PRACTICAL WORKS

CAN - V.M.D. SYSTEM

Multiplexed Didactic Vehicle

Multiplexed Didactic Vehicle (V.M.D.)
System reference: VMD 01C

Necessary software
- Integrated Development Environment
(Editor, assembler, linker, loader) Ref: EID210
- Compiler "C" Ref: EID210 101
- Real-time Core MTR86 (**option**) Ref: EID210 201

Necessary technical manuals
- The processor board EID210 001 Ref: EID210 011
- The system V.M.D. Ref : EID055 011
- The "CAN Expander" MCP25050 and Controller CAN SJA1000

Associated technical manuals of the Experiments
- The processor board EID210001 Ref: EID210041
- The simulator I/O board EID210001 Ref: EID211041
- With CAN-VMD and Real-time Core MTR86 Ref: EID050 051
- With Ethernet network board Ref: EID213 041

didalab GÉNIE ÉLECTRIQUE
Z.A. La Clef St Pierre - 5, rue du Groupe Manoukian 78990 ELANCOURT France
Tel. : 33 (0)1 30 66 08 88 - Fax : 33 (0)1 30 66 72 20
e-mail : ge@didalab.fr Web : www.didalab.fr

Edition of : 31/08/2015

Document reference : EID050041

SUMMARY

1 EXPERIMENT N°1 : SWITCH THE LIGHTS FLASH OF THE OPTICAL BLOCK

1.1 Topic

Purpose :	- Understand and use the proposed and specific data structures - Understand and use the proposed and specific functions. - Define and send a data frame to a CAN module recipient, which is accessible to a given address. - Test whether a frame has been received or not. - Display received and sent frames on the screen.
Specifications :	At first all the lamps of "Right Front lights" block are off. We would like to realize a function " chase " with 4 lamps block (after a regular time interval, we turn off the previous lamp and then turn on the next one). → The sent or received frames through the CAN circuit are displayed. → The delay is produced by the "software" (count the number of passages in the main loop) After a minimum of modifications, the program should allow to realize the same function with the other blocks " Left Front lights " and then " Right Back lights" and finally " Left Back lights ".

1.2 Necessary hardware and software :

PC Micro Computer using Windows® 95 or later
Editor Software-Assembler-Debugger
If programming in C, GNU; C / C + + Compiler Ref: EID210101
Processor board 16/32 bit 68332 microcontroller and its software environment
(Editor-Cross Assembler-Debugger) Ref: EID210001
CAN PC/104 Network board in ATON SYSTEMS Ref NIC: EID004001
CAN network with four power outputs modules for lights Ref: EID051001
USB connection cable, or if not available RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules ("energy" network)

Time : 3 hours

1.2 **Solution**

1.2.1 Analysis

Chase for the « Right Front Lights »

If we want to define the status of 4 power outputs module, the frame of type "Input Message"(IM) has to be sent back with a "Write Register" function on its register which is accessible to the "GPLAT" output port (from the technical manual of MCP25050 circuit page 22).
Indeed, seen from the module, it receives a command frame requesting it to change the status of its output register which imposes the condition of different outputs of the interface MCP25050 circuit.

For an IM "Input Message", the included register is the RXF1 which leads to the identifier base 0E 88 xx xx for the right front lights (see identifier table in Chapter 1). The 3 least significant bits must be turned to 0 (from the technical manual of MCP25050 circuit page 22) and also let the other indifferent bits turn to 0, which ultimately leads to the identifier 0E880000 (only 29 useful bits), with the label defined in the "T_Ident_IM_FVD" of "CAN_VMD.h" file.
Definition of the command frame (find in Chapter 1 of this document, the definition of the different fields of the frame for this example).

Remarks

- In the command frame, there are three parameters defined in the "data" area:
  - → Parameter "address" (in the rank 0 of "data") defines the concerned address register by writing.
    In the case before, it is GPLAT and address that is 1E h (Page 15 MCP25050 manual).
    02h+shift (note1) = 02h +1Ch =1Eh
  - → Parameter "mask" (in the rank 1 of "data") permits to store some unchanged bits of the register if they are not to be affected by the writing operation.
    In our case the power outputs are connected to the 4 least significant bits of the port. The 4 first bits should be masked with the mask value 0Fh.
  - → Parameter "value" (in the rank 2 of "data") allows to define the status of the unmasked outputs.
    In our case, to achieve the "chase" function, it will give the successive values: 00, then 01, then 02, then 04 and finally 08.

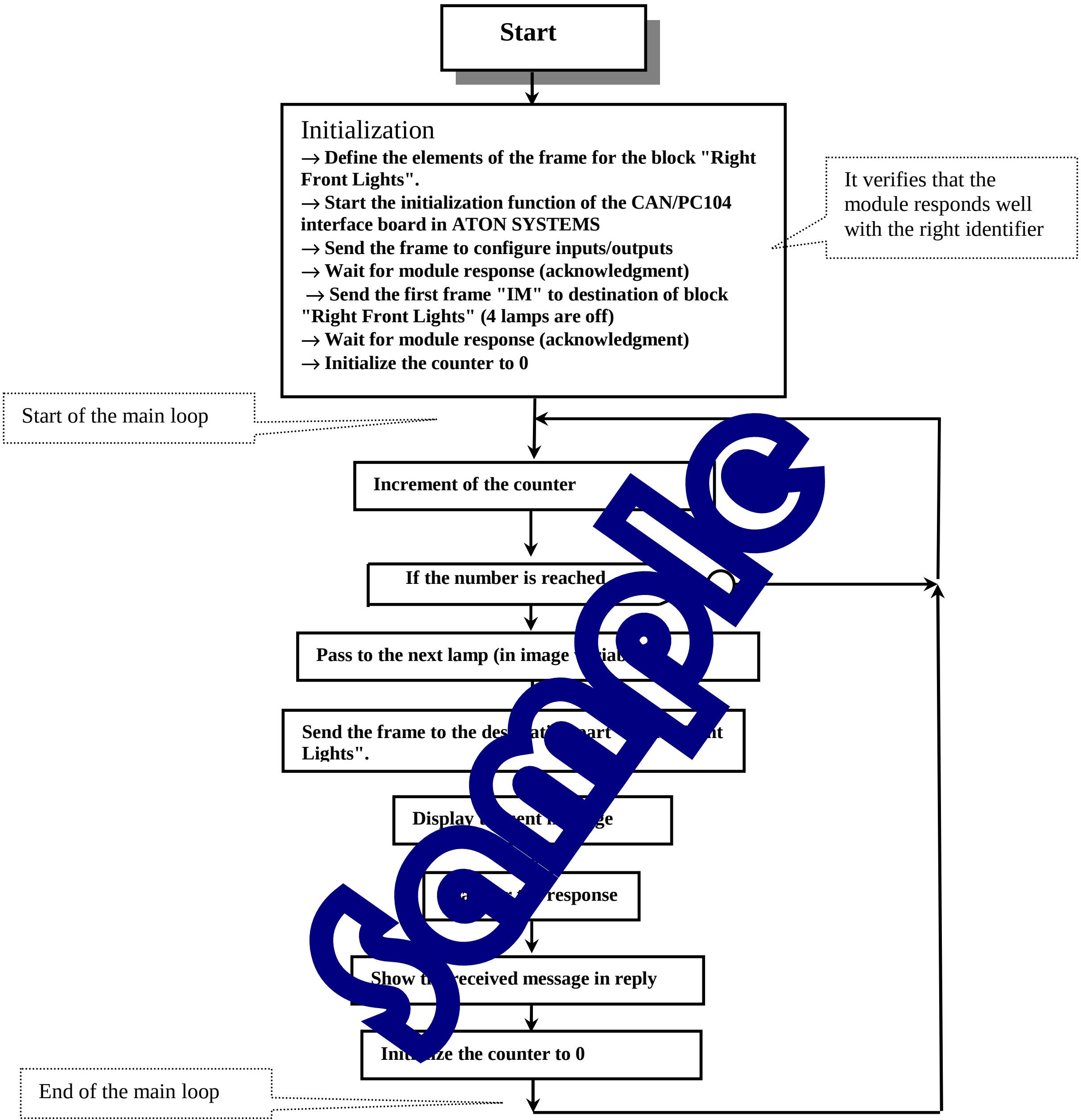
- After an IM "Input Message", the message has been received by the recipient, it should return an acknowledgment frame module whose identifier is defined by TXID1. In the case of "Right Front Lights" module, the identifier of the acknowledgment message will be 0E A0 00 00 whose label "T_Ident_AIM_FVD was defined in CAN_VMD.h file (from table in Chapter 1).

It's necessary to send the second command frame to the node only if it has answered with an acknowledgment frame "AIM" in the first time.

Chase on the other lights

The only changed things are the identifiers :
For Left Front Lights
Identifier writing message on output port (register RXF1): 0E080000 -> T_Ident_IM_FVG
Identifier acknowledgement message (register TXD1): 0E200000 -> T_Ident_AIM_FVG
For Right Back Lights
Identifier writing message on output port (register RXF1): 0F880000 -> T_Ident_IM_FRD
Identifier acknowledgement message (register TXD1): 0FA00000 -> T_Ident_AIM_FRD
For Left Back Lights
Identifier writing message on output port (register RXF1): 0F080000 -> T_Ident_IM_FRG
Identifier acknowledgement message (register TXD1): 0F200000 -> T_Ident_AIM_FRG

1.2.2 Flowchart



1.2.3 "C" Program

```
/***********************************************************************************************
*       Experiment on EID210 / CAN Network – V.M.D (Multiplexed Didactic Vehicle)
************************************************************************************************
*         EXPERIMENT n 1: SWITCH THE LIGHTS FLASH OF THE OPTICAL BLOCK
*-----------------------------------------------------------------------------------------------
*   SPECIFICATIONS :
*  *********************
*   After a regular time interval we turn off the previous lamp
*   Then turn on the next lamp (function chase)
*   -> The sent and received frames on the circuit CAN are displayed
*   -> The delay is produced by the software
*         (count the number of passages in the main loop)
*-----------------------------------------------------------------------------------------------
*  File Name:  CAN_VMD_TP1.C
************************************************************************************************/

// Included files
//*********************
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"

//==========================
//  MAIN FUNCTION
//==========================

main()
{
// Declaration of local variables
Trame Trame_Recue;
Trame T_IM_Feux;  // Frame of type IM "Input Message" commanding for [illegible]
int Compteur_Passage,Cptr_TimeOut;
char I_Message_Pb_Affiche;

// Initializations
//*****************
clsscr();
/* Initialization of SJA1000 of the ATON-Systems board " on PC104 [illegible]rcuit */
Init_Aton_CAN();
//  Definition of the frame to turn on the right front lights
// From doc SJA1000 and doc MCP25050 page 22 (function "Write Reg[illegible]e 15 (GPLAT Address)
T_IM_Feux.trame_info.registre=0x00;
T_IM_Feux.trame_info.champ.extend=1; // Work in extended m[illegible]
T_IM_Feux.trame_info.champ.dlc=0x03; // There will be 3 d[illegible] of 8 [illegible]sent[illegible]es)
T_IM_Feux.ident.extend.identificateur.ident=0x0E880000;// [illegible] the right front lights block
T_IM_Feux.data[0]=0x1F;      // first data -> "Address"[illegible]GPDDR give a I/O direction) address = 1Fh  page 16
T_IM_Feux.data[1]=0x7F;      // second data -> "Mask"[illegible] on the 4 least significant bits
T_IM_Feux.data[2]=0xF0;      // third data-> "Value"[illegible]gnificant bits are the outputs

// Send frame to define the direction of the inputs a[illegible]
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Feux);// Send frame throu[illegible]
           Cptr_TimeOut=0;
           do{Cptr_TimeOut++;}while((Lire_Tr[illegible])&&(Cptr_TimeOut<200));
           if(Cptr_TimeOut==200)
                    {if(I_Message_P[illegible])
                             {I_M[illegible]
                              got[illegible]
                              pri[illegible]e command frame in initialization \n");
                              pri[illegible]  Check [illegible]her power supply 12V is OK \n");}}
    }while(Cptr_TimeOut==200);
clsscr();
// Initialize the outputs to 0
T_IM_Feux.data[0]=0x1E;      // first da[illegible]ress" of concerned register (GPLAT defines the status of the outputs) 02h+1Ch = 1Eh
T_IM_Feux.data[1]=0x0F;      // second data[illegible]ask" -> the outputs are on the 4 least significant bits
T_IM_Feux.data[2]=0x00;      // third data-> "Value" ->  at first all outputs are 0 (lights are off)
// Send a frame to set the outputs to 0
Ecrire_Trame(T_IM_Feux);  // Send frame through the CAN network
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response "Acknowledgment"
// Initializations of the diverse variables
Compteur_Passage=0;

// Display the title
gotoxy(1,2);
printf("    EXPERIMENT  N°1:   SWITCH THE LIGHTS FLASH OF THE OPTICAL BLOCK         \n");
printf("    ****************************************************** \n");

// MAIN LOOP
//*******************
while(1)
          {Compteur_Passage++;
          if (Compteur_Passage==400000)
                   {// It's the end of the delay
                   // We pass to the next lamp by modifying the parameter "Value" of message
                   switch(T_IM_Feux.data[2])
                            {case 0 : T_IM_Feux.data[2]=0x01;
                                       break;
                             case 1 : T_IM_Feux.data[2]=0x02;
                                      break;
                             case 2 : T_IM_Feux.data[2]=0x04;
                                      break;
                             case 4 : T_IM_Feux.data[2]=0x08;
                                      break;
                             case 8 : T_IM_Feux.data[2]=0;
                                      break;
                             default : T_IM_Feux.data[2]=0;}
                   gotoxy(1,5),printf("Write on the Right Front Lights:\n");
```

```
                    Affiche_Trame(T_IM_Feux);     // Display the frame "IM" sent on the screen
                    Ecrire_Trame(T_IM_Feux);      // Send the frame through CAN network
                    do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response "Acknowledgment"
                    gotoxy(1,10);
                    printf("Received frame by response\n");
                    Affiche_Trame(Trame_Recue);   // Display the received "AIM" frame, then send to the screen
                    Compteur_Passage=0;}
          }// End of the main loop
}// End of the main function
```

2 EXPERIMENT N°2 : ACQUIRE THE STATE OF LIGHTS STALK

2.1 Topic

Purpose :	- Define and then send a remote frame to an input module, which is accessible to a defined address. - Testing whether a frame has been received. - Extract the expected information from a response frame. - Display received and sent frames on the screen. - Display the expected data on the screen.
Specifications :	After a regular time interval, we interrogate the 8 inputs' module on which is connected the lights stalk so that we can know its condition. → The sent or received frames by the CAN circuit are displayed. → The delay is produced by the "software" (count the number of passages in the main loop) → The different commands imposed by the position of the lights stalk will be displayed individually.

Necessary hardware and software :

PC Micro Computer using Windows or later
Editor Software-Assembler-Debugger
If programming in C, GNU; C / C++ Compiler Ref: EID210101
Processor board 16/32 bit 68332 microcontroller and its software environment
(Editor-Cross Assembler-Debugger) Ref: EID210001
CAN PC/104Network board in ATON SYSTEMS Ref NIC: EID004001
CAN network with four power outputs modules for lights Ref: EID051001
USB connection cable, or if not available RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules ("energy" network)

Time : 3 hours

2.2 **Solution**

2.2.1 Analysis

After a regular time interval, we interrogate the 8 inputs' module on which is connected to the lights stalk.

Definition of the remote frame which will be sent

In this case, the frame sent by the CAN controller (SJA1000 circuit CAN_PC104 board) will be regarded by the receiver (MCP25050 module) as an "Information Request Message", with a function "Read register" (see in MPC25025 technical documentation page 22).

From the table given on page 22 of the MCP25050 manual, the identifier itself will contain the address of the read register. This address is on the identifier bits ID15 to ID8 in extended mode (bits that are received and put in the RXBEID8 register). The concerned registry is GPPIN with 1Eh address "(see in MPC25025 technical documentation page 37) ..
On the other hand, the least significant 3 bits of the identifier in extended mode must be set to 1.
The identifier defined in Chapter 1 should be completed as follows:

05 04 xx xx → According to the table in Chapter 1

05 04 1E 07 → 1E: Register address; 07: Function Co[illegible]

(Only 29 bits are taken into account)

→ Definition of structured variables under "**Trame**" modu[illegible]

```
Trame T_IRM_Commodo_Feux;
```

// Frame for the interrogation of the 8 inputs' module on wh[illegible] is conn[illegible] lights stalk.

Remark: T_IRM_Commodo Structured variable onl[illegible] in[illegible]s [illegible]useful bytes, 1 byte for trame_info and 4 bytes for the identifier on extended mode (which incl[illegible]es [illegible]cerned register address by the lecture).

→ Access and definition of the different eleme[illegible] o[illegible] [illegible]uctu[illegible] variable "T_IRM_Commodo"

```
T_IRM_Commodo.trame_info.register=0x[illegible]; // [illegible] are initialized to 0
T_IRM_Commodo.trame_info.champ.ext[illegible]; [illegible]k [illegible] extended mode
T_IRM_Commodo.trame_info.champ.rt[illegible]0x[illegible]; // [illegible]rame type
T_IRM_Commodo.trame_info.champ.dl[illegible]0x[illegible] // There will be 1 data byte
T_IRM_Commodo_ad.ident.extend.iden[illegible]fica[illegible]r.ident=0x05041E07;
  //! It has 29 bits.   To s[illegible] the [illegible]s of the lights stalk
```

Labels defining the different identif[illegible]s [illegible] de[illegible]ed in the CAN_VMD.h file

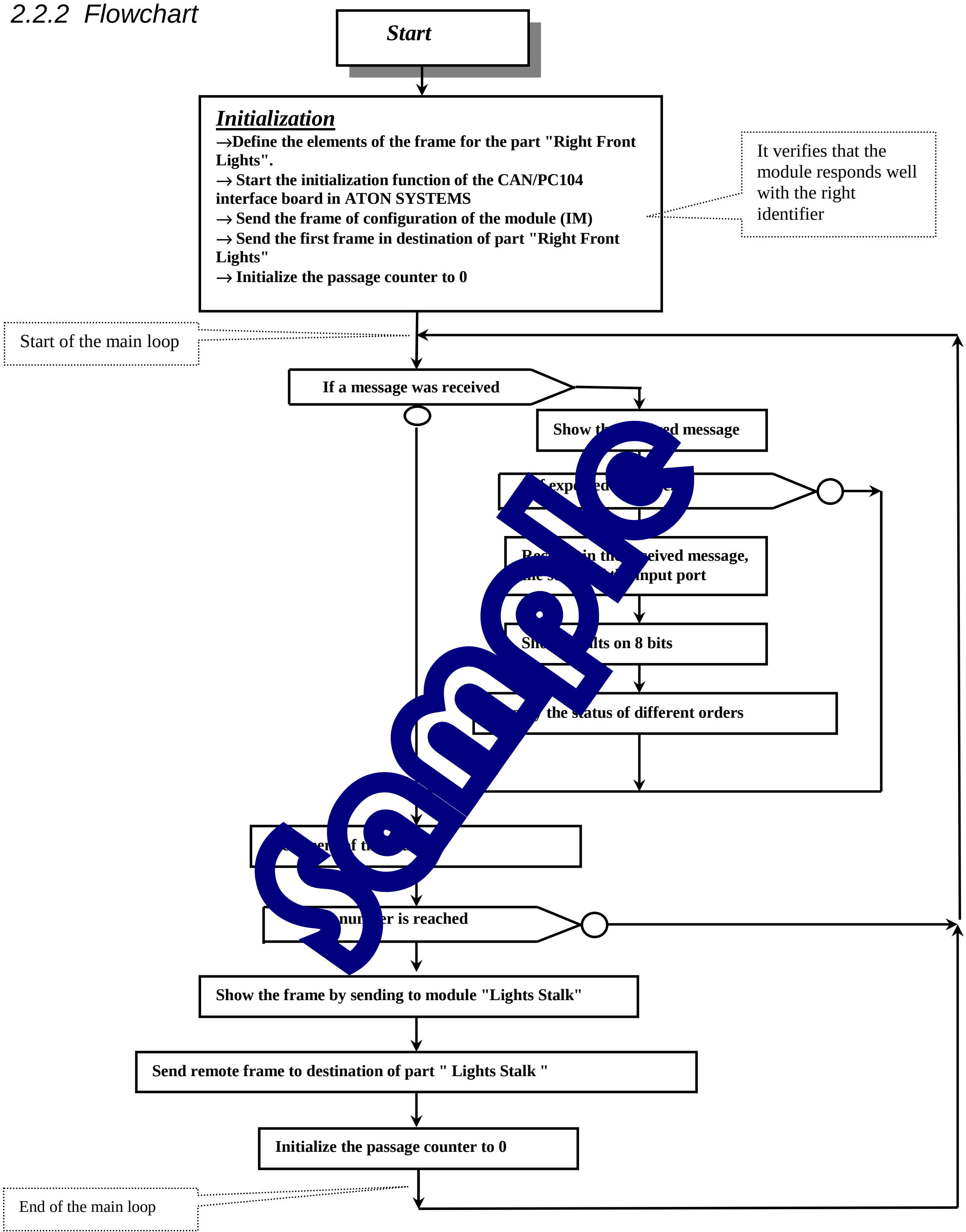
Definition of the remote fr[illegible]me [illegible] was [illegible]ceived by response

According to the definition o[illegible] identifie[illegible] given in Chapter 1, a frame with response of IRM has the same identifier as the original remote [illegible]e.
Seen from the module (the MCP25[illegible]) the response to an IRM (Information Request Message) is an OM (Output Message).
The difference with the original remote frame is that this response frame contains the parameter "value" (rank 0 of the part of "data" in the frame). This parameter is the image of the input port. Thus we recover the status of various orders.

Access to the different binary status commands

The parameter "value" of the response frame, recovered from the data in rank 0 is a byte inputs image. The different bits of this byte are extracted individually because of a data structure defined in the CAN_VMD file.

2.2.2 Flowchart



2.2.3 "C" Program

```
/*********************************************************************************************
*       Experiment on  EID210 / CAN Network – V.M.D   (Multiplexed Didactic Vehicle)
*********************************************************************************************
*         EXPERIMENT n 2 : ACQUIRE THE STATE OF LIGHTS STALK
*-------------------------------------------------------------------------------------------
*   SPECIFICAIONS :
*  *********************
*   After a regular time interval, we interrogate the 8 inputs' module
          on which is connected the lights stalk
*   -> The sent and received frames on the circuit CAN are displayed.
*   -> The input status are displayed.
*   -> The delay is produced by the software
*         (count the number of passages in the main loop)
 *-------------------------------------------------------------------------------------------
*  File Name :  CAN_VMD_TP2.C
* *****************
*********************************************************************************************/
// Included files
//*******************
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"

//==========================
//  MAIN FUNCTION
//==========================
main()
{// Definition of local variables
int Compteur_Passage;
Trame Trame_Recue;
Trame T_IRM_Commodo_Feux;    // Frame for interrogating 8E Module on  ight  S   k
                                                   // IRM -> Informati    F      ssage Frame
Trame T_IM_Commodo_Feux;     // Frame for interrogating 8E M   ule o   ig
                                                   // IM -> I     tio  Mes      >  mmand Frame
unsigned char Cptr_TimeOut,I_Message_Pb_Affiche;
// Initializations
//*****************
clsscr();
/* Initialization of the ATON-Systems board SJA1000 on PC10          /
Init_Aton_CAN();
// To initialize the connection of the inputs
T_IM_Commodo_Feux.trame_info.registre=0x00;
T_IM_Commodo_Feux.trame_info.champ.extend=1; // Work in  exte    mod
T_IM_Commodo_Feux.trame_info.champ.dlc=0x03; // There w    be     ta  f 8 bits (3 bytes)
T_IM_Commodo_Feux.ident.extend.identificateur.ident=Iden     IM_C     o_Feux;
T_IM_Commodo_Feux.data[0]=0x1F;        // first          ss"     oncerned register
                                                                DDR    es a I/O direction) address = 1Fh  page 16
T_IM_Commodo_Feux.data[1]=0x7F; // second data    "M  k"
                                                   he    put    re on the 4 least significant bits(see in page 16)
T_IM_Commodo_Feux.data[2]=0x7F; // third da     "Valu        the bits are inputs
// Send frame to define the direction of th  inp    and      ts
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Commodo_Feux);  // Se   fram        n the CAN network
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((             am  ecue)==0)&&(Cptr_TimeOut<200));
          if(Cptr_TimeOut==200)
                   {if(I_Message_P  Affi  e==0)
                             {I_M   ag     fic   1;
                              got   (2
                                  ("  No res   se to the command frame in initialization \n");
                                 int     Chec   hether the power supply 12V is OK \n");}}
    }while(Cptr_TimeOut=   0);
clsscr();
// For the remote frame se      the Li   s Stalk -> 'IRM'   (Information Request Message)
// Define the identification  d
T_IRM_Commodo_Feux.trame_info.r     tre   00;
T_IRM_Commodo_Feux.trame_info.ch        nd=1;
T_IRM_Commodo_Feux.trame_info.champ    c=0x01;
T_IRM_Commodo_Feux.trame_info.champ.rtr=1;
T_IRM_Commodo_Feux.ident.extend.identificateur.ident=Ident_T_IRM_Commodo_Feux;
                                                                    // Find the definition in the CAN_VMD.h file
Ecrire_Trame(T_IRM_Commodo_Feux); // Send the first frame
// Initialise the diverse variables
Compteur_Passage=0;
// To display the title
gotoxy(1,2);
printf("    EXPERIMENT  N°2:   ACQUIRE THE STATE OF LIGHTS STALK   \n");
printf("    **************************************************** \n");
// MAIN LOOP
//*******************
while(1)
          {  // Test whether the frame has been received
             if (1==Lire_Trame(&Trame_Recue))  // The function return 1 in this case
                   {gotoxy(4,10);
                   printf("Received frame in response is correspond to the request: It's a 'OM' (Output Message) \n");
                   Affiche_Trame(Trame_Recue);
                   if (Trame_Recue.ident.extend.identificateur.ident==Ident_T_IRM_Commodo_Feux)
                             { // Once We received the stalk status, display them
                             Etat_Commodo_Feux.valeur=~Trame_Recue.data[0];
                             gotoxy(4,16);
                             printf("Status of the inputs imposed by stalk:\n");
                             printf("    Bytes recovered and supplemented (en Hexa) =%2.2x\n",Etat_Commodo_Feux.valeur);
                             printf("    Side light= %d ,    Dipped light= %d ,   Head light=
%d\n",Cde_Veilleuse,Cde_Code,Cde_Phare);
                             printf("    Left indicator= %d   ,    Right indicator= %d\n",Cde_Clign_Gauche,Cde_Clign_Droit);
                             printf("    Horn= %d\n",Cde_Klaxon);
                             printf("    Stop light= %d\n",Cde_Stop);
                             printf("    Command Warning= %d\n",Cde_Warning);
                             }
                   }
```

```
			Compteur_Passage++;
			if (Compteur_Passage==5000)
				{Compteur_Passage=0;// It’s the end of the delay
				gotoxy(4,6);
				printf("Requested frame of stalk status: It’s an 'IRM'   Input Request Message\n");
				Affiche_Trame(T_IRM_Commodo_Feux);
				Ecrire_Trame(T_IRM_Commodo_Feux);
				}
			}// End of the main loop
		}// End of the main function
```

3 EXPERIMENT N°3 : CHECK THE FUNCTION OF AN OPTICAL BLOCK

3.1 Topic

Purpose :	- Analyze a diagram structure in order to define the hardware function realization by a software function. - Link up the remote frames and control frames to satisfy imposed specifications. - Test the response of received frames. - Extract the expected information from the response frame. - Analyze the received information and make a diagnosis. - Represent imposed specifications by a diagram of status. - Code and program a diagram of status. - Carry out cyclical actions, during a specific period.
Specifications :	We want to cycle the diff ent ptical block (Nothing, Side light, Side light + Dipped li ,S ht + H d light etc ...) and the indicator. We also realize the contro . - Verify that the ordered ret well the acknowledgment frame. - Check (by software nction) ontrolled lamps are effectively lit (by current using). → We show w d lamps and the test result. → The bli ng (of icator) will be independent to the permutation of the other lamp as e control period. We impose fol ing periods, in priority order of descending: - La nutation period: 3.5 S, ica Switching period: 1.6 S. pe ould be easily modified. m will allow an easy change of the target block.

N cessary hardware and software :

PC Micro Computer using Windows ® 95 or later
Editor Software-Assembler-Debugger
If programming in C, GNU; C / C + + Compiler Ref: EID210101
Processor board 16/32 bit 68332 microcontroller and its software environment
(Editor-Cross Assembler-Debugger) Ref: EID210001
CAN PC/104 Network board in ATON SYSTEMS Ref NIC: EID004001
CAN network with four power outputs modules for lights Ref: EID051001
USB connection cable, or if not available RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules ("energy" network)

Time : 4 hours

3.2 **Solution**

3.2.1 Analysis

Detection principle of electric power off (filament break)

The power circuit of the reference "VN05" which leads 4 power outputs Modules, generates a diagnostic signal marked "STAT" -> STATus (refer to the data sheet of the VN05 circuit).

In the case that we activate the power circuit, if the load current is close to 0 (bulb filament cut off for example), the signal "STATus" changes to 0. The output current threshold which sequences the setting to 0 for the "STATus" output is from 5 mA (minimum value) to 180 mA (maximum value).
In the case of 4 power outputs module, the LEDs who are intended to indicate whether a power output is at work or not, doesn't consume enough current to inhibit this diagnostic function.

According to the technical manual of CAN-VMD system and the structured diagram of 4 power outputs module, these 4 signals "STATus" are connected to the interface CAN MCP25050 circuit and therefore constitute diagnostic inputs that can be read through the CAN network:

- The output "STATus" of the power circuit driven by GP0 is connected to GP4,
- The output "STATus" of the power circuit driven by GP1 is connected to GP5,
- The output "STATus" of the power circuit driven by GP2 is connected to GP6,
- The output "STATus" of the power circuit driven by GP3 is connected to GP7.

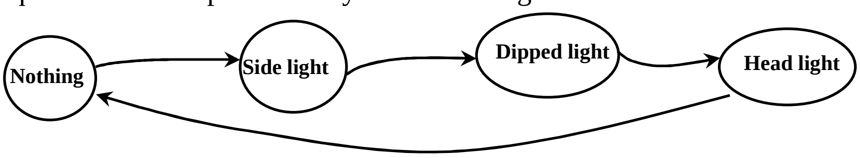
Activation of the lamps and diagnostic of the right front optical block

To turn on the lamps, it should send frame type IM "Input Message" with the "Write Register " function on its register accessible to the " GPPIN " output port (find in technical manual MCP25050 circuit page 22).
In this case, according to the table given in chapter 1, for the right front block, the identifier will be 0E880000, the " addr" parameter will be 1E (GPLAT register address - page 37 of the "Data sheet" MCP25050), the "mask " parameter will be 0F (4 outputs on the 4 least significant bits of the port), and the "value" parameter is defined by the expected [illegible]

To recover the logic status [illegible] an input port, it should send the concerned module IRM frame "Information Request message" with "Read Register" function. In this case, the identifier contains the considered address of the register (GPPIN address 1E - find in technical manual MCP25050 circuit page 37) as well as the function dipped light of 07.
Therefore, for the right front light (according to the table given in Chapter 1 in this document) the identifier to give will ultimately be 0E 841E 07 and the frame does not include a parameter in the "data" area.
The module receiver of this frame will respond with the same identifier, but with the rank 0 of the "Data" area, the status of the input port.

Lamp Swapping

We want the order of the lamp status is like following:
Nothing, then only side light, then side light + dipped light, then side light + head light.
This temporal sequence can be represented by the status diagram shown below:



We pass the previous status to the next status after the end of each "Light delay".

The same s for the "Indicator" lamp, the status change will be after the end of each "blinking delay."

