



**Cégep Limoilou**

ÉLECTRONIQUE PROGRAMMABLE ET ROBOTIQUE

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

## Projet de 5<sup>e</sup> session

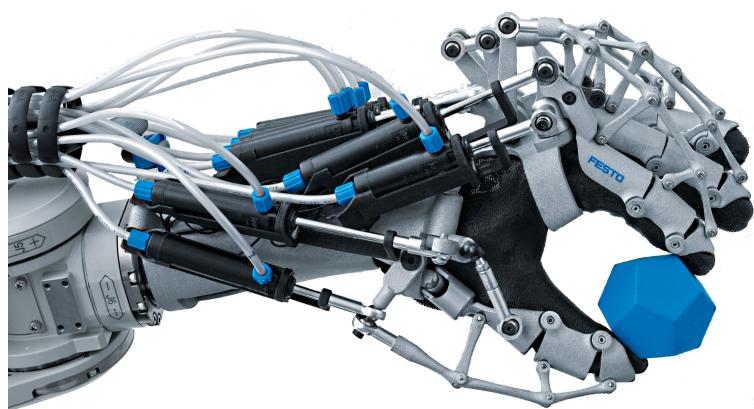
---

### 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 !

28 novembre 2014

## Table des matières

<b>1 Présentation du projet</b>	<b>5</b>
1.1 Explication du projet . . . . .	5
1.2 Schéma bloc du système . . . . .	5
1.2.1 Bloc 1 . . . . .	6
1.2.2 Bloc 2 . . . . .	6
1.2.3 Bloc 3 . . . . .	6
1.2.4 Bloc 4 . . . . .	6
1.3 Liste des logiciels . . . . .	6
1.4 Liste des trames . . . . .	7
<b>2 Le matériel</b>	<b>8</b>
2.1 Bloc 1 . . . . .	8
2.2 Bloc 2 . . . . .	8
2.3 Bloc 3 . . . . .	8
2.4 Bloc 4 . . . . .	8
2.5 Explication des types de liens . . . . .	8
2.5.1 RS232 . . . . .	8
2.5.2 Xbee . . . . .	8
2.6 Explication des trames . . . . .	8
2.6.1 RS-232 . . . . .	8
2.6.2 CAN . . . . .	10
2.6.3 XBEE . . . . .	12
2.7 Liste des pièces . . . . .	12
2.7.1 Liens web . . . . .	12
2.7.2 Datasheets . . . . .	13
<b>3 Interface PC</b>	<b>14</b>
3.1 Gestion de l'historique . . . . .	16
3.1.1 Exemple d'historique typique . . . . .	16
3.2 Structure du programme . . . . .	16
3.2.1 Les Ghosts Labels . . . . .	16
3.3 Explication des trames . . . . .	16
3.4 Ordre de gestion des tâches . . . . .	16
<b>4 Logiciel du SOC8200</b>	<b>17</b>
4.1 Description du programme . . . . .	17
4.2 Schéma bloc . . . . .	17
4.2.1 Du script shell . . . . .	17
4.3 Gestion des processus et du temps de CPU . . . . .	17
4.4 Format et récupération des logs . . . . .	17
4.5 Liste des tests et logiciels . . . . .	17
<b>5 Logiciel de la station 1 et 2 et du bolide</b>	<b>18</b>
5.1 La station no.1 . . . . .	18
5.2 La station no.2 . . . . .	18
5.3 Schéma des héritages de classes, table FESTO . . . . .	19
5.4 Schéma des héritages de classes, Bolide et Station 1 . . . . .	20
5.5 Le bolide . . . . .	21
5.6 Procédure de compilation sur IAR . . . . .	22
5.7 Procédure de vérification . . . . .	22

<b>6 Logiciel du module PIC18F258</b>	<b>23</b>
6.1 Description du fonctionnement du programme . . . . .	23
6.2 Procédure de compilation sur MPLAB . . . . .	23
6.3 Procédure de vérification . . . . .	23
<b>7 Calculs</b>	<b>24</b>
7.1 Calcul du pas de conversion de la pile . . . . .	24
7.2 Calcul du baudrate . . . . .	24
<b>8 Fichiers Gerbers</b>	<b>25</b>
<b>9 Conclusion</b>	<b>28</b>
9.1 Ce que le projet m'a apporté . . . . .	28
9.1.1 Vincent Chouinard . . . . .	28
9.1.2 Hicham Safoine . . . . .	28
9.1.3 Gabriel Fortin-Bélanger . . . . .	28
9.1.4 Louis-Norman Ang-Houle . . . . .	28
9.2 Difficultés et corrections . . . . .	28
9.2.1 Vincent Chouinard . . . . .	28
9.2.2 Hicham Safoine . . . . .	28
9.2.3 Gabriel Fortin-Bélanger . . . . .	28
9.2.4 Louis-Norman Ang-Houle . . . . .	28
9.3 Ce que j'ai aimé ou pas . . . . .	28
9.3.1 Vincent Chouinard . . . . .	28
9.3.2 Hicham Safoine . . . . .	28
9.3.3 Gabriel Fortin-Bélanger . . . . .	28
9.3.4 Louis-Norman Ang-Houle . . . . .	28
<b>10 ANNEXE 1 : Code source du programme pour PC</b>	<b>29</b>
<b>11 ANNEXE 2 : Code source du Bolide et de la station 1</b>	<b>30</b>
<b>12 ANNEXE 4 : Code source de la table FESTO</b>	<b>31</b>
<b>13 ANNEXE 5 : Code source du programme PIC</b>	<b>32</b>
<b>14 ANNEXE 4 : Code source du script Shell du SOC8200</b>	<b>33</b>

## Table des figures

1	Vue d'ensemble du projet . . . . .	5
2	Programme de contrôle principal . . . . .	14
3	Options CAN avancées . . . . .	15
4	Historique des actions . . . . .	16
5	Héritage de la classe CLFesto . . . . .	19
6	Héritage de la classe cInOutBase . . . . .	19
7	Héritage de la classe clVéhicule . . . . .	20
8	Héritage de la classe I2C . . . . .	20
9	Héritage de la classe SPI . . . . .	20
10	Héritage de la classe cInOutBase . . . . .	21
11	Héritage de la classe Station1 . . . . .	21
12	Choix de la cible sur IAR . . . . .	22
13	Couche TOP . . . . .	25
14	Couche BOT . . . . .	25
15	Silk Screen TOP . . . . .	26
16	Solder mask TOP . . . . .	26
17	Drill . . . . .	27
18	Correctif . . . . .	27

## Liste des tableaux

1	Index des identifiants matériel CAN . . . . .	10
2	Index des trames CAN . . . . .	10
3	Index des communications CAN . . . . .	10
4	Informations sur le bus I2C du bolide . . . . .	21

# 1 Présentation du projet

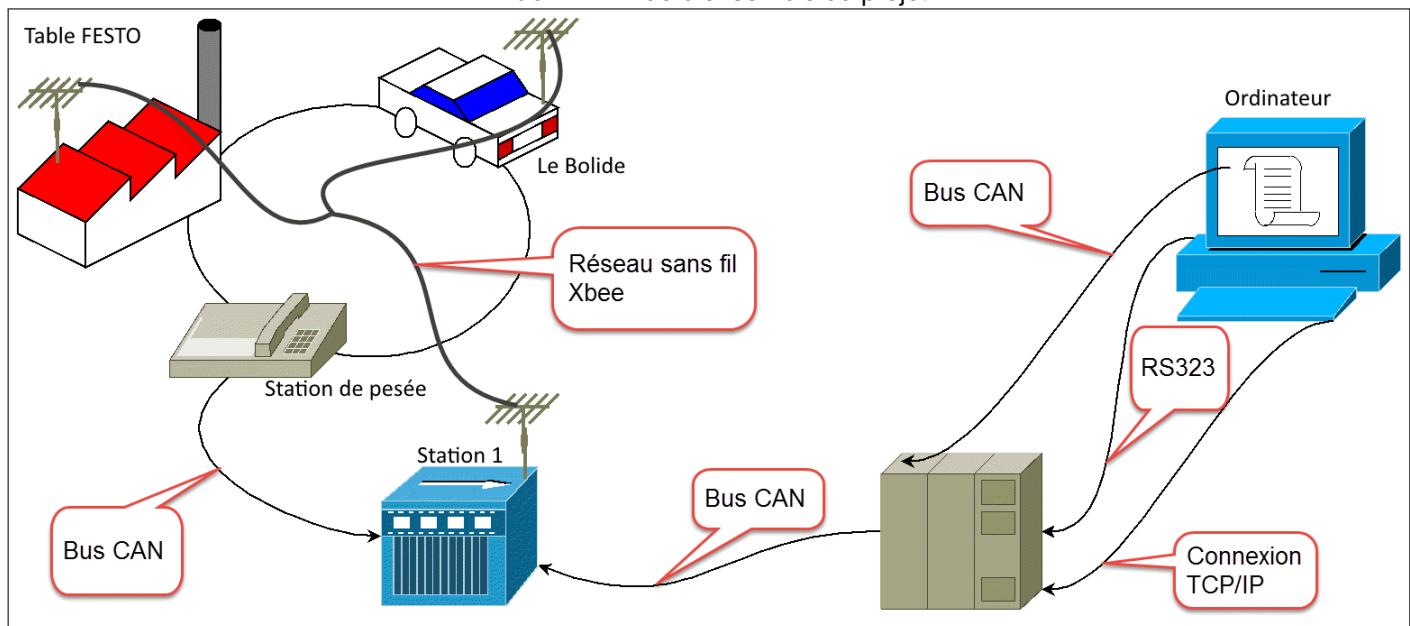
Le projet de la cinquième session consiste à réaliser

