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

Vu-mètre pour batterie

Projet ERGE / ERISI

Cédric BAUD
Guillaume BERNADAC
Année 2014-2015
Groupe K4B

Philippe AUGER
Thierry LEQUEU

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

Illustration 1: Kart biplaces ERDF

Vu-mètre pour batterie

Projet ERGE / ERISI

Cédric BAUD
Guillaume BERNADAC
Année 2014-2015
Groupe K4B

Philippe AUGER
Thierry LEQUEU

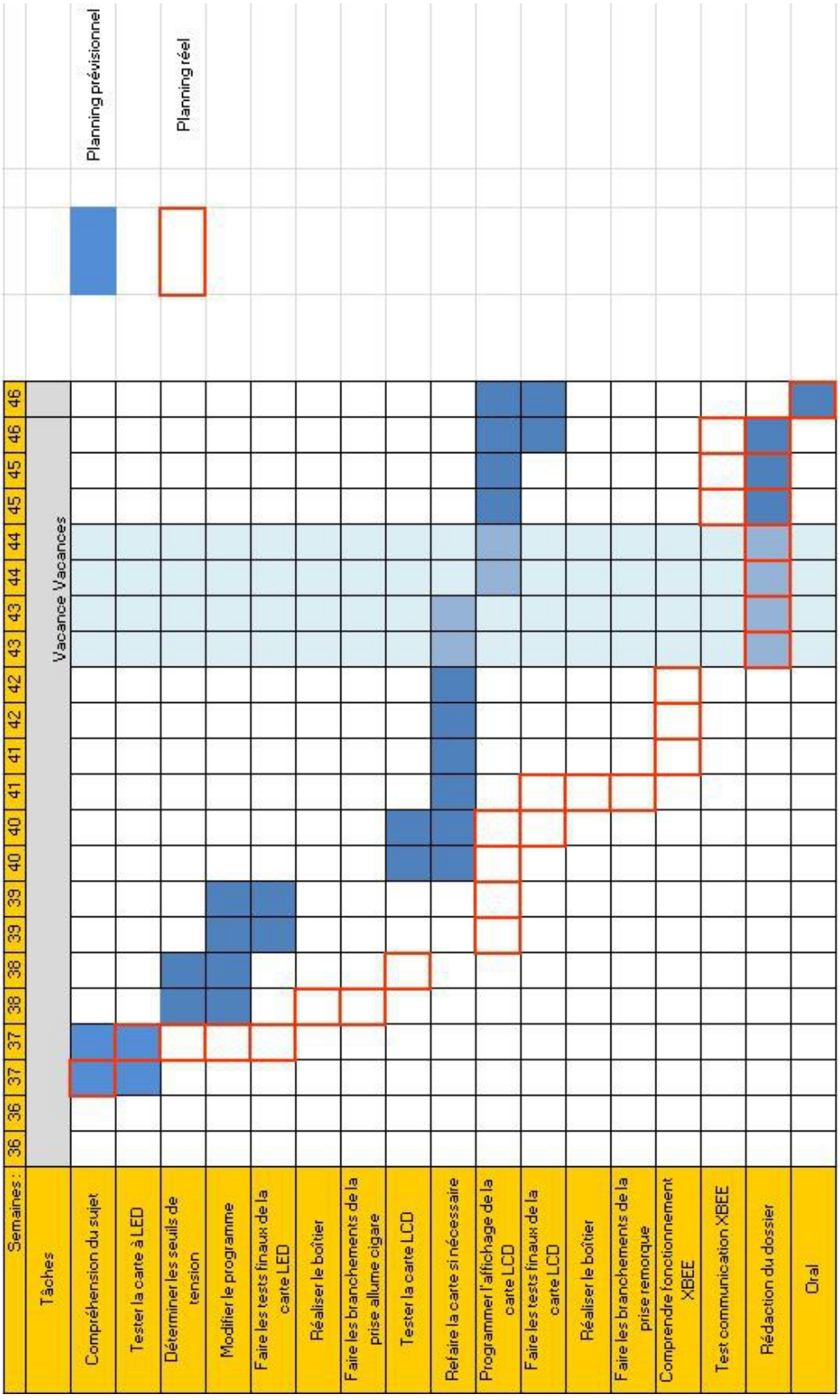
Sommaire

Introduction

Au cours du quatrième semestre de notre formation en DUT Génie Électrique et Informatique Industrielle, nous avons été amenés, dans le cadre des cours d'étude et réalisation (E&R GE et E&R ISI) à mener à bien un projet. Différents thèmes nous ont alors été proposés lors de la première séance. En accord avec les autres étudiants du groupe nous avons décidé de nous lancer dans la réalisation d'un vu-mètre à LED pour batterie 12V et d'un vu-mètre à écran LCD affichant la tension aux bornes de 4 batteries 12V pour kart.

Au travers de ce dossier nous allons expliquer comment, depuis le départ, nous avons réalisé ce projet. Pour ce faire, nous aborderons dans une première partie le vu-mètre à LED, puis nous nous intéresseront au vu-mètre LCD, enfin nous traiterons de la communication du vu-mètre LCD avec un ordinateur via des modules XBEE.

Plannings prévisionnel et réel



1. Vu-mètre à LED

La première partie de notre projet consiste à réaliser un Vu-mètre à LED pour une batterie de 12V. Un vu-mètre est un appareil qui est utilisé à la base pour mesurer et afficher le niveau d'un signal audio. Ici c'est le même principe appliqué à un niveau de tension.

1.1. Cahier des charges

Réaliser un Vu-mètre à LED pour batterie 12V ayant les caractéristiques suivantes :

- Comporter 11 LED dont X rouges, X oranges et X vertes
- Allumer ces LED de façon proportionnelle à la tension de la batterie depuis la gauche pour les basses tensions voisines de 10V vers la droite pour les hautes tensions voisines de 15V.
- Rentrer dans un boîtier étanche d'une taille adaptée
- Être muni d'une prise « allume cigare » permettant de mesurer le niveau de batterie d'une voiture

1.2. Point de départ

Ce projet avait déjà été commencé par des étudiants qui avaient réalisé la carte électronique permettant d'allumer des LED en fonction d'une tension appliquée. Ceux-ci avaient également écrit un programme qui commandait les LED en fonction de la tension. La carte était tout à fait fonctionnelle mais le programme n'allumait pas les LED correctement. En effet, elles s'allumaient depuis la droite pour les basses tensions voisines de 10V vers la gauche pour les hautes tensions voisines de 15V. De plus, les tensions dites « de seuil » à partir desquelles la LED suivante devait s'allumer n'étaient pas correctement réglées.

