du circuit MCP25050 page 22).
Dans ce cas, d'après tableau donné chapitr[illegible]r le b[illegible]avant droit, l'identificateur sera 0E880000, le paramètre "addr" sera 1E (adresse d[illegible]istre[illegible]T – page 37 de la "Data sheet" du MCP25à50), le paramètre "mask" sera 0F (les 4[illegible]ont sur le[illegible]oits de poids faibles du port), le paramètre "value" sera défini par l'état souhaité.

Pour récupérer les ét[illegible]logi[illegible]s imp[illegible]port d'entrée, il faut envoyer au module considéré une trame IRM "Information Requ[illegible]" avec [illegible]fonction "Read register". Dans ce cas, l'identificateur contient l'adresse du registre con[illegible]ré (G[illegible]IN d'adresse 1E - d'après notice technique du circuit MCP25050 page 37) ainsi que le code fonctio[illegible]
Par conséquent, pour le feu[illegible](d'après le tableau donné au chapitre 1 de ce document) l'identificateur à donner sera en[illegible]0E8[illegible]/ et la trame ne comportera pas de paramètre en zone "data".
Le module réc[illegible]de [illegible]rame répondra avec le même identificateur, mais avec au rang 0 de la zone "Dat[illegible]l'état [illegible]d'entr[illegible]

[illegible]mutati[illegible]lamp[illegible]

O[illegible]s[illegible]s état[illegible]ampes suivante:
aucu[illegible]uis, veilleuse seule puis, veilleuse + code puis, veilleuse + phare.
Cet enchaîn[illegible]quentiel peut être représenté par le diagramme d'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 de 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 *)(0xFFFA24)
#define PICR   *(short *)(0xFFFA22)
```

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 ci-après 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 générée par 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 individuellement à ses 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 auxiliaires images de l'état des lampes:
- pour un bloc optique av

```
#define Veilleuse Imag          it.b0
#define Code          Feux.
#define Pha                   .b1
#define Cli                   .bit.b3
```

- pour un bloc optique ar

```
#define          Image_F          .b0
#define St          Feux
#define Clig                 t.b2
#define Autre I              t.b3        // sur le V.M.D. c'est le claxon qui est piloté
```

Pour allumer une lampe, il suffira de:
- mémoriser l'image,
Ph 1; // pour exemple
- transférer l'image dans le paramètre "value" de trame de commande (trame IM),
M_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" n'est exécutée qu'une seule fois,
- une boucle principale qui est parcourue tant que l'on ne fait pas un "Reset".

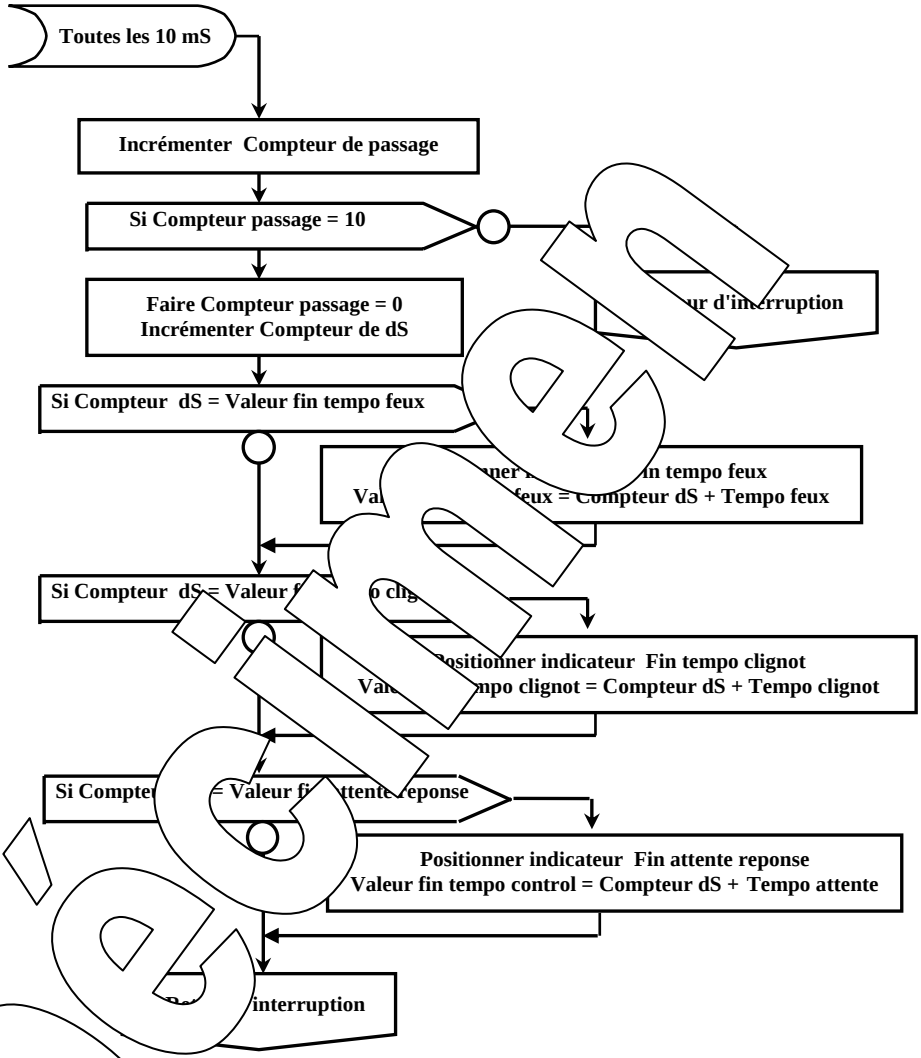
Dans la boucle principale, on n'envoie une trame que lorsque la mesure où le module récepteur de la trame précédemment envoyée a répondu:

- 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 "Timeout" temps maxi d'attente réponse après envoi d'une trame.

3.2.2 Organigrammes

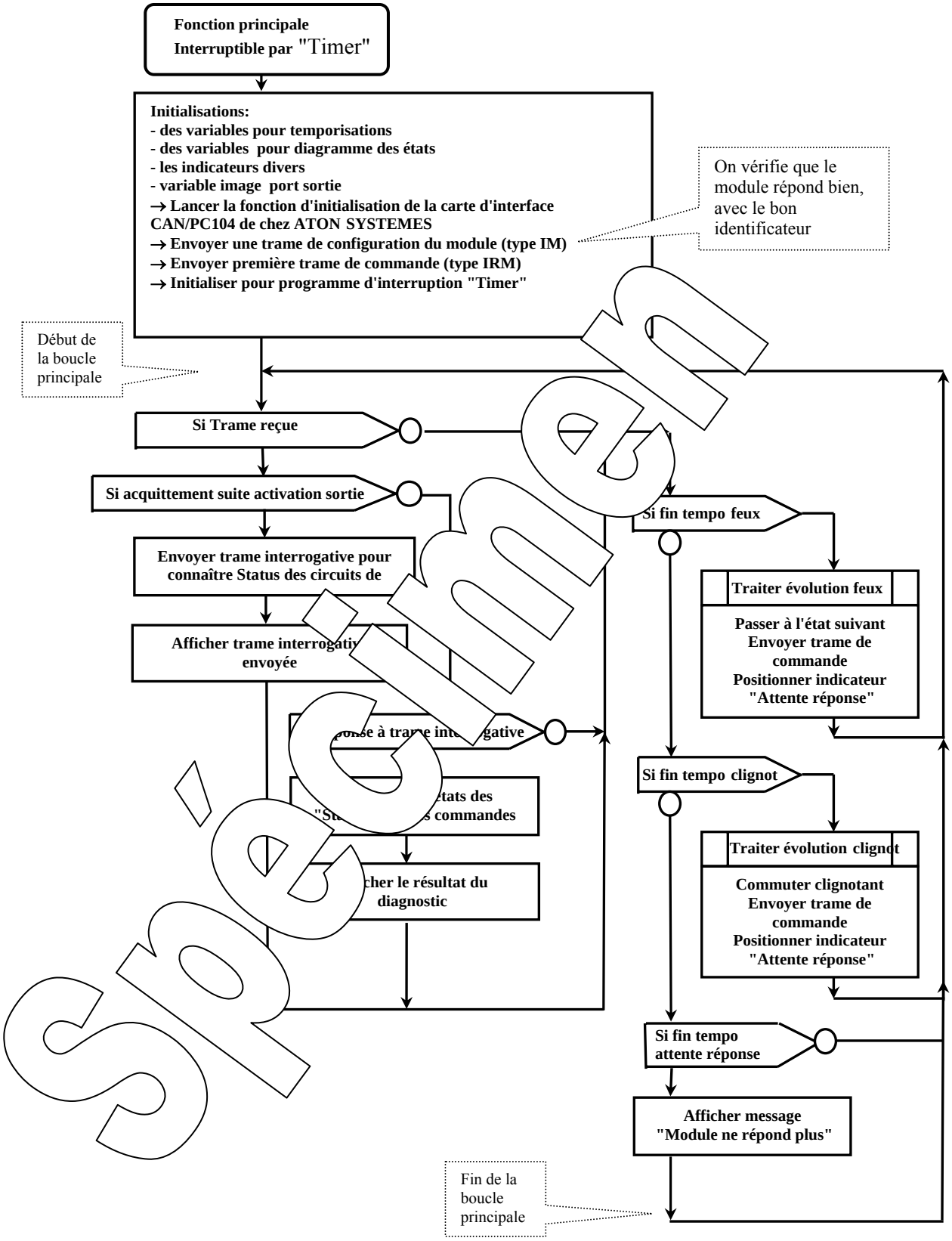
Organigramme décrivant la génération des indicateurs de fin de temporisation:



Remarque

Lors de l'initialisation il faudra procéder aux affectations suivantes:
Compteur de passage = 0,
Valeur fin tempo feux = Tempo feux ,
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'acquittement.
        *       - 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 décr          te:
        *       - 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 d          ible.
        *-----------------------------------------------------------------------------------------------------
        *  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 dizi      seco          S
#define Tempo_Clignot 16  // En dizièm    e s conde          S
#define Tempo_Att_Rep 40  // Attente répons     dizie          conde ->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_      ux,Image_Eta          rs;
#define Valeur_Feux Im      _Fe          ur          n accès port complet
#define Veilleuse Image_Fe
#define Code Image_Feux.b
#define Phare Image_Feux.          b2
#define Clignot Image_Feu          b
#define S_Veilleuse Image_          x.bi     // Status veilleuse
#define S_Code Image_Etat_Fe          bF
#define S_Phare Im          Feu
#define S_Clignot          Feux.          7
// Pour les indi          S
#define I_Att_          uit In          urs.bit.b0
#defi          Fin_          x Ind          rs.bit.b1
#d          n_Temp          teurs.bit.b2
#          in_Tempo_          ateurs.bit.b3
          e I_Fi          Att          dicateurs.bit.b4
          e I          errog          teurs.bit.b5
#          _Mes          urs.bit.b6
#d          ge_          iche In          eurs.bit.b7
// De          on des          s
Trame Trame_Re      ue;
Trame Trame_
Trame T_IM_F          // 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_Trame_Envoyee Trame_Envoyee.ident.extend.identificateur.ident
#define Ident_Trame_Recue Trame_Recue.ident.extend.identificateur.ident

// Pour les temporisations
WORD Compteur_Passage,Compteur_dS; // dS -> dizième 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
//*****************
clsscr();
//  Définition des trames pour activer ou lire un bloc optique
// D'aprés doc SJA1000 et  doc MCP25050 pages 22 (fonction "W   Reg   r")  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   étend
T_IM_Feux.trame_info.champ.dlc=0x03; // Il y aura 3 do     bits   ts envoyés)
T_IM_Feux.ident.extend.identificateur.ident=Ident_T_IM_    dentificateur Feux aVant Gauche
                                                          // Voir définitions dans CAN_VMD.h

// Pour trame interrogative ->  IRM  (Information    rame)
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_   ent_Ident_    FVG; // Voir définitions dans CAN_VMD.h


/* Initialisation DU SJA1000 de la carte ATON    s" su    dc PC104*/
Init_Aton_CAN();
// Envoi trame pour définir de la di      des en     rties
T_IM_Feux.data[0]=0x1F;  // premiè     e -> "Adres     du registre concernée -> GPDDR
T_IM_Feux.data[1]=0x7F;  // deuxiè     nnée -> "    sque" ->  Voir Doc page 16
T_IM_Feux.data[2]=0xF0;  // troisi     onnée -    leur" -> les sorties sur 4 bits de poids faibles
I_Message_Pb_Affiche=0
do {Ecrire_Trame(T_IM_    ux);// Envoy     rer    trame sur réseau CAN
          Cptr_TimeOut=
          do{Cptr_TimeOut++    e_    ame_Recue)==0)&&(Cptr_TimeOut<200));
       if(Ident_Trame_Rec     uden     M_FVG)Cptr_TimeOut=200; // Test si identificateur correct
          if(Cptr_TimeOut=     )
                 {if(I_Mes     fic     0)
                        {     e_Pb     che=1;
                         go
                         rin     reponse a la trame de commande en initialisation \n");
                         ntf(     ifier si alimentation 12 V  est OK \n");}
                         for(Temp     <100000;Temp++);} // Pour attendre un peu!
    }while(Cpt     t==20
clss
//     semble
I     s.valeur=0;
     = Etat
     Fe
//                la p     trame -> état initial des sorties (qui sont initialisées à 0
//          les     es à 0
T_IM_     ata[0]=0;     // première donnée -> "Adresse" du registre concernée (GPLAT définit l'état des sorties)
                                                                  03h+1Ch = 1Eh
T_IM_Feux.da          // deuxième donnée -> "Masque" -> les sorties sont sur les 4 bits de poids faibles
T_IM_Feux.da     ;  // troisième donnée -> "Valeur" -> au départ toutes les sorties sont à 0 (lampes éteintes)

Ecrire_Trame(T_IM_Feux);
Trame_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 (1==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'acquittement suite à l'envoi d'une commande feux
                         I_Att_Rep_Acquit=0;
                         I_Autorise_Emis_Mes=1;
                         gotoxy(1,8);
                         printf("    Trame d'acquittement suite à une 'IM' (Input Message)\n");
                         Affiche_Trame(Trame_Recue);
                         // On peut envoyer trame interrogative afin de v rifier l'état des feux
                         Ecrire_Trame(T_IRM_Feux);
                        Valeur_Fin_Tempo_Att_Rep = Compteur_dS+Tempo_A
                        I_Fin_Tempo_Att_Rep=0;
                         gotoxy(1,12);
                         printf("    Trame interrogative envoyee -> '                format      Request Message) \n");
                         printf("    En vue de tester le bon fonctionnem                 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         à u      terrogation
                               I_Att_Rep_Interrog=0;
                               I_Autorise_Emis_Mes
                               gotoxy(1,16);
                               printf("    Trame      se a          terrogation -> 'OM' (Output Message) \n");
                               Affiche_Trame(T          e);

                              // Analyse de          les e affichage résultat diagnostique
                                       Ima          Feux          Trame_Recue.data[0];
                                       if(Ve          ==1 &&          lleuse==0)
                                       {gotoxy          printf("!!     Probleme sur Veilleuse  \n");}
                                       if(Veille          & S_Veilleuse==1)

                                       {g          20),p          f("!!     Veilleuse   OK                  \n");}
                                       if(C          & S_Code==0)
                                       {gotoxy          printf("!!     Probleme sur Code         \n");}
                                       if(Code==          S_Code==1)
                                       {goto          (1,21),printf("!!     Code   OK                       \n");}
                                       if(P          ==1 && S_Phare==0)
                                       {got          ,22),printf("!!     Probleme sur Phare   \n");}
                                       if(P          ==1 && S_Phare==1)
                                                     1,22),printf("!!     Phare   OK                    \n");}
                                                  ignot==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");}

                }
                       Tem
                                  in_Te          eux=0;
                                  asse à l'état suivant
                                  sw          tat)
                                  {case Etat_aucune :      {Etat=Etat_veilleuse;
                                                            Veilleuse =1; }
                                   break;
                                   case Etat_veilleuse :   {Etat=Etat_code;
                                                            Code=1;}
                                   reak;
                                   ase 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!
                         {clsscr();
                        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 inte                on
                        Valeur_Fin_Tempo_Att_Rep = Compteur_dS+Tempo_At                 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

Objectifs :	- Réaliser une application de contrôle commande d'un système pilotable par réseau CAN. - Visualiser sur l'écran l'état du système. - Réaliser des temporisations précises. - Réaliser des tâches de vérification e[illegible]de.
Cahier des charges :	A intervalles de temps régu[illegible]rrog[illegible]module sur lequel est connecté le commodo lumières afin [illegible]onn[illegible]e s[illegible] état. En fonction de l'état du com[illegible]umi[illegible]on active les différentes lampes des blocs optiques avant[illegible]ière. → La temporisation [illegible]u fonctionnement des clignotants est réalisée par "Timer pro[illegible]le" (i[illegible]au micro-processeur). → Les diffé[illegible]com[illegible]imposées par la position de la manette co[illegible]do sera[illegible]chées [illegible]viduellement. → On c[illegible]le bo[illegible]ionnement des ampoules des différents blocs optiques.

Matériels et logiciels nécessaires :

Micro ordinateur de type [illegible] sous Windows 95 ou ultérieur
Logiciel Editeur-Assem[illegible]-Progr[illegible]
Si programmation en C , [illegible]ateur GNU C/C++ Réf : EID210100
Carte processeur [illegible]ts à [illegible]trôleur 68332 et son environnement logiciel (Editeur-Cros[illegible]m[illegible]Debogueur) Réf : EID210000
Carte réseau [illegible]2C/10[illegible]hez ATON SYSTEMES Réf : EID004000
Ré[illegible]N ave[illegible]
[illegible]odule 8 Entrées [illegible]s destiné au commodo Réf : EID050000
[illegible]odu[illegible]uissa[illegible]stinés aux feux arrière gauche, droit et aux feux avant gauche et droit Réf : E[illegible]
Câble de liaison U[illegible]ou à défaut câble RS232, Réf : EGD000003
Alimentatio[illegible]8V, 1A pour l'alimentation de l'unité centrale Réf : EGD000001,
Alimentation [illegible]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)

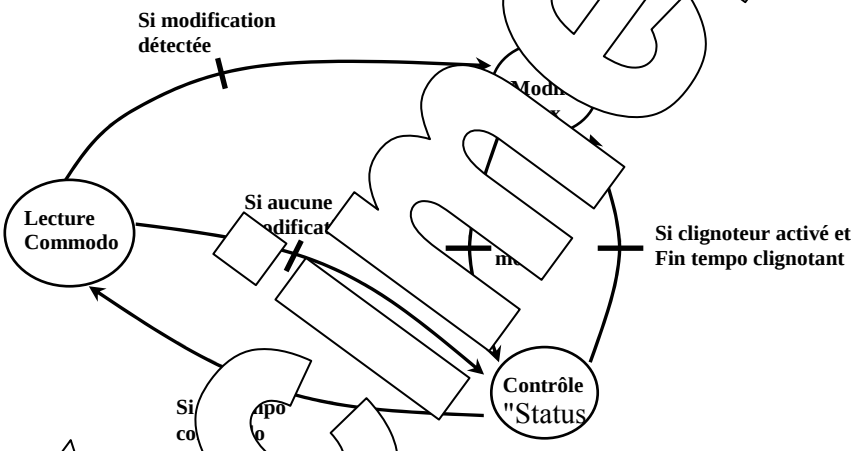
Tâches secondaires

→ On interroge successivement les 4 blocs optiques et on vérifie si les valeurs des "Status" corroborent les commandes envoyées. Si une différent est détectée, on message d'alert[illegible] affiché.

→ Suite à l'envoi d'une trame, on vérifiera que la trame reçue en répo[illegible]rrecte (acquittement du module ayant reçu une trame de commande, ou réponse cohérente du mod[illegible]ant [illegible]e trame de interrogative.

Diagramme des états principaux

On peut matérialisé le fonctionnement global par le diagram[illegible]ts su[illegible]



Etat "Modification f[illegible]x"

Il s'agit d'envoyer un[illegible]ra[illegible]omm[illegible]e de type IM) à chacun des blocs optiques (Voir TP n°1). On décide d'envoyer les tra[illegible]de commande dans l'ordre suivant:

→ Feux aVant [illegible]he[illegible]G)

→ puis Feux aVant Droi[illegible]

→ Feux aRrière C[illegible]e (F[illegible]

→ et enfin Feux [illegible]it (FRD).

Seules deux i[illegible]tions [illegible]à changer lorsque l'on passe d'un bloc optique à un autre:

→ l'[illegible]ificate[illegible]

→ [illegible]mètre "Va[illegible]

E[illegible]do"

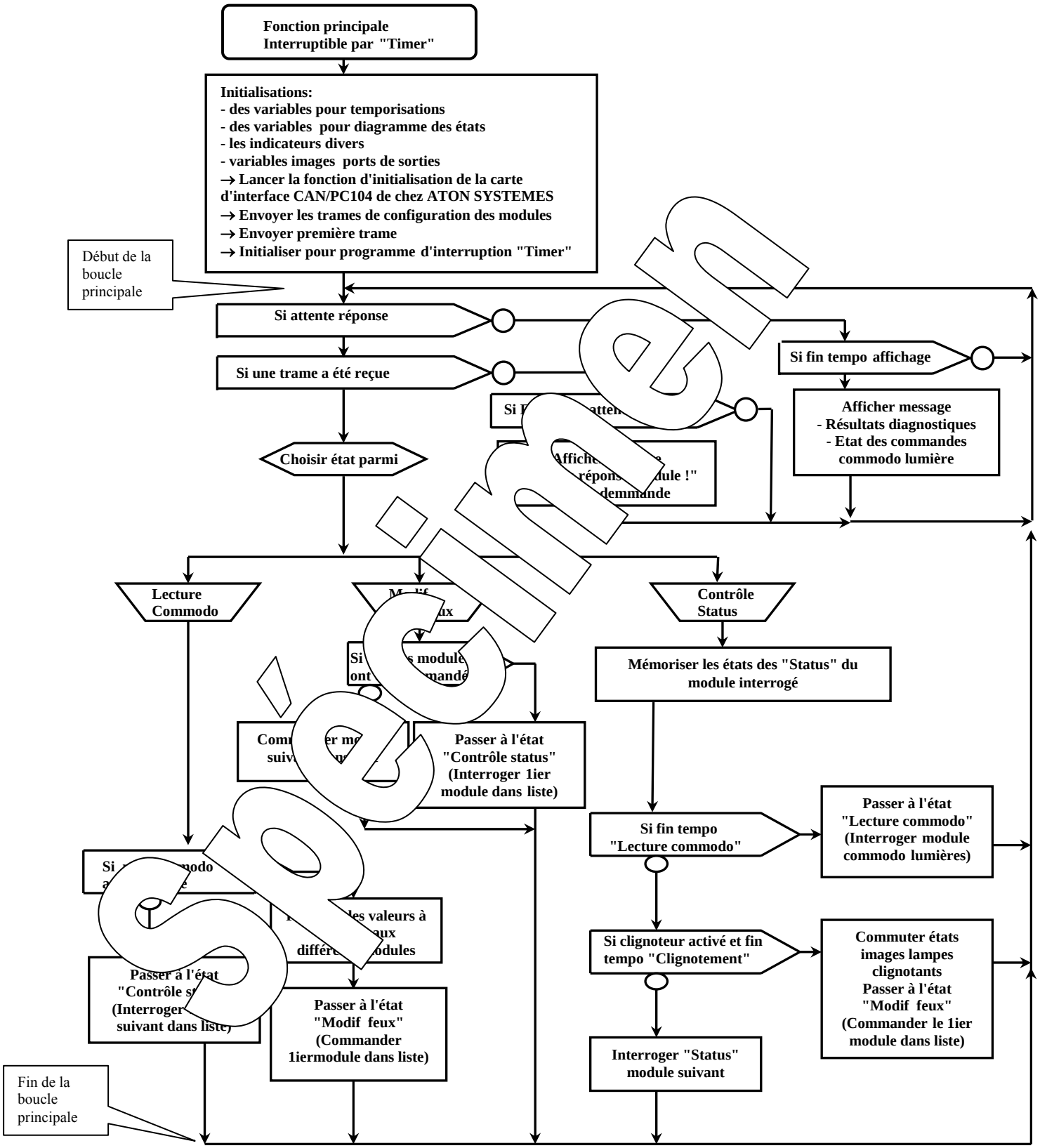
Il s'a[illegible]nvoyer [illegible]rame in[illegible]rogative (trame de type IRM) au module 8 entrées sur lequel est connecté le commodo (V[illegible]2). L'analyse de la trame réponse et la comparaison avec l'état mis en mémoire permet de détecter[illegible]ement éventuel.

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 connecter 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 permutati                utres 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 décroissant
*         - 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 b
*----------------------------------------------------------------------------------------
*  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 diz          secondi          S
#define Tempo_Clignot 8                 // En dizie          onde
#define Tempo_Att_Rep 20      // Attente réponse en u                secon      2 S
#define Tempo_Affichage 10    // Attente réponse en diz          conde -> 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 varia
//----------------------
// Pour les indicateurs di      s                inal
union word_bits Indicateurs;
#define I_Attente_Reponse Ind          urs.b
#define I_Fin_Tempo_Commodo I          teur          .b1
#define I_Fin_Tempo_Clignot I          u          t.b
#define I_Fin_Tempo_Affichage          s.bit
#define I_Fin_Tempo_Att_Rep Indi          bit
#define I_En_Att_Rep          rs.b
#define I_Clignot_Ga          urs.
#define I_Clignot_Dr          bit.b
#define I_Message          che          rs.bit.b8
// Déclaration d
Trame          e Recue,
Tra          voyee,
Tr                  //          put Message" pour commande module 4 Sorties de puissance
                    // Tra          Information Request Message" pour interroger "Status" lampes
                                 ommodo
                    s ide          urs Trame envoyée <-> Trame reçue
#                   Trame          ident.extend.identificateur.ident
#de          me_R          ame_Rec          t.extend.identificateur.ident
#defi          _IRM  T          dent.exten          identificateur.ident
#define Ident_T_IM  T          t.extend.identificateur.ident
#define Valeur          a[2]
// Pour les te
WORD Compteur          pteur_dS; // dS -> dizième 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
//*****************
clsscr();
/* Initialisation DU SJA1000 de la carte ATON-Systemes" sur  */
Init_Aton_CAN();
//  Définition des trames pour activer ou lire un bloc optique
// D'aprés 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 envoyé
Ident_T_IM=Ident_T_IM_FRD;// C'est l'identificateur du bloc optique arrière droi
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 de po es
//Configuration du registre de dirrection et
//Verification de la presence des modules
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Envoyer une première trame sur réseau CAN
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_ 0));
       if(Ident_Trame_Recue!=Ident_T_AIM_FRD)Cptr_TimeOut=200; // Test si i tei rect
          if(Cptr_TimeOut==200)
                  {if(I_Message_Pb_Affiche==0)
                          {I_Message_Pb_Affiche=1;
                           gotoxy(2,10);
                           printf(" Pas de reponse du F e Dro rame de commande en initialisation \n");
                           printf("  Verifier si alime est O }
                  for(Temp=0;Temp<100000;Temp++);} // Po peu!
    }while(Cptr_TimeOut==200);
Ident_T_IM=Ident_T_IM_FRG;// C'est l'identif teur d tique a uche
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Envoyer une premi ram sur r
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((Lire_Trame( ue)==0 TimeOut<200));
       if(Ident_Trame_Recue!=Ident_T_AIM_FRG)Cptr_ 0; // identificateur correct
          if(Cptr_TimeOut==200)
                  {if(I_Message_Pb_Affiche
                          {I_Message_P 1;
                           gotoxy(2,1
                           printf("  reponse Fe aRri e Gauche a la trame de commande en initialisation \n");
                           printf("  ier si al ation 12 V  est OK \n");}
                  for(Temp=0;Temp<1000 ++);} / attendre un peu!
    }while(Cptr_TimeOut== 0);
Ident_T_IM=Ident_T_IM_FV C'est l'identi du optique aVant gauche
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// E voy re réseau CAN
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++; e((Li ame(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
       if(Ident_Trame_Recue!= T_A G)C TimeOut=200; // Test si identificateur correct
          if(Cptr_TimeOut==20
                  {if(I_Messa iche=
                          {I_ e=1;
                           goto
                           ntf eponse Feux aVant Gauche a la trame de commande en initialisation \n");
                           f(" Verifier si alimentation 12 V  est OK \n");}
                  emp=0, 0000;Temp++);} // Pour attendre un peu!
    }while(Cptr_ 200);
Ident  dent_I  / C'es ntificateur du bloc optique aVant Droit
I_M  ffiche=
d  ame(T_IM);  première trame sur réseau CAN
   Cptr_ Out=0;
   d  ut++;  ire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
   if e!=Id FVD)Cptr_TimeOut=200; // Test si identificateur correct
          =200)
                  {I  ssage_Pb_ che==0)
                          {I_Messag_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");}
                  emp=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 Commdo Lumiere
T_IM.data[2]=0xFF;  // troisième donnée -> "Valeur" ->8 bits sont des entrées
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Envoyer une première trame sur réseau CAN
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
       if(Ident_Trame_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 Lumiére 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);
```

