



**Cégep Limoilou**

ÉLECTRONIQUE PROGRAMMABLE ET ROBOTIQUE

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

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

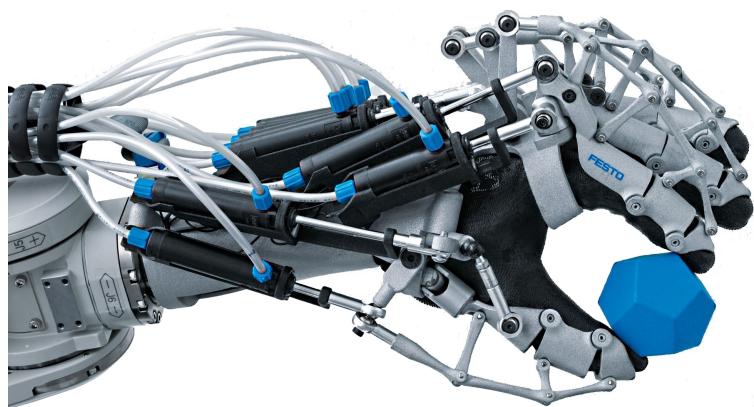
---

### 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 !

9 décembre 2014

## Table des matières

<b>1 Présentation du projet</b>	<b>5</b>
1.1 Explication du projet . . . . .	5
1.2 Schéma bloc d'ensemble du système . . . . .	5
1.2.4 Schéma bloc de la station de pesée . . . . .	6
1.2.1 Schéma bloc du bolide . . . . .	7
1.2.2 Schéma bloc de la table FESTO . . . . .	8
1.2.3 Schéma bloc du SOC8200 . . . . .	9
1.2.5 de la station no.1 . . . . .	10
1.3 Liste des logiciels . . . . .	10
1.4 Liste des trames . . . . .	11
<b>2 Le matériel</b>	<b>12</b>
2.1 Bloc 1 . . . . .	12
2.2 Bloc 2 . . . . .	12
2.3 Bloc 3 . . . . .	12
2.4 Bloc 4 . . . . .	12
2.5 Explication des types de liens . . . . .	12
2.5.1 RS232 . . . . .	12
2.5.2 Xbee . . . . .	12
2.6 Explication des trames . . . . .	13
2.6.1 RS-232 . . . . .	13
2.6.2 CAN . . . . .	14
2.6.3 XBEE . . . . .	16
2.7 Liste des pièces . . . . .	16
2.7.1 Liens web . . . . .	17
2.7.2 Datasheets . . . . .	18
<b>3 Interface PC</b>	<b>20</b>
3.1 Gestion de l'historique . . . . .	22
3.1.1 Exemple d'historique typique . . . . .	22
3.2 Structure du programme . . . . .	22
3.2.1 Les Ghosts Labels . . . . .	22
3.3 Explication des trames . . . . .	22
3.4 Ordre de gestion des tâches . . . . .	22
<b>4 Logiciel du SOC8200</b>	<b>23</b>
4.1 Description du programme . . . . .	23
4.2 Schéma bloc du script shell . . . . .	23
4.3 Gestion des processus et du temps de CPU . . . . .	23
4.4 Format et récupération des logs . . . . .	23
4.5 Liste des tests et logiciels . . . . .	23
<b>5 Logiciel de la station 1 et du bolide</b>	<b>24</b>
5.1 La station no.1 . . . . .	24
5.2 La station no.2 . . . . .	24
5.4 Le bolide . . . . .	25
5.3 Schéma des héritages de classes . . . . .	26
5.5 Procédure de compilation sur IAR . . . . .	27
5.6 Procédure de vérification . . . . .	27

<b>6 Logiciel du module PIC18F258</b>	<b>28</b>
6.1 Description du fonctionnement du programme . . . . .	28
6.2 Procédure de compilation sur MPLAB . . . . .	28
6.3 Procédure de vérification . . . . .	28
<b>7 Schémas OrCAD</b>	<b>29</b>
<b>8 Fichiers Gerbers</b>	<b>33</b>
<b>9 Calculs</b>	<b>36</b>
9.1 Calcul du pas de conversion de la pile . . . . .	36
9.2 Calcul du baudrate . . . . .	36
<b>10 Conclusions</b>	<b>37</b>
10.1 Ce que le projet m'a apporté . . . . .	37
10.1.1 Vincent . . . . .	37
10.1.2 Hicham . . . . .	37
10.1.3 Gabriel . . . . .	37
10.1.4 Louis-Norman . . . . .	37
10.2 Difficultés et corrections . . . . .	37
10.2.1 Vincent . . . . .	37
10.2.2 Hicham . . . . .	37
10.2.3 Gabriel . . . . .	37
10.2.4 Louis-Norman . . . . .	37
10.3 Ce que j'ai aimé ou pas . . . . .	37
10.3.1 Vincent . . . . .	37
10.3.2 Hicham . . . . .	37
10.3.3 Gabriel . . . . .	37
10.3.4 Louis-Norman . . . . .	37
<b>11 ANNEXE 1 : Code source du programme pour PC</b>	<b>38</b>
<b>12 ANNEXE 2 : Code source du Bolide et de la station 1</b>	<b>39</b>
<b>13 ANNEXE 4 : Code source de la table FESTO</b>	<b>40</b>
<b>14 ANNEXE 5 : Code source du programme PIC</b>	<b>41</b>
<b>15 ANNEXE 4 : Script Shell du SOC8200</b>	<b>42</b>

## Table des figures

1	Vue d'ensemble du projet . . . . .	5
2	Schéma bloc du Bolide . . . . .	7
3	Schéma bloc de la table FESTO . . . . .	8
4	Schéma bloc du SOC8200 . . . . .	9
5	Programme de contrôle principal . . . . .	20
6	Options CAN avancées . . . . .	21
7	Historique des actions . . . . .	22
8	Héritage de la classe de contrôle du véhicule . . . . .	26
9	Héritage de la classe de contrôle de la station no.1 . . . . .	26
10	Qui hérite du SPI ? . . . . .	26
11	Qui hérite de l'I2C ? . . . . .	26
12	Choix de la cible sur IAR . . . . .	27
13	Compiler avec MicroC PRO pour PIC . . . . .	28
14	Schémas carte IO I2C, page 1 . . . . .	29
15	Schémas carte IO I2C, page 2 . . . . .	30
16	Schémas carte IO I2C, page 3 . . . . .	31
17	Schémas carte IO I2C, page 4 . . . . .	32
18	Couche TOP . . . . .	33
19	Couche BOT . . . . .	33
20	Silk Screen TOP . . . . .	34
21	Solder mask TOP . . . . .	34
22	Drill . . . . .	35
23	Correctif . . . . .	35

## Liste des tableaux

1	Index des identifiants matériel CAN . . . . .	14
2	Index des trames CAN . . . . .	14
3	Index des communications CAN . . . . .	14
4	Informations sur le bus I2C du bolide . . . . .	25

# 1 Présentation du projet

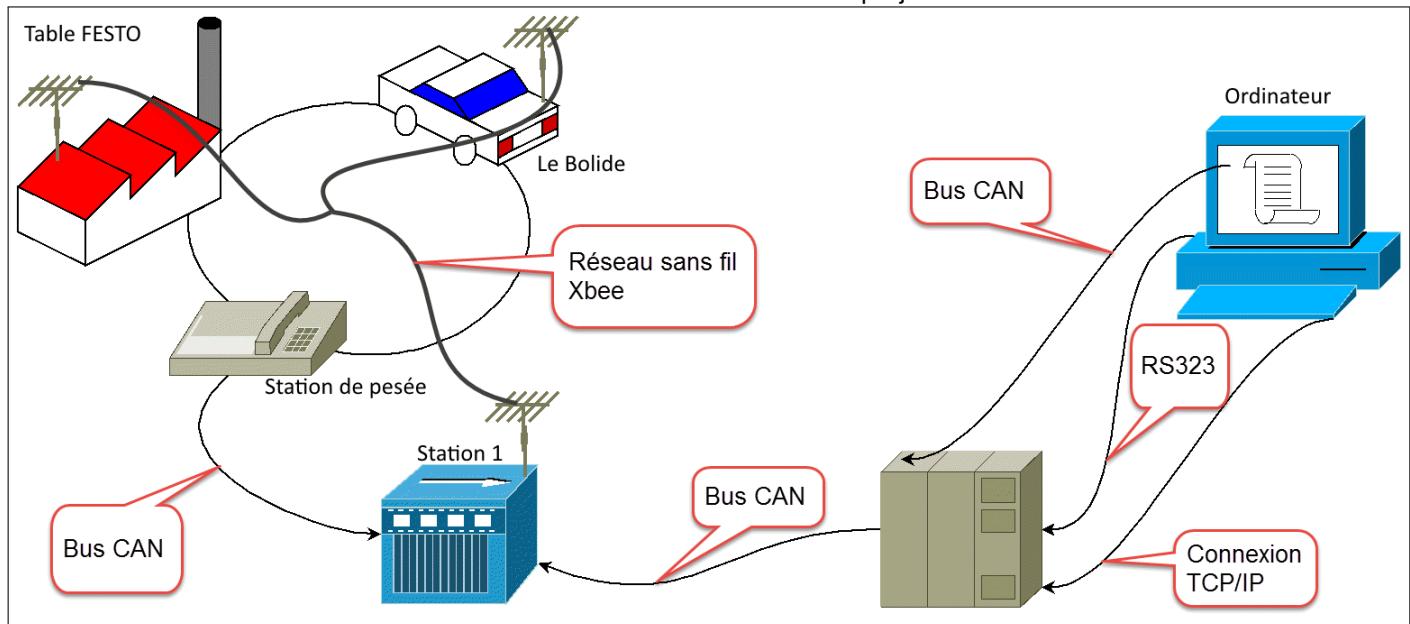
Le projet de la cinquième session consiste à réaliser (mettre un cours extrait des consignes)

- ⇒ Le Bolide
- ⇒ Carte Dallas DS89C450
- ⇒ Carte uPSD 3254A
- ⇒ SOC8200
- ⇒ Table FESTO
- ⇒ Carte PIC machin-chose-binouche
- ⇒ 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 d'ensemble du système