1.3. Travail à réaliser

Notre travail a donc été dans un premier temps de comprendre le programme déjà écrit, puis nous avons du modifier celui-ci afin de le rendre conforme au cahier des charges, enfin nous avons dû réaliser le boîtier en prenant soin de percer celui-ci afin de pouvoir branche une prise allume cigare.

1.4. Compréhension du programme :

Le programme qui allume les LED en fonction de la tension de la batterie est relativement simple.

Il est composé de différents tests « if » les uns à la suite des autres.

Un test if est une fonction qui teste le paramètre que l'utilisateur lui donne : s'il est vrai alors on réalise une certaine action, s'il est faux on en réalise une autre.

Ici on teste la tension mesurée : si celle-ci est supérieure à une certaine valeur on allume une LED, sinon on l'éteint. Cette opération est répétée pour chacune des LED qui composent notre vu-mètre.

```
if(TensionBat>=945)
{
PORTD.5=1;
}
else
{
PORTD.5=0;
}
```

Illustration 2: code correspondant au if

La tension mesurée par la carte « passe » dans un convertisseur analogique numérique (CAN) qui à partir de la tension en Volt mesurée renvoie une valeur comprise entre 0 et 1024.

Théoriquement, dans notre cas, la carte pourra mesurer une tension entre 0 et 16V, 0V correspondant au 0 numérique et 16V au 1024 numérique.

Cette valeur renvoyée par le convertisseur sera récupérée dans notre programme par le biais de la fonction read-adc(0) (0 correspondant au convertisseur utilisé de l'ATmega).

Les 11 LED sont commandées par les bits 1 à 8 du registre PORTC ainsi que par les bits 5 à 7 du registre PORTD, les LED étant reliées aux pattes des ports C et D du microcontrôleur (voir annexe 2)

Détermination des seuils d'allumage des LED :

On dispose de 11 LED pour afficher une tension comprise entre 10 et 15V. On choisi donc les seuils théoriques suivants :

LED 1 → 10V

LED 2 → 10,5V

LED 3 → 11V

LED 4 → 11,5V

LED 5 → 12V

LED 6 → 12,5V

LED 7 → 13V

LED 8 → 13,5V

LED 9 → 14V

LED 10→ 14,5V

LED 11→ 15V

On détermine les seuils numériques par un produit en croix à partir de l'égalité suivante :

1024 → 16V

On en déduit donc les seuils suivants :

LED 1 → 10V → 640

LED 2 → 10,5V → 672

LED 3 → 11V → 704

LED 4 → 11,5V → 736

LED 5 → 12V → 768

LED 6 → 12,5V → 800

LED 7 → 13V → 832

LED 8 → 13,5V → 864

LED 9 → 14V → 896

LED 10→ 14,5V → 928

LED 11→ 15V → 960

Avec ces différentes valeurs de seuils nous avons testé le fonctionnement de notre programme en lançant celui-ci dans l'ATmega8535 et nous avons réalisé que les LED s'allumaient aux alentours de la tension que nous souhaitions, nous avons donc corrigé les erreurs dues à la précision des composants électroniques en modifiant légèrement ces valeurs théoriques.

Nous avions à présent des seuils d'allumage des LED correctes au niveau de leur valeur mais ces derniers n'étaient pas très francs : quand la tension mesurée devenait proche du seuil la diode clignotait.

Pour remédier à ce problème, la solution a été de faire une moyenne des mesures : de mesurer la tension un certain nombre de fois, d'additionner chaque résultat et de diviser le résultat final par le nombre de mesure.

Nous avons alors décidé de faire les moyennes sur 100 mesures. Cependant le programme ne fonctionnait plus avec ce moyennage.

Après avoir réfléchi aux causes de ce problème, nous nous sommes rendu compte que la variable que nous utilisions pour faire les mesures était déclarée en int (entier codé sur 16 bits) ne pouvait dépasser 32 768 par valeurs positives. En effet 16 bits permettent d'écrire 2^16 nombres soit 65 536, et comme les variables int peuvent également être négatives, elles ne peuvent dépasser 65 536/2=32 768, or pour 16V (tension maximum d'une batterie), on a 1024*100 = 102 400 ce qui dépasse les 32 768 et explique donc pourquoi notre programme ne fonctionnait pas.

On a donc décider de déclarer notre variable en « unsigned int » ce qui nous permettait d'obtenir 65 536 nombres, et en réduisant notre nombre de mesure à 60 nous restions en dessous de ce seuil même pour la valeur maximale théorique de 1 024 (1 024*60= 61 440).

Finalement avec ce moyennage sur 60 valeurs nous avions des LED dont les seuils d'allumage était bon et franc.

```
TensionBat=0;

for(i=0;i<=60;i++)
{
  TensionBat+=read_adc(0);

}
TensionBat=TensionBat/60;
```

Illustration 3: code correspondant au moyennage

2. Le vu-mètre à écran LCD

La seconde partie de notre projet a été la réalisation d'un vu-mètre à écran LCD affichant le niveau des quatre batteries 12V d'un kart électrique. Dans cette partie, nous serons amenés à utiliser quatre batteries de 12 V montées en série que l'on appellera batterie 12V, batterie 24V, batterie 36V et batterie 48V.

2.1. Cahier des charges

Réaliser un Vu-mètre à LED pour batterie 12V ayant les caractéristiques suivantes :

- Afficher le niveau des 4 batteries, indépendamment les unes des autres
- Être alimenté par la première batterie 12V
- Rentrer dans un boîtier étanche d'une taille adaptée
- Être muni d'une prise permettant le branchement du vu-mètre sur le kart

2.2. Point de départ

Ici encore, le projet avait été commencé par des étudiants qui avaient réalisé la carte électronique comportant l'écran LCD. Nous disposions également du programme écrit par ces mêmes étudiants. Ce programme comportait des erreurs liées à la version du logiciel CodeVision AVR ce qui nous empêchait de tester ce dernier.

2.3. Travail à réaliser

Nous avons donc dû trouver la bonne version du logiciel, puis essayer de comprendre les différentes erreurs afin de les corriger. Il a ensuite fallu modifier le programme afin que les tensions s'affichent convenablement et que notre produit soit conforme au cahier des charges. Enfin il a fallu réaliser le boîtier et le branchement de la prise permettant d'adapter le vu-mètre au kart.