- ⇒ Le Bolide
- ⇒ Carte Dallas DS89C450
- ⇒ Carte uPSD 3254A
- ⇒ SOC8200
- ⇒ Table FESTO
- ⇒ Carte PIC16F88
- ⇒ Carte d'extension I<sub>2</sub>C
- ⇒ 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

FIGURE 1 – Vue d'ensemble du projet



**1.2.1 Bloc 1****1.2.2 Bloc 2****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
- xTerminator
- CAPS
- tinyBootloader

**Gestion du projet**

- 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

f

## 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 et d'une prise Ethernet. Il n'y a rien à faire, mise à part brancher les bons câbles aux bons endroits. Son rôle est de contrôler et diriger toute l'opération et de veiller au bon fonctionnement de chaque composantes à l'aide d'une application en Csharp. Le bloc 1 est le cerveau de l'usine.

### 2.2 Bloc 2

Le bloc 2 est composé du bolide et de la station no.1. Cette dernière, dont le cerveau est une carte uPSD, joue le rôle de centralisateur CAN. En effet, cette station reçoit des consignes<sup>1</sup> en provenance du PC, consignes qu'elle s'empresse d'expédier aux bons endroits via Xbee. De plus, cette station reçoit des informations de la station de pesée<sup>2</sup>, de la table FESTO<sup>3</sup> et du bolide<sup>4</sup>. Ces informations sont systématiquement retransmises au PC via le bus CAN.

### 2.3 Bloc 3

### 2.4 Bloc 4

Le bloc 4 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.

## 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'envoyer 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

Le protocole RS-232 sert à envoyer et à recevoir des HeartBeat. Le PC et le SOC 8200 s'envoient tous deux un HeartBeat par seconde à 9600 bauds. Un HeartBeat, c'est simplement le mot "Allo". Le PC et le SOC2800 "écoutent" les HeartBeats,

1. Sous forme de trames

2. Sous forme de trames CAN

3. Via Xbee

4. Idem

et si ces derniers ne sont pas entendus, chaque dispositif prend pour acquis que l'autre est hors-service et prend la relève de la gestion du bus CAN.

## 2.6.2 CAN

Chaque dispositif matériel, du Bolide au PC, dispose d'un identifiant CAN allant de 000 à 005. Chaque fonctionnalité dispose d'un code d'identification suivi de la donnée à transmettre.

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

TABLE 3 – Index des communications CAN

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

Exemples de trames CAN à transmettre au PC.

```
CAN.SendToPC("0100FF"); // Arrêté
CAN.SendToPC("0101FF"); // En marche
CAN.SendToPC("0102FF"); // Hors circuit
CAN.SendToPC("02xxFF"); // Vitesse de xx
CAN.SendToPC("03xxFF"); // Batterie chargée à xx %
CAN.SendToPC("0400FF"); // Bloc métallique
CAN.SendToPC("0401FF"); // Bloc noire
CAN.SendToPC("0402FF"); // Bloc orange
CAN.SendToPC("050064"); // Le bloc est lourd
CAN.SendToPC("0700FF"); // Rendu à la station 1
CAN.SendToPC("0701FF"); // Rendu à la station 2
CAN.SendToPC("0702FF"); // Rendu à la station 3
```

### 2.6.3 XBEE

Trois modules Xbee sont présent sur l'ensemble du projet, soit sur la station no.1 (la carte uPSD), sur la station no.2 (la table FESTO) et la station no.6, c'est à dire le bolide. La particularité des Xbee est que lorsqu'ils sont adéquatement configurés, tout ce qu'envoie un Xbee est reçu et lu par tous les autres Xbee à proximité, et c'est pourquoi nous avons défini un système de trames.

## 2.7 Liste des pièces

---

- Carte Dallas
- Carte uPSD
- SOC 8200 (avec clavier)
- PIC18FmachinTruc
- Carte d'extension SPI
- Carte d'extension I2C
- Carte CAN MCP2515
- Xbee
- Table FESTO
- Carte d'extension IO
- Carte connecteur IO
- Carte connecteur DAC ADC
- Carte Xbee vers DB9
- Câble Ethernet croisé
- Câble Ethernet régulier
- Câble DB9
- Carte Xbee vers DB8
- Le Bolide
- 
- 

---

### 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 PDF<sup>5</sup> afin de lire l'intégralité de leurs premières pages.

74HC14

PNP Transistor		Product Specification																					
Hex Inverting Schmitt trigger		74HC/HCT14																					
<b>FEATURES</b>																							
<ul style="list-style-type: none"> <li>• Output capable of standard</li> <li>• 10-ohm load</li> </ul>																							
<b>GENERAL DESCRIPTION</b>																							
<p>The 74HC/HCT14 are high-speed肖特基CMOS devices and are pin compatible with low-power Schottky TTL, ECL/TTL, The very high speed and low power consumption make them ideal for high-speed logic applications.</p> <p>The 74HC/HCT14 provides six inverting buffers with Schmitt-gate action. They are capable of functioning at clock rates up to 150 MHz. The output is open drain, allowing external pull-up resistors.</p>																							
<b>GRAPHIC REFERENCE DATA</b>																							
GND = 0 V, TA = 25°C, V <sub>DD</sub> = 5 V, f = 100 Hz																							
<table border="1"> <thead> <tr> <th>SYMBOL</th> <th>PARAMETER</th> <th>CONDITIONS</th> <th>TYPICAL</th> <th>UNIT</th> </tr> </thead> <tbody> <tr> <td>I<sub>DS</sub></td> <td>Drain-to-source saturation current</td> <td>V<sub>D</sub> = 0.5 V, V<sub>G</sub> = 0.5 V</td> <td>-12</td> <td>mA</td> </tr> <tr> <td>I<sub>S</sub></td> <td>Input leakage current</td> <td>-</td> <td>32</td> <td>PF</td> </tr> <tr> <td>I<sub>OD</sub></td> <td>Output drain-to-source leakage current per gate</td> <td>V<sub>D</sub> = 5 V, V<sub>G</sub> = 0.5 V</td> <td>2</td> <td>PF</td> </tr> </tbody> </table>				SYMBOL	PARAMETER	CONDITIONS	TYPICAL	UNIT	I <sub>DS</sub>	Drain-to-source saturation current	V <sub>D</sub> = 0.5 V, V <sub>G</sub> = 0.5 V	-12	mA	I <sub>S</sub>	Input leakage current	-	32	PF	I <sub>OD</sub>	Output drain-to-source leakage current per gate	V <sub>D</sub> = 5 V, V <sub>G</sub> = 0.5 V	2	PF
SYMBOL	PARAMETER	CONDITIONS	TYPICAL	UNIT																			
I <sub>DS</sub>	Drain-to-source saturation current	V <sub>D</sub> = 0.5 V, V <sub>G</sub> = 0.5 V	-12	mA																			
I <sub>S</sub>	Input leakage current	-	32	PF																			
I <sub>OD</sub>	Output drain-to-source leakage current per gate	V <sub>D</sub> = 5 V, V <sub>G</sub> = 0.5 V	2	PF																			
<b>Notes</b>																							
<p>1. Current required to determine the dynamic power dissipation (<math>P_D</math>) is 100 Hz.</p> <p>2. The input voltage range is <math>V_{IN}</math> = 0 V to <math>V_{DD}</math> = 5 V.</p> <p>3. Input frequency is <math>f = 100</math> Hz.</p> <p>4. Load condition is <math>I_{L} = 10</math> mA.</p> <p>5. Output load is <math>I_{OD} = 2</math> PF.</p> <p>6. <math>V_{DD} = 5</math> V, <math>V_{G} = 0.5</math> V.</p> <p>7. <math>V_{DD} = 5</math> V, <math>V_{G} = 0.5</math> V.</p> <p>8. <math>V_{DD} = 5</math> V, <math>V_{G} = 0.5</math> V.</p> <p>9. <math>V_{DD} = 5</math> V, <math>V_{G} = 0.5</math> V.</p>																							
<b>ORDERING INFORMATION</b>																							
See "74HC/HCT14/NOL Logo, Package Information".																							

