



Cégep Limoilou

ÉLECTRONIQUE PROGRAMMABLE ET ROBOTIQUE

247-6[1-2-3-4]7-LI

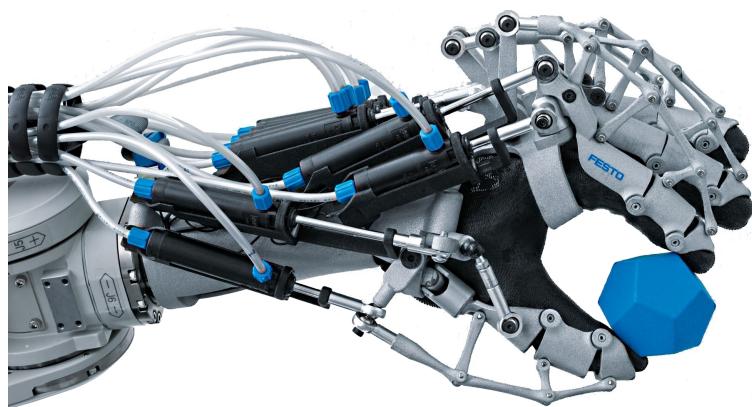
Projet de 5^e session

Étudiants :

Vincent Chouinard
Hicham Safoine
Gabriel Fortin-Bélanger
Louis-Nomand Ang-Houle

Professeurs :

Ali Tadli
Alain Champagne
Stéphane Deschênes
Étienne Tremblay



L'usine à gaz, et le gaz, c'est de l'air !

18 novembre 2014

Table des matières

1 Présentation du projet	4
1.1 Explication du projet	4
1.2 Schéma bloc du système	4
1.2.1 Bloc 1	4
1.2.2 Bloc 2	4
1.2.3 Bloc 3	4
1.2.4 Bloc 4	4
1.3 Liste des logiciels	4
1.4 Liste des trames	6
2 Le matériel	10
2.1 Bloc 1	10
2.2 Bloc 2	10
2.3 Bloc 3	10
2.4 Bloc 4	10
2.5 Explication des types de liens	10
2.5.1 RS232	10
2.5.2 Xbee	10
2.6 Explication des trames	10
2.6.1 RS-232	10
2.6.2 CAN	10
2.6.3 XBEE	10
2.7 Liste des pièces	10
2.7.1 Liens web	11
2.7.2 Datasheets	11
3 Interface PC	12
3.1 Structure du programme	13
3.2 Explication des trames	13
3.3 Ordre de gestion des tâches	13
4 Logiciel du SOC8200	14
4.1 Description du programme	14
4.2 Schéma bloc	14
4.2.1 Du code	14
4.2.2 Du script shell	14
4.3 Gestion des processus et du temps de CPU	14
4.4 Format et récupération des logs	14
4.5 Liste des tests et logiciels	14
5 Logiciel de la station 1 et 2 et du bolide	15
5.1 La station no.1	15
5.2 La station no.2	15
5.3 Le bolide	15
5.4 Procédure de compilation sur IAR	15
5.5 Procédure de vérification	15

6 Logiciel du module PIC18F258	15
6.1 Description du fonctionnement du programme	15
6.2 Procédure de compilation sur MPLAB	15
6.3 Procédure de vérification	15
7 Calculs	15
7.1 Calcul du pas de conversion de la pile	15
7.2 Calcul du baudrate	15
8 Conclusion	17
8.1 Ce que le projet m'a apporté	17
8.1.1 Vincent Chouinard	17
8.1.2 Hicham Safoine	17
8.1.3 Gabriel Fortin-Bélanger	17
8.1.4 Louis-Norman Ang-Houle	17
8.2 Difficultés et corrections	17
8.2.1 Vincent Chouinard	17
8.2.2 Hicham Safoine	17
8.2.3 Gabriel Fortin-Bélanger	17
8.2.4 Louis-Norman Ang-Houle	17
8.3 Ce que j'ai aimé ou pas	17
8.3.1 Vincent Chouinard	17
8.3.2 Hicham Safoine	17
8.3.3 Gabriel Fortin-Bélanger	17
8.3.4 Louis-Norman Ang-Houle	17

Table des figures

1 Programme de contrôle principal	12
2 Options CAN avancées	12
3 Historique des actions	13

Liste des tableaux

1 Index des identifiants matériel CAN	6
2 Index des trames CAN	6
3 Index des communications CAN	6
4 Informations sur le bus I ₂ C du bolide	15