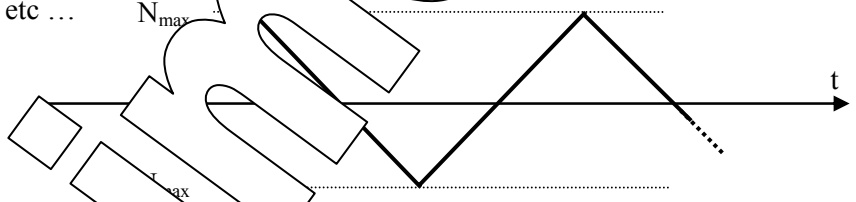
```
clsscr();

// 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_Trame(T_IRM);
//Trame_Envoyee = T_IRM;
//I_Attente_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_Trame(T_IRM);
while((Lire_Trame(&Trame_Recue)==0));
Trame_Envoyee = T_IRM;
I_Attente_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 millisecon
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 L'ETAT DES AMPOULES");
printf("    ********************************************************************");
// Boucle principale
//******************
while(1)  {if(I_Attente_Reponse) // Une trame est atten
                  {
                   if (1==Lire_Trame(&Tr      ue)) //      reçue,      tion retourne 1
                          // Une Trame a      re      e
                          {I_Attente_Reponse=0
                           I_Fin_Tempo_Att_Re      n réal      po attente réponse
                           Valeur_Fin_Tempo_Att_      pteur_      empo_Att_Rep;
                           I_En_Att_Rep=0;
                           if(Etat==Etat      ommodo_
                                  //      ns l'éta      e Commodo"
                                  Valeur_Comm      ux=~(Trame_Recue.data[0]); //On récupére l'état du commodo
                                  if(Val      _Comm      do_Feux!= Valeur_Commodo_Feux_Mem)
                                  // O      tecté 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 aRière
                                   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 aRière
                                   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 aRière
                                   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
                                  else      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_Trame(T_IM); // Envoyer trame
```

```
                                        Trame_Envoyee =T_IM;
                                        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_IM); // Envoyer trame
                                        Trame_Envoyee =T_IM;
                                        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_IM); // Envoyer trame
                                        Trame_Envoyee =T_IM;
                                        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) // Mo        vant        liste
                                         {case 0 :          {/        e contr        ux aVant Gauche
                                                            Vale        FVG=Tr        cue.data[0];
                                                            // On p        t être)  au feu suivant
                                                                T_IRM_FVD;} //Feux aVant Droit
                                          break;
                                          case 1 :                le co        e du Feux aVant Droit
                                                             Val        tatus_FVD=Trame_Recue.data[0];
                                                             //        pass        peut être) au feu suivant
                                                             T_I        ent_T_IRM_FRD;} //Feux aRrière Droit
                                          break;
                                          case 2 :           //es        ontrole du Feux aRrière Droit
                                                             _FRD=Trame_Recue.data[0];
                                                             //        (peut être) au feu suivant
                                                             Ident_T_IRM=Ident_T_IRM_FRG;} //Feux aRrière Gauche
                                          break;
                                                             //C'est le controle du Feux aRrière Gauche
                                                             F_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 te        est te        d'aller lire le commodo
                                   Fin_T        odo)
                                          {// R        l'état "Lecture Commodo"
                                           I_Fin_        ommodo=0;
                                           Etat = E        Lect_Commodo_Feux;
                                           T_IRM=Ident_T_IRM_Commodo_Feux;
                                           // On envoie sur le bus la première trame
                                          Ec        rame(T_IRM);
                                          Trame_Envoyee = T_IRM;
                                          I_Attente_Reponse=1;}
                                 // On        si clignotants activés ET si fin tempo clignotant
                                 e if((I        not_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 feux 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"

                  e        Si t
                           (I_Fin_T        tt_Rep) // On attend depuis trop longtemps une réponse !
                            {clsscr();
                            gotoxy(1,2); // On réaffiche le titre
printf("    TP        VER 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 aRierre Droit
                    gotoxy(1,20),printf("Feux arriere droit:\n");
                    if(Veilleuse_FRD==1 && S_Veilleuse_FRD==0){gotoxy(1,21),printf("!                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("!!                  otant 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("!! Prob                Stop Ar          it       \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                Prob          Lampe Arriere Gauche      \n");}
                    if(Veilleuse_FRG==0 && S_Veilleuse_FRG==1){gotoxy(1,1                                                     \n");}
                    if(Clignot_FRG==1 && S_Clignot_FRG==0){gotoxy(1,18),                f("!!          leme sur Clignotant Arriere Gauche \n");}
                    if(Clignot_FRG==0 && S_Clignot_FRG==1){gotoxy(1,18),                f("                                     \n");}
                    if(Stop_FRG==1 && S_Stop_FRG==0){gotoxy(1,19),printf(                e su          p 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,                printf(                Klaxon Arriere Gauche      \n");}
                    if(Klaxon_FRG==0 && S_Klaxon_FRG==1){gotoxy(                  ntf(                                      \n");}
                    // Pour l'éta commodo
                    gotoxy(4,24);
                    printf("Etat des differentes entrees imp          r le          \n");
printf(" Veilleuse=%d , Code=%d , Phare=%d , clign. G.=%d                eilleu          de,Cde_Phare,Cde_Clign_Gauche);
printf(" Klaxon=%d ,    Feux de stop=%d ,    clign. D= %d\                n,Cde_S          _Clign_Droit);
                                    } // Fin si on n'es                te de r          nse aprés message d'alerte
                            } // Fin if fin tempo aff
            }
} // 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          ontrol S          "
                    // Pour préparer la          rogative po          dule suivant dans la liste
                      switch(Rang_Contro          ) // Module          uivant dans la liste
                            {case 0 :          T_IRM=Ide          IRM_FVG; //Feux aVant Gauche
                             break;
                             case 1 : Id          RM=Ider          RM_FVD; //Feux aVant Droit
                             break;
                             ca          dent_          IRM_FRD; //Feux aRrière Droit

                                        T_IRM=Ident_T_IRM_FRG; //Feux aRrière Gauche
                             ak;
                             ult          ang          trol_Stat=0;
                                        dent_T_IRM=Ident_T_IRM_FVG;} //Feux aVant Gauche
                             }
            // On envoi          int          r le bus
            Ecrire_Tra
            Trame_Env
            I_Atten          se=1;
} // FIN fonctio          à Etat          l Status"


//          "Passer à l          ion Feux"
          asser_a_E          dif_Fe
          I_TEMP
                    Passe          t "Modif feux"
                    comman          ier feux
 Ide          t_T_IR          // Feux a          Gauche: Premier dans la liste
 Valeur          valeur_FV
 Rang_Modif_Feux=1;
 // On envoie          ommande sur le bus
 Ecrire_Trame_
 I_Attente_Repon
} //  FIN Fonction "Passer à l'Etat Modification Feux"

// Fin du fichier source C
```

5 TP N°5: COMMANDER LE MOTEUR ESSUIE GLACE

5.1 Sujet

Objectifs :	- Définir les différentes trames de commande à faire transiter par le réseau CAN en fonction d'une action souhaitée. - Commander un actionneur électrique (moteur à courant continu), dans les deux sens de rotation, par l'intermédaire d'un pré-actionneur commandable par réseau CAN. - Faire varier la vitesse d'un moteur é[illegible]e par l'intermédaire d'un pré-actionneur (interface de type PWM) [illegible]le par réseau CAN.
Cahier des charges :	Réaliser le cycle périodique [illegible]nné [illegible]ivant: - montée progressive de la [illegible]esse [illegible]s le sen[illegible] positif, jusqu'à la vitesse maxi - descente progressive de [illegible]e [illegible]us [illegible] vitesse nulle, - montée progressive de la [illegible] dan[illegible] sens négatif, jusqu'à la vitesse maxi - descente progress[illegible] vit[illegible] à vitesse nulle, - etc … 

Matériels et logiciels nécessaires

Micro ordinateur de t[illegible]e [illegible] Wi[illegible]ou ultérieur
Logiciel Editeur-Assem[illegible]-D[illegible]ueur
Si programmation en C, [illegible]mp[illegible]eur [illegible]U C/C++ Réf : EID210100
Carte processeur 16/32 b[illegible]croc[illegible]ôleur 68332 et son environnement logiciel
(Editeur-CrossA[illegible]ur-[illegible])Réf : EID210000
Carte réseau CA[illegible]e chez ATON SYSTEMES Réf: EID004000
-1 mo[illegible]'ectro[illegible]"asservissement moteur" Réf : EID052000
[illegible] parti[illegible]tive [illegible]ystème essuie glace Réf: EID053000
C[illegible]iaison US[illegible]aut câble RS232, Réf: EGD000003
[illegible]enta[illegible]C 8 [illegible]our l'alimentation de l'unité centrale Réf: EGD000001
A[illegible]r l'ali[illegible]ion des modules CAN (réseau "énergie").

Durée estim[illegible]eures

5.2 Eléments de solution

5.2.1 Analyse

Principe:

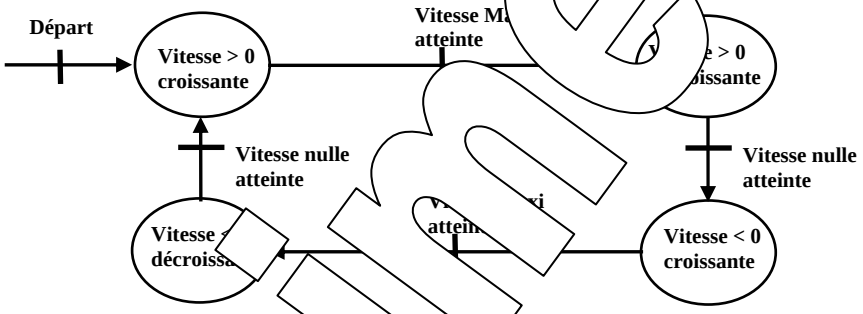
Le module d'interface CAN mis en œuvre dans ce TP est le module repéré "Asservissement".
Ce module permet la commande d'un moteur à courant continu 24V/ 1A, dans les deux sens de rotation, en mode "PWM" (modulation de largeur d'impulsions).
Un circuit de puissance (réf: L6202"), pilotable par signaux logiques est implanté sur le module.
D'après le schéma électronique du module "Asservissement", ce pilotage se fait par les sorties GP2, GP3 et GP4 de l'interface CAN (circuit MCP25050):

→ GP2 ou "PWM1" sortie logique à rapport cyclique variable, qui permet de faire varier la vitesse du moteur dans le sens positif,

→ GP3 ou "PWM3" sortie logique à rapport cyclique variable, qui permet de faire varier la vitesse du moteur dans le sens négatif,

→ GP4 ou "ValidIP" sortie logique qui, lorqu'elle est positionnée à 1, valide le circuit d' Interface de Puissance "L6202".

Compte tenu de la façon de commander le moteur, le cycle de fonctionnement comporte donc 4 états:



Un diagramme à 4 états nécessite, pour son codage, 2 variables binaires:

- indicateur "I_Sens _Variation":
  I_Sens_Variation = 1 lorsque la vitesse est croissante en module
  I_Sens_Variation = 0 lorsque la vitesse est décroissante en module
- indicateur "I_Sens _Rotation":
  I_Sens_Rotation = 0 lorsque la vitesse est positive (pilotée par PWM1)
  I_Sens_Rotation = 1 lorsque la vitesse est négative (pilotée par PWM2).

Définitions des trames pour commander le moteur:

Définition des éléments d'identification d'une trame de type "IM" permettant de commander le module "Asservissement"

Dans ce cas, la trame envoyée par le contrôleur CAN (Circuit SJA1000 sur carte CAN_PC104) sera vue par le récepteur (circuit MCP2505 du module) comme une IM "Input message", avec la fonction "Write register" (voir documentation technique du MPC25025 pages 22). On pourra ainsi modifier les différents registres du module "Asservissement".
Le codage dans le registre 1, pour un "IM" envoyé à la carte "Asservissement"est :
0x00880000

→ Définition des variables structurées sous le modèle "**Trame**":

```
Trame T_IM_Asservissement;
```

→ Définition des éléments d'identification de la variable structurée "**T_IM_Asservissement**"

```
T_IM_Asservissement.trame_info.registre=0x00;  //On initialise tous les bits à 0
T_IM_Asservissement.trame_info.champ.extend=1; //On travaille en mode étendu
T_IM_Asservissement.trame_info.champ.dlc=0x03; //Il y aura 3 octets de données
T_IM_Asservissement.ident.extend.identificateur.ident=0x00880000;
```

A chacune des trames de commande "IM" il y aura donc à définir les trois octets associés.

Définition des trois octets de données associées pour:

→ *définir les entrées et sorties*

Il faut initialiser le registre GPDDR ("Data Direction Register") en écrivant un 1 si bit d'entrée et un 0 si bit de sortie(doc MCP25050 p27). D'après schéma de la carte:

GP7=fs -> entrée; GP6=fcg -> entrée; GP5=fcd -> entrée; GP4=ValidIP -> sortie;
GP3=PWM1 -> sortie; GP2=PWM2 -> sortie; GP1=AN1 -> entrée; GP0=AN0 -> entrée;

```
T_IM_Asservissement.data[0]=0x1F; // Adresse du registre GPDDR en écriture
                                                          (doc MCP25050 p16)
T_IM_Asservissement.data[1]=0x7F; // Masque: bit 7 non concerné (doc MCP25050 p16)
T_IM_Asservissement.data[2]=0xE3; // Valeur: à charger dans le registre adressé
```

→ *initialiser la sortie GP2 en sortie PWM1 (variation de la vitesse du moteur dans le sens positif)*

D'après la notice technique du circuit MCP25050 (pages 30 à 32), la génération du signal PWM1 se fait à partie du "Timer1" et la fréquence de ce signal est choisie grâce au registre "T1CON" d'adresse 05_H (page 15 Doc MCP25050).

bit 7 =1 TMR1ON Validation du "Timer 1"
bits 5:4 seront mis à 0 pour avoir un coefficient de division de fréquence égal à 1 ("TMR1 prescaler value" = 1)

```
T_IM_Asservissement.data[0]=0x21; // Adresse du registre T1CON en écriture
 (doc MCP25050 p15) 05H + décalage = 05H + 1CH = 21H
T_IM_Asservissement.data[1]=0xB3; // Masque sur le registre (doc MCP25050 p32)
T_IM_Asservissement.data[2]=0x80; // Valeur à charger dans le registre adressé
```

→ *définir la fréquence de la sortie PWM1:*

Cette fréquence dépend de la valeur chargée dans le registre "PR1"

```
T_IM_Asservissement.data[0]=0x23; // adresse du registre PR1 en écriture
 (doc MCP25050 p15) 07H + décalage = 07H + 1CH = 23H
T_IM_Asservissement.data[1]=0xFF; // Masque sur le registre (doc MCP25050 p32)
T_IM_Asservissement.data[2]=0xFF; // On chargera 255 dans le registre
```

La fréquence du quartz implanté sur la carte "asservissement" étant égale à 16Mhz, la fréquence du signal PWM1 sera donc égale à : $F_{PWM} = 16.10^6/4.256 = 15,6$ KHz

Ce qui est une fréquence correcte pour piloter le moteur en PWM (fréquence sensiblement inaudible).

→ *initialiser la sortie GP3 en sortie PWM2: (variation de la vitesse du moteur dans le sens négatif)*

D'après la notice technique du circuit MCP25050 (pages 30 à 32), la génération du signal PWM2 se fait à partie du "Timer2" et la fréquence de ce signal est choisie grâce au registre "T2CON" d'adresse 06_H (page 15 Doc MCP25050).

bit 7 =1 Validation du "Timer 2"
bits 5:4 seront mis à 0 pour avoir un coefficient de division de fréquence égal à 1 ("TMR2 prescaler value" = 1)

```
T_IM_Asservissement.data[0]=0x22; // Adresse du registre T2CON en écriture
 (doc MCP25050 p15) 06H + décalage = 06H + 1CH = 22H
T_IM_Asservissement.data[1]=0xB3; // Masque sur le registre (doc MCP25050 p32)
T_IM_Asservissement.data[2]=0x80; // Valeur à charger dans le registre adressé
```

→ *définir la fréquence de la sortie PWM2:*

Cette fréquence dépend de la valeur chargée dans le registre "PR2"

```
T_IM_Asservissement.data[0]=0x24; // adresse du registre PR2 en écriture
 (doc MCP25050 p15) 08H + décalage = 08H + 1CH = 24H
T_IM_Asservissement.data[1]=0xFF; // Masque sur le registre (doc MCP25050 p32)
T_IM_Asservissement.data[2]=0xFF; // On chargera 255 dans le registre
```

La fréquence du quartz implanté sur la carte "asservissement" étant égale à 16Mhz, la fréquence du signal PWM1 sera donc égale à : $F_{PWM} = 16.10^6/(4.256) = 15,6$ KHz (idem fréquence sur PWM1)

→ *changer le rapport cyclique de la sortie PWM1 (Commande du moteur dans le sens positif)*

```
T_IM_Asservissement.data[0]=0x25; //adresse du registre PWM1DCH en écriture
 (doc MCP25050 p15) 09H + décalage = 09H + 1CH = 25H
T_IM_Asservissement.data[1]=0xFF; //masque sur le registre (doc MCP25050 p33)
T_IM_Asservissement.data[2]=0x00; //On initialise tous les bits à 0 -> Commande nulle
donc vitesse nulle (jusqu'à 255 pour la commande Maxi donc vitesse maxi)
```

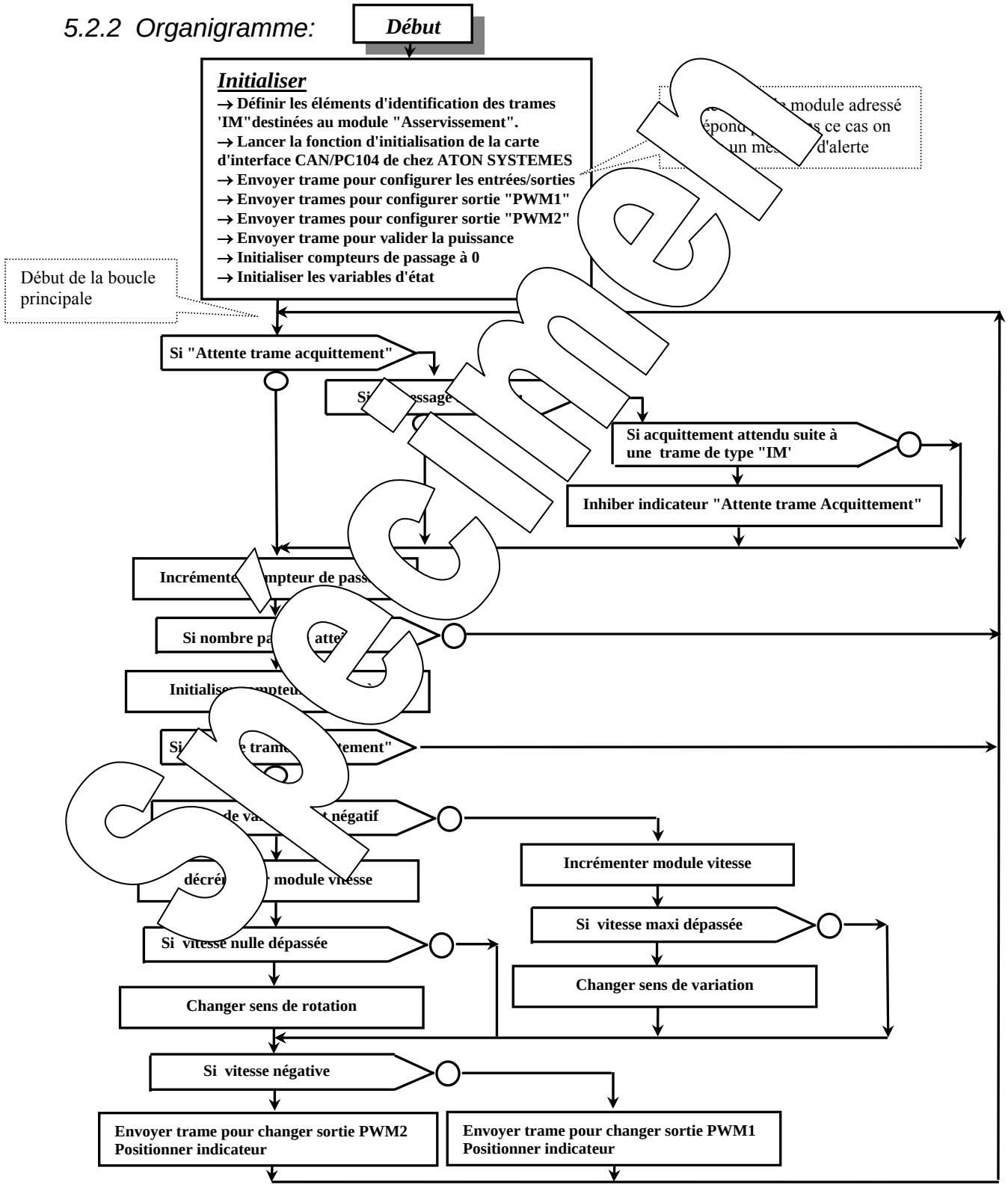
→ *changer le rapport cyclique de la sortie PWM2 (Commande du moteur dans le sens négatif)*

```
  T_IM_Asservissement.data[0]=0x26; //adresse du registre PWM2DCH en écriture
   (doc MCP25050 p15) 09H + décalage = 09H + 1CH = 25H
  T_IM_Asservissement.data[1]=0xFF; //masque sur le registre (doc MCP25050 p33)
  T_IM_Asservissement.data[2]=0x00; //On initialise tous les bits à 0 -> Commande nulle
donc vitesse nulle (jusqu'à 255 pour la commande Maxi donc vitesse maxi)
```

→ *pour valider le circuit de puissance*

```
  T_IM_Asservissement.data[0]=0x1E; //adresse du registre GPLAT en écriture
  T_IM_Asservissement.data[1]=0x10; //masque sur le registre (doc MCP25050 p27)
  T_IM_Asservissement.data[2]=0x10; //On initialise le bit 4 du registre à 1
```

