Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

Afficheur LCD pour Squart électrique

MONTAGNE Vincent, GONIAK Benoit
2éme année, groupe Q2
2007-2009

Enseignants
LEQUEU T.
GLIKSOHN B.

Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

Afficheur LCD pour Squart électrique

MONTAGNE Vincent, GONIAK Benoit
2éme année, groupe Q2
2007-2009

Enseignants
LEQUEU T.
GLIKSOHN B.

Sommaire

Introduction

Au début du dernier semestre de notre formation, nous avons choisi un projet destiné à mettre en application nos connaissances, à travailler en autonomie, à respecter un cahier des charges et à nous replacer autant que possible dans le monde de l'entreprise.

Dans le cadre de ce projet, nous avons décidé de nous intéresser à la réalisation et à la programmation d'une carte d'acquisition et d'un afficheur pour le Squart électrique.

Le Squart est un véhicule disponible à l'IUT Génie Electrique et Informatique Industrielle de Tours ; il s'agit d'un véhicule tout terrain, à propulsion, de motorisation thermique spécialement conçu pour le loisir. Il s'agit en fait d'un croisement entre un quad et un karting. Le Squart sur lequel nous travaillons dans le cadre de ce projet d'étude et réalisation est un Squart électrique. Sa motorisation et les modifications ont été réalisée par la société E-OXO.

Le Squart est maintenant dédié à une utilisation plus urbaine, ce qui a été rendu possible par le changement de ses roues et pneumatiques par des pièces de type scooter. De plus, la présence de commande au volant le rend accessible, par exemple, aux personnes à mobilité réduite.

Au cours des semestres précédents, Vincent Montagne, étudiant à l'IUT précédemment cité, est déjà intéressé à ce projet, mais ce dernier est resté inachevé. Ayant enfin l'occasion de mener à bien ce travail, nous avons choisi ce sujet. L'essentiel de la partie « conception » ayant déjà été effectué, la carte d'acquisition a pu être fournie dès le début de l'étude, ce qui a représenté un gain de temps considérable.

Cette carte a pour objectif d'acquérir les données provenant du Squart, telles que les tensions des batteries et les courants, la température et les tensions moteur,... Notre but est donc de programmer, au cours des différentes séances d'Etude et Réalisation, le relevé et l'affichage de ces données sur un écran LCD placé à la vue du conducteur.

L'objectif final est bien évidemment d'avoir à disposition une carte opérationnelle et un affichage clair au terme du délai qui nous est alloué, mais aussi d'apprendre à gérer un projet en autonomie, avec pour seul base un cahier des charges.

1. Cahier des charges

Un projet commençant toujours par la rédaction d'un cahier des charges clair et précis, nous avons commencé notre étude en résumant le problème qui nous était soumis, les objectifs que nous devions atteindre et les différentes contraintes inhérentes à ces objectifs.

1.1. Problématique

Le Squart est, comme il a déjà été expliqué, un véhicule à moteur destiné à être utilisé, en temps normal, hors circulation. Toutefois, le pilote qui le commande doit avoir accès à différentes informations pour se déplacer en toute sécurité. Ces informations sont, par exemple, la vitesse de déplacement, ou la température du moteur (afin de prévenir de toute avarie).

La liste (non exhaustive) des informations que nous devions rendre disponible pour le pilote nous a été fournie par Monsieur LEQUEU Thierry[1]. Nous avons ainsi pu apprendre rapidement que notre afficheur devrait rendre compte :

- des valeurs des quatre tensions des batteries ;
- de valeurs de la température du moteur (nous avions par ailleurs la possibilité de mesurer et d'afficher plusieurs températures) ;
- des valeurs des courants dans les batteries ;
- des valeurs des courants dans le moteur ;
- comme évoqué ci-dessus, de la vitesse du véhicule.

De plus, il nous a été demandé d'établir une liaison série sans fil entre le Squart et un ordinateur, ce qui rendrait possible l'affichage de toutes les informations précédemment citées sur un écran d'ordinateur éloigné. L'affichage sur l'écran d'ordinateur se fera pas l'intermédiaire d'un programme LabView d'ores et déjà réalisé par nos soins avant le début du projet.

Le schéma ci-dessous peut ainsi résumer de façon claire la problématique qui nous a été énoncée.

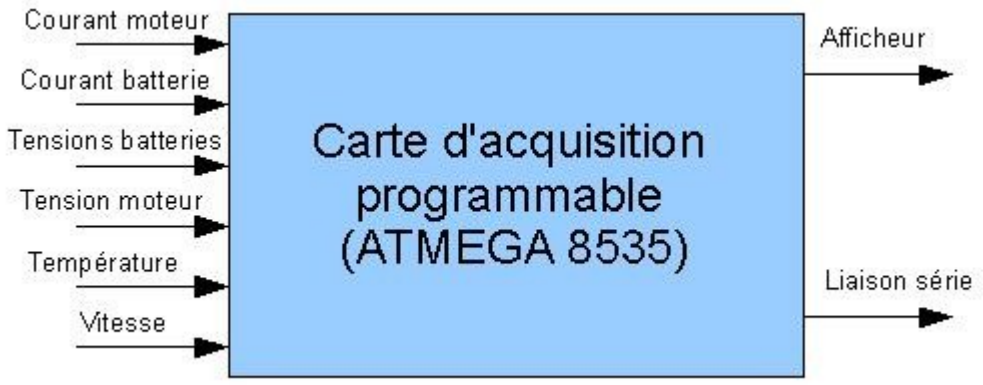


Illustration 1: Schéma fonctionnel de niveau 1

1 LEQUEU Thierry : enseignant à l'IUT Génie Electrique et Informatique Industrielle

En marge de ces objectifs, nous avons pris connaissance des différentes contraintes qui allaient nous accompagner au cours du projet.

1.2. Contraintes

Ce projet ayant déjà déjà entamé, la totalité du matériel a été définie préalablement, et nous avons dû nous adapter à ces choix. De plus, des contraintes liées à l'utilisation même de l'afficheur et de la carte d'acquisition sont apparues.

En effet, l'écran où les informations doivent être rendues disponibles doit être face au conducteur (à l'avant du véhicule), alors que la carte d'acquisition doit se trouver à l'arrière. Si cela ne semble pas poser de problème direct dans la programmation, des difficultés pourraient apparaître lorsque nous tenterons d'installer notre projet sur le Squart.

La carte d'acquisition ayant été réalisée et fournie par Monsieur LEQUEU Thierry, nous devons comprendre son fonctionnement avant de l'utiliser.

L'afficheur est également fourni, nous n'avons donc pas de marge de manœuvre à ce sujet : il s'agit d'un écran LCD à quatre lignes de seize caractères.

Ceci a généré un problème ; en effet, nous avons une grande quantité de données à afficher sur l'écran, et celui-ci n'est pas assez grand pour les contenir, ensemble, de façon claire. Il a donc été prévu d'utiliser des boutons poussoirs pour changer le mode d'affichage[2].

Pour finir, des contraintes budgétaires ont également été posées. Le coût de la réalisation doit rester inférieur à 100 euros. Nous noterons toutefois que notre projet étant essentiellement tourné vers la programmation, peu d'achats seront nécessaires à son accomplissement. Ce dernier point ne semble donc, a priori, pas poser de restriction particulière.

1.3. Présentation de la carte d'acquisition

C'est une carte d'acquisition de données programmable pour véhicule électrique. Elle nous permettra d'acquérir deux courants, ainsi que les quatre tensions des batteries 12V, 24V, 36V et 48V, et la vitesse du Squart. Elle permet également de mesurer la tension moteur et, par programmation, d'en déduire son sens de rotation.

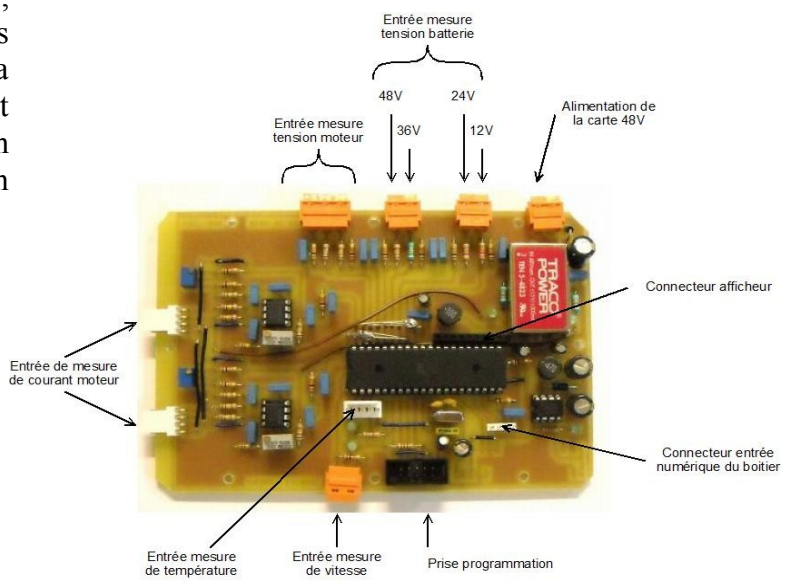


Illustration 2: Présentation des connecteurs de la carte[1]

2 Voir 3.6. Choix du menu

La carte est reliée à un afficheur 4 lignes, 16 caractères permettant un affichage pour l'utilisateur. L'élément principal est l'ATmega 8535 qui permet, par programmation, de traiter les informations. Elle est de plus alimentée en +48V, le TRACO POWER permet d'obtenir du +15V/-15V pour alimenter les AOP des entrées de mesure de courant.

A l'aide du LM2574, une tension de +5V est en outre recréée à partir du +15V. Elle est destinée à alimenter l'ATmega, les modules de communication série, l'afficheur et les capteurs de température.

Nous disposons sur l'ATmega de huit entrées analogiques sur le PORT A, qui seront toutes utilisées avec cette carte.