1 Présentation du projet

Le projet de la cinquième session consiste à réaliser

- ⇒ Le Bolide
- ⇒ Carte Dallas DS89C450
- ⇒ Carte uPSD 3254A
- ⇒ SOC8200
- ⇒ Table FESTO
- ⇒ Carte PIC16F88
- ⇒ Carte d'extension I₂C
- ⇒ Carte d'extension SPI
- ⇒ Une pile de 10.8 volts
- ⇒ Quatre moteurs et autant de pneus

1.1 Explication du projet

1.2 Schéma bloc du système

1.2.1 Bloc 1

Le bloc 1 est composé d'un ordinateur muni d'une carte d'extension PCI vers bus CAN. Son rôle est de contrôler et diriger toute l'opération et de veiller au bon fonctionnement de chaque composante à l'aide d'une application en Csharp. Le bloc 1 est le cerveau de l'usine.

1.2.2 Bloc 2

Le bloc 2 est composé d'un système embarqué Linux basé sur le SOC8200. Son rôle principal est d'agir comme sniffeur d'information et d'afficher sur son écran toutes les données qui transitent sur le bus CAN. Toutefois, ce dernier est en mesure de détecter une défaillance du PC via un gestionnaire de HeartBeat et de prendre la relève en tant que cerveau de l'opération. Le SOC8200 agit comme vice-président du bus CAN.

1.2.3 Bloc 3

1.2.4 Bloc 4

1.3 Liste des logiciels

Terminaux

- UART Master 1.0.3
- Serializ3r 1.0.2
- TerraTerm
- Putty
- GTKterm 0.99.7-rc1

Gestion du projet

- xTerminator
- CAPS
- tinyBootloader
- MS Project 2012
- Git Hub

Compilateurs et IDE

- Visual Studio 2013
- Visual Studio 2010
- IAR 8.20
- MPLAB

Éditeur de texte

- Notepad++
 - gedit
 - medit 1.2.0
 - Schémas électriques**
 - OrCAD 16.2
 - Système d'exploitation**
 - Windows 7 SP1
 - Windows 8.1
 - Windows XP SP3
 - Fedora 20
 - CentOS
 - Lubuntu 14.10
 - Autres**
 - VMWare Workstation 10
 - TeXmaker 4.3
 - Dukto R6
 - Dia
 - Festo configuration tool
-

1.4 Liste des trames

TABLE 1 – Index des identifiants matériel CAN

Device	ID matériel
Ordinateur	000
SOC8200	001
Station 1	002
Station 2	003
Station 3	004
Véhicule	005

TABLE 2 – Index des trames CAN

Fonctionnalité	Composante	Données	TimeStamp
Démarre le véhicule	0x00	0x00	TimeStamp
Arrête le véhicule	0x00	0x01	TimeStamp
Le véhicule est arrêté	0x01	0x00	TimeStamp
Le véhicule est en marche	0x01	0x01	TimeStamp
Le véhicule est hors circuit	0x01	0x02	TimeStamp
Vitesse (0-100)	0x02	0x00 à 0x64	TimeStamp
Battre	0x03	0x00 à 0x64	TimeStamp
Couleur du bloc	0x04	0x00 à 0x02	TimeStamp
Poids du bloc	0x05	0x00 à 0x64	TimeStamp
Envoyer l'heure	0x06	à déterminer	TimeStamp
No. de la station	0x07	0x00 à 0x02	TimeStamp
Demande de l'historique	0xC0	0x00	TimeStamp
Direction horaire et antihoraire	0x08	0x00 à 0x01	TimeStamp