5.2.2 Organigramme:



5.2.3 Programme en "C"

```
/*******************************************************************************
*       TP sur  EID210 / Réseau CAN - VMD  (Véhicule Multiplexé Didactique)             *
*******************************************************************************
*   TP n°5:  Commander le moteur de l'essuie glace avant
*-------------------------------------------------------
*   CAHIER DES CHARGES :
*  *********************
*  On souhaite que l'essuie réalise le cycle suivant:
*         - rotation droite avec une vitesse croissante de 0 à Vmax
*         - rotation droite avec une vitesse décroissante de Vmax à 0
*         - rotation gauche avec une vitesse croissante de 0 à Vmax
*         - rotation gauche avec une vitesse décroissante de Vmax à 0
*         etc ...
*  On affichera à l'écran, dans quelle partie du cycle on se situe, avec quel niveau de vitesse
*-------------------------------------------------------------------------------------------------
*  NOM du FICHIER :  CAN_VMD_TP5.C
* *****************
*************************************************************************************************/

// Déclaration des fichiers d'inclusion
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"
#include "Aton_can.h"
// Déclaration des variables
int Cptr_TimeOut;
// Pour les indicateurs divers (variables binaires)
union byte_bits Indicateurs;
#define I_Sens_Rotation Indicateurs.bit.b0
#define I_Sens_Variation Indicateurs.bit.b1
#define I_Autorise_Rot        Indicateurs.bit.b2
#define I_Attente_Trame_Acquittement Indicateurs.bit.b3
#define I_Message_Pb_Affiche Indicateurs.bit.b4
// Déclarations des diverses trammes de communication sur Bus CAN
Trame Trame_Recue;  // Pour trame qui vient d'être reçue par co          N
// Les trames de type "IM" (trame de commande)
Trame T_IM_Asservissement;    // Pour les trame de type "IM" à d          a carte "Asservissement"
                                        //  IM ->          ssage          de commande

//============================
// DEFINITION DES FONCTIONS
//============================

// De la fonction principale
//============================
main()
{
//  INITIALISATIONS
//------------------

int Cptr_Incrementation_Vitesse=0,Cptr_Af
BYTE Module_Vitesse=0;
/* Initialisation DU SJA1000 de la carte      */
Init_Aton_CAN();
clsscr();
I_Message_Pb_Affiche=0;
// Trame de type "IM" (tr     de comm de): B          ication
T_IM_Asservissement.trame_     o.     e0;
T_IM_Asservissement.trame_info     ;
T_IM_Asservissement.trame_inf     mp.d     ;
T_IM_Asservissement.trame_inf     mp.rt     ;
T_IM_Asservissement.ident.ext     en     ate     ent=Ident_T_IM_Asservissement;
// Pour définir des Entrées/So
T_IM_Asservissement.data[0]=0x1     // Ad     du registre GPDDR  (direction de E/S)
T_IM_Asservissement.da     xFF;          -> Bit 7 non concerné
T_IM_Asservissement.d          r ->  1 si Entrée et 0 si Sortie
         // GP7=fs B          Entrée; GP5=fcd Entrée;  GP4=ValidIP Sortie;
         // GP3=PW     cle,          1 Sortie; GP1=AN1 Entrée; GP0=AN0 Entrée;
do {Ecrire_Trame(     ervissement);
         Cptr_     =0;
         while(Cptr_     lwhile(Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<100));
         if(Cptr_Time          ==100)
                  {if(I          _Affiche==0)
                           {I          age_Pb_Affiche=1;
                                    10);
                           printf("Pas de reponse a la trame de commande en initialisation \n");
                           printf("Verifier si alimentation 12 V  est OK \n");}}
         Cptr_TimeOut=0;
// Pour mettre à 0 les sorties
T_IM_Asservissement.data[0]=0x1E;      // Adresse du registre GPLAT  (Registre I/O)
T_IM_Asservissement.data[1]=0x1C;      // Masque -> sorties GP4,3,2 sont consernées
T_IM_Asservissement.data[2]=0x00;      // Valeur ->  les 3 sorties à 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir sortie GP2 en PWM1
T_IM_Asservissement.data[0]=0x21;      // Adresse du registre T1CON
T_IM_Asservissement.data[1]=0xB3;      // Masque -> seuls bit 7;5;4;1;0 consernés
T_IM_Asservissement.data[2]=0x80;      // Valeur ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir fréquence signal sortie PWM1
T_IM_Asservissement.data[0]=0x23;      // Adresse du registre PR1
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0xFF;      // Valeur ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
```

```
T_IM_Asservissement.data[0]=0x22;      // Adresse du registre T2CON
T_IM_Asservissement.data[1]=0xB3;      // Masque -> seuls bit 7;5;4;1;0 consernés
T_IM_Asservissement.data[2]=0x80;      // Valeur ->  TMR2ON=1; Prescaler2=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir fréquence signal sortie PWM2
T_IM_Asservissement.data[0]=0x24;      // Adresse du registre PR2
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0xFF;      // Valeur ->  PR2=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour initialiser rapport cyclique PWM1 à 0
T_IM_Asservissement.data[0]=0x25;      // Adresse du registre PWM1DC
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0;         // Valeur ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour initialiser rapport cyclique PWM2 à 0
T_IM_Asservissement.data[0]=0x26;      // Adresse du registre PWM2DC
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0;         // Valeur ->  PWM2DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour Valider le circuit de puissance
T_IM_Asservissement.data[0]=0x1E;      // Adresse du registre GPLAT  (Registre I/O)
T_IM_Asservissement.data[1]=0x10;      // Masque -> sortie GP4 (ValidIP) est conservée
T_IM_Asservissement.data[2]=0x10;      // Valeur ->  ValidIP=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Masque pour les commandes IM futures
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont cons

// Initialisation des variables application
Module_Vitesse=0;
Indicateurs.valeur=0; // Sens positif, vitesse croissante
I_Attente_Trame_Acquittement=0;
Cptr_TimeOut=0;
clsscr();
// Pour afficher titre
gotoxy(1,2);
printf("    TP  n: 5    COMMANDER LE BALAI D'ESSUI GLACE    \n");
printf("    ******************************************** \n");


// BOUCLE PRINCIPALE
//******************
while(1)
         {if(I_Attente_Trame_Acquittement
                  {Cptr_TimeOut++;
                   if(Lire_Trame(&Trame_Recue)=
                    {if(Trame_Recue.ident.exten                 ficateu          Ident_T_AIM_Asservissement)
                           {// C'est bien l'"Acq                 dule          issement"
                             I_Attente_Trame_Acquit                 }
                    else {if(Cptr_TimeOut==
                              {clsscr();
                               gotoxy(1
                               printf(                de reponse  une t me de commande en cours de cycle \n");
                               printf(                arger le          amme puis relancer\n");
                               do{}whil                 }
                  }
         // Bloc pour fa       évoluer l'état
         Cptr_Incrementat       Vi
         if (Cptr_Incrementati                 000)
                  {// Il est          évoluer la vitesse moteur
                  Cptr_Incr       ation          sse=;
                  if(I_Sens_       tio          Si          doit diminuer la vitesse
                                    esse
                                    Vite          255)Module_Vitesse=0,I_Sens_Variation=0,I_Sens_Rotation=!I_Sens_Rotation;}
                  else {Module_                 on, on augmente la vitesse
                        if(Mo                 0)Module_Vitesse=255,I_Sens_Variation=1;}
                  i                 on)          on tourne en négatif
                                    ttente_Trame_Acquittement==0)
                                     {T_IM_Asservissement.data[0]=0x25;     // Adresse du registre PWM1DC
                                      T_IM_Asservissement.data[2]=Module_Vitesse;
                                               // En théorie : 0--> 0 tr/min et 255 ->5000 tr/min
                                      Ecrire_Trame(T_IM_Asservissement);
                                      I_Attente_Trame_Acquittement=1,Cptr_TimeOut=0;}}
                  else                Attente_Trame_Acquittement==0)
                                     {T_IM_Asservissement.data[0]=0x26;     // Adresse du registre PWM2DC
                                      T_IM_Asservissement.data[2]=Module_Vitesse;
                                      rire_Trame(T_IM_Asservissement);
                                      _Attente_Trame_Acquittement=1,Cptr_TimeOut=0;}}
                  } //          modification vitesse moteur
         // Blo       pour          her l'état
         Cptr
         if(C                 e==20000)
                    _Affichage=0;
                  gotoxy(1,10);
                  if(I_Sens_Rotation==0)
                           {printf("Trame pour commander le moteur en negatif               Valeur\n");
                            Affiche_Trame(T_IM_Asservissement);}
                  else  {printf("Trame pour commander le moteur en positif               Valeur\n");
                           Affiche_Trame(T_IM_Asservissement);}}
         } //  Fin boucle principale
} // Fin fonction principale
```

6 TP N°6: FAIRE BATTRE LE BALAI D'ESSUIE GLACE

6.1 Sujet

Objectifs :	- Définir les différentes trames interrogatives ou de commande à faire transiter par le réseau CAN en fonction d'une action souhaitée. - Commander un actionneur électrique (moteur à courant continu), dans les deux sens de rotation par l'intermédaire d'un pré-actionneur commandable par réseau CAN. - Acquérir l'état de capteurs TOR (fin de course) acessibles par réseau CAN et en déduire les actions à mener pour satisfaire le cahier des charges imposé.
Cahier des charges :	Après avoir fait tourner le moteur dans le sens positif (déplacement à droite) jusqu'à atteindre l'action sur le fin de course droit, l'essuie glace réalise le cycle continu suivant: → déplacement à gauche jusqu'à action sur le fin de course gauche → déplacement à droite jusqu'à action sur le fin de course droite → etc

Matériels et logiciels nécessaires :

Micro ordinateur de type PC sous Windows 95 ou ultérieur
Logiciel Editeur-Assembleur-Désassembleur
Si programmation en C, Compilateur ANSI C + Réf : EID210100
Carte processeur 16/32 bits à base de MC68332 et son environnement logiciel
(Editeur-CrossAssembleur-Désassembleur) Réf: EID210000
Carte réseau CAN PC/104 de chez APPLICATION SYSTEMES Réf : EID004000
-1 module électronique "assemblé" Réf: EID052000
- la partie opérative de l'essuie glace Réf: EID053000
Câble de liaison PC/104 par défaut cable RS232, Réf: EGD000003
Alimentation 220V AC 8V, pour l'alimentation de l'unité centrale Réf: EGD000001
Alimentation 12V pour l'alimentation des modules CAN (réseau "énergie").

Documents annexes

6.2 Eléments de solution

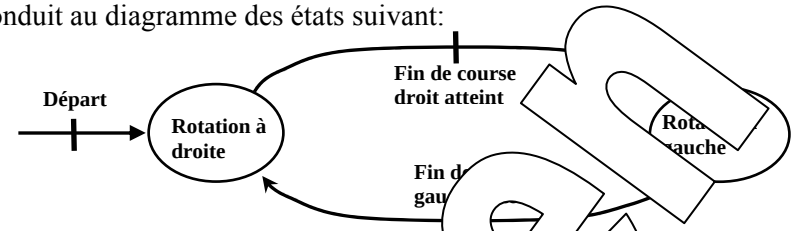
6.2.1 Analyse

Principe:

Le module d'interface CAN mis en œuvre dans ce TP est le module repéré "Asservissement".
Ce module permet la commande d'un moteur à courant continu 24V/ 1A, dans les deux sens de rotation, en mode "PWM" (modulation de largeur d'impulsion). Cette possibilité a été expérimentée dans le TP5.
Il permet d'acquérir 3 entrées TOR sur lesquelles on pourra connecter les capteurs de fin de courses:

→ fcd (fin de course droit) relié à l'entrée GP5 du contrôleur CAN
→ fcg (fin de course gauche) relié à l'entrée GP6 du contrôleur CAN
→ fs (fin de surcource) relié à l'entrée GP7 du contrôleur CAN

Le cycle demandé conduit au diagramme des états suivant:



Trames de configuration et de commande (type "IM") - TP5
Acquérir l'état des fins de course:

Définition de la trame interrogative permettant de c état des fins de course:
Dans ce cas, la trame envoyée par le contrôleur (Circu 1000 sur carte CAN_PC104) sera vue par le récepteur (circuit MCP25050 sur module) com "Information Request Message", avec la fonction "Read register" (voir documentation technique MPC pages 22).

D'après le tableau donné page 22 de la notice du 25050, l'identificateur lui-même contiendra l'adresse du registre lu. Cette adresse est placée sur les ID15 de l'identificateur en mode étendu (bits qui seront réceptionnés et placés dans le registre RXBE Le registre concerné est GPPIN d'adresse 1Eh " (voir documentation technique du MPC pages
D'autre part, les trois bits de poi oïdes de l'identificateur en mode étendu devront être positionnés à 1.
L'identificateur défini dans le ch e 1 dev nc être complété comme suit:

00 84 xx xx E (Seuls 29 bits sont pris en compte)

D'après tableau	Adr registre	Code fonction

→ Définition de variables urée us le modèle "**Trame**":

```
Trame T_IRM_ tio // Trame destinée à l'interrogation du module asservissement pour acquérir Fins de Courses.
```

Remarque: L ble st ée **T_IRM_Acquisition_FC** comportera 5 octets utiles seulement, 1octet pour tra et 4 pou ntificateur en mode étendu (qui comprendra l'adresse du registre concerné par l

→ des ts éléments de la variable structurée " **Lecture_FC** "

```
T_IRM_Acquisi _FC.t e_info.registre=0x00;  //On initialise tous les bits à 0
T_IRM_Acquisi _FC.trame_info.champ.extend=1; //On travaille en mode étendu
T_IRM_Ac n_FC.trame_info.champ.dlc=0x01; //Il y aura 1 octet de données demandé
T_IRM_A on _FC.ident.extend.identificateur.ident=0x00841E07;
```

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 de réponse). Ce paramètre est l'image des entrées. On récupère donc l'état des différents capteurs.

Définition de variables structurées images de l'état des fins de course

La trame reçue en réponse à cette trame interrogative comportera en data[0], l'état des fins de course. On recopie cette donnée reçue dans une variable image.

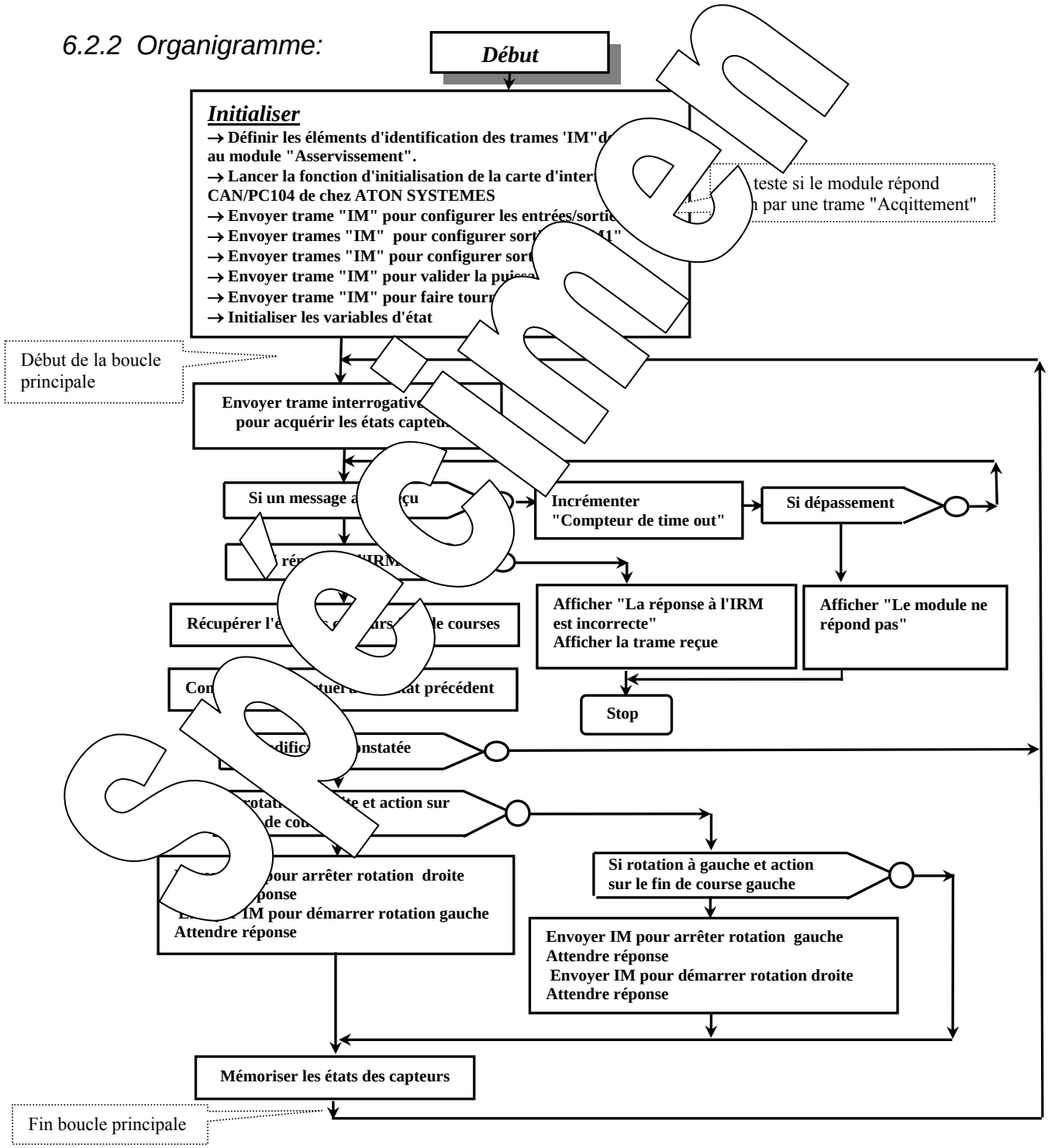
```
union byte_bits FC;
#define Etats_FC FC.valeur // Pour l'ensemble des états fins de courses
#define fs FC.bit.b7  // Pour fin de surcourse
#define fcg FC.bit.b6 // Pour fin de course gauche
#define fcd FC.bit.b5 // Pour fin de course droit
```

Afin de pouvoir détecter les changements d'état des capteurs, on mémorise l'état dans une deuxième variable structurée -> Etat_mémorisé

```
union byte_bits FC_Mem; // Pour fins de courses mémorisés
#define Etats_FC_Mem FC_Mem.valeur  // Pour l'ensemble des états mémorisés
```

Si l'état d'une variable acquise est différente de sa valeur mémorisée, c'est qu'il y a eu un changement d'état. On en déduit donc les actions à mener.

6.2.2 Organigramme:



6.2.3 Programme en "C"

```
        /*****************************************************************************
        *       TP sur  EID210 / Réseau CAN - VMD  (Véhicule Multiplexé Didactique)
        *****************************************************************************
        *   TP n°5:  Commander le moteur de l'essuie glace avant
        *-----------------------------------------------------
        *   CAHIER DES CHARGES :
        *  *********************
        *  On souhaite que l'essuie réalise le cycle suivant:
        *       - rotation droite avec une vitesse croissante de 0 à Vmax
        *       - rotation droite avec une vitesse décroissante de Vmax à 0
        *       - rotation gauche avec une vitesse croissante de 0 à Vmax
        *       - rotation gauche avec une vitesse décroissante de Vmax à 0
        *       etc ...
        *  On affichera à l'écran, dans quelle partie du cycle on se situe, avec quel niveau de vitesse
        *-----------------------------------------------------------------------------------------------
        *  NOM du FICHIER :  CAN_VMD_TP5.C
        * *****************
        *****************************************************************************/
// Déclaration des fichiers d'inclusion
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"
#include "Aton_can.h"
// Déclaration des variables
int Cptr_TimeOut;
// Pour les indicateurs divers (variables binaires)
union byte_bits Indicateurs;
#define I_Sens_Rotation Indicateurs.bit.b0
#define I_Sens_Variation Indicateurs.bit.b1
#define I_Autorise_Rot   Indicateurs.bit.b2
#define I_Attente_Trame_Acquittement Indicateurs.b
#define I_Message_Pb_Affiche Indicateurs.bit.b4
// Déclarations des diverses trammes de communic                CAN
Trame Trame_Recue;       // Pour trame qui vie      re r      controleur CAN
// Les trames de type "IM" (trame de      nde)
Trame T_IM_Asservissement;       //      les tra      ne "IM      destination de la carte "Asservissement"
                                 // IM ->          essage -> trame de commande


//============================
// DEFINITION DES FONCTIONS
//============================

// De la fonction principale
//============================
main()
{
//  INITIALISATIONS
//------------------

int Cptr_Incrementation_V     se=0      _Affichage=0;
BYTE Module_Vitesse=0;
/* Initialisation DU SJA10          car      104 */
Init_Aton_CAN();
clsscr();
I_Message_Pb_Affi
// Trame de type "                  commande): Données d'identification
T_IM_Asservisse      rame_      gistre=0x00;
T_IM_Asservisse      ame_in      mp.extend=1;
T_I      sseme      in      mp.dlc=0x03;
T      issement,      amp.rtr=0;
T      sse.vis      nt.id      nd.identificateur.ident=Ident_T_IM_Asservissement;
      ur dé      ntre      ies
T                  ta[0]                  // Adresse du registre GPDDR  (direction de E/S)
T_                 [1]=0x                 // Masque -> Bit 7 non concerné
T_IM_      sement      [2]=0xE3; // Valeur ->  1 si Entrée et 0 si Sortie
        // GP7=fs R      ; GP6=fcg Entree; GP5=fcd Entrée;  GP4=ValidIP Sortie;
        // G      tie; GP2=PWM1 Sortie; GP1=AN1 Entrée; GP0=AN0 Entrée;
do {Ecrire_T      Asservissement);
         Cptr_      ut=0;
         do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<100));
         if(Cptr_TimeOut==100)
                {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==100);
// Pour mettre à 0 les sorties
T_IM_Asservissement.data[0]=0x1E;         // Adresse du registre GPLAT  (Registre I/O)
T_IM_Asservissement.data[1]=0x1C;         // Masque -> sorties GP4,3,2 sont consernées
T_IM_Asservissement.data[2]=0x00; // Valeur ->  les 3 sorties à 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
```

```
// Pour définir sortie GP2 en PWM1
T_IM_Asservissement.data[0]=0x21;         // Adresse du registre T1CON
T_IM_Asservissement.data[1]=0xB3;         // Masque -> seuls bit 7;5;4;1;0 consernés
T_IM_Asservissement.data[2]=0x80; // Valeur ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir fréquence signal sortie PWM1
T_IM_Asservissement.data[0]=0x23;         // Adresse du registre PR1
T_IM_Asservissement.data[1]=0xFF;         // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0xFF; // Valeur ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir sortie GP3 en PWM2
T_IM_Asservissement.data[0]=0x22;         // Adresse du registre T2CON
T_IM_Asservissement.data[1]=0xB3;         // Masque -> seuls bit 7;5;4;1;0 consernés
T_IM_Asservissement.data[2]=0x80; // Valeur ->  TMR2ON=1; Prescaler2=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir fréquence signal sortie PWM2
T_IM_Asservissement.data[0]=0x24;         // Adresse du registre PR2
T_IM_Asservissement.data[1]=0xFF;         // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0xFF; // Valeur ->  PR2=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour initialiser rapport cyclique PWM1 à 0
T_IM_Asservissement.data[0]=0x25;         // Adresse du registre PWM1DC
T_IM_Asservissement.data[1]=0xFF;         // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0;    // Valeur ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour initialiser rapport cyclique PWM2 à 0
T_IM_Asservissement.data[0]=0x26;         // Adresse du registre PWM2DC
T_IM_Asservissement.data[1]=0xFF;         // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0;    // Valeur ->  PWM2DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour Valider le circuit de puissance
T_IM_Asservissement.data[0]=0x1E;         // Adresse du registre PORTB (Registre I/O)
T_IM_Asservissement.data[1]=0x10;         // Masque -> seule la GP4 (VALIDIP) est conserné
T_IM_Asservissement.data[2]=0x10; // Valeur -> VALIDIP=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attente réponse
// Masque pour les commandes IM futures
T_IM_Asservissement.data[1]=0xFF;                 masque                les bits sont consernés

// Initialisation des variables application
Module_Vitesse=0;
Indicateurs.valeur=0; // Sens positif vitesse croissante
I_Attente_Trame_Acquittement=0;
Cptr_TimeOut=0;
clsscr();
// Pour afficher titre
gotoxy(1,2);
printf("    TP  n: 5    COMMANDE BRAS ROBOTISE GLACE    \n");
printf("    ********** ***********  ******** \n");


// BOUCLE PRINCIPALE
//******************
while(1)
        {if(I_Attente_Trame_Acquittement==1)
                {Cptr_TimeOut++;
                if(Lire_Trame(&Trame_Recue)==1)
                if(Trame_Recue.ident.extend.identificateur.ident==Ident_T_AIM_Asservissement)
                        {// Si c'est bien l'"Acq IM" du module "Asservissement"
                        I_Attente_Trame_Acquittement=0;}}
                else if(Cptr_TimeOut==65000)
                        {clsscr();
                        gotoxy(1,10);
                        printf(" Pas de reponse a une trame de commande en cours de cycle \n");
                        printf(" Recharger le programme puis relancer\n");
                        do{}while(1);}}
                }
        // Bloc faisant évoluer l'état
        Cptr_Incrementation_Vitesse++;
        if (Cptr_Incrementation_Vitesse==2000)
                {// Il est temps de faire évoluer la vitesse moteur
                Cptr_Incrementation_Vitesse=0;
                if(I_Sens_Variation) // Si on doit diminuer la vitesse
                        {Module_Vitesse--;
```

```
if(Module_Vitesse==255)Module_Vitesse=0,I_Sens_Variation=0,I_Sens_Rotation=!I_Sens_Rotation;}
                else {Module_Vitesse++; // Sinon, on augmente la vitesse
                        if(Module_Vitesse==0)Module_Vitesse=255,I_Sens_Variation=1;}
                if(I_Sens_Rotation) // Si on tourne en négatif
                        {if(I_Attente_Trame_Acquittement==0)
                                {T_IM_Asservissement.data[0]=0x25;        // Adresse du registre PWM1DC
                                 T_IM_Asservissement.data[2]=Module_Vitesse;
                                        // En théorie : 0--> 0 tr/min et 255 ->5000 tr/min
                                 Ecrire_Trame(T_IM_Asservissement);
                                 I_Attente_Trame_Acquittement=1,Cptr_TimeOut=0;}}
                else    {if(I_Attente_Trame_Acquittement==0)
                                {T_IM_Asservissement.data[0]=0x26;        // Adresse du registre PWM2DC
                                 T_IM_Asservissement.data[2]=Module_Vitesse;
                                 Ecrire_Trame(T_IM_Asservissement);
                                 I_Attente_Trame_Acquittement=1,Cptr_Ti      ;}}
                } // Fin modification vitesse moteur
        // Bloc pour afficher l'état
        Cptr_Affichage++;
        if(Cptr_Affichage==20000)
                {Cptr_Affichage=0;
                gotoxy(1,10);
                if(I_Sens_Rotation==0)
                        {printf("Trame pour commander le mote                    Valeur\n");
                         Affiche_Trame(T_IM_Asservissement);
                else  {printf("Trame pour commander le moteur  os                Valeur\n");
                         Affiche_Trame(T_IM_Asservissement);}
        } //  Fin boucle principale
} // Fin fonction principale
```

7 TP N°7: REGULER LA VITESSE DU BALAI D'ESSUIE GLACE

7.1 Sujet

Objectifs :	- Acquérir le résultat d'une convertion Analogique -> Numérique via un réseau CAN. - Réaliser un échantillonnage à période imposée. - Expérimenter différents modes de commande (boucle ouverte, boucle fermée) d'un système analogique pilotable par réseau CAN. - Mettre en œuvre différents types de co[illegible] numérique (action proportionnelle, action intégrale)
Cahier des charges :	Etape 1: Commander le moteur "ess[illegible] vite[illegible] en boucle ouvert, avec pour grandeur de commande le [illegible]al i[illegible] du [illegible]tentiométre analogique présent sur la carte "Asservissement" Etape 2: Commander le mot[illegible]lace" en boucle fermée avec action de correction de ty[illegible]portion[illegible] consigne de vitesse est donnée par le potentiomètre [illegible] Et[illegible] Comma[illegible]le m[illegible]essuie glace" en boucle fermée avec action de type proportio[illegible] intég[illegible] NB[illegible] chaq[illegible]on affichera les résultats de conversion: [illegible]nsigne vitesse ([illegible]ésultat conversion entrée AN0 -> Potentiométre) [illegible]esure [illegible]se (résultat conversion entrée AN1 -> sortie F/U).