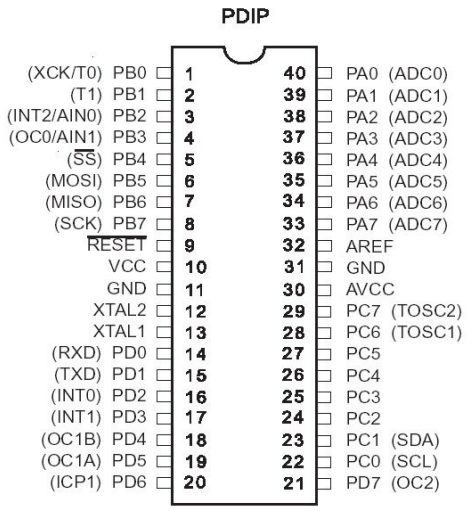


Illustration 3: Entrées / sorties de l'ATmega 8535[1]

Le PORT B est utilisé pour la programmation de l'ATmega, le PORT C est utilisé pour le branchement de l'afficheur, et le PORT D est utilisé pour la liaison série et l'entrée interruption de la vitesse.

Les tensions atteignent les points de mesure (ADC 4 à ADC 7, par exemple) par l'intermédiaire de ponts diviseurs. Ceux-ci sont calculés pour que les tensions images entrantes sur les convertisseurs Analogique-Numérique de l'ATmega ne dépassent pas +5V. La compensation pour obtenir la valeur de tension réelle sera logicielle.

Les entrées de courant, quant à elles, sont suivies d'un montage à base d'amplificateurs opérationnels, qui permet un réglage du gain et de l'offset de la tension d'entrée image du courant.

Les capteurs de température sont placée sur un bus I2C.

Enfin, l'entrée permettant la mesure de la vitesse est une entrée interruption de l'ATmega. Cela permettra, grâce à la programmation, d'obtenir la vitesse du véhicule par une méthode que nous décrirons plus avant dans ce rapport.

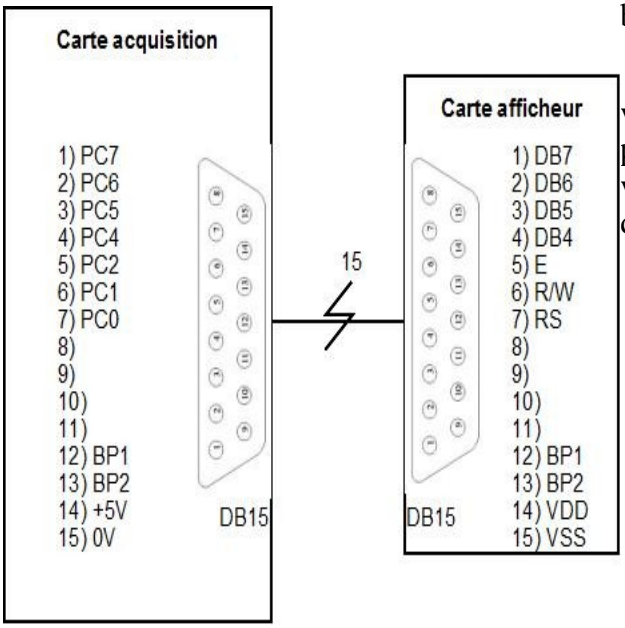


Illustration 4: Correspondance du câblage entre la prise et le boitier

Le matériel étant à présent connu, et nos objectifs clairement établis, nous pouvons créer le planning prévisionnel de notre projet.

2. Plannings prévisionnel et réel

Nous avons maintent une vision plus réelle du projet, ce qui va nous permettre d'estimer le temps que nous devrons investir sur chaque étape, dans le but d'achever la programmation de notre afficheur dans les délais accordés.

Nous pouvons donc maintenant établir un planning prévisionnel. Ainsi, nous saurons quels sont les objectifs de chaque séance, et nous pourrons nous organiser en nous servant de ces prévisions.

Le but du planning prévisionnel sera également de servir d'indicateur par rapport au temps qu'il nous reste et aux tâches que nous avons encore à accomplir.

Les plannings réel et prévisionnel sont visibles ci-après :

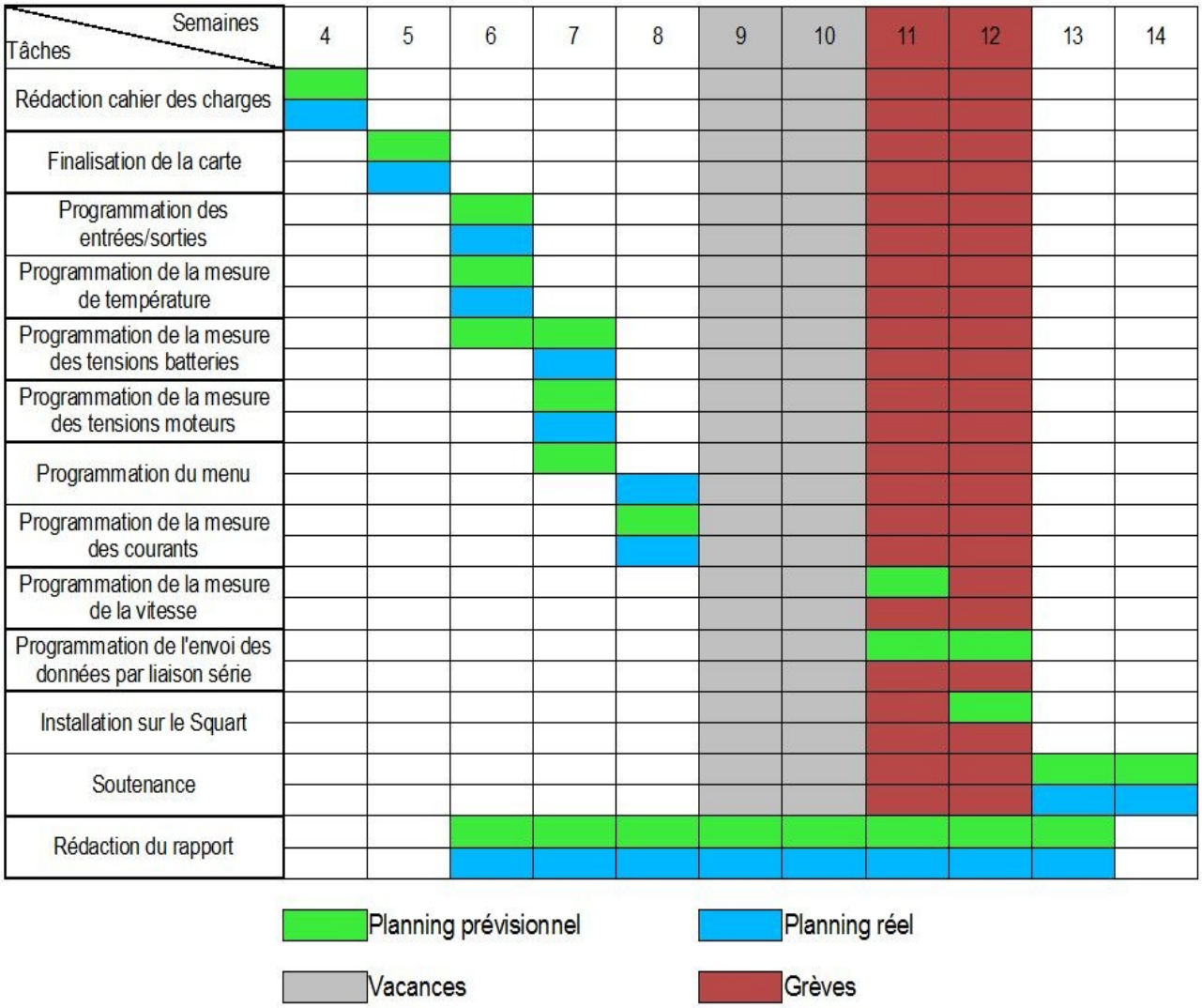


Illustration 5: Plannings prévisionnel et réel du projet

Finalement, comme l'indique le planning réel, établi au fur et à mesure de notre avancée dans le projet, nous n'avons pas pu achever la programmation de l'afficheur dans les délais impartis. En effet, on peut noter que nous avons pris du retard lors de l'étape « Programmation de la mesure des tensions moteurs ». Ce retard aurait toutefois pu être rattrapé au cours des semaines 11 et 12, mais les grèves ont fortement perturbé la tenue des délais, et nous n'avons finalement pas pu mener notre projet à son terme.

Le planning prévisionnel étant désormais établi, nous avons pu entamer le cœur du projet, à savoir la programmation en elle-même.

3. Programmation

La partie préalable à la programmation (établissement du cahier des charges, prise de connaissance du matériel,...) étant achevée, nous avons commencé à coder le programme. Le logiciel utilisé est Code Vision AVR, qui utilise le langage C comme langage de programmation.

3.1. Présentation du programme

Suite à l'élaboration de notre planning prévisionnel, nous pouvons établir l'ordinogramme complet de notre programme final. Il s'agit en fait d'une suite de parties indépendantes les unes des autres : mesure de la température, mesure des tensions des batteries, mesure des tensions moteurs,... Nous pouvons donc programmer ces différentes parties dans l'ordre qui nous convient, sans préférence : il nous suffira, au terme de notre projet, de placer toutes les parties bout à bout, et nous obtiendrons ainsi le programme final.