Carry out the delay

We manage to implement the "programmable timer" inside the microprocessor. We set it up so that it generates an interruption every 10 mS. A counting peripheral is used to position the binary indicators informing at the end of a particular time delay.

Realize the initializations :
The two register inside the "**PICR**" and "**PITR**" microcontroller were defined in the Cpu_reg.h file

```
#define PITR   *(short *)(0xFFFA24)
#define PICR   *(short *)(0xFFFA22)
```

It is sufficient to realize the initialization as following:

```
SetVect(96,&irq_bt);         // load the auto vector
PITR = 0x0048;               // an interrupt every 10 milliseconds
PICR = 0x0760;               //  96 = 60H
```

where "**irq_bt**" is only the name of the interrupt function (see next chapter of this flowchart of this interrupt function)

Data Structure

It is useful to store in the memory, an image of the lamp status of the optical block (image of the output port module status). When we want to change the status of a lamp, we deal with one of the bits in this file. Then this image becomes a block of the control frame.

The sent data in a control frame (or recovered by a remote fram[illegible] e[illegible] 8 bits which we use the "byte_bits" data structure defined in the "Structure-Don[illegible]es[illegible]" [illegible]e.

```
/*   Union to access in one byte (BYTE) either directly or by individual[illegible] 8 b[illegible]
//*********************************************************************[illegible]***************/
        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 value;
        };
```

For light control

The image variable is defined [illegible]
byte_bits Image_Feux;

Then we define binary variab[illegible]s, and i[illegible]ges of the lamps status:
- for a front optical block

```
#define Veilleuse Image_Feux.bi[illegible]
#define Code Image_Feux.bit.b1
#define Phare Image_Feux.bit.b2
#define Clignot Image_Feux.bit.b3
```

- for a back optical block

```
#define Lumiere Image_Feux.bit.b0
#define Stop Image_Feux.bit.b1
#define Clignot Image_Feux.bit.b2
#define Autre Image_Feux.bit.b3        // on V.M.D. it's the horn which is driven
```

To light a lamp, simply:
- Change the condition in the image memory,
  Phare=1; // for example
- Transfer the image memory of the parameter "value" to the control frame (frame IM)
  T_IM_Feux.data[2]= Image_Feux.value;
- send the command frame,
  Ecrire_Trame(T_IM_Feux);   // IM -> to remember that this is a control frame

For the bulbs' function control

Similarly for controlling the bulbs status:
- The image variable is defined

    byte_bits Image_Etat_Feux;

Then we define binary variables, and images of the lamps status:

```
#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
```

To read the status, it sends a remote frame
Ecrire_Trame(T_IRM_Feux); // IRM -> to remember that this is a remote frame
In the response frame, we recover the result of reading the port in the 0 rank parameter,

    Image_Etat_Feux.value = Trame_recue.data[0];

We can then compare the logical status of the "status" bits based on the controlled outputs, for example:
- if Phare = 1 and Etat_Phare = 0 It's the bulb grilled or nothing is connected,
- if Phare = 1 and Etat_Phare = 1 It's OK.

For the coding of the status diagram
We consider a coding of decimal type for each status and a variable that will successively take the corresponding values:

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

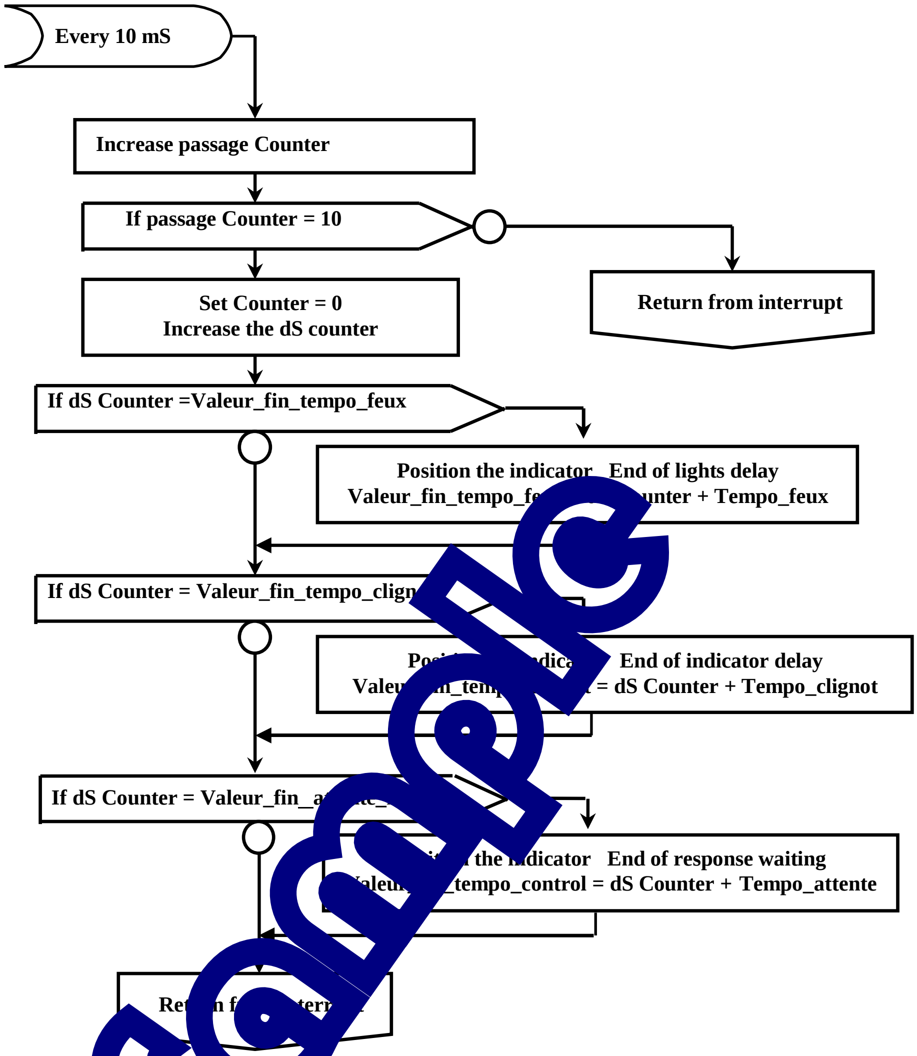
Program Structure
The main function consists of two parts:
- one part "Initialization" which is executed only one time,
- a main loop that is traversed without "turnover".

In the main loop, we only send one frame when the module receiver of the previous frame has responded:
- by an acknowledgement frame in case of a response to an "IM",
- by a response frame with parameter in the case of a response to an "IRM".

We can consider using a "Timeout" with a maximum waiting time response after sending a frame.

3.2.2 Flowchart

Flowchart describing the generation of indicators " delay limit switch" function:



Remark

In the "Initialization" it will do as the following:

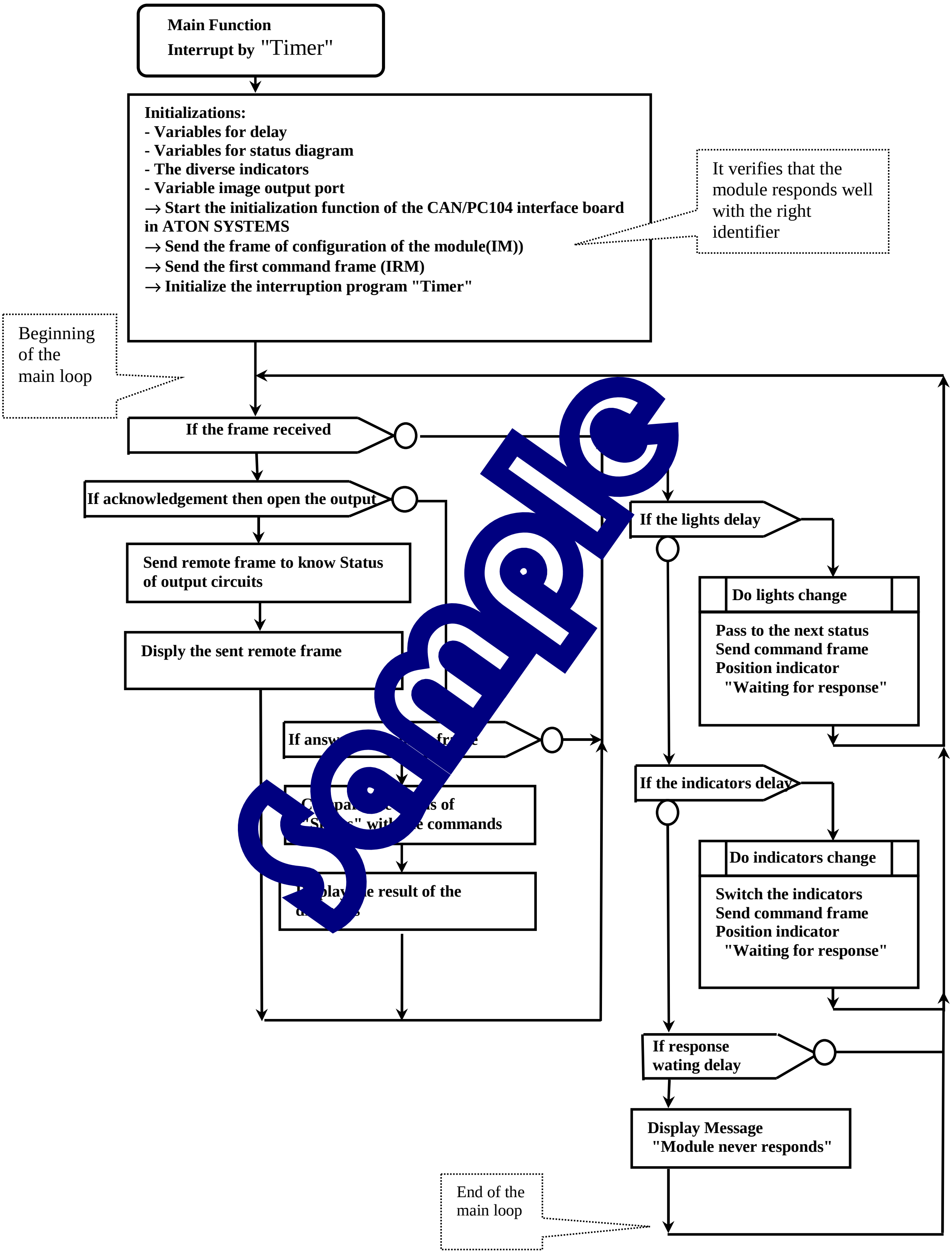
ds Counter = 0,

Value_fin_tempo_feux = Tempo_feux ,

Value_fin_tempo_clignot = Tempo_clignot,

Value_fin_attente_reponse = Tempo_attente_reponse.

General flowchart of the main function



3.2.3 "C" Program

```
/**************************************************************************************************
*        Experiment on  EID210 /       CAN Network – V.M.D  (Multiplexed Didactic Vehicle)
**************************************************************************************************
*         EXPERIMENT N°3: CHECK THE FUNCTION OF AN OPTICAL BLOCK
*-------------------------------------------------------------------------------------------------
*   SPECIFICATIONS :
*  *********************
* We want to order the different status of a right front optical block
(Nothing, Side light, Side light + Dipped light ,Side light + Head light etc ...) and the indicator.
*  We also realize the control functions:
*         - Verify that the ordered module return well the acknowledgment frame.
*         - Check (by software function) that controlled lamps are effectively lit (by current using).
*         - We show what are the concerned lamps and the test result.
 The blinking (of the indicator) will be independent to the permutation of the other lamps
as well as the control period.
* We impose the following periods, in priority order of descending:
  - Lamps permutation period: 3.5 S,
  - Indicator Switching period: 1.6 S.
*  These periods should be easily modified.
*         -  The program will allow an easy change to the target block.
*-------------------------------------------------------------------------------------------------
*  File Name :  CAN_VMD_TP3.C
* *****************

**************************************************************************************************/
// Included files
//*******************
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"

// For the delays
#define Tempo_Feux 30        // 30 ms-> 3 S
#define Tempo_Clignot 16     // 16 ms-> 1,6 S
#define Tempo_Att_Rep 40     // Response waiting for 40 ms ->4 S
// For the coding of the status diagram
#define Etat_aucune 0
#define Etat_veilleuse 1
#define Etat_code 2
#define Etat_phare 3
// Declaration of variables
//---------------------------
// For the image variables
union byte_bits Image_Feux,Image_Etat_Feux,Indicateurs;
#define Valeur_Feux Image_Feux.valeur   // For an accessible Port
#define Veilleuse Image_Feux.bit.b0
#define Code Image_Feux.bit.b1
#define Phare Image_Feux.bit.b2
#define Clignot Image_Feux.bit.b3
#define S_Veilleuse Image_Etat_Feux.bit.b4 // Side ligh[illegible]at[illegible]
#define S_Code Image_Etat_Feux.bit.b5
#define S_Phare Image_Etat_Feux.bit.b6
#define S_Clignot Image_Etat_Feux.bit.b7
// For the diverse indicators
#define I_Att_Rep_Acquit Indicateurs.bit.b0
#define I_Fin_Tempo_Feux Indicateurs.bit.b1
#define I_Fin_Tempo_Clignot Indicateurs.bit.b[illegible]
#define I_Fin_Tempo_Control Indicateurs.bit.[illegible]
#define I_Fin_Tempo_Att_Rep Indicateu[illegible]s.bit[illegible]
#define I_Att_Rep_Interrog Indicate[illegible]it.[illegible]
#define I_Autorise_Emis_Mes Indica[illegible]rs[illegible]
#define I_Message_Pb_Affiche Indic[illegible]eur[illegible]it[illegible]
// Declaration of frames
Trame Trame_Recue;
Trame Trame_Envoyee;
Trame T_IM_Feux;   // Frame of type "In[illegible]ssa[illegible] to command 4 power outputs module
Trame T_IRM_Feux;  // Frame of type "Inf[illegible]ion[illegible]equest Message" to interrogate the lamps’ conditions
// For comparison of identifiers between S[illegible]F[illegible]e <-> Received Frame
#define Ident_Trame_Envoyee Trame_Envoyee.ide[illegible]extend.identificateur.ident
#define Ident_Trame_Recue Trame_Recue.ident.extend.identificateur.ident

// For the delays
WORD Compteur_Passage,Compteur_dS; // dS -> ms
WORD Valeur_Fin_Tempo_Feux,Valeur_Fin_Tempo_Clignot,Valeur_Fin_Tempo_Att_Rep;
// For the status diagram
unsigned char Etat;
// For the control of the communication
int Cptr_TimeOut,Temp;


//  Interrupt function "Time Base"
//===================================
void irq_bt()
// Function runs every 10 mS
{Compteur_Passage++;
if(Compteur_Passage==10)  // A 1/10 second has passed
          {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;}
          }
} // End of the interrupt function


//=========================
```

```
//  MAIN FUNCTION
//==========================
main()
{
// Initializations
//*****************
clsscr();
//  Definition of frames to enable or read an optical block
// According to doc SJA1000 and doc MCP25050 pages 22 (function "Write Register") and 37 (Address GPPIN)
// For the command frame  -> IM
T_IM_Feux.trame_info.registre=0x00;
T_IM_Feux.trame_info.champ.extend=1; // Work in extended mode
T_IM_Feux.trame_info.champ.dlc=0x03; // There will be 3 data of 8 bits (3 bytes)
T_IM_Feux.ident.extend.identificateur.ident=Ident_T_IM_FVD; // The identifier Left Front Light
                                                                                    // See the definition in CAN_VMD.h
// For the remote frame ->  IRM  (Information Request Frame)
T_IRM_Feux.trame_info.registre=0x00;
T_IRM_Feux.trame_info.champ.extend=1;
T_IRM_Feux.trame_info.champ.dlc=0x01;
T_IRM_Feux.trame_info.champ.rtr=1;
T_IRM_Feux.ident.extend.identificateur.ident=Ident_T_IRM_FVD; // see the definition in CAN_VMD.h

/* Initialization of SJA1000 of the ATON-Systems board on PC104 circuit */
Init_Aton_CAN();
// Send the frame to define the direction of the inputs and outputs
T_IM_Feux.data[0]=0x1F;       // first data -> "Address" of concerned register -> GPDDR
T_IM_Feux.data[1]=0x7F;       // second data -> "Mask" -> the outputs are on the 4 least significant bits(see in page 16)
T_IM_Feux.data[2]=0xF0;       // third data-> "Value" -> the 4 least significant bits are the outputs
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Feux);// Send frame through the CAN network
           Cptr_TimeOut=0;
           do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<2
       if(Ident_Trame_Recue!=Ident_T_AIM_FVD)Cptr_TimeOut=200; // Test whether     ident   er is correct or not
           if(Cptr_TimeOut==200)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response to the command f   e in     izatio   n");
                               printf("  Check whether the power supp    2V is
                    for(Temp=0;Temp<100000;Temp++);} // To wait for a wh
    }while(Cptr_TimeOut==200);
clsscr();
// For the group of the indicators
Indicateurs.valeur=0;
Etat = Etat_aucune;
Valeur_Feux=0x00;
// we send the first frame through the bus -> initial status of t   outp      h are recover to 0
// Recover the output to 0
T_IM_Feux.data[0]=0x1E;       // first data -> "Address" of            ister   LAT define the stat of the outputs) 03h+1Ch = 1Eh
T_IM_Feux.data[1]=0x0F;       // second data -> "Mask" ->   e outpu         th    least significant bits
T_IM_Feux.data[2]=0x00;       // third data-> "Value" ->    fi   all          at 0 (lamps off)

Ecrire_Trame(T_IM_Feux);
Trame_Envoyee = T_IM_Feux;
I_Att_Rep_Acquit=1;
// For time base and the delay
//****************************************
SetVect(96,&irq_bt);          // load the auto
PITR = 0x0048;                // an interrupt   ery 10 m       nds
PICR = 0x0760;                //  96 = 60H
Compteur_Passage = 0,Compteur_dS = 0
Valeur_Fin_Tempo_Feux = Tempo_Feux
Valeur_Fin_Tempo_Clignot = Tempo_C  gno
Valeur_Fin_Tempo_Att_Rep = Tempo_A  _Rep
I_Autorise_Emis_Mes=1;
//   Display the title
gotoxy(1,1);
printf("    EXPERIMENT N°3:   OPEN THE LIG    AN   ONTROL THE LAMPS STATUS   \n");
printf("    ******************************     *** ******************\n");
// Main loop
//*******************
while(1)
          {
          if (1==Lire_Trame(&Trame_Recue)) //If the frame received, the function return to 1
                    { // We just received a frame in response
                     gotoxy(1,3);  // To clear the warning message of the no reply address module
                     printf("                                                                               \n");
                    if(I_Att_Rep_Acquit)
                              { // Wait for an acknowledgment from the expected frame after sending a command frame
                                I_Att_Rep_Acquit=0;
                                I_Autorise_Emis_Mes=1;
                                gotoxy(1,8);
                                printf("    Ackowlegement from the frame by 'IM' (Input Message)\n");
                                Affiche_Trame(Trame_Recue);
                                // You can send remote frame to check the status of the lights
                                Ecrire_Trame(T_IRM_Feux);
                               Valeur_Fin_Tempo_Att_Rep = Compteur_dS+Tempo_Att_Rep;
                               I_Fin_Tempo_Att_Rep=0;
                                gotoxy(1,12);
                                printf("    Sent remote frame-> 'IRM' (Information Request Message) \n");
                                printf("    In order to test if the lights work well \n");
                                Trame_Envoyee = T_IRM_Feux;
                                Affiche_Trame(Trame_Envoyee);
                                I_Att_Rep_Interrog=1;
                              }
                       else if(I_Att_Rep_Interrog)
                                        {// Wait for a response frame by interrogation
                                         I_Att_Rep_Interrog=0;
                                         I_Autorise_Emis_Mes=1;
                                         gotoxy(1,16);
                                         printf("    Response frame to the interrogation -> 'OM' (Output Message) \n");
                                         Affiche_Trame(Trame_Recue);
```

```
                    // Analysis of the status of bulbs and display diagnostic result
                               Image_Etat_Feux.valeur=Trame_Recue.data[0];
                               if(Veilleuse==1 && S_Veilleuse==0)
                                        {gotoxy(1,20),printf("!!    Problem on Side light \n");}
                               if(Veilleuse==1 && S_Veilleuse==1)
                                        {gotoxy(1,20),printf("!!     Side light   OK                  \n");}
                               if(Code==1 && S_Code==0)
                                        {gotoxy(1,21),printf("!!     Problem on Dipped light         \n");}
                               if(Code==1 && S_Code==1)
                                        {gotoxy(1,21),printf("!!     Dipped light   OK             \n");}
                               if(Phare==1 && S_Phare==0)
                                        {gotoxy(1,22),printf("!!     Problem on head light   \n");}
                               if(Phare==1 && S_Phare==1)
                                        {gotoxy(1,22),printf("!!     Head light   OK                  \n");}
                               if(Clignot==1 && S_Clignot==0)
                                        {gotoxy(1,23),printf("!!     Problem on Indicator      \n");}
                               if(Clignot==1 && S_Clignot==1)
                                        {gotoxy(1,23),printf("!!     Indicator   OK               \n");}
                    }
          }
           if(I_Fin_Tempo_Feux)
                    {I_Fin_Tempo_Feux=0;
                    // We passe to the next status
                    switch(Etat)
                              {case Etat_aucune :          {Etat=Etat_veilleuse;
                                                                   Veilleuse =1; }
                               break;
                               case Etat_veilleuse :       {Etat=Etat_code;
                                                                   Code=1;}
                               break;
                               case Etat_code :   {Etat=Etat_phare;
                                                                   Code=0,Phare=1;}
                     break;
                               case Etat_phare :  {Etat=Etat_aucune;
                                                                   Veilleuse=0,Phare=0;}
                               break;}
                    // It sends the command frame with the updated sta[illegible]
                     T_IM_Feux.data[2]=Valeur_Feux;
                    if(I_Autorise_Emis_Mes)
                              {Ecrire_Trame(T_IM_Feux);  // [illegible] com[illegible]d [illegible]gh the CAN network
                              gotoxy(1,5);
                              printf("    Command Frame    [illegible]IM' ([illegible]ssage[illegible]);
                              Trame_Envoyee = T_IM_Feux;
                              Affiche_Trame(Trame_Envoyee);
                              I_Att_Rep_Acquit=1;
                              I_Autorise_Emis_Mes=0;
                              Valeur_Fin_Tempo_Att_Rep = [illegible]mpteur_d[illegible]Att_Rep;
                              I_Fin_Tempo_Att_Rep=0;}}
           else if(I_Fin_Tempo_Clignot)
                    {
                     I_Fin_Tempo_Clignot=0;
                     // We chang the status of the[illegible]
                     Clignot=~Clignot;
                    // We send the command fram[illegible]
                    T_IM_Feux.data[2]=Valeur_Fe[illegible]
                     if(I_Autorise_Emis_Mes)
                              {Ecrire_Trame[illegible]IM_Feux)[illegible]nd command frame through the CAN network
                              Trame_Envoyee[illegible]T_[illegible]ux;
                              I_Att_Rep_Acq[illegible]=1;
                              I_Autorise_Emi[illegible]s=0;
                              Valeur_Fi[illegible]Rep[illegible]Compteur_dS+Tempo_Att_Rep;
                              I_Fin_[illegible]
           else if(I_Fin_Tempo_Att_[illegible])
                    // It will tak[illegible]oo[illegible]for[illegible]o wait for a response!
                     {clss[illegible];
                    gotox[illegible]
                    prin[illegible]"  [illegible]XP[illegible]MENT N° [illegible]OPEN THE LIGHTS AND CONTROL THE LAMPS STATUS\n");
                    prin[illegible]" [illegible]***************************************** \n");
                    prin[illegible]"!! [illegible]module address never responds        !!\n");
                    Ecrir[illegible]ame(T_IRM[illegible]ux); // A query is repeated
                    Valeur_[illegible]o_At[illegible]Rep = Compteur_dS+Tempo_Att_Rep; // The timer is recover
                    I_Fin_Tempo_A[illegible]Rep[illegible]; }
        } // End of the main loop
} // End of the main function
```

4 EXPERIMENT N°4 : COMMAND THE LAMPS BY LIGHTS STALK

4.1 Topic

Purpose :	- Carry out a control application commanding a controllable system by the CAN network. - Display system status on the screen. - Carry out a precise delay. - Realize the tasks of verification and control.
Specifications :	After a regular time interval, fro[illegible]quired module on which is connected with the lights stalk, so th[illegible] we [illegible] status. According to the status [illegible] stalk [illegible]e activate the different lamps of front and back blocks. → The necessary dela[illegible] indicators is realized by "programmable timer" (inside the mi[illegible]). → The different com[illegible]sed by the position of the lights stalk will be displayed indiv[illegible] → It contro[illegible]f[illegible]ctio[illegible] of different blocks.

N[illegible]hardware and software :

PC Micro Computer using W[illegible]ndows [illegible] or later
Editor Software-Assembler-De[illegible]er
If programming in C, GNU; C / C[illegible]+ [illegible]mpiler Ref: EID210101
Processor board 16/32 bit 68332 microcontroller and its software environment
(Editor-Cross Assembler-Debugger) Ref: EID210001
CAN PC/104 Network board in ATON SYSTEMS Ref NIC: EID004001
CAN network with:

- 8 logic inputs module for the lights stalk Ref: EID050001
- 4 power outputs module for left/right back/front lights Ref: EID051001

USB connection cable, or if not available RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules ("energy" network)

Time : 2×4 hours

4.2 Solution

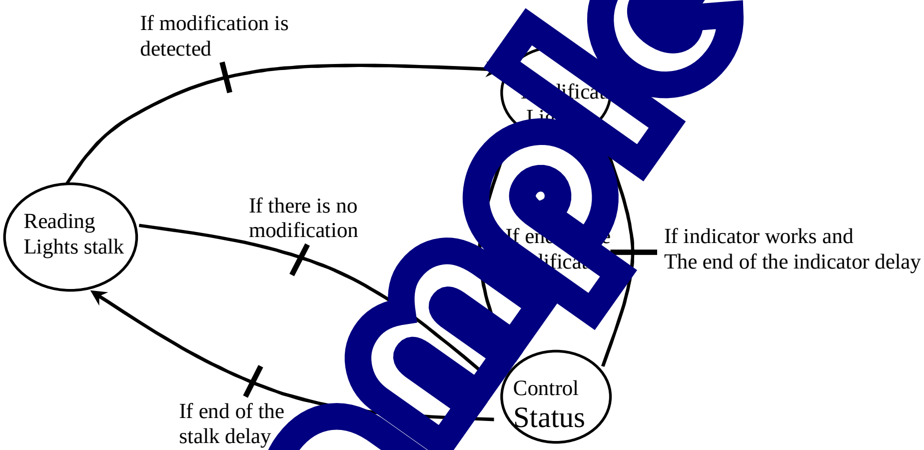
4.2.1 Analysis

Work to do :

Main task
We consider a cyclic operation where the state of the lights stalk is required after a regular time interval imposed by a time base. The next state of the lights stalk is then compared to the previous state (inherited once before). If a position change has been detected, then a light conditions change begins. (We control successively 4 blocks with the new values)

Sub-task
→ We interrogate successively 4 blocks and check if the values of "Status" corroborate the sent commands. If a different is detected, an alarm message is displayed.
→ After sending a frame, we will verify that the received frame in response is correct (acknowledgment of module having received a control frame, or coherent response module having received a remote frame).

Main status diagram
We can implement the overall operation by the status diagram as the following:



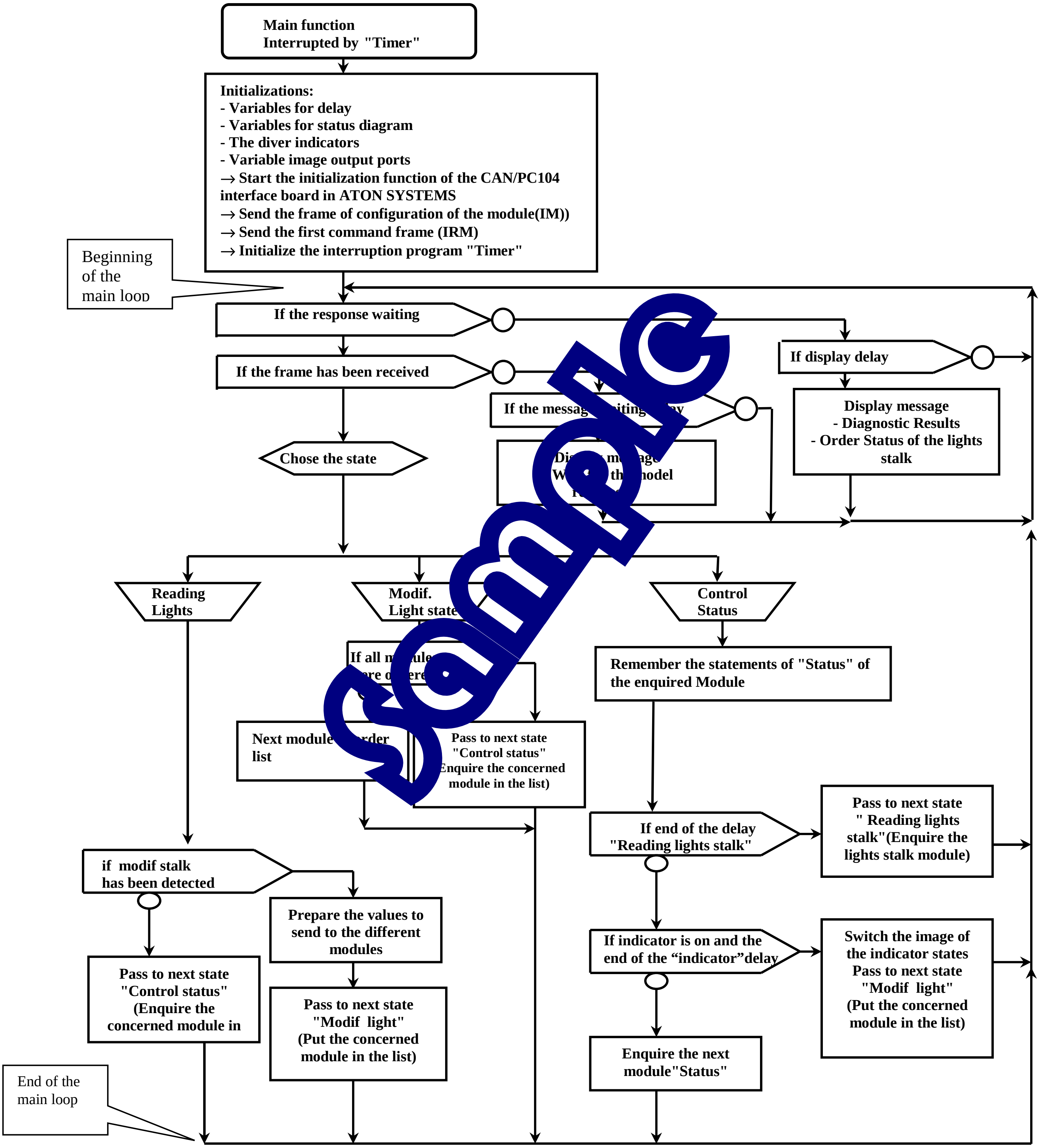
"Light Modification" Condition
It sends a command frame (frame [illegible]) each block (see EX n° 1). We decide to send control frames in the following order:
→ Left Front Lamp (LFL),
→ then Right Front Lamp (RFL),
→ then Left Back Lamp (LBL),
→ at last Right Back Lamp (RBL).
Only two pieces of information are to change when we pass one part to another:
→ the identifier,
→ the parameter "Value".

"Reading Lights Stalk" Condition
It sends a remote frame (frame type IRM) to 8 logic inputs module on which is connected to the lights stalk (See EX n°2). The Analysis of the response frame and the comparison with the stored status can detect every change.

"Control status" Condition
It sends remote frame (frame-type IRM) to 4 different power outputs module on which is connected the blocks (See EX n°3).