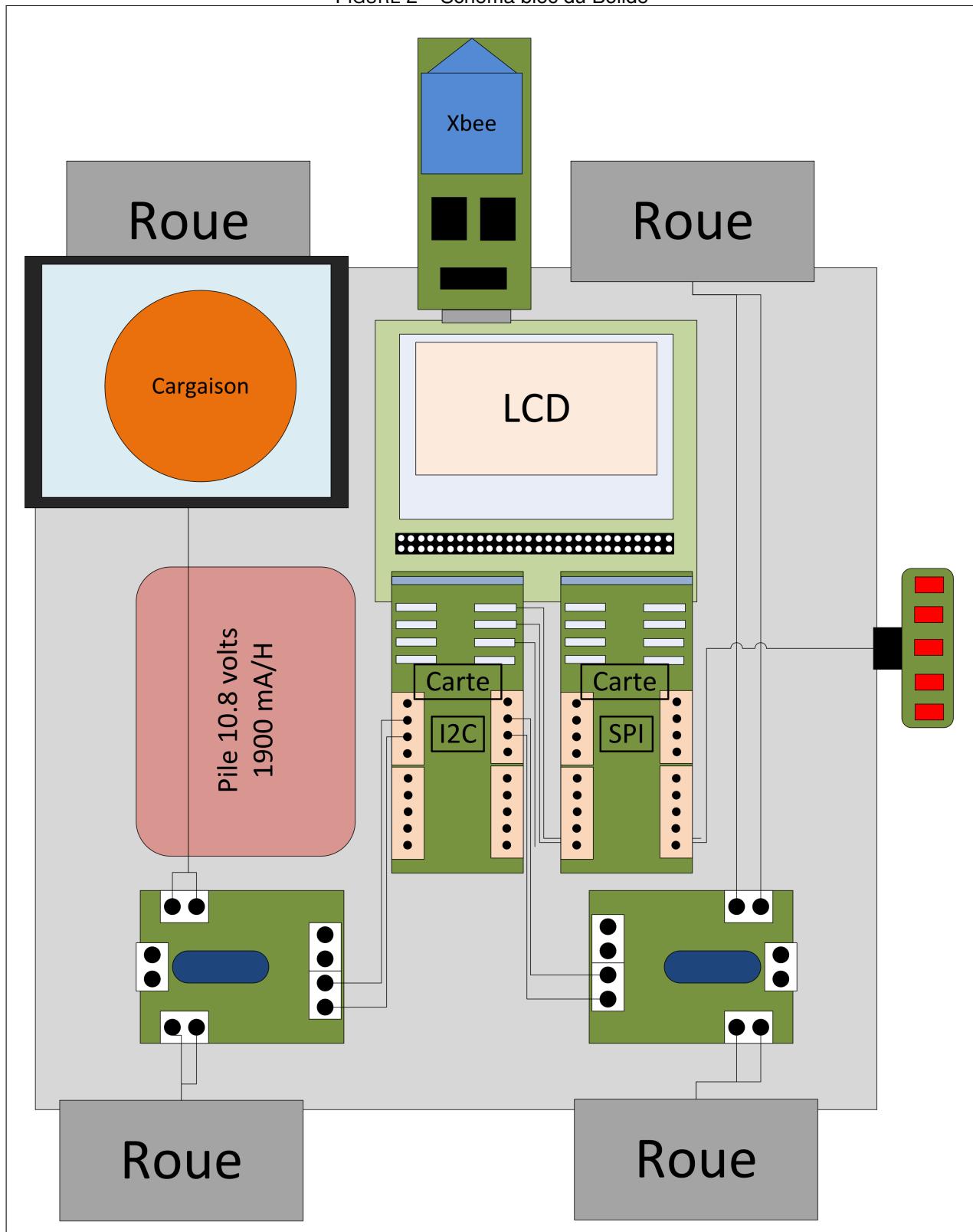
FIGURE 1 – Vue d'ensemble du projet



#### 1.2.4 Schéma bloc de la station de pesée

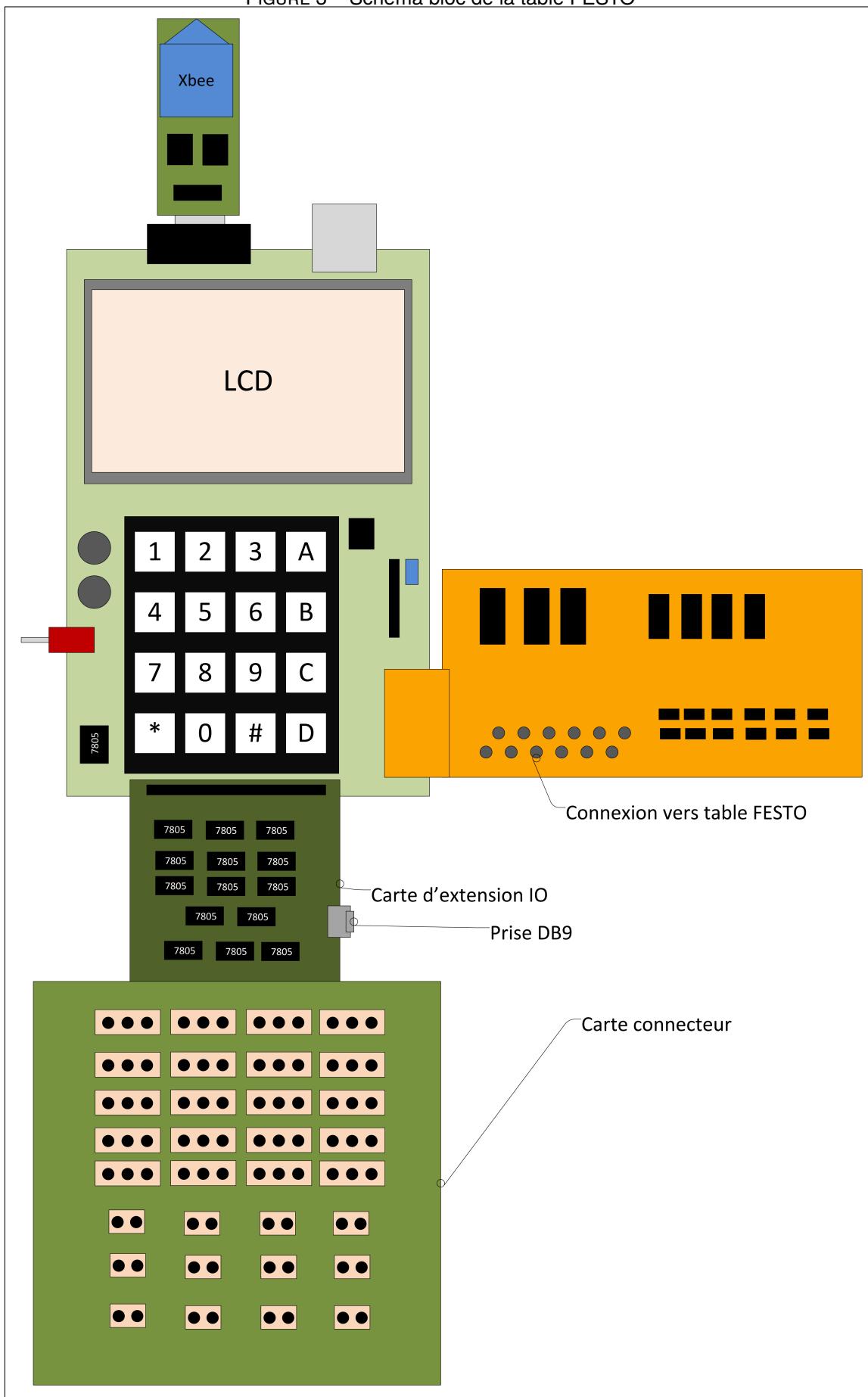
### 1.2.1 Schéma bloc du bolide

FIGURE 2 – Schéma bloc du Bolide



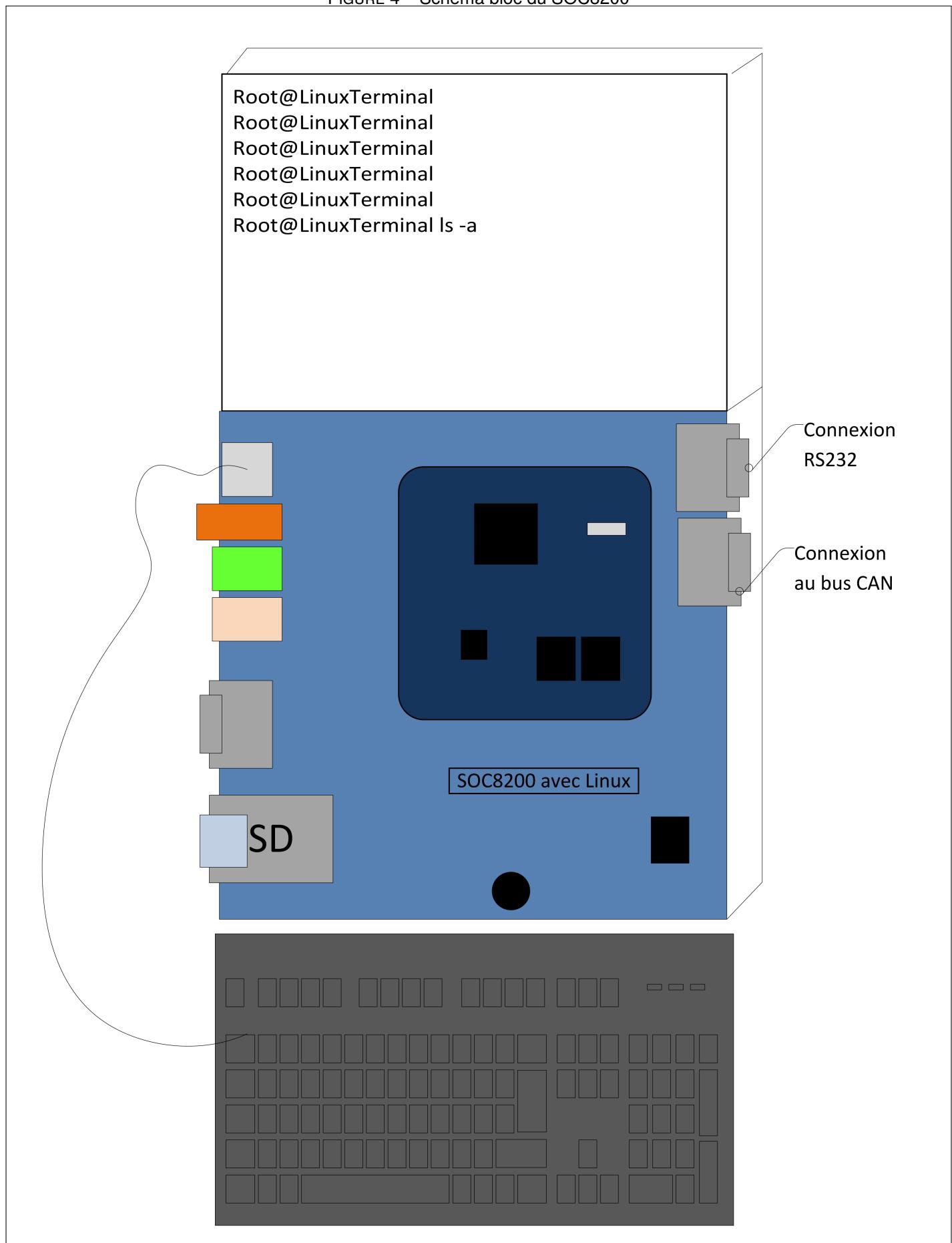
### 1.2.2 Schéma bloc de la table FESTO

FIGURE 3 – Schéma bloc de la table FESTO



### 1.2.3 Schéma bloc du SOC8200

FIGURE 4 – Schéma bloc du SOC8200



### 1.2.5 de la station no.1

## 1.3 Liste des logiciels

---

### Terminaux

- UART Master 0.97
- Serializ3r 1.0.2
- TerraTerm
- Putty
- GTKterm 0.99.7-rc1
- Terminator
- CAPS
- tinyBootloader

### Gestionnaires de projet

- MS Project 2010
- Git Hub

### Compilateurs et IDE

- Visual Studio 2013

- Visual Studio 2010

- IAR 8.20

- MPLAB X version 6.00

### Éditeurs de texte

- Notepad++
- BowPad
- medit 1.2.0

### Schémas électriques

- OrCAD 16.2

### Systèmes d'exploitation

- Windows 7 SP1

- Windows 8.1

- Windows XP SP3

- CentOS

- Arch Linux

### Autres

- VMWare Workstation 10
- TeXmaker 4.3
- Dukto R6
- Dia
- Microsoft Visio 2013
- Festo configuration tool
- L<sup>A</sup>T<sub>E</sub>X(avec plug-in Doxygen)

## 1.4 Liste des trames

f

## 2 Le matériel

### 2.1 Bloc 1

D'un point de vue matériel, le bloc 1 est le plus simple, car il est composé d'un ordinateur Windows muni 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 composante 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

Le bloc 3 est composé d'une carte PIC (mettre modèle) et d'une balance (mettre modèle.) Comme son nom l'indique, la station de pesée pèse le bloc et envoie l'information (le poids) au PC et au SOC8200 via le bus CAN. La carte PIC fait office de convertisseur CAN vers RS232.