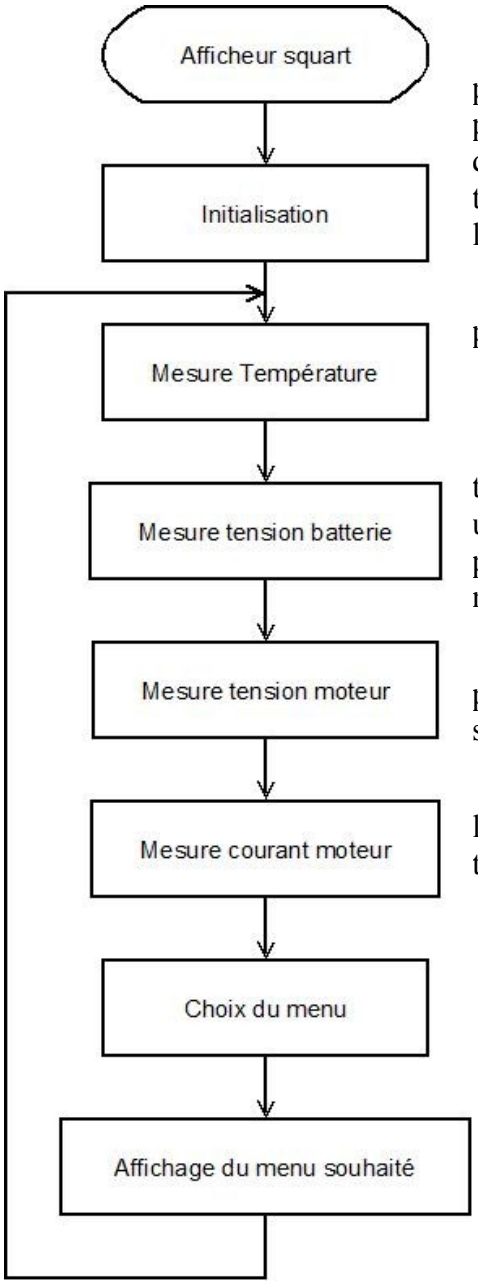


Illustration 6: Ordinogramme général du programme final

Toutefois, comme nous l'avons vu auparavant[3], notre projet n'a pas pu être achevé dans sa totalité. Les deux dernières parties du programme (« Mesure de la vitesse » et « Envoi des données par liaison série ») n'ont pas pu être codées dans les temps. Nous les avons par conséquent supprimées de l'ordinogramme initialement prévu.

On peut lire ci-contre l'ordinogramme général de notre programme final.

La partie nommée « Initialisation » n'a pas été codée en tant que telle par nos soins. En effet, le logiciel que nous utilisons a géré lui-même cette fonction, grâce aux différents paramètres que nous lui avons apporté lors de la création de notre projet[4].

Lors de cette phase d'initialisation, nous avons à paramétrer chaque bit de chaque port, soit en entrée soit en sortie suivant l'utilisation que nous faisons du bit.

De plus nous devons entrer les caractéristique de l'afficheur LCD que nous utilisons ainsi que le port ou il se trouve. Ici le LCD est branché sur le port C.

3 Voir 3. Plannings prévisionnel et réel

4 Voir Annexe 1 : Présentation de CodeWizardAVR permettant une initialisation simplifiée.

3.2. Mesure de la température

La première partie du programme que nous avons décidé de coder est la partie consistant à mesurer la température du moteur du Squart. En effet, il s'agit de la mesure la plus simple à mettre en œuvre, puisque la fonction nous a été fournie par Monsieur Thierry LEQUEU.

Nous n'avons donc eu qu'à utiliser ladite fonction (nommée « lm75_temperature_10 »). Celle-ci ne demande qu'un paramètre, qui est l'adresse où se situe le capteur. Par une brève analyse du montage, nous avons pu apprendre que notre capteur de température se situait à l'adresse 1 de l'ATMEGA.

Nous avons ainsi pu établir l'ordinogramme et le programme permettant de mesurer la température relativement rapidement.

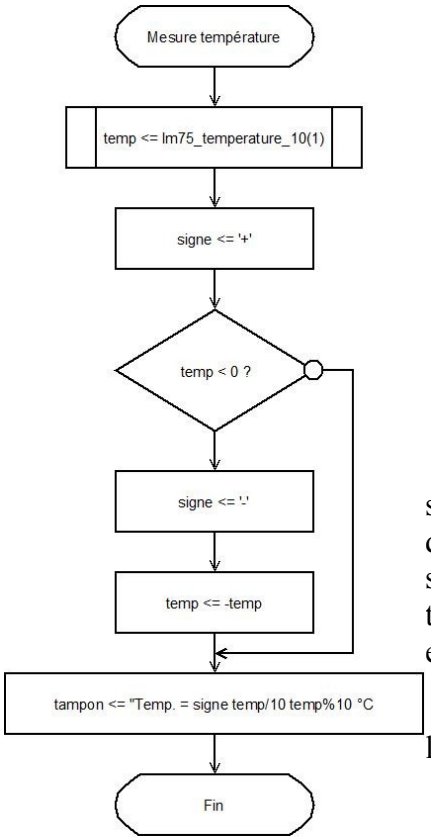


Illustration 7: Mesure de la température, ordinogramme

```
temp=lm75_temperature_10(1)
;
signe='+';
if (temp<0)
{
    signe='-';
    temp=-temp;
};
```

Nous préciserons que sur la carte d'acquisition, une LED s'allume lorsqu'un seuil de température est atteint. Nous avons défini le seuil de température « haut » comme étant à 25°C, et le seuil de température « bas » comme étant à 24°C. Ainsi, lorsque la température relevée par le capteur dépasse 25°C, la LED s'allume, et elle ne s'éteint que lorsque la température repasse sous 24°C.

Ceci a été programmé lors de l'initialisation du composant[5], à l'aide d'une seule ligne de code :

```
lm75_init(1,24,25,0);
```

Cette partie du programme a été simulée et testée non pas sur un moteur, mais à température ambiante, dans une salle de classe. Nous avons constaté que l'information affichée à partir de ces lignes de code était cohérente (la température oscillait entre 18°C et 20°C). Nous avons donc estimé que cette partie du programme était fonctionnelle.

5 Prototype de la fonction d'initialisation : lm75_init(Adresse, Seuil_min, Seuil_max ,0).

3.3. Mesure des tensions des batteries.

Après avoir effectué la partie permettant de mesurer la température du moteur, nous nous sommes employés à obtenir les valeurs des quatre batteries du Squart électrique. Nous avons établi, d'après les schémas de la carte fournie au début du projet, que nous pouvions relever ces valeurs sur les entrées ADC 4, 5, 6, 7 de notre ATMEGA (correspondant respectivement aux batteries 48V, 36V, 24V, 12V).

Ainsi, pour lire en temps réel la tension de la batterie 48 V, il nous suffit a priori de faire un appel de fonction destiné à stocker la valeur sur l'entrée 4 de l'ATMEGA, de ce type :

Variable = read_adc(4) .

Nous n'avions alors plus qu'à afficher la valeur de cette variable sur l'écran mis à notre disposition, et le pilote du Squart aurait accès à l'information. En répétant la manipulation pour chaque batterie, la fonction était finalement entièrement achevée.

Nous pouvions donc établir l'ordinogramme suivant :

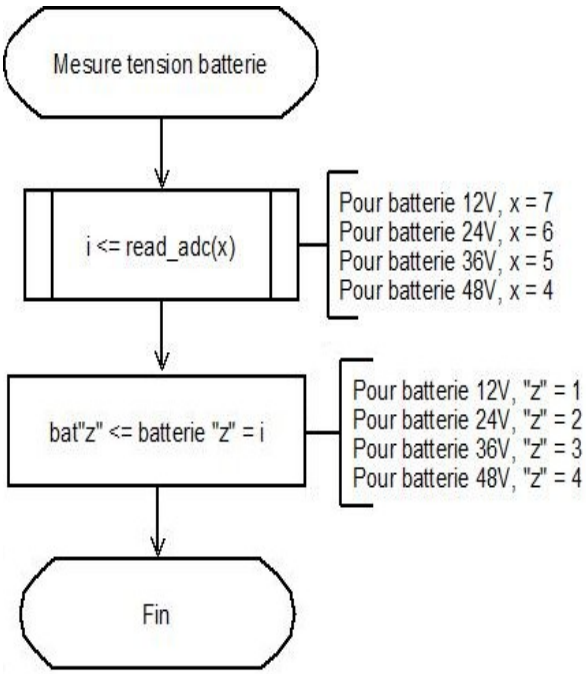


Illustration 10: Mesure des tensions des batteries, ordinogramme théorique

Ceci étant notre mode opératoire initial, nous avons pu commencer les tests.

Afin de simuler la présence des batteries, nous avons utilisé un générateur de tension continue branché sur la carte en lieu et place de la batterie 12V. Notre source de tension étant réglée pour générer du 10V=, nous attendions, sur notre afficheur, l'information suivante :

« Batterie1 = 10V ».

Après compilation, nous nous sommes aperçus que l'information affichée était erronée. En effet, le 10V n'apparaissait pas ; la valeur de tension affichée était 582 V.

Afin d'identifier notre problème, nous avons effectué divers relevés de mesures, en faisant varier la valeur de notre tension d'entrée. Nous avons pu regrouper nos relevés dans le tableau suivant :

Test entrée 12V		
Tension entrée (V)	Valeur lue avant traitement (V)	Valeur lue après traitement (V)
0	0	0
5	290	5
10	582	10
15	880	15

Après une brève analyse, nous avons pu déduire qu'un coefficient de 58,2 multipliait nos tensions d'entrée. Il semblerait que ce coefficient provienne du pont diviseur de tension, en amont de l'ATMEGA, et du Convertisseur Analogique-Numérique, qui dispose d'un gain (que nous ignorons).

Ainsi, pour retrouver la tension réelle (10 V au lieu de 582 V, 5 V au lieu de 290 V,...), nous devons diviser par 58,2 la valeur acquise à l'adresse 7 de l'ATMEGA, puis l'afficher.

Illustration 11: Mesure des tensions de la batterie 12V, courbes récapitulatives des tests

Il nous a fallu reproduire cette étude à trois reprises (afin d'établir le coefficient correspondant à chaque batterie). Nous avons alors pu établir les tableaux de mesure suivants :

- Batterie 24 V

Test entrée 24V		
Tension entrée (V)	Valeur lue avant traitement (V)	Valeur lue après traitement (V)
0	0	0
10	303	10
20	607	20
30	915	30

Nous pouvons ici observer que le coefficient est de : 303/10 = 607/20 = 915/30 = 30.

Nous devrons donc diviser par 30 la valeur acquise à l'adresse 6 de l'ATMEGA, puis l'afficher.