2.4. Récupération du programme

Dans un premier temps, il nous a fallu récupérer le programme précédent, écrit par les étudiants ayant travaillé sur le projet afin de mesurer l'avancée de ces derniers et de pouvoir commencer sur une base de travail. N'étant pas stocké sur le site de M. LEQUEU, nous avons numérisé leur programme se trouvant en annexe de leur dossier afin de le copier dans un nouveau projet de CodeVision. Pour créer ce nouveau projet, nous avons utilisé CodeWizardAVR, un outil permettant de configurer automatiquement les registres[1] du microcontrôleur en fonction des applications souhaitées. Après avoir réglé quelques problèmes de compatibilité entre les deux versions de CodeVisionAVR (nom de bibliothèques changeantes), nous avons transféré le programme vers le microcontrôleur.

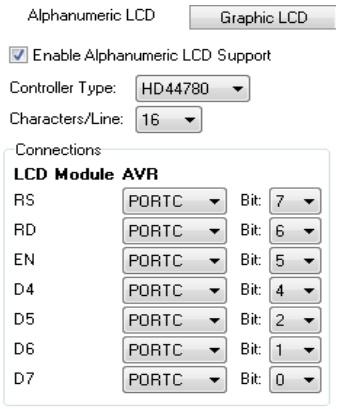
Nous nous sommes alors aperçu que l'affichage par l'écran LCD n'était pas géré. Nous nous sommes alors penchés sur le problème.

1 Registre : mémoire (groupe de bits) permettant de communiquer avec le microcontrôleur. Par exemple, on peut, grâce au registre PORTC modifier l'état des entrées/sorties du port C du microcontrôleur.

2.5. Gestion de l'affichage LCD

Après des recherches, nous avons compris que dans la version de l'IUT de CodeVision l'attribution des pattes de l'écran LCD par rapport aux pattes du microcontrôleur était automatique. Or, les personnes ayant réalisé le routage de la carte contenant l'écran LCD et le microcontrôleur n'ont pas pris en compte cette attribution automatique et n'ont donc pas relié ces pattes de la façon dont l'impose le logiciel. Refaire la carte prenant trop de temps, nous avons préféré prendre en compte leur attribution et s'y adapter. Nous avons donc changé de version de CodeVision et choisis la version 2.05.3 du logiciel. En effet, cette version permet de choisir, par l'intermédiaire de CodeWizardAVR, l'attribution des pattes du LCD par rapport au microcontrôleur. Pour connaître cette attribution, nous avons étudié le routage, le schéma du microcontrôleur (cf annexe 3) et les différentes pattes de l'écran LCD (cf annexe 3) à travers leurs documentations respectives. On en a déduit la configuration suivante :

L'écran LCD choisi est alphanumérique possédant 16 caractères par lignes et ses pattes sont reliées au port C du microcontrôleur comme on peut le voir sur l'illustration (la patte RS du LCD est reliée à la patte 7 du port C, etc...).



2.6. Tests sur la précision des mesures de tension

Après avoir réussi à configurer l'écran LCD et à afficher les tensions mesurées par le microcontrôleur, nous avons commencé à tester ces valeurs afin de vérifier le fonctionnement du programme. Pour ce test, nous nous sommes servis des générateurs de tension présents dans la salle.

Illustration 5: Générateur de tension

Sur le kart; les batteries sont disposées de la façon suivante :

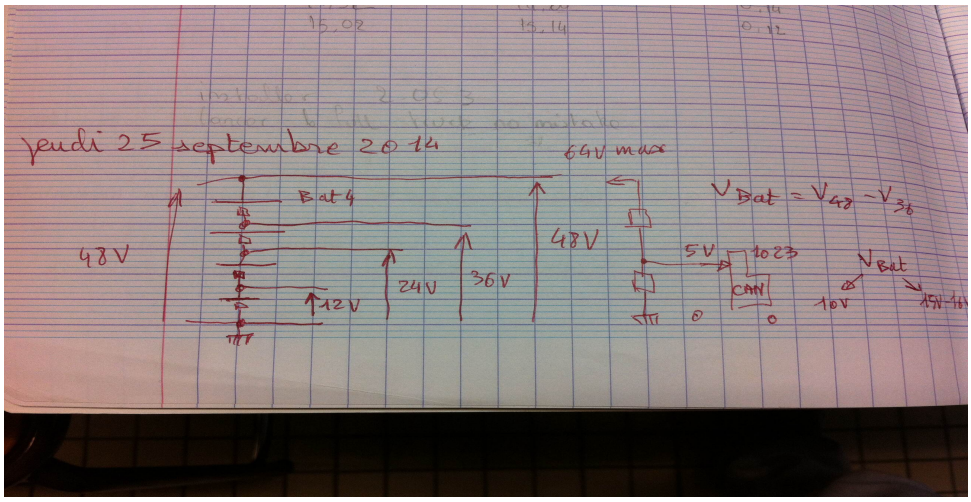


Illustration 6: schéma du montage des batteries

On a donc utilisé les 4 sorties de tensions de deux de ces générateurs que l'on a mis en série afin de simuler au mieux la réalité. Ensuite, on a branché aux bornes de chaque sortie des générateurs un voltmètre. En effet, après vérification, on a remarqué que les tensions affichées sur les générateurs de tensions n'étaient vraiment pas précises (jusqu'à 0,3V de différence avec la réalité). En faisant varier les différentes tensions des batteries entre 10 et 15V, nous avons pu comparer les tensions affichées sur l'écran LCD avec les tensions données par le Voltmètre. Les tableaux de mesures et les courbes résultantes de ces tensions pour les quatre batteries sont présents en annexe 5.

On peut remarquer que pour les batteries 12V et 36V, les tensions mesurées par le microcontrôleur et les tensions affichées par le Voltmètre sont quasiment identiques, à une constante près. Cependant, pour les deux autres batteries, ces tensions sont vraiment différentes.

On a compris que l'erreur venait du fait que dans leur programme, la technique de calcul des tensions n'était pas la bonne. En effet, pour chaque batterie, ils sommaient toutes les tensions précédentes et divisaient le tout par le nombre de tensions additionnées. Par exemple, pour calculer la tension de la batterie 36V, ils additionnaient les tensions des batteries 12V, 24V et 36V et divisaient la tension obtenue par 3. Ainsi, ils calculaient une moyenne des tensions au lieu de calculer la tensions aux bornes de chaque batterie. Nous avons donc commencé à étudier plus clairement la question du calcul des tensions.