### 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'envoi 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.

- 
1. Sous forme de trames
  2. sous forme de trames CAN
  3. Via Xbee
  4. Idem

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

## 2.6.2 CAN

Chaque composante matérielle, du Bolide au PC, dispose d'un identifiant CAN unique allant de 000 à 005. Chaque fonctionnalité dispose d'un code d'identification suivi de deux octets de données à transmettre.

TABLE 1 – Index des identifiants matériel CAN

<b>Device</b>	<b>ID matériel</b>
Ordinateur	000
SOC8200	001
Station 1	002
Station 2	003
Véhicule	004
Station de pesés	005

TABLE 2 – Index des trames CAN

<b>Fonctionnalité</b>	<b>Identifiant</b>	<b>Données</b>
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 0xFF
Batterie	0x03	0x00 à 0xFF
Couleur du bloc	0x04	0x00 à 0x02
Poids du bloc	0x05	0x00 à 0xFF
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

<b>Émetteur</b>	<b>Action</b>	<b>ID receveur</b>	<b>Donnée envoyée</b>	<b>Récepteur</b>	<b>Erreur</b>
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 batterie	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 présents 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.4, 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.

Note : mettre image d'un Xbee

## 2.7 Liste des pièces

---

- Carte Dallas
  - Carte uPSD
  - SOC 8200 (avec clavier et écran)
  - PIC18FmachinTruc
  - Carte d'extension SPI
  - Carte d'extension I2C
  - Carte CAN MCP2515
  - Xbee
  - Table FESTO
  - Carte connecteur IO 24 volts
  - Carte d'extension DAC ADC
  - Carte Xbee vers DB9
  - Câble Ethernet croisé
  - Câble Ethernet régulier
  - Câble DB9
  - Le Bolide
  - SaberTooth motor drive 2x5
  -
-

## 2.7.1 Liens web

Mettre ien vers GitHUB

DS89C450

<http://datasheets.maximintegrated.com/en/ds/DS89C430-DS89C450.pdf>

uPSD3254

[http://www.datasheetcatalog.com/datasheets\\_pdf/U/P/S/D/UPSD3254.shtml](http://www.datasheetcatalog.com/datasheets_pdf/U/P/S/D/UPSD3254.shtml)

DAC65574

[http://www.datasheetcatalog.com/datasheets\\_pdf/D/A/C/6/DAC6574.shtml](http://www.datasheetcatalog.com/datasheets_pdf/D/A/C/6/DAC6574.shtml)

LM3914

[http://www.datasheetcatalog.com/datasheets\\_pdf/L/M/3/9/LM3914.shtml](http://www.datasheetcatalog.com/datasheets_pdf/L/M/3/9/LM3914.shtml)

PCF8574

[http://www.datasheetcatalog.net/datasheets\\_pdf/P/C/F/8/PCF8574.shtml](http://www.datasheetcatalog.net/datasheets_pdf/P/C/F/8/PCF8574.shtml)

OPT101

[http://www.datasheetcatalog.com/datasheets\\_pdf/O/P/T/1/OPT101.shtml](http://www.datasheetcatalog.com/datasheets_pdf/O/P/T/1/OPT101.shtml)

SeberTooth 2x5

## 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

## LineSensors

PCF8573

DAC6574

The figure shows the front page of the DAT4574 Datasheet. It features the Texas Instruments logo at the top left, followed by the part number 'DAT4574' and its package type '16-pin PLCC'. The title 'QUAD, 10-BIT LOW-POWER, VOLTAGE OUTPUT, I<sub>C</sub> INTERFACE DIGITAL-TO-ANALOG CONVERTER' is centered above a detailed feature list. Below the features is a large application section with a block diagram of the device's internal architecture.

## LM3914

TEXAS  
INSTRUMENTS

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.

74HC14

## LineSensors

### Registers:

You read the ADC fine sensor by issuing an ADC read of the programmed address. A single byte representing the status of the sensors is returned, ranging from 0 to 31, where 0 means 16 sensors are white and 31 means 16 sensors see black.

Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Decimal Meaning
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	1	0	12
1	0	1	1	0	13
0	1	1	1	0	14
1	1	1	1	0	15
0	0	0	0	1	16
1	0	0	0	1	17
0	1	0	0	1	18

PCF8573

DAC6574

The figure shows a detailed block diagram of the DAT4517. At the top, a 'Digital Input' section includes a 'Digital Filter' and a 'Digital-to-Analog Converter'. The DAC output is connected to a 'Digital-to-Optical Converter' (labeled 'Digital-to-Optical'). This converter has two outputs: one to an 'Optical Transmitter' and another to a 'Digital-to-Optical Receiver'. The receiver's output is connected to a 'Digital-to-Analog Converter' (labeled 'Digital-to-Analog'), which then feeds into an 'Analog Filter'. Below the digital sections, there is a 'Power Management' block containing a 'Power Control' section with a 'Power Control Logic' and a 'Power Control DAC'. A 'Power Control DAC' also receives input from the 'Digital-to-Optical Receiver'. The 'Power Management' block is connected to an 'AC/DC Power Supply' and an 'AC/DC Power Supply' for the 'Digital-to-Optical Receiver'. The 'Digital-to-Optical Receiver' also receives power from an 'AC/DC Power Supply'. The 'Digital-to-Optical Receiver' and 'Digital-to-Analog Converter' are connected to a 'Digital-to-Optical Converter' at the bottom, which is labeled 'Digital-to-Optical'.

LM3914

uPSD3254A

DS89C450

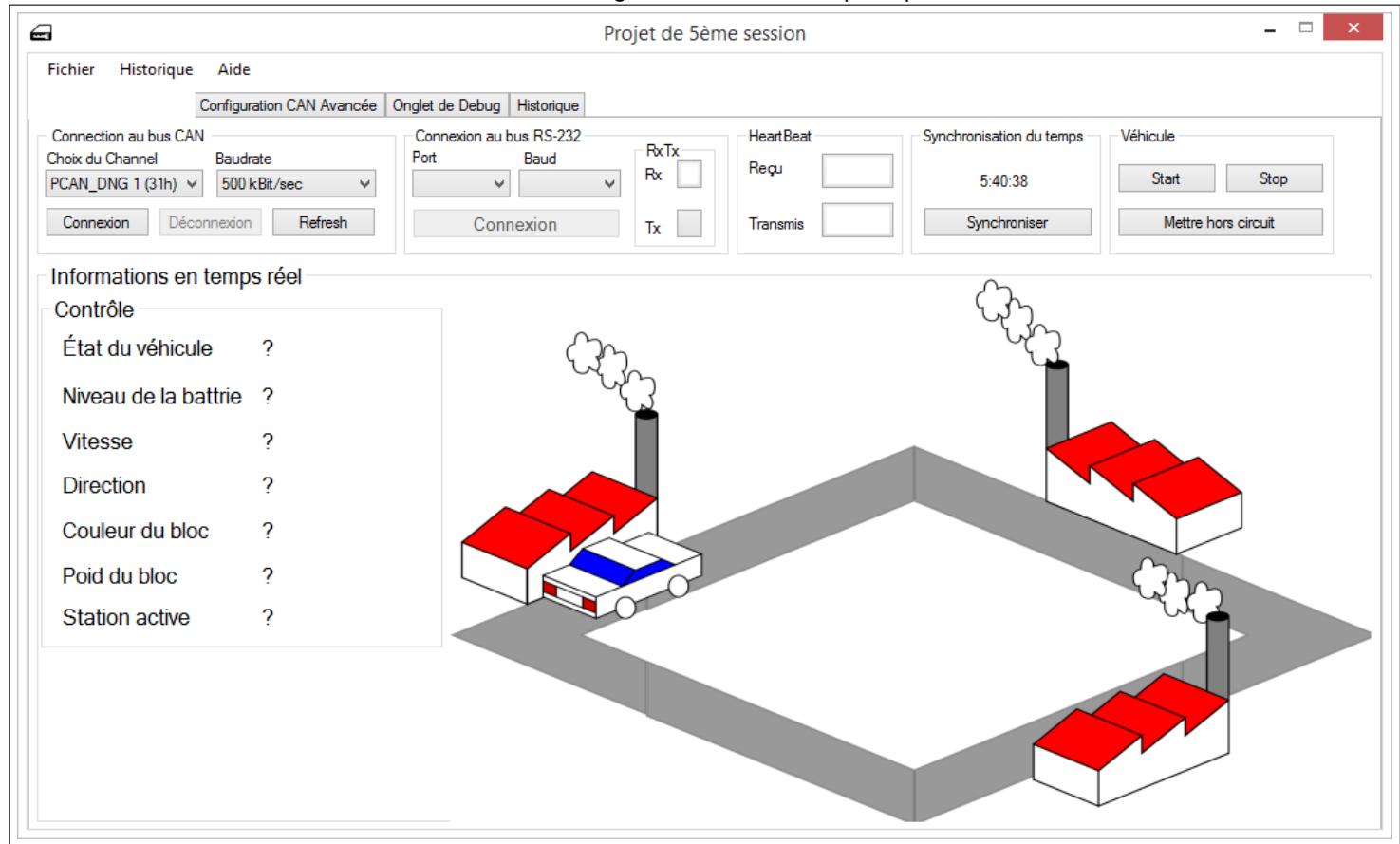
OPT101

MONOLITHIC PHOTODIODE AND SINGLE-SOURCE SUPPLY TRANSIMPEDANCE AMPLIFIER	
<b>FEATURES</b>	<b>DESCRIPTION</b>
<ul style="list-style-type: none"> <li>• SINGLE SUPPLY +20 to -10V</li> <li>• PHOTOELECTRIC DETECTION WITH HIGH GAIN</li> <li>• INTERNAL 100 MOHM RESISTOR</li> <li>• HIGH RESPONSIVENESS (640NM/550NM)</li> <li>• BANDWIDTH 100 KHZ</li> <li>• LOW QUIESCENT CURRENT: 15MA</li> <li>• VARIOUS SENSITIVITY LEVELS</li> <li>• PLASTIC LEADLESS CHIP CARRIER</li> <li>• SURFACE-MOUNT PACKAGES</li> </ul>	<p>The OPT101 is a monolithic photodiode with resulting transimpedance amplifier. The device is designed for use with light intensity, the optoelectric detector is designed to operate at visible wavelengths. The internal circuitry is designed to provide a high input impedance and a low quiescent current.</p> <p>The OPT101 is a combination of photodiode and transimpedance amplifier in single chip allowing the elimination of external components such as resistors, capacitors, and biasing resistors. The device can be used for logic gate circuits, noise pick-up, and gain reading. The OPT101 is also suitable for applications such as barcode readers, optical scanners, smoke detectors, and current changers.</p>
<b>APPLICATIONS</b>	
<ul style="list-style-type: none"> <li>• MEDICAL INSTRUMENTATION</li> <li>• LASER POSITION DETERMINATION</li> <li>• POSITION AND POSITIONING SENSORS</li> <li>• OPTICAL SPECTROANALYZERS</li> <li>• BARCODE SCANNERS</li> <li>• SMOKE DETECTORS</li> <li>• CURRENT CHANGERS</li> </ul>	
<p><b>A</b> Please be aware that an important notice concerning liability, standard warranty, use in critical applications of Texas Instruments semiconductor products, and information regarding trademarks and copyrights is contained in the Terms and Conditions page of our Internet site.</p> <p><b>4</b> Information is the property of their respective owners.</p>	
 Texas Instruments www.ti.com	