4.2.2 Flowchart

For the indicators of " the end of delay" function see EX n°3
Main function general flowchart



4.2.3 "C" Program

```
/**************************************************************************************************
*       Experiment on  EID210 / CAN Network  V.M.D   (Multiplexed Didactic Vehicle)
***************************************************************************************************
*         EXPERIMENT N°4: COMMAND THE LIGHTS BY LIGHTS STALK
*--------------------------------------------------------------------------------------------------
*   SPECIFICATIONS :
*  *********************
* After a regular time interval, from the enquired module on which is connected with the lights stalk,
* so that we can know its status.
* According to the status of the lights stalk, we activate the different lamps of front and back blocks
* The necessary delay to operate the indicators is realized by "programmable timer" (inside the microprocessor).
* The different commands imposed by the position of the lights stalk will be displayed individually
* We also realize the control functions:
*         - Check (by software function) that controlled lamps are effectively lit
*         - We show what are the concerned lamps and the test result.
*         - The blinking (of the indicator) will be independent to the permutation of the other lamps
*           as well as the control period
*         - It detects if a module does not respond.
* We impose the following periods, in priority order of descending:
*         - Indicator switching period 0,8 S,
 *        - Lights stalk reading period 0.2 S,
*         - Detection of no response delay 1S
*  These periods should be easily modified.
*         -  The program will allow an easy change of the target block.
*-------------------------------------------------------------------------------------------------
*  File Name:  CAN_VMD_TP4.C
* *****************

***************************************************************************************************/

// Included files
//*******************
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"

void Passer_a_Etat_Modif_Feux(void);
void Passer_a_Etat_Control_Stat(void);
// Declaration of constants
//----------------------------

// For the delays
#define Tempo_Commodo 2                // 2 ms-> 0,2 S
#define Tempo_Clignot 8                // 8 ms -> 0,8 S
#define Tempo_Att_Rep 20      // Response waiting time 20 ms -> 2[illegible]
#define Tempo_Affichage 10    // Response waiting time 10 ms [illegible]

// For the coding of the status diagram
#define Etat_Lect_Commodo_Feux 0
#define Etat_Modif_Feux 1
#define Etat_Control_Stat 2

// Declaration of variables
//--------------------------

// For the diverse indicators (binary variables[illegible]
union word_bits Indicateurs;
#define I_Attente_Reponse Indicateurs.bit.b0[illegible]
#define I_Fin_Tempo_Commodo Indicateu[illegible]s.bit[illegible]
#define I_Fin_Tempo_Clignot Indicat[illegible]bit[illegible]
#define I_Fin_Tempo_Affichage Indi[illegible]3
#define I_Fin_Tempo_Att_Rep Indica[illegible]urs[illegible]
#define I_En_Att_Rep Indicateurs.b[illegible].b5
#define I_Clignot_Gauche Indicateu[illegible]bit.b6
#define I_Clignot_Droit Indicateurs.[illegible]
#define I_Message_Pb_Affiche Indicateur[illegible]b8

// Declaration of frames
Trame Trame_Recue;
Trame Trame_Envoyee;
Trame T_IM;         // Frame of type "Input Message" to command 4 power outputs module
Trame T_IRM;        // Frame of type "Information Request Message" to interrogate the lamps' conditions
                              // or for the lights stalk

// For comparison of identifiers between Sent Frame <-> Received Frame
#define Ident_Trame_Envoyee Trame_Envoyee.ident.extend.identificateur.ident
#define Ident_Trame_Recue Trame_Recue.ident.extend.identificateur.ident
#define Ident_T_IRM  T_IRM.ident.extend.identificateur.ident
#define Ident_T_IM  T_IM.ident.extend.identificateur.ident
#define Valeur_T_IM T_IM.data[2]

// For the delays
WORD Compteur_Passage,Compteur_dS; // dS -> ms
WORD Valeur_Fin_Tempo_Commodo,Valeur_Fin_Tempo_Clignot,Valeur_Fin_Tempo_Affichage,Valeur_Fin_Tempo_Att_Rep;
// For the status diagram
unsigned char Etat,Rang_Control_Stat,Rang_Modif_Feux;
// For the memory
unsigned char Valeur_Commodo_Feux_Mem;
//  Interruption function "Time Base"
void irq_bt()
// Function runs every 10 mS
{Compteur_Passage++;
if(Compteur_Passage==10) // 1/10 second has passed
          {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;}
          }
} // End of the interrupt function

//==========================
//  MAIN FUNCTION
//==========================
main()
{
int Cptr_TimeOut,Temp;
// Initializations
//*****************
clsscr();
/* Initialization of SJA1000 of the ATON-Systems board on PC104 circuit */
Init_Aton_CAN();
//  Definition of frames to enable or read an optical block
// According to doc SJA1000 and doc MCP25050 pages 22 (function "Write Register") and 37 (GPPIN Address)
// For the command frame  -> IM
T_IM.trame_info.registre=0x00;
T_IM.trame_info.champ.extend=1; // Work in extended mode
T_IM.trame_info.champ.dlc=0x03; // There will be 3 data of 8 bits (3 bytes)
Ident_T_IM=Ident_T_IM_FRD;// The identifier Right Back Light
T_IM.data[0]=0x1F;  // first data -> "Address" of concerned register -> GPDDR
T_IM.data[1]=0x7F;  // second data -> "Mask" -> the outputs are on the 4 least significant bits(see in page 16
T_IM.data[2]=0xF0;  // third data-> "Value" -> the 4 least significant bits are the outputs
//Setup of the register direction and
//Check of the presence of modules
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Send the first frame through the CAN network
           Cptr_TimeOut=0;
           do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
       if(Ident_Trame_Recue!=Ident_T_AIM_FRD)Cptr_TimeOut=200; // Test whether the identifier is correct or not
           if(Cptr_TimeOut==200)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response in Right Back Lamp to the command frame in initialization \n");
                               printf("  Check whether the power supply 12V is OK \n");}
                    for(Temp=0;Temp<100000;Temp++);} // To wait for a while!
    }while(Cptr_TimeOut==200);

Ident_T_IM=Ident_T_IM_FRG;// It's the identifier Left Back Light
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Send the first frame through the CAN network
           Cptr_TimeOut=0;
           do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
       if(Ident_Trame_Recue!=Ident_T_AIM_FRG)Cptr_TimeOut=200; // Test whether the identifier is correct or not
           if(Cptr_TimeOut==200)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response in Right Back Lamp to the command frame in initialization \n");
                               printf("  Check whether the power supply 12V is OK \n");}
                    for(Temp=0;Temp<100000;Temp++);} // To wait for a while!
    }while(Cptr_TimeOut==200);

Ident_T_IM=Ident_T_IM_FVG;// It's the identifier Left Front Light
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Send the first frame through the CAN network
           Cptr_TimeOut=0;
           do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
       if(Ident_Trame_Recue!=Ident_T_AIM_FVG)Cptr_TimeOut=200; // Test whether the identifier is correct or not
           if(Cptr_TimeOut==200)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response in Right Back Lamp to the command frame in initialization \n");
                               printf("  Check whether the power supply 12V is OK \n");}
                    for(Temp=0;Temp<100000;Temp++);} // To wait for a while!
    }while(Cptr_TimeOut==200);

Ident_T_IM=Ident_T_IM_FVD;// It's the identifier Right Front Light
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Send the first frame through the CAN network
           Cptr_TimeOut=0;
           do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<200));
       if(Ident_Trame_Recue!=Ident_T_AIM_FVD)Cptr_TimeOut=200; // Test whether the identifier is correct or not
           if(Cptr_TimeOut==200)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response in Right Back Lamp to the command frame in initialization \n");
                               printf("  Check whether the power supply 12V is OK \n");}
                    for(Temp=0;Temp<100000;Temp++);} // To wait for a while!
    }while(Cptr_TimeOut==200);

Ident_T_IM=Ident_T_IM_Commodo_Feux;// It's the identifier Lights stalk
T_IM.data[2]=0xFF;  // third data-> "Value" -> the 8 bits are the input
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Send the first frame through the CAN network
           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 whether the identifier is correct or not
           if(Cptr_TimeOut==200)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
```

```
                              printf(" No response in Lights stalk to the command frame in initialization \n");
                              printf("  Check whether the power supply 12V is OK \n");}
                    for(Temp=0;Temp<100000;Temp++);} // To wait for a while!
    }while(Cptr_TimeOut==200);
clsscr();

// Recover the output to 0 (GPLAT register)
T_IM.data[0]=0x1E;  // first data -> "Address" of concerned register (GPLAT define the stat of the outputs)03h+1Ch = 1Eh
T_IM.data[1]=0x0F;  // second data -> "Mask" -> the outputs are on the 4 fable sufficient bits
T_IM.data[2]=0x00;  // third data-> "Value" -> at first all outputs are at 0 (lamps off)

// For the interrogative trame ->  IRM  (Information Request Frame)
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; // We begin by reading the data of lights stalk
Etat = Etat_Lect_Commodo_Feux;
Valeur_Commodo_Feux_Mem=0;
// Send the first frame through the CAN network
Ecrire_Trame(T_IRM);
Trame_Envoyee = T_IRM;
I_Attente_Reponse=1;
// For the "Control Status" condition
Rang_Control_Stat=0;
// For the "Modif Lights" condition
Rang_Modif_Feux=0;
// For all indicators
Indicateurs.valeur=0;
// Send the first frame through the CAN network
Ecrire_Trame(T_IRM);
while((Lire_Trame(&Trame_Recue)==0));
Trame_Envoyee = T_IRM;
I_Attente_Reponse=1;


// For time base and the delay
//***************************************
SetVect(96,&irq_bt);          // load the auto vector
PITR = 0x0048;                // an interrupt every 10 milliseconds
PICR = 0x0760;                //  96 = 60H
// For the delays
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;
//   Display the title
gotoxy(1,2);
printf("    EXPERIMENT N°4:   COMMAND THE LIGHTS BY LIGHTS STALK   n");
printf("    ****************************************************   ***   \n");

// Main loop
//*******************
while(1)
          {
          if(I_Attente_Reponse) // A frame is waited
                    {
                     if (1==Lire_Trame(&Trame_Recue)) //If      received, the function return to 1
                              // A frame has been reci    d
                              {I_Attente_Reponse=
                               I_Fin_Tempo_Att     ,        t the reply waiting delay
                               Valeur_Fin_Tem  Att_Rep      ur_dS + Tempo_Att_Rep;
                               I_En_Att_Rep=
                               if(Et   Etat ect        x)
                                         I    in        ing lights stalk " state
                                        {
                                         f(     e_Recue.ident.extend.identificateur.ident==Ident_T_IRM_Commodo_Feux)
                                         {aleur_Commodo_Feux=~(Trame_Recue.data[0]); //It recovers the lights stalk state
                                          if(Valeur_Commodo_Feux!= Valeur_Commodo_Feux_Mem)
                                          // We detected a change in the stalk state
                                                  {Valeur_Commodo_Feux_Mem = Etat_Commodo_Feux.valeur; // Store in the memory
                                                   // Pre-define the condition of the different lamps
                                                   // Combinations defined in 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) // If the order for the head light
                                                            {Valeur_FVG=Cde_FV_P,Valeur_FVD=Cde_FV_P;  // Front lamps
                                                             Valeur_FRG=Cde_FR_P,Valeur_FRD=Cde_FR_P;} // Back lamps
                                                   else if(Cde_Code) // If the order for the dipped light
                                                            {Valeur_FVG=Cde_FV_C,Valeur_FVD=Cde_FV_C;  // Front lamps
                                                             Valeur_FRG=Cde_FR_C,Valeur_FRD=Cde_FR_C;} // Back lamps
                                                   else if(Cde_Veilleuse) // If the order for the side light
                                                            {Valeur_FVG=Cde_FV_V,Valeur_FVD=Cde_FV_V;  // Front lamps
                                                             Valeur_FRG=Cde_FR_V,Valeur_FRD=Cde_FR_V;} // Back lamps
                                                   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;}  // To recover the indicator delay
                                                  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;}  // To recover the indicator delay
                                                   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();
                    } // End if the lights stalk order is detected
                    else      Passer_a_Etat_Control_Stat(); } // End of the "Reading Lights stalk" case
else      if(Etat==Etat_Modif_Feux)
                    {// We are in the "Modif Lamps" case
                    //We prepare he frame to command the following lamps
                    switch(Rang_Modif_Feux) //Following modules in the list
                    {// The left front lights has already been ordered
                     case 1 : {Ident_T_IM=Ident_T_IM_FVD; //Right Front Lamp
                                   Valeur_T_IM=Valeur_FVD;
                                   Rang_Modif_Feux++;
                                   Ecrire_Trame(T_IM); // Send the frame
                                   Trame_Envoyee =T_IM;
                                   I_Attente_Reponse=1;}
                     break;
                     case 2 : {Ident_T_IM=Ident_T_IM_FRD; //Right Back Lamp
                                   Valeur_T_IM=Valeur_FRD;
                                   Rang_Modif_Feux++;
                                   Ecrire_Trame(T_IM); // Send the frame
                                   Trame_Envoyee =T_IM;
                                   I_Attente_Reponse=1;}
                     break;
                     case 3 : {Ident_T_IM=Ident_T_IM_FRG; //Left Back Lamp
                                   Valeur_T_IM=Valeur_FRG;
                                   Rang_Modif_Feux++;
                                   Ecrire_Trame(T_IM); // Send the frame
                                   Trame_Envoyee =T_IM;
                                   I_Attente_Reponse=1;
                              }
                     break;
                     default :          // It's the end of this case
                                   // We return     ontrol status
                                   {Passer_a_    _Cont    _Stat();}
                     break;
                    }
                    } // End of the " mo    ligh    ca
else      if(Etat==Etat_Control_Stat)
                    {//while(I_Fin_Tem    Affic     ){};
                     //I_Fin_Tempo_Affic    =0;
                     switch(Rang_Control_S      // F     ing module in the list
                              {case 0 :              {//     rol of the Left Front Lamp
                                                      ler   Status_FVG=Trame_Recue.data[0];
                                                           may pass to the next lamp
                                                      den   _T_IRM=Ident_T_IRM_FVD;} //Right Front lamp
                               break
                               case     :               / Control of the Right Front  Lamp
                                                      aleur_Status_FVD=Trame_Recue.data[0];
                                                      // We may pass to the next lamp
                                                        nt_T_IRM=Ident_T_IRM_FRD;} //Right Back lamp
                               break;
                               c    2 :                / Control of the Right Back  Lamp
                                                      Valeur_Status_FRD=Trame_Recue.data[0];
                                                      // We may pass to the next lamp
                                                      Ident_T_IRM=Ident_T_IRM_FRG;} //Left Back lamp

                                                      {// Control of the Left Back Lamp
                                                      Valeur_Status_FRG=Trame_Recue.data[0];
                                                      // We may pass to the next lamp
                                                      Ident_T_IRM=Ident_T_IRM_FVG;} //Left Front lamp
                                 eak;

                          s whether it's the time to read the data of lights stalk
                     if(I_    _Tempo_Commodo)
                              {// Pass to the "Reading Lights stalk" case
                               I_Fin_Tempo_Commodo=0;
                               Etat = Etat_Lect_Commodo_Feux;
                               Ident_T_IRM=Ident_T_IRM_Commodo_Feux;
                               // Send the first frame through the CAN network
                              Ecrire_Trame(T_IRM);
                              Trame_Envoyee = T_IRM;
                              I_Attente_Reponse=1;}
                    // Test whether The indicator activates AND the end of the "indicator" delay
                else if((I_Clignot_Gauche||I_Clignot_Droit)&&I_Fin_Tempo_Clignot)
                              {I_Fin_Tempo_Clignot=0;
                               if(I_Clignot_Gauche)
                                        {// Switch the left indicator on
                                         Valeur_FVG^=Masque_Clign_AV;
                                         Valeur_FRG^=Masque_Clign_AR;
                                         Passer_a_Etat_Modif_Feux();}
                               if(I_Clignot_Droit)
                                        {// Switch the right indicator on
                                         Valeur_FVD^=Masque_Clign_AV;
                                         Valeur_FRD^=Masque_Clign_AR;
                                         Passer_a_Etat_Modif_Feux();}
                               }// End of the indicator function
                              // Continue the Lamp control function
                    else      {if(Rang_Control_Stat==3)Rang_Control_Stat=0;
                               else Rang_Control_Stat++;// Pass to the next lamp
                              // We send the remote frame through the bus
                               Ecrire_Trame(T_IRM);
                               Trame_Envoyee =T_IRM;
                               I_Attente_Reponse=1;}
                    } // End of "Control Status" case
          } // End if the frame is received
          else if (I_Fin_Tempo_Att_Rep) // we wait for a long time until receiving the reply !
                    {clsscr();
                    gotoxy(1,2); // Display the title
printf("    EXPERIMENT N°4: COMMAND THE LIGHTS BY LIGHTS STALK    \n");
printf("    ********************************************************************** \n");
                     I_Fin_Tempo_Att_Rep=0; // We repeat the reply waiting delay
                     Valeur_Fin_Tempo_Att_Rep = Compteur_dS + Tempo_Att_Rep;
                     gotoxy(1,4),printf(" !! Wait for the model response  !!  \n");
                     gotoxy(1,9),printf("                                      \n");
```

```
                              gotoxy(1,20),printf("                              \n");
                              gotoxy(1,16),printf("                              \n");
                              I_En_Att_Rep=1;
                              // We retry the lights stalk enquire
                              Etat = Etat_Lect_Commodo_Feux;
                              Ident_T_IRM=Ident_T_IRM_Commodo_Feux;
                              // We send the first frame through the bus
                              Ecrire_Trame(T_IRM);
                              Trame_Envoyee = T_IRM;
                             I_Attente_Reponse=1;
                             }

                              //We retry the lights stalk enquire
          if(I_Fin_Tempo_Affichage)
                    {I_Fin_Tempo_Affichage=0;
                             if(I_En_Att_Rep==0) // Then we can display
                                       {                   // Left Front Lamp diagnostic result
                                                           gotoxy(1,4),printf("Left Front Block:     \n");
                                                           if(Veilleuse_FVG==1 && S_Veilleuse_FVG==0)
                                                           {gotoxy(1,5),printf("!! Problem on left front Side light   \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("!! Problem on left front Dipped light        \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("!! Problem on left front Head light        \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("!! Problem on left front Indicator   \n");}
                                                           if(Clignot_FVG==0 && S_Clignot_FVG==1)
                                                           {gotoxy(1,8),printf("                                         \n");}
                                                           // Front Right Lamp diagnostic result
                                                           gotoxy(1,9),printf("Right Front Block:\n");
                                                           if(Veilleuse_FVD==1 && S_Veilleuse_FVD==0)
                                                           {gotoxy(1,10),printf("!! Problem on right front side light \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("!! Problem on right front dipped light     \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("!! Problem on right front head light     \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("!! Problem on right front indicator \n");}
                                                           if(Clignot_FVD==0 && S_Clignot_FVD==1)
                                                           {gotoxy(1,13),printf("                                        \n");}
                                                           // Back Right Lamp diagnostic result
                                                           gotoxy(1,20),printf("Right Back Block:\n");
                                                           if(Veilleuse_FRD==1 && S_Veilleuse_FRD==0)
                                                           {gotoxy(1,21),printf("!! Problem on right back light     \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("!! Problem on right back indicator \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("!! Problem on right back stop light       \n");}
                                                           if(Stop_FRD==0 && S_Stop_FRD==1)
                                                           {gotoxy(1,23),printf("                                         \n");}
                                                           // Left Back Lamp diagnostic result
                                                           gotoxy(1,16),printf("Left Back Block:\n");
                                                           if(Veilleuse_FRG==1 && S_Veilleuse_FRG==0)
                                                           {gotoxy(1,17),printf("!! Problem on left back light     \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("!! Problem on left back indicator \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("!! Problem on left back stop light        \n");}
                                                           if(Stop_FRG==0 && S_Stop_FRG==1)
                                                           {gotoxy(1,19),printf("                                          \n");}
                                                           if(Klaxon_FRG==1 && S_Klaxon_FRG==0)
                                                           {gotoxy(1,19),printf("!! Problem on left back horn      \n");}
                                                           if(Klaxon_FRG==0 && S_Klaxon_FRG==1)
                                                           {gotoxy(1,19),printf("                                          \n");}
                                                           // For the lights stalk state
                             gotoxy(4,24);
                             printf("Condition of the different entries imposed by the lights stalk:\n");
printf(" Side light=%d , Dipped light=%d , Head light=%d , Left indicator=%d  \n",Cde_Veilleuse,Cde_Code,Cde_Phare,Cde_Clign_Gauche);
printf(" Horn=%d ,    Stop light=%d ,     Right indicator= %d\n",Cde_Klaxon,Cde_Stop,Cde_Clign_Droit);
                                       } // End if not wait for the response after alert message
                             } // End if it's the end of display delay
          }// End of the main loop
}// End of the main function
}

// Function "Pass to the CONTROL STATUS case"
void Passer_a_Etat_Control_Stat(void)
{Etat=Etat_Control_Stat; // We return to the "Control status" case
                    // To prepare the remote frame to the following module in the list
                      switch(Rang_Control_Stat) // Following module in the list
                             {case 0 : Ident_T_IRM=Ident_T_IRM_FVG; //Left Front Lamp
                              break;
                              case 1 : Ident_T_IRM=Ident_T_IRM_FVD; //Right Front Lamp
                              break;
                              case 2 : Ident_T_IRM=Ident_T_IRM_FRD; //Right Back Lamp
                              break;
                              case 3 : Ident_T_IRM=Ident_T_IRM_FRG; //Left Back Lamp
```

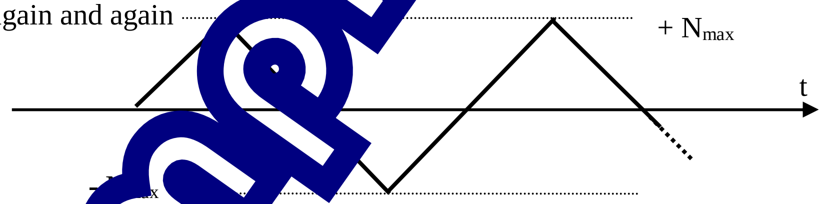
```
                              break;
                              default : {Rang_Control_Stat=0;
                                                 Ident_T_IRM=Ident_T_IRM_FVG;} //Left Front Lamp
                              break;
                             }
          // We send the remote frame through the bus
          Ecrire_Trame(T_IRM);
          Trame_Envoyee =T_IRM;
          I_Attente_Reponse=1;
} // End of Function "Pass to the CONTROL STATUS case"

// Function "Go to the MODIFICATION LIGHT case"
void Passer_a_Etat_Modif_Feux(void)
{WORD I_TEMP;
 Etat=Etat_Modif_Feux; // Pass to the MODIFICATION LIGHT case
 // The frame is prepared to order first lights
 Ident_T_IM=Ident_T_IM_FVG; // Left Front Lamp: First in the list
 Valeur_T_IM=Valeur_FVG;
 Rang_Modif_Feux=1;
 // We send the command frame through the bus
 Ecrire_Trame(T_IM);
 I_Attente_Reponse=1;
} //  End of the Function "Pass to the MODIFICATION LIGHT case"

 // End of the file source C
```

5 EXPERIMENT N°5 : COMMAND THE WINDSHIELD WIPER MOTOR

5.1 Topic

Purpose :	- Define the different control frames to pass through the CAN network based on an expected action. - Order an electric actuator (DC motor), in both directions of rotation, by a controllable pre-actuator through CAN network. - Make the electric motor speed varied by a controllable pre-actuator (interface PWM) through CAN network.
Specifications :	Carry out the periodic cycle operating as following: - The speed increases gradually from zero to the maximum speed in the positive direction, - Then it decreases gradually from the maximum speed to zero, - The speed increases gradually from zero to the maximum speed in the negative direction, - Then it decreases gradually from the maximum speed to zero, - again and again 

Necessary hardware and software :

PC Micro Computer using Windows® 9[illegible]
Software: Editor -Assembler-Debugger
If programming in C, GNU, C / C ++ compiler Ref: EID210101
Processor board 16/32 bit 68332 microcontroller and its software environment
(Editor-Cross Assembler-Debugger) Ref: EID210001
CAN PC/104 Network board in ATON SYSTEMS Ref NIC: EID004001
- 1 electronic module "servo-system motor" Ref : EID052001
- The operative block of the windshield wiper system Ref: EID053001
USB connection cable, or if not available use RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules ("energy" network)

Time : 4 hours

5.2 **Solution**

5.2.1 Analysis

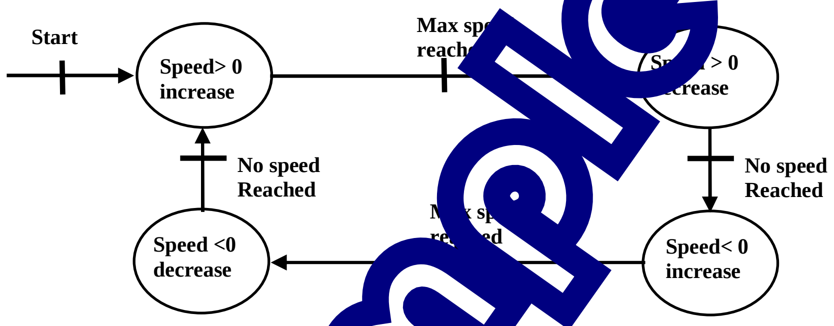
Principle:
The CAN interface module used in this experiment is a "Servo-system" module.
This module allows the control of a DC motor 24V / 1A, in both directions of rotation, in "PWM" mode (pulse width modulation).
A power circuit (Ref: L6202), piloted by logic signals is located on the module.
According to the electronic diagram of the "Servo-system" module, this guide for "Servo-system" is done by GP2, GP3 and GP4 output of CAN interface (MCP25050 circuit):

→ GP2 or "PWM1" logic output with cyclic duty variables, which allows to vary the motor speed in the positive direction,

→ GP3 or "PWM3" logic output with cyclic duty variables, which allows to vary the motor speed in the negative direction,

→ GP4 or "ValidIP" logic output, when it is set to 1, which validates the "L6202" Power Interface circuit.

According to the motor control factor, the cycle thus consists of 4 states:



A diagram with 4 necessary states, for its encoding, 2 binary variables:

- indicator "I_Sens _Variation"
  - I_Sens_Variation = 0 the speed is increasing in the module
  - I_Sens_Variation = 1 the speed is decreasing in the module
- indicator "I_Sens _Rotation"
  - I_Sens_Rotation = 0 the speed is positive (driven by PWM1)
  - I_Sens_Rotation = 1 the speed is negative (driven by PWM2).

Definitions of frames for controlling the motor:
Definition of the identification details of a frame type "IM" for controlling the "Servo-system" module:
In this case, the frame sent by the CAN controller (SJA1000 circuit CAN_PC104 board) will be seen by the receiver (MCP25050 system module) as an IM "Input message" with the "Write register" function (see in technical documentation MPC25025 pages 22). We can also modify the different registers of the "Servo-system" module.
The identifier defined in Chapter 1, for an "IM" sent to "Servo-system" board is:

0x00880000

→ Definition of structured variables under the "**Trame**" model :
```
Trame T_IM_Asservissement;
```
→ Defining identification details of the "**T_IM_Asservissement**" structured variable
```
T_IM_Asservissement.trame_info.register=0x00;  // All bits are initialized to 0
T_IM_Asservissement.trame_info.champ.extend=1; // Work in extended mode
T_IM_Asservissement.trame_info.champ.dlc=0x03; // There is 3 data of 8 bits (3 bytes)
T_IM_Asservissement.ident.extend.identificateur.ident=0x00880000;
```
In each control frames "IM", there will have to define the three associated bytes.

Definition of associated data of three bytes for:

→ *define the input and output*

It should initialize the GPDDR register ("Data Direction Register") by writing 1 into the input bit and 0 into the output bit`(doc MCP25050 p27)`. According to the board schema:

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

```
T_IM_Asservissement.data[0]=0x1F; // GPDDR register address
                                                       (doc MCP25050 p16)
T_IM_Asservissement.data[1]=0x7F; // Mask: 7 Bits not affected (doc MCP25050 p16)
T_IM_Asservissement.data[2]=0xE3; // Value: load into the addressed register
```

→ *initialize the output GP2 by PWM1 output (variation of motor speed in the positive direction)*

According to the technical manual MCP25050 circuit (pages 30-32), the generation of the PWM1 signal is block of the "Timer 1" and the frequency of this signal is selected through the "T1CON" register from 05H address (page 15 Doc MCP25050).

bit 7 =1	TMR1ON	Validation of "Timer 1"
bits 5:4	will set to 0 to have a frequency division factor equal to 1 ("TMR1 prescaler value" = 1)	

```
T_IM_Asservissement.data[0]=0x21; // T1CON Register address
 (doc MCP25050 p15) 05H + shift = 05H + 1CH = 21H
T_IM_Asservissement.data[1]=0xB3; // Mask Register (doc MCP25050 p32)
T_IM_Asservissement.data[2]=0x80; // Value loaded into the addressed register
```

→ *Set the frequency of the t PWM1 output:*

This frequency depends on the value loaded into the "PR1" register.

```
T_IM_Asservissement.data[0]=0x23; // PR1Register address
 (doc MCP25050 p15) 07H + shift = 07H + 1CH = 23H
T_IM_Asservissement.data[1]=0xFF; // Mask Register (doc MCP25050 p32)
T_IM_Asservissement.data[2]=0xFF; // 255 loaded into the register
```

The quartz frequency located on the "Servo-system" board is equal to 16Mhz, the frequency of PWM1 signal will be equal to : $F_{PWM} = 16.10^6/(4.256) = 15{,}6$ KHz

This is the correct frequency to drive a motor by PWM (substantially inaudible frequency).