2.7. Étude des calculs de tensions

Les batteries étant branchées en série, l'opération nécessaire à l'obtention des tensions aux bornes de chaque batterie est la soustraction. Il s'agissait donc de calculer les tensions entre la masse et la borne + de chaque batterie en série et de soustraire chaque résultat dans l'ordre afin d'obtenir seulement les tensions aux bornes de chacune des batteries. Voilà la partie de programme où l'on calcule les tensions associées aux quatre batteries :

```
204    v48-=v36;
205    v36-=v24;
206    v24-=v12;
```

Illustration 7: Calcul des tensions

Ainsi, pour calculer la tension aux bornes de la batterie 48V, on a soustrait la tension v48, tension correspondant à la somme des tensions aux bornes de chaque batterie (y compris aux bornes de la batterie 48V), à v36, la somme des tensions aux bornes des batteries 36V, 24V, 12V. On a ensuite suivi le même raisonnement afin de calculer les tensions aux bornes des autres batteries. On peut noter toutefois que les valeurs précédentes de tensions, qui correspondent aux différentes sommes de tensions sont écrasées afin de ne garder que les tensions aux bornes de chaque batterie (qui portent donc, après ces opérations, le même nom).

Ensuite, afin d'obtenir des calculs encore plus précis, on a décidé de recalculer la loi mathématique liant la valeur numérique donnée par le CAN et sa tension analogique associée. Pour cela, nous avons affiché sur l'écran LCD la valeur de sortie de la fonction read_adc(), c'est à dire la valeur numérique allant de 0 à 1023 en associant chacune de ces valeurs aux tensions correspondantes mesurées par les Voltmètres branchés aux bornes de chaque batterie.

A l'aide du générateur de tension, nous avons simulé les quatre batteries indépendamment. L'avantage de ces générateurs de tension est qu'ils disposent de plusieurs modes de fonctionnement. Nous en avons utilisé deux : le mode "Normal" et "Série". Le mode "Normal" permet de générer une tension sur les deux sorties égale à 32,4 V maximum. Nous l'avons donc utilisé pour simuler la batterie 12V. Le mode "Série" permet de générer une tension sur une seule sortie de l'ordre de 65 V. Nous l'avons donc utilisé afin de simuler les batteries 24V, 36V et 48V.

Dans le cadre du test; nous avons pris toutes les mesures pouvant être mesurées par le microcontrôleur dans cette application. En effet; les tensions minimales de chaque batterie sont de 10V et les tensions maximales, de 15V. Ainsi, les plages de mesure étaient :

- 10V-15V; pour la batterie 12V
- 20V-30V pour la batterie 24V
- 30V-45V pour la batterie 36V
- 40V-60V pour la batterie 48V

La relation entre ces deux grandeurs (valeur numérique en sortie du CAN et valeur analogique de tension) étant presque linéaire, on a décidé de la linéariser à l'aide du logiciel Excel afin de la simplifier. On a ensuite déterminé cette loi pour chaque batterie sous forme de coefficient directeur de droite. Voici un tableau représentant le coefficient directeur de la droite liant la tension numérique (en abscisse) et sa tension analogique associée (en ordonnée) ainsi que son coefficient de détermination[1] :

Batterie	Coefficient de détermination (R^2)	Équation de droite
12V	1	y=0,0157x-0,128
24V	0,9984	y=0,0329x-1184
36V	0,9999	y=0,0499x+0,0132
48V	1	y=0,0638x+0,0626

1 Coefficient de détermination : Indicateur qui permet de juger de la qualité d'une linéarisation. Il varie de 0 à 1. Plus il est proche de 1 et plus la linéarisation est fidèle à la réalité.

Les tableaux de mesures détaillés ainsi que leurs courbes associées se trouvent en annexe 5.

Voici la partie du programme (calculs) où l'on effectue ces opérations :

```
196   v12=((v12/nbmesure)*0.0157-0.128)*1000;
197
198   v24=((v24/nbmesure)*0.0329-0.1184)*1000;
199
200   v36=((v36/nbmesure)*0.0499+0.0132)*1000;
201
202   v48=((v48/nbmesure)*0.0638+0.0626)*1000;
```

Illustration 8: calcul des tensions entre la masse et chaque batterie à partir des fonctions trouvées avec le logiciel excel

Grâce à ces équations, il est alors facile de "convertir" les tensions. En effet, pour trouver les tensions analogiques associées aux tensions numériques résultantes des fonctions read_acd(), il suffit de "remplacer" les abscisses (les "x") par ces dernières. Il ne restait plus qu'à soustraire les différentes tensions mesurées afin d'obtenir les tensions propres à chaque batterie.

Après avoir fait tous ces calculs, nous avons transféré le programme dans le microcontrôleur. On a alors observé que plus rien ne fonctionnait car on avait dépassé la capacité de calcul de ce dernier en lui faisant faire trop de calculs avec des float (décimaux). Nous avons donc cherché un méthode de calcul alternative afin de limiter la complexité des calculs.

2.8. Gestion et Affichage des tensions

Après avoir réfléchi sur la façon de gérer au mieux les tensions pour limiter le nombre de calculs du microcontrôleur, on a trouvé, avec l'aide de M. LEQUEU une technique simple : on a redéfini le type de toutes les tensions en entiers. En effet, les calculs d'entiers sont beaucoup moins complexes que les calculs de décimaux. Les tensions n'étaient alors plus considérées en Volt mais en millivolt. En traitant ces tensions en millivolt, il a été facile de les afficher en Volt. Voici la partie du programme qui réalise cette opération (exemple de la batterie 24V) :

```
231    nl=v24/1000;
232    sprintf(tampon,"%3d",nl);
233    lcd_gotoxy(9,1);
234    lcd_puts(tampon);
235    lcd_putsf(",");
236    nl=abs((v24/10))%100;
237    sprintf(tampon,"%2d",nl);
238    lcd_puts(tampon);
239    lcd_putsf("V");
```

Illustration 9: Affichage des tensions en Volt à partir des tensions en milliVolt

Pour réaliser l'affichage des tensions en Volt, il a fallu utiliser une variable supplémentaire, de type entier. Cette variable nous a permis de traiter la tension sans la modifier directement . Dans un premier temps, on assigne à n1, notre variable tampon, la tension v24, (tension correspondant, à cette étape du programme, à la différence de potentiel entre les bornes de la batterie 24V) divisée par 1000. Cette tension (mV) étant un entier, après l'avoir divisé par 1000, il ne reste plus que sa partie entière, en Volt. N1 prend donc la valeur de la tension v24 exprimée en Volt (sans les décimales). On a ensuite affiché cette valeur à son emplacement sur l'écran LCD à l'aide des fonctions de la bibliothèque "alcd". Il ne nous restait donc plus qu'à afficher les décimales (dixième et centième) de la tension v24 (en Volt). Après avoir placé un virgule après la partie entière, nous avons donc récupéré ces derniers. Pour cela, nous avons assigné à n1 (et donc écrasé l'ancienne valeur de n1) les deux derniers chiffres de la partie entière de la tension v24 (exprimée en mV) divisée par 10, à l'aide de l'opération mathématique "modulo" 100. Nous avons relevé la partie entière de v24 divisée par 10. Enfin, nous les avons affichés après la virgule et avons placé un V pour indiquer que la tension indiquée était en Volt.