## SaberTooth motor drive

### 3 Interface PC

FIGURE 5 – Programme de contrôle principal

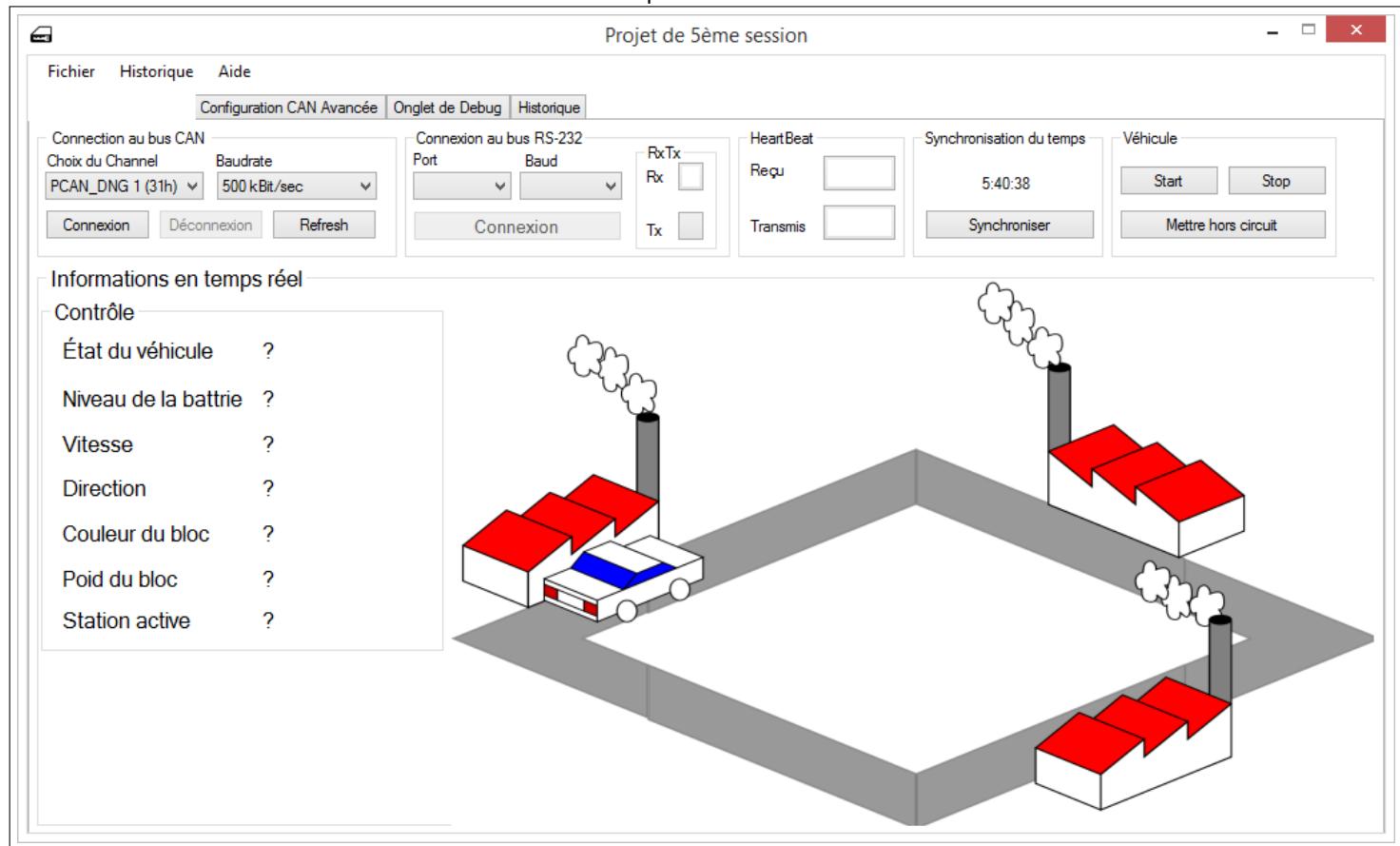


Notre programme, écrit en C# à l'aide de Visual Studio, peut se connecter au bus CAN via une carte SPI<sup>6</sup> et au bus RS232 via un câble DB9 ou USB<sup>7</sup>. La connexion RS232 sert à l'envoi et à la réception du HeartBeat afin d'informer le SOC8200 si l'ordinateur en venait à connaître une 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 », inscrire son heure de référence sur la station no.1 via le bus CAN.

6. Spécifier le fabricant

7. S'il y a présence d'un FTDI

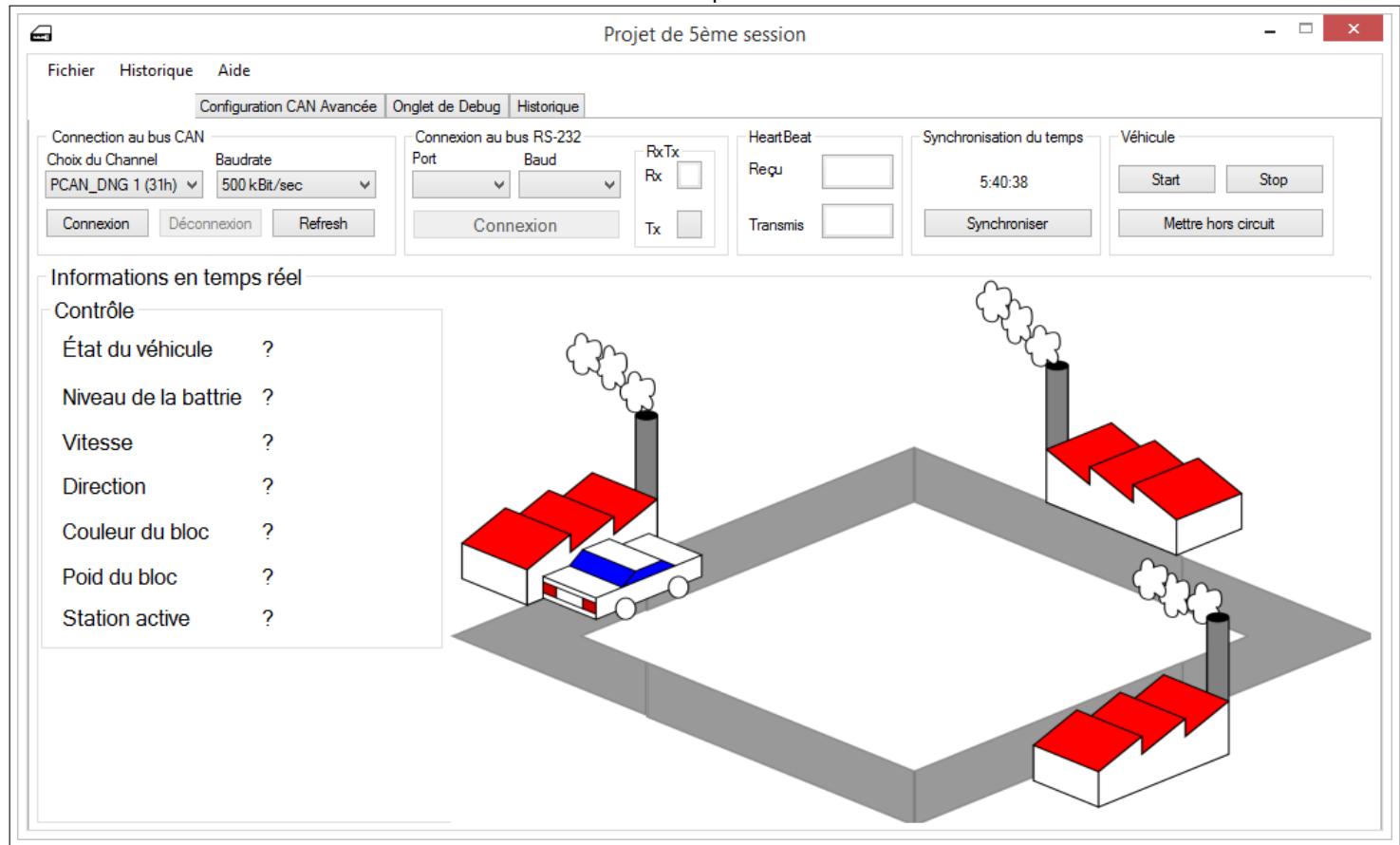
FIGURE 6 – Options CAN avancées



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

### 3.1 Gestion de l'historique

FIGURE 7 – Historique des actions



Toute action effectuée via le programme ainsi que toute donnée ayant transité sur le bus CAN, RS232 et TCP/IP est cataloguée en bonne et due forme dans un historique qu'il est possible de consulter et sauvegarder à tout moment.

#### 3.1.1 Exemple d'historique typique

Insérer copié-collé de l'historique ici

### 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 systèmes 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

D'un commun accord de l'équipe, le programme du SOC2800 est écrit en script Shell. La principale raison de ce choix est Sourcery Codebench lui-même. La gestion des projets avec Sourcery est un cauchemar. Quant à la nécessité de sauvegarder pour compiler et d'utiliser une machine virtuelle, elles ne viennent qu'aggraver la situation. De plus, son gestionnaire de licence<sup>8</sup> frustré quiconque souhaite l'utiliser. L'utilisation du script shell est à la fois plus simple et permet de faire plus en moins de temps.

### 4.2 Schéma bloc du script shell

De tous les scripts, *Projet.sh* est le maître, et contient l'équivalent du main. Ce fichier initialise le port série, le bus CAN, la lecture et l'envoi des heartbeat en asynchrone, ainsi que la lecture du clavier USB. Le script *Projet.sh* peut aussi prendre la relève du bus CAN si le PC est dans l'incapacité d'assurer ses fonctions. Quant aux autres scripts, soit *connexion.sh*, *RxCan.sh*, *tcp.sh* et *Envoi.sh*, ce ne sont que des fonctionnalités que *Projet.sh* appelle en asynchrone.

### 4.3 Gestion des processus et du temps de CPU

Le seul processus synchrone est script principal *Projet.sh*. Tous les scripts appelés par *Projet.sh* ainsi que leurs processus enfants sont exécutés en asynchrone. Lorsqu'un script doit passer un paramètre à un autre script, un fichier est créé (à l'aide de la commande echo ou cat) afin de contenir le paramètre en question. Les processus impliqués se contentent de lire des fichiers.

### 4.4 Format et récupération des logs

Toutes les trames CAN reçues sont enregistrées dans le fichier «histocan» dont voici un court aperçu :

```
can0      6  [7] 00 00 0B 04 34 00 00
can0      6  [7] 00 00 0B 04 34 00 00
can0      6  [7] 00 00 0B 04 35 00 00
can0      6  [7] 00 01 0B 04 36 00 00
can0      6  [7] 00 01 0B 04 37 00 00
can0      6  [7] 00 01 0B 04 37 00 00
can0      6  [7] 00 01 0B 04 38 00 00
can0      6  [7] 01 00 0B 04 39 00 00
```

De plus, un autre fichier (histocandate) contient l'heure et la date des trames reçues.

```
can0 6 [7] 00 00 0B 1D 08 00 00
Wed Nov  5 11:30:00 UTC 2014
can0 6 [7] 00 00 0B 27 38 00 00
Wed Nov  5 11:30:13 UTC 2014
can0 6 [7] 00 00 0B 27 38 00 00
Wed Nov  5 11:30:13 UTC 2014
can0 6 [7] 00 00 0B 27 39 00 00
```

### 4.5 Liste des tests et logiciels

8. 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 du bolide

Le programme de la station 1 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 deux 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 et ses divers capteurs.

### 5.1 La station no.1

La station no.1 est composée 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 composée de la table FESTO, de la carte uPSD, de la carte d'extensions IO que nous avons réalisées et d'un Xbee.