→ *initialize the output G32 by PWM2 output (variation of motor speed in the negative direction)*

According to the technical manual MCP25050 circuit (pages 30-32), the generation of the PWM2 signal is block of the "Timer 2" and the frequency of this signal is selected through the register "T2CON" from address 06H (page 15 Doc MCP25050).

bit 7 =1	TMR2ON	Validation of "Timer 2"
bits 5:4	will [illegible] to 0 for [illegible] frequency division factor equal to 1 ("TMR2 prescaler value" = 1)	

```
T_IM_Asservissement.data[0]=0x22; // Register T2CON address
 (doc MCP25050 p15) 06H + shift = 06H + 1CH = 22H
T_IM_Asservissement.data[1]=0x[illegible]; // Mask register (doc MCP25050 p32)
T_IM_Asservissement.data[2]=0x[illegible]0; // Value loaded into the addressed register
```

→ *Set the frequency of the t PWM2 output:*

Cette fréquence dépend de la value chargée dans le register "PR2"

```
T_IM_Asservissement.data[0]=0x24; // Register PR2 address
 (doc MCP25050 p15) 08H + shift = 08H + 1CH = 24H
T_IM_Asservissement.data[1]=0xFF; // Mask register (doc MCP25050 p32)
T_IM_Asservissement.data[2]=0xFF; // 255 loaded into the register
```

The quartz frequency located on the "Servo-system" board is equal to 16Mhz, the frequency of PWM1 signal will be equal to : $F_{PWM} = 16.10^6/(4.256) = 15{,}6$ KHz (same as PWM1)

→ *change the cyclic duty of the PWM1 output (motor control in the positive direction)*

```
T_IM_Asservissement.data[0]=0x25; // PWM1DCH Register address
 (doc MCP25050 p15) 09H + shift = 09H + 1CH = 25H
T_IM_Asservissement.data[1]=0xFF; // Mask register (doc MCP25050 p33)
T_IM_Asservissement.data[2]=0x00; //All bits are initialized to 0 -> No Command
therefore no speed (until 255 for maximum control therefore Maxi speed)
```

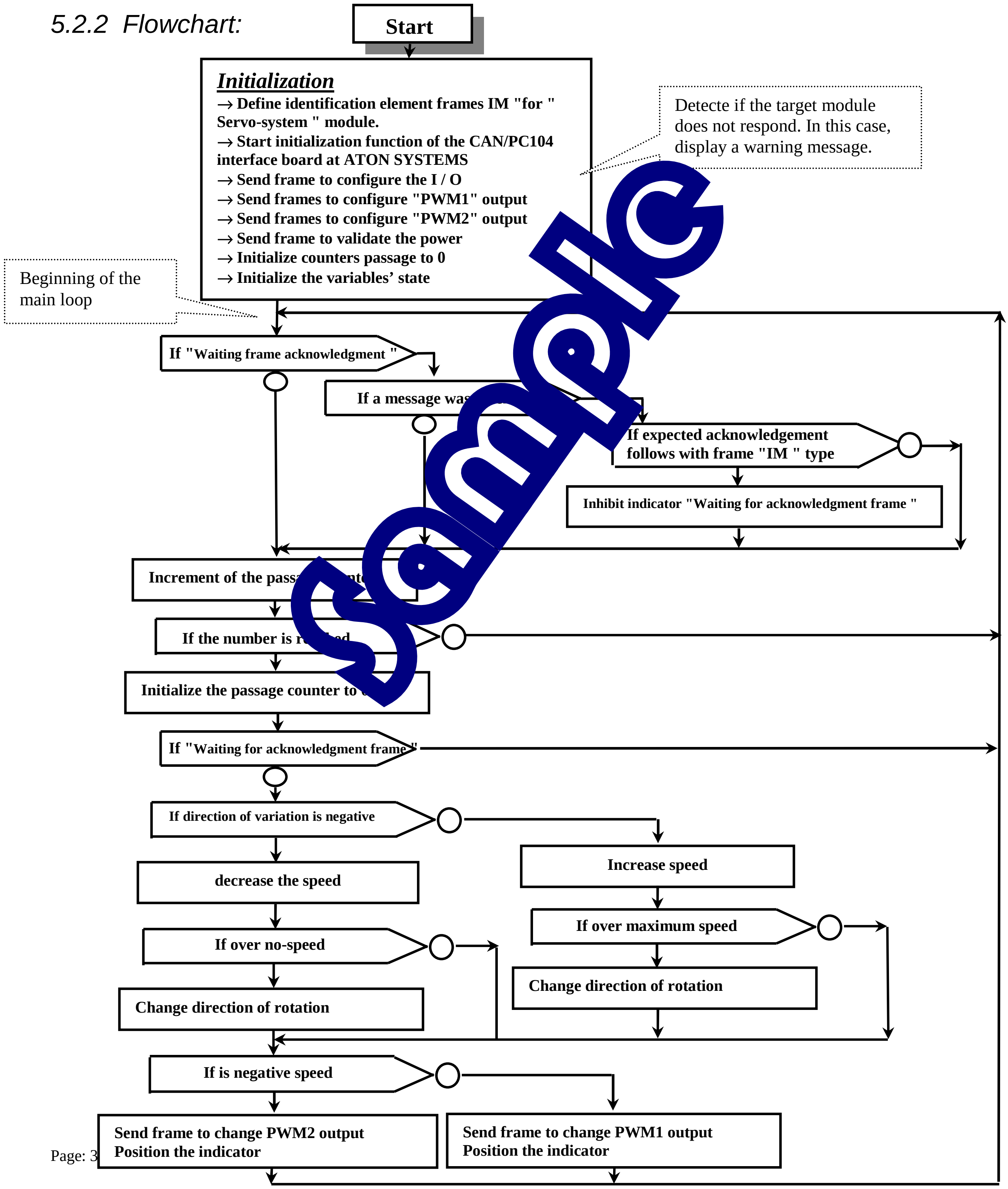
→ *change the cyclic duty of the PWM2 output (motor control in the negative direction)*

```
  T_IM_Asservissement.data[0]=0x26; // PWM2DCH Register address
   (doc MCP25050 p15) 09H + shift = 09H + 1CH = 25H
  T_IM_Asservissement.data[1]=0xFF; // Mask register (doc MCP25050 p33)
  T_IM_Asservissement.data[2]=0x00; // All bits are initialized to 0 -> No Command
therefore no speed (until 255 for maximum control therefore Maxi speed)
```

→ *to validate the power circuit*

```
  T_IM_Asservissement.data[0]=0x1E; // GPLAT Register address
  T_IM_Asservissement.data[1]=0x10; // Mask register (doc MCP25050 p27)
  T_IM_Asservissement.data[2]=0x10; // Set 4th register bit  to 1
```

5.2.2 Flowchart:



5.2.3 "C" Program

```
/***************************************************************************************
*       Experiment on  EID210 / CAN Network – V.M.D  (Multiplexed Didactic Vehicle)               *
****************************************************************************************
*    EXPERIMENT N°5:  COMMAND THE WINDSHIELD WIPER MOTOR
*-------------------------------------------------------
*   SPECIFICATIONS :
*  *********************
*  Carry out the periodic cycle which operates as following:
*         - right rotation with increasing speed from 0 to Vmax
*         - right rotation with decreasing speed FROM Vmax TO 0
*         - left rotation with increasing speed from 0 to Vmax
*         - left rotation with decreasing speed FROM Vmax TO 0
*         again and again ...
*  Display on the screen, in which there is the block of the cycle with speed level
*------------------------------------------------------------------------------------------
*  File Name:  CAN_VMD_TP5.C
* *****************
*******************************************************************************************/

// Declaration of included files
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"

// Declaration of variables
int Cptr_TimeOut;
// For various indicators (binary variables)
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
// Declaration of various communication frames through CAN Bus
Trame Trame_Recue;  // For the frame which has been received by CAN c[illegible]ler
// The frames of type "IM" (Command frame)
Trame T_IM_Asservissement;   // For the frame of type "IM" to the "[illegible]-sys[illegible]
                                                 //  IM -> Input Messag[illegible] Com[illegible]ame

//===========================
// DEFINITION OF FUNCTIONS
//===========================

// MAIN FUNCTION
//============================
main()
{
//  INITIALIZATIONS
//-----------------

int Cptr_Incrementation_Vitesse=0,Cptr_Affichage=0;
BYTE Module_Vitesse=0;
/* Initialization of SJA1000 of the ATON-Systems boa[illegible]on [illegible]ci[illegible]it */
Init_Aton_CAN();
clsscr();
I_Message_Pb_Affiche=0;
//The frames of type "IM" (Command frame): Id[illegible]ificati[illegible]
T_IM_Asservissement.trame_info.registre=0x00[illegible]
T_IM_Asservissement.trame_info.champ_[illegible]xtend=[illegible]
T_IM_Asservissement.trame_info.cham[illegible]0x0[illegible]
T_IM_Asservissement.trame_info.cha[illegible]rt[illegible]
T_IM_Asservissement.ident.extend.i[illegible]nti[illegible]ate[illegible]ident=[illegible]t_T_IM_Asservissement;
// To define Input / Output
T_IM_Asservissement.data[0]=0x1F;      // GPDD[illegible]egister address (I/O direction)
T_IM_Asservissement.data[1]=0xEF;      // Mask [illegible]7 bits not affected (doc MCP25050 p16)
T_IM_Asservissement.data[2]=0xE3;      [illegible]ue [illegible] 1 for input and 0 for output
          // GP7 fs = input; GP6 = fcg in[illegible] GP[illegible]cd = Input; GP4 = ValidIP Output;
          // GP3 = PWM2 output; GP2 = PWM1[illegible]p[illegible] GP1 = AN1 input; GP0 = AN0 Input;
do {Ecrire_Trame(T_IM_Asservissement);
           Cptr_TimeOut=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(" No response to the command frame in initialization  \n");
                               printf("  Check whether the power supply 12V is OK \n");}}
    }while(Cptr_TimeOut==100);
// To set outputs to 0
T_IM_Asservissement.data[0]=0x1E;      // GPLAT Register address (I/O register)
T_IM_Asservissement.data[1]=0x1C;      // Mask -> GP4,3,2 outputs are affected
T_IM_Asservissement.data[2]=0x00;      // Value ->  the 3 output set to 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP2output by PWM1
T_IM_Asservissement.data[0]=0x21;      // T1CON Register address
T_IM_Asservissement.data[1]=0xB3;      // Mask -> only bit 7,5,4,1,0 affected
T_IM_Asservissement.data[2]=0x80;      // Value ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM1 output signal frequency
T_IM_Asservissement.data[0]=0x23;      // PR1Register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0xFF;      // Value ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP3 output by PWM2
T_IM_Asservissement.data[0]=0x22;      // T2CON Register address
T_IM_Asservissement.data[1]=0xB3;      // Mask -> only bit 7,5,4,1,0 affected
T_IM_Asservissement.data[2]=0x80;      // Value ->  TMR2ON=1; Prescaler2=1
Ecrire_Trame(T_IM_Asservissement);
```

```
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM2 output signal frequency
T_IM_Asservissement.data[0]=0x24;       // PR2Register address
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0xFF;       // Value ->  PR2=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM1 cyclic duty to 0
T_IM_Asservissement.data[0]=0x25;       // PWM1DC Register address
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0;          // Value ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM2 cyclic duty to 0
T_IM_Asservissement.data[0]=0x26;       // PWM2DC Register address
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0;          // Value ->  PWM2DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To validate the power circuit
T_IM_Asservissement.data[0]=0x1E;       // GPLAT Register address (I/O register)
T_IM_Asservissement.data[1]=0x10;       // Mask -> GP4 (ValidIP) output is affected
T_IM_Asservissement.data[2]=0x10;       // Value ->  ValidIP=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// Mask for future orders of IM
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bits are affected

// Initialization of the application variables
Module_Vitesse=0;
Indicateurs.valeur=0; // Positive direction, speed increases
I_Attente_Trame_Acquittement=0;
Cptr_TimeOut=0;
clsscr();
// To display the title
gotoxy(1,2);
printf("    EXPERIMENT  N°5:   COMMAND THE WINDSHIELD WIPER BLADE
printf("    ********************************************** \n");


// MAIN LOOP
//*******************
while(1)
          {if(I_Attente_Trame_Acquittement)
                    {Cptr_TimeOut++;
                     if(Lire_Trame(&Trame_Recue)==1)
                      {if(Trame_Recue.ident.extend.identificateur  ent=        _AIM_Asservissement)
                              {// This is the "Ack IM" from "      tem"      e
                                I_Attente_Trame_Acquitteme     ;}}
                     else {if(Cptr_TimeOut==65000)
                                {clsscr();
                                 gotoxy(1,10);
                                 printf(" No response    the com        me during the cycle\n");
                                 printf(" Reload the    ogr    d re    t it \n");
                                 do{}while(1);}}
                    }
          //Change the state Block
          Cptr_Incrementation_Vitesse++;
          if (Cptr_Incrementation_Vitesse==20
                    {// It is time to change  e m    spe
                    Cptr_Incrementati   Vite  e=0
                    if(I_Sens_Varia     //    yes      the speed
                              {Mod    Vi  se
                               if(  dule          255)    le_Vitesse=0,I_Sens_Variation=0,I_Sens_Rotation=!I_Sens_Rotation;}
                    else {Module_V   sse++          increase the speed
                              if(Mo    Vitesse=     Module_Vitesse=255,I_Sens_Variation=1;}
                    if(I_Sens_Rotatio     if yes  turn to the negative direction
                              {if(I_Attente  rame  cquittement==0)
                                        {   Ass  vissement.data[0]=0x25;      //PWM1DC register address
                                         T_    rvissement.data[2]=Module_Vitesse;
                                                  // Principle : 0--> 0 rd/min et 255 ->5000 rd/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;      // PWM2DC register address
                                         T_IM_Asservissement.data[2]=Module_Vitesse;
                                         Ecrire_Trame(T_IM_Asservissement);
                                         I_Attente_Trame_Acquittement=1,Cptr_TimeOut=0;}}
                    } // End of the modification of motor speed
          //  Display the state Block
          Cptr_Affichage++;
          if(Cptr_Affichage==20000)
                    {Cptr_Affichage=0;
                    gotoxy(1,10);
                    if(I_Sens_Rotation==0)
                              {printf("Frame for controlling the motor in negative ............value \n");
                               Affiche_Trame(T_IM_Asservissement);}
                    else  {printf("Frame for controlling the motor in positive                 value\n");
                               Affiche_Trame(T_IM_Asservissement);}}
          } //  End of the main loop
} // End of the main function
```

6 EXPERIMENT N°6 : THE WINDSHIELD WIPER BLADE DIRECTION CONTROL

6.1 Topic

Purpose :	- Define the different control frames to pass through the CAN network based on a expected action. - Order an electric actuator (DC motor), in both directions of rotation, by a controllable pre-actuator through CAN network. - Acquire the state of ON/OFF sensors (at the limit switch) accessible through the CAN network and deduce the actions to satisfy imposed the specifications.
Specifications :	After having rotated the motor in the positive direction (rotate on right) until reaching the right limit switch. Then the wiper realizes the con[illegible] as following: → rotating to left side until reachingthe left limit switch → rotating to right side until reaching the right limit switch → again and again

Necessary hardware and software :

PC Micro Computer using Windows ® 95 or later
Software: Editor -Assembler-Debugg[illegible]
If programming in C, GNU; C / C ++ Compiler Ref: EID210101
Processor board 16/32 bit 68[illegible] and its software environment
(Editor-Cross Assembler-Debugger Ref: E[illegible]210001
CAN PC/104 Network board in [illegible]STEMS Ref NIC: EID004001
- 1 electronic "servo-system" module Ref : EID052001
- The operative block of the windshield wiper system Ref: EID053001
USB connection cable, or if not available use RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules ("energy" network)

Time : 4 hours

6.2 **Solution**

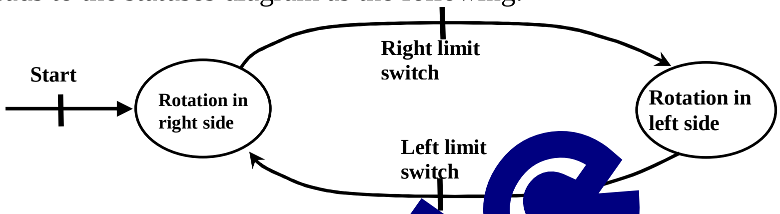
6.2.1 Analysis

Principle :

The CAN interface module used in this experiment is a "Servo-system" module.
This module allows the control of a DC motor 24V / 1A, in both directions of rotation, in "PWM" mode (pulse width modulation). This has been introduced in EX 5.
It allows acquiring 3 binary(on/off) inputs on which we can connect sensors in the limit switch:

→ fcd (right limit switch) connected to the GP5 input of the CAN controller
→ fcg (left limit switch) connected to the GP6 input of the CAN controller
→ fs (over limit switch) connected to the GP7 input of the CAN controller

The requested cycle leads to the statuses diagram as the following:



Configuration and control frames (type "IM") → [illegible]
Acquire the status of the limit switch:

Definition of remote frame to know the status of the limit[illegible]
In this case, the frame sent by the CAN controller (SJ[illegible]000 c[illegible] CAN_PC104 board) will be seen by the receiver (MCP25050 system module) as an IRM "In[illegible] R[illegible]quest Message" with the "Read register" function (see in technical documentation MPC25025 [illegible]s 22).

From the given table on page 22 of the manual [illegible]CP2[illegible]rc[illegible] the identifier itself will contain the address of the read register. This address is located o[illegible]e[illegible] to [illegible]D8 [illegible]entifier bits in extended mode (bits that are received and located in the RXBEID8 regi[illegible]r). The c[illegible]erned register is GPPIN of address 1Eh "(see in technical documentation MPC25025 pages [illegible]7).
On the other hand, the leat significant 2[illegible] the i[illegible]ntifier in extended mode must be set to 1.
The identifier defined in Chapter 1 s[illegible]uld be [illegible]eted as follows:

00 84 xx xx 0[illegible]84[illegible]7 (Only 29 bits are taken into account)

According to Table	[illegible]egister a[illegible]ess	Function code

→ Definition of structured varia[illegible]ur[illegible]er the "**Trame**" model:

```
Trame T_IRM_Acquisition_FC;    // Frame appointed for enquiry of the servo-system module to acquire the limit switch
```

Remark: The structured variable `T_IRM_Acquisition_FC` only contain 5 useful bytes, 1 byte for trame_info and 4 bytes for identifier in extended mode (which will include the concerned register address by reading).

→ Access and definition of the different elements of the " **Lecture_FC** " structured variable

```
T_IRM_Acquisition_FC.trame_info.register=0x00;  // All bits are initialized to 0
T_IRM_Acquisition_FC.trame_info.champ.extend=1; // Work in extended mode
T_IRM_Acquisition_FC.trame_info.champ.dlc=0x01; // There is 3 data of 8 bits (3 bytes)
T_IRM_Acquisition _FC.ident.extend.identificateur.ident=0x00841E07;
```

According to the definition of identifiers given in Chapter 1, a response frame with IRM has the same identifier as the remote frame which was original.
Seen from the module (the MCP25050), the response to an "IRM" (Information Request Message) is an "OM" (Output Message).
The difference with the original remote frame is that this response frame contains the "value" parameter (in rank 0 of the "data" of the response frame). This parameter is the image of input. Thus we can recover the status of different sensors.

Definition of structured variables images of the state of the limit switch

The received frame in response to this remote frame include in data [0], state of the limit switch. We copy the received data in a variable image.

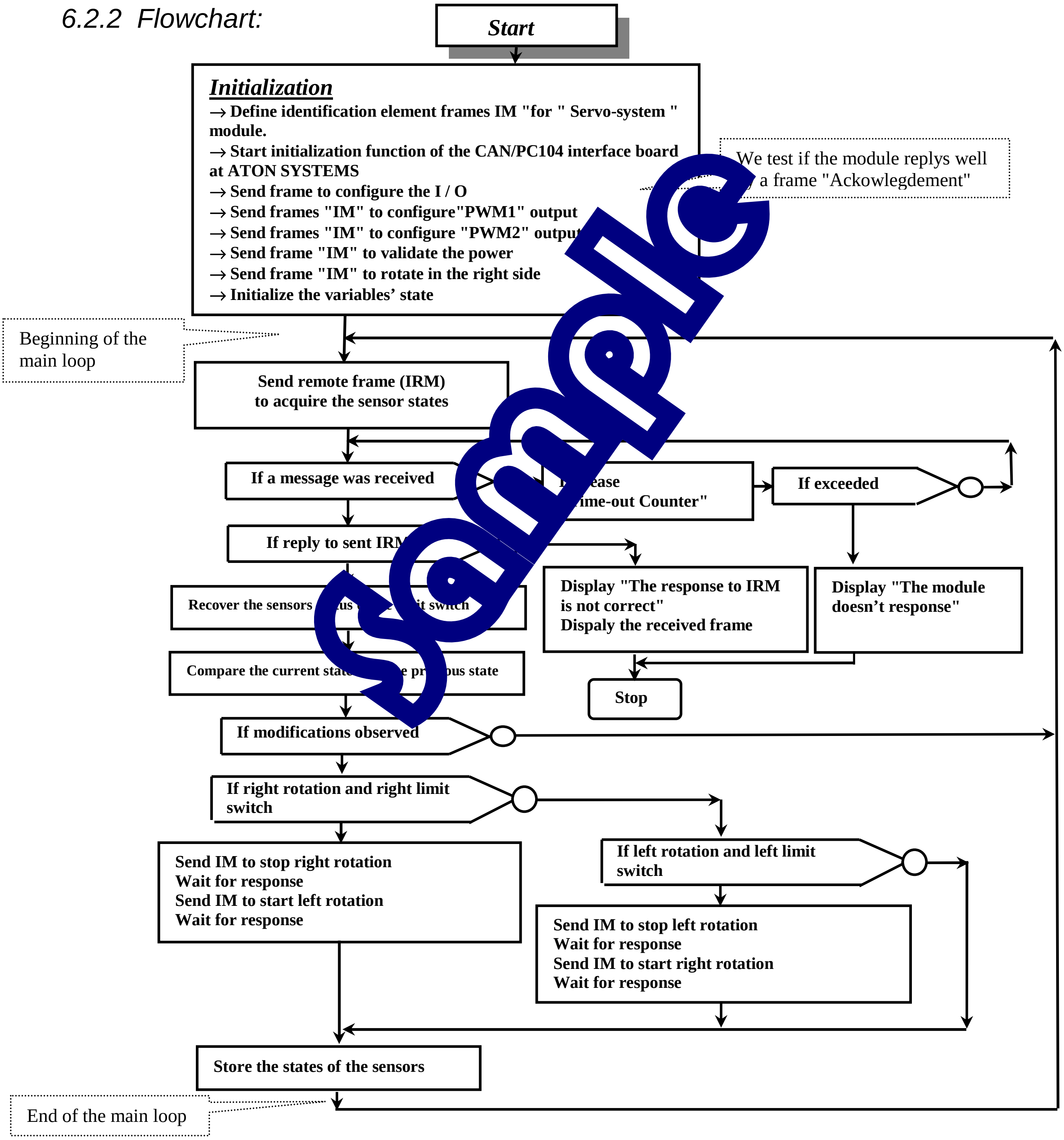
```
union byte_bits FC;
#define Etats_FC FC.value // For the group of state of the limit switch
#define fs FC.bit.b7  // For over limit switch
#define fcg FC.bit.b6 // For left limit switch
#define fcd FC.bit.b5 // For right limit switch
```

In order to detect the modification of the sensors’ states, it stores the states in a second structured variable -> State_stored

```
union byte_bits FC_Mem; // For end of the stored process
#define Etats_FC_Mem FC_Mem.value  // For the group of stored state
```

If the status of an acquired variable is different from its stored value, there has been a state change. Therefore we have to do something.

6.2.2 Flowchart:



6.2.3 "C" Program

```
          /*************************************************************************************
          *         Experiment on  EID210 / CAN Network – V.M.D  (Multiplexed Didactic Vehicle)
          *************************************************************************************
          *   EXPERIMENT N°6:  THE WINDSHIELD WIPER BLADE DIRECTION CONTROL
          *-------------------------------------------------------
          *   SPECIFICATIONS :
          *  *********************
          *  We want the continuous cycle as following:
          *         - rotating to right side until reaching left limit switch
          *         - stop at the end of right rotation and start the left rotation
          *         - rotating to left side until reching right limit switch
          *         - stop at the end of right rotation and start the right rotation
          *         again and again ...
          *  The program structure will be "scanning loop" type.
          *  Display on the screen what state it is..
          *-----------------------------------------------------------------------------------------------
          *  File Name:  CAN_VMD_TP6.C
          * *****************
          ******************************************************************************************/
// Declaration of included files
#include <stdio.h>
#include"Structures_Donnees.h"
#include"cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"
// Declaration of variables
// For various indicators (binary variables)
union byte_bits Indicateurs,FC,FC_Mem; // Bits Structure
#define I_Sens_Rotation Indicateurs.bit.b0
#define I_Attente_Reponse_IRM Indicateurs.bit.b1
#define I_Message_Pb_Affiche Indicateurs.bit.b2
// For the limit switch
#define Etat_FC FC.valeur // For the group of state of the limit swit
#define fs FC.bit.b7          // For over limit switch
#define fcg FC.bit.b6         // For left limit switch
#define fcd FC.bit.b5         // For right limit switch
#define Etat_FC_Mem FC_Mem.valeur // For storage of the state of the limi   itch
// Declaration of various frames of communication
Trame Trame_Recue;  // For the frame that has just been received by   con
// Frames of type "IM" (Input Message -> Command frame)
Trame T_IM_Asservissement;    // For the motor control
Trame T_IRM_Acquisition_FC;   // For acquisition of the state of   e l    wi
// Declaration of the constants
#define Module_Vitesse 50
//======================
// MAIN FUNCTION
//======================
main()
{//  INITIALIZATION
//------------------
// Declaration of local variables in the main functi
int Cptr_Incrementation_Vitesse=0,Cptr_Affichage=0,Cp   Time

Init_Aton_CAN();
clsscr();
// The frames of type "IM" (Command frame );  entificati     a
T_IM_Asservissement.trame_info.registre=0x0
T_IM_Asservissement.trame_info.cham    end=
T_IM_Asservissement.trame_info.cha    d
T_IM_Asservissement.trame_info.cha   .rt
T_IM_Asservissement.ident.extend.i  ntif             ent_T_IM_Asservissement;
// To set the Input/Output
T_IM_Asservissement.data[0]=0x1F;       // GPDDR   gister address (I/O direction) doc MCP25050 Page 16
T_IM_Asservissement.data[1]=0xEF;          sk -   Bit 7 not affected
T_IM_Asservissement.data[2]=0xE3;       //   lue     1 for input and 0 for output
          // GP7=fs Input; GP6=fcg Input;     f   Input;  GP4=ValidIP Output;
          // GP3=PWM2 Output; GP2=PWM1 Outpu    1=AN1 Input; GP0=AN0 Input;
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Asservissement); // It’s the first sent frame
           Cptr_TimeOut=0;                                 // We test if the module responds well
           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(" No response to the command frame in initialization \n");
                               printf("  Check whether the power supply 12V is OK \n");}}
    }while(Cptr_TimeOut==100);
// To set outputs to 0
T_IM_Asservissement.data[0]=0x1E;       // GPLAT Register address (I/O Register)
T_IM_Asservissement.data[1]=0x1C;       // Mask -> GP4,3,2 outputs are affected
T_IM_Asservissement.data[2]=0x00;       // Value ->  the 3 output set to 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP2 output by PWM1
T_IM_Asservissement.data[0]=0x21;       // T1CON Register address
T_IM_Asservissement.data[1]=0xB3;       // Mask -> only bit 7,5,4,1,0 affected
T_IM_Asservissement.data[2]=0x80;       // Value ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM1 output signal frequency
T_IM_Asservissement.data[0]=0x23;       // PR1 Register address
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0xFF;       // Value ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP3 output by PWM2
T_IM_Asservissement.data[0]=0x22;       // T2CON Register address
T_IM_Asservissement.data[1]=0xB3;       // Mask -> only bit 7,5,4,1,0 affected
```

```
T_IM_Asservissement.data[2]=0x80;      // Value ->  TMR2ON=1; Prescaler2=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM2 output signal frequency
T_IM_Asservissement.data[0]=0x24;      // PR2 Register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0xFF;      // Value ->  PR2=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM1 cyclic duty to 0
T_IM_Asservissement.data[0]=0x25;      // PWM1DC Register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=Module_Vitesse;      // Value ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM2 cyclic duty to 0
T_IM_Asservissement.data[0]=0x26;      // PWM2DC Register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0;         // Value ->  PWM2DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To validate the power circuit
T_IM_Asservissement.data[0]=0x1E;      // GPLAT Register address (I/O Register)
T_IM_Asservissement.data[1]=0x10;      // Mask -> GP4 (ValidIP) output is affected
T_IM_Asservissement.data[2]=0x10;      // Value ->  ValidIP=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// Mask for future orders of IM
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
// To acquire the status of the limit switch
// Frame of type "IRM" (remote frame): Identification data
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;
 Ecrire_Trame(T_IRM_Acquisition_FC);   // Send a frame to acquire the status of the limit switch
do{}while(Lire_Trame(&Trame_Recue)==0);          //Wait for the response
         Etat_FC=~Trame_Recue.data[0];          // Recover the status of the limit switch
         Etat_FC_Mem=Etat_FC;                             // Save it
// Initialization of variables
         I_Sens_Rotation=0;

// To display the title
gotoxy(1,2);
printf("    EXPERIMENT  N°6:   THE WINDSHIELD WIPER BLADE DIRECTION [illegible]");
printf("    ********************************************************[illegible]\n");

// MAIN LOOP
//*******************
while(1)
         {
         // To "acquire the status of the limit switch "
         Ecrire_Trame(T_IRM_Acquisition_FC); // Send a frame to acquire the status of the limit switch
         Cptr_TimeOut=0;
         do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<10000));
          if(Cptr_TimeOut==10000)
                  {clsscr(),gotoxy(2,10);
                   printf(" No reponse for an enqui[illegible] the limit switch \n");
                   printf("  Reload the program and [illegible]tart[illegible]");
                   do{}while(1);} // Stop
          else { if(Trame_Recue.ident.extend.identificateur.ident==Ident_T_IRM1_Asservissement)
                           // If it's the expected ident
                            {Etat_FC=~Trame_Recue.data[0];           // Recover the status of the limit switch
                            if(Etat_FC!=Etat_FC_Mem)                 // If there was a modification
                                                                     // of the status of the limit switch -> we solve
                            {Etat_FC_Mem=Etat_FC;          // Storage of the new status
               // Deal with the state change
                  if(I_Sens_Rotation)           // If we turn to left side
                           {if(fcg)              // If we reach the left limit switch
                                     {T_IM_Asservissement.data[2]=0;         // We stop the left rotation
                                      T_IM_Asservissement.data[0]=0x26;      // PWM2DC Register address
                                      Ecrire_Trame(T_IM_Asservissement);
                                      while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                                      T_IM_Asservissement.data[2]=Module_Vitesse; // We control the right rotation
                                      T_IM_Asservissement.data[0]=0x25;      // PWM1DC Register address
                                      Ecrire_Trame(T_IM_Asservissement);
                                      while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                                      I_Sens_Rotation=!I_Sens_Rotation;               // Change the status
                                     }}
                  else      {// else we turn to right side
                            if(fcd)           // If we reach the right limit switch
                                     {T_IM_Asservissement.data[2]=0;         // We stop the right rotation
                                      T_IM_Asservissement.data[0]=0x25;      // PWM1DC Register address
                                      Ecrire_Trame(T_IM_Asservissement);
                                      while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                                      T_IM_Asservissement.data[2]=Module_Vitesse; // We control the left rotation
                                      T_IM_Asservissement.data[0]=0x26;      // PWM2DC Register address
                                      Ecrire_Trame(T_IM_Asservissement);
                                      while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                                      I_Sens_Rotation=!I_Sens_Rotation;               // Change the status
                                     }}
                  }}} // End of " Deal with the state change "
         // To display system status
         Cptr_Affichage++;
         if(Cptr_Affichage==200)
                  {Cptr_Affichage=0;
                  gotoxy(1,10);
                  if(I_Sens_Rotation==0)
                           {printf("Motor in Left Rotation  \n");
                           Affiche_Trame(T_IM_Asservissement);}
                  else  {printf("Motor in Right Rotation  \n");
                           Affiche_Trame(T_IM_Asservissement);}
                  } // End of " display system status "
         } //  End of the main loop
} // End of the main function
```

7 EXPERIMENT N°7 : THE WINDSHIELD WIPER BLADE SPEED CONTROL

7.1 Topic

Purpose :	- Acquire the result of conversion Analog -> Digital through a CAN network. - Carry out an imposed sampling period. - Experiment with different control modes (open-loop, closed-loop) a controllable analog system driven by CAN network. - Implement different types of digital controller (function : proportion, integral)
Specifications :	Step 1: Order the " windshield wiper " [illegible]ot[illegible] speed, in open-loop, with a peripheral to command the signal f[illegible] the [illegible]ntiometer on the "Servo-system" board Step 2: Order the " windshie[illegible] wiper [illegible] in closed-loop with corrective action of the proportion functi[illegible]. [illegible]d order is given by the analog potentiometer Step 3: Order the " w[illegible]otor in closed-loop with proportion + integral function NB: In each [illegible]se, [illegible] conversion results are displayed: - S[illegible] spee[illegible] AN0 conversion result -> Potentiometer) - [illegible]peed (Input conversion result AN1 -> F / U output).