## LineSensors

# RobotShop<sub>...</sub>

la redondeur à votre service

## Registers

You can read the ECU's data memory by issuing an ECU read of the programmed address. A single byte representing the status of the sensors will be received, ranging from 0 to 31, where 0 means 8 sensors active while 31 means 8 sensors inactive.

Register 1	Register 2	Register 3	Register 4	Register 5	Desired Heading
0	0	0	0	0	0
1	0	0	0	0	1
0	1	0	0	0	2
1	1	0	0	0	3
0	0	1	0	0	4
1	0	1	0	0	5
0	1	1	0	0	6
1	1	1	0	0	7
0	0	0	1	0	8
1	0	0	1	0	9
0	1	0	1	0	10
1	1	0	1	0	11
0	0	1	0	0	12
0	1	1	1	0	13
1	1	1	1	0	14
0	0	0	0	1	15
1	0	0	0	1	16
0	1	0	0	1	17
1	1	0	0	1	18

PCF8573

DAC6574

LM3914

uPSD3254A

DS89C450

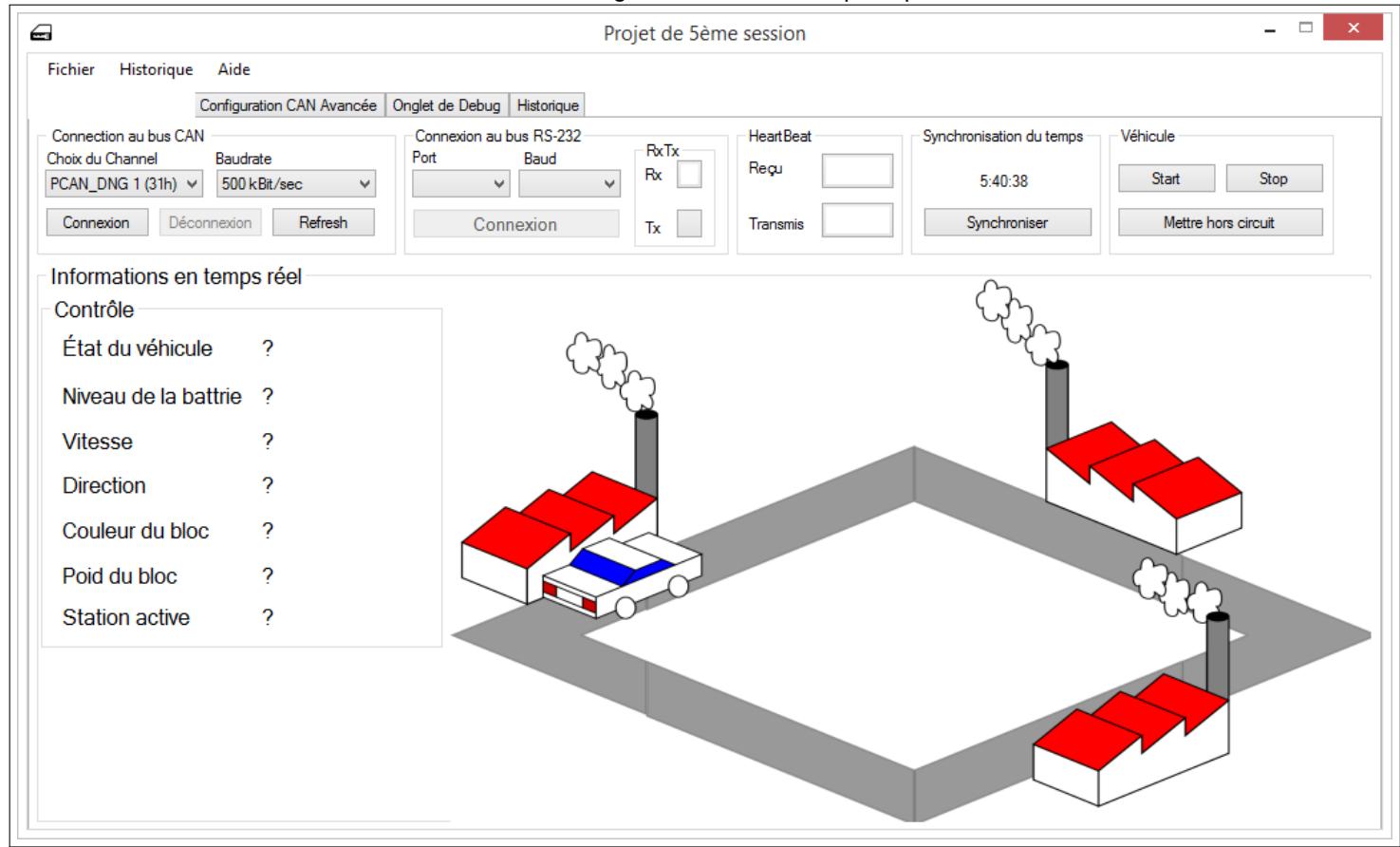
OPT101

## SaberTooth motor drive

5. Présenter les datasheets de la sorte économise du papier, donc des arbres, mais requiert de consulter le document .PDF afin de lire adéquatement les datasheets.