## 5.4 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

### 5.3 Schéma des héritages de classes

Quelles classes utilisent quelles autres classes dans notre code ?

FIGURE 8 – Héritage de la classe de contrôle du véhicule

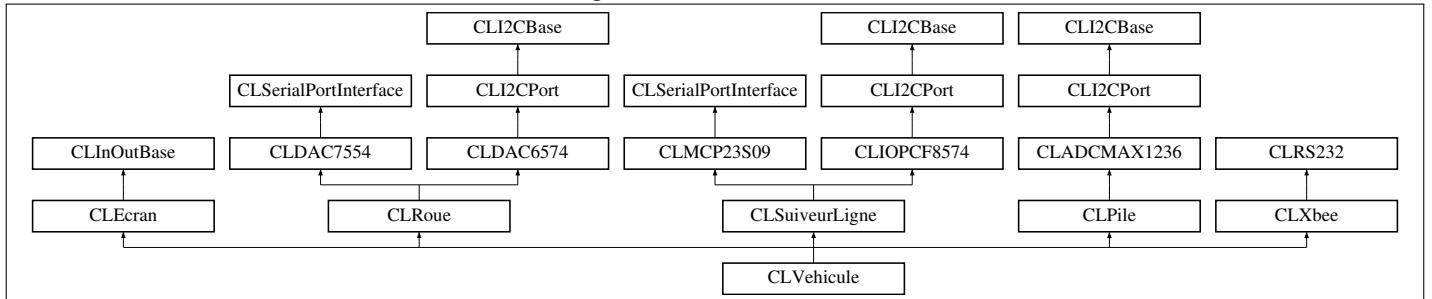


FIGURE 9 – Héritage de la classe de contrôle de la station no.1

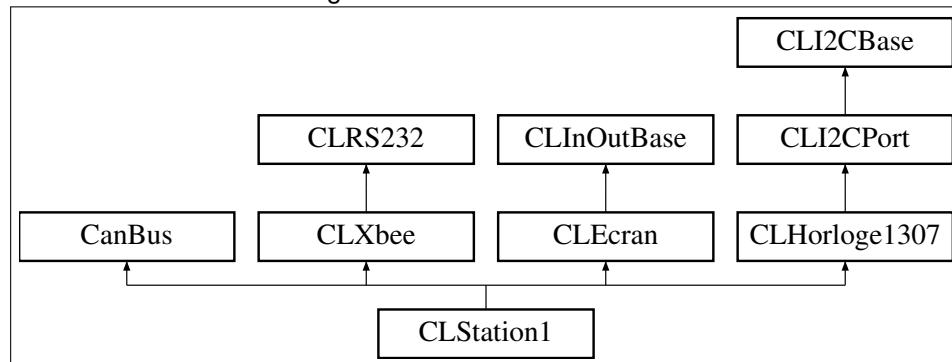


FIGURE 10 – Qui hérite du SPI ?

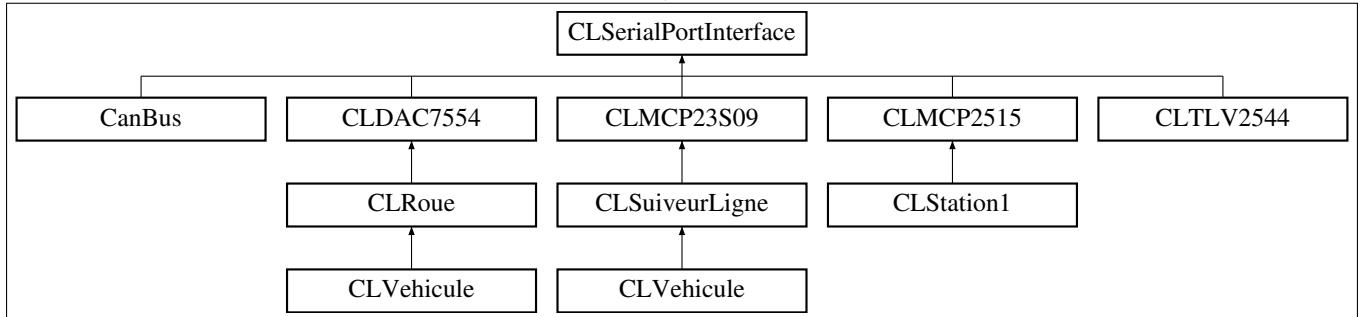
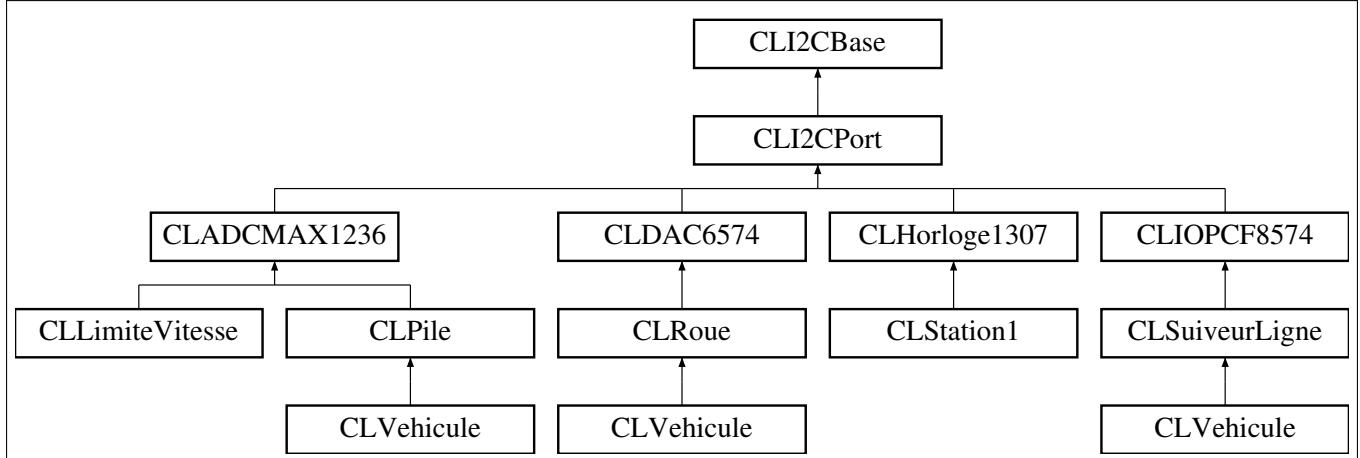


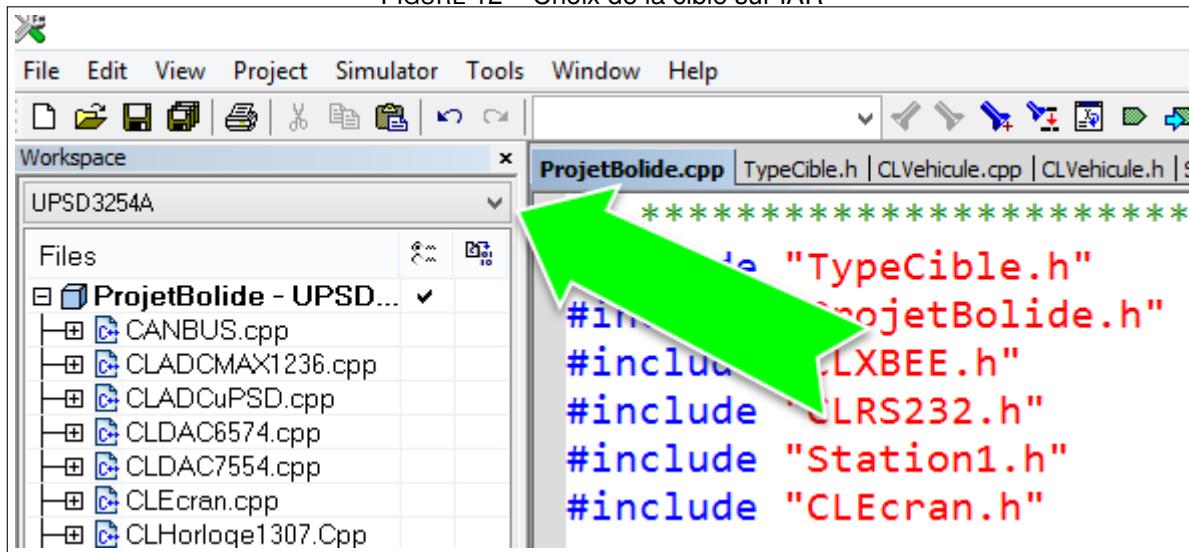
FIGURE 11 – Qui hérite de l'I2C ?



## 5.5 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. Pour lancer une compilation, rien de plus simple que d'appuyer sur F6.