Matériels et logiciels né[illegible]air

Micro ordinateur [illegible]e PC[illegible]ows 95 ou ultérieur
Logiciel Editeur[illegible]r-Debogueur
Si programma[illegible]n C, c[illegible]ilateur GNU C/C++ Réf : EID210100
Car[illegible]esseu[illegible]2 bits [illegible]crocontrôleur 68332 et son environnement logiciel
([illegible]CrossAsse[illegible]bogueur)Réf: EID210000
[illegible]e rés[illegible]PC[illegible]chez ATON SYSTEMES Réf : EID004000
[illegible]ctron[illegible]sservissement" Réf: EID052000.
[illegible]partie[illegible]ative du système essuie glace Réf: EID053000
Câble de liai[illegible]ou à défaut câble RS232, Réf: EGD000003
Alimentati[illegible]8V, 1A pour l'alimentation de l'unité centrale Réf: EGD000001
Alimentation 12V pour l'alimentation des modules CAN (réseau "énergie").

Durée estimée : 3x3 heures

7.2 Eléments de solution étape n°1

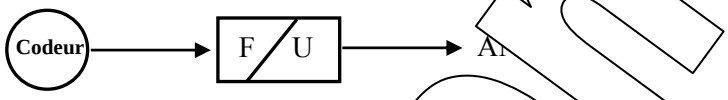
7.2.1 Analyse étape n°1

Principe:

Dans l'étape 1, la commande est dite "en boucle ouverte", c'est-à-dire que la commande du moteur ne dépend que de la grande de commande (valeur convertie de la tension issue du potentiométre appliquée sur l'entrée GP0 -> AN0 du circuit d'interface MC25050). Elle ne dépend pas de la mesure de la vitesse.

Remarques:

- La tension issue du potentiométre est dans la plage 0/5V. C'est une commande unipolaire, le moteur ne tournera que dans un seul sens de rotation.
- Une image unipolaire de la vitesse de rotation du moteur est accessible sur l'entrée analogique AN1. La tension disponible sur cette entrée résulte d'une convertion fréquence tension du signal codeur 1 voie accouplé au moteur.



Données techniques:

- Le codeur délivre impulsions par tr
- Le coefficient de transfert du convertisseur F/U v V/
- Le coefficient de réduction Vitesse Balai/Vitesse m

Trames de configuration et de commande (ty ") → ΣP5

Définition des trois octets de données associé ar:

→ activer et configurer la conver Analo vers N ique

D'après la notice technique du circu MC25050 s 34 à 37) :

il faut initialiser le registre ADCON0,

```
T_IM_Asservissement.data[0]=0x2A;   dresse du registre ADCON0 en écriture
 (doc MCP25050 p15) 0E_H + dé ge = 1C_H = 2A_H
T_IM_Asservissement.data[1 ]=0xC0; // Masque: Seul le bit 7 est concerné
T_IM_Asservissement.data[1 ]=0x80; // Valeur: ADON=1 -> Activation convertisseur
                                   t "prescaler rate" = 1:32
```

ainsi que le registre ADCON1:

```
T_IM_Asservissement.data[0]=    dresse du registre ADCON1 en écriture
 (doc MCP25050 p15 ata  0E_H + 1C_H = 2B_H
T_IM_Asservissement.data =0xFF; // Masque:  les 8 bits sont concernés
T_IM_Asservissement.dat ]=  3; // Valeur: (doc MCP25050 p36)
     b7=ADCS1=0; b6 =0 réquence Fosc/2
     b5=VCFG1 =Vs lage de tension d'entrée 0/+5V
     PCFG3:PC nvertion des entrées analogiques 1 et 0 ( sur GP1 et GP0)
```

A es rés de ersion A/N

uit, il udic utiliser l'IRM "Read A/D Regs, ce qui permet d'acquérir à la fois les états des e de cou les résultats de conversion des entrées analogiques (doc MCP25050 p22). L'id teur de dans le chapitre 1, pour une IRM est: 0x008400

→ Définiti bles structurées sous le modèle "**Trame**":

Trame T_IRM_Acquerir_FC_AN; // Trame destinée à l'interrogation du module asservissement pour acquérir fins de courses ainsi que les résultats de conversion A->N.

Remarque: La variable structurée **T_IRM_Acquerir_FC_AN** comportera 5 octets utiles seulement, 1octet pour trame_info et 4 octets pour l'identificateur en mode étendu

→ Accès et définition des différents éléments de la variable structurée " **T_IRM_Acquerir_FC_AN** "

```
T_IRM_ Acquerir_FC_AN.trame_info.registre=0x00;  //On initialise tous les bits à 0
T_IRM_ Acquerir_FC_AN .trame _info.champ.extend=1; //On travaille en mode étendu
T_IRM_ Acquerir_FC_AN .trame _info.champ.dlc=0x08; //Il y a 8 octets de données demandés
T_IRM_ Acquerir_FC_AN.ident.extend.identificateur.ident=0x00840000;
```

La trame réponse, suite à l'IRM, comportera en données associées 8 octets (`doc MCP25050 p22`):

- octet de rang 0 (data[0])→ valeur IOINTFL non utilile dans notre cas
- octet de rang 1 (data[1])→ valeur GPIO → Valeur des entrées sorties logiques
- octet de rang 2 (data[2])→ valeur AN0H → 8 bits MSB conversion entrée anologique 0
- octet de rang 3 (data[3])→ valeur AN1H → 8 bits MSB conversion entrée anologique 1
- octet de rang 4 (data[4])→ valeur AN10H → 2 fois 2 bits LSB conversion entrées ana. 1 et 0

Les 3 autres octets ne sont pas utiles dans notre application.

Le résultat de conversion est sur 10bits:

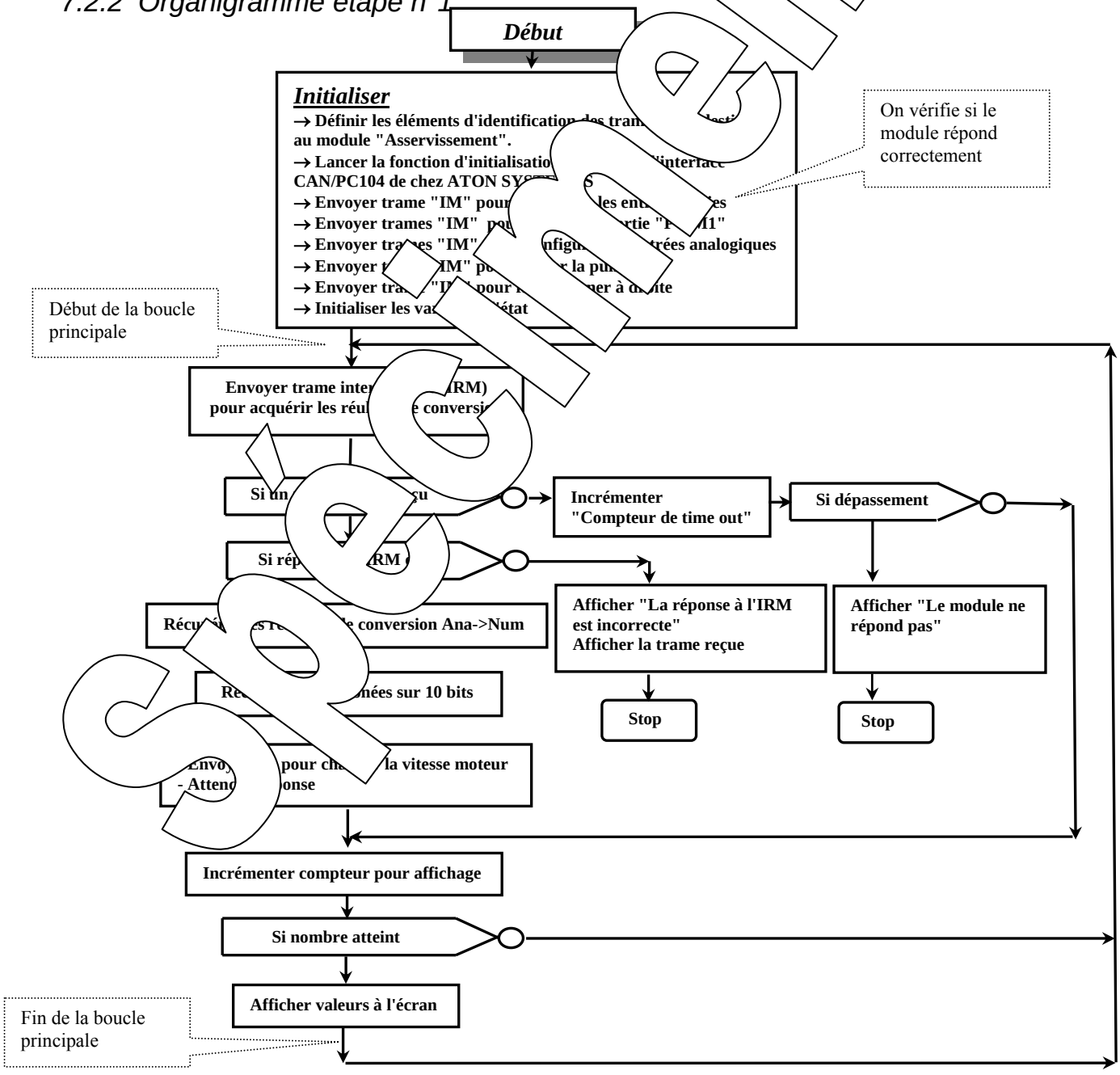
- pour résultat AN0 (potentiométre)

d9 d8 d7 d6 d5 d4 d3 d2	d1 d0 - - - - - -
AN0H ->data[2]	**AN10H -> data[4]**

- pour résultat AN1(capteur)

d9 d8 d7 d6 d5 d4 d3 d2	- - - - d1 d0 - -
AN1H ->data[3]	**AN10H -> data[4]**

7.2.2 Organigramme étape n°1



7.2.3 Programme en "C" de l'étape n°1

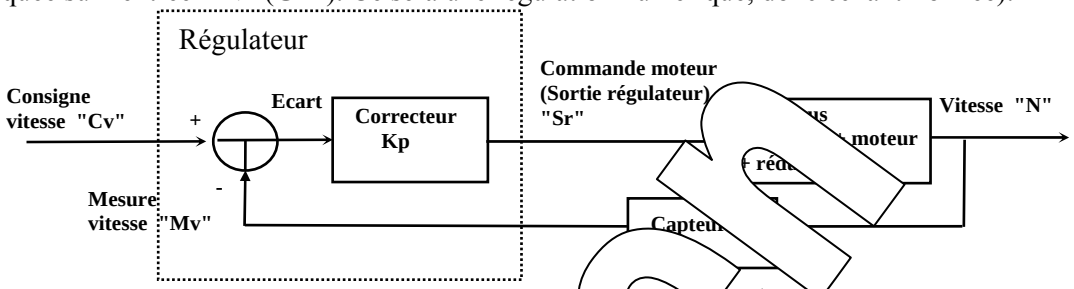
```
/*******************************************************************************
*       TP sur  EID210 / Réseau CAN - VMD  (Véhicule Multiplexé Didactique)
*******************************************************************************
*   TP n°7 :  Réguler la vitesse du balai d'essuie-glace
*------------------------------------------------------------------------------
*   CAHIER DES CHARGES :
*  *********************
*  Etape 1: Faire varier la vitesse du moteur avec le potentiomètre implanté sur le
*                  module "Asservissement"
*                  Afficher le résultat de conversion de l'entrée potentiométre ainsi que de
*                  l'entée image vitesse
*------------------------------------------------------------------------------
*  NOM du FICHIER :  CAN_VMD_TP7_1.C
* *****************
*******************************************************************************/

// Déclaration des fichiers d'inclusion
#include <stdio.h>
#include"Structures_Donnees.h"
#include"cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"
#include "Aton_can.h"

// Déclarations des diverses trames de communication
Trame Trame_Recue;  // Pour la trame qui vient d'etre reçue par le controleur
// Trmes de type "IM" (Input Message -> trame de commande)
Trame T_IM_Asservissement;    // Pour la commande du moteur
Trame T_IRM_Acquisition_FC_AN;         // Pour l'acquisition des entrées      et      course
// Déclaration des variables
// Pour les Indicateurs divers (variables binaires)
union byte_bits Indicateurs; // Structures de bits
#define I_Sens_Rotation Indicateurs.bit.b0
#define I_Attente_Reponse_IRM Indicateurs.bit.b1
#define I_Message_Pb_Affiche Indicateurs.bit.b2
// Pour les résultats de conversion
unsigned short S_Mesure_Vitesse,S_Val_Pot,S_Temp;
unsigned char AN0H,AN1H,AN10L;

//======================
// FONCTION PRINCIPALE
//======================
main()
{
//  INITIALISATIONS
//------------------
// Déclaration de variables locales à la fonction      le
// Les différents compteurs
unsigned int Cptr_Affichage,Cptr_TimeOut,Ctpr_Acquisiti
// Initilisation carte controleur réseau CA
Init_Aton_CAN();
// Effacer l'écran
clsscr();
// Initialisation des différentes trames      voi aux      s CAN
// Trame de type "IM" (tr e de commande,      ées d'id      cation
T_IM_Asservissement.tram      fo.registre=0x
T_IM_Asservissement.trame      fo.champ_extend
T_IM_Asservissement.trame_      o.      x03;
T_IM_Asservissement.trame_info
T_IM_Asservissement.ident.ext      dentifi      ur.ident=Ident_T_IM_Asservissement;
// Pour définir des Entrées/S      s
T_IM_Asservissement.data[0]=0      Adr      du registre GPDDR  (direction de E/S)
                                                    // doc MCP25050 Page 16
T_IM_Asservissement.data[1]=0xE      // Ma      -> Bit 7 non concerné
T_IM_Asservissement.da      xE3;      ->  1 si Entrée et 0 si Sortie
          // GP7=fs E      fcg E      fcd Entrée;  GP4=ValidIP Sortie;
          // GP3=PWM1      WM1 Sortie; GP1=AN1 Entrée; GP0=AN0 Entrée;
I_Message_Pb_Affic
do {Ecrire_Trame(      ervissement);      // C'est la première trame envoyée
          Cptr_T      // On teste si le module répond bien
          while(Cptr_      e_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<100));
          if(Cptr_Time
               {if(I_      ffiche==0)
                    I_Message_Pb_Affiche=1;
                    Cptr_      10);
                    pr      s de reponse a la trame de commande en initialisation \n");
                    print      érifier si alimentation 12 V  est OK \n");}}
          while(Cptr_TimeOu      );
clsscr();
// Pour mettre      les
T_IM_Asservis      =0x1E;      // Adresse du registre GPLAT  (Registre I/O)
T_IM_Asservis      1]=0x1C;      // Masque -> sorties GP4,3,2 sont consernées
T_IM_Asservissement.      ta[2]=0x00;      // Valeur ->  les 3 sorties à 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir sortie GP2 en PWM1
T_IM_Asservissement.data[0]=0x21;      // Adresse du registre T1CON
T_IM_Asservissement.data[1]=0xB3;      // Masque -> seuls bit 7;5;4;1;0 consernés
T_IM_Asservissement.data[2]=0x80;      // Valeur ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir fréquence signal sortie PWM1
T_IM_Asservissement.data[0]=0x23;      // Adresse du registre PR1
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0xFF;      // Valeur ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse

T_IM_Asservissement.data[0]=0x25;      // Adresse du registre PWM1DC
```

```
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=100;       // Valeur ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour Valider le circuit de puissance
T_IM_Asservissement.data[0]=0x1E;      // Adresse du registre GPLAT  (Registre I/O)
T_IM_Asservissement.data[1]=0x10;      // Masque -> sortie GP4 (ValidIP) est conserné
T_IM_Asservissement.data[2]=0x10;      // Valeur ->
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour activer les conversions Ana -> Num
T_IM_Asservissement.data[0]=0x2A;      // Adresse du registre ADCON0
T_IM_Asservissement.data[1]=0xF0;      // Masque -> bits 7..4 concernés
T_IM_Asservissement.data[2]=0x80;      // Valeur ->  ADON=1 et "prescaler rate"=1:32
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir le mode de conversion
T_IM_Asservissement.data[0]=0x2B;      // Adresse du registre ADCON1
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont concernés
T_IM_Asservissement.data[2]=0x0C;      // Valeur ->  voir doc MCP25050 page 36
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour acquérir les résultats de conversion A->N et fins de courses
// Trame de type "IRM" (trame interrogative): Données d'identification
T_IRM_Acquisition_FC_AN.trame_info.registre=0x00;
T_IRM_Acquisition_FC_AN.trame_info.champ.extend=1;
T_IRM_Acquisition_FC_AN.trame_info.champ.dlc=0x08; // On demande les valeurs de 8 oct
T_IRM_Acquisition_FC_AN.trame_info.champ.rtr=1;
T_IRM_Acquisition_FC_AN.ident.extend.identificateur.ident=Ident_T_IRM8_Asservissement;
Cptr_Affichage=0;
Ctpr_Acquisition=0;
// Pour afficher titre
gotoxy(1,2);
printf(" TP  n: 7_1    FAIRE VARIER LA VITESSE DU BALAI D'ESSUIE GLACE
printf(" ************************************************************* \n");
printf("     - en agissant sur le potentiometre sur carte Asservissement \n");
printf("     - en boucle ouverte (On ne tient pas compte de la mesure vi \n");
printf(" ************************************************************* \n");


// BOUCLE PRINCIPALE
//******************
while(1)
        {Ctpr_Acquisition++;
        if(Ctpr_Acquisition==10)
        {Ctpr_Acquisition=0;
        // Acquérir les résultats de conversion
        Ecrire_Trame(T_IRM_Acquisition_FC_AN); // De      me de      fin de course et AN
        Cptr_TimeOut=0;
        do{Cptr_TimeOut++;}while((Lire_Tr    Trame_Recu      Cptr_Ti    ut<20000));
         if(Cptr_TimeOut==20000)
                 {clsscr(),gotoxy(2,10);
                  printf(" Pas de reponse a la t      rrogati      uerir resultats de conversion \n");
                  printf("  Il faut recharger et re      progr      e \n");
                  do{}while(1);} // Stop
         else {if(Trame_Recue.ident.exte      ficateu      dent_T_IRM8_Asservissement)
                        // Si ident      correct
                         {AN0H =Tr      ue.data[2]  // On récupére les MSB de Tension potentiometre
                         AN1H =Tra      ue.data[2]  // On récupére les MSB de la mesure vitesse
                         AN10L =Tra      ue.data[      // On récupére les LSB AN1 et AN0
                         // Traiter      nées et      nstituer les résultats
                         S_Val_Pot=(u      he      H);      //Transfert avec transtypage
                         S_Va      S_Va      // Décaler de 2 bits vers poids forts
                         S_Val_Pot=(S_Val_Pot      ned      0L&0x0C); // Pour ne récupérer que les 2 bits AD1 et AD0
                         S_Val_Pot=S_Val_Pot|(S_Temp>>2);
                         S_Mesure_Vitesse=(unsigned short)(AN1H);        //Transfert avec transtypage
                         S_Mesure_Vitesse=S_Mesure_Vitesse      sur      ess      Mesure_Vitesse<<2; // Décaler de 2 bits vers poids forts
                         S_Temp=(unsigned short)(AN10L&0xC0); // Pour ne récupérer que les 2 bits AD1 et AD0
                         S_Mesure_Vitesse=S_Mesure_Vitesse|(S_Temp>>6);
                         T_      .data[0]=0x25;      // Adresse du registre PWM1DC (Charger cde vitesse)
                         T_IM      t.data[2]=(AN0H) ;  // Valeur ->  Commande vitesse
                         Ecrire_      M_Asservissement);
                         do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
                         // Fin acquisition et traitement
        Cptr_A      ;
        if(Cptr_      ==1000
                {
                 Cp
                 gotox
                 printf(      de l'entree Potentiométre:             \n");
                 printf("      la Commande moteur:                \n");
                 printf("  Val      'entree Mesure vitesse:            \n");
                 //      her grand
                 got      10);
                  p      Valeur de l'entree Potentiometre: %d\n",S_Val_Pot);
                 printf("  Valeur de la Commande moteur: %d \n",AN0H);
                 printf("  Valeur de l'entree Mesure vitesse: %d\n",S_Mesure_Vitesse);}
        } //      e principale
} // Fin fonction principale
```

7.3 Eléments de solution étape n°2

7.3.1 Analyse étape n°2

Principe:
Dans le cas d'une régulation en vitesse du moteur en mode proportionnel, la grandeur de commande est proportionnelle à la différence (Consigne vitesse - Mesure vitesse).
Pour le programme, la consigne vitesse sera le résultat de conversion de la tension potentiométrique appliquée sur l'entrée analogique AN0 (GP0)et la mesure vitesse, le résultat de conversion de la sortie du convertisseur F/U appliquée sur l'entrée AN1 (GP1). Ce sera une régulation numérique, donc échantillonnée).



Dans le cas d'une régulation par action proportionnelle, Sr ex sion: Sr = Kp.(Cv-Mv)
Le calcul se fera à intervalles de temps régulier appelés "péri onnage" et notée "Te".
Le coefficient Kp sera considéré dans le programme un e s réalité ses 4bits de poids faibles représenteont la partie fractionnaire: Kp=0x10 -> v =20h -> valeur =2; etc..
Kp=0x08 -> valeur =0,5; Kp=0x04 -> valeur Kp=0x valeur =0,125; etc…
En définitive Kp sera compris dans l'interval Kp ≤

Compléments de déclarations par pport à n°1
Réalisation de période d'échantillonn
Il est possible d'utiliser la capacité du circu 25050 envoyer spontanément et à intervalles de temps réguliers, un trame "Read A/D Re enant s "Data" les résultats de conversion (doc MCP25050 page 22).
Il faut pour cela initialiser la fon Scheduled t mission" (doc MCP25050 page 24).
Il faut pour cela initialiser, par d mes de "IM", les rgistre "STCON" et "IOINTEN".
Voir en partie "anal e" du TP5 la ition rames "T_IM_Asservissement.

→ *Pour définir la péri onnage (fréquence d'envoi des trames par le"'séquenceur")*
Cette fréquence dépend a v r chargée dans le registre "STON"

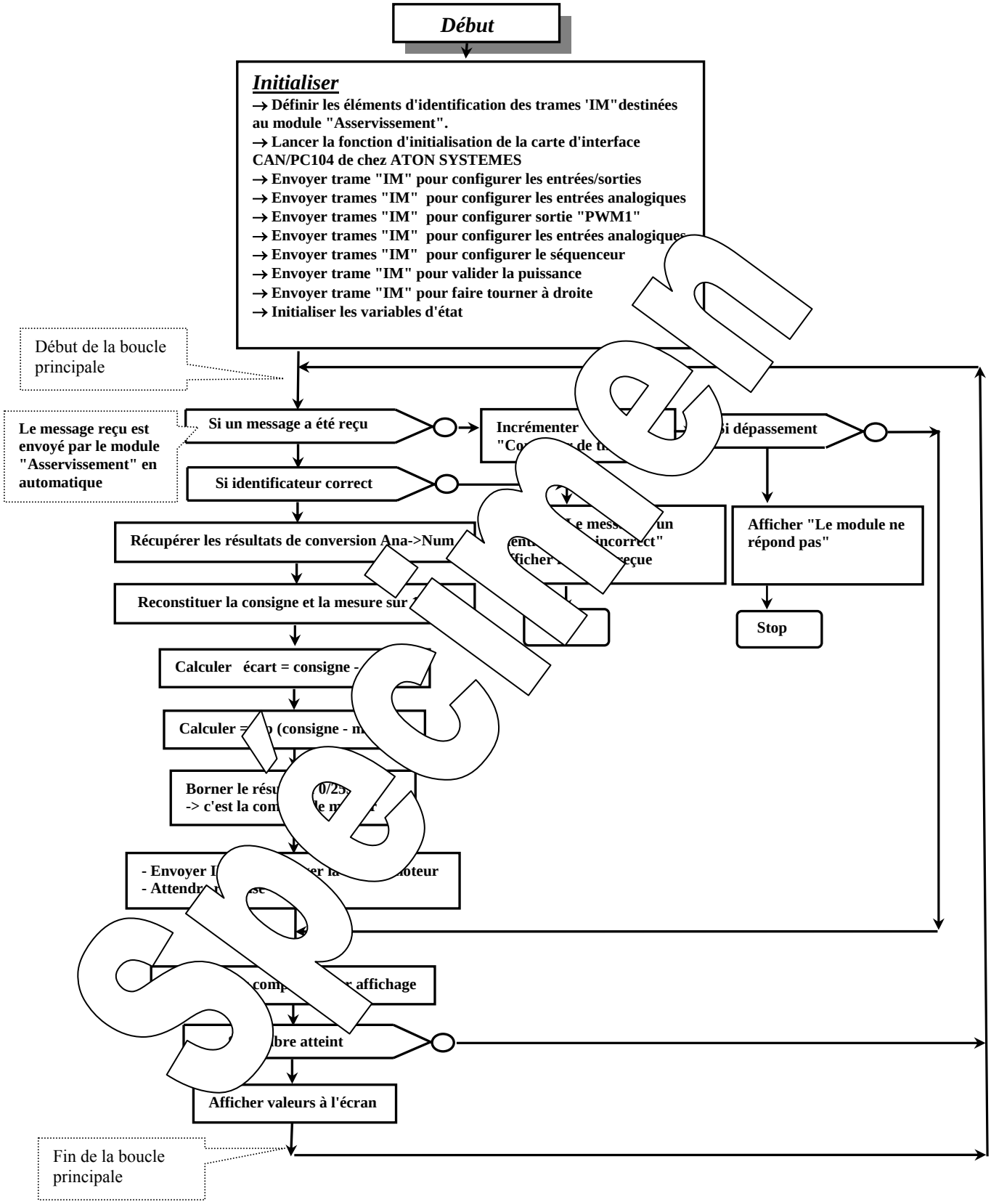
```
T_IM_Asservissemen 0]= C; // adresse du registre STON en écriture
 (doc MCP25050 p15)  ge = 10H + 1CH = 2CH
T_IM_Asservi da FF; // Masque: tous les bits sont concernés
T_IM_Asser me ta[2]=0xD2; // Valeur: (voir doc MCP25050 page 24).
     b7 -  = 1  our activer le séquenseur
     -> 1  ur trames à 8 octets (contenant les résultats de conversion)
     ,B4 =  ode de base = 16.4096.Tosc
     b3  00  ultiplieur de période = 3
```

I tz im sur la carte "asservissement" étant égale à 16Mhz (Tosc = $1/16.10^6$), la pé oi de mes se gale à $16.4096.3/16.10^6 = 12$ mS.