2.9. Tests finaux, câblage et test sur kart

Après avoir corrigé le problème des calculs et programmé l'affichage des tensions aux bornes de chaque batterie, nous avons réalisé les derniers tests afin de s'assurer du bon fonctionnement du produit. Pour cela, on a simulé à nouveau la configuration en série des quatre batterie. Ensuite, on a comparé les tensions mesurées et affichées sur l'écran LCD par le microcontrôleur aux bornes de chaque batterie avec les tensions mesurées par le Voltmètre aux bornes de chaque batterie. Afin de réaliser ces tests avec une bonne précision, nous nous sommes servis du Voltmètre MX579, pouvant afficher une tension au millième près :

Illustration 10: Voltmètre MX579

Les tableaux de valeurs détaillés représentant les tensions mesurées par le microcontrôleur, le Voltmètre et l'écart entre ces deux grandeurs sont en annexe 6.

On a pu remarquer que cet écart était constant, de l'ordre de 0,2V en moyenne. On a donc directement, dans le calcul de ces tensions, réajusté en ajoutant, ou en, enlevant cet écart (en rouge). Voici l'exemple du calcul dans le programme de la tension aux bornes de la batterie 24V :

```
196   v12=(((v12/nbmesure)*0.0157-0.128)+0.1)*1000;
197
198   v24=(((v24/nbmesure)*0.0329-0.1184)+0.3)*1000;
199
200   v36=(((v36/nbmesure)*0.0499+0.0132)+0.2)*1000;
201
202   v48=(((v48/nbmesure)*0.0638+0.0626)+0.4)*1000;
```

Illustration 11: Ajustement des tensions

Ces ajustements effectués, nous avons refait des tests, sans retranscrire leur résultat dans un tableau afin de s'assurer que notre modification avait bien fonctionné. Les tensions affichées sur le LCD correspondaient aux tensions affichées sur le Voltmètre avec une erreur de moins de 0,1V pour la plupart des mesures.

Nous avons donc validé le programme et nous sommes passés à la réalisation du boîtier destiné à contenir la carte.

Pour transmettre les tensions des batteries du kart vers le microcontrôleur, nous avons utilisé le câblage normalisé de la prise remorque, présente sur le kart :

Illustration 13: Photo prise remorque

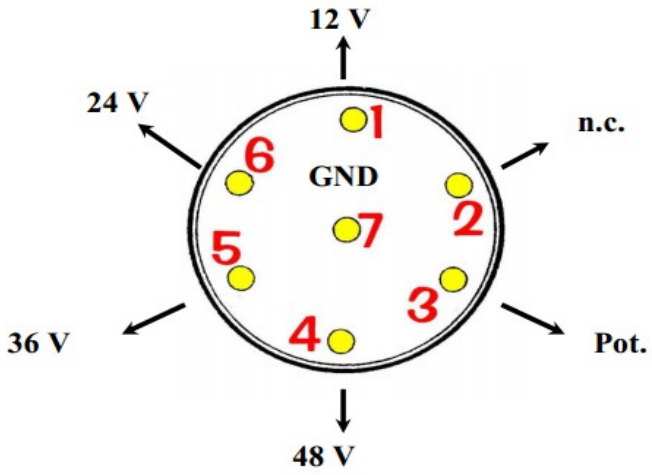


Illustration 12: Schéma prise remorque

Pour relier le microcontrôleur (borniers sur la carte) et le kart (prise femelle), nous avons utilisé une prise mâle. Cependant, nous ne connaissions pas la correspondance entre les fils à l'intérieur du câble et leur tension normalisée par rapport à la masse de la prise. Nous avons donc branché la prise remorque femelle reliée aux générateurs de tension

branchés en série (pour simuler les quatre batteries) à une prise remorque mâle, possédant un câble dénudé à une certaine distance de la prise afin de pouvoir mesurer la tension "de chaque fil" (différence de potentiel entre le fil et la masse du câble) et de pouvoir relier ces derniers au bornier de la carte plus tard. Nous avons donc obtenu le tableau suivant :

couleur du fil	batterie associée
noir	masse
jaune	12V
rouge	24V
marron	36V
vert	48V

Connaissant cette correspondance, on a ensuite pu relier chaque fil de la prise mâle au bornier de la carte en respectant l'ordre des batteries sur ce dernier. Puis, on a mis la carte dans le boîtier, percer ce dernier pour pouvoir y faire entrer le câble et fermer le tout de façon hermétique grâce aux vis et à un embout particulier.

Voici une photo du boîtier final :

Illustration 14: Photo vu-mètre avec boîtier et prise remorque

Enfin, après la conception du boîtier réalisée, on a testé le vu-mètre directement sur le kart. Les tests se sont montrés concluant et les erreurs de mesure de tensions étaient du même ordre que pendant les tests dans la salle.

3. vers une communication sans fil

Après avoir réalisé le Vu-mètre LCD, nous avons eu pour objectif d'étudier la communication sans fil de plusieurs Vu-mètres identiques

3.1. Etude de l'installation des modules XBEE

Après avoir réalisé le Vu-mètre LCD, nous avons eu pour objectif d'étudier la communication sans fil de plusieurs Vu-mètres identiques à celui-là. Le but était d'envoyer les tensions des batteries à un ordinateur lors du passage du kart près de la ligne d'arrivée. Étant disponibles à l'IUT, nous avons choisi d'utiliser les modules XBEE. Les modules XBEE sont des circuits permettant de communiquer via ondes radio à une fréquence avoisinant les 2,4Ghz (Wi-fi). De plus, ils ont une portée d'émission avoisinant les 100 mètres, ce qui suffit pour notre application. Pour pouvoir les utiliser, il nous a fallu étudier le montage nécessaire à leur utilisation. Nous avons donc fait l'étude sur un module et sur un microcontrôleur indépendamment de notre carte Vu-mètre. Tout d'abord, il a fallu alimenter la carte XBEE en 12V. Après avoir alimenté l'ATmega en 5V, nous avons branché la patte TX (transmission) de ce dernier sur la patte RX (réception) du module XBEE et inversement. Cependant, l'ATmega étant alimenté en 5V, nous ne pouvions pas directement appliquer cette tension sur le module XBEE (patte RX), lui, alimenté en 3V. On a donc utilisé un diviseur de tension entre la patte TX de l'ATmega et la patte RX du module XBEE. Enfin, on a relié les masses de ces deux composants.