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

```

## 5.6 Procédure de vérification

Pour les vérification, rien de plus simple. Il suffit d'envoyer le fichier .hex dans le microcontrôleur et d'observer visuellement le fonctionnement du montage. Certains appellent cette technique « débogage à la chandelle<sup>9</sup>. » Cela étant dit, afficher des trames et autres données sur un écran LCD aide grandement.

9. Salut Étienne !!!

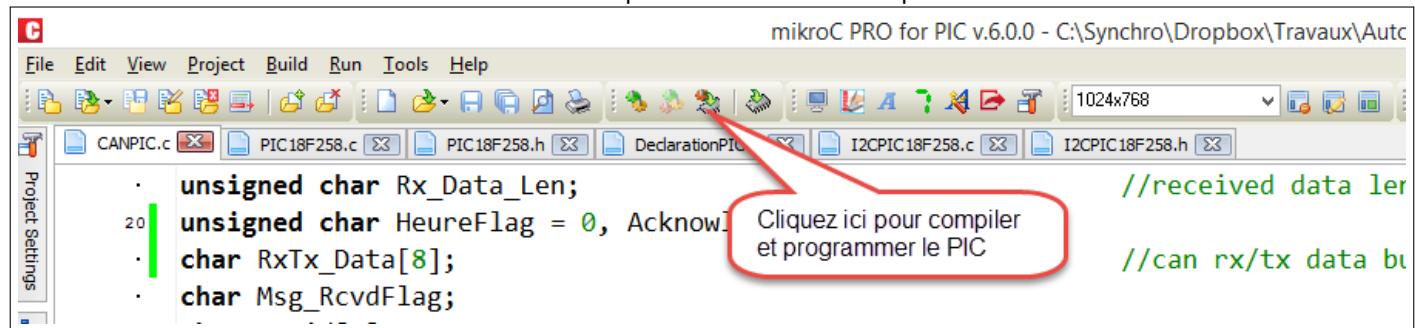
## 6 Logiciel du module PIC18F258

### 6.1 Description du fonctionnement du programme

La structure du programme est on ne peut plus classique, car il s'agit d'un programme écrit en C et disposant d'un bon vieux `while(1)` à l'intérieur du `void main(void)`. En bref, le programme initialise d'abord les différents registres du PIC, puis le bus CAN, I2C et RS232. Quant à la boucle, elle lit le poids du bloc, traduit les trames CAN en données RS232 et expédie le tout au... Où déjà ?

### 6.2 Procédure de compilation sur MPLAB

FIGURE 13 – Compiler avec MicroC PRO pour PIC



### 6.3 Procédure de vérification

## 7 Schémas OrCAD

FIGURE 14 – Schémas carte IO I2C, page 1

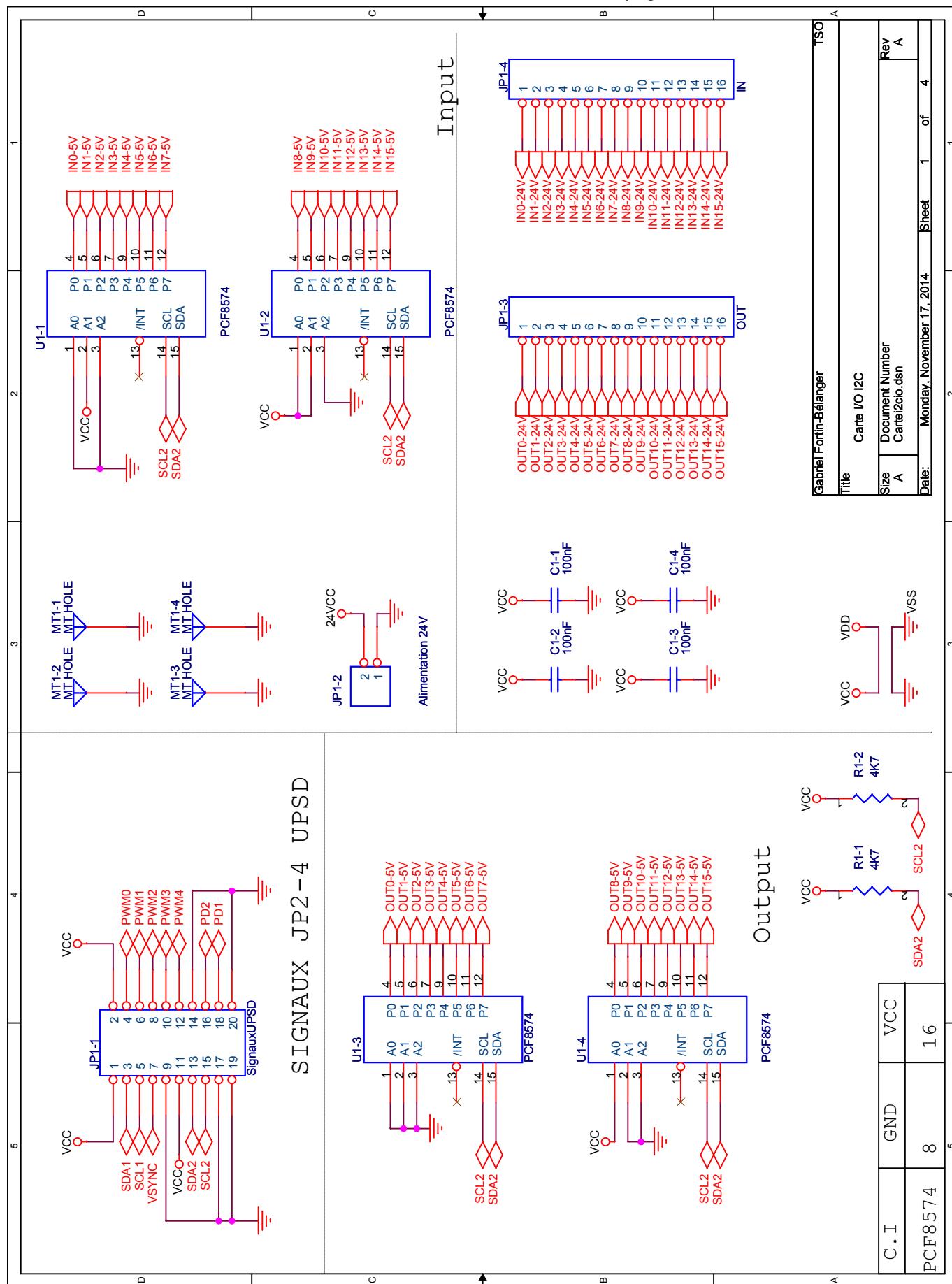


FIGURE 15 – Schémas carte IO I2C, page 2

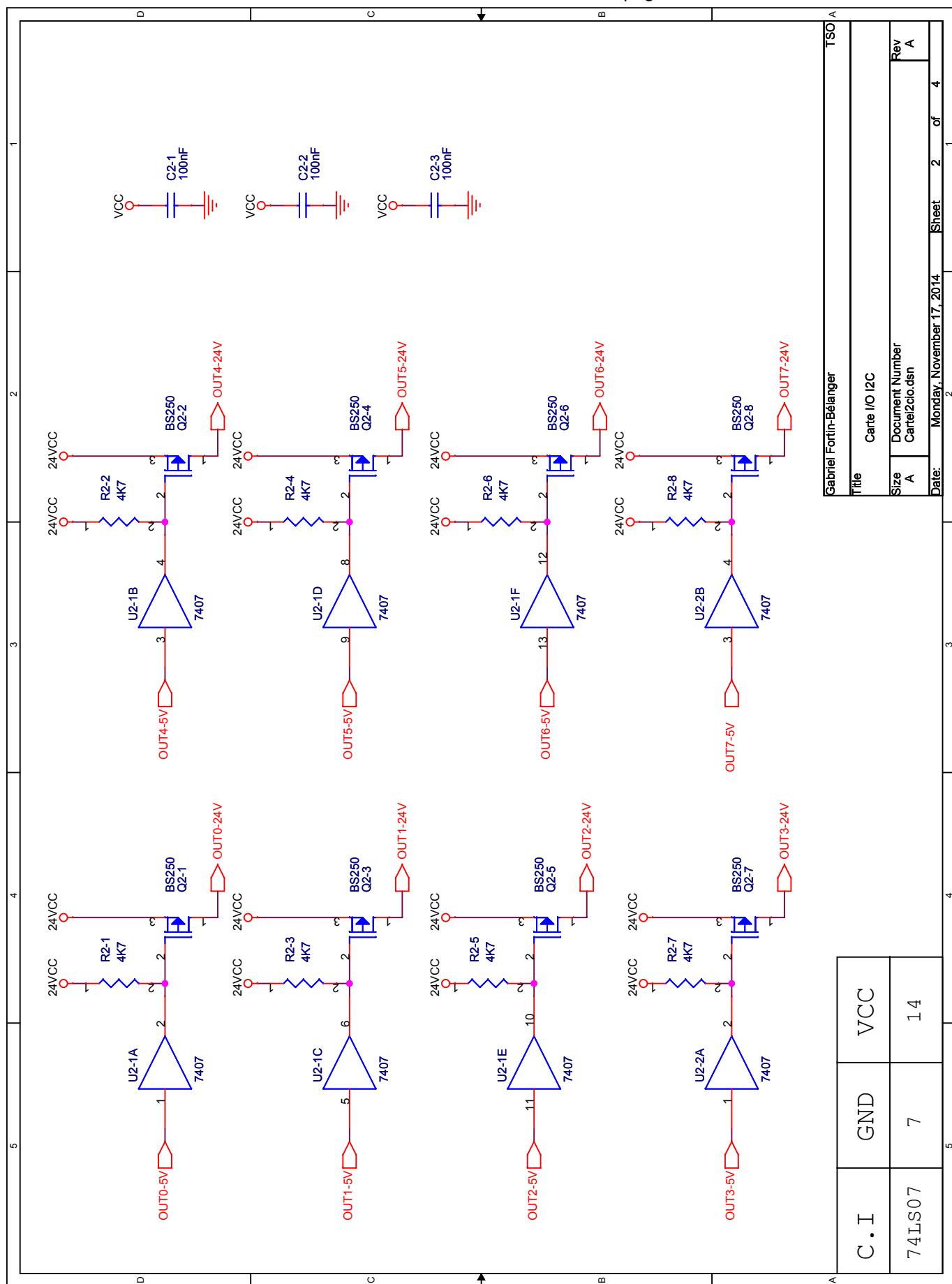