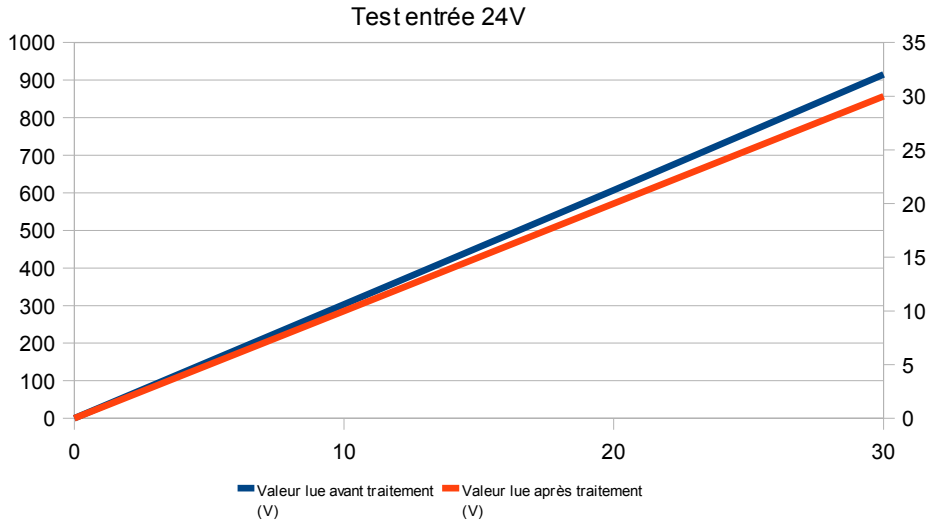


Illustration 12: Mesure des tensions de la batterie 24V, courbes récapitulatives des tests

◆ Batterie 36 V

Test entrée 36V		
Tension entrée (V)	Valeur lue avant traitement (V)	Valeur lue après traitement (V)
0	0	0
10	211	10
20	424	20
30	636	30

Nous pouvons ici observer que le coefficient est de : 211/10 = 424/20 = 636/30 = 21,2.

Nous devrons donc diviser par 21,2 la valeur acquise à l'adresse 5 de l'ATMEGA, puis l'afficher.

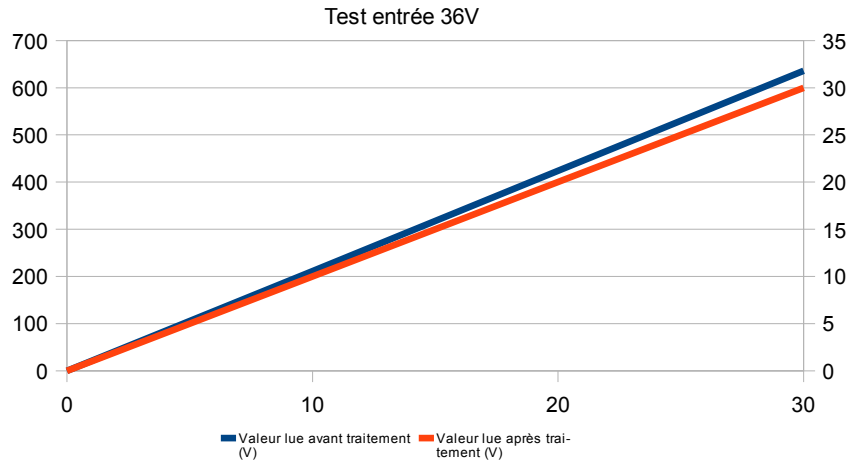


Illustration 13: Mesure des tensions de la batterie 36V, courbes récapitulatives des tests

◆ Batterie 48 V

Test entrée 48V		
Tension entrée (V)	Valeur lue avant traitement (V)	Valeur lue après traitement (V)
0	0	0
10	168	10
20	338	20
30	507	30

Nous pouvons ici observer que le coefficient est de : 168/10 = 338/20 = 507/30 = 16,8.

Nous devrons donc diviser par 16,8 la valeur acquise à l'adresse 4 de l'ATMEGA, puis l'afficher.

–

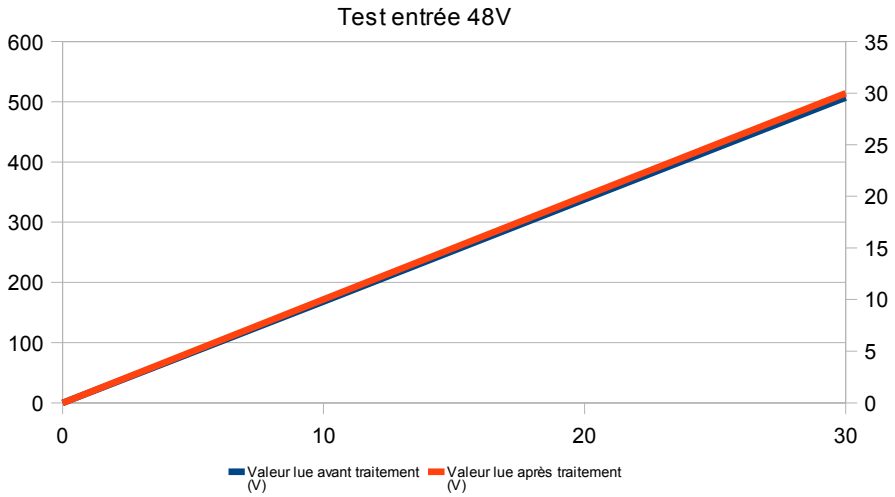


Illustration 14: Mesure des tensions de la batterie 48V, courbes récapitulatives des tests

Nous avons désormais à notre disposition les quatre coefficients qui vont nous permettre d'avoir un affichage de la tension réelle des batteries.

Ceci accompli, nous avons simplement rajouté une ligne de code destinée à diviser la valeur relevée par le coefficient déterminé précédemment. Ordinogramme et programme finaux ont donc pu être établis :

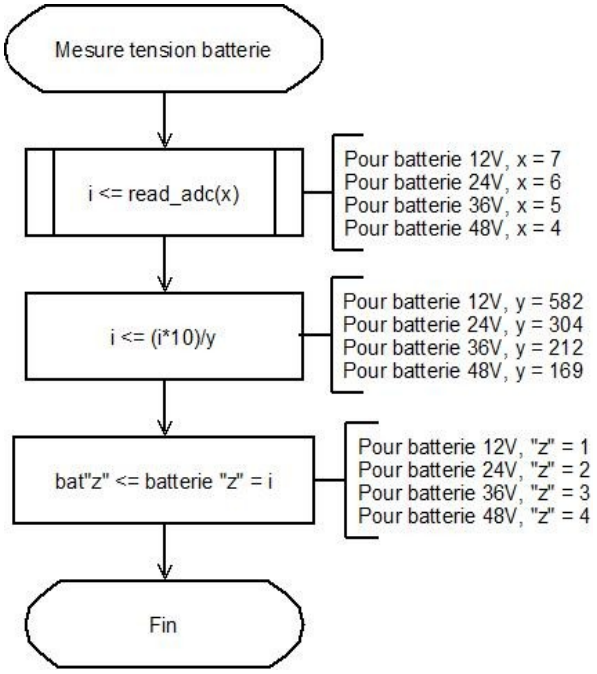


Illustration 15: Mesure des tensions du moteur, ordinogramme final

```
i=read_adc(7);      //12V
i=(i*10)/582;       //Conversion
sprintf(bat1,"Batterie 1 = %2dV",i);


i=read_adc(6);      //24V
i=(i*10)/304;       //Conversion
sprintf(bat2,"Batterie 2 = %2dV",i);


i=read_adc(5);       //36V
i=(i*10)/212;        //Conversion
sprintf(bat3,"Batterie 3 = %2dV",i);


i=read_adc(4);       //48V
i=(i*10)/169;        Conversion
sprintf(bat4,"Batterie 4 = %2dV",i);
```

Après tests, nous avons pu constater que ce programme fonctionnait selon les demandes du cahier des charges : nous pouvions bien afficher, sur quatre lignes distinctes, les quatre valeurs de tensions de batterie recherchés.

Nous devions à présent chercher à relever les valeurs de tension du moteur du Squart.

3.4. Mesure de la tension du moteur

Les tensions désirées sont, cette fois, disponibles sur les entrées 2 et 3. En effet, bien qu'il n'y ait qu'un seul moteur, nous avons à relever deux valeurs de tension. Ceci est dû au fait que suivant le sens dans lequel tourne le moteur, l'une des deux tensions sera nulle, et l'autre sera différente de 0 V. Ainsi, le hacheur 4 cadrans, en faisant changer le sens de rotation du moteur, annule l'une des deux tensions (nous ne pourrons pas avoir deux tensions différentes de 0 simultanément ; l'une des deux valeurs affichées sera toujours nulle).

Nous allons donc nous attacher à rendre disponibles les deux tensions.

Suivant le même mode opératoire que précédemment, nous avons relevé les deux valeurs et nous les avons stockées dans une variable, puis nous l'avons affichée.

En faisant, comme auparavant, varier la source de tension continue qui nous permettait de simuler la présence du moteur, nous avons pu établir le tableau suivant :

Test entrée tension moteur		
Tension entrée	Valeur lue avant traitement	Valeur attendue
0	0	0
10	168	10
20	338	20
30	507	30

Ainsi, pour les deux relevés de mesure, le coefficient était le même : nous devions diviser la valeur présentement affichée par 16,9 afin d'obtenir la véritable tension du moteur.

Nous avons alors rajouté la ligne de code attendue dans notre programme. L'ordinogramme et le programme sont donc, finalement, les suivants :

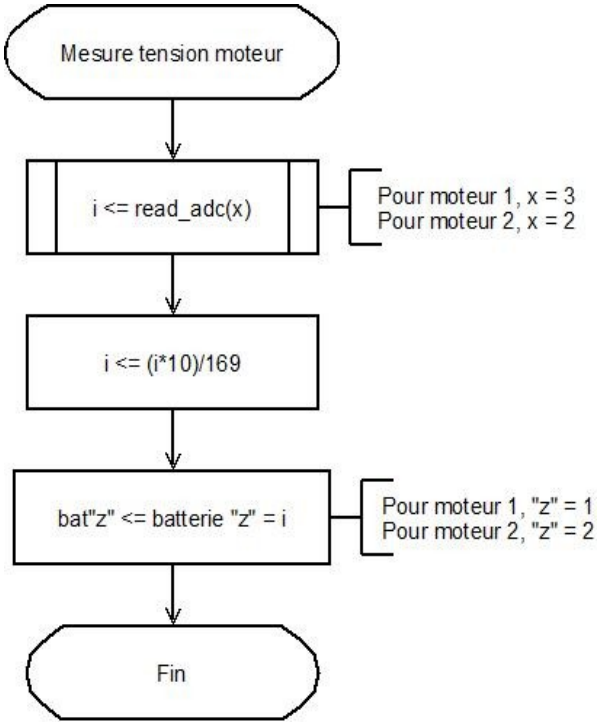


Illustration 16: Mesure des tensions du moteur, ordinogramme

```
//Moteur 1

i=read_adc(3);
i=(i*10)/169;   //Conversion
sprintf(mot1,"Moteur 1 = %2dV",i);


//Moteur2

i=read_adc(2);
i=(i*10)/169;    //Conversion
sprintf(mot2,"Moteur 2 = %2dV",i);
```

Finalement, nous avons pu constater que les valeurs attendues (variant entre 0V et 30V) étaient bien affichées sur l'écran LCD, comme nous l'attendions.