→ *Pour ac -conversion des convertisseurs Ana -> Num*
IL faut initialiser le registre "IOINTEN" et notamment les deux bits correspondant aux deux entrées analogiques utilisées dans cette application.

```
T_IM_Asservissement.data[0]=0x1C; // adresse du registre IOINTEN en écriture
 (doc MCP25050 p15) 00H + décalage = 00H + 1CH = 1CH
T_IM_Asservissement.data[1]=0x03; // Masque: seuls les bits 0 et 1 sont concernés
T_IM_Asservissement.data[2]=0x03; // Valeur: (voir doc MCP25050 page 27).
```

7.3.2 Organigramme étape n°2



7.3.3 Programme en langage "C"

```
/*****************************************************************************************
*       TP sur  EID210 / Réseau CAN - VMD  (Véhicule Multiplexé Didactique)
*****************************************************************************************
*   TP n 7:  Réguler la vitesse du balai d'essuie-glace
*----------------------------------------------------------------------------------------
*   CAHIER DES CHARGES :
*  ********************

          * Etape 2: Faire varier la vitesse du moteur avec le potentiomètre implanté sur le
*                   module "Asservissement"
*                   La commande du moteur se fait boucle fermé avec correcteur à action proportionnel
*                        Commande = Kp*Ecart = Kp * (Consigne - Mesure)
*----------------------------------------------------------------------------------------
*  NOM du FICHIER :  CAN_VMD_TP7_2.C
* *****************
*****************************************************************************************/
// Déclaration des fichiers d'inclusion
#include <stdio.h>
#include"Structures_Donnees.h"
#include"cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"
#include "Aton_can.h"
// Déclarations des diverses trames de communication
Trame Trame_Recue;  // Pour la trame qui vient d'etre reçue par le controleur
// Trmes de type "IM" (Input Message -> trame de commande)
Trame T_IM_Asservissement;    // Pour la commande du moteur
Trame T_IRM_Acquisition_FC_AN;         // Pour l'acquisition des entrées Ana et fins de
// Déclaration des variables
// Pour les Indicateurs divers (variables binaires)
union byte_bits Indicateurs; // Structures de bits
#define I_Sens_Rotation Indicateurs.bit.b0
#define I_Attente_Reponse_IRM Indicateurs.bit.b1
#define I_Message_Pb_Affiche Indicateurs.bit.b2
// Pour les résultats de conversion
unsigned short S_Mesure_Vitesse,S_Consigne,S_Temp;
unsigned char AN0H,AN1H,AN10L;
// Pour régulateur de vitesse
int Ecart,Resultat_Calcul;
unsigned char Cde_Moteur;
// Déclaration constante pour régulation
#define Kp 6       // Coefficient d'action proportionnell                tre un      er
                        //en fait valeur réelle Kp
//======================
// FONCTION PRINCIPALE
//======================
main()
{//  INITIALISATIONS
//------------------
// Déclaration de variables locales à la fonction pri
// Les différents compteurs
unsigned int Cptr_Affichage,Cptr_TimeOut,C                 ition;
// Initilisation carte controleur réseau
Init_Aton_CAN();
// Effacer l'écran
clsscr();
// Initialisation des dif rentes trames                i aux m       CAN
// Trame de type "IM" (t        de commande)                d'id          ication
T_IM_Asservissement.trame_                fo.registre=0x00
T_IM_Asservissement.trame_                =1;
T_IM_Asservissement.trame_inf
T_IM_Asservissement.trame_inf                mp.rt
T_IM_Asservissement.ident.ext                dent                teur  dent=Ident_T_IM_Asservissement;
// Pour définir des Entrées/S
T_IM_Asservissement.data[0]=0x                  // Adr       du registre GPDDR  (direction de E/S)
                                                                  // doc MCP25050 Page 16
T_IM_Asservissement.d                 FF;                    -> Bit 7 non concerné
T_IM_Asservissement.                                          r ->  1 si Entrée et 0 si Sortie
          // GP7=fs B                        Entrée; GP5=fcd Entrée;  GP4=ValidIP Sortie;
          // GP3=P                tie;          Sortie; GP1=AN1 Entrée; GP0=AN0 Entrée;
I_Message_Pb_Aff
do {Ec                Trame(                ervissem          // C'est la première trame envoyée
                    r_Time                                    // On teste si le module répond bien
          {Cptr_Time                        re_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<500));
          f(Cptr_TimeOu
                                (I_Me                Affiche==0)
                                                  _Pb_Affiche=1;
                                go                );
                                print                S de reponse a la trame de commande en initialisation \n");
                                printf(              Verifier si alimentation 12 V  est OK \n");}}
    }while(Cptr_TimeOu                );
clsscr();
// Pour mettre                ies
T_IM_Asservis                0]=0x1E;       // Adresse du registre GPLAT  (Registre I/O)
T_IM_Asservissement.data[1]=0x1C;       // Masque -> sorties GP4,3,2 sont consernées
T_IM_Asservissement.data[2]=0x00;       // Valeur ->  les 3 sorties à 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir sortie GP2 en PWM1
T_IM_Asservissement.data[0]=0x21;       // Adresse du registre T1CON
T_IM_Asservissement.data[1]=0xB3;       // Masque -> seuls bit 7;5;4;1;0 consernés
T_IM_Asservissement.data[2]=0x80;       // Valeur ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir fréquence signal sortie PWM1
T_IM_Asservissement.data[0]=0x23;       // Adresse du registre PR1
T_IM_Asservissement.data[1]=0xFF;       // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=0xFF;       // Valeur ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
```

```
// Pour initialiser rapport cyclique PWM1
T_IM_Asservissement.data[0]=0x25;      // Adresse du registre PWM1DC
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont consernés
T_IM_Asservissement.data[2]=100;       // Valeur ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour Valider le circuit de puissance
T_IM_Asservissement.data[0]=0x1E;      // Adresse du registre GPLAT  (Registre I/O)
T_IM_Asservissement.data[1]=0x10;      // Masque -> sortie GP4 (ValidIP) est conserné
T_IM_Asservissement.data[2]=0x10;      // Valeur ->
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour activer les conversions Ana -> Num
T_IM_Asservissement.data[0]=0x2A;      // Adresse du registre ADCON0
T_IM_Asservissement.data[1]=0xF0;      // Masque -> bits 7..4 concernés
T_IM_Asservissement.data[2]=0x80;      // Valeur ->  ADON=1 et "prescaler rate"=1:32
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir le mode de conversion
T_IM_Asservissement.data[0]=0x2B;      // Adresse du registre ADCON1
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont concernés
T_IM_Asservissement.data[2]=0x0C;      // Valeur ->  voir doc MCP25050 page 36
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour valider le séquenceur
T_IM_Asservissement.data[0]=0x2C;      // Adresse du registre STON
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont concernés
T_IM_Asservissement.data[2]=0xD2;      // Valeur ->  voir doc MCP25050 page 24
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour valider le séquencement sur entrées analogiques 0 et 1
T_IM_Asservissement.data[0]=0x2C;      // Adresse du registre STON
T_IM_Asservissement.data[1]=0x03;      // Masque -> tous les bits sont co
T_IM_Asservissement.data[2]=0x03;      // Valeur ->  voir doc MCP25050 p
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour futures trames IM dans la boucle principale (Cde moteur)
T_IM_Asservissement.data[1]=0xFF;      // Masque -> tous les bits sont co
// Pour acquérir les résultats de conversion A->N et fins de course
// Trame de type "IRM" (trame interrogative): Données d'identifi
T_IRM_Acquisition_FC_AN.trame_info.registre=0x00;
T_IRM_Acquisition_FC_AN.trame_info.champ.extend=1;
T_IRM_Acquisition_FC_AN.trame_info.champ.dlc=0x08; // On dem      va          8 rgistres
T_IRM_Acquisition_FC_AN.trame_info.champ.rtr=1;
T_IRM_Acquisition_FC_AN.ident.extend.identificateur.ident=          Asser      t; //
Cptr_Affichage=0,Ctpr_Acquisition=0,Cptr_TimeOut=0;
// Pour afficher titre
gotoxy(1,2);
printf("  TP  n: 7_2    FAIRE VARIER LA VI      U BALA      E GLACE      ");
printf("  ********************************      ********      ******* *** \n");
printf("      - en agissant sur le potentiomètre      rte As      ent \n");
printf("      - en boucle fermee en mode proport      = Kp(      Mesure) \n");
printf("  ****************************************      ****      ******* \n");
// BOUCLE PRINCIPALE
//******************
while(1)  {// Un trame donnant les états e      a inter      temps réguliers
           // Fonction 'Scéduleur' du mo      vissement      tivée
          if(Lire_Trame(&Trame_Recue)!=1      n trame ré      tat n      st pas arrivée ?
                    {Cptr_TimeOut++;
                     if(Cptr_TimeOut==100
                               {clsscr(),go          10);
                                printf(" Pas          s de conversion depuis trop longtemps\n");
                                pr          Il f          t relancer le programme \n");
                     else {Cptr_TimeOut=
                               if(Trame_          e.ident          tend.identificateur.ident==Ident_T_OB_Asservissement)
                                te          l l      ntificateur est correct
                                          _Re          ata[2];  // On récupére les MSB de Tension potentiometre
                                          me_Re          ata[3];  // On récupére les MSB de la mesure vitesse
                                AN          data[4]; // On récupére les LSB AN1 et AN0
                                // f          nées et reconstituer les résultats
                                onsig          igned short)(AN0H);     //Transfert avec transtypage
                                          igne=S_Consigne<<2;    // Décaler de 2 bits vers poids forts
                                          (unsigned short)(AN10L&0x0C); // Pour ne récupérer que les 2 bits AD1 et AD0
                                S_          e=S_Consigne|(S_Temp>>2);
                                S_M          Vitesse=(unsigned short)(AN1H);        //Transfert avec transtypage
                                S_          Vitesse=S_Mesure_Vitesse<<2;  // Décaler de 2 bits vers poids forts
                                          unsigned short)(AN10L&0xC0); // Pour ne récupérer que les 2 bits AD1 et AD0
                                          ure_Vitesse=S_Mesure_Vitesse|(S_Temp>>6);
                                          uler la grandeur de commande
                                          Consigne - S_Mesure_Vitesse;
                                Res          cul = (Kp*Ecart)>>4;
                                if(Res          _Calcul>255)Resultat_Calcul=255;
                                if(Resultat_Calcul<0)Resultat_Calcul=0;
                                Cde_Moteur=(unsigned char)(Resultat_Calcul);
                                T_IM_Asservissement.data[0]=0x25;      // Adresse du registre PWM1DC (Charger cde vitesse)
                                T_IM_Asservissement.data[2]=Cde_Moteur;         // Valeur ->  Commande vitesse
                                Ecrire_Trame(T_IM_Asservissement);
                                do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
                               }} // Fin acquisition et traitement
          Cptr_Affichage++;
          if(Cptr_Affichage==5000)
                    {Cptr_Affichage=0;
                    // Afficher grandeurs
                    gotoxy(1,12);
                     printf("  Valeur de l'entree Consigne: %d\n",S_Consigne);
                     printf("  Valeur de l'entree Mesure vitesse: %d\n",S_Mesure_Vitesse);
                     printf("  Ecart = Consigne - Mesure: %d\n",Ecart);
                     printf("  Valeur de la commande moteur:%d\n",Cde_Moteur);
                     printf("  On doit vérifier la relation:\n");
                     printf("  Commande moteur = (Kp*Ecart)/16 avec Kp= %d\n",Kp);}
          } //  Fin boucle principale
} // Fin fonction principale
```

7.4 Eléments de solution étape n°3

7.4.1 Analyse étape n°3

Principe:

Dans le cas d'une régulation en vitesse du moteur en mode proportionnel + Intégrale, la grandeur de commande est fonction de l'écart noté "ε" (ε =Consigne - Mesure) à l'instant d'échantillonage mais aussi de l'écart à l'instant d'échantillonnage précédent, suivant les relations de récurence suivantes:

- action intégrale: $S_{I\,n} = K_I \,.\, \varepsilon_n + S_{I\,n-1}$

  avec → $S_{I\,n}$ valeur action intégrale à t = n.Te (Te période d'echantillonnage)

  → $S_{I\,n-1}$ valeur action intégrale à t = (n-1).Te (à l'échantillonnage précédent)

  → K_I coefficient d'action intégrale

  → ε_n écart à t = n.Te (Te période d'echantillonnage)

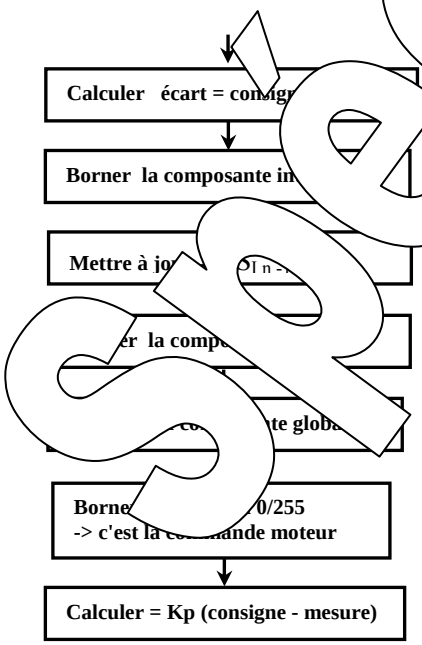
- action globale: $Sr_n = Kp.(S_{I\,n} + \varepsilon_n)$

Remarque:

- D'après l'expression de "$S_{I\,n}$", à chaque pas d'échantillonnage, cette valeur augmente de la valeur $K_I \,.\, \varepsilon_n$ (c'est une constante si ε_n est une constante → l'intégrale d'une constante est une rampe).
- Si la consigne est une constante, l'action intégrale impose en régime permanent (régime établi) $\varepsilon_n = 0$, ce qui entraîne que le signal de mesure devient égal au signal de consigne. Le coefficient de transfert en boucle fermée (Sortie/Consigne) devient égal à l'inverse du coefficient de transfert du capteur (Mesure/Sortie).
- Si pour une raison ou une autre, l'action intégrale provoque par exemple un écart (liaison de mesure en défaut, consigne de vitesse trop grande ..), il ne faut pas que la consigne atteigne des valeurs prohibitives. Pour cela il faut limiter $S_{I\,n}$ à $Sr_{Max}/Kp = 255/Kp$.

Les coefficients Kp et K_I seront considérés dans le programme comme un entier, mais réalité leurs 4bits de poids faibles représenteont la partie fractionnaire: K=0x00 -> valeur =1 ; K=20h -> valeur =2; etc..
K=0x08 -> valeur =0,5; K=0x04 -> valeur =0,25; K=0x02 -> valeur =0,125; etc…
En définitive Kp sera compris dans l'intervalle: $0 \leq Kp \leq 255$

7.4.2 Organigramme partiel étape n°3



7.4.3 Programme partiel étape n°3

```
// Une trame résultat est envoyée régulièrement par le module "Asservissement"
// Fonction 'Scéduleur activée'
if(Lire_Trame(&Trame_Recue)==1) // Un trame résultat est arrivée ?
{if(Trame_Recue.ident.extend.identificateur.ident=Ident_T_OB_Asservissement)
 // Si l'identificateur est correct
 AN0H =Trame_Recue.data[2];// On récupére les MSB image Tension
 AN1H =Trame_Recue.data[3];// On récupére les MSB de la mesure vitesse
 AN10L =Trame_Recue.data[4]; // On récupére les LSB AN1 et AN0
  // Traiter les données et reconstituer les résultats
 S_Consigne=(unsigned short)(AN0H);    //Transfert avec transtypage
 S_Consigne=S_Consigne<<2;    // Décaler de 2 bits vers poids forts
 S_Temp=(unsigned short)(AN10L&0x0C); // Pour récupérer les 2 bit LSB
 S_Consigne=S_Consigne|(S_Temp>>2);
 S_Mesure_Vitesse=(unsigned short)(AN1H); //Transfert avec transtypage
 S_Mesure_Vitesse=S_Mesure_Vitesse<<2;  // Décaler de 2 bits vers MSB
 S_Temp=(unsigned short)(AN10L&0xC0); // Pour récupérer les 2 bits LSB
 S_Mesure_Vitesse=S_Mesure_Vitesse|(S_Temp>>6);
 Ecart = S_Consigne - S_Mesure_Vitesse;
 SIn = KI*Ecart;              // Pour le calcule de l'actiion intégrale
 SIn = SIn>>4;
 SIn = SIn+SIn1;              // a ajouter à la valeur précédente
 if(SIn>SIn_max)SIn=SIn_max; // Pour limiter l'action intégrale
 if(SIn<-SIn_max)SIn=-SIn_max;
 SIn1=SIn;                    // Mise à jour de la grandeur mémorisée
 Resultat_Calcul = Kp*(Ecart+SIn);
 Resultat_Calcul = Resultat_Calcul>>4;
 if(Resultat_Calcul>255)Resultat_Calcul=255; // Pour limiter valeurs
 if(Resultat_Calcul<0)Resultat_Calcul=0; // par la commande du moteur
 Cde_moteur=(unsigned char)(Resultat_Calcul);
 T_IM_Asservissement.data[0]=0x25; // Adresse du registre PWM1DC
 T_IM_Asservissement.data[2]=Cde_moteur; // Valeur ->  Commande vitesse
 Ecrire_Trame(T_IM_Asservissement);
 do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
}} // Fin acquisition et traitement
```

8 TP N°8: FAIRE LA COMMANDE DU SYSTEME ESSUIE GLACE

8.1 Sujet

Objectifs :	- Développer une application temps réel (incorporant une régulation de vitesse) définie par un cahier des charges. - Expérimenter différents modes de commande (boucle ouverte, boucle fermée) d'un système analogique pilotable par réseau CAN. - Mettre en œuvre différents types de correcteur numérique (action proportionnelle, action intégrale)
Cahier des charges :	A intervalles de temps réguliers, on [illegible] le module sur lequel est connecté le commodo essuie glace afin [illegible] état. En fonction de l'état du commodo d'essuie glace, on commande le moteur (intermittent, position1, position2, ...) → Les différentes commandes imposées par la position de la molette commodo seront affichées [illegible] → On contrôlera également le fonctionnement du moteur.

Matériels et logiciels nécessaires :

Micro ordinateur de type PC sous Windows 95 ou supérieur
Logiciel Editeur-Assembleur-Débogueur
Si programmation en C, Compilateur GNU C [illegible] Réf : EID210100
Carte processeur 16 bits à microcontrôleur 68332 et son environnement logiciel
(Editeur-CrossAssembleur-Débogueur) Réf : EID210000
Carte réseau CAN PC/[illegible] de marque ATON SYSTEMES Réf : EID004000
- -1 module électronique "assemblement" Réf: EID052000.
- - la partie opérative [illegible] essuie glace Réf: EID053000
- -1 module "Interfaces diverses" Réf: EID050000.

(éventuellement) [illegible] commodo essuie glace Réf: ??

Câble de liaison [illegible] droit câble RS232, Réf: EGD000003
Alimentation [illegible] pour l'alimentation de l'unité centrale Réf: EGD000001
Alimentation [illegible] pour l'alimentation des modules CAN (réseau "énergie").

Durée estimée : [illegible] heures

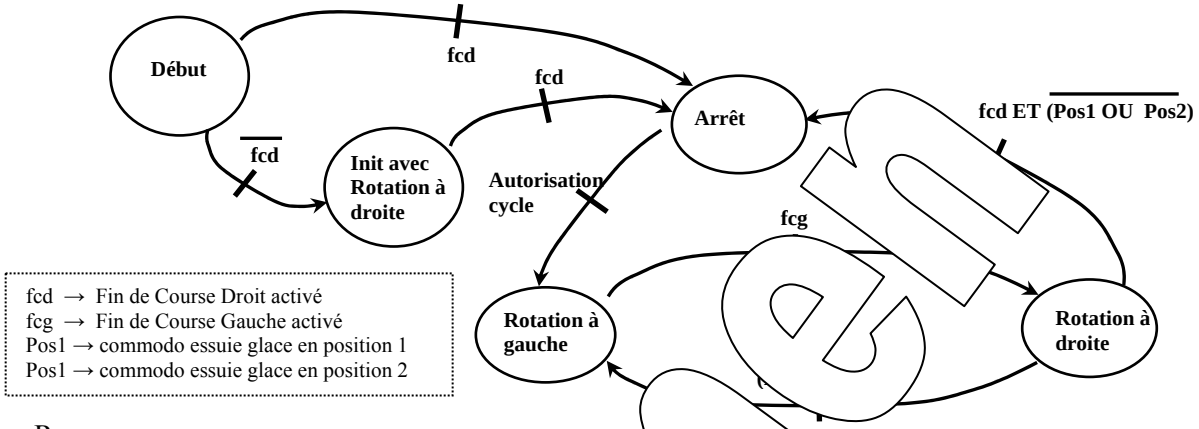
8.2 Eléments de solution

8.2.1 Analyse

Principe:

Dans ce TP le réseau CAN est constitué (outre la carte contrôleur, de deux modules
- un module repéré "Asservissement" sur lequel est connecté le moteur et les capteurs de fins de courses
- un module "8 entrées" sur lequel est éventuellement connecté un commodo d'essuie glace

Le cycle demandé conduit au diagramme des états suivant:



Remarques:

- Dans les deux états "Rotation Gauche" et "Ro oite" ... se dépend de la position commodo essuie glace: Position 1 ou intermittent → Vitesse ... 2 → Vitesse rapide.
- Si le commodo est dans la positio ... nterm ... une t ... temps met régulièrement à 1 la variable "Autorisation cycle". Cette dernière ... t remise a ... ctivati ... n de l'état "rotation gauche". L'intervalle de temps entre deux activation de "Autoris ... " dép ... la position de la molette du commodo.

Trames de configuration et de c ... de (ty ... ") du module asservissement → Idem TP5
Trames d'acquisition (type "IF ... des fins de c ... rses du module asservissement → Idem TP6
Trames de configuration (type ... ") du ... le 8 entrées "Comodo essuie glace"

L'identificateur déf ... dans le chap ... nom ... "IM" (Input Message -> Trame de commande) envoyé à la carte "Commodo esst ... gl ... : ... 0x05880000

→ Définition de variabl ... ruc ... s sous le modèle "**Trame**":

```
Trame T_IM_Commodo ... ;
```

→ Définition des éléme ... tatifi ... n de la variable structurée "**T_IM_Asservissement**"

```
T_IM_Commodo ... rame ... istre=0x00;  //On initialise tous les bits à 0
T_IM_Commodo ... .IM ... hamp.extend=1; //On travaille en mode étendu
T_IM_Commo ... .t ... nfo.champ.dlc=0x03; //Il y aura 3 octets de données
T_IM_Comm ... .iden ... tend.identificateur.ident=0x05880000;
```

→ *... et con ... ersion Analogique vers Numérique*

... s la notice tech ... u circuit MCP25050 (pages 34 à37) :

... ini ... egist ... CON0

```
... .data ... x2A; // Adresse du registre ADCON0 en écriture
( ... CP2505 ... 5) 0E_H décalage = 0E_H + 1C_H = 2A_H
T_IM_ Com ... .data[1]=0xF0; // Masque: Seul le bit 7 est concerné
T_IM_ C ... .data[2]=0x80; // Valeur: ADON=1 -> Activation convertisseur
```

et "presca ... ate" = 1:32

ainsi que le registre ADCON1.

```
T_IM_ Commodo_EG.data[0]=0x2B; // Adresse du registre ADCON1 en écriture
 (doc MCP25050 p15) 0F_H + décalage = 0E_H + 1C_H = 2B_H
T_IM_ Commodo_EG.data[1]=0xFF; // Masque:  les 8 bits sont concernés
T_IM_ Commodo_EG.data[2]=0xOE; // Valeur: (doc MCP25050 p36)
     b7=ADCS1=0; b6=ADCS0=0 → Fréquence Fosc/2
     b5=VCFG1=0; b4=VCFG0=0 → Plage de tension d'entrée 0/+5V
     PCFG3:PCFG0=1110       → Convertion de l'entrée analogique 0 ( sur GP0)
```

Trames d'acquisition (type "IRM" Input Request Message) de l'état commodo essuie glace:
L'identificateur défini dans le chapitre 1, pour un "IRM" envoyé à la carte "Commodo essuie glace"est :
0x05840000