TABLE 3 – Index des communications CAN

Émetteur	Action	ID receveur	Donnée envoyée	TimeStamp	Récepteur	Erreur
Ordinateur	Démarrer le véhicule	004	00 00	TimeStamp	Véhicule	F1
Ordinateur	Arrêter le véhicule	004	00 01	TimeStamp	Véhicule	F2
Véhicule	Dit : je suis arrêté	000	01 00	TimeStamp	Ordinateur	F3
Véhicule	Dit : j'avance	000	01 01	TimeStamp	Ordinateur	F4
Véhicule	Dit : je suis hors circuit	000	01 02	TimeStamp	Ordinateur	F5
Véhicule	Dit sa vitesse	000	02 [00 à 64]	TimeStamp	Ordinateur	F6
Véhicule	Dit le niveau de sa batterie	000	03 [00 à 64]	TimeStamp	Ordinateur	F7
Station 1	Dit bloc = métal	000	04 00	TimeStamp	Ordinateur	F8
Station 1	Dit bloc = orange	000	04 01	TimeStamp	Ordinateur	F9
Station 1	Dit bloc = noir	000	04 02	TimeStamp	Ordinateur	FA
Station 1	Dit le poid du bloc	000	05 [00 à 64]	TimeStamp	Ordinateur	FB
Voiture	Dit qu'elle est à la station 1	000	07 00	TimeStamp	Ordinateur	FC
Voiture	Dit qu'elle est à la station 2	000	07 01	TimeStamp	Ordinateur	FD
Ordinateur	Envoie l'heure	003	06 à déterminer	TimeStamp	Station 1	FE
Ordinateur	Demande le LOG	001	C0 00	TimeStamp	SOC8200	E0
Ordinateur	Exige Horaire	004	08 00	TimeStamp	Véhicule	E1
Ordinateur	Exige Antihoraire	004	08 01	TimeStamp	Véhicule	E2

Note : Il faut définir les TimeStamps et la checkSUM

Note : La station no.1 relaie les données entre l'ordinateur et la station no.3 (pesage), entre l'ordinateur et la station no.2 (table Festo) via xbee et entre la voiture et le PC via Xbee.

Note : FF, c'est la checkSUM, mais elle n'a pas encore été faite

Note : Il faut ajouter le TimeStamp

```
CAN.SendMCP("0100FF"); // Arrêté  
CAN.SendMCP("0101FF"); // En marche  
CAN.SendMCP("0102FF"); // Hors circuit  
CAN.SendMCP("0264FF"); // Vitesse maximale  
CAN.SendMCP("0364FF"); // Batterie pleine  
CAN.SendMCP("0400FF"); // Bloc métallique  
CAN.SendMCP("0401FF"); // Bloc noire  
CAN.SendMCP("0402FF"); // Bloc orange  
CAN.SendMCP("050064"); // Le bloc est lourd  
CAN.SendMCP("0700FF"); // Rendu à la station 1  
CAN.SendMCP("0701FF"); // Rendu à la station 2  
CAN.SendMCP("0702FF"); // Rendu à la station 3
```

Comment ajouter une entrée à l'historique

```
Historique.AppendText("\r\n");
Historique.AppendText("\r\n");
Historique.AppendText(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
Historique.AppendText("\r\n");
Historique.AppendText("Insérer messge ici"); // Enregistre le log de la connexion
```

Comment envoyer une trame sécurisée par try catch

```
try
{
    TPCANMsg CANMsg;
    TPCANStatus stsResult;

    CANMsg = new TPCANMsg();
    CANMsg.DATA = new byte[8];

    CANMsg.ID = 006; // 006 c'est pour faire des tests. Mettre 004 pour la version finale
    CANMsg.LEN = 4; // Note: l'index commence à zero, donc 3 = 4
    CANMsg.MSGTYPE = TPCANMessageType.PCAN_MESSAGE_STANDARD;

    for (int i = 0; i < CANMsg.LEN; i++)
    {
        if (i == 0) { CANMsg.DATA[0] = 00; } // Note: Ce programme en C# envoie en base 10
        if (i == 1) { CANMsg.DATA[1] = 00; } // 0x30 (en ascii) = caractère '0'
        if (i == 2) { CANMsg.DATA[2] = 00; } // 0 (en base 10) = caractère ascii '0' (0x30)
        if (i == 3) { CANMsg.DATA[3] = 00; }
        if (i == 4) { CANMsg.DATA[4] = 00; }
        if (i == 5) { CANMsg.DATA[5] = 00; }
        if (i == 6) { CANMsg.DATA[6] = 00; }
        if (i == 7) { CANMsg.DATA[7] = 00; }
    }

    stsResult = PCANBasic.Write(m_PcanHandle, ref CANMsg);

    if (stsResult == TPCANStatus.PCAN_ERROR_OK) // Si l'envoie est réussi
    {
        Historique.AppendText("\r\n");
        Historique.AppendText("\r\n");
        Historique.AppendText(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
        Historique.AppendText("\r\n");
        Historique.AppendText("XYZ command successfully sent");
    }
    else // Si l'envoi a échoué
    {
        Historique.AppendText("\r\n");
        Historique.AppendText("\r\n");
        Historique.AppendText(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
        Historique.AppendText("\r\n");
        Historique.AppendText("Error sending XYZ command");
    }
}
catch // En cas d'échec d'envoi sur le bus CAN
{
    Historique.AppendText("\r\n");
    Historique.AppendText("\r\n");
    Historique.AppendText(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
    Historique.AppendText("\r\n");
    Historique.AppendText("Error sending data on CAN bus (try catch error)");
}
```