### 3 Interface PC

FIGURE 2 – Programme de contrôle principal



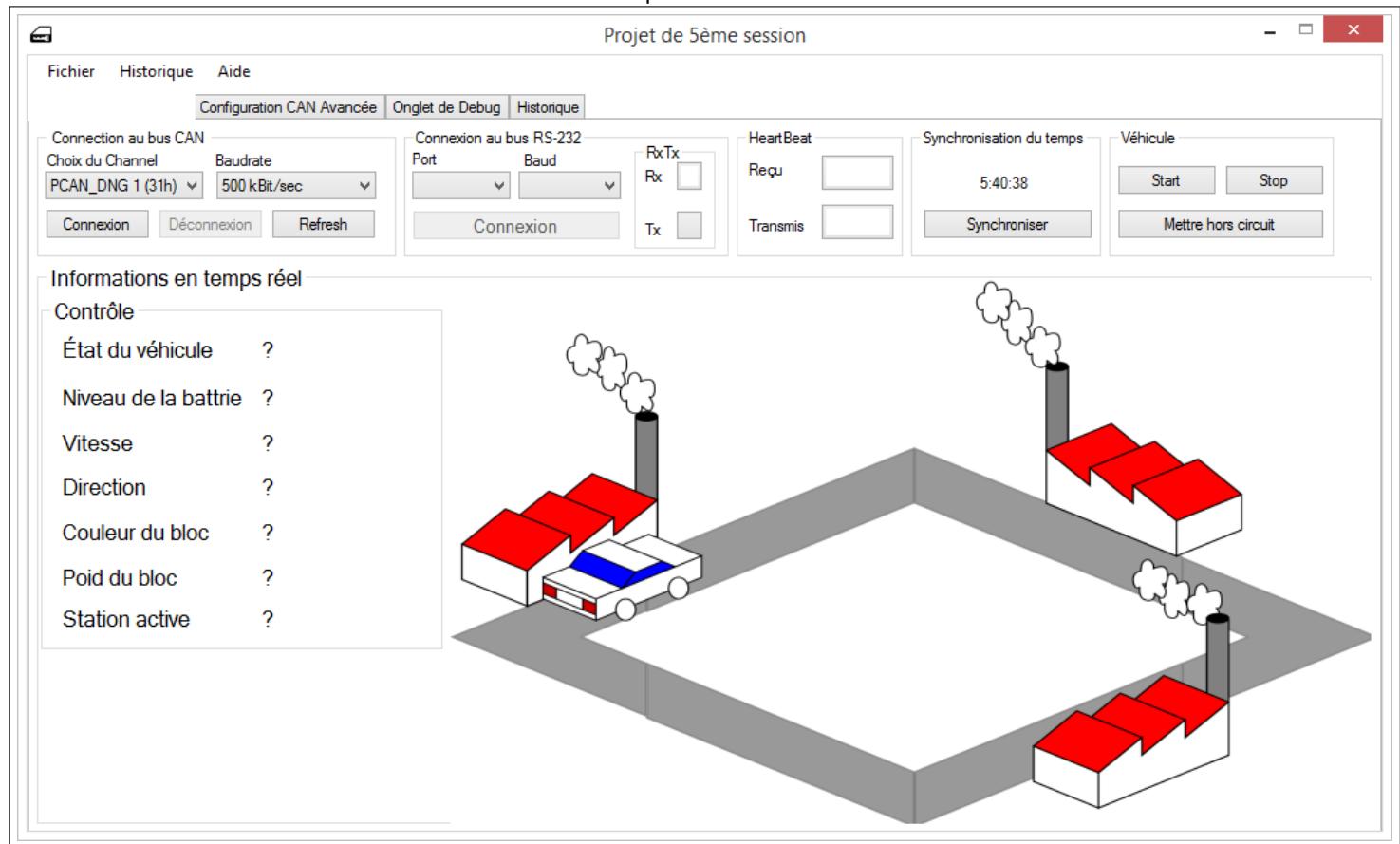
Notre programme, écrit en C# à l'aide de Visual Studio, peut se connecter au bus CAN via une carte SPI<sup>6</sup> et au bus RS232 via un câble DB9 ou USB<sup>7</sup>. La connexion RS232 sert à l'envoie et à la réception du HeartBeat afin d'informer le SOC8200 si l'ordinateur en venait à connaître une 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.1.

6. Spécifier le fabricant

7. S'il y a présence d'un FTDI

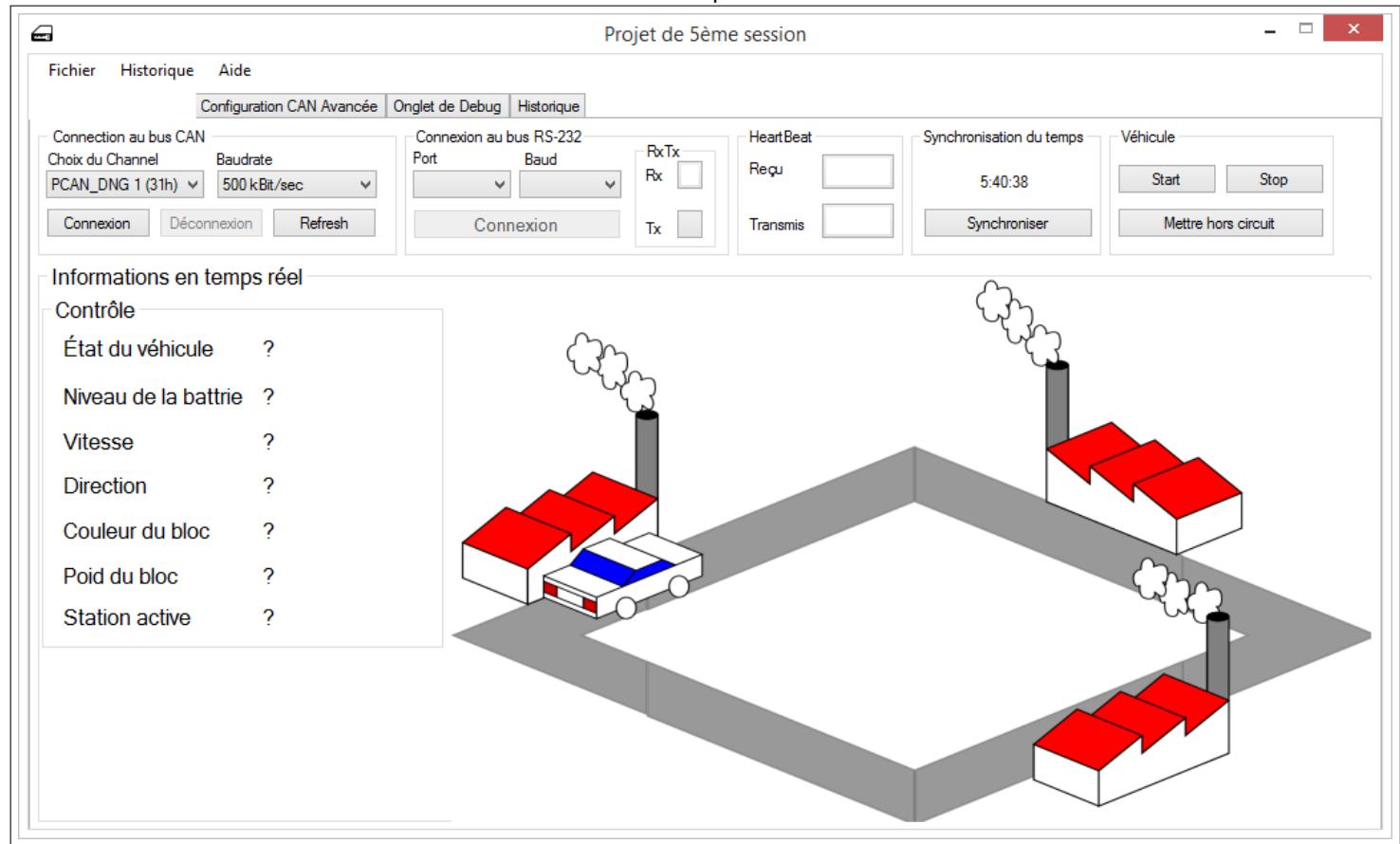
FIGURE 3 – Options CAN avancées



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.

### 3.1 Gestion de l'historique

FIGURE 4 – Historique des actions



Toute action effectué via le programme ainsi que toute donnée ayant transité sur le bus CAN et RS232 est catalogué en bonne et due forme dans l'onglet historique qu'il est possible de consulter et sauvegarder à tout moment.

#### 3.1.1 Exemple d'historique typique

wefsd

### 3.2 Structure du programme

#### 3.2.1 Les Ghosts Labels

Un ghost label est un label de texte présent sur l'interface, mais définit comme invisible. Il est donc impossible pour l'usager de le voir et d'y accéder. Leurs principales utilités est de faire office de variable globale afin de passer des paramètres entre fonctions et de déclencher des événements système lorsqu'ils sont lus ou modifiés.

### 3.3 Explication des trames

### 3.4 Ordre de gestion des tâches

## 4 Logiciel du SOC8200

### 4.1 Description du programme

Le programme du SOC2800 est écrit en script Shell<sup>8 9</sup>

### 4.2 Schéma bloc

#### 4.2.1 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

---

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

9. 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 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

La station no.1 est composé de uPSD et s'appelle Bloc no.2 dans le cahier de consignes. Cette station reçoit les directives du PC par le BUS CAN et les expédie sur le bus CAN (et vice-versa) aux endroits appropriés. C'est aussi à cette station qu'incombe la tâche de communiquer avec le bolide et la table FESTO via des Xbee

### 5.2 La station no.2

La station no.2 (qui s'appelle Bloc no.3 dans le cahier de consignes) est composé de la table FESTO, de la carte uPSD, de trois cartes d'extensions IO et d'un Xbee.

### 5.3 Schéma des héritages de classes, table FESTO

Quelles classes utilisent quelles autres classes dans le code de la table FESTO ?