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

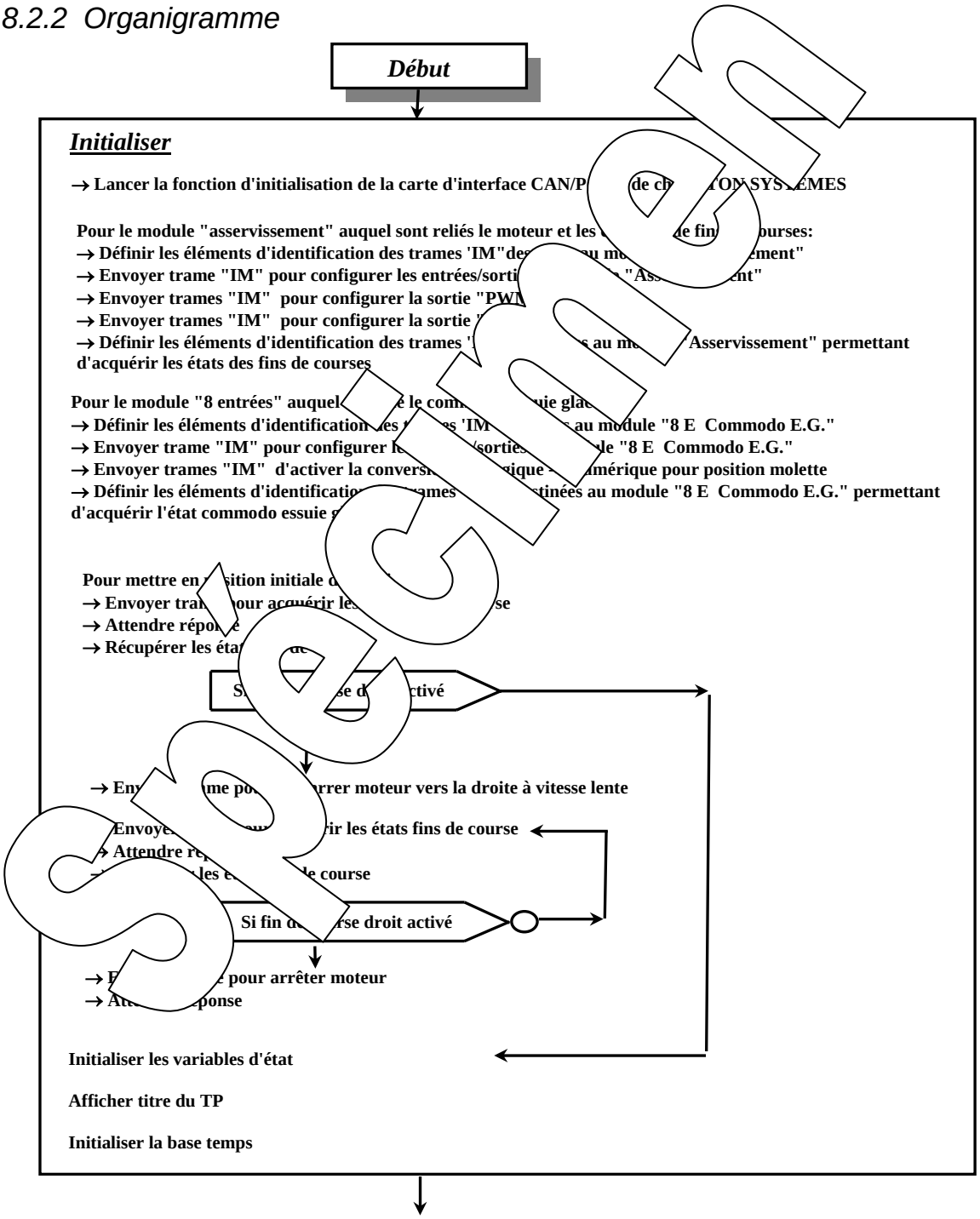
```
Trame T_IRM_Etat_Commodo_EG;      // Trame destinée à l'interrogation du module 8E pour acquérir état commodo.
```

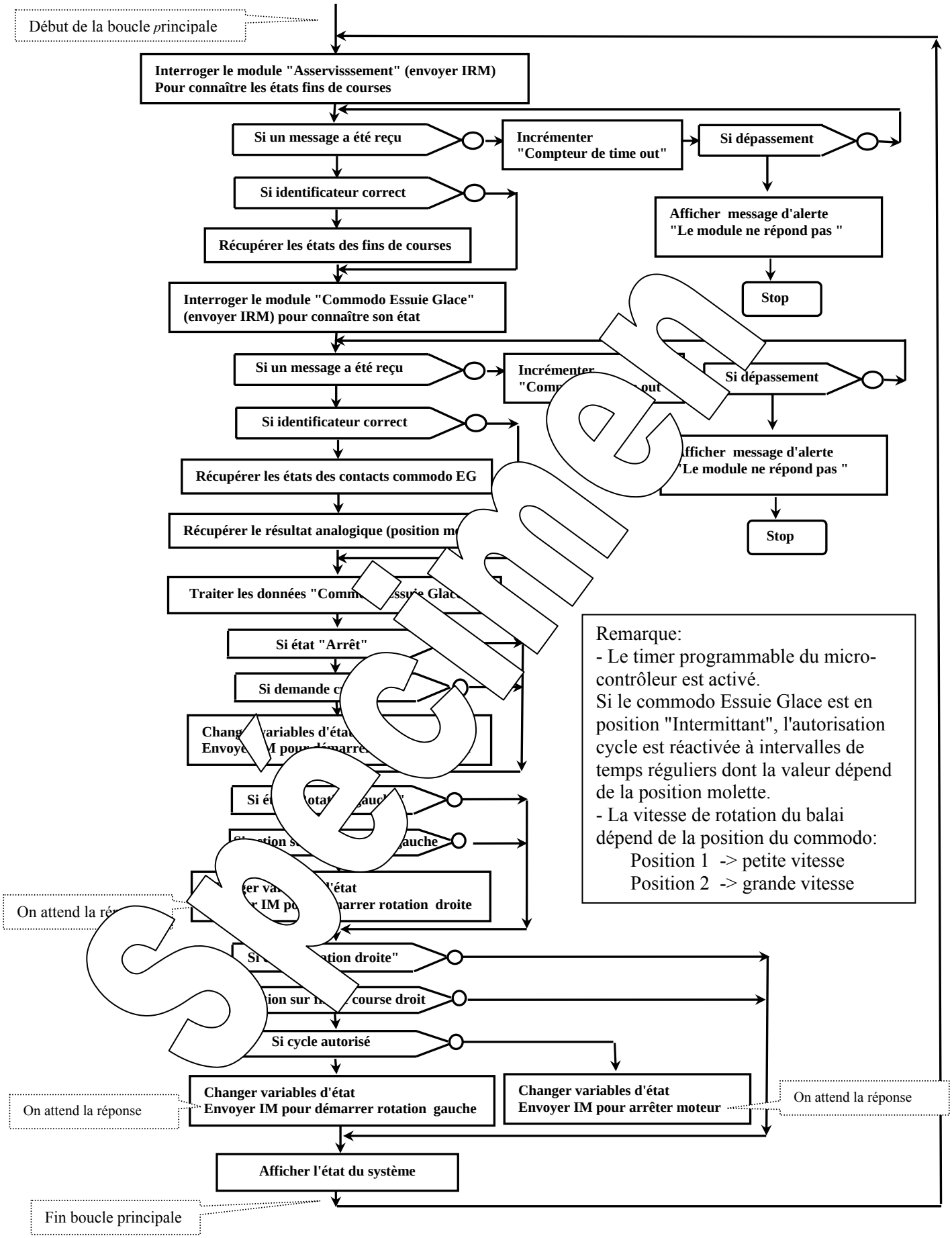
→ Accès et définition des différents éléments de la variable structurée " **Lecture_FC** "

```
T_IRM_ Etat_Commodo_EG.trame_info.registre=0x00;  //On initialise tous les bits à 0
T_IRM_ Etat_Commodo_EG.trame_info.champ.extend=1; //On travaille en mode étendu
T_IRM_ Etat_Commodo_EG.trame_info.champ.dlc=0x08; //Il y aura 8 octets de données
T_IRM_ Etat_Commodo_EG.ident.extend.identificateur.ident=0x05840000;
```

En réponse à cette tramme intérrogative, on récupére les états logiques dans la donnée de rang 1 et le résultat de conversion position molette dans la donnée de rang 2.

8.2.2 Organigramme




Début de la boucle principale
Interroger le module "Asservisssement" (envoyer IRM) Pour connaître les états fins de courses
Si un message a été reçu
Incrémenter "Compteur de time out"
Si dépassement
Afficher message d'alerte "Le module ne répond pas "
Stop
Si identificateur correct
Récupérer les états des fins de courses
Interroger le module "Commodo Essuie Glace" (envoyer IRM) pour connaître son état
Si un message a été reçu
Si dépassement
Si identificateur correct
Récupérer les états des contacts commodo EG
Stop
Si état "Arrêt"
Changer variables d'état
Envoyer IM pour démarrer rotation droite
On attend la réponse
Si cycle autorisé
Changer variables d'état
Envoyer IM pour démarrer rotation gauche
Changer variables d'état
Envoyer IM pour arrêter moteur
On attend la réponse
Afficher l'état du système
Fin boucle principale
Remarque:
- Le timer programmable du micro-contrôleur est activé.
Si le commodo Essuie Glace est en position "Intermittant", l'autorisation cycle est réactivée à intervalles de temps réguliers dont la valeur dépend de la position molette.
- La vitesse de rotation du balai dépend de la position du commodo:
Position 1 -> petite vitesse
Position 2 -> grande vitesse

8.2.3 Programme en "C"

```
        /*****************************************************************************
        *       TP sur  EID210 / Réseau CAN - VMD  (Véhicule Multiplexé Didactique)
        *****************************************************************************
        *   TP n°8:  Commande essuie glace avant
        *---------------------------------------------------------------------------
        *   CAHIER DES CHARGES :
        *  *********************
        *  On souhaite une commande normale de l'essuie glace à partir du commodo destiné
        *  à cet effet:
        *       - position 'arrêt'
        *       - position 'intermittant' -> le balai fait des "aller-retours" séparés par des
        *         dont la durée est réglée par le molette intégrée au commodo
        *       - position 'un' -> le balai fait des "aller-retours" à vitesse lente
        *       - position 'deux' -> le balai fait des "aller-retours" à vitesse rapide
        *  Dans le mode 'intermittent', l'intervalle de temps entre deux battements est générée
        *   par le 'temporisateur programmable intégré dans le micro-cont
        *  On affichera à l'écran, les états des diverses entrées commod
        *---------------------------------------------------------------------------
        *  NOM du FICHIER :  CAN_VMD_TP8.C
        * *****************
        *****************************************************************************/
// Déclaration des fichiers d'inclusion
#include <stdio.h>
#include"Structures_Donnees.h"
#include"cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"
#include "Aton_can.h"
// Déclaration des variables
// Pour les Indicateurs divers (variables binaires)
union byte_bits Indicateurs,FC; // Structures de bits
#define I_Autorise_Cycle Indicateurs.bit.b0                 aut       battement
#define I_Intermittent Indicateurs.bit.b1
#define I_Message_Pb_Affiche Indicateurs.bit.b2
#define Etat_Arret Indicateurs.bit.b3                       Eta       à l'arrêt en fin de course droite"
#define Etat_Rot_Droite Indicateurs.b    b4// E      lai en      on droite"
#define Etat_Rot_Gauche Indicateurs.     3// Eta      en r      on gauche"
// Pour les fins de course
#define Etat_FC FC.valeur // Pour l'ensemb      fins      se
#define fs FC.bit.b7      // Pour fin de su
#define fcg FC.bit.b6     // Pour fin de cours
#define fcd FC.bit.b5     // Pour fin d      se d
// Déclarations des diverses trames      nicatio
Trame Trame_Recue;        // Pour l      qui vient d      e reçue par le controleur
// Trames de type "IM" (Input Mess      > trame      ommande)
Trame T_IM_Asservissement;        /      r la co      e du moteur
Trame T_IM_Commodo_EG             /,      l'init      sation Commodo Essuie Glace
// Trames de type "IR      (Input Messag                  rrogative)
Trame T_IRM_Acquisitio     C;          / Po                tion de l'état des fins de courses
Trame T_IRM_Etat_Commodo_E                              l'acquisition de l'état Commodo Essuie Glace
// Varibles diverses
unsigned char Valeur_Anal      ue,C      ite      e,Tempo_Fin,Compteur_Passage_Irq,Compteur_Secondes;
// Pour les temporisation
// Déclaration constantes
#define Vitesse_Len      20
#define Vitesse_Ra
#define Tempo1 2
#define Tempo2
#define Tempo3
#def      po4 8
#d      o5 10
//      on d'interr                e de Temps"
//=========      ======      ===========
'rq
//      ute le
{if      tent)      i mode      rmittent actif
        {Compteur_P      e_Irq++;
         if(C      ssage_Irq==100)  // Une  Seconde s'est écoulée
                   eur_Passage_Irq=0;
                   pteur_Secondes++;
                  if(Compteur_Secondes>=Tempo_Fin)
                        {Compteur_Secondes=0;
                         I_Autorise_Cycle=1;}}}
} // Fin de la fonction d'interruption
//======================
// FONCTION PRINCIPALE
//======================
main()
{// Déclaration de variables locales à la fonction principale
int Cptr_Affichage=0,Cptr_TimeOut;
```

```
//  INITIALISATIONS
//------------------
// Pour initialiser la carte industrielle controleur CAN
Init_Aton_CAN();
clsscr(); // Pour effacer l'écran
// Trame de type "IM" (trame de commande): Données d'identification
T_IM_Asservissement.trame_info.registre=0x00;
T_IM_Asservissement.trame_info.champ.extend=1;
T_IM_Asservissement.trame_info.champ.dlc=0x03;
T_IM_Asservissement.trame_info.champ.rtr=0;

T_IM_Asservissement.ident.extend.identificateur.ident=Ident_T_IM_Asservissement;
// Pour définir des Entrées/Sorties
T_IM_Asservissement.data[0]=0x1F;         // Adresse du registre GPDDR  (direction de E/S) doc MCP25050 Page 16
T_IM_Asservissement.data[1]=0xEF;         // Masque -> Bit 7 non concerné
T_IM_Asservissement.data[2]=0xE3; // Valeur ->  1 si Entrée et 0 si Sortie
// GP7=fs Entrée;GP6=fcg Entree;GP5=fcd Entrée; GP4=ValidIP Sortie;GP3=PWM2 Sortie;GP2=PWM1 Sortie;GP1=AN1
Entrée;GP0=AN0 Entrée
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Asservissement);    // C'est la première trame envo
         Cptr_TimeOut=0;         // 'Asservissement' -> on teste s'il
         do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cpt
         if(Cptr_TimeOut==100)
                {if(I_Message_Pb_Affiche==0)
                        {I_Message_Pb_Affiche=1;
                         gotoxy(2,10), printf(" Pas de repons
                         printf("  Verifier si alimentation
    }while(Cptr_TimeOut==100);
clsscr(); // Pour effacer l'écran au cas ou message afiché
// Pour mettre à 0 les sorties
T_IM_Asservissement.data[0]=0x1E;         // Adresse du
T_IM_Asservissement.data[1]=0x1C;         // Masque ->
T_IM_Asservissement.data[2]=0x00; // Valeur ->  les 3 s
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendr
// Pour définir sortie GP2 en PWM1
T_IM_Asservissement.data[0]=0x21;         // Ad
T_IM_Asservissement.data[1]=0xB3;         // M
T_IM_Asservissement.data[2]=0x80; //
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); //
// Pour définir fréquence signal sortie PW
T_IM_Asservissement.data[0]=0x23;         //
T_IM_Asservissement.data[1]=0xFF;         //
T_IM_Asservissement.data[2]=0xFF; //
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); //
// Pour définir sortie GP3 en PWM2
T_IM_Asservissement.data[0]=0x22;         //
T_IM_Asservissement.data[1]=0xB3;         //
T_IM_Asservissement.data[2]=
Ecrire_Trame(T_IM_Asservis
do{}while(Lire_Tra
// Pour définir fréquence
T_IM_Asservissement.data[
T_IM_Asservissement.data[1
T_IM_Asservissement.data[2]=
Ecrire_Trame(T_IM_
do{}while(Lire_Tra
// Pour initialiser
T_IM_Asservis
T_IM_
T_
E
(
// Pour
T
T_IM_
T_IM_
Ecrire_Trame(
do{}while(Li
// Pour Val
T_IM_Asservissement.data[0]=0x1E;         // Adresse du registre GPLAT  (Registre I/O)
T_IM_Asservissement.data[1]=0x10;         // Masque -> sortie GP4 (ValidIP) est conserné
T_IM_Asservissement.data[2]=0x10; // Valeur ->  ValidIP=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Masque pour les commandes IM futures
T_IM_Asservissement.data[1]=0xFF;         // Masque -> tous les bits sont consernés
```

```
// Pour acquérir l'état des fin de course
// Trame de type "IRM" (trame interrogative): Données d'identification
T_IRM_Acquisition_FC.trame_info.registre=0x00;
T_IRM_Acquisition_FC.trame_info.champ.extend=1;
T_IRM_Acquisition_FC.trame_info.champ.dlc=1;
T_IRM_Acquisition_FC.trame_info.champ.rtr=1;
T_IRM_Acquisition_FC.ident.extend.identificateur.ident=Ident_T_IRM1_Asservissement   Demande état registre GPIN
// Pour initialiser le module "commmodo EG"
// Trame de type "IM" (trame de commande): Données d'identification
T_IM_Commodo_EG.trame_info.registre=0x00;
T_IM_Commodo_EG.trame_info.champ.extend=1;
T_IM_Commodo_EG.trame_info.champ.dlc=0x03; // On demande les valeurs de 8 rgistres
T_IM_Commodo_EG.trame_info.champ.rtr=0;
T_IM_Commodo_EG.ident.extend.identificateur.ident=Ident_T_IM_Commodo_EG; //
// Pour activer les conversions Ana -> Num
T_IM_Commodo_EG.data[0]=0x2A;     // Adresse du registre ADCON0
T_IM_Commodo_EG.data[1]=0xF0;     // Masque -> bits 7..4 concernés
T_IM_Commodo_EG.data[2]=0x80;     // Valeur ->  ADON=1 et "prescaler rate"
Ecrire_Trame(T_IM_Commodo_EG);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir le mode de conversion
T_IM_Commodo_EG.data[0]=0x2B;     // Adresse du registre ADCON1
T_IM_Commodo_EG.data[1]=0xFF;     // Masque -> tous les bits sont concer                    Entrée Analogique)
T_IM_Commodo_EG.data[2]=0x0E;     // Valeur ->  voir doc MCP25050 page 36 (
Ecrire_Trame(T_IM_Commodo_EG);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour acquérir les résultats de conversion A->N et états co       o EG
// Trame de type "IRM" (trame interrogative) Commodo EG: Donne      'id      ric      on
T_IRM_Etat_Commodo_EG.trame_info.registre=0x00;
T_IRM_Etat_Commodo_EG.trame_info.champ.extend=1;
T_IRM_Etat_Commodo_EG.trame_info.champ.dlc=0x08; // On d      de le      rgistres
T_IRM_Etat_Commodo_EG.trame_info.champ.rtr=1;
T_IRM_Etat_Commodo_EG.ident.extend.identificateur.iden                M8_Commodo_EG;
// Pour initialiser système (Enmener le balai en posi      sur                course droit)
Ecrire_Trame(T_IRM_Acquisition_FC);       // Envoi      'acqui      de l'état des fins de course
do{}while(Lire_Trame(&Trame_Recue)==0);   //On at      onse
Etat_FC=~Trame_Recue.data[0];     // On récupére      t      de course
if(fcd==0)       //SI le balai EG n'es      pas en      n dr      commande une rotation à droite
        {T_IM_Asservissement.data[0]      dresse      gistre PWM1DC
         T_IM_Asservissement.data[2]=      ess      Lente      // V      ur ->  Pour vitesse lente
         Ecrire_Trame(T_IM_Asservissement)      //      trame de commande moteur
         do{}while(Lire_Trame(&Trame_Recue)      //On      la réponse
         while(fcd==0)
                {Ecrire_Trame(T_IRM_Acquisition      // Envoi trame d'acquisition états fins de course
                 do{}while(Lire_Tra      e_Recu      //On attend la réponse
                 Etat_FC=~Trame_Re      [0];      //      écupére l'état des fin de course
         }} // Fin de mise en posi      initiale      balais
T_IM_Asservissement.data[0]=0x25;      resse d      istre PWM1DC
T_IM_Asservissement.da      a[2]=0;      //      ur ->      vitesse nulle
 Ecrire_Trame(T_IM_As      vissement);
 do{}while(Lire_Trame(      ame      )==0,      nd la réponse
// Initialisation des v      i      sy
Etat_Arret=1,Etat_Rot_Dro      ,E      _Gauche=0;
I_Autorise_Cycle=0,I_Inte      tent
Compteur_Secondes=0,Compt      as      _I
// Pour afficher titre du
gotoxy(1,2);
printf("  TP  n: 8      UDEN      ESSUI GLACE A PARTIR DU COMMODO   \n");
printf("  *******      ********** ********** ********** ********** ********** \n");
// Pour initiali      a be      temps et temporisations
SetVect(96,&ir      //      en place de l'autovecteur
PITR      0048;      //      terruption toutes les 10 millisecondes
PIC      0;      //      60H
//      RINCIPALE
      ********
(1)      cquer      at des fins de courses
      IRM_      ion_FC); // Envoi trame d'acquisition des états fin de course
      imeO
      Cptr_Tim      +;}while(!Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<10000));
      if(C      tr_T      ==10000)
                ().gotoxy(2,10);
                rintf(" Pas de reponse a la trame interrogative pour fins de courses \n");
                printf("  Il faut recharger et relancer le programme \n");
                do{}while(1);} // Stop
      else { if(Trame_Recue.ident.extend.identificateur.ident==Ident_T_IRM1_Asservissement)
                // On teste si l'identificateur est correct
                {Etat_FC=~Trame_Recue.data[0];}}  // On récupére l'état des fin de course
```

```
// Acquérir l'état Commodo Essuie Glace
        Ecrire_Trame(T_IRM_Etat_Commodo_EG); // Envoi trame d'acquisition des états fin de course
        Cptr_TimeOut=0;
        do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<10000));
         if(Cptr_TimeOut==10000)
                {clsscr(),gotoxy(2,10);
                 printf(" Pas de reponse a la trame interrogative pour fins de courses \n");
                 printf("  Il faut recharger et relancer le programme \n");
                 do{}while(1);} // Stop
         else { if(Trame_Recue.ident.extend.identificateur.ident==Ident_T_IRM8_Commodo_EG)
                        // On teste si l'identificateur est correct
                        {Valeur_Commodo_EG=~Trame_Recue.data[1];  // On récupére l'état commodo EG
                         Valeur_Analogique=Trame_Recue.data[2];}} // On récupére l'état molette     EG
        // Traiter l'état Commodo Essuie Glace
        if(Cde_EG_Av_Pos2)I_Autorise_Cycle=1,I_Intermittent=0,Cde_Vitesse=Vitesse_Rapide;
        // (on autorise les cycles du balai, avec une vitesse rapide, si on est en position 2 sur le commodo)
        else {if(Cde_EG_Av_Pos1)I_Autorise_Cycle=1,I_Intermittent=0,Cde_Vitesse=Vitesse_Lente;
        // on autorise les cycles du balai, avec une vitesse lente, si on [illegible] position 1 sur le commodo)
                else    {if(Valeur_Analogique>=200)I_Intermittent=0; //[illegible]"Arret"
                         else {I_Intermittent=1,Cde_Vitesse=Vitesse_Rapide; //[illegible]ion "Intermittent"
                         if(Valeur_Analogique>=150)Tempo_Fin=Tempo5; // Se[illegible]position de la molette
                         else {if(Valeur_Analogique>=140)Tempo_Fin=[illegible] la t[illegible]s ou moins longue
                                 else {if(Valeur_Analogique>=120)Tempo_[illegible]mpo3;
                                 else {if(Valeur_Analogique>=90)Tempo_Fin[illegible]
                                 else {if(Valeur_Analogique==0[illegible]=Te[illegible]}}
                         }
        // Traitement du diagramme des états "système"
        if(Etat_Arret)   // Si le système est dans l'état "Ar[illegible]
                {if(I_Autorise_Cycle)            // Si on un[illegible]té [illegible]isé
                        {I_Autorise_Cycle=0;
                         Etat_Arret=0,Etat_Rot_Gauche=[illegible] Evo[illegible]ystème
                         T_IM_Asservissement.data[2]=[illegible]e; /[illegible]ande la rotation à gauche

                                                    [illegible]sser[illegible]t.data[0]=0x26;        // Adresse du registre
PWM2DC
                         Ecrire_Trame(T_IM_Asser[illegible]
                         while(Lire_Trame(&Tra[illegible]de)[illegible]}} // On attend la réponse
        if(Etat_Rot_Gauche)      // S[illegible]le sys[illegible] dans[illegible]"Rotation à Gauche"
                {if(fcg)                 // Si[illegible]arrive[illegible]n de course gauche
                        {Etat_Rot_Ga[illegible]e=0[illegible]tat_R[illegible]te=1;     // Evolution etat système
                         T_IM_Asservisse[illegible]ta[2][illegible] // On arrête la rotation à gauche
                         T_IM_Asservisse[illegible][0]=0[illegible]     // Adresse du registre PWM2DC
                         Ecrire_Trame(T_IM_A[illegible]ement[illegible]
                         while(Lire_T[illegible](&Tra[illegible])==0){};    // On attend la réponse
                         T_IM_Asse[illegible]nt.data[illegible]Vitesse; // On commande la rotation gauche
                         T_IM_Ass[illegible]nt.data[0][illegible]5;        // Adresse du registre PWM1DC
                         Ecrire_T[illegible]T_IM_Ass[illegible]issement);
                         while(Li[illegible]ame(&Tr[illegible]ecue)==0){}; }} // On attend la réponse
        if(Etat_Rot_D[illegible]ite)      /[illegible]e syste[illegible]st dans l'état "Rotation à droite"
                {if([illegible])                 /[illegible]st arrivé en fin de course droit
                   {i[illegible]de [illegible]os1[illegible]s2)
                                    [illegible]oit[illegible]_Rot_Gauche=1;    // Evolution etat système
                                    [illegible]IM_[illegible]issement.data[2]=0;  // On arrête la rotation à droite
                                    [illegible]IM_[illegible]vis[illegible]ment.data[0]=0x25;       // Adresse du registre PWM1DC
                                    [illegible]i[illegible]am[illegible]IM_Asservissement);
                                    [illegible]ire_[illegible](&Trame_Recue)==0){};   // On attend la réponse
                                    [illegible]ment.data[2]=Cde_Vitesse; // On commande la rotation à gauche
                                    T_I[illegible]ement.data[0]=0x26;       // Adresse du registre PWM2DC
                                    [illegible]ire_[illegible](T_IM_Asservissement);
                                    [illegible](Lire_Trame(&Trame_Recue)==0){};}  // On attend la réponse
                                else
                                    {Et[illegible]_Droite=0,Etat_Arret=1;  // Evolution etat système
                                     T[illegible]ervissement.data[2]=0;   // On arrête la rotation à droite
                                     [illegible]sservissement.data[0]=0x25;       // Adresse du registre PWM1DC
                                     [illegible]e_Trame(T_IM_Asservissement);
                                     [illegible]ire_Trame(&Trame_Recue)==0){};    // On attend la réponse
                                    }}}
                [illegible]iche[illegible]tat syst[illegible]
                ---------[illegible]----------
        Cptr_[illegible]ffic[illegible];
        if([illegible]ge==200)
                [illegible]_Affichage=0; //…
                } // Fin "Afficher l'état"
        } //  Fin boucle principale
} // Fin fonction principale
```

9 TP N°9: ENSEMBLE DES COMMANDES AU VOLANT

9.1 Sujet

Objectifs :	- Développer une application compléte temps réel, comprenant à la fois des capteurs TOR (Tout Ou Rien) et des capteurs analogiques, des préactionneurs analogiques et intégrant une fonction de variation de vitesse.
Cahier des charges :	A intervalles de temps régul[illegible] ter[illegible] module sur lequel est connecté les commodos essuie glace [illegible] um[illegible] afin de [illegible]nnaître leurs états. En fonction de l'état du com[illegible] ssu[illegible]ace, on commande le moteur (intermittent, positi[illegible]ition En fonction de l[illegible] com[illegible]umière, on commande les différents feux des blocs opti[illegible]ants, [illegible]leuses, codes, phares, feux stop)

Matériels et logiciels nécessaires :

Micro ordinateur de type PC sou[illegible]ndows[illegible]u ultérieur
Logiciel Editeur-As[illegible]mbleur-Déb[illegible]
Si programmation en [illegible], C[illegible]eur [illegible]++ Réf : EID210100
Carte processeur 16/32 [illegible]a H[illegible]ontroleur 68332 et son environnement logiciel
(Editeur-CrossAssembl[illegible]Déb[illegible]ue[illegible] Réf: EID210000
Carte réseau CAN PC/I[illegible]nez [illegible]ON SYSTEMES Réf : EID004000
-2 modul[illegible]tron[illegible]rées" Réf: EID050000
(éventue[illegible] -1 commodo essuie glace
-1 commodo essuie Feux
[illegible] mod[illegible]S de [illegible]nce destinés aux 2feux arrières et aux 2feux avants Réf : EID051000
[illegible] module [illegible]e "asservissement" Réf: EID052000
[illegible] néra[illegible] l'essuie glace Réf: EID053000

Câbl[illegible]aison U[illegible]ou à défaut câble RS232, Réf: EGD000003
Alimentatio[illegible]5V, 1A pour l'alimentation de l'unité centrale Réf: EGD000001
Alimentati[illegible]our l'alimentation des modules CAN (réseau "énergie").