Comment prendre une décision en fonction de la trame reçue (avec try catch)

```
ListViewItem lviCurrentItem;
try
{
    foreach (MessageStatus msgStatus in m_LastMsgsList)
    {
        if (msgStatus.MarkedAsUpdated)
        {
            msgStatus.MarkedAsUpdated = false;
            lviCurrentItem = lstMessages.Items[msgStatus.Position];
            lviCurrentItem.SubItems[2].Text = msgStatus.CANMsg.LEN.ToString();
            lviCurrentItem.SubItems[3].Text = msgStatus.DataString;
            lviCurrentItem.SubItems[4].Text = msgStatus.Count.ToString();
            lviCurrentItem.SubItems[5].Text = msgStatus.TimeString;

            textBox1.Text = lviCurrentItem.SubItems[3].Text; // Enregistre le text reçu dans un string
        }
    }

    if (textBox1.Text == "41 42 43 44 45 46") // Si cette trame en particulier est reçue
    {
        label19.Text = "BINGO"; // Prend cette décision

        Historique.AppendText("\r\n");
        Historique.AppendText("\r\n");
        Historique.AppendText(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
        Historique.AppendText("\r\n");
        Historique.AppendText("Donnée reçue");
    }
}
catch
{
    Historique.AppendText("\r\n");
    Historique.AppendText("\r\n");
    Historique.AppendText(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
    Historique.AppendText("\r\n");
    Historique.AppendText("Erreur de réception try catch");
}
```

2 Le matériel

2.1 Bloc 1

D'un point de vu matériel, le bloc 1 est le plus simple, car il est composé d'un ordinateur Windows munie d'une carte PCI vers CAN et d'une prise RS232. Il n'y a rien à faire, mise à part brancher les bons câbles aux bons endroits.

2.2 Bloc 2

2.3 Bloc 3

2.4 Bloc 4

2.5 Explication des types de liens

2.5.1 RS232

Un lien RS232 9600 Bauds est établi entre l'ordinateur et le SOC8200. Ce lien sert à l'envoie et à la réception de HeartBeat, afin que le SOC8200 ou l'ordinateur soit informé de toute défaillance de l'autre.

2.5.2 Xbee

Lorsque les modules Xbee sont adéquatement configurés, ils font office de remplacement au câble RS232. En effet, nos Xbee discutent entre eux à l'aide du protocole de communication RS232 à 9600 bauds.