N[illegible]cessary hardware and software :

PC Micro Computer using Windows [illegible]95 or later
Software: Editor -Assembler-Debugger
If programming in C, GNU; C / C + + Compiler Ref: EID210101
Processor board 16/32 bit 68332 microcontroller and its software environment (Editor-Cross Assembler-Debugger) Ref: EID210001
CAN PC/104 Network board in ATON SYSTEMS Ref NIC: EID004001
- 1 electronic "servo-system" module Ref : EID052001
- The operative block of the windshield wiper system Ref: EID053001

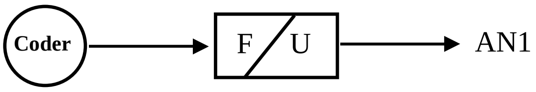
USB connection cable, or if not available use RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules ("energy" network)

Time : 3×3 hours

7.2 **Solution step n°1**

7.2.1 Analysis step n°1

Principle:
In step 1, the control is called "open-loop", that is to say the engine control depends only on the command peripheral (converted value of the voltage from the potentiometer applied on the input GP0 -> AN0 the MC25050 interface circuit). It does not depend on the speed measurement.
Remarks:
- The voltage from the potentiometer is in the range of 0/5V. This is a monopolar order; the motor will rotate in only one direction.
- A monopolar image of the motor rotation speed is available on the AN1 analog input. The available voltage about this input results from a frequency / voltage of the encoder signal 1 channel coupled to the motor conversion.



Technical data:
- The delivered coder impulsions by tr
- The transfer coefficient of the F / G converter is in V/Hz
- The reduction coefficient of Brush Speed / motor speed is

Configuration and control frames (type "IM") → the frame EX5
Definition of three bytes of data associated for:
→ enable and configure the Analog to Digital conversion
According to the technical manual MCP25050 circuit (pages 34 to 37):
Initialize the ADCON0 register,
```
T_IM_Asservissement.data[0]=0x2A; // ADCON0 register address
 (doc MCP25050 p15) 0EH + shift = 0EH + 1CH = 2AH
T_IM_Asservissement.data[1]=0xF0; // Mask -> only the bit 7 is affected
T_IM_Asservissement.data[2]=0x80; // Value: ADON = 1 -> Activation converter
                                  // "Prescaler rate" = 1:32
```
As the same for the ADCON1 register:
```
T_IM_Asservissement.data[0]=0x2B; // ADCON1 Register address
 (doc MCP25050 p15) 0FH + shift = 0FH + 1CH = 2BH
T_IM_Asservissement.data[1]=0xFF; // Mask -> 8 bits are affected
T_IM_Asservissement.data[2]=0x03; // Value: (doc MCP25050 p36)
      b7=ADCS1=0; b6=ADCS0=0 -> frequency Fosc/2
      b5=VCFG1=0; b4=VCFG0=0 → range of input voltage 0/+5V
      PCFG3:PCFG0=1100        → converting analog inputs 1 and 0 (GP1 and GP0 )
```

Acquire the results of A / D conversion

In fact, it is better to use IRM "Read A / D Regs", which allows to acquire once both the statements of logic inputs (limit switch) and the results of conversion of analog inputs (doc MCP25050 p22).
The identifier defined in Chapter 1 for IRM: 0x008400

→ Definition of structured variables under the "`Trame`" model :
`Trame T_IRM_Acquerir_FC_AN;` // Frame appointed for enquiry of the servo-system module to acquire the end of process as well as the A->D conversion result.
Remark: The **`T_IRM_Acquisition_FC`** structured variable only contain 5 useful bytes, 1 byte for trame_info and 4 bytes for identifier in extended mode

→ Access and definition of the different elements of the " **`T_IRM_Acquerir_FC_AN`** "structured variable
```
T_IRM_ Acquerir_FC_AN.trame_info.register=0x00;  // All bits are initialized to 0
T_IRM_ Acquerir_FC_AN .trame _info.champ.extend=1; // Work in extended mode
T_IRM_ Acquerir_FC_AN .trame _info.champ.dlc=0x08; // A requested data of 8 bytes
T_IRM_ Acquerir_FC_AN.ident.extend.identificateur.ident=0x00840000;
```
The response frame following the IRM includes associated 8 byte's data (`doc MCP25050 p22`):
- Rank 0 Byte (data[0])→ IOINTFL value → not useful in our case

- Rank 1 Byte (data[1])→ GPIO value → Value of logic Input / outputs
- Rank 2 Byte (data[2])→ AN0H value → 8 bits MSB conversion analog input 0
- Rank 3 Byte (data[3])→ AN1H value → 8 bits MSB conversion analog input 1
- Rank 4 Byte (data[4])→ AN10H value → 2 times 2 bits LSB conversion analog input. 1 and 0

The other 3 bytes are not useful in our application.
The result of conversion is 10 bits:

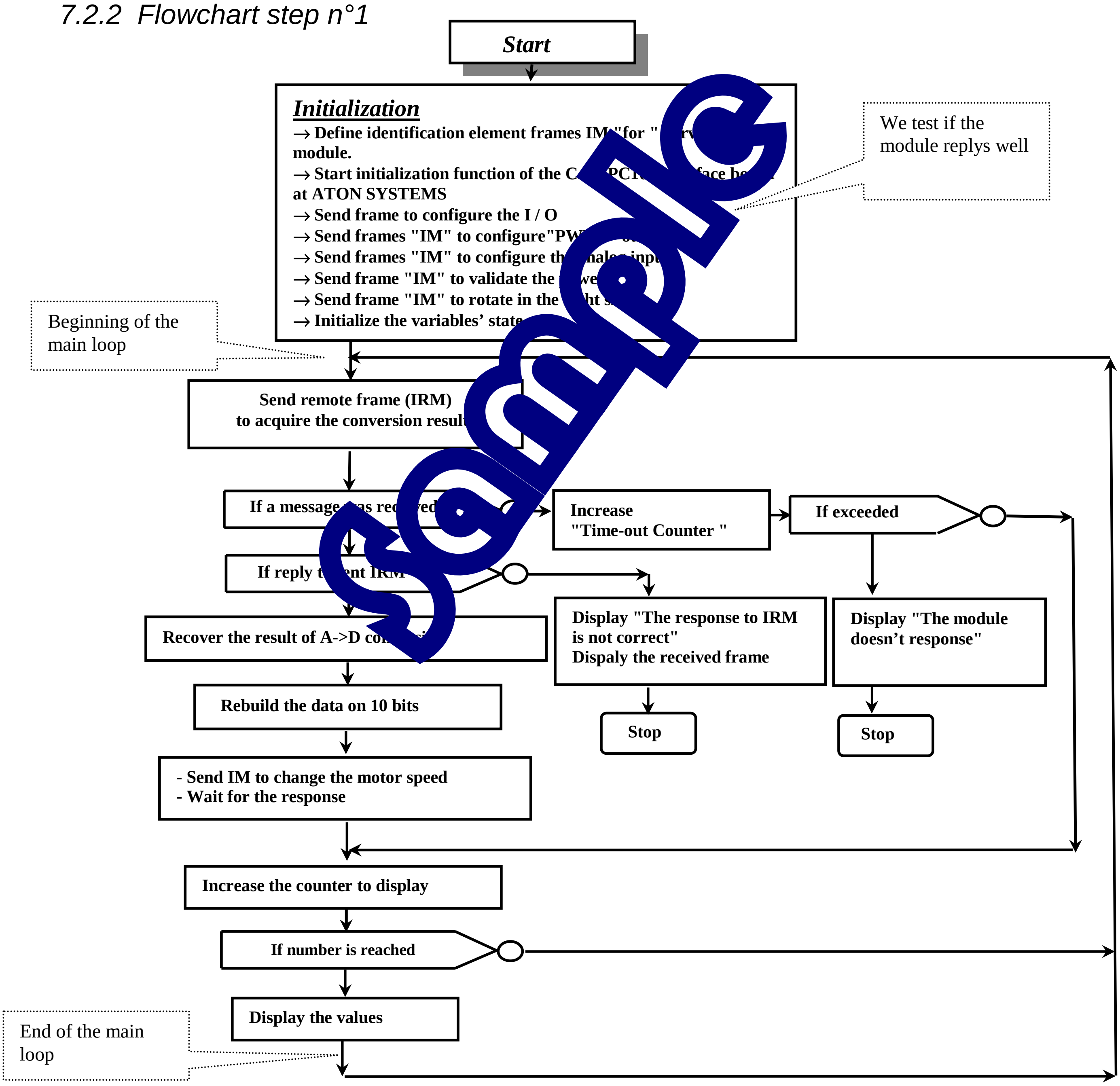
- for AN0 result (potentiometer)

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

- for AN1 result (sensor)

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

7.2.2 Flowchart step n°1



7.2.3 "C" Program step n°1

```
/************************************************************************************
*       Experiment on  EID210 / CAN Network – V.M.D  (Multiplexed Didactic Vehicle)
*************************************************************************************
*   EXPERIMENT N°7 :  THE WINDSHIELD WIPER BLADE SPEED CONTROL
*---------------------------------------------------------------------------------
*   SPECIFICATIONs :
*  ********************
*  Step 1: Vary the speed of the motor with the potentiometer implanted by the
*                    "Servo-system" module
*                    Display the conversion result of the potentiometer input as well as
*                    the input image speed
*-----------------------------------------------------------------------------------
*  File Name:  CAN_VMD_TP7_1.C
* *****************
************************************************************************************/

// Declaration of included files
#include <stdio.h>
#include"Structures_Donnees.h"
#include"cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"

// Declaration of various frames of communication
Trame Trame_Recue;  // For the frame that has just been received by the controller
// Frames of type "IM" (Input Message -> Command frame)
Trame T_IM_Asservissement;    // For the order of the motor
Trame T_IRM_Acquisition_FC_AN;         // For the acquisition of the analog input and the limit switch
// Declaration of variables
// For various indicators (binary variables)
union byte_bits Indicateurs; // Bit Structures
#define I_Sens_Rotation Indicateurs.bit.b0
#define I_Attente_Reponse_IRM Indicateurs.bit.b1
#define I_Message_Pb_Affiche Indicateurs.bit.b2
// For the conversion results
unsigned short S_Mesure_Vitesse,S_Val_Pot,S_Temp;
unsigned char AN0H,AN1H,AN10L;

//======================
// MAIN FUNCTION
//======================
main()
{
//  INITIALIZATION
//------------------
// Declaration of local variables in the main function
// Different counters
unsigned int Cptr_Affichage,Cptr_TimeOut,Ctpr_Acquisition;
// Initilisation of the contoller board CAN network
Init_Aton_CAN();
// Clear the screen
clsscr();
// Initialization of different frames and sending to th[illegible]N [illegible]s
// The frames of type "IM" (Command frame ); Identifi[illegible]ion dat[illegible]
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.identificateu[illegible]Asservissement;
// To set the Input / Output
T_IM_Asservissement.data[0]=0x1F;      // G[illegible]R R[illegible]ter [illegible]ss (I/O direction)
                                                       doc MCP25050 Page 16
T_IM_Asservissement.data[1]=0xEF;      [illegible] M[illegible] -[illegible] affected
T_IM_Asservissement.data[2]=0xE3;      [illegible]a[illegible] ->  1 f[illegible]input and 0 for output
          // GP7=fs Input; GP6=fc[illegible]nput[illegible] Inp[illegible] GP4=ValidIP Output;
          // GP3=PWM2 Output; GP2=[illegible]1 Ou[illegible]input; GP0=AN0 Input;
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Asservissement[illegible]t's t[illegible] first sent frame
           Cptr_TimeOut=0;                                   // We test if the module responds well
           do{Cptr_TimeOut++;}while((Lir[illegible]me([illegible]ame_Recue)==0)&&(Cptr_TimeOut<100));
           if(Cptr_TimeOut==100)
                    {if(I_Message_Pb_Affiche=[illegible]
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response to the command frame in initialization \n");
                               printf("  Check whether the power supply 12V is OK \n");}}
    }while(Cptr_TimeOut==100);
clsscr();
// To set outputs to 0
T_IM_Asservissement.data[0]=0x1E;      // GPLAT Register address (I/O register)
T_IM_Asservissement.data[1]=0x1C;      // Mask -> GP4,3,2 outputs are affected
T_IM_Asservissement.data[2]=0x00;      // Value ->  the 3 output set to 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP2 output by PWM1
T_IM_Asservissement.data[0]=0x21;      // T1CON Register address
T_IM_Asservissement.data[1]=0xB3;      // Mask -> only bit 7,5,4,1,0 affected
T_IM_Asservissement.data[2]=0x80;      // Value ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); //Wait for the response
// To set PWM1 output signal frequency
T_IM_Asservissement.data[0]=0x23;      // PR1 Register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0xFF;      // Value  ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM1 cyclic duty to 0
T_IM_Asservissement.data[0]=0x25;      // PWM1DC Register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=100;       // Value ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To validate the power circuit
T_IM_Asservissement.data[0]=0x1E;      // GPLAT Register address (I/O Register)
```

```
T_IM_Asservissement.data[1]=0x10;      // Mask -> GP4 (ValidIP) output is affected
T_IM_Asservissement.data[2]=0x10;      // Value ->  ValidIP=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To start the conversion Ana -> Dig
T_IM_Asservissement.data[0]=0x2A;      // ADCON0 Register address
T_IM_Asservissement.data[1]=0xF0;      // Mask -> bits 7..4 affected
T_IM_Asservissement.data[2]=0x80;      // Value ->  ADON=1 and "prescaler rate"=1:32
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To define the mode of conversion
T_IM_Asservissement.data[0]=0x2B;      // ADCON1 Register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0x0C;      // Value ->  see doc MCP25050 page 36
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To acquire the conversion A -> D results and the limit switch
// Frame of type "IRM" (remote frame): Identification data
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; // Ask for the values of 8 registers
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;
// To display the title
gotoxy(1,2);
printf(" EXPERIMENT N°7_1:   THE WINDSHIELD WIPER BLADE SPEED CONTROL   \n");
printf(" ******************************************************************** \n");
printf("     - Acting on the potentiometer on Servo-system board \n");
printf("     - In open-loop (Without the speed measurement) \n");
printf(" ******************************************************************** \n


// MAIN LOOP
//*******************
while(1)
          {Ctpr_Acquisition++;
          if(Ctpr_Acquisition==10)
          {Ctpr_Acquisition=0;
          // Acquire the conversion results
          Ecrire_Trame(T_IRM_Acquisition_FC_AN); // Demand a reading             limit switch and the analog conversion
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&    ptr    meOu   20   0));
           if(Cptr_TimeOut==20000)
                    {clsscr(),gotoxy(2,10);
                     printf(" No response to the remote frame whi   acqu        conversion results \n");
                     printf("  Reload the program and restart
                     do{}while(1);} // Stop
           else {if(Trame_Recue.ident.extend.identificate    ident==         TRM8    sservissement)
                              // If the identifier is cor   t
                               {AN0H =Trame_Recue.data      //      r       Vo   ge potentiometer
                               AN1H =Trame_Recue.data   ;   // Re       B speed measurement
                               AN10L =Trame_Recue.dat  4];      Reco   the LSB AN1 et AN0
                               // Deal with the data   d re       lts
                               S_Val_Pot=(unsigned sho   (AN0           //Transfer with the cast
                               S_Val_Pot=S_Val_Pot           // S    ted 2 bits to the left
                               S_Temp=(unsigned               ); // To recover only 2 bits AD1 and AD0
                               S_Val_Pot=S_Va  ot|(S_Te
                               S_Mesure_Vite  =(u    ned     )(AN1H);        //Transfer with the cast
                               S_Mes   Vite  e=S      Vi   sse<<2; // Shifted 2 bits to the left
                               S_Te      ig    s       &0xC0); // To recover only 2 bits AD1 and AD0
                               S_M   re_   es   S_Mesure   tesse|(S_Temp>>6);
                               T_I   sse        dat   ]=0x25;      // PWM1DC Register address (Load command speed)
                               T_I   sservi        a[2]=(AN0H) ;   // Value ->  Command speed
                               Ecri   Trame(T_I     sservissement);
                               do{}wh       Tra   (&Trame_Recue)==0); // Wait for the response
                              }} // End of   quis   ion and processing
          Cptr_Affichage++;
          if(Cptr_Affichage==1000)
                    {Cptr_Affichage=0;
                     Cptr_Affichage=0;
                     gotoxy(1,10);
                     printf("  Value of Potentiometer input:             \n");
                     printf("  Value of the motor Command                 \n");
                     printf("  Value of speed Measurement input:            \n");
                    // Display the amouts
                    gotoxy(1,10);
                     printf("  Value of Potentiometer input: %d\n",S_Val_Pot);
                     printf("  Value of the motor Command: %d \n",AN0H);
                     printf("  Value of speed Measurement input: %d\n",S_Mesure_Vitesse);}
          } //  End of the main loop
} // End of the main function
```

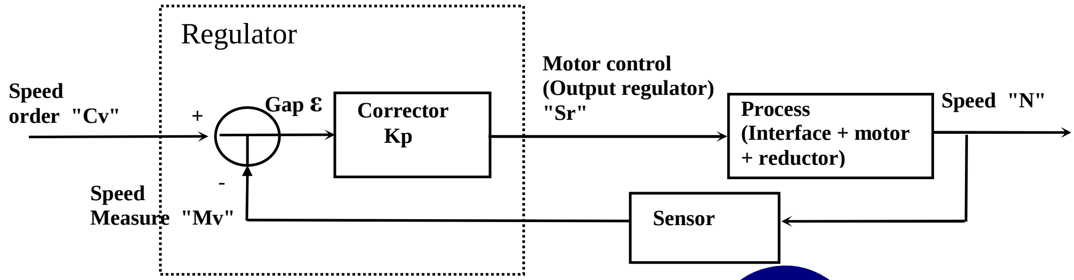
7.3 Solution step n°2

7.3.1 Analysis step n°2

Principle:

In the case of controlling the motor speed in proportional mode, the control variable is a function of the gap noted "ε" (ε = speed instruction – speed measure).

For the program, the speed order will be the conversion result of potentiometer voltage applied to the analog AN0 (GP0) input and speed measurement, the conversion result of the F / U converter output applied on the AN1 (GP1) input. (It will be a digital regulator, so it should be sampled).



In the case of regulation by proportional action, Sr will be expre... r = Kp (Cv, Mv).
The calculation for a regular time interval is called "sam... g p... io... d... oted "Te".
The coefficient Kp will be considered in the program a... whe... t act... lly 4 least significant bits represent the fractional block: Kp = 0x10 -> value = 1; Kp = 20h -> v... ue = ... etc. ..
Kp = 0x08 -> value = 0.5, Kp = 0x04 -> value = 0.25, K... > ... alue = 0.125, etc ...
Ultimately Kp is within the range: 0 ⩽ 15.9375 ⩽ K...

Supplementary declarations of step n°1

Carry out the sampling period:

It is possible to use the capability of MCP250... c... it se... ing spontaneously and after a regular time interval, a frame "Read A / D Regs" contai... ng ... the ... ta" is the conversion results (doc MCP25050 page 22).
This requires initializing the "Schedule... ... issi... function (doc MCP25050 page 24).
This requires initializing, by frames ... pe "IM... STCON" and "IOINTEN" registers.
See in EX 5 Chapter "analysi... the ... fi... of ... T_IM_Asservissement" frames.

→ *To set the sampling peri... (fr... ... ending frames by "'sequencer")*

This frequency depends on t... value lo... ded into the "STON" register

```
T_IM_Asservissement.data[0]=0x  ; // STON Register address
 (doc MCP25050 p15) 10H + shift = 10H + 1CH = 2CH
T_IM_Asservissement.data[1]=0xFF; // Mask : all bits are affected
T_IM_Asservissement.data[2]=0xD2; // Value: (see doc MCP25050 page 24).
     b7 -> STEN = 1  -> to activate the sequencer
     b6 -> STMS = 1 -> for the frames with 8 bytes (contain the conversion results)
     b5,B4 = 0 1 -> basic period = 16.4096.Tosc
     b3..b0 = 0010 -> period multiplier = 3
```

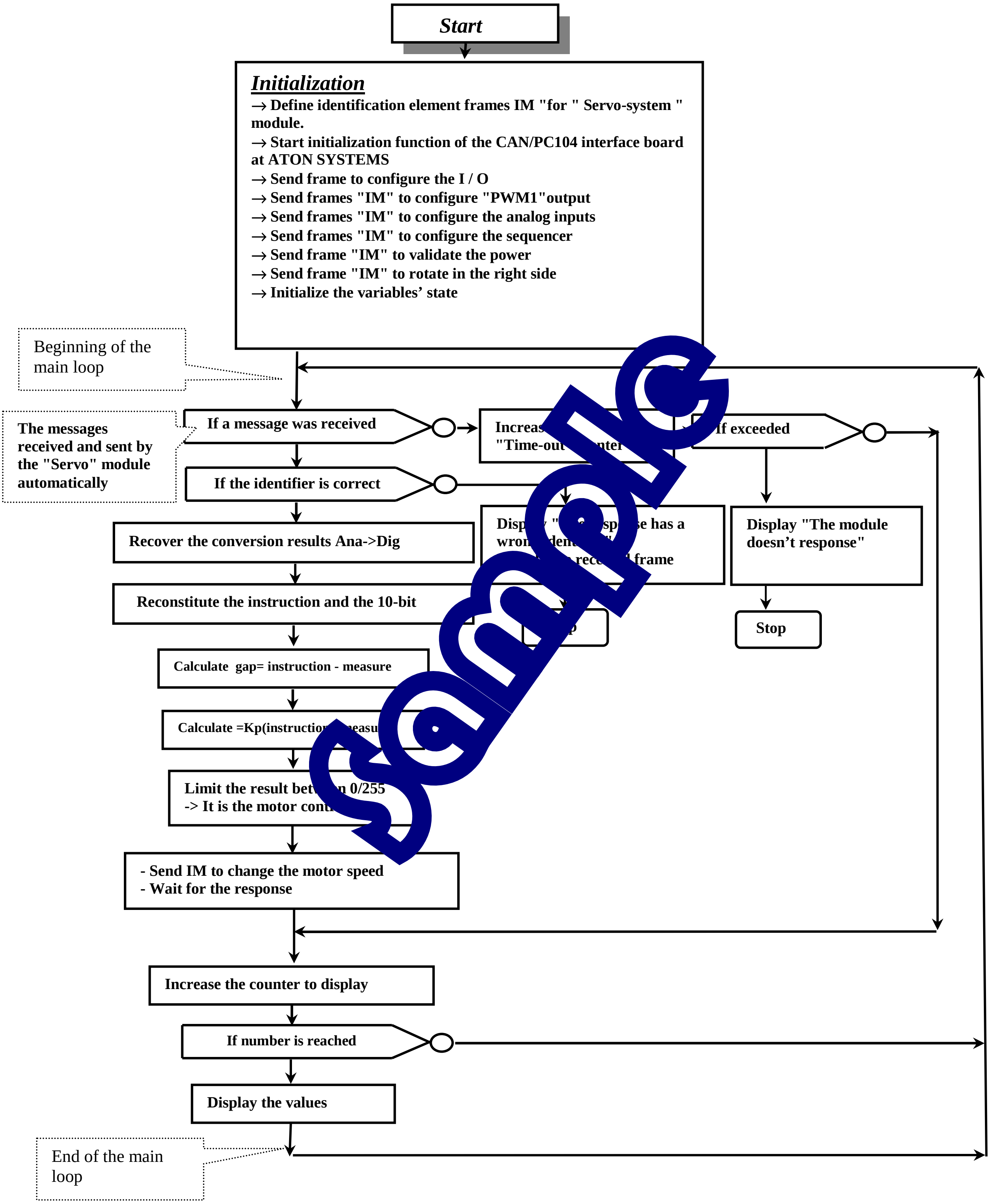
The quartz frequency located on the "Servo-system" board is equal to 16Mhz (Tosc = $1/16.10^6$), the period for sending frames will equal to $16.4096.3/16.10^6$ = 12 mS.

→ *To enable auto-conversion converters Ana -> Dig*

It must initialize the register "IOINTEN" especially two bits corresponding to the two analog inputs used in this application.

```
T_IM_Asservissement.data[0]=0x1C; // IOINTEN Register address
 (doc MCP25050 p15) 00H + shift = 00H + 1CH = 1CH
T_IM_Asservissement.data[1]=0x03; // Mask : only the bit 0 and 1 are affected
T_IM_Asservissement.data[2]=0x03; // Value: (see doc MCP25050 page 27).
```

7.3.2 Flowchart step n°2



7.3.3 "C" Program step n°2

```
/*******************************************************************************************
*       Experiment on  EID210 / CAN Network –  V.M.D  (Multiplexed Didactic Vehicle)
********************************************************************************************
*   EXPERIMENT N°7_2:  THE WINDSHIELD WIPER BLADE SPEED CONTROL
*-------------------------------------------------------------------------------------------
*   SPECIFICATIONS :
*  *********************
          * Step 2: Vary the speed of the motor with the potentiometer implanted by the
*                   "Servo-system" module
*                   The motor control is a closed-loop with a proportional corrector
*                        Command = Kp*Gap = Kp * (Instruction - Measure)
*--------------------------------------------------------------------------------------------
*  File Name :  CAN_VMD_TP7_2.C
* *****************
*********************************************************************************************
// Declaration of included files
#include <stdio.h>
#include"Structures_Donnees.h"
#include"cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"

// Declaration of various frames of communication
Trame Trame_Recue;  // For the frame that has just been received by the controller
// Frames of type "IM" (Input Message -> Command frame)
Trame T_IM_Asservissement;    // For the order of the motor
Trame T_IRM_Acquisition_FC_AN;          // For the acquisition of the analog input and the limit switch
// Declaration of variables
// For various indicators (binary variables)
union byte_bits Indicateurs; // Bit Structures
#define I_Sens_Rotation Indicateurs.bit.b0
#define I_Attente_Reponse_IRM Indicateurs.bit.b1
#define I_Message_Pb_Affiche Indicateurs.bit.b2
// For the conversion results
unsigned short S_Mesure_Vitesse,S_Consigne,S_Temp;
unsigned char AN0H,AN1H,AN10L;
// For speed regulator
int Ecart,Resultat_Calcul;
unsigned char Cde_Moteur;
// Declaration of the constant of the regulator
#define Kp 6       // Proportional action coefficient -> This must be a w    vari    e
                         //real value Kp/16 = 6/16
//======================
// MAIN FUNCTION
//======================
main()
{
//  INITIALIZATIONS
//------------------
// Declaration of local variables in the main function
// Different counters
unsigned int Cptr_Affichage,Cptr_TimeOut,Ctpr_Acquisiti
// Initilization of the contoller board CAN network
Init_Aton_CAN();
// Clear the screen
clsscr();
// Initialization of different frames and sending to t  AN mo   s
// The frames of type "IM" (Command frame); Ide          t
T_IM_Asservissement.trame_info.registre=0x00;
T_IM_Asservissement.trame_info.champ.extend=
T_IM_Asservissement.trame_info.champ lc=0x0
T_IM_Asservissement.trame_info.cham   0;
T_IM_Asservissement.ident.extend.i  ti     .ident=I   _T_IM_Asservissement;
// To set the Input/Output
T_IM_Asservissement.data[0]=0x1F;                   address (I/O direction)
                                                      // doc MCP25050 Page 16
T_IM_Asservissement.data[1]=0xEF;       // Mask   Bit 7 not affected
T_IM_Asservissement.data[2]=0xE3;          ue      1 for input and 0 for output
          // GP7=fs Input; GP6=fcg Input;  =fc  nput;  GP4=ValidIP Output;
          // GP3=PWM2 Output; GP2=PWM1 Out     =AN1 Input; GP0=AN0 Input;
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Asservissement); // It's the first sent frame
           Cptr_TimeOut=0;                                // We test if the module responds well
           do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_TimeOut<500));
           if(Cptr_TimeOut==500)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response to the command frame in initialization \n");
                               printf("  Check whether the power supply 12V is OK\n");}}
    }while(Cptr_TimeOut==500);
clsscr();
// To set outputs to 0
T_IM_Asservissement.data[0]=0x1E;      // GPLAT register address (I/O register)
T_IM_Asservissement.data[1]=0x1C;      // Mask -> GP4,3,2 outputs are affected
T_IM_Asservissement.data[2]=0x00;      // Value ->  the 3 outputs set to 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP2 output by PWM1
T_IM_Asservissement.data[0]=0x21;      // T1CON register address
T_IM_Asservissement.data[1]=0xB3;      // Mask -> only bit 7,5,4,1,0 affected
T_IM_Asservissement.data[2]=0x80;      // Value ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM1 output signal frequency
T_IM_Asservissement.data[0]=0x23;      // PR1 register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0xFF;      // Value ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM1 cyclic duty to 0
T_IM_Asservissement.data[0]=0x25;      // PWM1DC register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
```

```
T_IM_Asservissement.data[2]=100;       // Value ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To validate the power circuit
T_IM_Asservissement.data[0]=0x1E;      // GPLAT register address (I/O register)
T_IM_Asservissement.data[1]=0x10;      // Mask -> GP4 (ValidIP) output is affected
T_IM_Asservissement.data[2]=0x10;      // Value-> ValidIP=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To start the conversion Ana -> Dig
T_IM_Asservissement.data[0]=0x2A;      // ADCON0 register address
T_IM_Asservissement.data[1]=0xF0;      // Mask -> bits 7..4 affected
T_IM_Asservissement.data[2]=0x80;      // Value ->  ADON=1 and "prescaler rate"=1:32
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To define the mode of conversion
T_IM_Asservissement.data[0]=0x2B;      // ADCON1 register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0x0C;      // Value ->  see doc MCP25050 page 36
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To validate the sequencer
T_IM_Asservissement.data[0]=0x2C;      // STON register address
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0xD2;      // Value ->  see doc MCP25050 page 24
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To validate the sequencing of analog inputs 0 and 1
T_IM_Asservissement.data[0]=0x2C;      // STON register address
T_IM_Asservissement.data[1]=0x03;      // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0x03;      // Value ->  see doc MCP25050 page 24
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// For future frames IM in the main loop (Command Motor)
T_IM_Asservissement.data[1]=0xFF;      // Mask -> all bits are affected
// To acquire the conversion A -> D results and the limit switch
// Frame of type "IRM" (remote frame): Identification data
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; // Ask for the value[illegible] 8 re[illegible]rs
T_IRM_Acquisition_FC_AN.trame_info.champ.rtr=1;
T_IRM_Acquisition_FC_AN.ident.extend.identificateur.ident=Ident_T_IR[illegible]me[illegible] //
Cptr_Affichage=0;
Ctpr_Acquisition=0;
Cptr_TimeOut=0;
// To display the title
gotoxy(1,2);
printf("  EXPERIMENT N°7_2:   THE WINDSHIELD WIPER BLADE SPE[illegible] \n[illegible]
printf("  *******************************************[illegible]*****[illegible]\n");
printf("      - Acting on the potentiometer on servo-syst[illegible] board [illegible]
printf("      - In closed-loop on proportional mode Sr = [illegible](I[illegible]uctio[illegible]) \n");
printf("  *********************************************[illegible]****[illegible]**** \n");

// MAIN LOOP
//*******************
while(1)
          {// A frame showing the status is recei[illegible] re[illegible]ar time interval
           // Function 'Test manager' of 'Serv[illegible] activated
          if(Lire_Trame(&Trame_Recue)!=1) // [illegible]rame of [illegible] has not arrived ?
                    {Cptr_TimeOut++;
                     if(Cptr_TimeOut[illegible]0000)
                              {clss[illegible]to[illegible]2,[illegible]
                               pri[illegible]("[illegible]r[illegible] of conv[illegible]ion results for a long time \n");
                               pri[illegible]("[illegible]m and restart it \n");
                               do{[illegible]ile(1);}[illegible]
           else {Cptr_TimeOut=0;
                     if(Trame_Recue.[illegible]tend[illegible]entificateur.ident==Ident_T_OB_Asservissement)
                              // If the [illegible]fie[illegible]is correcte
                              {AN0H =Tram[illegible]cue[illegible]ta[2];   // Recover MSB Voltage potentiometer
                               AN1H =Trame_[illegible]ata[3];   // Recover MSB speed measurement
                               AN10L =Trame_Re[illegible]e.data[4]; // Recover the LSB, AN1 and AN0
                               // Deal with the data and restore results
                               S_Consigne=(unsigned short)(AN0H);     //Transfer with the cast
                               S_Consigne=S_Consigne<<2;    // Shifted 2 bits to the left
                               S_Temp=(unsigned short)(AN10L&0x0C); // To recover only 2 bits AD1 and AD0
                               S_Consigne=S_Consigne|(S_Temp>>2);
                               S_Mesure_Vitesse=(unsigned short)(AN1H);         //Transfer with the cast
                               S_Mesure_Vitesse=S_Mesure_Vitesse<<2; // Shifted 2 bits to the left
                               S_Temp=(unsigned short)(AN10L&0xC0); // To recover only 2 bits AD1 and AD0
                               S_Mesure_Vitesse=S_Mesure_Vitesse|(S_Temp>>6);
                               // Calculate the command measure
                               Ecart = S_Consigne - S_Mesure_Vitesse;
                               Resultat_Calcul = (Kp*Ecart)>>4;
                               if(Resultat_Calcul>255)Resultat_Calcul=255;
                               if(Resultat_Calcul<0)Resultat_Calcul=0;
                               Cde_Moteur=(unsigned char)(Resultat_Calcul);
                               T_IM_Asservissement.data[0]=0x25;      // PWM1DC register address (load speed command)
                               T_IM_Asservissement.data[2]=Cde_Moteur;          // Value ->  Speed command
                               Ecrire_Trame(T_IM_Asservissement);
                               do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
                              }} // End of acquisition and processing
          Cptr_Affichage++;
          if(Cptr_Affichage==5000)
                    {Cptr_Affichage=0;
                     gotoxy(1,12);
                     printf("  Value of instruction input: %d\n",S_Consigne);
                     printf("  Value of speed measure input: %d\n",S_Mesure_Vitesse);
                     printf("  Gap = Instruction - Measure: %d\n",Ecart);
                     printf("  Value of the motor command:%d\n",Cde_Moteur);
                     printf("  We must verify the relationship:\n");
                     printf("  Motor command = (Kp*Gap)/16 with Kp= %d\n",Kp);}
          } //  End of the main loop
} // End of the main function
```

7.4 Solution step n°3

7.4.1 Analysis step n°3

Principle:

In the case of controlling the motor speed proportional + integral mode, the control variable is a function of the gap noted "ε" (ε = speed instruction – speed measure) at a sampling instant but also the gap at the previous sampling instant, depending on the relationship of the following recursion**:**

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

  with → $S_{I\,n}$ integral action value t = n.Te (Te sampling period)

  → $S_{I\,n-1}$ integral action value t = (n-1).Te (to the previous sampling)

  → K_I coefficient of integral action

  → ε_n Gap t = n.Te (Te sampling period)

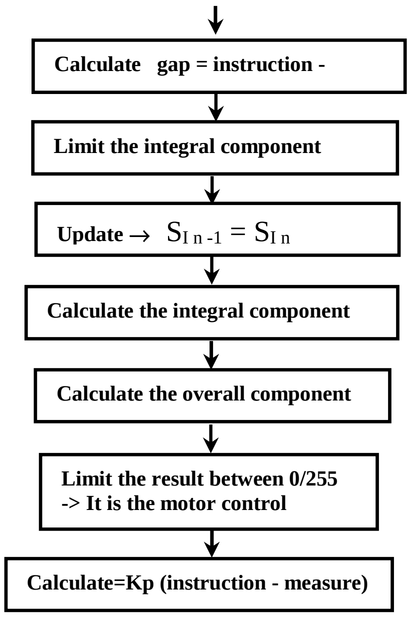
- Global : $Sr_n = Kp.(S_{I\,n} + \varepsilon_n)$

Remark:

- According to the " $S_{I\,n}$ " expression ,at each sampling interval, this variable increases the value of KI. ε n (It is a constant. If ε n is a constant ε n → the integral of a constant is a function(ε n x + c)).
- If the instruction is a constant, integral action requires the term (steady-state) ε n = 0. This causes that the signal measurement becomes equal to the reference signal. The coefficient of closed-loop transfer (Output / Instruction) becomes equal to the inverse of the transfer coefficient of the sensor (Measure / Output).
- No matter what reason it is, the integral action cannot cancel the gap (Measure serial in default, speed instruction too high ..). $S_{I\,n}$ is forbidden to reach the prohibitive value. For this reason, it must limit the $S_{I\,n}$ below Srmax / Kp = 255/Kp.