A la suite de ces lignes de code, notre afficheur rendait disponible la valeur de la température du moteur, et les tensions des batteries et du moteur du Squart.

Nous devions maintenant gérer l'affichage des courants.

3.5. Mesure des courants

Notre objectif, désormais, était de mesurer les courants.

Le capteur de courant mis à notre disposition étant composé de quarante spires, la valeur affichée sera quarante fois supérieure à celle placée en entrée.

Sur la carte d'acquisition, deux entrées (ADC(0) et ADC(1)) permettent de relever les courants.

3.5.1. L'entrée ADC(0)

La valeur du courant est modifiée en fonction des valeurs de deux potentiomètres placés sur la carte[6]. L'un de ces deux potentiomètres règle le gain (il multiplie la valeur du courant par une valeur inconnue), et l'autre agit sur l'offset (il additionne à la valeur du courant d'entrée une constante inconnue). Nous devions donc déterminer les valeurs de gain et d'offset nécessaires à l'affichage du courant placé en entrée.

Afin de simplifier nos calculs, nous devons fixer soit l'offset, soit le gain, et déterminer respectivement le gain ou l'offset. Nous avons décidé de fixer le gain au minimum (en agissant sur le potentiomètre correspondant) et de déterminer l'offset.

- Initialement, notre source de courant était débranchée. Nous devions donc avoir, théoriquement, une valeur affichée de 0 A. Or, la valeur obtenue était de 450 A. Notre offset était donc de 450 A.

- Lorsque nous avons réglé la source de courant à 5 A, la valeur affichée était de 880 A, alors que nous attendons 200 A.

Afin d'avoir en sortie une valeur de 0 A quand la tension d'entrée vaut 0, nous devons tout d'abord supprimer l'offset ; lorsque nous relèverons notre valeur de courant, nous commencerons donc par en soustraire 450 A. On aura alors, pour une valeur de 5 A en entrée, une valeur affichée de 880 – 450 = 430 A , alors que nous attendons 200 A.

Nous avons donc un coefficient de 400 / 200 = 2,15 entre l'entrée et l'affichage. Avant d'afficher, il nous faut donc diviser par 2,15 la valeur relevée.

Finalement, pour avoir 200 A affichés sur notre écran lorsqu'on a 5A en entrée, on devra relever la valeur I(mesuré) du courant et lui appliquer la formule suivante :

I(affiché) = (I(mesuré) – 450) / 2,15.

On peut résumer cette formule grâce aux courbes ci-après.

6 Voir Annexe 2 : Schéma de présentation de l'emplacement des potentiomètre de réglage des entrées de mesure des courants.

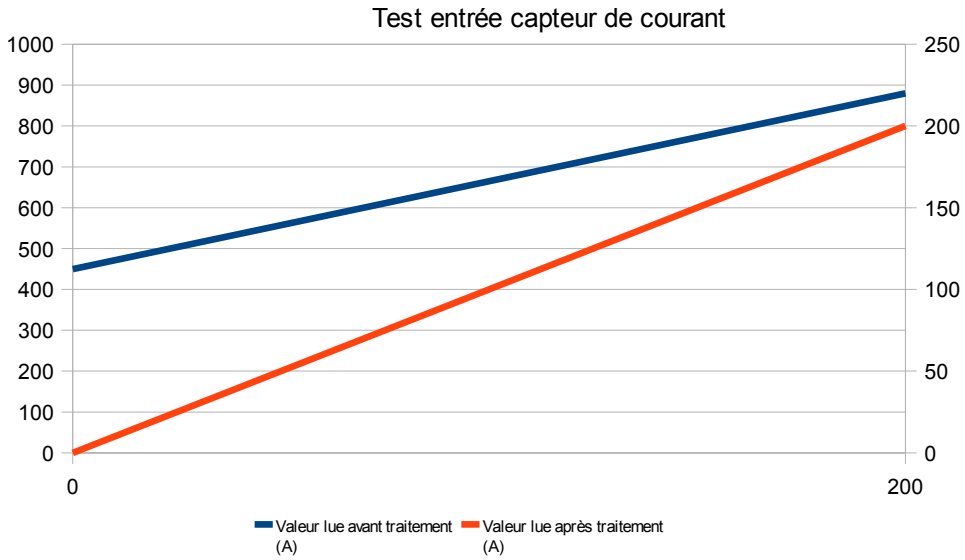


Illustration 17: Mesure des courants, Tests

On peut donc désormais construire l'ordinogramme et coder le programme qui permettra de lire le courant sur cette entrée :

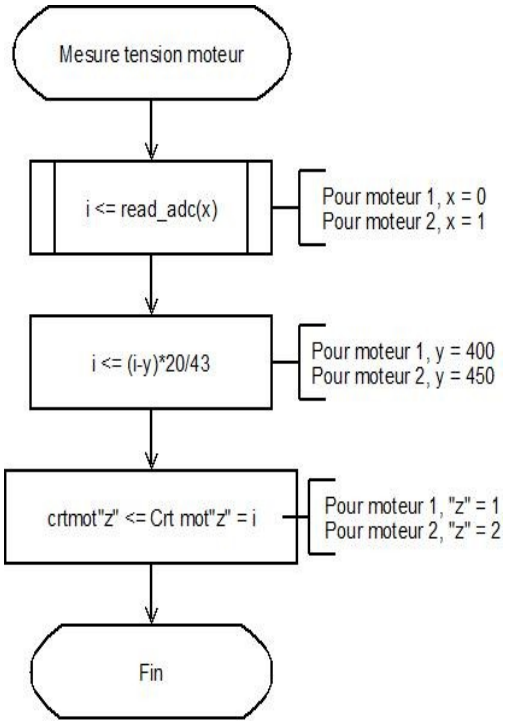


Illustration 18: Mesure des courants, ordinogramme

```
i=read_adc(0); //mesure
i=(i-400)*20/43;  //Conversion
if (i<1) i=0;
sprintf(crtmot1, "Crt Mot1 = %dA", i);
```

Après avoir testé ce morceau de programme, nous avons constaté que les valeurs affichées correspondaient bien aux valeurs attendues. Nous avons alors pu répété les mêmes tests pour l'entrée ADC (1).

3.5.2. L'entrée ADC(1)

De la même façon, on s'attache à déterminer la formule qui permettra l'affichage de 200 A lorsqu'on a 5 A en entrée. On fixe donc le gain au minimum, puis on détermine l'offset.

- Lorsque la source de courant est débranchée (on attend alors 0 A sur l'afficheur), la valeur de courant est de 400 A. Par conséquent, nous avons sur cette entrée un offset de 400 A.

- Lorsqu'on applique un courant de 5 A en entrée du montage, la valeur de courant affichée est de 830 A.

Grâce au même raisonnement que précédemment, on élimine l'offset (en soustrayant à la valeur I(mesuré) une constante de 400 A). On a alors une valeur affichée, pour 5 A en entrée, de : 830 - 400 = 430 A, alors qu'on attend 200 A.

On a donc, comme auparavant, un coefficient de 430 / 200 = 2,15 entre la valeur mesurée et la valeur attendue. On en déduit que la formule qui permettra, cette fois, d'afficher la valeur attendue, est :

I(affiché) = (I(mesuré) – 430) / 2,15.

Le code ayant la même structure que précédemment, à la formule près, il ne sera pas explicité ici.

Etant donnée le grand nombre d'informations à afficher à ce niveau, dans la programmation, nous avons choisi de nous intéresser, arrivés à cette étape du projet, à l'utilisation des boutons poussoirs pour choisir un menu et rendre clair l'affichage.

3.6. Choix du menu

La quantité de données à afficher rend impossible un affichage clair et compréhensible sur un seul et unique écran. Nous avons donc décidé de créer, en quelque sorte, un menu déroulant géré à l'aide de deux boutons poussoirs.

L'idée est de créer une variable de type entière (que nous nommerons Ecran) et d'incrémenter cette variable sur appui d'un bouton poussoir, ou de la décrémenter sur un autre appui. Ainsi, nous pouvons créer, selon le schéma ci-dessous, le déroulement que nous souhaitons.

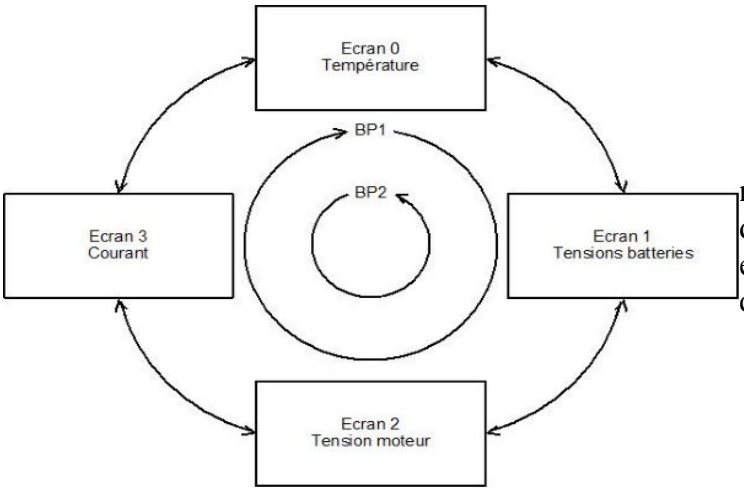
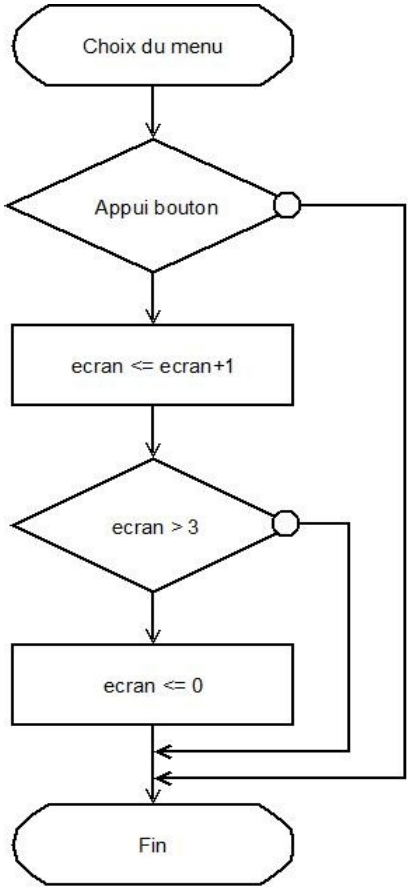


Illustration 19: Choix du menu, schéma de fonctionnement

Grâce à deux boutons poussoirs, nous pouvons bien afficher autant d'informations que nécessaires, puisqu'il est possible de faire défiler autant d'écrans que nous le voulons.