2.6 Explication des trames

2.6.1 RS-232

2.6.2 CAN

2.6.3 XBEE

2.7 Liste des pièces

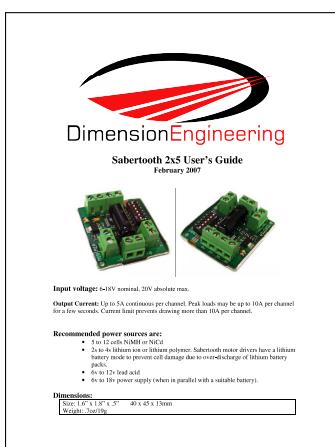
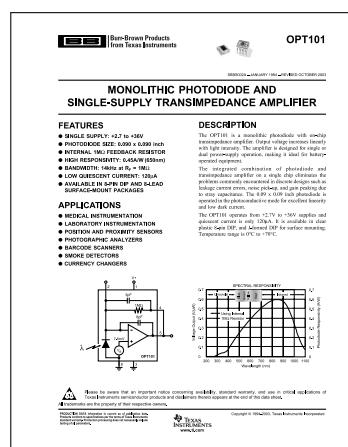
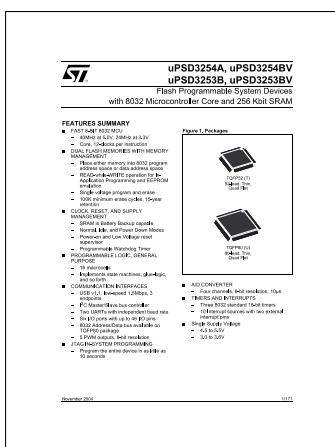
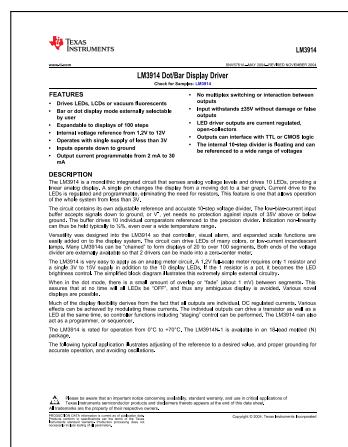
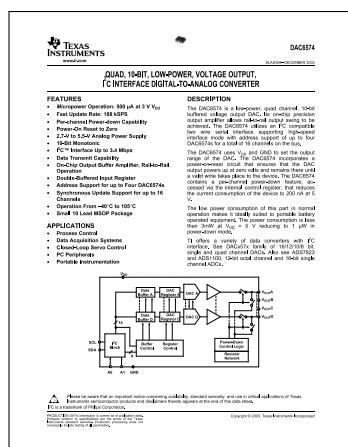
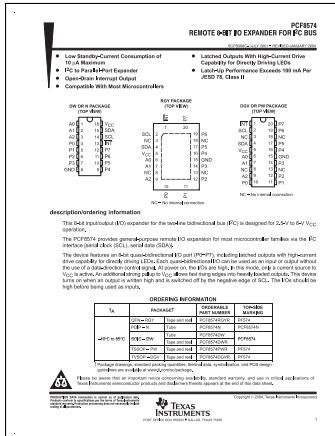
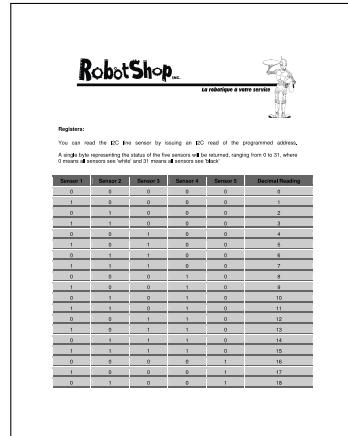
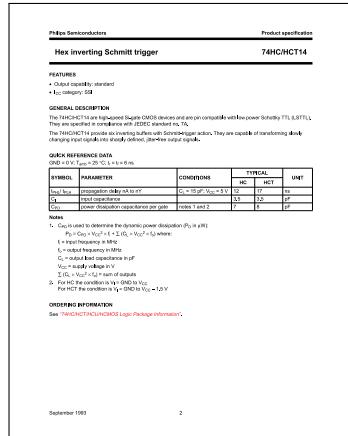
-
- | | | |
|-------------------------|----------------------------|---|
| • Carte Dallas | • XBEE | • |
| • Carte uPSD | • Table FESTO | • |
| • SOC 8200 | • Carte d'extension IO | • |
| • PIC18Fmachin | • Carte connecteur DAC ADC | • |
| • Carte d'extension SPI | • | • |
| • Carte d'extension I2C | • | |
| • Carte CAN MCP2515 | • | |
-