FIGURE 5 – Héritage de la classe CLFesto

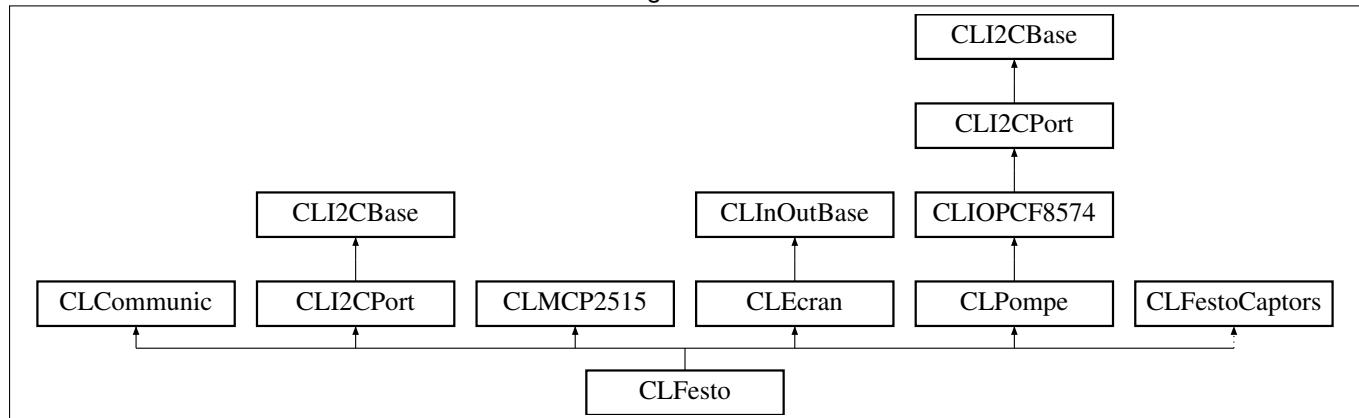
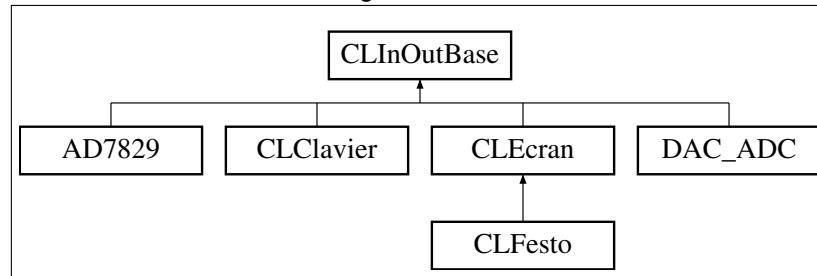


FIGURE 6 – Héritage de la classe CLInOutBase



## 5.4 Schéma des héritages de classes, Bolide et Station 1

Quelles classes utilisent quelles autres classes dans le code du bolide et de la station no. 1 ?

FIGURE 7 – Héritage de la classe clVéhicule

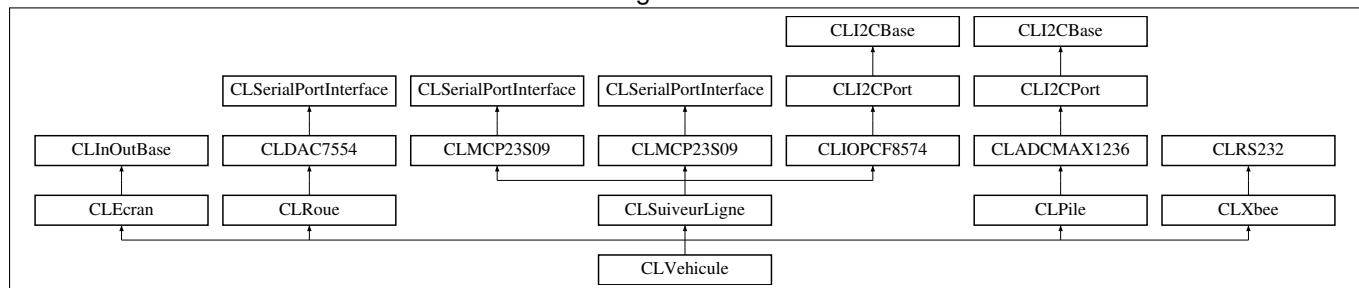


FIGURE 8 – Héritage de la classe I2C

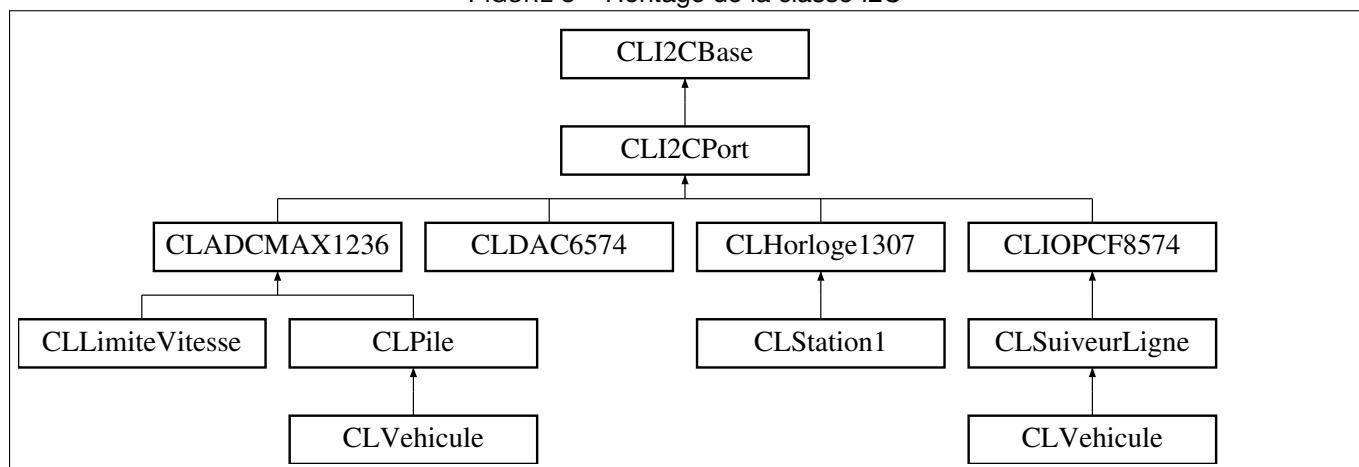


FIGURE 9 – Héritage de la classe SPI

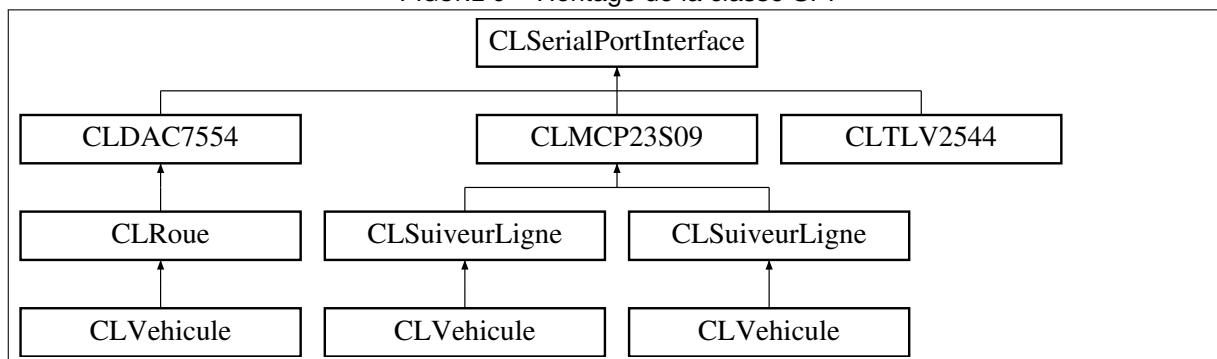


FIGURE 10 – Héritage de la classe CLInOutBase

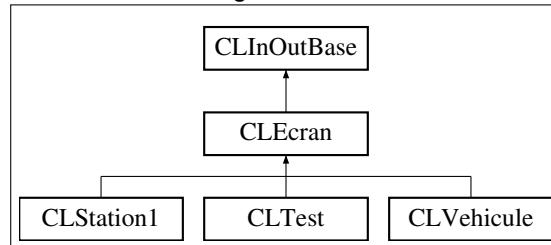
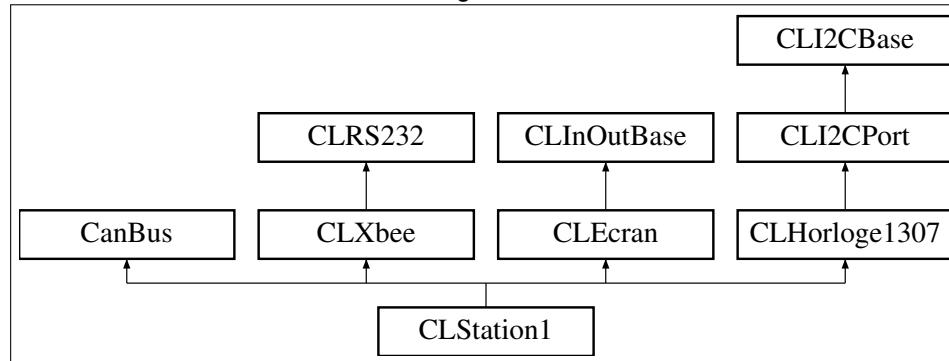