Une fois le principe établi, nous avons réfléchi au nombre d'écrans nécessaires à la bonne lisibilité du programme. Sachant que l'afficheur LCD que nous utilisons dispose de quatre lignes et que nous devions afficher, théoriquement, quatorze lignes[7], qui ont été ramenées à sept compte tenu de notre programme, nous avons estimé que nous avions besoin de trois écrans indépendants (l'ensemble des informations pouvait être affiché sur deux écrans, mais par souci de lisibilité, nous avons estimé qu'une troisième possibilité de défilement n'était pas inutile).

Nous avons donc pu coder directement la partie correspondant au « Choix du menu » ; celle-ci est visible ci-dessous. L'ordinogramme représenté ici est celui du programme gérant le bouton poussoir noté PIND.3 (bouton d'incrémentation).




```
if (PIND.3==1) //Bouton poussoir
{              //d'incrémentation
   ecran=ecran+1;
   if (ecran>3)
   {
      ecran=0;
   }
}

if (PIND.4==1) //Bouton poussoir
{              //de décrémentation
   ecran=ecran-1;
   if (ecran<0)
   {
      ecran=3;
   }
}
```

Illustration 20: Choix du menu, bouton d'incrémentation, ordinogramme.

Grâce à ce programme, le pilote peut désormais accéder, tout en pilotant, par simple appui sur l'un des deux boutons poussoirs, aux informations qu'il désire.

Il nous reste désormais à créer la partie « Affichage » sur l'écran LCD ; à chaque valeur de la variable Ecran, nous devons simplement associer un affichage particulier.

7 Quatorze lignes : huit lignes pour les courants et tensions des batteries, quatre lignes pour les courants et tensions des moteurs, une ligne pour la vitesse, une ligne pour la température.

3.7. Affichage du menu choisi

Grâce à deux boutons poussoirs, notre objectif était de choisir l'affichage désiré. Une fois le principe établi (comme expliqué au chapitre précédent), nous avons soigneusement déterminé l'affichage en fonction de la valeur de la variable Ecran. Au final, notre décision a été la suivante :

- Si la variable Ecran vaut 0, le pilote pourra lire la température du moteur (une ligne occupée) ;
- Si la variable Ecran vaut 1, le pilote pourra lire la tension des batteries (quatre lignes occupées) ;
- Si la variable Ecran vaut 2, le pilote pourra lire la tension du moteur (deux lignes occupées).
- Si la variable Ecran vaut 3, le pilote pourra lire les courants dans le moteur (deux lignes occupées).

Cette fonction a donc été relativement simple à réaliser. Nous avons eu à utiliser des structures conditionnelles, puis à placer les lignes de codes permettant d'afficher les informations sur l'écran LCD en fonction de la valeur de la variable Ecran. Entre chaque changement d'écran, toutefois, nous noterons que nous devons effacer les lignes en place avant d'écrire les nouvelles informations (à chaque appui sur un des boutons poussoirs, on efface les informations lisibles, puis on affiche les données attendues).

Nous avons donc, pour l'écran 0 (affichage de la température du moteur), par exemple, l'ordinogramme et le programme suivants :

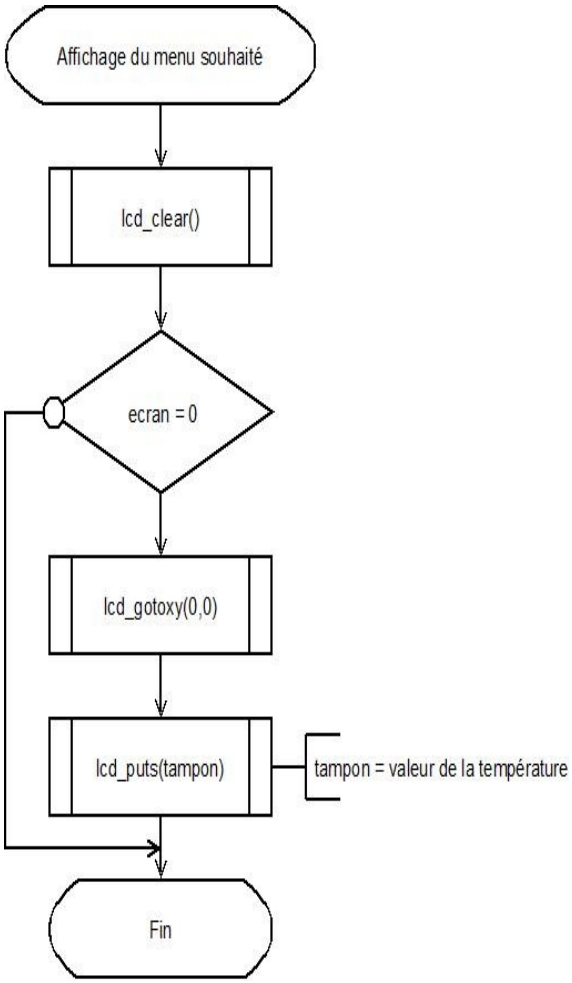


Illustration 21: Affichage du menu choisi, ordinogramme

```
lcd_clear(); //nettoyage de
l'écran

if (ecran==0) //Température
{             //du moteur
    lcd_gotoxy(0,0);
    lcd_puts(tampon);
}

if (ecran==1) //Tensions
{             //Batteries
    lcd_gotoxy(0,0);
    lcd_puts(bat1);
    lcd_gotoxy(0,1);
    lcd_puts(bat2);
    lcd_gotoxy(0,2);
    lcd_puts(bat3);
    lcd_gotoxy(0,3);
    lcd_puts(bat4);
}

if (ecran==2) //Tensions Moteur
{
    lcd_gotoxy(0,0);
    lcd_puts(mot1);
    lcd_gotoxy(0,1);
    lcd_puts(mot2);
}

if (ecran==3) //Courants Moteur
{
    lcd_gotoxy(0,0);
    lcd_puts(crtmot1);
    lcd_gotoxy(0,1);
    lcd_puts(crtmot2);
}
```

La fonction « lcd_gotoxy(x,y) » permet de nous positionner sur l'afficheur. Nous nous positionnons donc, pour l'écran 0 par exemple (voir ordinogramme), en (0,0), c'est-à-dire sur la première ligne de l'afficheur, à l'extrémité gauche. Il nous suffit ensuite d'écrire la température (variable « tampon »), et le pilote du Squart pourra la lire à tout moment.

En reproduisant ces lignes de code pour chaque donnée attendue par le cahier des charges, on a bien un déroulement des menus sur appui des boutons poussoirs, et un affichage clair et opérationnel.

3.8. Étude des parties inachevées du programme

Comme il a déjà été dit, les grèves ont perturbé l'avancement de notre projet, et nous n'avons pas pu programmer la partie « Mesure de la vitesse », ou réaliser la liaison série sans fil demandée. Néanmoins, nous avons pris le temps d'y réfléchir et, avant les grèves, de nous pencher dessus.

3.8.1. Mesure de la vitesse

La mesure de la vitesse était une des fonctions essentielles de notre afficheur. En effet, on reconnaîtra aisément qu'un compteur de vitesse fait partie des éléments indispensables d'un tableau de bord, sur quelque véhicule que ce soit. Toutefois, cette partie étant relativement complexe à traiter, nous avions envisagé de la réaliser à la fin du projet seulement. Malgré le manque de temps, nous avons réfléchi à la meilleure façon de programmer notre compteur de vitesse.

Finalement, nous avons décidé d'utiliser un aimant, fixé sur l'arbre moteur, et un interrupteur électromagnétique juste au-dessus. Ainsi, à chaque tour de roue, l'aimant aurait enclenché cet interrupteur en passant devant, et une impulsion aurait pu être détectée par le programme. Cette impulsion aurait alors pu lancer une interruption système, et en comptant le temps entre chaque impulsion, nous aurions pu déterminer la vitesse du Squart.

En effet, connaissant le rayon de la roue (après mesures, nous estimerons que ce rayon vaut R = 25,5 centimètres, soit 0,255 mètre), nous pouvons déterminer le périmètre P de la roue.

$$P = 2 * \Pi * R = 1{,}57 \text{ mètre.}$$

De plus, si on mesure le temps entre deux impulsions lancées par l'interrupteur, on trouve le temps T nécessaire au véhicule pour effectuer un tour de roue, soit 1,57 mètre.

On arrive donc à calculer la vitesse V du véhicule par la formule :

$$V = D / T = 1{,}57 / T$$

Si, par exemple, le véhicule a besoin de 0,5 seconde pour effectuer un tour de roue, sa vitesse sera de 1,57 / 0,5 = 3,14 mètre par seconde, soit environ 11 km/h.

Le seul réel problème lié à cette méthode est dû au fait que le rayon de la roue varie en fonction de la pression des pneus. Or, si le rayon de la roue varie, son périmètre est également modifié, et cela ne sera pas pris en compte par le programme. Pour citer un exemple, si la pression d'un pneu est inférieure à la normale, le rayon de la roue sera alors inférieur à 25,5 cm. Le périmètre sera donc inférieur, et entre deux impulsions, le temps sera plus court. Or, dans notre programme, D= 1,57 m.