2.7.1 Liens web

Mettre ien vers GitHUB

2.7.2 Datasheets

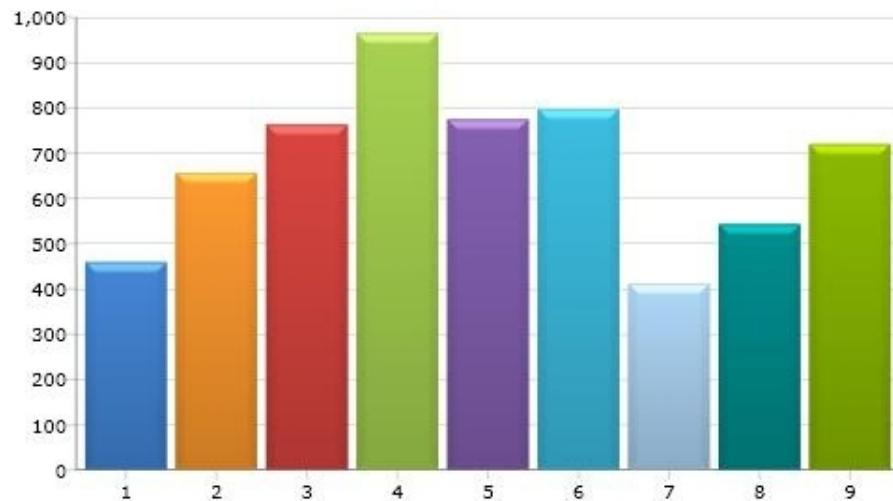
Note : Toutes les datasheets sont en format non-compressé. Vous pouvez zoomer sur le document PDF¹ afin de lire l'intégralité de leurs première page.



- Inclure les datasheet de la sorte économise du papier, donc des arbres

3 Interface PC

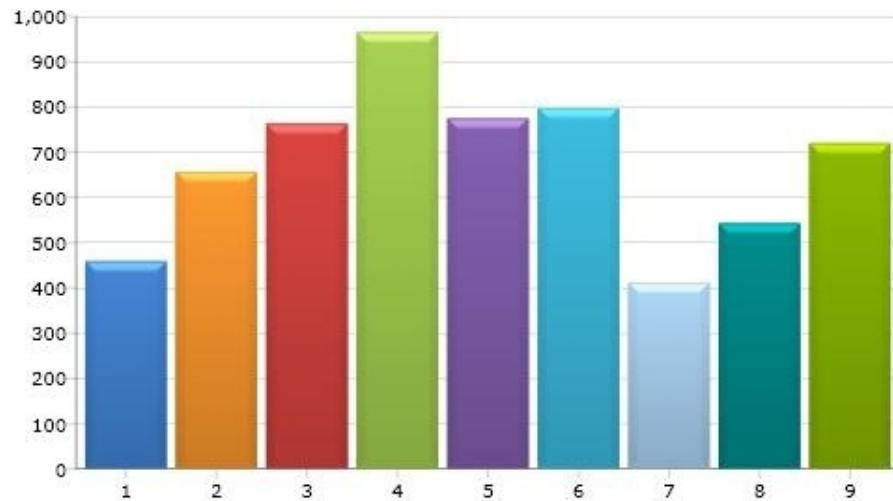
FIGURE 1 – Programme de contrôle principal



Notre programme, écrit en Csharp à l'aide de Visual Studio, peut se connecter au bus CAN via une carte SPI² et au bus RS232 via un câble DB9 ou USB³. La connexion RS232 sert à l'envoie et à la réception du HeartBeat afin d'informer le SOC8200 si l'ordinateur en venait à connaître une défaillance. De plus, des témoins lumineux s'allument en présence de données transmises et reçues.

Le programme peut lire l'heure interne du PC et, par un simple clic sur le bouton «Synchroniser», ajuster l'heure de référence de la station no.2.

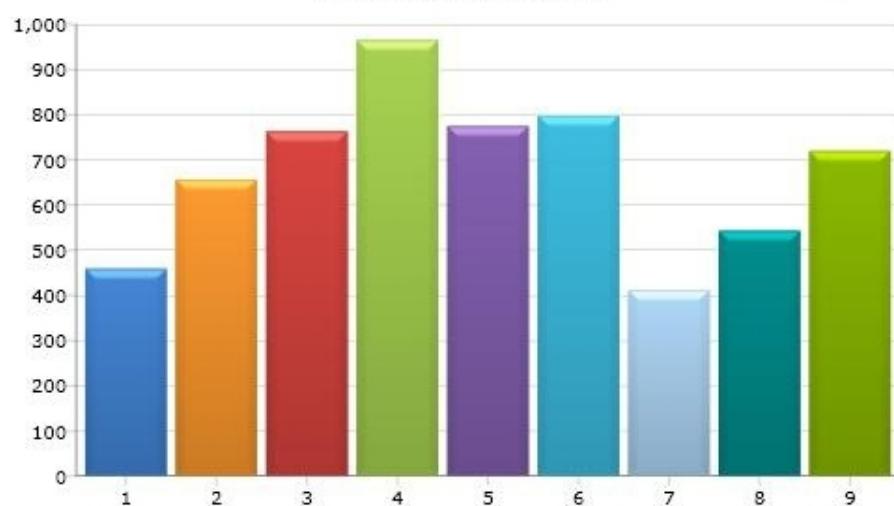
FIGURE 2 – Options CAN avancées



Il est possible d'utiliser des fonctionnalités CAN avancées tels que les masques et filtres de données. De plus, cette fenêtre permet de visualiser les données CAN reçues à l'état brutes et non traitées, ce qui peut s'avérer utile pour du débogage.