FIGURE 11 – Héritage de la classe Station1



## 5.5 Le bolide

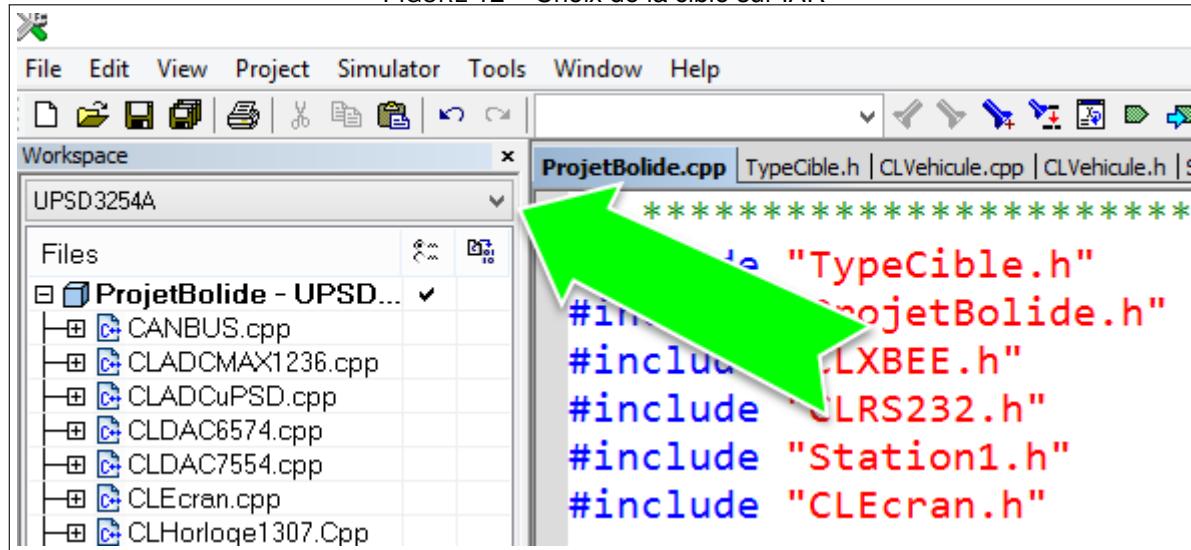
Composante	Adresse I2C	Description
MAX1236	0x68	Convertisseur analogique-numérique
DS1307	0xD0	Circuit d'horloge RTC
PCF8574	0x40	I/O Expander pour bus I2C
DAC6574	0x98	Convertisseur numérique-analogique
OPT101	0x50	Suiveur de ligne

TABLE 4 – Informations sur le bus I2C du bolide

## 5.6 Procédure de compilation sur IAR

Sur IAR, vous pouvez utiliser le menu déroulant, illustré à la figure suivante, afin de compiler le code pour la carte Dallas ou pour la carte uPSD.

FIGURE 12 – Choix de la cible sur IAR



De plus, des paramètres de compilation optionnelle vous permettent, via la décommentation, de compiler le code pour la carte Dallas ou uPSD, pour la carte d'extension I2C ou SPI et pour un capteur de ligne à 3 ou à 5 photorécepteurs.

### Appercu des directives de compilation conditionnelles

```

#ifndef define UPSD3254A
#ifndef define DALLAS89C450
#ifndef define SPI.DALLAS
#ifndef define I2C.DALLAS
#ifndef define PCF.5.CAPTEURS
#ifndef define PCF.3.CAPTEURS

```

## 5.7 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...

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

## 8 Fichiers Gerbers

Une carte d'extension, dont voici les images GERBER<sup>10</sup>, à été réalisée avec OrCAD 16.2 et gravée à l'aide de la rutilante LPKF.

FIGURE 13 – Couche TOP

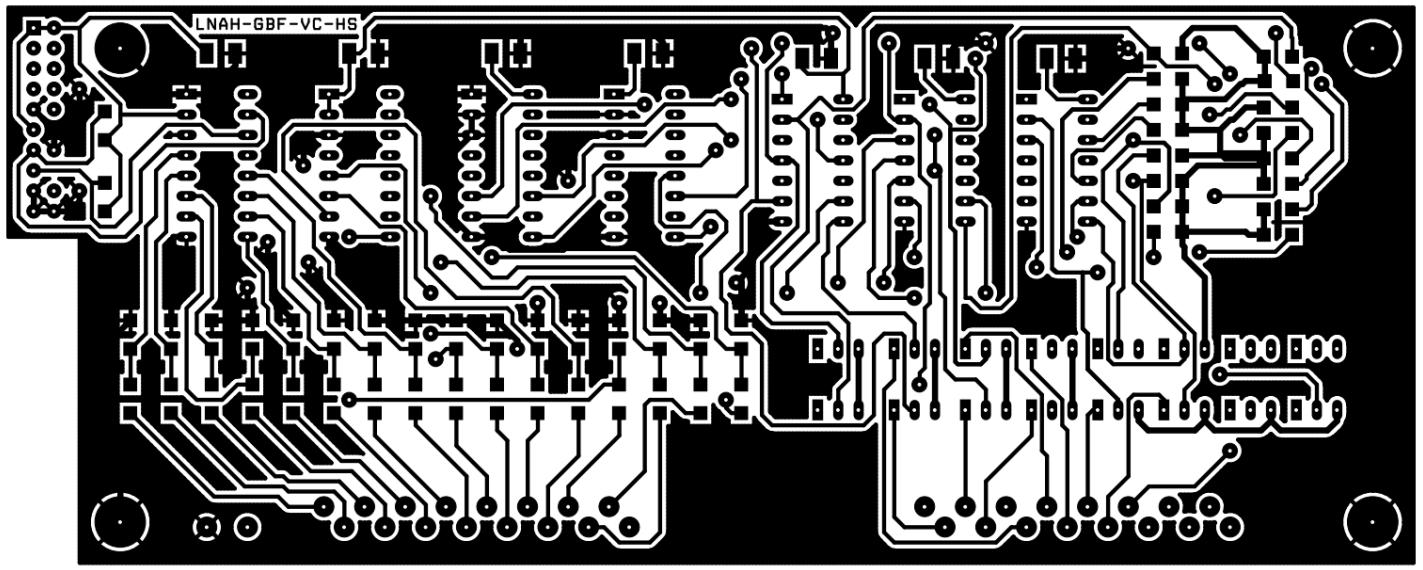
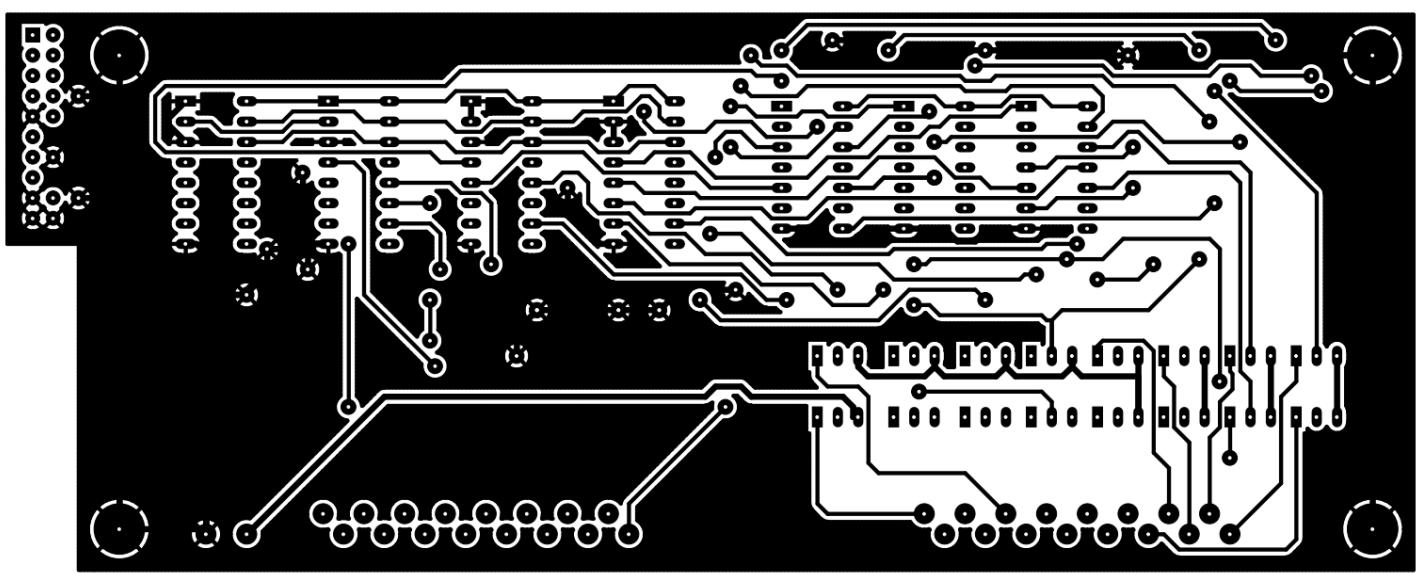