Durée estimée : 4 heures

9.2 Eléments de solution

9.2.1 Analyse

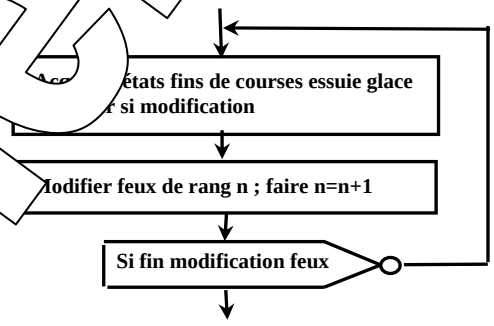
Le programme devra gérer deux processus indépendants:
- le "système Feux" commandé par la commodo "feux" en y intégrant le contrôle des ampoules,
- le "système Essuie Glace" commandé par le commodo "Essuie glace".

Il devra également réaliser l'affichage des états et résultats de test.

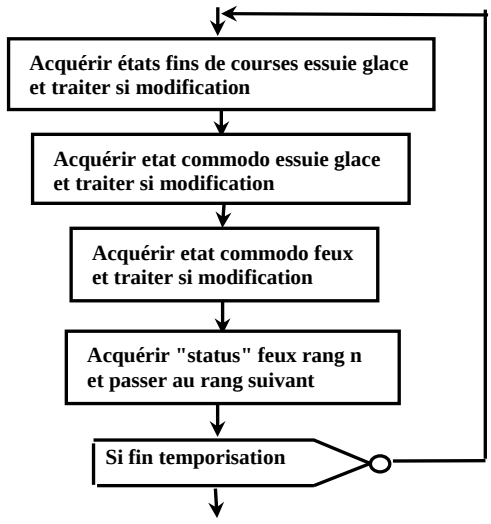
Certaines tâches à réaliser sont plus prioritaires que d'autres. On peut les classer comme suit (dans un ordre de priorité décroissant):
- acquisition des états fins de courses "essuie glace", dans le cas ou celui-ci est en cours de cycle,
- modification de l'état des feux, dans le cas où une modificati[illegible]t en cours,
- acquisition de l'état des commodos (feux et essuie glace),
- changer l'état des feux clignotants si l'un est activé et que [illegible]po[illegible] associée est terminée,
- contrôle de l'état des ampoules des différents feux (interro[illegible]des "s[illegible]),
- affichage des états à l'écran.

Remarques:

- La modification de l'état des feux se fait séquentiellement
(l'un après l'autre) par l'envoi de 4 trames (une par b
optique).
Rien n'empêche d'acquérir les états fins de cour[illegible]
glace entre chaque envoi d'une trame de modi[illegible]
Le cycle décrit ci-contre est donc e[illegible]té 4 [illegible]haque
fois qu'il a été détecté une modifica[illegible]n d[illegible]l'état [illegible]odo
feux mais aussi dans le cas où un clign[illegible]t est a[illegible]
que la temporisation associée est arrivée à s[illegible]e.



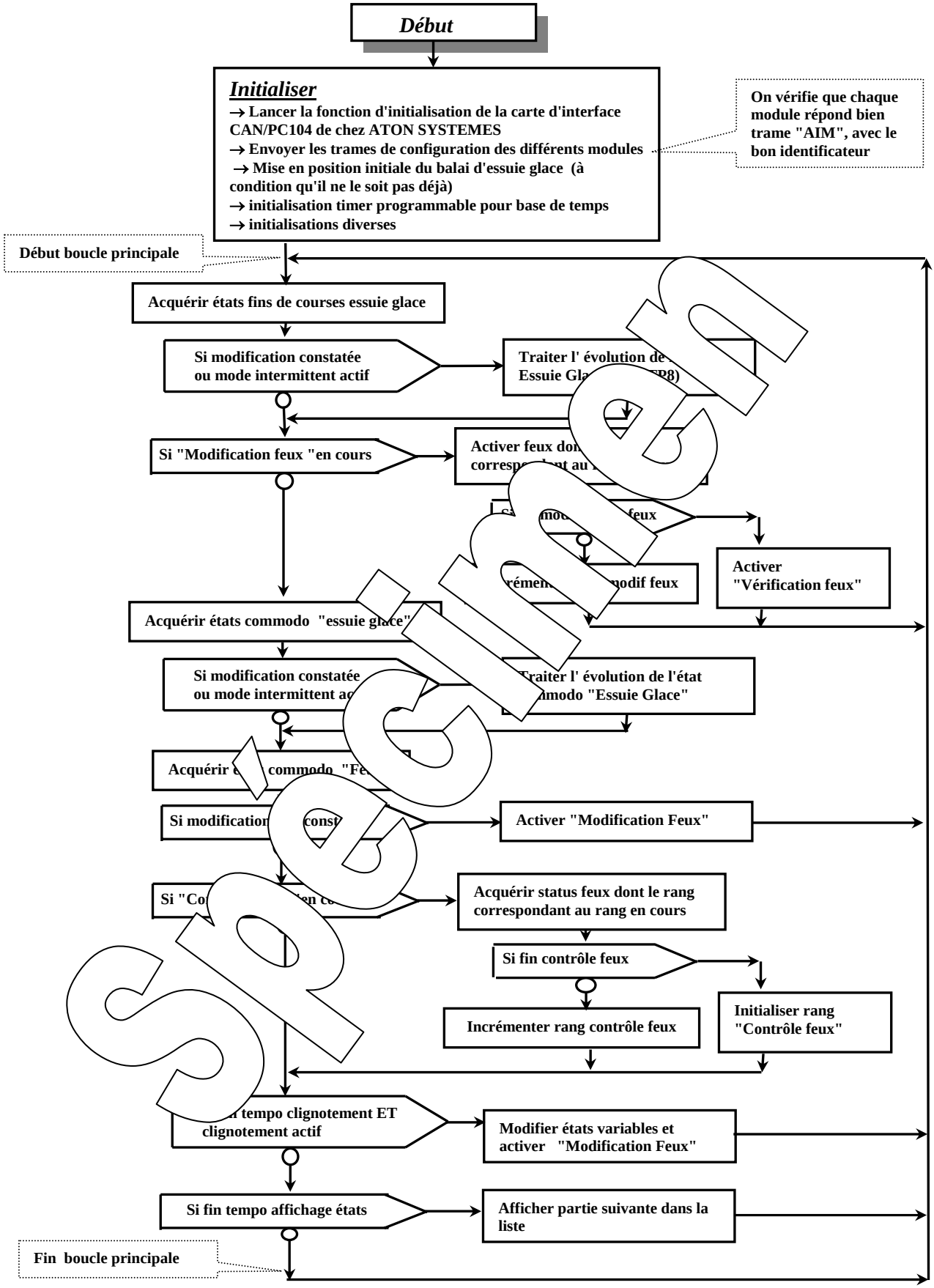
- Si on n'est pas en cours de mod[illegible]tion de[illegible]x, le
programme décrit a[illegible]s un autre c[illegible]ui en[illegible]
l'acquisition des états [illegible]es c[illegible]dos [illegible]suie glace)
ainsi que l'acquisition de[illegible]a[illegible]n des blocs optiques.
On change de bloc à cha[illegible]pa[illegible]e d[illegible]s le cycle.
Deux fins de temporisati[illegible]test[illegible]dans cette boucle:
-> fin tempori[illegible]clig[illegible]ui-ci est activé
-> fin tempo[illegible]hage résulats.



Si [illegible]ur ac[illegible]T fin [illegible]orisation clignoteur
d[illegible]ntraîne le[illegible]dent "Modification feux"
[illegible]ettan[illegible]e à j[illegible]at des feux.

Si la [illegible]tempo[illegible]on d'affi[illegible]age est détectée, on
affiche une [illegible]ement de l'écran, la durée d'affichage
de l'ensem[illegible]ran durant trop de temps. Un ciquième
seulement de l'écran est raffraîchi à chaque fois.

9.2.2 Organigramme général



9.2.3 Programme en "C"

```
        /*****************************************************************************************
        *       TP sur  EID210 / Réseau CAN - VMD  (Véhicule Multiplexé Didactique)
        *****************************************************************************************
        *         TP n 9:   Ensemble des commandes au volant (Feux + Essuie glace)
        *----------------------------------------------------------------------------------------
        *   CAHIER DES CHARGES :
        *  *********************
        *  Réunion des TP 4 et 8
        *----------------------------------------------------------------------------------------
        *  NOM du FICHIER :  CAN_VMD_TP9.C
        * *****************
        *****************************************************************************************/
// Déclarations
// -> des 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"
// -> des prototypes des fonctions
void Envoi_IM_Et_Test(void);
// Déclarations des constantes
//----------------------------
// Pour les temporisations
#define Tempo_Clignot 8               // En dizième de seconde -> 0,8 S
#define Tempo_Affichage 10   // Attente réponse en dizième de seconde -> 0
// Pour le codage des états
#define Etat_Attente 0
#define Etat_Modif_Feux 1
#define Etat_Control_Stat 2

// Pour l'essuie Glace
#define Vitesse_Lente 80
#define Vitesse_Rapide 120
#define Tempo1 2    // en secondes
#define Tempo2 4 // Pour mode intermittent
#define Tempo3 6
#define Tempo4 8
#define Tempo5 10

// Déclaration des variables
//--------------------------
// Pour affichage message alerte
char Texte[25];
//Pour Essuie Glace
unsigned char Valeur_Analogique,Cde_Vitesse,Tempo_
int Compteur_EG_Secondes,Compteur_EG_Passage_Irq;
// Pour les indicateurs divers (variables binaires)
union word_bits Indicateurs;
#define I_Fin_Tempo_Clignot Indicateurs.b
#define I_Fin_Tempo_Affichage Indicateur
#define I_Clignot_Gauche Indicateurs.bit
#define I_Clignot_Droit Indicateurs.bit.
#define I_Message_Pb_Affi e Indicateurs.
#define I_Intermittent I ateurs.bit.b9
#define I_Autorise_Cycle icateu it.b10
// Pour le diagramme des é s
union byte_bits Etat_EG;
#define Etat_EG_Arret Etat_EG b0
#define Etat_EG_Rot_Gauche Et bit
#define Etat_EG_Rot_Droite Eta

// Pour les fins de courses Essu
union byte_bits FC;
#define Valeur_FC_EG  Po te des fins de course
#define fs FC.bit.b7  r fin de surcourse
#define fcg FC.bit.   fin de course gauche
#define fcd FC.b      // in de course droit
unsign har Val  Mem;

//  n des tra
T  Recue;
   Trame_E
   T_IM  me de  put Message" pour commande module 4 Sorties de puissance
T  e d'ac  suite à une IM
Tra  //  de type  rmation Request Message" pour interroger "Status" lampes
                  // Ou co do

// Pour la comp  identificateurs Trame envoyée <-> Trame reçue
#define Ident  Trame_Envoyee.ident.extend.identificateur.ident
#define Ident  Trame_Recue.ident.extend.identificateur.ident
#define Ident_T_IM  T_IM.ident.extend.identificateur.ident
#define Ident_T_AIM  T_AIM.ident.extend.identificateur.ident
#define Ident_T_IRM  T_IRM.ident.extend.identificateur.ident

#define Valeur_T_IM T_IM.data[2]

// Pour les temporisations
WORD Compteur_Passage,Compteur_dS; // dS -> dizième de Seconde
WORD Valeur_Fin_Tempo_Clignot,Valeur_Fin_Tempo_Affichage,Rang_Affich;
// Pour le diagramme des états
unsigned char Etat,Rang_Control_Stat,Rang_Modif_Feux;
// Pour la mémoire
unsigned char Valeur_Commodo_Feux_Mem,Valeur_Commodo_EG_Mem;
//  Fonction d'interruption "Base de Temps"
void irq_bt()
```

```
// Fonction exécutée toute les 10 mS
{// Pour les feux

 Compteur_Passage++;
 if(Compteur_Passage==10) // Une 1/10 Seconde s'est écoulée
          {Compteur_Passage=0;
           Compteur_dS++;
           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;}
                   }
          }
// Pour l'Essuie Glace
if(I_Intermittent) // Si mode intermittent actif
          {Compteur_EG_Passage_Irq++;
           if(Compteur_EG_Passage_Irq==100)  // Une  Seconde s'est écoulée
                   {Compteur_EG_Passage_Irq=0;
                    Compteur_EG_Secondes++;
                    if(Compteur_EG_Secondes>=Tempo_Fin)
                             {Compteur_EG_Secondes=0;
                              I_Autorise_Cycle=1;}
                   }}
} // Fin de la fonction d'interruption

//==========================
//  FONCTION PRINCIPALE
//==========================
main()
{
// Initialisations
//*****************
clsscr();
/* Initialisation DU SJA1000 de la carte ATON-Systemes" sur  */
Init_Aton_CAN();
//  Définition des trames pour activer ou lire un bloc optique
// D'aprés doc SJA1000 et  doc MCP25050 pages 22 (fonction "Writ                er")                GPPIN)
// Pour les trames de commande  -> IM (Input Message)
T_IM.trame_info.registre=0x00;
T_IM.trame_info.champ.extend=1; // On travaille en mode étend
T_IM.trame_info.champ.dlc=0x03; // Il y aura 3 données de                octets
// Pour la configuration des modules de 4 sorties et 4 Ent                -> GP
T_IM.data[0]=0x1F;  // première donnée -> "Adresse" du reg
T_IM.data[1]=0x7F;  // deuxième donnée -> "Masque" -> V                MCP                e 16
T_IM.data[2]=0xF0;  // troisième donnée -> "                ies so                bits de poids faibles
Ident_T_IM=Ident_T_IM_FRD;    // C'est l'i                cateur                aRrière                pour l'IM
Ident_T_AIM=Ident_T_AIM_FRD;  // C'est l'id                ica                eur du                ière D                pour l'Acquittement
strcpy(Texte,"Feux aRriere Droit    ");
Envoi_IM_Et_Test();
Ident_T_IM=Ident_T_IM_FRG;    // C'est l'identifica                Feux ar                gauche pour l'IM
Ident_T_AIM=Ident_T_AIM_FRG;  // C'est l'identificate                arri                gauche pour l'Acquittement
strcpy(Texte,"Feux aRriere Gauche    ");
Envoi_IM_Et_Test();
Ident_T_IM=Ident_T_IM_FVG;    // C'est l                ateur du Fe                nt Gauche pour l'IM
Ident_T_AIM=Ident_T_AIM_FVG;  // C'est l                ificateur                Feux                ant Gauche pour l'Acquittement
strcpy(Texte,"Feux aVant Droit    ");
Envoi_IM_Et_Test();
Ident_T_IM=Ident_T_IM_FVD;    // C'est l'                icateur                eux aVant Droit pour l'IM
Ident_T_AIM=Ident_T_AIM_                // C'est l'i                eux aVant Droit pour l'Acquittement
strcpy(Texte,"Feux aVant D                t
Envoi_IM_Et_Test();
// Pour la configuration des m                2

T_IM.data[2]=0x7F; // troisiè                nnée                Val                " -> Pour les modules 8 entrées GP7 non concerné
Ident_T_IM=Ident_T_IM_Commodo                // C'                identificateur du Commodo Feux pour l'IM
Ident_T_AIM=Ident_T_AIM_Commodo                // C'                identificateur du Commodo Feux pour l'Acquittement
strcpy(Texte,"Commodo Feux
Envoi_IM_Et_Test();
Ident_T_IM=Ident_T_I                // C'est l'identificateur du Commodo Essuie Glace pour l'IM
Ident_T_AIM=Ident_T_A                // C'est l'identificateur du Commodo Essuie Glace pour l'Acquittement
strcpy(Texte,"Com                uie G                ;
Envoi_IM_Et_Test
// Po                onfigu                modul                ervissement" sur lequel est connecté le moteur Essuie Glace
T_I                0xE3;                ntrée et 0 si Sortie
                GP7=fs Enti                ree; GP5=fcd Entrée;  GP4=ValidIP Sortie;
                GP3=PWM3 Sort                WM1 Sortie; GP1=AN1 Entrée; GP0=AN0 Entrée;
T_IM=T                serv                // C'est l'identificateur du module "Asservissement
                serv                // C'est l'identificateur du Commodo Essuie Glace pour l'Acquittement
                e Glace
Envo                ();
// Si                ve jusque                est que tous les 7 modules ont répondu !
//-------------------------------------------------------------
// Pour la conf                l'entrée analogique (Position molette sur le commodo Essuie Glace)
// Pour activ                sions Ana -> Num
T_IM.data[0]=0x                dresse du registre ADCON0
T_IM.data[1]=0xF0;  // Masque -> bits 7..4 concernés
T_IM.data[2]=0x80;  // Valeur ->  ADON=1 et "prescaler rate"=1:32
Ident_T_IM=Ident_T_IM_Commodo_EG;    // C'est l'identificateur du Commodo Essuie Glace pour l'IM
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir le mode de conversion
T_IM.data[0]=0x2B;  // Adresse du registre ADCON1
T_IM.data[1]=0xFF;  // Masque -> tous les bits sont concernés
T_IM.data[2]=0x0E;  // Valeur ->  voir doc MCP25050 page 36 (GP0 -> Entrée Analogique)
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour configurer les sortie PWM du module "Asservissement"
Ident_T_IM=Ident_T_IM_Asservissement;    // C'est l'identificateur du module "Asservissement"
```

```
// Pour définir sortie GP2 en PWM1
T_IM.data[0]=0x21;  // Adresse du registre T1CON
T_IM.data[1]=0xB3;  // Masque -> seuls bit 7;5;4;1;0 consernés
T_IM.data[2]=0x80;  // Valeur ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir fréquence signal sortie PWM1
T_IM.data[0]=0x23;  // Adresse du registre PR1
T_IM.data[1]=0xFF;  // Masque -> tous les bits sont consernés
T_IM.data[2]=0xFF;  // Valeur ->  PR1=255
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir sortie GP3 en PWM2
T_IM.data[0]=0x22;  // Adresse du registre T2CON
T_IM.data[1]=0xB3;  // Masque -> seuls bit 7;5;4;1;0 consernés
T_IM.data[2]=0x80;  // Valeur ->  TMR2ON=1; Prescaler2=1
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour définir fréquence signal sortie PWM2
T_IM.data[0]=0x24;  // Adresse du registre PR2
T_IM.data[1]=0xFF;  // Masque -> tous les bits sont consernés
T_IM.data[2]=0xFF;  // Valeur ->  PR2=255
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour initialiser le rapport cyclique du signal PWM1
T_IM.data[0]=0x25;  // Adresse du registre PWM1DC
T_IM.data[1]=0xFF;  // Masque -> tous les bits sont consernés
T_IM.data[2]=0;     // Valeur ->  PWM1DC=0
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour initialiser le rapport cyclique du signal PWM2 à 0
T_IM.data[0]=0x26;  // Adresse du registre PWM2DC
T_IM.data[1]=0xFF;  // Masque -> tous les bits sont consernés
T_IM.data[2]=0;     // Valeur ->  PWM2DC=0
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
// Pour Valider le circuit de puissance
T_IM.data[0]=0x1E;  // Adresse du registre GPLAT  (Registre I/O)
T_IM.data[1]=0x10;  // Masque -> sortie GP4 (ValidIP) est conserné
T_IM.data[2]=0x10;  // Valeur ->  ValidIP=1
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Attendre réponse
T_IM.data[1]=0xFF;  // Masque -> Pour les IM futures

clsscr(); // Pour effacer l'écran
// Pour les trames interrogatives ->  IRM  (Information Se
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;
// Pour Mettre le balai d'essuie glace en positio       iale
Ident_T_IRM= Ident_T_IRM1_Asservissement; // Les          ourses       nectés sur module "Asservissement"
Ecrire_Trame(T_IRM);          // Envoi trame d'acqui          l'éta       fins de course (1 octet en réponse)
do{}while(Lire_Trame(&Trame_Recue)==0);          //On       répon
Valeur_FC_EG=~Trame_Recue.data[0];     // O          ére l          fin de course
if(fcd==0)          //SI le balai EG n'est          osition          commande une rotation à droite
          {T_IM.data[0]=0x25;           //          du registre          C
           T_IM.data[2]=Vitesse_Lente;          eur ->  Pou          itess          lente
           Ident_T_IM=Ident_T_IM_Asservi          t;// Ce          e module "Asservissement qui sera concerné
           Ecrire_Trame(T_IM);          i trame          mande moteur
           do{}while(Lire_Trame(&Trame_Re          .          On attend la réponse
           do       {Ecri          rame(T_IRM);          ame d'acquisition états fins de course
                     do{}w          e(Li          &Tr          //On attend la réponse
                     Valeur_FC          e.d          // On récupére l'état des fin de course
                     Valeur_FC          C_EG;
                    }while(fc
          T_IM.data[2]=0;          leu          Po          itesse nulle
          Ident_T_IM=Ident_T_I          emen
          Ecrire_Trame(T_IM);
          do{}while(Lir          me(&I          //On attend la réponse
          T_IM.data[0          Po          dresse du registre sortie GPLAT
          }
// Initialisation          t Essuie Glace
Etat_EG_Arret=1,E          tot_Dr          tat_EG_Rot_Gauche=0;
Compteur_EG_Seco          pteur_          age_Irq=0;
Vale          Mem=0,
Va          _Feux_M

          variabl          at
          Etat
          atus"
Ra
// P          Modif
Rang_Mo          eux=1;
// Pour l'ensemble des          ateurs
Indicateurs.va
// Pour l'affi
Rang_Affich=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;
```

```
//   Afficher titre
gotoxy(1,2);
 printf("      TP  n: 9    Ensemble des commandes au volant\n");
 printf("    ********************************************** \n");

// Boucle principale
//******************
while(1)
 {// Acquisition état des fins de courses Essuie Glace
         Ident_T_IRM=Ident_T_IRM1_Asservissement; // On commence par l'état lecture des Fins de Courses EG
         Ecrire_Trame(T_IRM);
         do{}while(Lire_Trame(&Trame_Recue)==0);          //On attend la réponse
         if(Ident_Trame_Recue==Ident_T_IRM1_Asservissement) // On teste si identificateur correct
                  {Valeur_FC_EG=~Trame_Recue.data[0];}// On récupère les états Fins de Courses
         if((Valeur_FC_EG!=Valeur_FC_EG_Mem)||(I_Autorise_Cycle==1))// Si modif états fins de courses
                  { Valeur_FC_EG_Mem=Valeur_FC_EG;
                   // Traiter évolution Etat Essuie Glace
                   if(Etat_EG_Arret)  // Si le système est dans l'état "Arret"
                            {if(I_Autorise_Cycle)                 // Si on un cycle a été autorisé
                                     {I_Autorise_Cycle=0;
                                      Etat_EG_Arret=0,Etat_EG_Rot_Gauche=1; // Evolution etat système
                                      Ident_T_IM=Ident_T_IM_Asservissement;
                                      T_IM.data[2]=Cde_Vitesse; // On commande la rotation à gauche
                                      T_IM.data[1]=0xFF;          // Masque
                                      T_IM.data[0]=0x26;          // Adresse du re                2DC
                                      Ecrire_Trame(T_IM);
                                      T_IM.data[0]=0x1E;          // Adresse du re                ecritures futures)
                                      while(Lire_Trame(&Trame_Recue)==0){}; //                nd la
                                     }}// Fin si Etat_EG_Arret
                  if(Etat_EG_Rot_Gauche)       // Si le système est dans l'état                à Gauch
                            {if(fcg)           // Si on est arrivé en fin de course                tion etat système
                                     {Etat_EG_Rot_Gauche=0,Etat_EG_Rot_Dro
                                      Ident_T_IM=Ident_T_IM_Asservissem
                                      T_IM.data[2]=0;    // On arrête                 uche
                                      T_IM.data[1]=0xFF;          //                 e
                                      T_IM.data[0]=0x26;          //                se d          ist    PWM2DC
                                      Ecrire_Trame(T_IM);
                                      while(Lire_Trame(&Trame_Recue)==0                n att          a réponse
                                      T_IM.data[2]=Cde_Vitesse; // On co                ro             gauche
                                      T_IM.data[0]=0x25;                  dres                       PWM1DC
                                      Ecrire_Trame(T_IM);
                                      T_IM.data[0]=0x1E;                                  e du registre sortie (ecritures futures)
                                      while(Lire_Trame(&Tr                de)=                // On attend la réponse
                                     }}// Fin si Etat_Ro
                  if(Etat_EG_Rot_Droite)       // Si le                    dans l                otation à droite"
                            {if(fcd)           // Si on                    fin de                se droit
                               {if(Cde_EG_Av_Pos1|Cde                os2)
                                     {Eta                 Rot_D                rat_EG                he=1;      // Evolution etat système
                                      Id                 ZM=Iden                servis
                                      T_I                ta[ ]=0;                rête la                tation à droite
                                      T_IM.data[                x25;                // Adresse du registre PWM1DC
                                      Ecrire_T                   );
                                      while(Lire_                rame_Re                0){}; // On attend la réponse
                                      T_IM.data[2]=                se; //                commande la rotation à gauche
                                      T_IM.data[0]=0x                                  // Adresse du registre PWM2DC
                                      E                 e(T_IM);
                                      T_IM.data[0]=0x1E;                               // Adresse du registre sortie (ecritures futures)
                                      while(Lire_Trame(&Trame_Recue)==0){}; }          // On attend la réponse
                                 else {E                 _Rot_Droi                Etat_EG_Arret=1;   // Evolution etat système
                                       T_IM=Ide                M_Asservissement;
                                       T_IM.data[2]=0;                 // On arrête la rotation à droite
                                       T_IM.data[0]=0x                                 // Adresse du registre PWM1DC
                                       Ecri                   );
                                       T_IM.da                                         // Adresse du registre sortie (ecritures futures)
                                       while(Lire_Trame(&Trame_Recue)==0){};}// On attend la réponse
                                      Fin si Etat_EG_Rot_Droite
                  }   // Fi          ter          uti          tat Essuie Glace

         // Traiter la modific                feux
         if(Etat==Etat_Modif_Feu
                  {                  Rang_                Module suivant dans la liste
                                                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_Trame(T_IM); // Envoyer trame
                                                while(Lire_Trame(&Trame_Recue)==0){};} // On attend la réponse
                                      break;
                                      case 2 : {Ident_T_IM=Ident_T_IM_FRD; //Feux aRrière Droit
                                                Valeur_T_IM=Valeur_FRD;
                                                Rang_Modif_Feux++;
                                                Ecrire_Trame(T_IM); // Envoyer trame
                                                while(Lire_Trame(&Trame_Recue)==0){};} // On attend la réponse
                                      break;
                                      case 3 : {Ident_T_IM=Ident_T_IM_FRG; //Feux aRrière Gauche
                                                Valeur_T_IM=Valeur_FRG;
                                                Rang_Modif_Feux++;
                                                Ecrire_Trame(T_IM); // Envoyer trame
                                                while(Lire_Trame(&Trame_Recue)==0){};} // On attend la réponse
                                      break;
                                      case 4 : {Ident_T_IM=Ident_T_IM_FVG; //Feux aRrière Gauche
                                                Valeur_T_IM=Valeur_FVG;
                                                Ecrire_Trame(T_IM); // Envoyer trame
                                                while(Lire_Trame(&Trame_Recue)==0){};
                                                Etat=Etat_Control_Stat; // On a fini, on passe au controle
                                                Rang_Control_Stat=1;}        // de l'état des lampes
                                      break;
                                     }  // Fin "switch"
                  } // FIN si "Etat modif feux"
```

```
else   // On n'est pas en état modif feux
       {  // Acquisition état Commodo Essuie Glace
          //On prépare la trame pour interroger le module Commodo Essuie Glace
          T_IRM.trame_info.champ.dlc=0x08;     // 8 octets en réponse car voie ana
          Ident_T_IRM=Ident_T_IRM8_Commodo_EG; // Commodo Essuie Glace
          Ecrire_Trame(T_IRM); // On envoie la trame interrogative sur le bus
          T_IRM.trame_info.champ.dlc=0x01;     // 1 octets en réponse pour interrogations futures
          do{}while(Lire_Trame(&Trame_Recue)==0);       //On attend la réponse
          if(Ident_Trame_Recue==Ident_T_IRM8_Commodo_EG)
          {Valeur_Commodo_EG=~Trame_Recue.data[1];       // On récupére l'état commodo EG
          Valeur_Analogique=Trame_Recue.data[2];}       // On récupére l'état molette EG
          // Traiter la modification de l'état du Commodo Essuie Glace
if(Cde_EG_Av_Pos2)I_Autorise_Cycle=1,I_Intermittent=0,Cde_Vitesse=Vitesse_Rapide;
          // on autorise les cycles, avec une vitesse rapide, si on est en position 2 sur le commodo)
else {if(Cde_EG_Av_Pos1||Cde_Lave_Glace_Av)I_Autorise_Cycle=1,I_Intermittent=0,Cde_Vitesse=Vitesse_Lente;
          // on autorise les cycles, avec une vitesse lente, si on est en position 1 sur le commodo)
                  else      {if(Valeur_Analogique>=200)I_Intermittent=0; // Position "Arret"
                             else {I_Intermittent=1,Cde_Vitesse=Vitesse_Lente; // Position "Intermittent"
                             if(Valeur_Analogique>=150)Tempo_Fin=Tempo5;// Selon la position de la molette
                             else {if(Valeur_Analogique>=140)Tempo_Fin=Tempo4;// la tempo plus ou moins longue
                             else {if(Valeur_Analogique>=120)Tempo_Fin=Tempo3;
                             else {if(Valeur_Analogique>=90)Tempo_Fin=Tempo2;
                             else {if(Valeur_Analogique==0)Tempo_Fin=Tempo1;}}}}}}}
              // Fin Traiter la modificationCommodo Essuie Glace
              // Fin Acquisition état Essuie Glace
          // Acquisition de l'état "Lecture état Commodo Feux"
          //On prépare la trame pour interroger le module Commodo Essuie Gl
          Ident_T_IRM=Ident_T_IRM_Commodo_Feux; // Module "Commodo Feux"
          Ecrire_Trame(T_IRM);// On envoie la trame interrogative sur
          do{}while(Lire_Trame(&Trame_Recue)==0);          //On attend la
          if(Ident_Trame_Recue==Ident_T_IRM_Commodo_Feux)// On vérifie l'ide                ur
                   {Valeur_Commodo_Feux=~(Trame_Recue.data[0]);}                    commodo
                    if(Valeur_Commodo_Feux!= Valeur_Commodo_Feu
                    // Si on a détecté une modification de l'é
                             {Valeur_Commodo_Feux_Mem = Valeu                  // On met en mémoire
                              // On prédéfinit l'état des dif                les
                              // Combinaisons définies dans CA
                              Valeur_FVG=Cde_Nulle,Valeur_FVD=C
                              Valeur_FRG=Cde_Nulle,Valeur_FRD=Cde
                              I_Clignot_Droit=0;
                              I_Clignot_Gauche=0;
                              if(Cde_Phare) // Si Com                    Cde_FV_P;  // Les feux aVant
                                       {Valeur_FVG                    de_FR_P;} // Les feux aRière
                              else if(Cde_Code)                    de Cod
                                       {Valeur_F                    de_FV_C;  // Les feux aVant
                                        Vale                    r_FRD=Cde_FR_C;} // Les feux aRière
                              els                    // Si                veilleuse
                                       {Valeu                    VD=Cde_FV_V;  // Les feux aVant
                                                                 FRD=Cde_FR_V;} // Les feux aRière
                              if(Cde_C                    oit)
                                                          gn_AV;
                                                          gn_AR;
                                                          it=1;
                                                          Clignot = Compteur_dS + Tempo_Clignot;
                                                          t = 0;}  // Pour Init tempo_clignot
                                                          gn_Gauche)
                                                          Masque_Clign_AV;
                                                          Masque_Clign_AR;
                                                          Gauche=1;
                                                          Tempo_Clignot = Compteur_dS + Tempo_Clignot;
                                                          Clignot = 0;}  // Pour Init tempo_clignot
                                       if(Cd
                                                          Masque_Klaxon;
                                                          Masque_Klaxon;}
                                       de_Stop)
                                                          Masque_Stop;
                                                          Masque_Stop;}
                              Etat=                       Modif_Feux;
                                                          Feux=1;
                                                          si modif commodo feux détecté
                                       FIN                (else) état modif feux
if(Etat==Et                            // Controler les status des sorties Feux
                  n cont                bon état des ampoules
                  h(Rang_                 _Stat) // Module suivant dans la liste
                     {cas                        {// C'est le controle du Feux aVant Droite
                                                  Ident_T_IRM=Ident_T_IRM_FVD;
                                                  Ecrire_Trame(T_IRM); // Envoyer trame
                                                  while(Lire_Trame(&Trame_Recue)==0){}; // On attend la réponse
                                                  Valeur_Status_FVD=Trame_Recue.data[0]; // On récupère état FVD
                                                  Rang_Control_Stat++;}
                     break;
                     case 2                // C'est le controle du Feux aRrière Droit
                                                  Ident_T_IRM=Ident_T_IRM_FRD;
                                                  Ecrire_Trame(T_IRM); // Envoyer trame
                                                  while(Lire_Trame(&Trame_Recue)==0){}; // On attend la réponse
                                                  Valeur_Status_FRD=Trame_Recue.data[0]; // On récupère état FRD
                                                  Rang_Control_Stat++;}
                     break;
                     case 3 :  {// C'est le controle du Feux aRrière Gauche
                                                  Ident_T_IRM=Ident_T_IRM_FRG;
                                                  Ecrire_Trame(T_IRM); // Envoyer trame
                                                  while(Lire_Trame(&Trame_Recue)==0){}; // On attend la réponse
                                                  Valeur_Status_FRG=Trame_Recue.data[0];
                                                  Rang_Control_Stat++;}
                     break;
                     case 4 :  {// C'est le controle du Feux aVant Gauche
                                                  Ident_T_IRM=Ident_T_IRM_FVG;
                                                  Ecrire_Trame(T_IRM); // Envoyer trame
                                                  while(Lire_Trame(&Trame_Recue)==0){}; // On attend la réponse
                                                  Valeur_Status_FVG=Trame_Recue.data[0];
                                                  Rang_Control_Stat=1;} // On recommance au début!
                     break;
                     }  // Fin "switch"
```

```
                }// Fin si etat "controle status"
        // Si clignotement actif et fin temporisation clignoteur
        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;
                         Etat=Etat_Modif_Feux;
                         Rang_Modif_Feux=1;}
                 if(I_Clignot_Droit)
                        {// Commuter ampoules clignotants gauche
                         Valeur_FVD^=Masque_Clign_AV;
                         Valeur_FRD^=Masque_Clign_AR;
                         Etat=Etat_Modif_Feux;
                         Rang_Modif_Feux=1;}
                }  // Fin prise compte clignotant
        if(I_Fin_Tempo_Affichage)
                {I_Fin_Tempo_Affichage=0;
                 switch(Rang_Affich)
                 {case 1: // 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");}

                        Rang_Affich++;}
                        break;
                case 2:   // Résultat diagnostic Feux aVant
                        {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");}
                        Rang_Affich++;}
                        break;
                case 3:   // 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");}
                        Rang_Affich++;}
                        break;
                case 4:   // 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");}
                        Rang_Affich++;}
                        break;
                case 5:   // 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);
printf(" Etat commodo Essuie Glace  Etat = %d ;   Mollette = %d \n",Valeur_Commodo_EG,Valeur_Analogique);
                        Rang_Affich=1;}
                } // Fin "switch"
                }   // Fin if fin tempo affichage
        }// FIN de la boucle principale
}// FIN de la fonction principale
```

```
// Fonction "Envoi trame et test si module adressé répond
void Envoi_IM_Et_Test(void)
{int Cptr_TimeOut,Temp;
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Envoyer trame sur réseau CAN
           Cptr_TimeOut=0;
           do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
       if(Ident_Trame_Recue!=Ident_T_AIM)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 sur %S \n",Texte);
                               printf("  Verifier si presenc module et si alimentation 12 V  est OK \n");}
                    for(Temp=0;Temp<100000;Temp++);} // Pour attendre un peu!
    }while(Cptr_TimeOut==200);

}
```