3.2. Entrée d'instructions dans le module XBEE et tests de communication

3.2.1 Putty

Une fois l'étude du montage terminée, on a cherché un moyen d'entrer des instructions dans le module. On a d'abord téléchargé Putty, une interface de commande pour le module XBEE. Après l'avoir branché sur l'ordinateur via USB et cherché sur quel port de communication il était branché, on a lancé Putty. Cependant, ne connaissant pas les instructions connues du module, on a cherché dans sa documentation les principales :

- +++ : entrée dans le mode de commande
- ATID : nom du réseau du module
- ATMY : adresse source
- ATDH et ATMY : adresse destinataire (poids fort et faible)

On a donc entré le même nom de réseau pour les deux modules XBEE afin de les faire communiquer. Ensuite, on a entré les adresses source et destinataire de manière croisée, afin que les deux modules se « ciblent » mutuellement. Après les avoir paramétré, on a cherché à envoyer un caractère avec un des modules et de vérifier sa réception sur l'autre afin de s'assurer de la bonne communication entre les deux modules. On a donc récupéré un programme test de la communication UART[1] sur le site de M.LEQUEU qu'on a transféré sur le microcontrôleur à l'aide de CodeVisionAVR. On a alors observé que la communication entre les deux modules n'était pas assurée.En effet, la LED présente sur la carte XBEE USB « RX » de la carte 2 ne s'allumait pas (pas de réception) alors que la LED « TX » de la carte 1 était allumée (transmission).

1 UART : composant utilisé pour faire communiquer le port série avec l'ordinateur. Ici, la communication, entre l'ordinateur et le module XBEE est assurée par USB.

3.2.2 XCTU

Après cet échec, on a alors décidé de changer d'interface de communication. On a donc utilisé XCTU. Ce logiciel était plus simple d'utilisation que Putty car les instructions données aux modules sont transparentes. En effet, sur XCTU, les instructions données aux modules sont à rentrer dans des champs prédéfinis (ex : « Adresse du réseau : »).

Après avoir réglé quelques problèmes, notamment dans le téléchargement des drivers[2] pour les cartes USB XBEE, nous avons modifié les adresses des deux modules, comme précédemment afin de les faire communiquer. On a ensuite testé cette communication. Pour cela, on a utilisé la console du logiciel. La capture d'écran du logiciel se trouve en annexe 7. La partie de gauche est réservée à la transmission. C'est dans cette dernière qu'on envoie des données. La partie de droite, elle, est réservée à la réception. Tout ce qui est reçu sur le module se trouve dans cette partie.

Cependant, les tests de communication ont présenté les mêmes résultats que précédemment. On a alors complété les adresses destinataires (poids faible et fort) des modules avec les numéros de série (celle du 2 pour la configuration du module 1 et celle du 1 pour la configuration du module 2) comme on peut le voir en annexe 8. Ces modifications effectuées, les deux modules arrivaient à communiquer. On a donc assuré la communication des autres modules de la salle entre eux à l'aide de XCTU.

Enfin, on a voulu comprendre comment étaient configurés les modules de base. On a alors lu leur paramètres avec le logiciel. Ces derniers sont affichés en annexe 9. On a pu constater que les adresses source et destinataire était mises à 0. On a alors supposé qu'elles correspondaient à des adresses de broadcast, c'est à dire qu'avec ces adresses, les modules envoient leurs données à tous les modules à portée.

2 Driver : pilote en français. C'est un logiciel qui permet à l'ordinateur de reconnaître et d'utiliser un matériel informatique.

Conclusion

Grâce à ce projet nous avons réalisé deux produits finis prêts à être utilisés. Nous avons fait un vu-mètre permettant de visualiser le niveau d'une batterie 12 V grâce à 11 LED. Une utilisation pourrait être la mesure du niveau d'une batterie de voiture par le biais de la prise allume cigare. Notre deuxième produit est un Vu-mètre pour quatre batteries 12V montées en série. Ce dernier affiche la tension aux bornes de chacune des batteries sur un écran LCD intégré à la carte. Cette carte reliée à un kart électrique avec une prise remorque permet de tester les batteries de celui-ci. Nous avons également travaillé sur un système permettant de faire communiquer cette carte avec un ordinateur : les modules XBEE.

Nous avons vu dans la première partie la façon dont a été programmé la carte à LED grâce à une succession de tests if. Nous avons passé en revue les différentes étapes ainsi que les problèmes que nous avons rencontrés pour la conception du vu-mètre à écran LCD dans la seconde partie. Enfin nous avons vu comment faire fonctionner deux modules XBEE afin de transmettre des données entre une carte électronique et un ordinateur dans notre dernière partie.

L'étude que nous avons faite sur les deux cartes ainsi que sur la communication sans fil de celles-ci permettra donc de finaliser le projet de transmission de données depuis les différents karts d'un circuit vers l'ordinateur central. Il reste cependant à étudier le fonctionnement de plusieurs modules XBEE envoyant tous leurs données sur un même PC.

Résumé

Au cours de ce projet, nous avons programmé deux cartes, réalisé les boîtiers de chacune d'elles et commencé à faire communiquer une carte électronique avec un PC dans le but d'appliquer cette communication à notre vu-mètre. Nous nous sommes dans un premier temps penché sur la carte à LED pour laquelle il nous a fallu modifier le code déjà existant qui n'était pas fonctionnel, l'adapter au cahier des charges et enfin réaliser physiquement le boîtier en intégrant la prise allume cigare. Nous avons ensuite travaillé sur la deuxième carte à écran LCD, plus complexe qui devait afficher les tensions de quatre batteries. Pour cette dernière nous avons rencontré davantage de problèmes liés à la programmation. De plus le montage de la prise remorque sur le boîtier s'est avéré plus difficile que celui de la prise allume cigare. Nous avons enfin établi une communication entre deux modules XBEE ce qui permettra d'améliorer notre seconde carte pour qu'elle transmette des données sans fil à un ordinateur. Pour finir nous avons pris du recul sur le déroulement de notre projet et avons constaté des différences relativement importantes entre notre planning prévisionnel et notre planning réel. Ces différences étant probablement dues à la difficulté d'évaluation des différentes tâches à accomplir ainsi qu'aux problèmes rencontrés. De plus nous avons dû traiter une troisième partie que nous n'avions pas prévu au départ : la communication sans fil.