FIGURE 14 – Couche BOT



10. Les couches présentées ne sont pas à l'échelle

FIGURE 15 – Silk Screen TOP

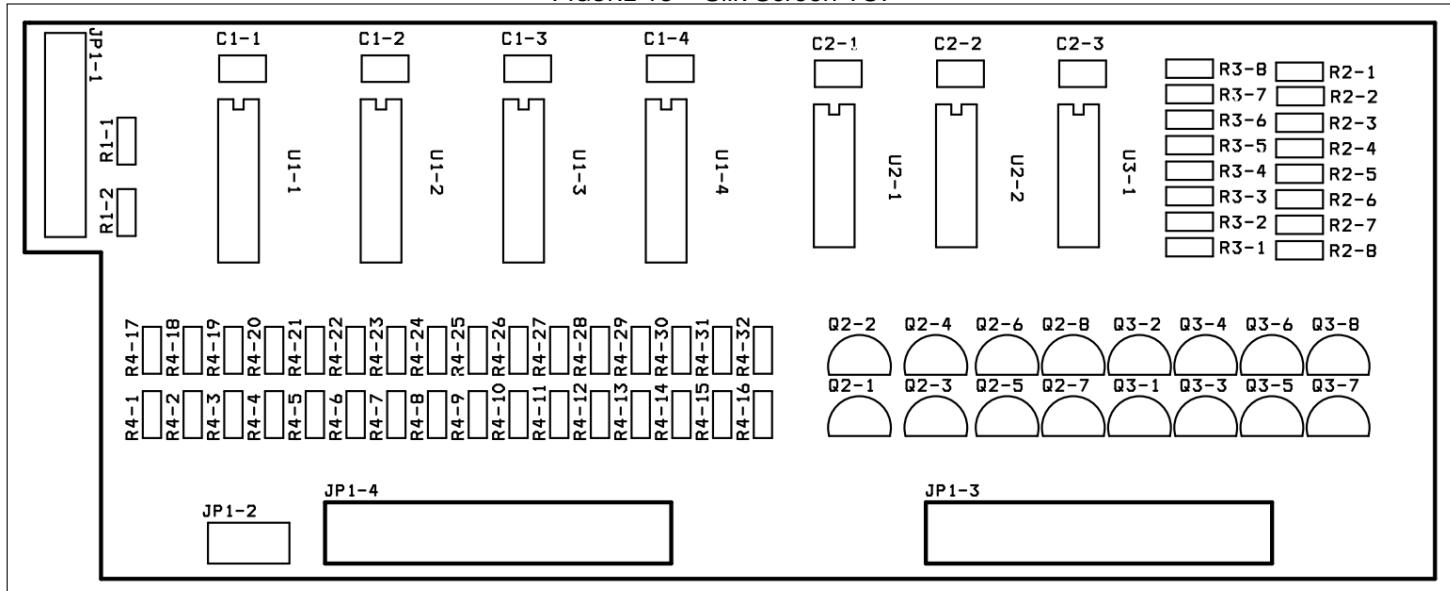


FIGURE 16 – Solder mask TOP

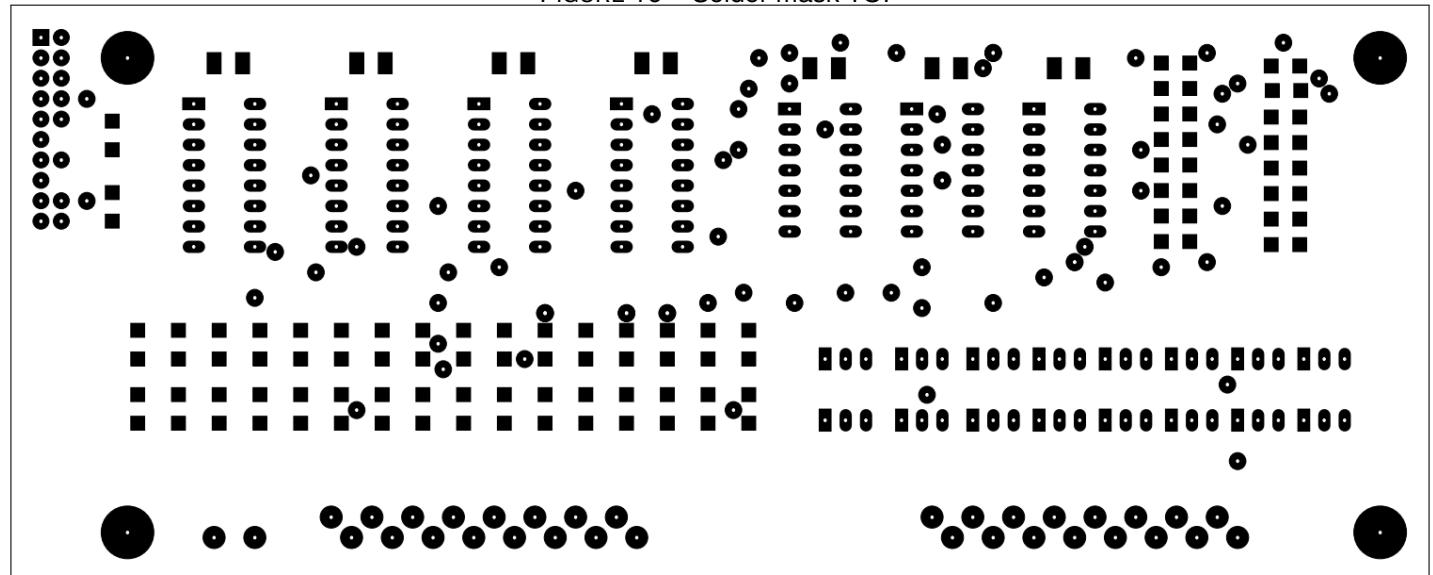


FIGURE 17 – Drill

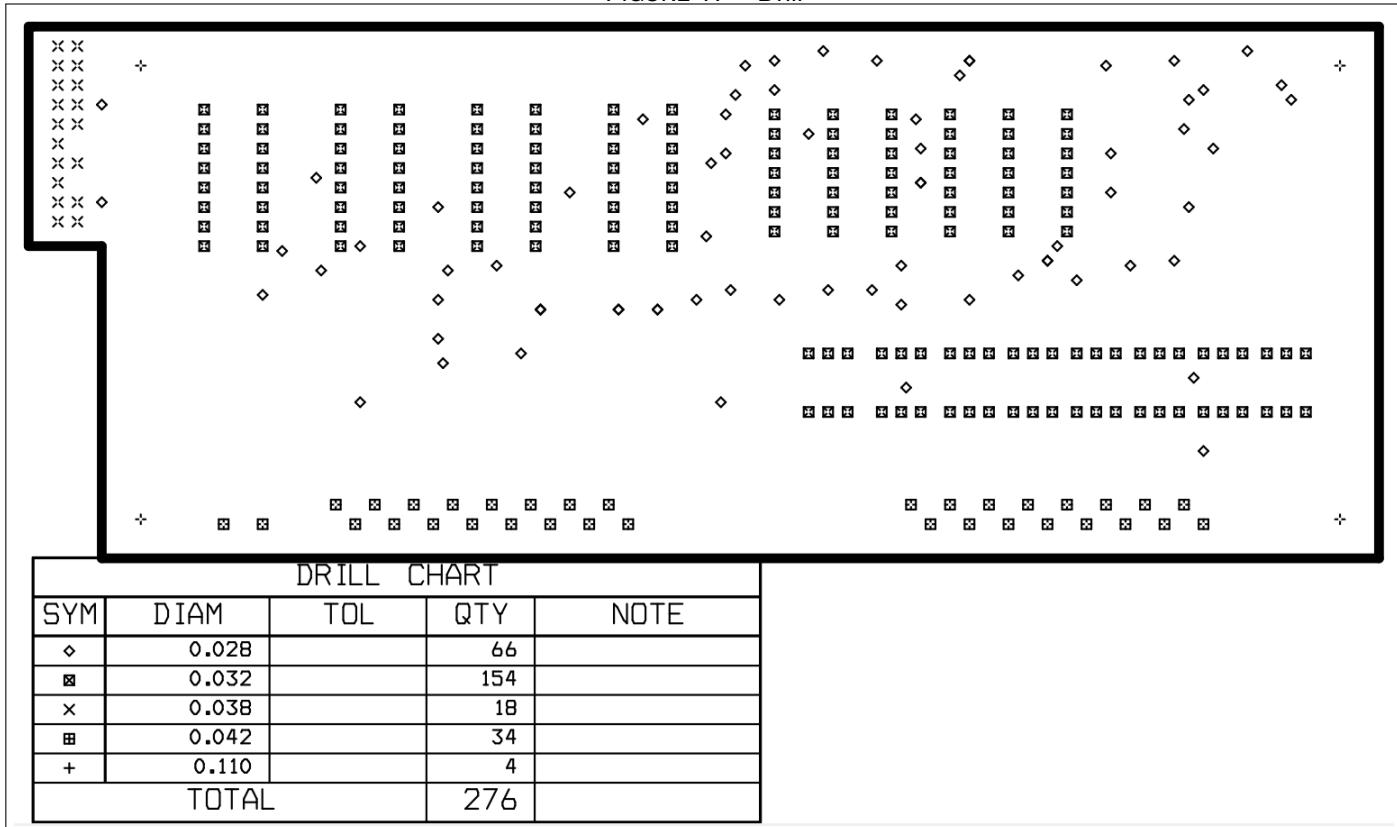
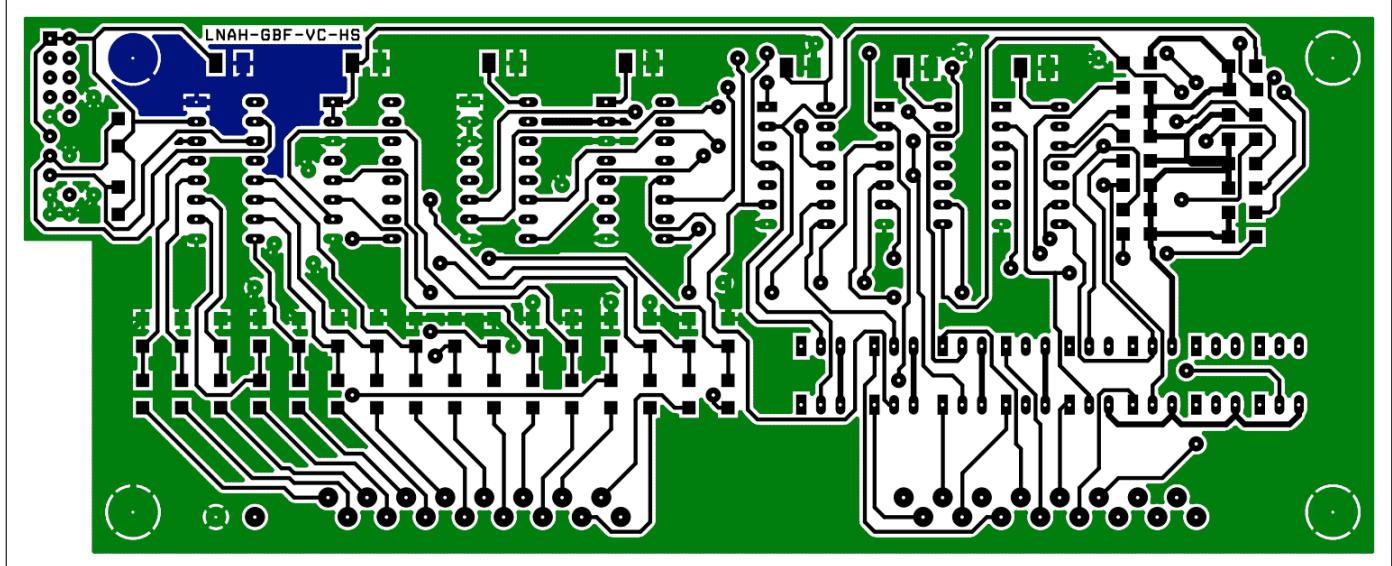


FIGURE 18 – Correctif



Il est à noter que nous avons dû réaliser un correctif lors de la soudure de la carte IO du projet. La zone de ground plane en bleu devait être reliée à GND, c'est-à-dire la zone verte, mais ce n'était pas le cas. C'est pourquoi un bout de fil a été soudé afin de pailler à ce manque.

## 9 Conclusion

### 9.1 Ce que le projet m'a apporté

9.1.1 Vincent Chouinard

9.1.2 Hicham Safoine

9.1.3 Gabriel Fortin-Bélanger

9.1.4 Louis-Norman Ang-Houle

### 9.2 Difficultés et corrections

9.2.1 Vincent Chouinard

9.2.2 Hicham Safoine

9.2.3 Gabriel Fortin-Bélanger

9.2.4 Louis-Norman Ang-Houle

### 9.3 Ce que j'ai aimé ou pas

9.3.1 Vincent Chouinard

9.3.2 Hicham Safoine

9.3.3 Gabriel Fortin-Bélanger

9.3.4 Louis-Norman Ang-Houle

## 10 ANNEXE 1 : Code source du programme pour PC

```
using System;
```

## 11 ANNEXE 2 : Code source du Bolide et de la station 1

```