Toute action effectuée via le programme ainsi que toute donnée ayant transité sur le bus CAN et RS232 est catalogué en

2. Spécifier le fabricant
3. S'il y a présence d'un FTDI

FIGURE 3 – Historique des actions

bonne et due forme dans l'onglet historique qu'il est possible de consulter et sauvegarder à tout moment.

3.1 Structure du programme

Parler des GhostLabels

3.2 Explication des trames

3.3 Ordre de gestion des tâches

4 Logiciel du SOC8200

4.1 Description du programme

Le programme du SOC2800 est écrit en script Shell⁴ ⁵

4.2 Schéma bloc

4.2.1 Du code

4.2.2 Du script shell

4.3 Gestion des processus et du temps de CPU

4.4 Format et récupération des logs

4.5 Liste des tests et logiciels

4. Car c'est plus simple que de se battre avec le gestionnaire de licence de Sourcery Codebench

5. L'un des membres de l'équipe fait dire que les licences sont une horreur inacceptable sur un système Linux

5 Logiciel de la station 1 et 2 et du bolide

Le programme de la station 1, 2 et du bolide est écrit en C++ à l'aide d'IAR WorkBench 8.20 et la compilation conditionnelle offre de le compiler pour chacune des trois stations mentionnées. De plus, la compilation conditionnelle permet au bolide d'utiliser soit une carte d'extension I2C, soit une carte d'extension SPI pour contrôler ses moteurs.

5.1 La station no.1

5.2 La station no.2

5.3 Le bolide

Composante	Adresse I2C	Description
MAX1236	0x68	Convertisseur analogique-numérique
DS1307	0xD0	Circuit d'horloge RTC
PCF8574	0x40	I/O Expander pour bus I2C
DAC6574	0x98	Convertisseur numérique-analogique
OPT101	0x50	Suiveur de ligne

TABLE 4 – Informations sur le bus I2C du bolide

5.4 Procédure de compilation sur IAR

5.5 Procédure de vérification

6 Logiciel du module PIC18F258

6.1 Description du fonctionnement du programme

6.2 Procédure de compilation sur MPLAB

6.3 Procédure de vérification

7 Calculs

7.1 Calcul du pas de conversion de la pile

$$V_{MAX} = 10.8V \Rightarrow MAX1236 = 12bit \Rightarrow Pas = \frac{4K\Omega \cdot \left(\frac{10.8V}{10K\Omega} \right)}{2^{12}(pas)} = 1.33(mV/pas)$$

7.2 Calcul du baudrate

$$Baud = \frac{2^{SMOD}}{32} \cdot \frac{Crystal(Hz)}{12 \cdot (256 - TH1)}$$

Alors...

$$\underbrace{9600 = \frac{2^1}{32} \cdot \frac{24 \cdot 10^6(Hz)}{12 \cdot (256 - 243)}}_{uPSD3254} \Leftarrow \& \Rightarrow \underbrace{9600 = \frac{2^0}{32} \cdot \frac{11.0597 \cdot 10^6(Hz)}{12 \cdot (256 - 253)}}_{DS89C450}$$

8 Conclusion

8.1 Ce que le projet m'a apporté

8.1.1 Vincent Chouinard

8.1.2 Hicham Safoine

8.1.3 Gabriel Fortin-Bélanger

8.1.4 Louis-Norman Ang-Houle

8.2 Difficultés et corrections

8.2.1 Vincent Chouinard

8.2.2 Hicham Safoine

8.2.3 Gabriel Fortin-Bélanger

8.2.4 Louis-Norman Ang-Houle

8.3 Ce que j'ai aimé ou pas

8.3.1 Vincent Chouinard

8.3.2 Hicham Safoine

8.3.3 Gabriel Fortin-Bélanger

8.3.4 Louis-Norman Ang-Houle