The Kp and K_I coefficients will be considered in the program [illegible]ble variable, but actually 4 least significant bits represent the fractional part: K=0x10 -> value =1; K=20h -> value =2; etc..
K=0x08 -> value =0,5; K=0x04 -> value =0,25; K=0x02 -> value =0,125; etc…
Ultimately Kp is within the range: 15,9375 ≤ K[illegible]

7.4.2 Part of Flowchart Step n°3



7.4.3 Part of Program Step n°3

```
          {// A frame showing the status is received after a regular time interval
           // Function 'Test manager' of module 'Servo-system' is activated
          if(Lire_Trame(&Trame_Recue)!=1) // A frame of results has not arrived ?
                    {Cptr_TimeOut++;
                     if(Cptr_TimeOut==10000)
                    {clsscr(),gotoxy(2,10);
                     printf(" No frame of conversion results for a long time \n");
                               printf("   Reload the program and restart it \n");
                              do{}while(1);}} // Stop
           else {Cptr_TimeOut=0;

if(Trame_Recue.ident.extend.identificateur.ident=Ident_T_OB_Asservissement)
          //If the identifier is correcte
          {AN0H =Trame_Recue.data[2];   // Recover MSB Voltage potentiometer
           AN1H =Trame_Recue.data[3];   // Recover MSB speed measurement
          AN10L =Trame_Recue.data[4]; // Recover the LSB, AN1 and AN0
                     // Deal with the data and restore results
          S_Consigne=(unsigned short)(AN0H);      //Transfer with the cast
          S_Consigne=S_Consigne<<2;     // Shifted 2 bits to the left
          S_Temp=(unsigned short)(AN10L&0x0C); // To recover only 2 bits AD1 and
AD0
          S_Consigne=S_Consigne|(S_Temp>>2);
          S_Mesure_Vitesse=(unsigned short)(AN1H);         //Transfer with the cast
          S_Mesure_Vitesse=S_Mesure_Vitesse<<2;  // Shifted 2 bits to the left
          S_Temp=(unsigned short)(AN10L&0xC0); // To recover only 2 bits AD1 and
AD0
          S_Mesure_Vitesse=S_Mesur[illegible]esse|(S_Temp>>6);
          Ecart = S_Consigne - [illegible]esse;
          SIn = KI*Ecart;         [illegible]To calculate the integral
                    SIn = SIn[illegible];
                    SI[illegible] = SIn[illegible]n[illegible]       // Add the previous value
                    [illegible]SI[illegible]ax)[illegible]in[illegible]x; // Limit the value of the integral
                    [illegible](SI[illegible]ax)SIn=-[illegible]_max;
                    [illegible]1=SIn[illegible]               // Update the stored value
                    [illegible]_calcul = Kp*(Ecart+SIn);
                              Re[illegible]at_Calcul = Resultat_Calcul>>4;
if(Resultat_Calcul>25[illegible]Ca[illegible]=255;      // To limit the accepted value
          if(Resul[illegible]at_Calcul=0;          // By the motor command
           Cde_mo[illegible]r=(unsig[illegible])(Resultat_Calcul);
          T_IM_A[illegible]erv[illegible]nt[illegible]ta[0]=0x25;      // PWM1DC register address
          T_IM_A[illegible]erv[illegible]nt[illegible]ta[2]=Cde_moteur;// Value ->  Speed Command
          Ecrire[illegible]ame[illegible]A[illegible]ervissement);
          do{}whi[illegible]ire[illegible](&Trame_Recue)==0); // wait for the response

// End of[illegible]cquisi[illegible]proc[illegible]ing
```

8 EXPERIMENT N°8 : THE WINDSHIELD WIPER SYSTEM CONTROL

8.1 Topic

Purpose :	- Develop a real time application (with a speed control) defined by a specification. - Try different control modes (open-loop, closed-loop) of a controllable analog system driven by CAN network. - Carry out different types of digital controller (proportional action and integral action)
Specifications :	After a regular time interval, we can know its status by the module on which is connected the wiper stalk. According to the state of the wiper stalk, we control the motor (intermittent, position 1, position 2, etc.) → The different commands imposed by the position of the stalk wheel are displayed individually. → We control the motor working condition.

Necessary hardware and software :

PC Micro Computer using Windows ® 95 or later
Software: Editor -Assembler-Debugger
If programming in C, GNU; C / C ++ Compiler Ref: EID210101
Processor board 16/32 bit 68332 microcontroller and its software environment
(Editor-Cross Assembler-Debugger) Ref: EID210001
CAN PC/104 Network board in [illegible]EMS Ref NIC: EID004001

- 1 electronic "servo-system" module Ref : EID052001
- The operative block of the windshield wiper system Ref: EID053001
- 1 electronic module "8 inputs" Ref: EID050001.
- (If necessary) 1 wiper stalk Ref: EID057001

USB connection cable, or if not available use RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules (network "energy")

Time : 4 hours

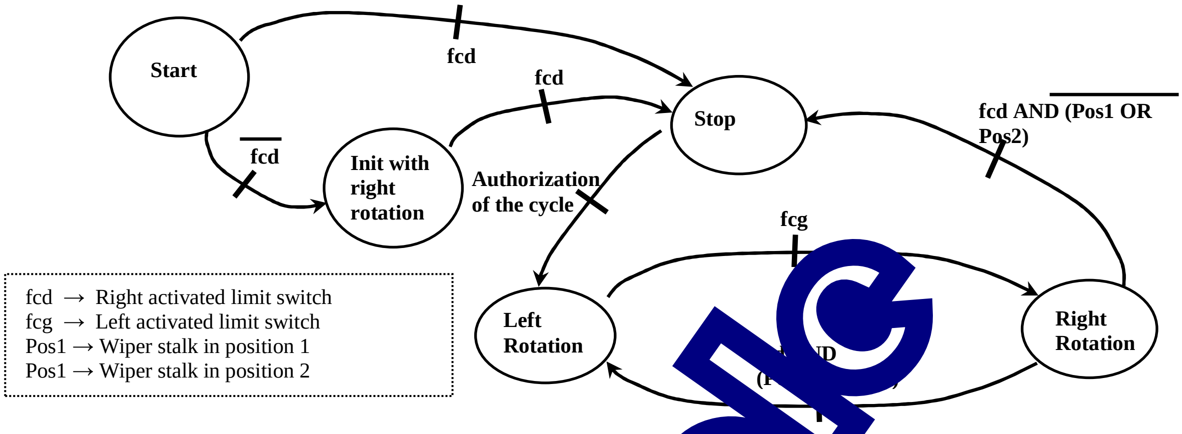
8.2 Solution

8.2.1 Analysis

Principle:

In this experiment the CAN network is made (besides the controller card, two modules) of:
- a "Servo-system" module on which is connected the motor and the sensors at the limit switch
- a "8 inputs" module which is possibly connected a wiper stalk

The requested cycle leads to the states diagram as the following:



Remarks:
- In both states, "Left Rotation" and "Right Rotation" the speed depends on the wiper stalk position: Position 1 or intermittent → low speed, Position 2 → high speed.
- If the wiper stalk is in the "Intermittent" position, a timer sets regularly the variable "Authorization cycle" to 1. It recovers to 0 by the activation of "Left rotation" state. The time interval between two activation "Authorization cycle " depends on the position of the wipers stalk wheel.

Configuration and control Frame (type "IM") of servo-system module → Idem TP5
Acquisition frames (type "IRM") of the limit switch of the servo-system module → Idem TP6
Configuration frames ("IM" type) of module 8 inputs "Wiper stalk"

The identifier defined in Chapter 2, for an "IM" (Input Message -> Command Frame) sent to the "Wiper stalk" board is: 0x05880000

→ Definition of structured variables under the "`Trame`" model:

```
Trame T_IM_Commodo_EG;
```

→ Definition of the different elements of the "`T_IM_Asservissement`" structured variable

```
T_IM_Commodo_EG.trame_info.register=0x00;  // All bits are initialized to 0
T_IM_Commodo_EG.trame_info.champ.extend=1; // Work in extended mode
T_IM_Commodo_EG.trame_info.champ.dlc=0x03; //There will be a 3 bytes data
T_IM_Commodo_EG.ident.extend.identificateur.ident=0x05880000;
```

→ *Enable and configure the conversion from Analog to Digital*

According to the technical manual MCP25050 circuit (pages 34 to 37) :

Initialize the ADCON0 register

```
T_IM_ Commodo_EG.data[0]=0x2A; // ADCON0 register address
  (doc MCP25050 p15) 0EH + shift = 0EH + 1CH = 2AH
T_IM_ Commodo_EG.data[1]=0xF0; // Mask : only the bit 7 is affected
T_IM_ Commodo_EG.data[2]=0x80; // Value: ADON=1 -> Activation of the converter
 and "prescaler rate" = 1:32
```

As well as ADCON1 register.

```
T_IM_ Commodo_EG.data[0]=0x2B; // ADCON1 register address
  (doc MCP25050 p15) 0FH + shift = 0EH + 1CH = 2BH
T_IM_ Commodo_EG.data[1]=0xFF; // Mask:  all 8 bits are affected
T_IM_ Commodo_EG.data[2]=0x0E; // Value: (doc MCP25050 p36)
      b7=ADCS1=0; b6=ADCS0=0 → Frequency Fosc/2
      b5=VCFG1=0; b4=VCFG0=0 → Range of input voltage 0/+5V
```

PCFG3:PCFG0=1110 → Converting the analog input 0 (on GP0)

Acquisition frames (type "IRM" Input Request Message) of the wiper stalk state:

The identifier defined in Chapter 1, for an "IRM"; sent to the " wiper stalk " board is :

0x05840000

→ Definition of structured variables under the "`Trame`" model:

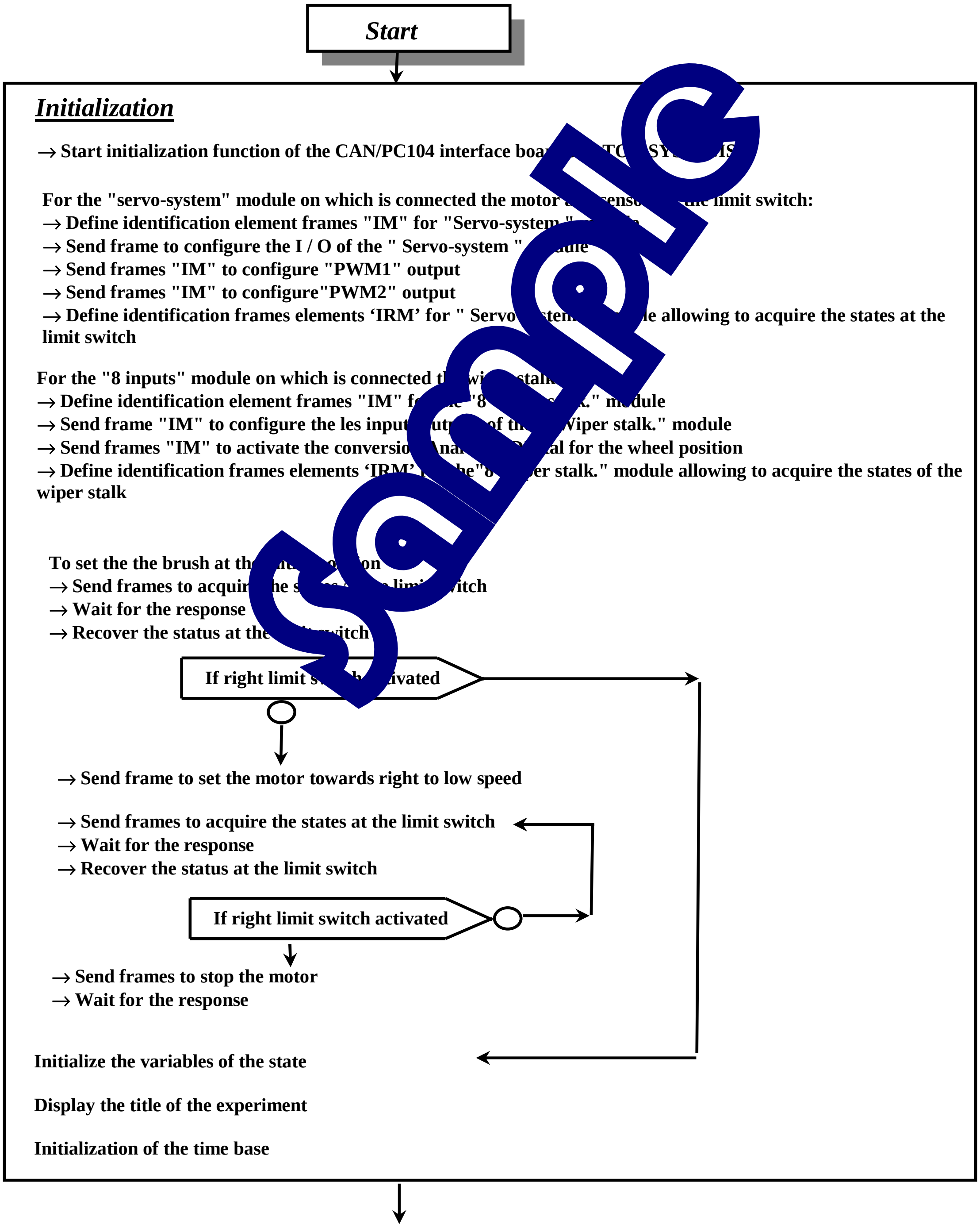
`Trame T_IRM_Etat_Commodo_EG;` // Frame appointed for enquiry of the 8E module to acquire the status of the wiper stalk.

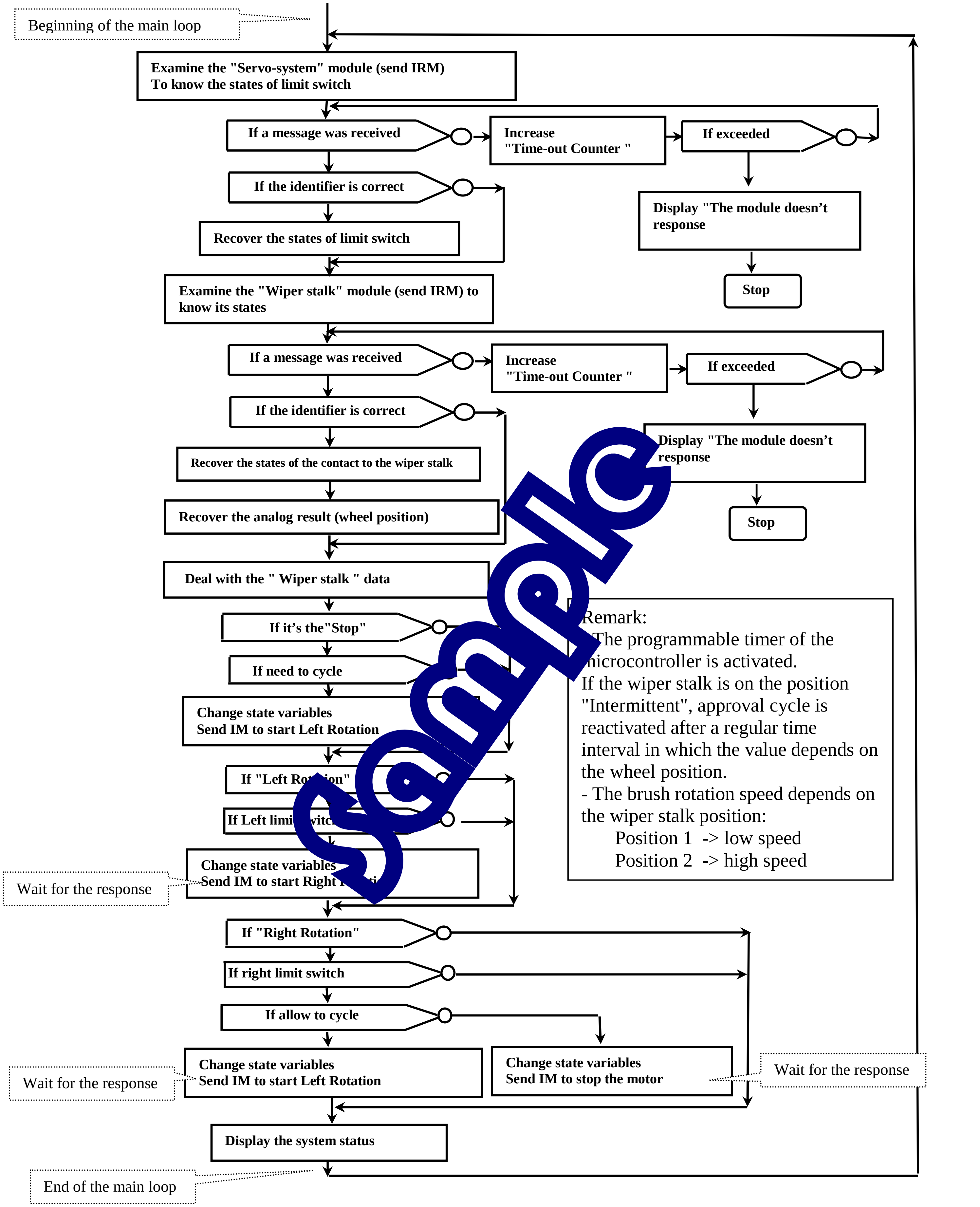
→ Access and definition of the different elements of the " `Lecture_FC` " structured variable

```
T_IRM_ Etat_Commodo_EG.trame_info.register=0x00;  // All bits are initialized to 0
T_IRM_ Etat_Commodo_EG.trame_info.champ.extend=1; // Work in extended mode
T_IRM_ Etat_Commodo_EG.trame_info.champ.dlc=0x08; // There will be a 8 bytes data
T_IRM_ Etat_Commodo_EG.ident.extend.identificateur.ident=0x05840000;
```

In response to this query frame, we recover the logic states in the rank 1 data and the conversion result of the wheel position in the rank 2 data.

8.2.2 Flowchart




Beginning of the main loop
Examine the "Servo-system" module (send IRM)
To know the states of limit switch
If a message was received
Increase
"Time-out Counter "
If exceeded
If the identifier is correct
Display "The module doesn't
response
Recover the states of limit switch
Stop
Examine the "Wiper stalk" module (send IRM) to
know its states
If a message was received
Increase
"Time-out Counter "
If exceeded
If the identifier is correct
Display "The module doesn't
response
Recover the states of the contact to the wiper stalk
Recover the analog result (wheel position)
Stop
Deal with the " Wiper stalk " data
If it's the"Stop"
If need to cycle
Change state variables
Send IM to start Left Rotation
Change state variables
Wait for the response
Remark:
The programmable timer of the
icrocontroller is activated.
If the wiper stalk is on the position
"Intermittent", approval cycle is
reactivated after a regular time
interval in which the value depends on
the wheel position.
- The brush rotation speed depends on
the wiper stalk position:
Position 1 -> low speed
Position 2 -> high speed
If "Right Rotation"
If right limit switch
If allow to cycle
Change state variables
Send IM to start Left Rotation
Wait for the response
Change state variables
Send IM to stop the motor
Wait for the response
Display the system status
End of the main loop