On a donc : (D reste constant , T diminue) → la vitesse affichée augmente.

De même, si la pression est supérieure à la normale, on aura une vitesse affichée plus faible que celle réelle. La pression des pneus est donc un facteur de précision important. La vitesse pourra être différente selon la pression des pneus, ou même en fonction de la roue où le capteur sera installé (en effet, la pression variant entre chaque pneu, le rayon de chaque roue est différent, donc la vitesse de chaque roue est différente).

Malgré cela, cette méthode reste à nos yeux la plus efficace, et il s'agit de celle que nous avions choisi pour mesurer la vitesse du Squart.

3.8.2. Liaison série sans fil

Lors du lancement de ce projet, nous souhaitions que les données acquises et traitées par la carte puissent être transmises sur un ordinateur. Pour cela, il a été implanté sur la carte un module de transmission série sans fil à 433MHz.

Un programme sous Labview a été développé, permettant l'affichage de toutes les données en temps réel, avec possibilité d'enregistrement de celles-ci pour une étude ultérieure.

La problématique était donc de transmettre via la liaison série sans fil les informations, et de les recevoir sur l'ordinateur.

Pour cela, nous avions à notre disposition une carte développée par M. LEQUEU. Cette carte était munie d'un récepteur de liaison série sans fil, et elle pouvait traiter le signal pour le rendre compatible avec une connexion série d'un ordinateur. Nous avons donc effectué des tests en envoyant des données depuis notre carte, mais aucune information n'était reçue par l'ordinateur.

Nous avons donc testé les modules de liaison série sans fil à tour de rôle afin de vérifier leur bon état de fonctionnement. Finalement, nous n'avons pu que constater que ceux-ci n'étaient pas en cause.

La réception sur l'ordinateur pouvait être la cause de nos problèmes ; nous avons donc tenté de faire dialoguer deux cartes munies d'ATmega entre elles, mais nous n'avons eu cette fois encore aucun résultat concluant.

Nous avons donc supposé que le problème provenait de la liaison sans fil. Nous avons alors tenté de relier les entrées TX à RX du même ATmega. En supprimant la liaison série sans fil, nous nous attendions à ce que nous recevions les caractères émis. Cependant, nous n'avons toujours pas réussi a obtenir ce que nous transmettions.

Malgré l'aide de M. LEQUEU, qui avait au préalable testé ces modules en faisant dialoguer deux cartes avec ATmega, nous ne sommes parvenus à l'heure actuelle à aucun résultat nous permettant d'utiliser la liaison série dans notre projet.

3.9. Programme final

Notre programme final est donc, comme il a déjà été dit, la simple mise en cascade des différentes parties étudiées tout au long du projet. Il prend donc en compte la mesure de la température du moteur, la mesure des tensions des batteries et du moteur, et la mesure des courants. Il existe quatre écrans indépendants, que le pilote peut sélectionner en appuyant sur l'un des deux boutons poussoirs.

Le programme final est visible ci-après.

```
#include <mega8535.h>
// I2C Bus functions
#asm
   .equ __i2c_port=0x18 ;PORTB
   .equ __sda_bit=0
   .equ __scl_bit=1
#endasm
#include <i2c.h>
// LM75 Temperature Sensor functions
#include <lm75.h>
// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
// Standard Input/Output functions
#include <stdio.h>
#include <delay.h>
#define ADC_VREF_TYPE 0x00
// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
```

```
void main(void)
{
// Declare your local variables here
int temp,ecran;
int i;
unsigned char tampon[20];
unsigned char bat1[20];
unsigned char bat2[20];
unsigned char bat3[20];
unsigned char bat4[20];
unsigned char mot1[20];
unsigned char mot2[20];
char signe;
unsigned char crtmot1[20];
unsigned char crtmot2[20];
// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;
// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;
```

```
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
```

```
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;
// ADC initialization
// ADC Clock frequency: 125,000 kHz
// ADC Voltage Reference: AREF pin
// ADC High Speed Mode: On
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x87;
SFIOR&=0xEF;
```

```
SFIOR|=0x10;
// I2C Bus initialization
i2c_init();
// LM75 Temperature Sensor initialization
// thyst: 35°C
// tos: 40°C
// O.S. polarity: 0
lm75_init(1,24,25,0);                    //initialisation du LM75
// LCD module initialization
lcd_init(16);
ecran=0;
while (1)
     {
       temp=lm75_temperature_10(1);          //Relevé de la température du moteur.
       signe='+';
       if (temp<0)
       {
          signe='-';
          temp=-temp;
       };
       sprintf(tampon,"Temp. = %c%i.%u\xdfC",signe,temp/10,temp%10);

       i=read_adc(7);    //12V                          //conversion tension batteries
       i=(i*10)/582;
       sprintf(bat1,"Batterie 1 = %2dV",i);
       i=read_adc(6);    //24V
       i=(i*10)/304;
       sprintf(bat2,"Batterie 2 = %2dV",i);
       i=read_adc(5);     //36V
       i=(i*10)/212;
       sprintf(bat3,"Batterie 3 = %2dV",i);
```

```
i=read_adc(4);      //48V
i=(i*10)/169;
sprintf(bat4,"Batterie 4 = %2dV",i);

i=read_adc(3);                                  //conversion tension moteurs
i=(i*10)/169;
sprintf(mot1,"Moteur 1 = %2dV",i);
i=read_adc(2);
i=(i*10)/169;
sprintf(mot2,"Moteur 2 = %2dV",i);
i=read_adc(0);                                   //Conversion courant 0
i=(i-400)*20/43;
if (i<1) i=0;
sprintf(crtmot1, "Crt Mot1 = %dA", i);
i=read_adc(1);                                   //Conversion courant 1
i=(i-450)*20/43;
if (i<1) i=0;
sprintf(crtmot2, "Crt Mot2 = %dA", i);

if (PIND.3==1)                                    //deroulement du menu par boutons
{
   ecran=ecran+1;
   if (ecran>3)
  {
     ecran=0;
  }
}
if (PIND.4==1)
{
   ecran=ecran-1;
   if (ecran<0)
```

```
   {
      ecran=3;
    }
 }

lcd_clear();                              //création menu
if (ecran==0)
{
   lcd_gotoxy(0,0);
   lcd_puts(tampon);
}
if (ecran==1)
{
   lcd_gotoxy(0,0);
   lcd_puts(bat1);
   lcd_gotoxy(0,1);
   lcd_puts(bat2);
   lcd_gotoxy(0,2);
   lcd_puts(bat3);
   lcd_gotoxy(0,3);
   lcd_puts(bat4);
}
if (ecran==2)
{
   lcd_gotoxy(0,0);
   lcd_puts(mot1);
   lcd_gotoxy(0,1);
   lcd_puts(mot2);
}
if (ecran==3)
{
```

```
            lcd_gotoxy(0,0);
            lcd_puts(crtmot1);
            lcd_gotoxy(0,1);
            lcd_puts(crtmot2);
        }
        delay_ms(50);
       };
}
```

Conclusion

Dans le cadre de notre projet d'Etude et Réalisation, nous avons choisi de réaliser l'affichage d'informations sur un écran LCD, qui devait par la suite être intégré à un Squart électrique. Ce projet avait déjà été entamé, et nous avons donc pu accorder davantage de temps à la programmation effective du programme, (la partie électronique a pu être largement réduite).

Toutefois, après six séances d'Etude et Réalisation, plusieurs données manquent encore à l'affichage. Ceci s'explique notamment par les perturbations liées aux grèves, mais aussi par le retard accumulé lors de nos tests. En effet, comme nous l'avons vu, la réalisation des fonctions destinées à mesurer les tensions et courants ont nécessité la réalisation de relevés de mesures, qui ont parfois été plus longs que nous ne l'avions prévu. Ainsi, même si le mode opératoire était relativement répétitif, beaucoup de temps a dû être consacré à la programmation de ces fonctions.

De plus, la partie concernant les mesures de courant a posé un réel problème de réflexion, et nous avons été contraint de l'étudier durant davantage de temps que nous le pensions.

Le logiciel Code Vision AVR, que nous ne connaissions pas, s'est finalement révélé simple d'utilisation ; la prise en main a pu être effectuée rapidement. Malgré quelques messages encore inexpliqués, il s'est montré très efficace et nous a permis de gagner du temps, notamment (comme nous l'avons vu) lors de l'écriture de la fonction « Initialisation » de notre programme.

Si les perturbations liées aux grèves nous ont empêchés de mener à bien notre projet, nous avons pu travailler en autonomie, sur un projet à long terme, et cela nous a été profitable dans le cadre de la préparation au stage en entreprise.

Ce projet reste donc, finalement, inachevé, mais n'en demeure pas moins instructif. Nous nous attacherons par ailleurs à le mener à son terme avant le départ en stage. La majorité du travail est effectuée, la mesure de la vitesse devrait être effectuée rapidement, et les problèmes liés à la réalisation de la liaison série devraient trouver une solution dans les jours à venir.

Résumé

264 mots

Si la programmation de l'affichage n'a pas pu être entièrement achevée, nous avons pu faire progresser le projet. Bien que l'objectif principal (obtenir un affichage complet et lisible dans les délais impartis) ne soit pas atteint, nous avons travaillé en autonomie et gagné en expérience et en assurance.

Nos compétences techniques n'ont pas fondamentalement évoluées, puisque la programmation en elle-même ne faisait pas appel à des notions particulièrement complexes. La difficulté du projet résidait dans la quantité de tests à réaliser, et à l'application d'une logique rigoureuse (notamment en ce qui concerne la détermination des courants). De fait, les problèmes ont, dans la majorité des cas, pu être résolus. De plus, ce projet nous a permis de découvrir Code Vision AVR, qui a facilité le lancement de notre projet (par le codage « automatique » de la fonction « Initialisation » du programme) et, plus important encore, d'utiliser un composant programmable en langage C (chose nouvelle pour nous depuis le premier semestre, où le langage de programmation utilisé était le Verilog).