Table des illustrations

4. Annexes

4.1. Annexe 1 : programme carte 1

```
#pragma used+
sfrb TWBR=0;
sfrb TWSR=1;
sfrb TWAR=2;
sfrb TWDR=3;
sfrb ADCL=4;
sfrb ADCH=5;
sfrw ADCW=4;
sfrb ADCSRA=6;
sfrb ADMUX=7;
sfrb ACSR=8;
sfrb UBRRL=9;
sfrb UCSRB=0xa;
sfrb UCSRA=0xb;
sfrb UDR=0xc;
sfrb SPCR=0xd;
sfrb SPSR=0xe;
sfrb SPDR=0xf;
sfrb PIND=0x10;
sfrb DDRD=0x11;
sfrb PORTD=0x12;
sfrb PINC=0x13;
sfrb DDRC=0x14;
sfrb PORTC=0x15;
sfrb PINB=0x16;
sfrb DDRB=0x17;
sfrb PORTB=0x18;
sfrb PINA=0x19;
sfrb DDRA=0x1a;
sfrb PORTA=0x1b;
sfrb EECR=0x1c;
sfrb EEDR=0x1d;
sfrb EEARL=0x1e;
sfrb EEARH=0x1f;
sfrw EEAR=0x1e;
sfrb UBRRH=0x20;
sfrb UCSRC=0X20;
sfrb WDTCR=0x21;
sfrb ASSR=0x22;
sfrb OCR2=0x23;
sfrb TCNT2=0x24;
sfrb TCCR2=0x25;
sfrb ICR1L=0x26;
sfrb ICR1H=0x27;
sfrb OCR1BL=0x28;
sfrb OCR1BH=0x29;
sfrw OCR1B=0x28;
```

```
sfrb OCR1AL=0x2a;
sfrb OCR1AH=0x2b;
sfrw OCR1A=0x2a;
sfrb TCNT1L=0x2c;
sfrb TCNT1H=0x2d;
sfrw TCNT1=0x2c;
sfrb TCCR1B=0x2e;
sfrb TCCR1A=0x2f;
sfrb SFIOR=0x30;
sfrb OSCCAL=0x31;
sfrb OCDR=0x31;
sfrb TCNT0=0x32;
sfrb TCCR0=0x33;
sfrb MCUCSR=0x34;
sfrb MCUCR=0x35;
sfrb TWCR=0x36;
sfrb SPMCR=0x37;
sfrb TIFR=0x38;
sfrb TIMSK=0x39;
sfrb GIFR=0x3a;
sfrb GICR=0x3b;
sfrb OCR0=0X3c;
sfrb SPL=0x3d;
sfrb SPH=0x3e;
sfrb SREG=0x3f;
#pragma used-

#asm
        #ifndef __SLEEP_DEFINED__
        #define __SLEEP_DEFINED__
        .EQU __se_bit=0x40
        .EQU __sm_mask=0xB0
        .EQU __sm_powerdown=0x20
        .EQU __sm_powersave=0x30
        .EQU __sm_standby=0xA0
        .EQU __sm_ext_standby=0xB0
        .EQU __sm_adc_noise_red=0x10
        .SET power_ctrl_reg=mcucr
        #endif
#endasm

typedef char *va_list;

#pragma used+

char getchar(void);
void putchar(char c);
void puts(char *str);
void putsf(char flash *str);

char *gets(char *str,unsigned int len);

void printf(char flash *fmtstr,...);
void sprintf(char *str, char flash *fmtstr,...);
void snprintf(char *str, unsigned int size, char flash *fmtstr,...);
```

```
void vprintf (char flash * fmtstr, va_list argptr);
void vsprintf (char *str, char flash * fmtstr, va_list argptr);
void vsnprintf (char *str, unsigned int size, char flash * fmtstr, va_list argptr);
signed char scanf(char flash *fmtstr,...);
signed char sscanf(char *str, char flash *fmtstr,...);

#pragma used-

#pragma library stdio.lib

#pragma used+

void delay_us(unsigned int n);
void delay_ms(unsigned int n);

#pragma used-

unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (0x00 & 0xff);

delay_us(10);

ADCSRA|=0x40;

while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}

unsigned int TensionBat;
unsigned int i;

void main(void)
{

PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xFF;

PORTD=0x00;
DDRD=0xFF;

TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
```

```
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

MCUCR=0x00;
MCUCSR=0x00;

TIMSK=0x00;

ACSR=0x80;
SFIOR=0x00;

ADMUX=0x00 & 0xff;
ADCSRA=0x84;
SFIOR&=0xEF;

while (1)
{
TensionBat=0;

for(i=0;i<=60;i++)
{
TensionBat+=read_adc(0);

}
TensionBat=TensionBat/60;

if(TensionBat>=953)
{
PORTC.7=1;
}
else
{
PORTC.7=0;
}
if(TensionBat>=941)
{
PORTC.6=1;
}
else
{
PORTC.6=0;
}
if(TensionBat>=917)
{
PORTC.5=1;
```

```
}
else
{
PORTC.5=0;
}
if(TensionBat>=881)
{
PORTC.4=1;
}
else
{
PORTC.4=0;
}
if(TensionBat>=850)
{
PORTC.3=1;
}
else
{
PORTC.3=0;
}
if(TensionBat>=820)
{
PORTC.2 =1;
}
else
{
PORTC.2 =0;
}
if(TensionBat>=785)
{
PORTC.1 =1;
}
else
{
PORTC.1 =0;
}
if(TensionBat>=751)
{
PORTC.0 =1;
}
else
{
PORTC.0 =0;
}
if(TensionBat>=721)
{
PORTD.7 =1;
}
else
{
PORTD.7 =0;
}
if(TensionBat>=689)
{
```

```
PORTD.6 =1;
}
else
{
PORTD.6 =0;
}
if(TensionBat>=655)
{
PORTD.5 =1;
}
else
{
PORTD.5 =0;
}
}
};
```

4.2. Annexe 2 : programme carte 2

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.05.4 Evaluation
Automatic Program Generator
© Copyright 1998-2011 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 22/09/2014
Author  : Freeware, for evaluation and
non-commercial use only
Company :
Comments:


Chip type               : ATmega8535
Program type            : Application
AVR Core Clock frequency: 16,000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 128
*****************************************************/
#include <mega8535.h> //initialisation de l'ATmega
#include <delay.h> // initialisation des attentes
#include <alcd.h> // initialisation du LCD

//Initialisation des fonctions de base

#include <stdlib.h>
#include <stdio.h>

#define ADC_VREF_TYPE 0x00

//déclaration des variables
long  int n,n1;
long int v12,v24,v36,v48;

long  int nbmesure;
char      tampon [6];