FIGURE 16 – Schémas carte IO I2C, page 3

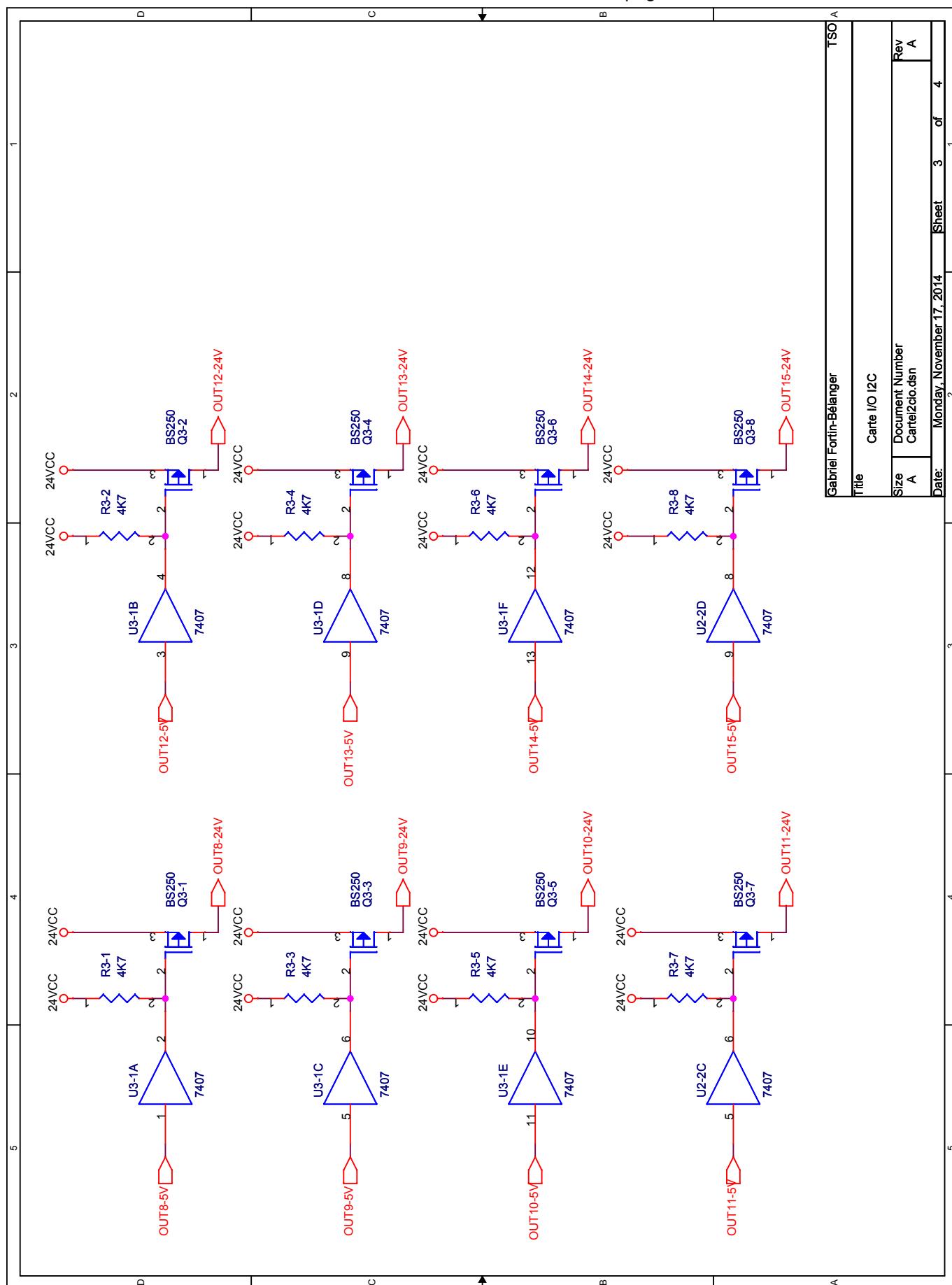
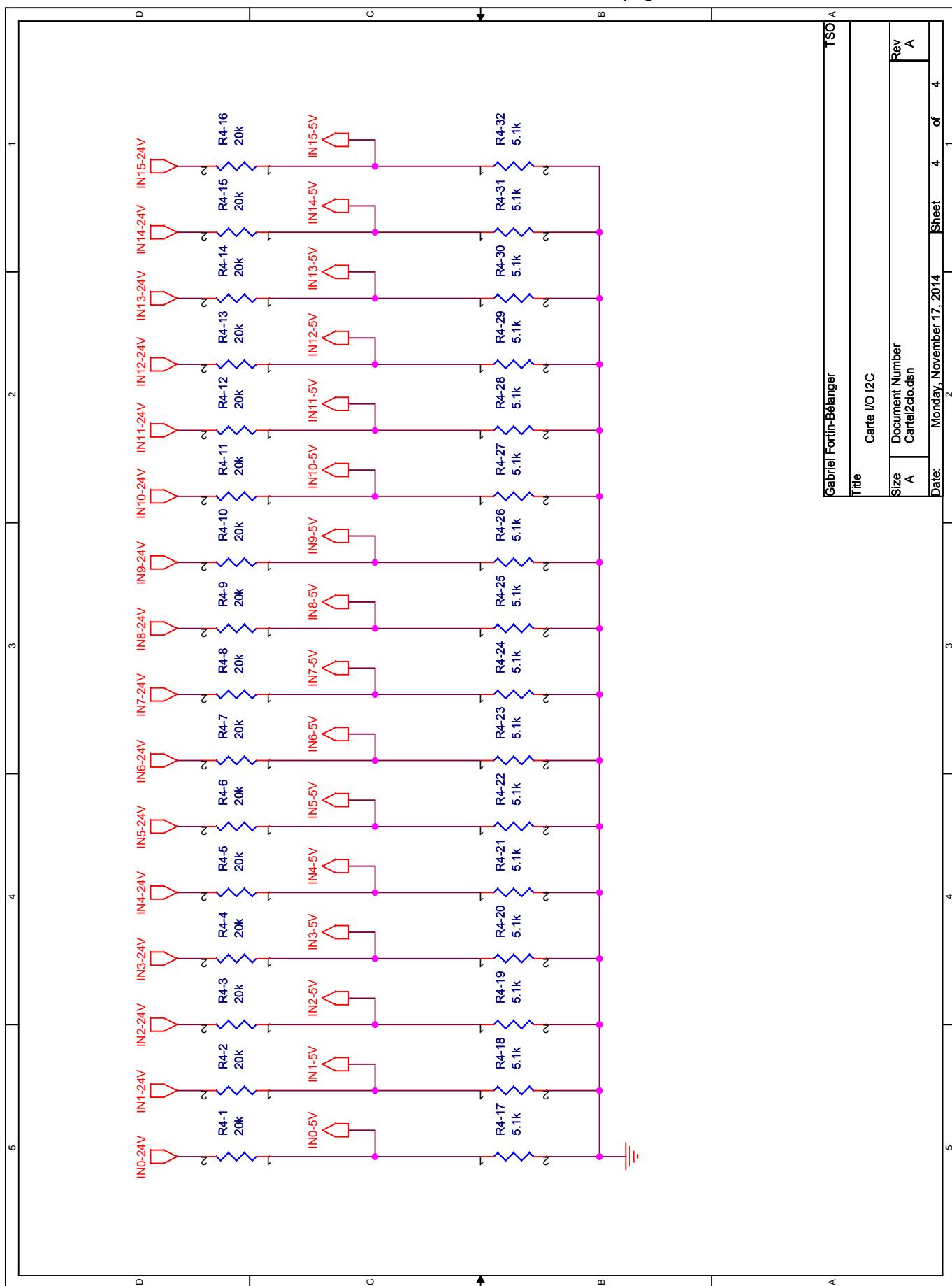


FIGURE 17 – Schémas carte IO I2C, page 4



Gabriel Fortin-Bélanger	
Title	Carte I/O I2C
Size	A
Date:	Monday, November 17, 2014
Rev	A

## 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 départementale.

FIGURE 18 – Couche TOP

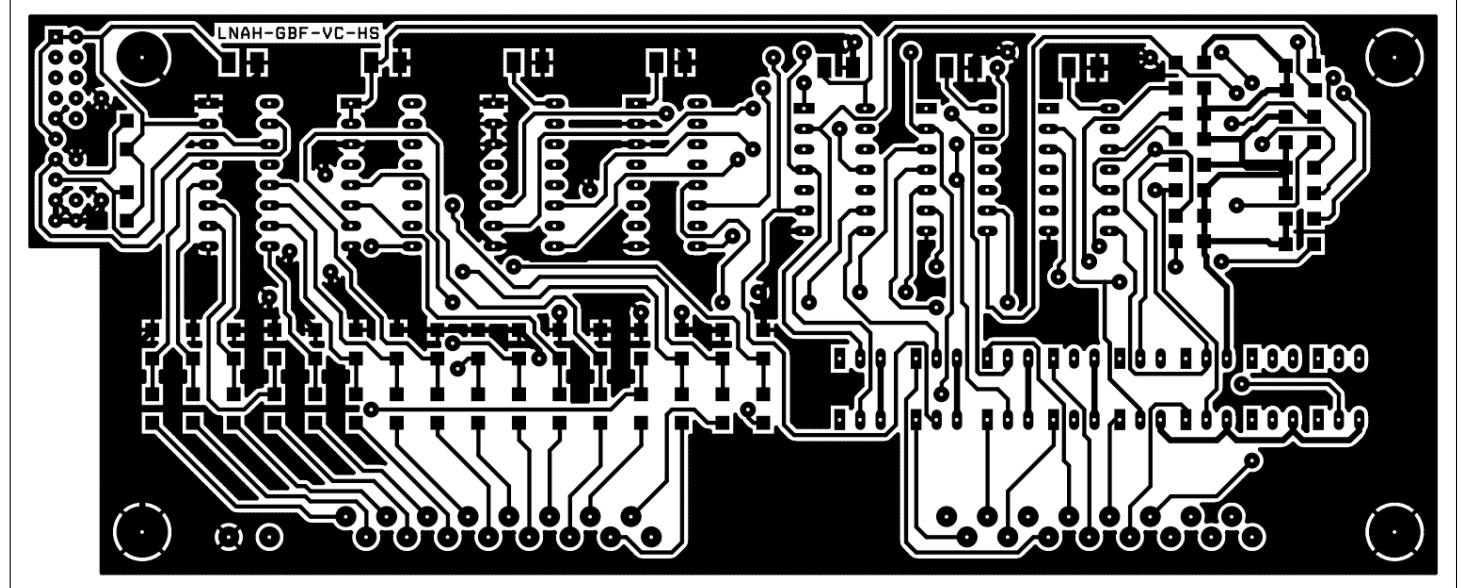


FIGURE 19 – Couche BOT

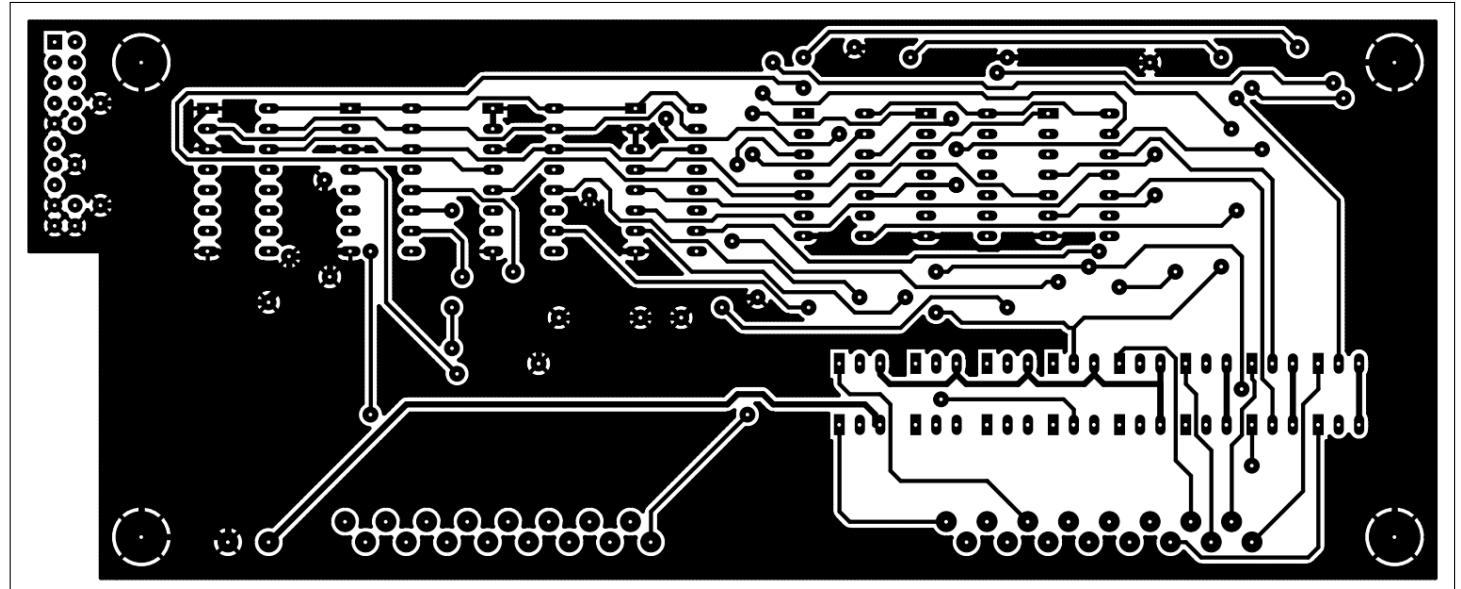


FIGURE 20 – Silk Screen TOP

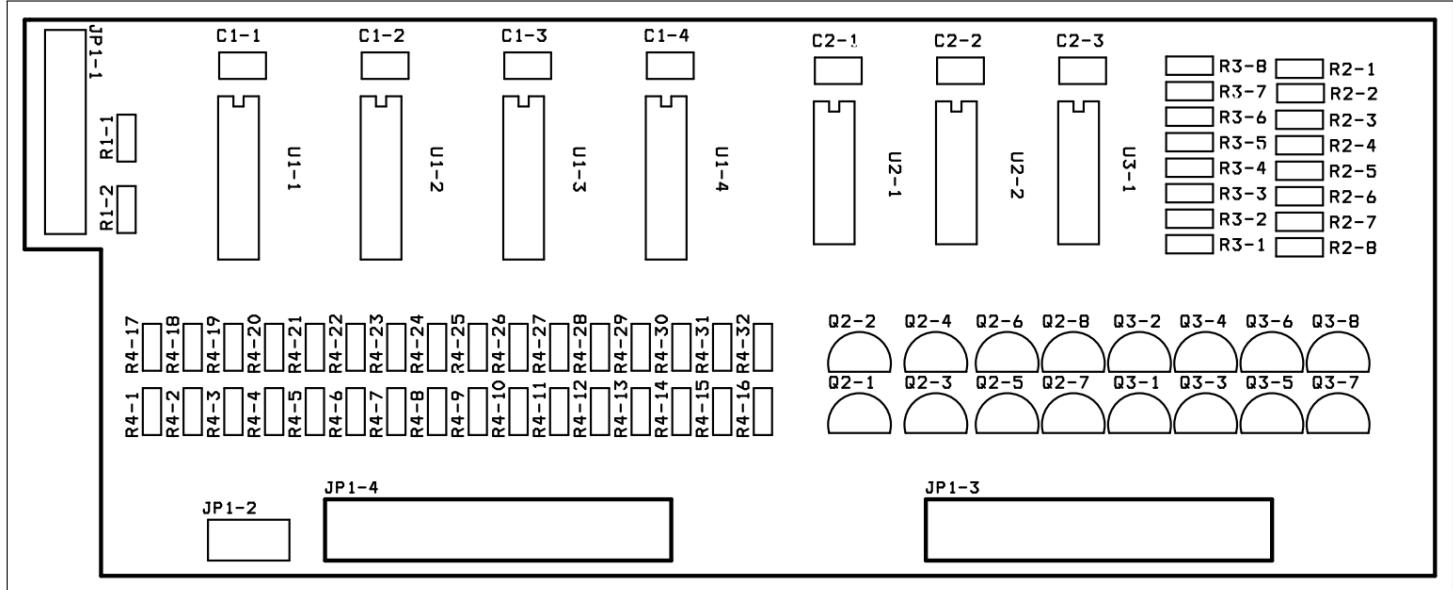


FIGURE 21 – Solder mask TOP

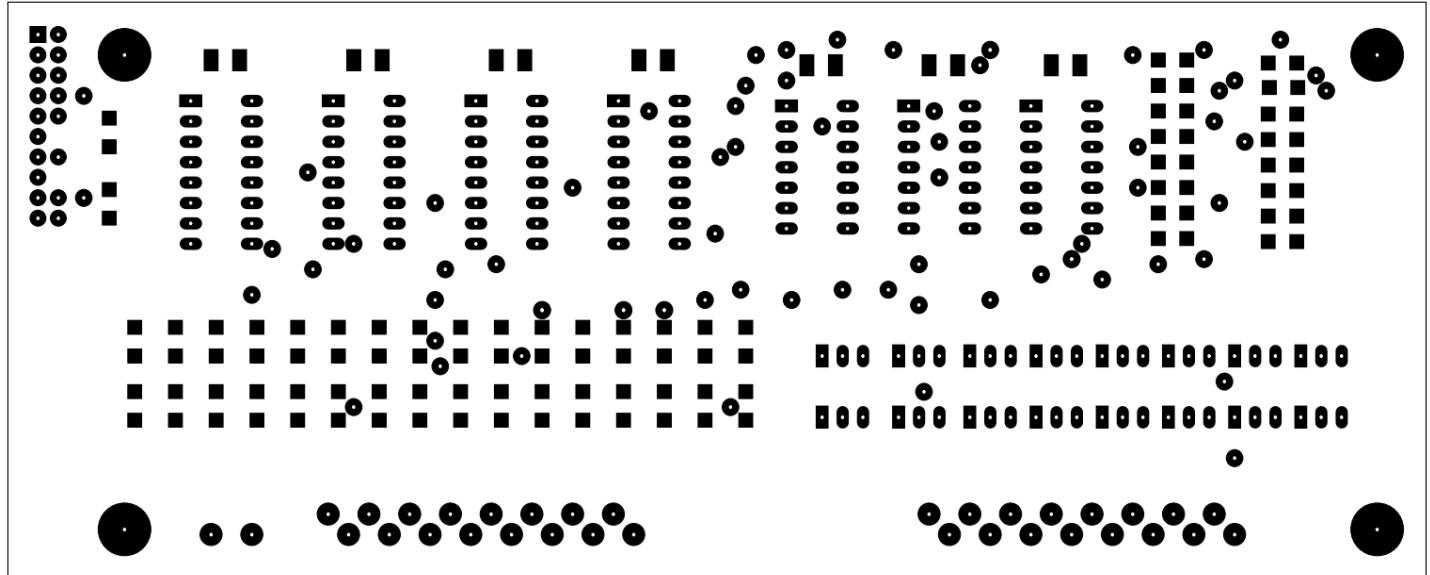


FIGURE 22 – Drill

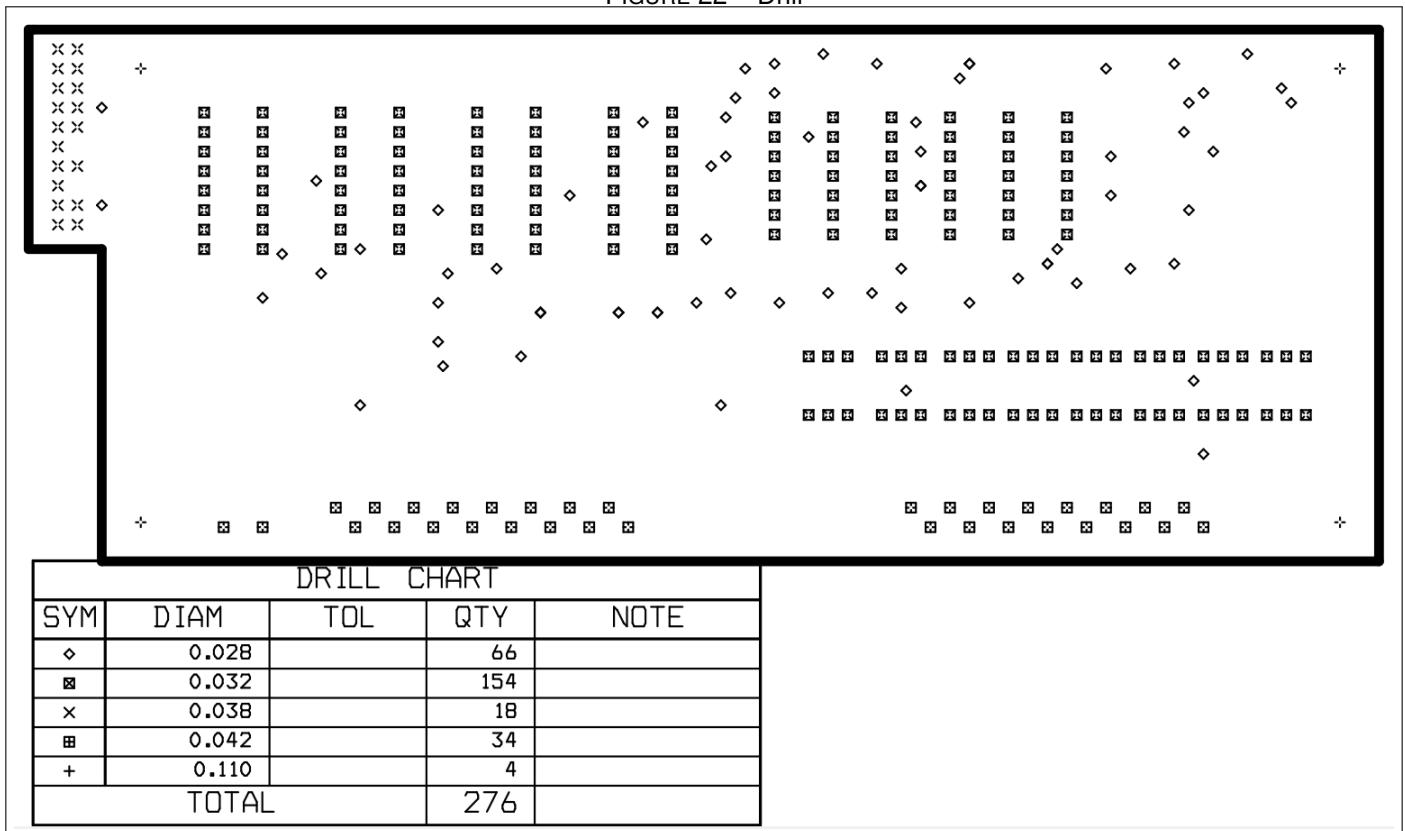
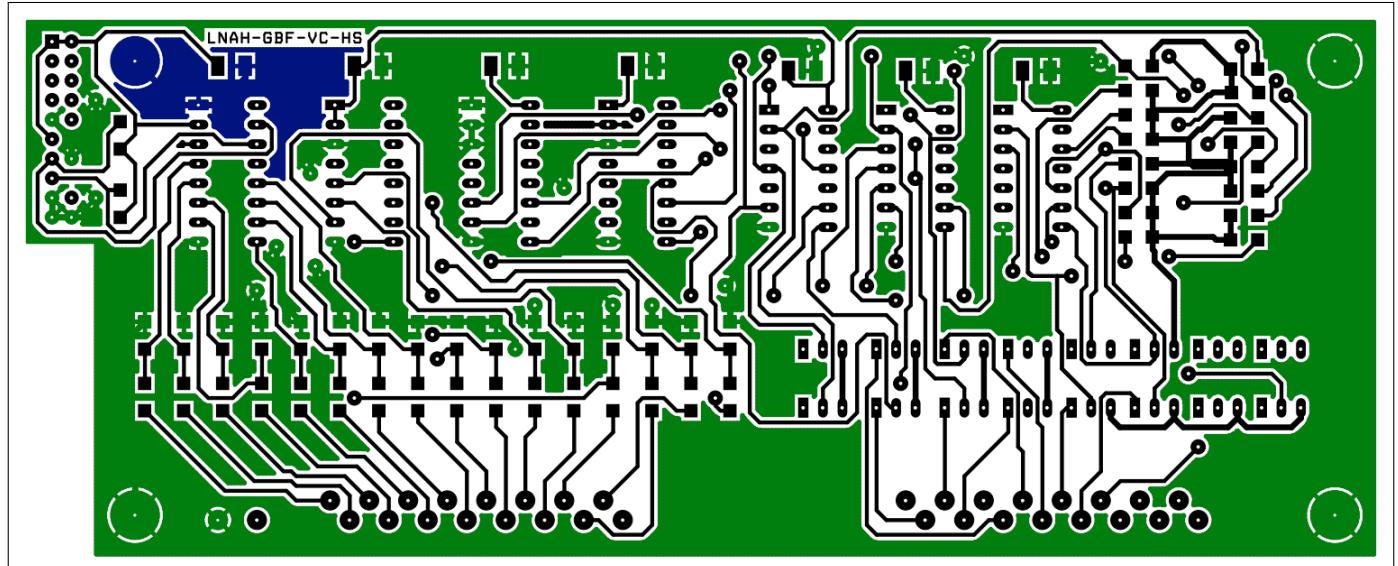


FIGURE 23 – Correctif



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. De plus, quelques ponts de soudure sont manifestés là et là, grillant de ce fait nos PCF8574. Rien d'insurmontable.

## 9 Calculs

### 9.1 Calcul du pas de conversion de la pile

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

### 9.2 Calcul du baudrate

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

Alors...

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

## 10 Conclusions

### 10.1 Ce que le projet m'a apporté

10.1.1 Vincent

dd

10.1.2 Hicham

dd

10.1.3 Gabriel

dd