Nous avons néanmoins rencontré des obstacles que nous n'avons pas encore pu franchir. La mesure de la vitesse, fonction essentielle du programme, ne peut être effectuée pour le moment, et la liaison série sans fil reste défectueuse pour une raison inconnue.

Au-delà des difficultés rencontrées, le projet nous a permis de revoir des notions d'électronique (comme les Convertisseurs Analogique-Numérique) tout en programmant ; bien que la partie consacrée à l'informatique soit plus importante que celle réservée à l'électronique, nous avons pu réviser certaines bases, ce qui pourra nous être utile pour le stage.

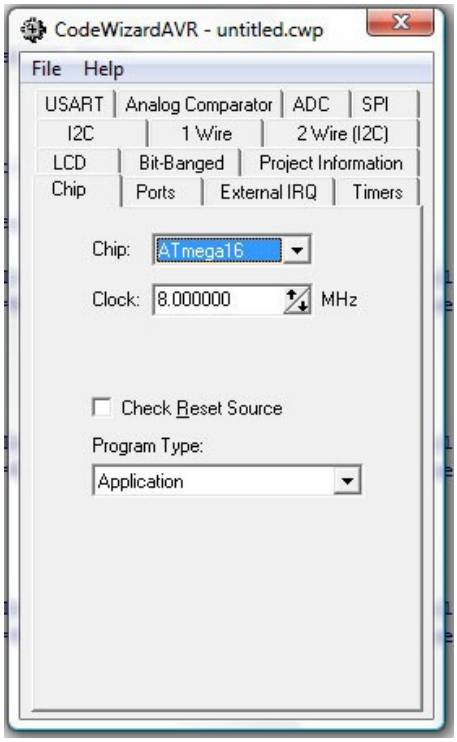
Index des illustrations

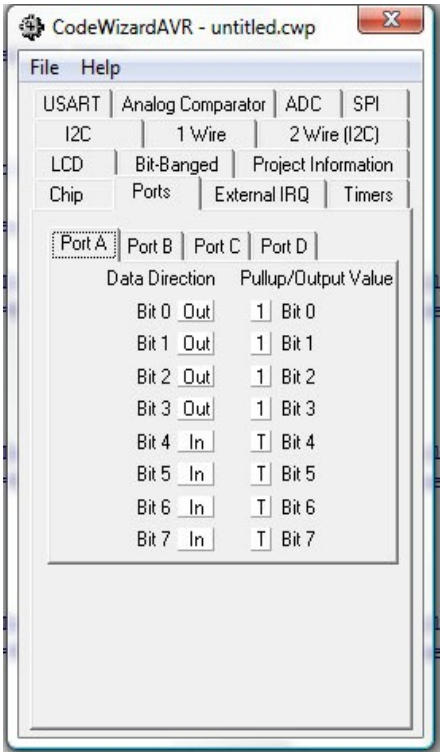
Bibliographie

[0] **LEQUEU THIERRY**, "*La documentation de Thierry sur le net*", [En ligne]. <http://www.thierry-lequeu.fr/> (Page consultée le semestre 4).
[1] **Saurabh Sankule**, "*Atmel Microcontrollers (AVR) - Compiler/IDE*", [En ligne]. <http://home.iitk.ac.in/~sankule/robotics/avr/compiler.html> (Page consultée le semestre 4).

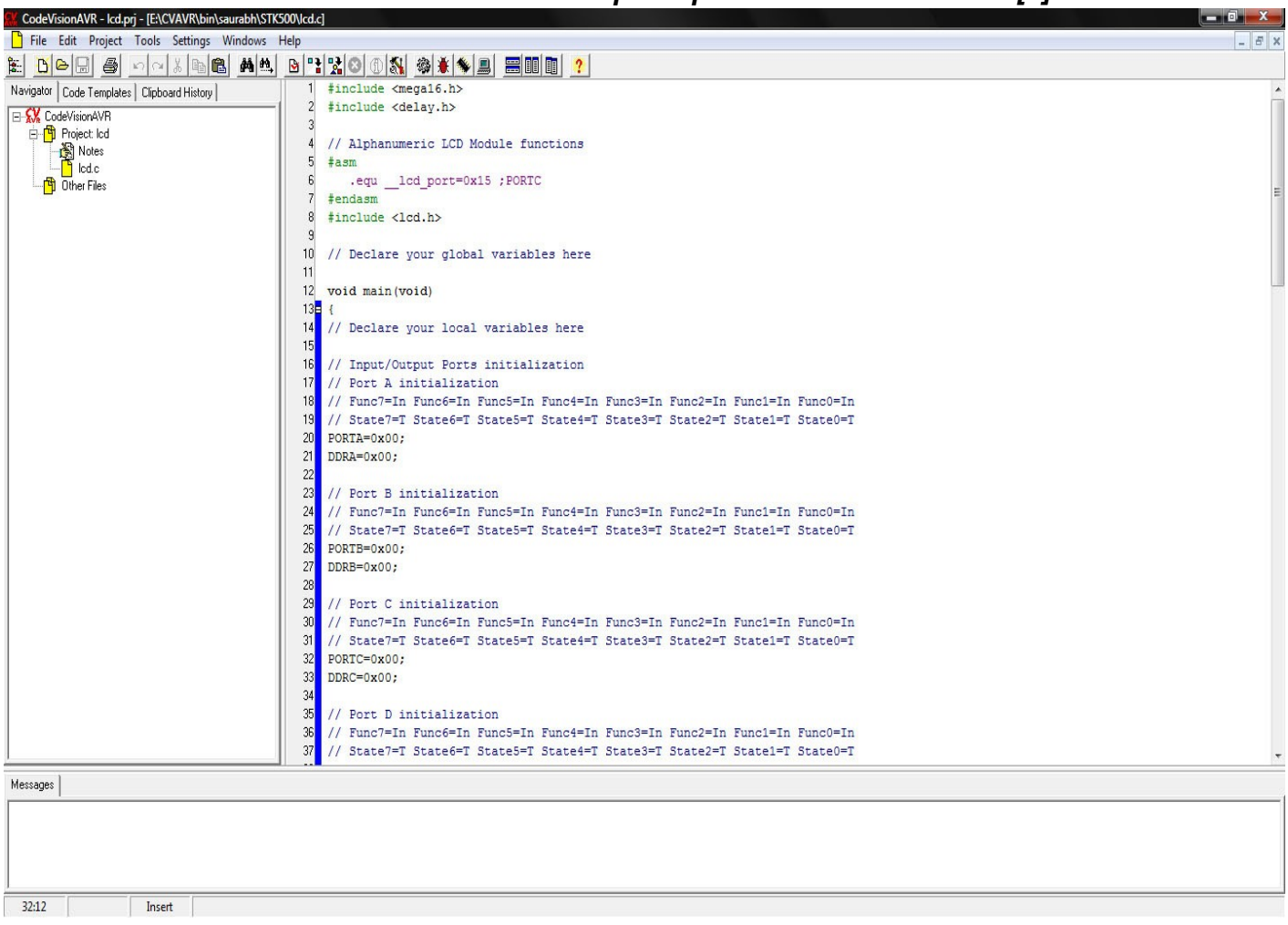
Annexes

Annexe 1 : Présentation de CodeWizardAVR, permettant une initialisation simplifiée.[2]

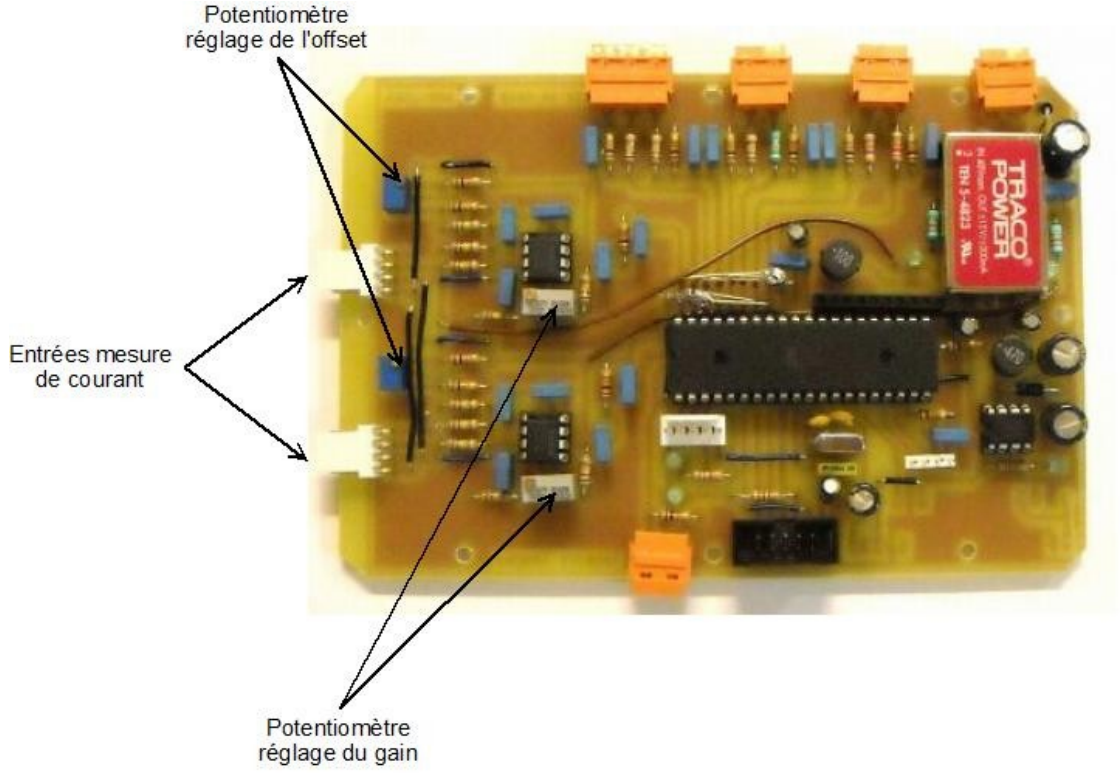




Présentation de la fenêtre principale de CodeVisionAVR[2]



Annexe 2 : Schéma de présentation de l'emplacement des potentiomètres de réglage des entrées de mesure des courants.[1]



Annexe 3 : Présentation du programme labview préalablement développé.