10 ANNEXES

10.1 Fichier de définition propre au système CAN_VMD

```
//*****************************************************

//  Structures de données pour application  CAN VMD
//  Nom du fichier: CAN_VMD.h

//*****************************************************
#ifndef _VMD_H
#define _VMD_H
// Format de message
typedef struct {
    /* nombre d'octets à envoyer, -1 si c'est une Remote Frame */
    int dlc;
    unsigned char id1;  /* 8 bits de poids forts de l'ID. */
    unsigned char id2;  /* 3 bits de poids faible de l'ID. */
    unsigned char data[8];
} Message;
//  Pour l'identificateur en mode standard
typedef union
{struct   {unsigned short ident:11;
           unsigned short rtr:1;
           unsigned short nul:4;
          } identificateur;
 struct   {unsigned char ident1;
           unsigned char ident2;
          } registre;
 unsigned short valeur;
} ident_standard;
//  Pour l'identificateur en mode étendu
typedef union
{struct   {unsigned long ident:29;
           unsigned long rtr:1;
           unsigned long x:2;
          } identificateur;
 struct   {unsigned char ident1;
           unsigned char ident2;
           unsigned char ident3;
           unsigned char ident4;
          } registre;
 unsigned long valeur;
} ident_extend;
// Pour registre d'information Trame SJA1000
typedef union
          {struct   {unsigned char extend:1;
                     unsigned char rtr:1;
                     unsigned char nul:2;
                     unsigned char dlc:4;
                    } champ;
           unsigned char registre;
          } tr_info;

// Pour Trame circulant [illegible] réseau CAN
typedef struct
{
tr_info trame_info;
union     {ident_standard sta
           ident_extend   ext
          } ident;
unsigned char data[8];
} Trame;
 typedef union
{struct   {unsigned c
           unsigned c
           unsigne
           unsign
           unsigh
           signed
           signed ch
           unsigned char
          }bit
ed ch
}
Po
#def commodo Etat_Comm Feux.valeur
#define Cde_Veilleuse mmodo_Feux.bit.GP0
#define Cde_War modo_Feux.bit.GP1
#define Cde_Ph odo_Feux.bit.GP2
#define Cde_C modo_Feux.bit.GP3
#define Cde_Clign e Etat_Commodo_Feux.bit.GP4
#define Cde_Clign_Droit Etat_Commodo_Feux.bit.GP5
#define Cde_Stop Etat_Commodo_Feux.bit.GP6
#define Cde_Klaxon Etat_Commodo_Feux.bit.GP7
// Pour la Commande des feux
#define Cde_Nulle 0x00
#define Cde_FV_V 0x01 // Feux aVant en Veilleuse
#define Cde_FR_V 0x01 // Feux aRrière en Veilleuse
#define Cde_FV_C 0x03 // Feux aVant en Code
#define Cde_FR_C 0x01 // Feux aRrière en Code
#define Cde_FV_P 0x05 // Feux aVant en Phare
#define Cde_FR_P 0x01 // Feux aRrière en Phare
#define Masque_Clign_AV 0x08 // Pour Clignotant AVant
#define Masque_Clign_AR 0x04 // Pour Clignotant ARrière
#define Masque_Klaxon 0x08 // Pour Claxon
#define Masque_Stop 0x02 // Pour Stop
```

```
//  Variables pour la commande et le control des feux
//-----------------------------------------------------
//  Pour variables images de la commande effectuée des différents feux
union byte_bits Image_FVG,Image_FVD,Image_FRD,Image_FRG;
// Pour Feux aVant Gauche
#define Valeur_FVG Image_FVG.valeur     // Pour un accés port complet
#define Veilleuse_FVG Image_FVG.bit.b0
#define Code_FVG Image_FVG.bit.b1
#define Phare_FVG Image_FVG.bit.b2
#define Clignot_FVG Image_FVG.bit.b3
// Pour Feux aVant Droit
#define Valeur_FVD Image_FVD.valeur     // Pour un accés port complet
#define Veilleuse_FVD Image_FVD.bit.b0
#define Code_FVD Image_FVD.bit.b1
#define Phare_FVD Image_FVD.bit.b2
#define Clignot_FVD Image_FVD.bit.b3
// Pour Feux aRrière Droit
#define Valeur_FRD Image_FRD.valeur     // Pour un accés port complet
#define Veilleuse_FRD Image_FRD.bit.b0
#define Stop_FRD Image_FRD.bit.b1
#define Clignot_FRD Image_FRD.bit.b2
#define Klaxon_FRD Image_FRD.bit.b3
// Pour Feux aRrière Gauche
#define Valeur_FRG Image_FRG.valeur     // Pour un accés port complet
#define Veilleuse_FRG Image_FRG.bit.b0
#define Stop_FRG Image_FRG.bit.b1
#define Clignot_FRG Image_FRG.bit.b2
#define Klaxon_FRG Image_FRG.bit.b3
// Pour les variables images des "Status" des sorties de puissance
union byte_bits Image_Stat_FVG,Image_Stat_FVD,Image_Stat_FRD,Image_Stat_FRG;
// Pour image "Status" Feux aVant Gauche
#define Valeur_Status_FVG Image_Stat_FVG.valeur // Pour un accés port comple
#define S_Veilleuse_FVG Image_Stat_FVG.bit.b4
#define S_Code_FVG Image_Stat_FVG.bit.b5
#define S_Phare_FVG Image_Stat_FVG.bit.b6
#define S_Clignot_FVG Image_Stat_FVG.bit.b7
// Pour image "Status" Feux aVant Droit
#define Valeur_Status_FVD Image_Stat_FVD.valeur // Pour un accés port comp
#define S_Veilleuse_FVD Image_Stat_FVD.bit.b4
#define S_Code_FVD Image_Stat_FVD.bit.b5
#define S_Phare_FVD Image_Stat_FVD.bit.b6
#define S_Clignot_FVD Image_Stat_FVD.bit.b7
// Pour image "Status" Feux aRrière Droit
#define Valeur_Status_FRD Image_Stat_FRD.valeur // Pour un          t comp
#define S_Veilleuse_FRD Image_Stat_FRD.bit.b4
#define S_Stop_FRD Image_Stat_FRD.bit.b5
#define S_Clignot_FRD Image_Stat_FRD.bit.b6
#define S_Klaxon_FRD Image_Stat_FRD.bit.b7
// Pour image "Status" Feux aRrière Gauche
#define Valeur_Status_FRG Image_Stat_FRG.va   //  our un          t comp
#define S_Veilleuse_FRG Image_Stat_FRG.bit.b4
#define S_Stop_FRG Image_Stat_FRG.bit.b5
#define S_Clignot_FRG Image_Stat_FRG.bit.b6
#define S_Klaxon_FRG Image_Stat_FRG.bit.b7


//  Déclaration des identificateurs, pou       érents modu    r le bus  VMD
//-------------------------------------------------------------------------
// Pour Feux aVant Gauche
#define Ident_T_IRM_FVG 0x     E041E07      /     aVant Ga      en interrogation (Information Request Message)
#define Ident_T_IM_FVG 0x      080000       //      Vant Ga      en commande (Input Message)
#define Ident_T_AIM_FVG 0      200000              t Gauche en Acquittement commande
// Pour Feux aVant Droit
#define Ident_T_IRM_FVD 0x0841      ux a      t en interrogation (Information Request Message)
#define Ident_T_IM_FVD 0x0E880      x aVant Droit en commande (Input Message)
#define Ident_T_AIM_FVD 0x0EA       eux aVant Droit en Acquittement commande
// Pour Feux aRierre Gauche
#define Ident_T_IRM_FRG 0x0F04      Fe      ière Gauche en interrogation (Information Request Message)
#define Ident_T_IM_FRG 0x0F0800     // Fe      ière Gauche en commande (Input Message)
#define Ident_T_AIM_FRG 0x0F20000   //      rière Gauche en Acquittement commande
// Pour Feux aRierre
#define Ident_T_IRM_                //      aRrière Droit en interrogation (Information Request Message)
#define Ident_T_IM_FR               // Feux aRrière Droit en commande (Input Message)
#define Ident_T_A       0x0FA0      // Feux aRrière Droit en Acquittement commande
// Pour Commodo
#def      T_IM_      Feux 0      000 // Commodo feux en commande (Information Message)
#d        T_AIM_C      0000 // Commodo feux : Acquittement suite à une IM
#        t_T_IRM_Comm      1E07 // Commodo feux en interrogation (Information Request Message)
/       Essuie
#       e Ide       do_EG      00 // Commodo EG en commande (Information  Message)
#        T       o_EG      0 // Commodo EG en commande (Information  Message)
#d       _EG 0x      // Commodo EG en interrogation:  1 octet demandé
#de     IRM8_C      EG 0x05      0 // Commodo EG en interrogation:  8 octets demandés
#define      T_OB_Comm      0x05900000 // Commodo EG "On Bus" (Par scéduleur)
// Pour module Asservissement
#define Ident_T      sement 0x00880000     // "Asservissement" en commande (Input Message)
#define Ident_T      vissement 0x00841E07   // "Asservissement" en interrogation: 1 octet en réponse
#define Ident_T      ervissement 0x00840000   // "Asservissement" en interrogation: 8 octets en réponse
#define Ident_T_AIM_Asservissement 0x00A00000    // "Asservissement" en Acquittement commande
#define Ident_T_OB_Asservissement 0x00900000     // "Asservissement" en On Bus (par scéduleur)
// Pour le "Commodo Essuie Glace"
Port_8ES Commodo_EG;
#define Etat_Commodo_EG Commodo_EG.valeur
#define Valeur_Commodo_EG Commodo_EG.valeur
#define Cde_EG_Av_Int Commodo_EG.bit.GP0         // Commande Essuie Glace Avant en Intermittant
#define Entree1 Commodo_EG.bit.GP1
#define Entree2 Commodo_EG.bit.GP2
#define Cde_EG_Av_Pos1 Commodo_EG.bit.GP3 // Commande Essuie Glace Avant en Position1
#define Cde_EG_Av_Pos2 Commodo_EG.bit.GP4 // Commande Essuie Glace Avant en Position2
#define Cde_EG_Ar Commodo_EG.bit.GP5    // Commande Essuie Glace Arrière
#define Cde_Lave_Glace_Ar Commodo_EG.bit.GP6
#define Cde_Lave_Glace_Av Commodo_EG.bit.GP7
#endif
```

10.2 Fichier de définition propre à la carte ATON

```
// ***********************************************************

//  Fichier de définition pour carte controleur CAN "ATON_CAN"
//  Nom de fichier:  Aton_CAN.h
//***********************************************************
#ifndef _PELICAN_H
#define _PELICAN_H
#define SJA         0xB30280                            /* Adresse de la carte SJA    */
#define MODE        *(unsigned char *)  (SJA)           /* Registre de controle */
#define COMMAND     *(unsigned char *)  (SJA+0x01)      /* Registre de commande       */
#define STATUS      *(unsigned char *)  (SJA+0x02)      /* Registre status */
#define INTERRUPT           *(unsigned char *)  (SJA+0x03)      /* Registre d'interruption */
#define INTERRUPT_ENABLE    *(unsigned char *)  (SJA+0x04)
#define BUS_TIMING_0        *(unsigned char *)  (SJA+0x06)      /* Registre bus timing 0      */
#define BUS_TIMING_1        *(unsigned char *)  (SJA+0x07)      /* Registre bus timing 1      */
#define OUTPUT_CONTROL      *(unsigned char *)  (SJA+0x08)      /* Registre output control    */
#define ARBITRATION_LOST_CAPTURE        *(unsigned char *) (SJA+0x0B)
#define ERROR_CODE_CAPTURE              *(unsigned char *) (SJA+0x0C)
#define ERROR_WARNING_LIMIT             *(unsigned char *) (SJA+0x0D)
#define RX_ERROR_COUNTER                *(unsigned char *) (SJA+0x0E)
#define TX_ERROR_COUNTER                *(unsigned char *) (SJA+0x0F)
#define RX_FRAME_INFO                   *(unsigned char *) (SJA+0x10)
#define TX_FRAME_INFO                   *(unsigned char *) (SJA+0x10)

/* Mode SIMPLE FRAME */
#define RX_ID_1_S           *(unsigned char *) (SJA+0x11)
#define RX_ID_2_S           *(unsigned char *) (SJA+0x12)
#define RX_DATA_S           (unsigned char *) (SJA+0x13)   /* Adresse de
#define TX_ID_1_S           *(unsigned char *) (SJA+0x11)
#define TX_ID_2_S           *(unsigned char *) (SJA+0x12)
#define TX_DATA_S           (unsigned char *) (SJA+0x13)   /* Adresse de

/* Mode EXTENDED */
#define RX_ID_1_E           *(unsigned char *) (SJA+0x11)
#define RX_ID_2_E           *(unsigned char *) (SJA+0x12)
#define RX_ID_3_E           *(unsigned char *) (SJA+0x13)
#define RX_ID_4_E           *(unsigned char *) (SJA+0x14)
#define RX_DATA_E           (unsigned char *) (SJA+0x15)

#define TX_ID_1_E           *(unsigned char *) (SJA+0x11)
#define TX_ID_2_E           *(unsigned char *) (SJA+0x1
#define TX_ID_3_E           *(unsigned char *) (SJA+0x
#define TX_ID_4_E           *(unsigned cha
#define TX_DATA_E           (unsigned char

/* Les 2 modes */
#define RX_MESSAGE_COUNTER          *(unsigned cha
#define RX_BUFFER_START_ADDRESS     *(unsigned char
#define CLOCK_DIVIDER               *(unsigned char *)

/* acceptance code  et mask en mode reset
#define ACCEPT_CODE0                *(unsigned char *) (SJA+0x10)
#define ACCEPT_CODE1                *(unsigned char *) (SJA+0x11)
#define ACCEPT_CODE2                *(unsigned char *) (SJA+0x12)
#define ACCEPT_CODE3                *(unsigned char *) (SJA+0x13)

#define ACCEPT_MASK0                *(unsigned char *) (SJA+0x14)
#define ACCEPT_MASK1                *(unsigned char *) (SJA+0x15)
#define ACCEPT_MASK2                *(unsigned char *) (SJA+0x16)
#define ACCEPT_MASK3                *(unsigned char *) (SJA+0x17)

/*** Configurations des bits
/* Registre STATUS */
#define BUS_STATUS
#define ERROR_STATUS
#define TRANSMIT_STA
#define RECEIVE_STAT
#define TRANSMISS
#define TRANSMIT
#defi
#de

/*** ***** TYPES
/******
/*
/*
// ***
/** In
void init_sja1000_peli
/** Initialisa
void init_sja1
Trame Receive_Trame();
/** Envoi d'une trame */
void send_trame_peli (Message mes);
/** Reception d'un message. */
Message receive_trame_peli ();
/** Affichage des regsitres en PeliCAN */
void print_reg_peli ();
/* Affiche l'etat du STATUS REGISTER */
void show_status ();
/* Affiche le message passe en parametres sur une ligne */
void print_little (Message mes);
// initialisation de la carte CAN ATON
void Init_Aton_CAN();
char Ecrire_Trame(Trame message);
char Lire_Trame(Trame *message_recu);
void Affiche_Trame(Trame trame);
```

Page laissée vierge