10.1.4 Louis-Norman

dd

---

### 10.2 Difficultés et corrections

10.2.1 Vincent

ss

10.2.2 Hicham

ss

10.2.3 Gabriel

ss

10.2.4 Louis-Norman

ss

---

### 10.3 Ce que j'ai aimé ou pas

10.3.1 Vincent

a

10.3.2 Hicham

a

10.3.3 Gabriel

a

10.3.4 Louis-Norman

a

---

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

```
using System;
```

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

```
using System;
```

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

```
using System;
```

## 14 ANNEXE 5 : Code source du programme PIC

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

## 15 ANNEXE 4 : Script Shell du SOC8200

---

prog.sh

```
compteur=0
```

---

read.sh

```
using System;
```

---

tcp.sh

```
using System;
```

---

can.sh

```
using System;
```

---

PortSerie.sh

```
while true
do
    head -1 /dev/ttysCMA0 > ./junk
    echo "Allo" > ./hbeat
done
```

---

RxCAN.sh

```
candump can0 >> ./histocan &
nbligne=1
ancienvar=0

while true
do
    var='tail -1 ./histocan'
    if [ "$var" != "$ancienvar" ]
    then
        echo $var
        echo $var >> ./histocandate
        date >> ./histocandate
        ancienvar=$var
    fi
# nbligne='wc -l ./histocan'
done
```

---

MachinChouette.sh

```
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.