8.2.3 "C" Program

```
/*************************************************************************************
*       Experiment on  EID210 / CAN Network ¨V.M.D  (Multiplexed Didactic Vehicle)
**************************************************************************************
*   EXPERIMENT N°8:  THE WINDSHIELD WIPER SYSTEM CONTROL
*-----------------------------------------------------------------------------------
*   SPECIFICATIONS :
*  ********************
*  We want to control the windshield wiper by the wiper stalk
*  The function works like this:
*         - position 'stop'
*         - position 'intermittent' -> the brush swings back and forth separated with the
*           interval which is controlled by the wheel integrated in the wiper stalk
*         - position 'one' -> the brush swings back and forth in low speed
*         - position 'two' -> the brush swings back and forth in high speed
*  In the mode 'intermittent', the time interval between two beats is generated
*   by the ' programmable delay ' integrated in the micro-controller
*  Display on the screen the diverse wiper stalk inputs states.
*-------------------------------------------------------------------------------------
*  File Name:  CAN_VMD_TP8.C
* ****************
****************************************************************************************/
// Declaration of included files
#include <stdio.h>
#include"Structures_Donnees.h"
#include"cpu_reg.h"
#include "eid210_reg.h"
#include "CAN_vmd.h"

// Declaration of variables
// For various indicators (binary variables)
union byte_bits Indicateurs,FC; // Bits Structure
#define I_Autorise_Cycle Indicateurs.bit.b0      // Authorize to cycle
#define I_Intermittent Indicateurs.bit.b1
#define I_Message_Pb_Affiche Indicateurs.bit.b2
#define Etat_Arret Indicateurs.bit.b3            // Status "Stop th   rush      right l   t switch"
#define Etat_Rot_Droite Indicateurs.bit.b4       // Status "Brush on   ht r
#define Etat_Rot_Gauche Indicateurs.bit.b5       // Status "Brush on  le   rotat
// For the limit switch
#define Etat_FC FC.valeur // For the group of state of the limit swit
#define fs FC.bit.b7          // For over limit switch
#define fcg FC.bit.b6         // For left limit switch
#define fcd FC.bit.b5         //  For right limit switch
// Declaration of various frames of communication
Trame Trame_Recue;  // For the frame that has just been received    the     r    er
// Frames of type "IM" (Input Message -> Command frame)
Trame T_IM_Asservissement;    // For the motor control
Trame T_IM_Commodo_EG;                 // For the initial   tion          per    k
// Frames type "IRM" (Input Request Message -> remote fra   )
Trame T_IRM_Acquisition_FC;   // To acquire the status        mi     tch
Trame T_IRM_Etat_Commodo_EG;           // For the acq    ition         er Stalk state
// Diverse varibles
unsigned char Valeur_Analogique,Cde_Vitesse,Tempo_Fi   omp      ass  e_Irq,Compteur_Secondes;

//  For the delays
//Declaration of constants
#define Vitesse_Lente 60
#define Vitesse_Rapide 120
#define Tempo1 2    // unit in second
#define Tempo2 4
#define Tempo3 6
#define Tempo4 8
#define Tempo5 10
//  Interruption function of "Time    se"
//====================================
void irq_bt()
// Function runs every 10 mS
{if(I_Intermittent) // If the intermittent          on
          {Compteur_Passage_Irq++;
           if(Compteur_Passage_Irq==100)  // One second has passed
                    {Compteur_Passage_Irq=0;
                     Compteur_Secondes++;
                     if(Compteur_Secondes>=Tempo_Fin)
                              {Compteur_Secondes=0;
                               I_Autorise_Cycle=1;}
                    }}
} // End of the interruption function

//======================
// MAIN FUNCTION
//======================
main()
{
// Declaration of local variables in the main function
int Cptr_Affichage=0,Cptr_TimeOut;
//  INITIALIZATIONS
//-----------------
// To initialize the CAN industrial controller board
Init_Aton_CAN();
clsscr(); // Clear the screen
// he frames of type "IM" (Command frame); Identification data
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;
// To set the Input / Output
T_IM_Asservissement.data[0]=0x1F;      // GPDDR register address (I/O direction)
                                                          // doc MCP25050 Page 16
T_IM_Asservissement.data[1]=0xEF;      // Mask -> Bit 7 not affected
T_IM_Asservissement.data[2]=0xE3;      // Value ->  1 for input and 0 for output
```

```
          //GP7=fs Input;GP6=fcg Input;GP5=fcd Input; GP4=ValidIP Output;
          //GP3=PWM2 Output;GP2=PWM1 Output;GP1=AN1 Input;GP0=AN0 Input;
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM_Asservissement); // It's the first sent frame
           Cptr_TimeOut=0;              // Servo-system' -> We test if the module response well
           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(" No response to the command frame in initialization \n");
                               printf("  Check whether the supply 12V is OK\n");}}
    }while(Cptr_TimeOut==100);
clsscr(); // To clear the screen in case of the message display
// To set outputs to 0
T_IM_Asservissement.data[0]=0x1E;       // GPLAT register address (I/O register)
T_IM_Asservissement.data[1]=0x1C;       // Mask -> GP4,3,2 outputs are affected
T_IM_Asservissement.data[2]=0x00;       // Value ->  the 3 output set to 0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP2 output by PWM1
T_IM_Asservissement.data[0]=0x21;       // T1CON register address
T_IM_Asservissement.data[1]=0xB3;       // Mask -> only bit 7,5,4,1,0 affected
T_IM_Asservissement.data[2]=0x80;       // Value ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM1 output signal frequency
T_IM_Asservissement.data[0]=0x23;       // PR1 register address
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bits are affected
T_IM_Asservissement.data[2]=0xFF;       // Value ->  PR1=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP3 output by PWM2
T_IM_Asservissement.data[0]=0x22;       // Register T2CON address
T_IM_Asservissement.data[1]=0xB3;       // Mask -> only bit 7,5,4,1,0 affected
T_IM_Asservissement.data[2]=0x80;       // Value ->  TMR2ON=1; Prescaler2=1
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM2 output signal frequency
T_IM_Asservissement.data[0]=0x24;       // Register PR2 address
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bits are affec[illegible]
T_IM_Asservissement.data[2]=0xFF;       // Value->  PR2=255
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM1 cyclic duty to 0
T_IM_Asservissement.data[0]=0x25;       // PWM1DC register address
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bits are af[illegible]ed
T_IM_Asservissement.data[2]=0;          // Value ->  PWM1DC=0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM2 cyclic duty to 0
T_IM_Asservissement.data[0]=0x26;       // PWM2DC register a[illegible]
T_IM_Asservissement.data[1]=0xFF;       // Mask -> all bit[illegible]e a[illegible]
T_IM_Asservissement.data[2]=0;          // Value ->  PWM2[illegible]0
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for th[illegible]spon[illegible]
// To validate the power circuit
T_IM_Asservissement.data[0]=0x1E;       // GPLAT regi[illegible]er [illegible]ss ([illegible]register)
T_IM_Asservissement.data[1]=0x10;       // Mask -> GP[illegible](Val[illegible]ut is affected
T_IM_Asservissement.data[2]=0x10;       // Value ->  V[illegible]dIP=1[illegible]
Ecrire_Trame(T_IM_Asservissement);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wai[illegible]
// Mask for the future command IM
T_IM_Asservissement.data[1]=0xFF;       // M[illegible]-> [illegible] bits [illegible] affected
// To acquire the state of limit swi[illegible]
// Frames type "IRM" (remote frame[illegible]i[illegible]ati[illegible]
T_IRM_Acquisition_FC.trame_info.re[illegible]stre[illegible]00[illegible]
T_IRM_Acquisition_FC.trame_info.ch[illegible]p.ex[illegible]
T_IRM_Acquisition_FC.trame_info.ch[illegible].dlc=1;
T_IRM_Acquisition_FC.trame_info.cha[illegible]rtr=1;
T_IRM_Acquisition_FC.ident.extend.ide[illegible]teur[illegible]ent=Ident_T_IRM1_Asservissement;
                                        //[illegible]and[illegible]the state of the register GPIN

// To initialize the state "Wiper stalk"
// Frames type "IM" (command frame): Identific[illegible]tion data
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; // Ask for the values of 8 registers
T_IM_Commodo_EG.trame_info.champ.rtr=0;
T_IM_Commodo_EG.ident.extend.identificateur.ident=Ident_T_IM_Commodo_EG; //
// To activate the GP0  and GP7 as inputs
T_IM_Commodo_EG.data[0]=0x1F;           // GPDDR register address
T_IM_Commodo_EG.data[1]=0x7F;           // Mask -> All bits affected
T_IM_Commodo_EG.data[2]=0x7F;           // Value ->  8 bits as inputs
Ecrire_Trame(T_IM_Commodo_EG);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To start the conversion Ang -> Dig
T_IM_Commodo_EG.data[0]=0x2A;           // ADCON0 register address
T_IM_Commodo_EG.data[1]=0xF0;           // Mask -> bits 7,6,5,4 affected
T_IM_Commodo_EG.data[2]=0x80; // Value ->  ADON=1 and "prescaler rate"=1:32
Ecrire_Trame(T_IM_Commodo_EG);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To define the mode of conversion
T_IM_Commodo_EG.data[0]=0x2B;           // ADCON1 register address
T_IM_Commodo_EG.data[1]=0xFF;           // Masque -> tous les bits sont concernés
T_IM_Commodo_EG.data[2]=0x0E; // Value ->  see doc MCP25050 page 36(GP0 -> Analog Input)
Ecrire_Trame(T_IM_Commodo_EG);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To acquire the A-> D conversion results and wiper stalk states
// Frame type "IRM" (remote frame) Wiper stalk: Identification data
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; // Ask for the values of 8 registers
T_IRM_Etat_Commodo_EG.trame_info.champ.rtr=1;
T_IRM_Etat_Commodo_EG.ident.extend.identificateur.ident=Ident_T_IRM8_Commodo_EG;

// To recover system (Move the brush to the initial position on limit switch)
```

```
Ecrire_Trame(T_IRM_Acquisition_FC);    // Sending acquisition frame of the limit switch status
do{}while(Lire_Trame(&Trame_Recue)==0);          //Wait for the response
Etat_FC=~Trame_Recue.data[0];          // Recover the state of the limit switch
if(fcd==0)         //If the wiper brush is not on the right position, we control it to do the right rotation
          {T_IM_Asservissement.data[0]=0x25;     // PWM1DC register address
           T_IM_Asservissement.data[2]=Vitesse_Lente;      // Value ->  low speed
           Ecrire_Trame(T_IM_Asservissement);     // Send frame of motor contro
           do{}while(Lire_Trame(&Trame_Recue)==0);         //Wait for the response
           while(fcd==0)
                   {Ecrire_Trame(T_IRM_Acquisition_FC); // Sending acquisition frame of the limit switch status
                    do{}while(Lire_Trame(&Trame_Recue)==0);         // Wait for the response
                    Etat_FC=~Trame_Recue.data[0];         // Recover the state of the limit switch
                   }
          }// End of Initially positioning the brushes
 T_IM_Asservissement.data[0]=0x25;     // PWM1DC register address
 T_IM_Asservissement.data[2]=0;        // Value ->  no speed
 Ecrire_Trame(T_IM_Asservissement);
 do{}while(Lire_Trame(&Trame_Recue)==0);         // Wait for the response

// Initialization of system state variables
Etat_Arret=1,Etat_Rot_Droite=0,Etat_Rot_Gauche=0;
I_Autorise_Cycle=0,I_Intermittent;
Compteur_Secondes=0,Compteur_Passage_Irq=0;
// To display the title
gotoxy(1,2);
printf("  EXPERIMENT N :8   THE WINDSHIELD WIPER SYSTEM CONTROL  \n");
printf("  ***************************************************************** \n");
// To set the time base and the delay
SetVect(96,&irq_bt);         // load the auto-vector
PITR = 0x0048;               // An interrupt every 10 milliseconds
PICR = 0x0760;               //  96 = 60H

// MAIN LOOP
//*******************
while(1)
  {       // Acquire the status of limit switch
          //--------------------------------------
          Ecrire_Trame(T_IRM_Acquisition_FC); // Sending acquisitio[illegible]ame [illegible]limit s[illegible]ch status
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((Lire_Trame(&Trame_Recue)==0)&&(Cptr_[illegible]Out<[illegible]));
           if(Cptr_TimeOut==10000)
                   {clsscr(),gotoxy(2,10);
                    printf(" No response to the remote frame of t[illegible]limit s[illegible]
                    printf("  Reload the program and restart it\[illegible];
                    do{}while(1);} // Stop
           else { if(Trame_Recue.ident.extend.identificateur.iden[illegible]=Id[illegible]IR[illegible]Asservissement)
                             // If the identifier is correct
                             {Etat_FC=~Trame_Recue.data[0];[illegible]/ Re[illegible] the state of the limit switch
          // Acquire the state of the Wiper Stalk
          //---------------------------------------
          Ecrire_Trame(T_IRM_Etat_Commodo_EG); // Sendin[illegible]cq[illegible]tion [illegible]he limit switch status
          Cptr_TimeOut=0;
          do{Cptr_TimeOut++;}while((Lire_Trame(&Trame[illegible]cue)==0)[illegible]TimeOut<10000));
           if(Cptr_TimeOut==10000)
                   {clsscr(),gotoxy(2,10);
                    printf(" No response to the remo[illegible]frame [illegible] limit switch \n");
                    printf("  Reload the program[illegible]rt i[illegible]");
                    do{}while(1);} // Stop

           else { if(Trame_Recue.ident.exten[illegible]ide[illegible]cate[illegible]dent==Ident_T_IRM8_Commodo_EG)
                             // If t[illegible]iden[illegible]fie[illegible]
                             {Vale[illegible]od[illegible]G=[illegible]e.data[1];         // Recover the state of wiper stalk
                              Val[illegible]An[illegible]gi[illegible]=Trame_R[illegible]e.data[2];}} // Recover the state of the wiper stalk wheel
          // Deal with the state o[illegible]Wipe[illegible]
          //------------------------[illegible]
          if(Cde_EG_Av_Pos2)I_Autor[illegible]Cycle=1,[illegible]ntermittent=0,Cde_Vitesse=Vitesse_Rapide;
          // Start of the brush worki[illegible]e wi[illegible]high speed, if it is in the wiper stalk position 2)
          else {if(Cde_EG_Av_Pos1)I_Autori[illegible]Cyc[illegible]1,I_Intermittent=0,Cde_Vitesse=Vitesse_Lente;
          // Continue the brush working [illegible] wi[illegible] low speed, if it is in the wiper stalk position 1)
                   else      {if(Valeur_An[illegible]e>=200)I_Intermittent=0; // Position "stop"
                                        else [illegible]_Intermittent=1,Cde_Vitesse=Vitesse_Rapide; // Position "Intermittent", high speed
                                        if(Valeur_Analogique>=150)Tempo_Fin=Tempo5;     // According to the position of wheel
                                        else {if(Valeur_Analogique>=140)Tempo_Fin=Tempo4;          // the delay longer or shorter
                                        else {if(Valeur_Analogique>=120)Tempo_Fin=Tempo3;
                                        else {if(Valeur_Analogique>=90)Tempo_Fin=Tempo2;
                                        else {if(Valeur_Analogique==0)Tempo_Fin=Tempo1;}}}}}}
                             }
          // Deal with the state diagram "system"
          //---------------------------------------------
          if(Etat_Arret)      // If the system is in the condition "Stop"
                   {if(I_Autorise_Cycle)                  // If the cycle is authorized
                             {I_Autorise_Cycle=0;
                              Etat_Arret=0,Etat_Rot_Gauche=1;       // Change the system state
                              T_IM_Asservissement.data[2]=Cde_Vitesse; // Turn to left rotation
                              T_IM_Asservissement.data[0]=0x26;     // PWM2DC register address
                              Ecrire_Trame(T_IM_Asservissement);
                              while(Lire_Trame(&Trame_Recue)==0){}; //Wait for the response
                              //if(I_Intermittent)Compteur_Secondes=0,Compteur_Passage_Irq=0;
                             }}
          if(Etat_Rot_Gauche)          // If the system is in the condition "Left Rotation"
                   {if(fcg)            // If it's in the condition of end of "Left Rotation"
                             {Etat_Rot_Gauche=0,Etat_Rot_Droite=1;  // Change the system state
                              T_IM_Asservissement.data[2]=0;        // Stop "Left Rotation"
                              T_IM_Asservissement.data[0]=0x26;     // PWM2DC register address
                              Ecrire_Trame(T_IM_Asservissement);
                              while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                              T_IM_Asservissement.data[2]=Cde_Vitesse; //  Do "Left Rotation"
                              T_IM_Asservissement.data[0]=0x25;     // PWM1DC register address
                              Ecrire_Trame(T_IM_Asservissement);
                              while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                             }}
          if(Etat_Rot_Droite)          // If the system is in the condition "Right Rotation"
                   {if(fcd)            // If it's in the condition of end of "Right Rotation"
```

```
                        {if(Cde_EG_Av_Pos1|Cde_EG_Av_Pos2)
                                {Etat_Rot_Droite=0,Etat_Rot_Gauche=1; // Change the system state
                                 T_IM_Asservissement.data[2]=0;       // Stop "Right Rotation"
                                 T_IM_Asservissement.data[0]=0x25;    // PWM2DC register address
                                 Ecrire_Trame(T_IM_Asservissement);
                                 while(Lire_Trame(&Trame_Recue)==0){};          // Wait for the response
                                 T_IM_Asservissement.data[2]=Cde_Vitesse; // Do "Left Rotation"
                                 T_IM_Asservissement.data[0]=0x26;    // PWM1DC register address
                                 Ecrire_Trame(T_IM_Asservissement);
                                 while(Lire_Trame(&Trame_Recue)==0){};          // Wait for the response
                                }
                         else
                               {Etat_Rot_Droite=0,Etat_Arret=1;       // Change the system state
                                T_IM_Asservissement.data[2]=0;        // Stop "Right Rotation"
                                T_IM_Asservissement.data[0]=0x25;     // PWM1DC register address
                                Ecrire_Trame(T_IM_Asservissement);
                                while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                               }}}
            // Display system status
            //-------------------------
            Cptr_Affichage++;
            if(Cptr_Affichage==200)
                      {Cptr_Affichage=0;
                       gotoxy(1,6);
                       if(I_Intermittent)printf("    Wiper before intermittent position \n");
                       else {if(Cde_EG_Av_Pos1)printf("  Wiper before low speed condition \n");
                       else {if(Cde_EG_Av_Pos2)printf("  Wiper before high speed condition \n");
                       else {printf("  Wiper before stop position \n");}}}
                       printf("  Front wiper washer control: %d\n",Cde_Lave_Glace_Av);
                       gotoxy(1,10);
                       printf("  Order of the back wiper: %d\n",Cde_EG_Ar);
                       printf("  Back wiper washer control: %d\n",Cde_Lave_Glace_Ar);
                      } // End of " Display system status "
            } //  End of the main loop
  } // End of the main function
```

9 EXPERIMENT N°9 : ALL THE STEERING WHEEL COMMAND

9.1 Topic

Purpose :	- Develop a complete real time application, at the same time including binary (on/off)sensors and analog sensors, analog pre-actuators and incorporating a speed variable function.
Specifications :	After a regular time interval, we c[illegible] its status by the module on which is connected the wiper stalk and lamps. According to the state of the wiper stalk, the motor (intermittent, position 1, position 2, etc.) is controlled. According to the state of the lights stalk, we control the different lamps in different optical blocks (indicators, side light, dipped light, head light, stop lights)

Necessary hardware and software :

PC Micro Computer using Windows 95 or [illegible]
Software: Editor -Assembler-Debugger
If programming in C, GNU C / C++ Compiler Ref: EID210101
Processor board 16/32 bit 68332 microcontroller and its software environment
(Editor-Cross Assembler-Debugger) Ref: EID210001
CAN PC/104 Network board in ARCOM SYSTEMS Ref NIC: EID004001

- 1 "servo-system motor" electronic module Ref : EID052001
- The operative block of the windshield wiper system Ref: EID053001
- 2 "8 inputs" electronic modules Ref: EID050001.
- (If necessary) -1 Wiper stalk
- -1 Lights stalk
- 4 modules of 4 power outputs for the 2 front/back lamps Ref : EID051001

USB connection cable, or if not available use RS232 cable, Ref: EGD000003
AC / AC Power source 8V 1A Ref: EGD000001
12V Power source supply for the CAN modules ("energy" network)

Time : 4 hours

9.2 **Solution**

9.2.1 Analysis

The program will run two independent processes:

- The "lights system " controlled by the "lights" stalk with the control of bulbs
- The "wiper system " controlled by the "wiper" stalk.

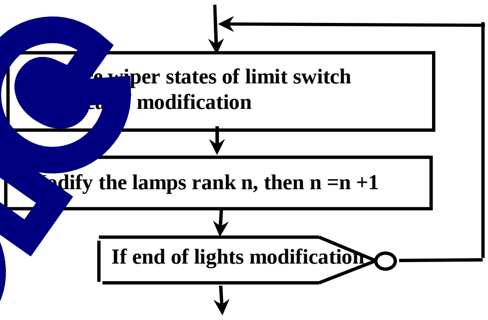
It will also display the status and test results.

Some tasks to be realized are higher priority than others. They can be classified as follows (in descending order of priority):

- Acquisition the "wiper" states of limit switch, in this case it is during the cycle,
- Modification of the status of lights, where a modification is in progress,
- Acquisition of the status of stalks (lights and wipers)
- Change the state of indicators if one is enabled and the associated delay is finished,
- Monitor the status of the different light bulbs (asking for the "status")
- Display the states on the screen.

Remarks:

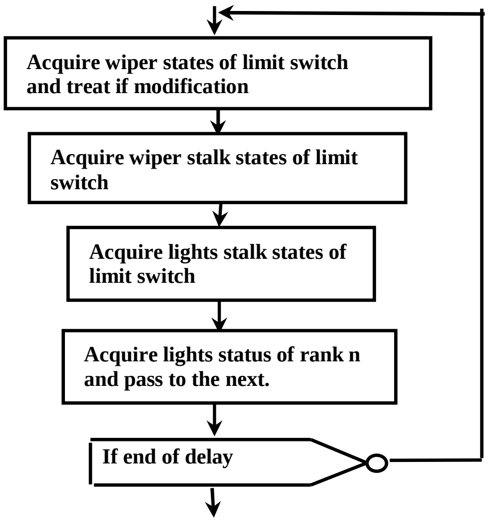
- The modification the lights states is sequentially (one after another) by sending 4 frames (one by optical block). Nothing prevents for acquiring the wiper states of limit switch during every sent frame of the lights modification. So the cycle described above is running 4 times each time when it has detected a modification in the lights stalk status but also in the case where a indicator is on and the associated delay reaches its end.



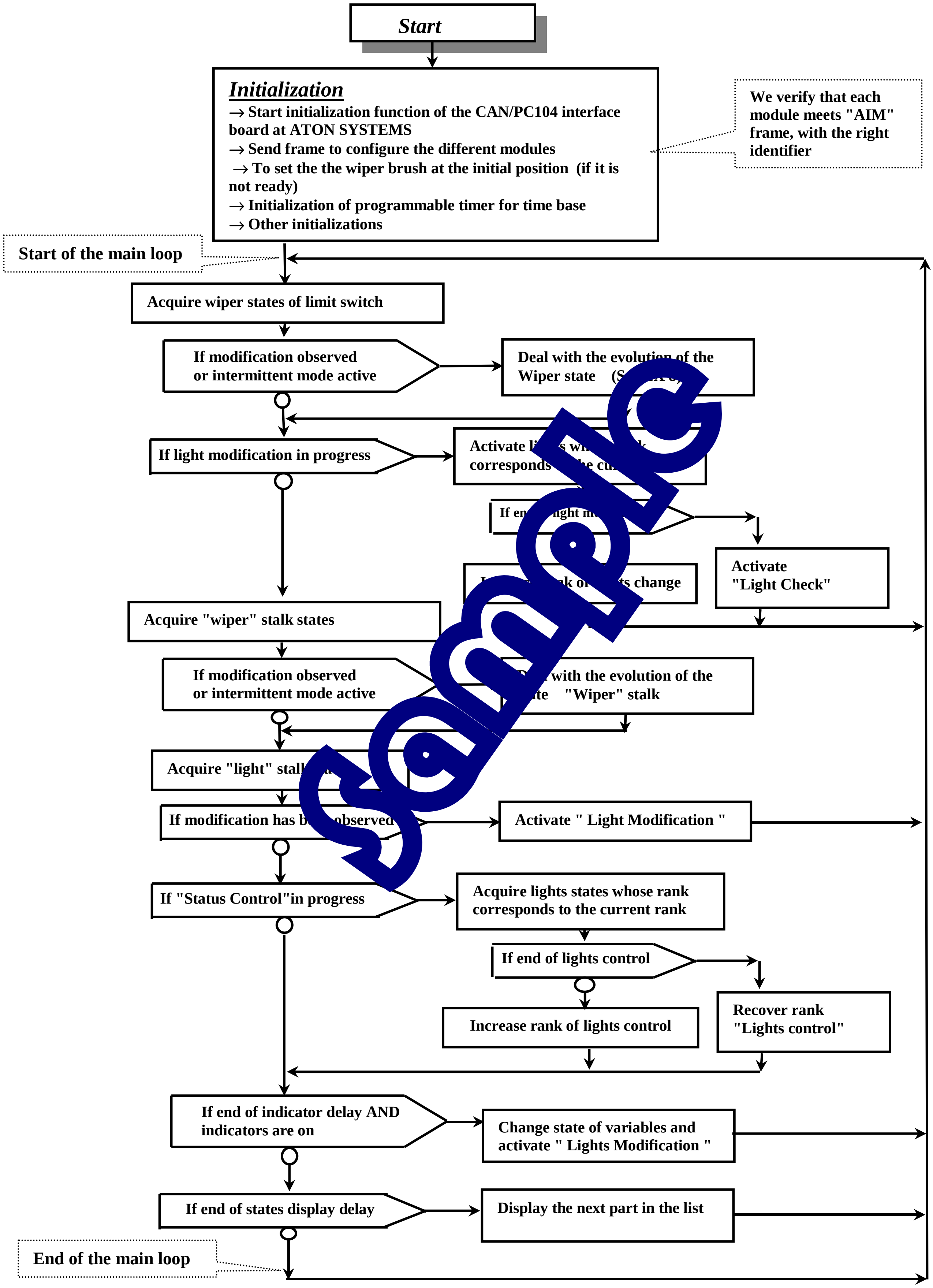
- If we are not in the progress of the lights modification, then the program describes another cycle which includes the acquisition of (lights and wipers) stalk states and the acquisition of optical block status.
We change the block on each pass through the cycle.
Two ends of delays are checked in this [illegible]
-> end indicator delay ([illegible] is active)
-> end display results delay.

If the indicator is active and the end of indicator delay is detected, it leads the previous cycle "Lights Modification" to charge the lamps states.

If end of display results delay is detected, show only one block on the screen, the period of displaying the entire screen takes too long time. Only the display at the bottom of the screen is refreshed each time.