using System;
```

## 12 ANNEXE 4 : Code source de la table FESTO

```
using System;
```

## 13 ANNEXE 5 : Code source du programme PIC

```
void main( void )
{
}
```

## 14 ANNEXE 4 : Code source du script Shell du SOC8200

```
using System;
```

**Bonus**

\*54 · 43.  $\vdash .\alpha, \beta \in 1. \supset: \alpha \cap \beta = \Lambda. \equiv .\alpha \cup \beta \in 2$

*Dem.*

$$\begin{aligned}
 & \vdash . * 54 \cdot 26. \supset \vdash .\alpha = \iota'x. \beta = \iota'y. \supset: \alpha \cup \beta \in 2. \equiv .x \neq y. \\
 & [\ast 51 \cdot 231] \quad \equiv .\iota'x \cap \iota'y = \Lambda. \\
 & [\ast 13 \cdot 12] \quad \equiv .\alpha \cap \beta = \Lambda \tag{1} \\
 & \vdash .(1). * 11 \cdot 11 \cdot 35. \supset \\
 & \quad \vdash: .(\exists x, y).\alpha = \iota'x. \beta = \iota'y. \supset: \alpha \cup \beta \in 2. \equiv .\alpha \cap \beta = \Lambda \tag{2} \\
 & \vdash .(2). * 11 \cdot 54. * 52 \cdot 1. \supset \vdash .Prop
 \end{aligned}$$

From this proposition it will follow, when arithmetical addition has been defined, that  $1 + 1 = 2$ .

Cette démonstration mathématique prouve hors de tout doute que  $1 + 1 = 2$ .  
Si les mathématiques sont vrais, alors notre projet devrait aller.