9.2.2 General Flowchart



9.2.3 "C" Program

```
/*************************************************************************************************
          *       Experiment on  EID210 / CAN Network – V.M.D  (Multiplexed Didactic Vehicle)
**************************************************************************************************
*         EXPERIMENT N°9:   ALL THE STEERING WHEEL COMMAND
*-------------------------------------------------------------------------------------------------
*   SPECIFICATIONS :
*  *********************
*  Put EX 4 and EX 8 together
*------------------------------------------------------------------------------------------------
*  File Name :  CAN_VMD_TP9.C
* *****************

**************************************************************************************************/
// Declaration of included files
//***************************
#include <stdio.h>
#include "Structures_Donnees.h"
#include "cpu_reg.h"
#include "eid210_reg.h"
#include "Can_vmd.h"

// -> Function Prototypes
void Envoi_IM_Et_Test(void);
// Declaration of constants
//-----------------------------
//  For the delays
#define Tempo_Clignot 8                 // 8 ms-> 0,8 S
#define Tempo_Affichage 10   // wait for the response 1 ms-> 0,1 S
// For the coding of the status diagram
#define Etat_Attente 0
#define Etat_Modif_Feux 1
#define Etat_Control_Stat 2

//For the wiper
#define Vitesse_Lente 80
#define Vitesse_Rapide 120
#define Tempo1 2    //  unit in second
#define Tempo2 4 // For intermittent mode
#define Tempo3 6
#define Tempo4 8
#define Tempo5 10

// Declaration of variables
//---------------------------
// To dispaly the warning information
char Texte[25];
//For the wiper
unsigned char Valeur_Analogique,Cde_Vitesse,Tempo_Fin;
int Compteur_EG_Secondes,Compteur_EG_Passage_Irq;
// For the diverse indicators (binary variables)
union word_bits Indicateurs;
#define I_Fin_Tempo_Clignot Indicateurs.bit.b2
#define I_Fin_Tempo_Affichage Indicateurs.bit.b3
#define I_Warning Indicateurs.bit.b5
#define I_Clignot_Gauche Indicateurs.bit.b6
#define I_Clignot_Droit Indicateurs.bit.b7
#define I_Message_Pb_Affiche Indicateurs.bit.b8
#define I_Intermittent Indicateurs.bit.b9
#define I_Autorise_Cycle Indicateurs.bit.b10
// For the status diagram of Wiper
union byte_bits Etat_EG;
#define Etat_EG_Arret Etat_EG.bit.b[illegible]
#define Etat_EG_Rot_Gauche Etat_EG[illegible]it.[illegible]
#define Etat_EG_Rot_Droite Etat_E[illegible]it.[illegible]

// For the end of process "Wiper"
union byte_bits FC;
#define Valeur_FC_EG FC.valeur // For [illegible]up [illegible]state of the limit switch
#define fs FC.bit.b7          // For over[illegible]it [illegible]tch
#define fcg FC.bit.b6         // For left [illegible]itch
#define fcd FC.bit.b5         // For right li[illegible]switch
unsigned char Valeur_FC_EG_Mem;

// Declaration of frames
Trame Trame_Recue;
Trame Trame_Envoyee;
Trame T_IM;         // Frame of type "Input Message" to command 4 power outputs module
Trame T_AIM;        // Acknowledgment frame following an IM
Trame T_IRM;        // Frame of type "Information Request Message" to interrogate the lamps' condition
                              // Or stalk

// For comparison of identifiers between Sent Frame <-> Recieved Frame
#define Ident_Trame_Envoyee Trame_Envoyee.ident.extend.identificateur.ident
#define Ident_Trame_Recue 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]

// For the delays
WORD Compteur_Passage,Compteur_dS; // dS -> ms
WORD Valeur_Fin_Tempo_Clignot,Valeur_Fin_Tempo_Affichage,Rang_Affich;
// For the status diagram
unsigned char Etat,Rang_Control_Stat,Rang_Modif_Feux;
// For the memory
unsigned char Valeur_Commodo_Feux_Mem,Valeur_Commodo_EG_Mem;
//  Interruption function "Time Base"
void irq_bt()
// Function runs every 10 mS
{// For all the Lights
 Compteur_Passage++;
 if(Compteur_Passage==10) // 1/10 second has passed
```

```
          {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_Warning)
                              {I_Fin_Tempo_Clignot = 1;
                           Valeur_Fin_Tempo_Clignot = Compteur_dS + Tempo_Clignot;}
                    }
          }
// For the Wiper
if(I_Intermittent) // If intermittent mode is on
          {Compteur_EG_Passage_Irq++;
           if(Compteur_EG_Passage_Irq==100)  // 1/100 second has passed
                    {Compteur_EG_Passage_Irq=0;
                     Compteur_EG_Secondes++;
                     if(Compteur_EG_Secondes>=Tempo_Fin)
                              {Compteur_EG_Secondes=0;
                               I_Autorise_Cycle=1;}
                    }}
} // End of the interrupt function

//==========================
//  MAIN FUNCTION
//==========================
main()
{
// Initializations
//*****************
clsscr();
/* Initialization of SJA1000 of the ATON-Systemes board on PC104 circuit */
Init_Aton_CAN();
//  Definition of frames to enable or reading an optical block
// According to doc SJA1000 and doc MCP25050 pages 22 (function "Write Register") [illegible]IN Address)
// For the command frame  -> IM (Input Message)
T_IM.trame_info.registre=0x00;
T_IM.trame_info.champ.extend=1; // Work in extended mode
T_IM.trame_info.champ.dlc=0x03; // There will be 3 data of 8 bits (3 bytes)
// To configure the modules status of 4 outputs and 4 inputs
T_IM.data[0]=0x1F;  // first data -> "Address" of concerned register -> GPDDR
T_IM.data[1]=0x7F;  // second data -> "Mask" -> see in doc MCP25050
T_IM.data[2]=0xF0;  // third data-> "Value" -> the 4 least significant bits as outputs
Ident_T_IM=Ident_T_IM_FRD;    // This is the identifier of the right back lights for IM
Ident_T_AIM=Ident_T_AIM_FRD;  // This is the identifier of the right back lights for Acknowledgement
strcpy(Texte,"Right Back Light  ");
Envoi_IM_Et_Test();
Ident_T_IM=Ident_T_IM_FRG;    // This is the identifier of the left back lights for IM
Ident_T_AIM=Ident_T_AIM_FRG;  // This is the identifier of the left back lights for Acknowledgement
strcpy(Texte,"Left Back Light   ");
Envoi_IM_Et_Test();
Ident_T_IM=Ident_T_IM_FVG;    // This is the identifier of the left front lights for IM
Ident_T_AIM=Ident_T_AIM_FVG;  // This is the identifier of the left front lights for Acknowledgement
strcpy(Texte,"Left Front Light ");
Envoi_IM_Et_Test();
Ident_T_IM=Ident_T_IM_FVD;    // This is the identifier of the right front lights for IM
Ident_T_AIM=Ident_T_AIM_FVD;  // This is the identifier of the right front lights for Acknowledgement
strcpy(Texte,"Right Front Light  ");
Envoi_IM_Et_Test();
// To configure the modules of 8 Inputs

T_IM.data[2]=0x7F; // third data-> "Value" -> For modules of 8 inputs: GP7 not affected
Ident_T_IM=Ident_T_IM_Commodo_Feux;    // This is the identifier of the lights stalk for IM
Ident_T_AIM=Ident_T_AIM_Commodo_Feux;  // This is the identifier of the lights stalk for Acknowledgement
strcpy(Texte,"Lights Stalk   ");
Envoi_IM_Et_Test();
Ident_T_IM=Ident_T_IM_Commodo_EG;    // This is the identifier of the wiper stalk for IM
Ident_T_AIM=Ident_T_AIM_Commodo_EG;  // This is the identifier of the wiper stalk for Acknowledgement
strcpy(Texte,"Wiper Stalk   ");
Envoi_IM_Et_Test();
// To configure the "Servo-system" modules on which is connected the Wiper Motor
T_IM.data[2]=0xE3;  // Value ->  1 for input and 0 for output
          // GP7=fs Input; GP6=fcg Input; GP5=fcd Input;  GP4=ValidIP Output;
          // GP3=PWM2 Output; GP2=PWM1 Output; GP1=AN1 Input; GP0=AN0 Input;
Ident_T_IM=Ident_T_IM_Asservissement;    // This is the identifier of the "Servo-system" module
Ident_T_AIM=Ident_T_AIM_Asservissement;  // This is the identifier of the "Servo-system" module for Acknowledgement
strcpy(Texte,"Servo-system-module board   ");
Envoi_IM_Et_Test();
// Until arriving here, it shows that all 7 modules have responded!
//----------------------------------------------------------------------
// To configure the analog input (Wheel Position on the wiper stalk)
// To activate the conversion Ana -> Dig
T_IM.data[0]=0x2A;  // ADCON0 register address
T_IM.data[1]=0xF0;  // Mask -> bits 7..4 affected
T_IM.data[2]=0x80;  // Value ->  ADON=1 and "prescaler rate"=1:32
Ident_T_IM=Ident_T_IM_Commodo_EG;    // This is the identifier of the wiper stalk for IM
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To activate from the GP0 to GP7 as input
T_IM.data[0]=0x1F;  // GPDDR register address
T_IM.data[1]=0x7F;  // Mask -> bits 7..0 affected
T_IM.data[2]=0x7F;  // Value ->  8 bits as input
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To define the mode of conversion
T_IM.data[0]=0x2B;  // ADCON1 register address
T_IM.data[1]=0xFF;  // Mask -> all bits are affected
T_IM.data[2]=0x0E;  // Value ->  see doc MCP25050 page 36 (GP0 -> Analog Input)
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To configure the PWM output of the "Servo-system" module
Ident_T_IM=Ident_T_IM_Asservissement;    // This is the identifier of the "Servo-system" module
// To set GP2 output by PWM1
```

```
T_IM.data[0]=0x21;  // T1CON register address
T_IM.data[1]=0xB3;  // Mask -> only bit 7,5,4,1,0 affected
T_IM.data[2]=0x80;  // Value ->  TMR1ON=1; Prescaler1=1
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM1 output signal frequency
T_IM.data[0]=0x23;  // PR1 register address
T_IM.data[1]=0xFF;  // Mask -> all bits are affected
T_IM.data[2]=0xFF;  // Value ->  PR1=255
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set GP3 output by PWM2
T_IM.data[0]=0x22;  // Register T2CON address
T_IM.data[1]=0xB3;  // Mask -> only bit 7,5,4,1,0 affected
T_IM.data[2]=0x80;  // Value ->  TMR2ON=1; Prescaler2=1
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To set PWM2 output signal frequency
T_IM.data[0]=0x24;  // Register PR2 address
T_IM.data[1]=0xFF;  // Mask -> all bits are affected
T_IM.data[2]=0xFF;  // Value->  PR2=255
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM1 cyclic duty to 0
T_IM.data[0]=0x25;  // PWM1DC register address
T_IM.data[1]=0xFF;  // Mask -> all bits are affected
T_IM.data[2]=0;     // Value ->  PWM1DC=0
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To initialize PWM2 cyclic duty to 0
T_IM.data[0]=0x26;  // PWM2DC register address
T_IM.data[1]=0xFF;  // Mask -> all bits are affected
T_IM.data[2]=0;     // Value ->  PWM2DC=0
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// To validate the power circuit
T_IM.data[0]=0x1E;  // GPLAT register address (I/O register)
T_IM.data[1]=0x10;  // Mask -> GP4 (ValidIP) output is affected
T_IM.data[2]=0x10;  // Value ->  ValidIP=1
Ecrire_Trame(T_IM);
do{}while(Lire_Trame(&Trame_Recue)==0); // Wait for the response
// Mask for the future command IM
T_IM.data[1]=0xFF;  // Mask -> all bits are affected

clsscr(); // To clear the screen
// For the query frame ->  IRM  (Information Request Frame)
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;
// To set the wiper brush in the initial position
Ident_T_IRM= Ident_T_IRM1_Asservissement; // The limit switc[illegible]ted [illegible]Servo-system" module
Ecrire_Trame(T_IRM);          // Sending status acquisitio[illegible]ame[illegible] swi[illegible](1 byte in response)
do{}while(Lire_Trame(&Trame_Recue)==0);          //Wait f[illegible] the res[illegible]
Valeur_FC_EG=~Trame_Recue.data[0];     // Recover the st[illegible] o[illegible] lim[illegible]
if(fcd==0)          //If the wiper brush is not on the[illegible] p[illegible] con[illegible]ol it to do the right rotation
          {T_IM.data[0]=0x25;          // PWM1DC reg[illegible]er addres[illegible]
           T_IM.data[2]=Vitesse_Lente;  // Value -> [illegible] s[illegible]
           Ident_T_IM=Ident_T_IM_Asservissement;// I[illegible]il[illegible]rvo-system" module who is affected
           Ecrire_Trame(T_IM);          // Send frame [illegible]moto[illegible]ol
           do{}while(Lire_Trame(&Trame_Recue)==0[illegible]//Wa[illegible]for the response
           do       {Ecrire_Trame(T_IRM);       [illegible]isition frame of the limit switch status
                     do{}while(Lire_Trame(&Tr[illegible]_Recue)=[illegible]          // Wait for the response
                     Valeur_FC_EG=~Trame_Rec[illegible]da[illegible]Recover the state of the limit switch
                     Valeur_FC_EG_Me[illegible]leur[illegible]C_E[illegible]
                    }while(fcd==0)[illegible]
          T_IM.data[2]=0;     // V[illegible]e -[illegible]ro[illegible]o speed
          Ident_T_IM=Ident_T_IM_As[illegible]rvis[illegible]
          Ecrire_Trame(T_IM);
          do{}while(Lire_Trame(&Tra[illegible]ecue)==0)[illegible]          // Wait for the response
          T_IM.data[0]=0x1E;  // For [illegible] out[illegible]t GPLAT Address
          }
// Initialization of Wiper condition var[illegible]
Etat_EG_Arret=1,Etat_EG_Rot_Droite=0,Etat_E[illegible]Gauche=0;
Compteur_EG_Secondes=0,Compteur_EG_Passage_Irq[illegible]0;
Valeur_FC_EG_Mem=0;
Valeur_Commodo_Feux_Mem=0;

// Initialization of condition variable
Etat = Etat_Attente;
// For the condition " Status Control "
Rang_Control_Stat=1;
// For the condittion "Light modify"
Rang_Modif_Feux=1;
// For all indicators
Indicateurs.valeur=0;
// For the display
Rang_Affich=1;

// For the time base and delay
//*****************************************
SetVect(96,&irq_bt);          // load the auto vector
PITR = 0x0048;                // an interrupt every 10 milliseconds
PICR = 0x0760;                //  96 = 60H
// For the delays
Compteur_Passage = 0,Compteur_dS = 0;
Valeur_Fin_Tempo_Clignot = Tempo_Clignot;
Valeur_Fin_Tempo_Affichage = Tempo_Affichage;
//   Display the title
gotoxy(1,2);
 printf("     EXPERIMENT N°9:   ALL THE STEERING WHEEL COMMAND \n");
 printf("    ************************************************* \n");

// Main loop
//*******************
while(1)
 {// Acquire the Wiper condition of limit switch
```

```
Ident_T_IRM=Ident_T_IRM1_Asservissement; // We start by reading the Wiper state of limit switch
Ecrire_Trame(T_IRM);
do{}while(Lire_Trame(&Trame_Recue)==0);          // Wait for the response
if(Ident_Trame_Recue==Ident_T_IRM1_Asservissement) // Test whether the identifier is correct or not
          {Valeur_FC_EG=~Trame_Recue.data[0];}// Recover the state of the limit switch
if((Valeur_FC_EG!=Valeur_FC_EG_Mem)||(I_Autorise_Cycle==1))// If limit switch “modif” contidition
          { Valeur_FC_EG_Mem=Valeur_FC_EG;
           // Deal with the evolution of the state    "Wiper" stalk
           if(Etat_EG_Arret)  // If the system is in the condition "Stop"
                    {if(I_Autorise_Cycle)                   // If a cycle has been authorized
                              {I_Autorise_Cycle=0;
                               Etat_EG_Arret=0,Etat_EG_Rot_Gauche=1; // Change the system state
                               Ident_T_IM=Ident_T_IM_Asservissement;
                               T_IM.data[2]=Cde_Vitesse; // Left Rotation
                               T_IM.data[1]=0xFF;          // Mask
                               T_IM.data[0]=0x26;          // PWM2DC register Address
                               Ecrire_Trame(T_IM);
                               T_IM.data[0]=0x1E;          // Register output address
                               while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                              }}// Finish if meet "Etat_EG_Arret"
          if(Etat_EG_Rot_Gauche)        // If the system is in the condition "Left Rotation"
                    {if(fcg)            // If we reached the end of right rotation
                              {Etat_EG_Rot_Gauche=0,Etat_EG_Rot_Droite=1;      // Change the system state
                               Ident_T_IM=Ident_T_IM_Asservissement;
                               T_IM.data[2]=0;    // Stop left rotation
                               T_IM.data[1]=0xFF;          // Mask
                               T_IM.data[0]=0x26;          // PWM2DC register Address
                               Ecrire_Trame(T_IM);
                               while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                               T_IM.data[2]=Cde_Vitesse; // Order to turn left rotation
                               T_IM.data[0]=0x25;          // PWM1DC register Address
                               Ecrire_Trame(T_IM);
                               T_IM.data[0]=0x1E;          // Registe[illegible]tput [illegible]ess
                               while(Lire_Trame(&Trame_Recue)==0){}; [illegible] the response
                              }}// Finish if meet "Etat_Rot_Gauche"
          if(Etat_EG_Rot_Droite)        // If the system is [illegible]he c[illegible]it[illegible]Rotation"
                    {if(fcd)            // If we reached th[illegible] r[illegible]t r[illegible]io
                       {if(Cde_EG_Av_Pos1|Cde_EG_Av_Pos2)
                              {Etat_EG_Rot_Droite=0,Etat_EG[illegible]Gauc[illegible] Change the system state
                               Ident_T_IM=Ident_T_IM_Asserviss[illegible]t;
                               T_IM.data[2]=0;    // Stop right r[illegible]ion
                               T_IM.data[0]=0x25;          [illegible]gi[illegible] Address
                               Ecrire_Trame(T_IM);
                               while(Lire_Trame(&Trame_R[illegible]ue)[illegible]{}; [illegible]it for the response
                               T_IM.data[2]=Cde_Vitesse [illegible]/ [illegible]to [illegible]rn left rotation
                               T_IM.data[0]=0x26;          /[illegible]2DC[illegible]egister Address
                               Ecrire_Trame(T_IM);
                               T_IM.data[0]=0x1E;          [illegible]/ Out[illegible]register address
                               while(Lire_Trame([illegible]0){};                  // Wait for the response
                       else {Etat_EG_Rot_Droite[illegible]Etat_EG[illegible] Change the system state
                               Ident_T_IM=Ident[illegible]IM[illegible]ervi[illegible]
                               T_IM.data[2]=[illegible]//[illegible]ig[illegible]rota[illegible]on
                               T_IM.data[0][illegible]25;          [illegible]PWM1DC register Address
                               Ecrire_Trame[illegible]IM[illegible]
                               T_IM.data[0][illegible]1E;          [illegible]// Output register address
                               while(Lire_Tr[illegible](&Tr[illegible]ue)==0){};}// Wait for the response
                              }} // Fini[illegible] "E[illegible]_Rot_Droit"
          }   // End of Wiper conditi[illegible]
// Deal with the modification of th[illegible]ights
if(Etat==Etat_Modif_Feux)
          {  switch(Rang_M[illegible]f_Feu[illegible] //[illegible]in[illegible]odule in the list
                              //[illegible] ri[illegible]lights has already been ordered
                               [illegible]se[illegible] : {Ident[illegible]IM=Ident_T_IM_FVD; //Right frnot light
                                                   Valeur_T_IM=Valeur_FVD;
                                                   Rang_Modif_Feux++;
                                                   Ecrire_Trame(T_IM); // Send the frame
                                                   while(Lire_Trame(&Trame_Recue)==0){};} // Wait for the response
                               br[illegible];
                               [illegible]2 [illegible]Ident_T_IM=Ident_T_IM_FRD; //Right back light
                                                   Valeur_T_IM=Valeur_FRD;
                                                   Rang_Modif_Feux++;
                                                   Ecrire_Trame(T_IM); // Send the frame
                                                   while(Lire_Trame(&Trame_Recue)==0){};} // Wait for the response
                               break;
                               case 3 : {Ident_T_IM=Ident_T_IM_FRG; //Left back light
                                                   Valeur_T_IM=Valeur_FRG;
                                                   Rang_Modif_Feux++;
                                                   Ecrire_Trame(T_IM); // Send the frame
                                                   while(Lire_Trame(&Trame_Recue)==0){};} // Wait for the response
                               break;
                               case 4 : {Ident_T_IM=Ident_T_IM_FVG; //Left front light
                                                   Valeur_T_IM=Valeur_FVG;
                                                   Ecrire_Trame(T_IM); // Send the frame
                                                   while(Lire_Trame(&Trame_Recue)==0){};
                                                   Etat=Etat_Control_Stat; // When finish, pass to the control menu
                                                   Rang_Control_Stat=1;}        // of lamps condition
                               break;
                              }  // End "switch" function
          } // End Condition " modification of the lights "
else   // Not in the Condition " modification of the lights "
          {  // Acquire the Wiper stalk condition
             // Prepare a frame to ask for the Wiper Stalk Module condition
             T_IRM.trame_info.champ.dlc=0x08;     // 8 bytes in reponse because of analog
             Ident_T_IRM=Ident_T_IRM8_Commodo_EG; // Wiper Stalk
             Ecrire_Trame(T_IRM); // We send the remote frame through the bus
             T_IRM.trame_info.champ.dlc=0x01;     // 1 bytes in reponse to the next interrogation
             do{}while(Lire_Trame(&Trame_Recue)==0);       //Wait for the response
             if(Ident_Trame_Recue==Ident_T_IRM8_Commodo_EG)
             {Valeur_Commodo_EG=~Trame_Recue.data[1];      // Recover the state of wiper stalk
             Valeur_Analogique=Trame_Recue.data[2];}       // Recover the state of wiper wheel
             // Deal with the modification of the wiper stalk state
if(Cde_EG_Av_Pos2)I_Autorise_Cycle=1,I_Intermittent=0,Cde_Vitesse=Vitesse_Rapide;
           // Start the cycle with high speed, if it’s in position 2 on the stalk
```

```
else {if(Cde_EG_Av_Pos1||Cde_Lave_Glace_Av==1)I_Autorise_Cycle=1,I_Intermittent=0,Cde_Vitesse=Vitesse_Lente;
           // Start the cycle with low speed, if it's in position 1 on the stalk
                    else      {if(Valeur_Analogique>=200)I_Intermittent=0; // "Stop" Position
                               else {I_Intermittent=1,Cde_Vitesse=Vitesse_Lente; // "Intermittent" Position
                               if(Valeur_Analogique>=150)Tempo_Fin=Tempo5;// Depending on the position of the wheel
                               else {if(Valeur_Analogique>=140)Tempo_Fin=Tempo4;// the longer or shorter delay
                               else {if(Valeur_Analogique>=120)Tempo_Fin=Tempo3;
                               else {if(Valeur_Analogique>=90)Tempo_Fin=Tempo2;
                               else {if(Valeur_Analogique==0)Tempo_Fin=Tempo1;}}}}}}}
              // End the function "Deal with the modification of the wiper stalk state"
              // End of acquisition the Wiper stalk condition
          // Acquisition of the conditon"Lecture the state of Lights Stalk "
          //Prepare a frame to ask for the Lights Stalk Module condition
          Ident_T_IRM=Ident_T_IRM_Commodo_Feux; //Module " Lights Stalk "
          Ecrire_Trame(T_IRM);// Send the remote frame through the bus
          do{}while(Lire_Trame(&Trame_Recue)==0);          //Wait for the response
          if(Ident_Trame_Recue==Ident_T_IRM_Commodo_Feux)// Check the identifier
                    {Valeur_Commodo_Feux=~(Trame_Recue.data[0]);}// Recover the state of lights stalk
                     if(Valeur_Commodo_Feux!= Valeur_Commodo_Feux_Mem)
                     // Si on a détecté une modification de l'état commodo
                              {Valeur_Commodo_Feux_Mem = Valeur_Commodo_Feux; // Save in the memory
                               // Pre-difine the status of different bulbs
                               // The defination all in CAN_VMD.h
                               Valeur_FVG=Cde_Nulle,Valeur_FVD=Cde_Nulle;
                               Valeur_FRG=Cde_Nulle,Valeur_FRD=Cde_Nulle;
                               I_Warning=0;
                               I_Clignot_Droit=0;
                               I_Clignot_Gauche=0;
                               if(Cde_Phare) // If control the head light
                                        {Valeur_FVG=Cde_FV_P,Valeur_FVD=Cde_FV_P;  // Front lights
                                         Valeur_FRG=Cde_FR_P,Valeur_FRD=Cde_FR_P;} // Back lights
                               else if(Cde_Code) // If control the dipped light
                                        {Valeur_FVG=Cde_FV_C,Valeur_FVD=Cde_FV_C;  // Front lights
                                         Valeur_FRG=Cde_FR_C,Valeur_FRD=Cde_FR_C;} // Back lights
                               else if(Cde_Veilleuse) // If control the side light
                                        {Valeur_FVG=Cde_FV_V,Valeur_FVD=           // Front lights
                                         Valeur_FRG=Cde_FR_V,Valeur_FR  de_FR      // Back lights
                               if(Cde_Warning)
                                        {Valeur_FVD|=Masque_Clign_AV
                                         Valeur_FRD|=Masque_     AR
                                         Valeur_FVG|=Masque    ig  V;
                                         Valeur_FRG|=Masq    lign_
                                         I_Warning=1;
                                         Valeur_Fin_Tempo_Clig      Comp     _dS + Tempo_Clignot;
                                         I_Fin_Tempo_Clignot         / To     ialize the "tempo_clignot" delay
                               else{
                                if(Cde_Clign_Droit)
                                        {Valeur_FVD|=Mas  e_C   AV
                                         Valeur_FRD|=Ma   e_    AR;
                                         I_Clignot_Droi   ;
                                         Valeur_Fin_Temp  lign      mpteur_dS + Tempo_Clignot;
                                         I_Fin_Tem         = 0;}  To initialize the "tempo_clignot" delay
                                if(Cde_Clign_Gau    )
                                        {Valeur_  G|=Masqu          AV;
                                         Valeur   G|    que_C
                                         I_C       _G       ;
                                         Va    _Fin_Tem      not = Compteur_dS + Tempo_Clignot;
                                         I_   _T    Clig    = 0;}  // To initialize the "tempo_clignot" delay
                                   }
                               if(Cde_Klaxon
                                            G|=M   ue_Klaxon;
                                         Va          sque_Klaxon;}
                               if(C   Stop)
                                           ur_F    Masque_Stop;
                                                 |=Masque_Stop;}
                                  ta   tat        x;
                                   ng   dif_Feux
                                                  t the end of modification of the stalk state
                    }  //  ND If           in "Modified lights" condition
if(Etat==Etat_Control_Sta   / Control    e status of the outputs Lights
          {// Control the b      tes
           switch(Rang_Control_S   ) //   ollowing Module in the list
                    {case 1 :          // Control the Right Front Lights
                                         Ident_T_IRM=Ident_T_IRM_FVD;
                                         Ecrire_Trame(T_IRM); // Send frame
                                         while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                                         Valeur_Status_FVD=Trame_Recue.data[0]; // Recover the state of Right Front Lights
                                         Rang_Control_Stat++;}
                    break;
                    case 2 :  {// Control the Right Back Lights
                                         Ident_T_IRM=Ident_T_IRM_FRD;
                                         Ecrire_Trame(T_IRM); // Send frame
                                         while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                                         Valeur_Status_FRD=Trame_Recue.data[0]; // Recover the state of Right Back Lights
                                         Rang_Control_Stat++;}
                    break;
                    case 3 :  {// Control the Left Back Lights
                                         Ident_T_IRM=Ident_T_IRM_FRG;
                                         Ecrire_Trame(T_IRM); // Send frame
                                         while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                                         Valeur_Status_FRG=Trame_Recue.data[0];
                                         Rang_Control_Stat++;}
                    break;
                    case 4 :  {// Control the Left Front Lights
                                         Ident_T_IRM=Ident_T_IRM_FVG;
                                         Ecrire_Trame(T_IRM); // Send frame
                                         while(Lire_Trame(&Trame_Recue)==0){}; // Wait for the response
                                         Valeur_Status_FVG=Trame_Recue.data[0];
                                         Rang_Control_Stat=1;} // Return to the beginning!
                    break;
                    }  // End "switch"
          }// End if meet " status control "
// If indicator is on and meet the end of the indicator delay
if((I_Clignot_Gauche||I_Clignot_Droit||I_Warning)&&(I_Fin_Tempo_Clignot))
          {I_Fin_Tempo_Clignot=0;
           if(I_Warning)
                    {// Switch left indicator bulbs on
```

```
                        Valeur_FVG^=Masque_Clign_AV;
                        Valeur_FRG^=Masque_Clign_AR;
                        Valeur_FVD^=Masque_Clign_AV;
                        Valeur_FRD^=Masque_Clign_AR;
                        Etat=Etat_Modif_Feux;
                        Rang_Modif_Feux=1;}
              if(I_Clignot_Gauche)
                       {// Switch right indicator bulbs on
                        Valeur_FVG^=Masque_Clign_AV;
                        Valeur_FRG^=Masque_Clign_AR;
                        Etat=Etat_Modif_Feux;
                        Rang_Modif_Feux=1;}
              if(I_Clignot_Droit)
                       {// Switch right indicator bulbs on
                        Valeur_FVD^=Masque_Clign_AV;
                        Valeur_FRD^=Masque_Clign_AR;
                        Etat=Etat_Modif_Feux;
                        Rang_Modif_Feux=1;}
             }  // End "bliker" function
   if(I_Fin_Tempo_Affichage)
             {I_Fin_Tempo_Affichage=0;

              switch(Rang_Affich)
              {case 1: // Diagnostic result : Left Front Lights
                       {gotoxy(1,4),printf(" Left Front Optical Block:     \n");
                       if(Veilleuse_FVG==1 && S_Veilleuse_FVG==0)
                                 {gotoxy(1,5),printf("  !! Problem on Left Front Side lights  \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("  !! Problem on Left Front Dipped light  \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("  !! Problem on Left Front Head light   \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("  !! Problem on Left Front Indicator   \n");}
                       if(Clignot_FVG==0 && S_Clignot_FVG==1)
                                 {gotoxy(1,8),printf("                                           \n");}
                       Rang_Affich++;}
                       break;
             case 2:   // Diagnostic result : Right Front Lights
                       {gotoxy(1,9),printf(" Right Front Optical Block:\n");
                       if(Veilleuse_FVD==1 && S_Veilleuse_FVD==0)
                                 {gotoxy(1,10),printf("  !! Problem on Right Front Side lights  \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("  !! Problem on Right Front Dipped light  \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("  !! Problem on Right Front Head light   \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("  !! Problem on Right Front Indicator   \n");}
                       if(Clignot_FVD==0 && S_Clignot_FVD==1)
                                 {gotoxy(1,13),printf("                                        \n");}
                       Rang_Affich++;}
                       break;
             case 3:   // Diagnostic result : Right Back Lights
                       {gotoxy(1,20),printf(" Right Back Lights:\n");
                       if(Veilleuse_FRD==1 && S_Veilleuse_FRD==0)
                                 {gotoxy(1,21),printf("  !! Problem on Right Back Side lights   \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("  !! Problem on Right Back Indicator   \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("  !! Problem on Right Back Stop light    \n");}
                       if(Stop_FRD==0 && S_Stop_FRD==1)
                                 {gotoxy(1,23),printf("                                          \n");}
                       Rang_Affich++;}
                       break;
             case 4:   // Diagnostic result : Left Back Lights
                       {gotoxy(1,16),printf(" Left Back Lights:\n");
                       if(Veilleuse_FRG==1 && S_Veilleuse_FRG==0)
                                 {gotoxy(1,17),printf("  !! Problem on Left Back Side lights   \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("  !! Problem on Left Back Indicator  \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("  !! Problem on Left Back Stop light   \n");}
                       if(Stop_FRG==0 && S_Stop_FRG==1)
                                 {gotoxy(1,19),printf("                                       \n");}
                       Rang_Affich++;}
                       break;
             case 5:   // For the stalk state
                       {gotoxy(4,24);
                       printf(" States of different inputs imposed by the Lights stalk:\n");
printf(" Pilotlights=%d , Dipped lights=%d , Head light=%d , Left indicator=%d  \n",Cde_Veilleuse,Cde_Code,Cde_Phare,Cde_Clign_Gauche);
printf(" Horn=%d ,    Stop light=%d ,    Right indicator= %d\n",Cde_Klaxon,Cde_Stop,Cde_Clign_Droit);
printf(" Wiper Stalk State = %x ;   Wheel = %d \n",Valeur_Commodo_EG,Valeur_Analogique);
                       Rang_Affich=1;}
             } // End "switch"
```

```
                    }   // End if it's the end of the dipaly delay
          }// End of the main loop
} // End of the main function

// Function "Send frame and test if the target module replies
void Envoi_IM_Et_Test(void)
{int Cptr_TimeOut,Temp;
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Send a frame through the CAN network
           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 if the identifier is correct
           if(Cptr_TimeOut==200)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response to the command frame through %s \n",Texte);
                               printf("  Check the existance of the module and whether the supply 12V is OK \n");}
                    for(Temp=0;Temp<100000;Temp++);} //  Wait for a moment!
    }while(Cptr_TimeOut==200);
}
```

```
// Function "Send frame and test if the target module replies
void Envoi_IM_Et_Test(void)
{int Cptr_TimeOut,Temp;
I_Message_Pb_Affiche=0;
do {Ecrire_Trame(T_IM);// Send a frame through the CAN network
           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 if the identifier is correct
           if(Cptr_TimeOut==200)
                    {if(I_Message_Pb_Affiche==0)
                              {I_Message_Pb_Affiche=1;
                               gotoxy(2,10);
                               printf(" No response to the command frame throu[illegible]%S[illegible]xte);
                               printf("  Check the existance of the module and [illegible]ply 12V is OK \n");}
                    for(Temp=0;Temp<100000;Temp++);} // Wait for a mom[illegible]
    }while(Cptr_TimeOut==200);

}
```

10 ANNEX

10.1 Definition File Only For CAN_VMD System

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

//  Data structures for CAN VMD application
//  File Name: CAN_VMD.h

//*****************************************************
#ifndef _VMD_H
#define _VMD_H
// Message Form
typedef struct {
    /* number of bytes to send, -1 if it is Remote Frame */
    int dlc;
    unsigned char id1;  /* 8 most significant bits of ID. */
    unsigned char id2;  /* 3 least significant bits of ID. */
    unsigned char data[8];
} Message;
//  For identifier in standard mode
typedef union
{struct   {unsigned short ident:11;
           unsigned short rtr:1;
           unsigned short nul:4;
          } identifier;
 struct   {unsigned char ident1;
           unsigned char ident2;
          } register;
 unsigned short value;
} ident_standard;
//  For identifier in extended mode
typedef union
{struct   {unsigned long ident:29;
           unsigned long rtr:1;
           unsigned long x:2;
          } identifier;
 struct   {unsigned char ident1;
           unsigned char ident2;
           unsigned char ident3;
           unsigned char ident4;
          } register;
 unsigned long value;
} ident_extend;
// For information frame SJA1000 register
typedef union
          {struct   {unsigned char extend:1;
                     unsigned char rtr:1;
                     unsigned char nul:2;
                     unsigned char dlc:4;
                    } field;
           unsigned char register;
          } tr_info;

// For frame circulating through CAN Network
typedef struct
{
tr_info trame_info;
union     {ident_standard standard
           ident_extend   extend;
          } ident;
unsigned char data[8];
} Frame;
 typedef union
{struct   {unsigned char GP7:1;
           unsigned char GP6:1;
           unsigned char GP5:1;
           unsigned char GP4:1;
           unsigned char GP3:1;
           unsigned char GP2:1;
           unsigned char GP1:1;
           unsigned char GP0:1;
          }bit;
unsigned char value;
}Port_8ES;
Port_8ES Etat_Commodo_Feux;
#define Value_Commodo_Feux Etat_Commodo_Feux.value
#define Cde_Veilleuse Etat_Commodo_Feux.bit.GP0
#define Cde_Warning Etat_Commodo_Feux.bit.GP1
#define Cde_Phare Etat_Commodo_Feux.bit.GP2
#define Cde_Code Etat_Commodo_Feux.bit.GP3
#define Cde_Clign_Gauche 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
// For lights control
#define Cde_Nulle 0x00
#define Cde_FV_V 0x01 // Front Light of Pilotlight
#define Cde_FR_V 0x01 // Back Light of Pilotlight
#define Cde_FV_C 0x03 // Front Light of Dipped light
#define Cde_FR_C 0x01 // Back Light of Dipped light
#define Cde_FV_P 0x05 // Front Light of Head light
#define Cde_FR_P 0x01 // Back Light of Head light
#define Mask_Clign_AV 0x08 // Front Indicator
#define Mask_Clign_AR 0x04 // Back Indicator
#define Mask_Klaxon 0x08 // Horn
#define Mask_Stop 0x02 // Stop Light
```

```
//  Variables to command and control lights
//---------------------------------------------------
//  For images of the affected control variables for different lights
union byte_bits Image_FVG,Image_FVD,Image_FRD,Image_FRG;
// Left Front Lights
#define Value_FVG Image_FVG.value       // For an access to whole port
#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
// Right Front Lights
#define Value_FVD Image_FVD.value       // For an access to whole port
#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
// Right Back Lights
#define Value_FRD Image_FRD.value       // For an access to whole port
#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
// Left Back Lights
#define Value_FRG Image_FRG.value       // For an access to whole port
#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
// For the variable images "Status" of power outputs
union byte_bits Image_Stat_FVG,Image_Stat_FVD,Image_Stat_FRD,Image_Stat_FRG;
// For the image "Status" Left Front Lights
#define Value_Status_FVG Image_Stat_FVG.value // For an access to whole port
#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
// For the image "Status" Right Front Lights
#define Value_Status_FVD Image_Stat_FVD.value // For an access to whole port
#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
// For the image "Status" Right Back Lights
#define Value_Status_FRD Image_Stat_FRD.value // For an access to whole port
#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
// For the image "Status"     Left Back Lights
#define Value_Status_FRG Image_Stat_FRG.value // For an access to whole port
#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


//  Declaration of identifiers for different modules through the frames
//----------------------------------------------------------------------------
// Left Front Lights
#define Ident_T_IRM_FVG 0x0E041E07      // Left Front Lights in interrogation (Information Request Message)
#define Ident_T_IM_FVG 0x0E080000       // Left Front Lights in command (Input Message)
#define Ident_T_AIM_FVG 0x0E200000          // Left Front Lights in Acknowledgement
// Right Front Lights
#define Ident_T_IRM_FVD 0x0E841E07      // Right Front Lights in interrogation (Information Request Message)
#define Ident_T_IM_FVD 0x0E880000       // Right Front Lights in commande (Input Message)
#define Ident_T_AIM_FVD 0x0EA00000      // Right Front Lights in Acknowledgement
// Left Back Lights
#define Ident_T_IRM_FRG 0x0F041E07      // Left Back Lights in interrogation (Information Request Message)
#define Ident_T_IM_FRG 0x0F080000       // Left Back Lights in command (Input Message)
#define Ident_T_AIM_FRG 0x0F200000      // Left Back Lights in Acknowledgement
// Right Back Lights
#define Ident_T_IRM_FRD 0x0F841E07      // Right Back Lights in interrogation (Information Request Message)
#define Ident_T_IM_FRD 0x0F880000       // Right Back Lights in command (Input Message)
#define Ident_T_AIM_FRD 0x0FA00000      // Right Back Lights in Acknowledgement
// Lights Stalk
#define Ident_T_IM_Commodo_Feux 0x05080000 // Lights Stalk in command (Information Message)
#define Ident_T_AIM_Commodo_Feux 0x05200000 // Lights Stalk: Acknowledgement for IM
#define Ident_T_IRM_Commodo_Feux 0x05041E07 // Lights Stalk in interrogation (Information Request Message)
// Wiper Stalk
#define Ident_T_IM_Commodo_EG 0x05880000 // Wiper Stalk in command (Information  Message)
#define Ident_T_AIM_Commodo_EG 0x05A00000 // Wiper Stalk in command (Information  Message)
#define Ident_T_IRM1_Commodo_EG 0x05841E07 // Wiper Stalk in interrogation:  1 byte asked
#define Ident_T_IRM8_Commodo_EG 0x05840000 // Wiper Stalk in interrogation:  8 bytes asked
#define Ident_T_OB_Commodo_EG 0x05900000 // Wiper Stalk "On Bus" (by test manager)
// Servo system
#define Ident_T_IM_Asservissement 0x00880000     // " Servo system " in the command (Input Message)
#define Ident_T_IRM1_Asservissement 0x00841E07   // " Servo system " in interrogation: 1 byte in the response
#define Ident_T_IRM8_Asservissement 0x00840000   // " Servo system " in interrogation: 8 bytes in the response
#define Ident_T_AIM_Asservissement 0x00A00000    // " Servo system " in Acknowledgement
#define Ident_T_OB_Asservissement 0x00900000     // " Servo system " On Bus (by test manager)
// Wiper stalk Command
Port_8ES Commodo_EG;
#define Etat_Commodo_EG Commodo_EG.value
#define Value_Commodo_EG Commodo_EG.value
#define Cde_EG_Av_Int Commodo_EG.bit.GP0         // Front Wiper control before intermittent position
#define Input1 Commodo_EG.bit.GP1
#define Input2 Commodo_EG.bit.GP2
#define Cde_EG_Av_Pos1 Commodo_EG.bit.GP3 // Front Wiper control before in Position 1
#define Cde_EG_Av_Pos2 Commodo_EG.bit.GP4 // Front Wiper control before in Position 2
#define Cde_EG_Ar Commodo_EG.bit.GP5    // Back Wiper control
#define Cde_Lave_Glace_Ar Commodo_EG.bit.GP6
#define Cde_Lave_Glace_Av Commodo_EG.bit.GP7
#endif
```

10.2 Definition File Only For The ATON Board

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

//  Definition file for "ATON_CAN" board CAN controller
//  File Name:  Aton_CAN.h
//*************************************************************
#ifndef _PELICAN_H
#define _PELICAN_H
#define SJA         0xB30280                               /* SJA Board address         */
#define MODE        *(unsigned char *)  (SJA)              /* Control Register */
#define COMMAND     *(unsigned char *)  (SJA+0x01)         /* Command Register */
#define STATUS      *(unsigned char *)  (SJA+0x02)         /* Status Register */
#define INTERRUPT             *(unsigned char *)  (SJA+0x03)         /* Interruption Register  */
#define INTERRUPT_ENABLE      *(unsigned char *)  (SJA+0x04)
#define BUS_TIMING_0          *(unsigned char *)  (SJA+0x06)         /* Bus timing 0 Register      */
#define BUS_TIMING_1          *(unsigned char *)  (SJA+0x07)         /* Bus timing 1 Register      */
#define OUTPUT_CONTROL        *(unsigned char *)  (SJA+0x08)         /* Output control Register    */
#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)

/* SIMPLE FRAME Mode */
#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)   /* Message beginning Address */
#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)   /* Message beginning Address */

/* EXTENDED Mode */
#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)   /* Message beginning Address */

#define TX_ID_1_E           *(unsigned char *) (SJA+0x11)
#define TX_ID_2_E           *(unsigned char *) (SJA+0x12)
#define TX_ID_3_E           *(unsigned char *) (SJA+0x13)
#define TX_ID_4_E           *(unsigned char *) (SJA+0x14)
#define TX_DATA_E           (unsigned char *) (SJA+0x15)   /* Message beginning Address */

/* The two Modes */
#define RX_MESSAGE_COUNTER          *(unsigned char *) (SJA+0x1D)
#define RX_BUFFER_START_ADDRESS     *(unsigned char *) (SJA+0x[illegible])
#define CLOCK_DIVIDER               *(unsigned char *) (SJA+0x[illegible])

/* Acceptable code and mask mode recover only */
#define ACCEPT_CODE0                    *(unsigned char *)  (SJA+0x[illegible])
#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 of bit registers ***/
/* Register STATUS */
#define BUS_STATUS                  0x[illegible]
#define ERROR_STATUS                0x40
#define TRANSMIT_STATUS             0x20
#define RECEIVE_STATUS              0x10
#define TRANSMISSION_COMPLETE       0x08
#define TRANSMIT_BUFFER_STATUS      0x04
#define DATA_OVERRUN_STATUS         0x02
#define RECEIVE_BUFFER_STATUS       0x01

// *************** TYPES ********************
// ***************************************************************
// *                           DECLARATIONS
// *   Prototypes of the specific CAN - SJA1000 functions
// ****************************************************************
/** Initialization of SJA1000 in PeliCAN mode. */
void init_sja1000_peli_acc (char acc_code0, char acc_code1, char acc_code2, char acc_code3,
                            char acc_mask0, char acc_mask1, char acc_mask2, char acc_mask3);
/** Initialization of SJA1000 in PeliCAN mode. */
void init_sja1000_peli ();
Trame Receive_Trame();
/** Send a frame */
void send_trame_peli (Message mes);
/** Receive a message. */
Message receive_trame_peli ();
/** Display the regsitres on PeliCAN */
void print_reg_peli ();
/* Display the state of STATUS REGISTER */
void show_status ();
/* Display the passed message as parameters on the line */
void print_little (Message mes);
// Initialization of the board CAN ATON
void Init_Aton_CAN();
char Ecrire_Trame(Trame message);
char Lire_Trame(Trame *message_recu);
void Affiche_Trame(Trame trame);
```