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
```

```
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}



void main(void)
{
nbmesure=2000;

// 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

// 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

// 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

// 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

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=0xFF
// OC0 output: Disconnected

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=0xFFFF
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2 output: Disconnected

// External Interrupt(s) initialization
// INT0: Off
```

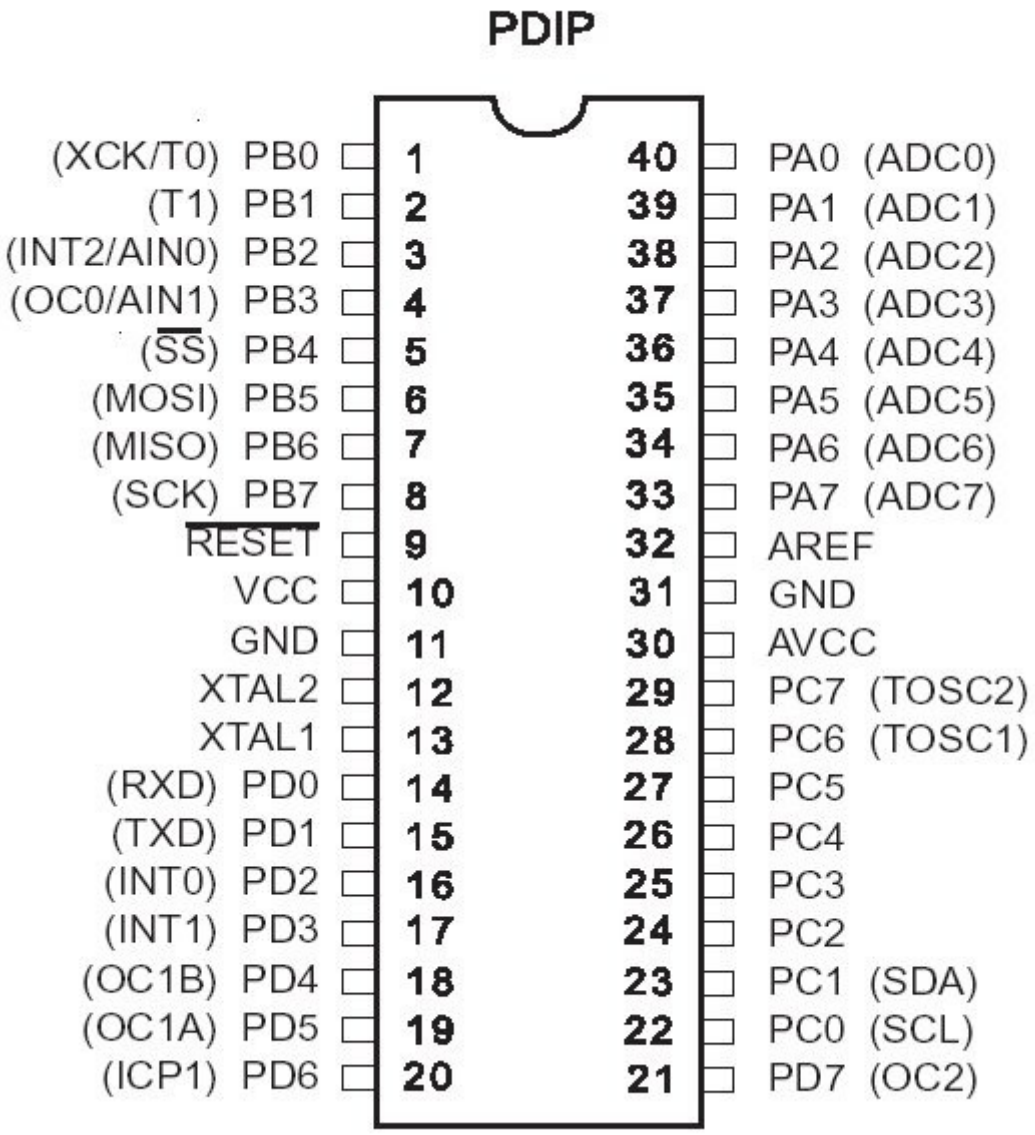
```
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// USART initialization
// USART disabled
UCSRB=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 1000,000 kHz
// ADC Voltage Reference: AREF pin
// ADC High Speed Mode: Off
// ADC Auto Trigger Source: ADC Stopped
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x84;
SFIOR&=0xEF;

// SPI initialization
// SPI disabled
SPCR=0x00;

// TWI initialization
// TWI disabled
TWCR=0x00;

// Alphanumeric LCD initialization
// Connections are specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTC Bit 7
// RD - PORTC Bit 6
// EN - PORTC Bit 5
// D4 - PORTC Bit 4
// D5 - PORTC Bit 2
// D6 - PORTC Bit 1
// D7 - PORTC Bit 0
// Characters/line: 16 ???

//initialisation du LCD
lcd_init(16);

//Ecriture de présentation
lcd_gotoxy(0,0); // Curseur premiere colonne premiere ligne
lcd_putsf("Bienvenue"); //Ecrire Bienvenue
lcd_gotoxy(0,1);
lcd_putsf("KartMaster");
```

```
lcd_gotoxy(0,2);
lcd_putsf("BC - BG - TL");
lcd_gotoxy(0,3);
lcd_putsf("");
delay_ms(1000); //Attendre 1 seconde
lcd_clear();

//Fonctionnement permanant
while (1)
{
//compter le nombre de cycle
n++;

//lecture de la sortie du CAN
v48+=(read_adc(1)); //48V
v36+=(read_adc(2)); //36V
v24+=(read_adc(3)); //24V
v12+=(read_adc(4)); //12V

//nombre de mesures souahitées atteintes
if (n==nbmesure)
{

//CALCULS DU GROUPE PRECEDENT

//transformation des bits en tension
/*n2=(n2/nbmesure)/15.58*100+12; //+48V->65V=5V
n3=(n3/nbmesure)/19.97*100+20; // 36V->50.5V=5V
n4=((n4/nbmesure+5))/33.25*100; // +24V->33V=5V
n5=((n5/nbmesure+14))/64.43*100; //+12V->16V=5V
*/

//NOS CALCULS

//transformation des bits en tension en mV
v12=(((v12/nbmesure)*0.0157-0.128)+0.1)*1000;

v24=(((v24/nbmesure)*0.0329-0.1184)+0.3)*1000;

v36=(((v36/nbmesure)*0.0499+0.0132)+0.2)*1000;

v48=(((v48/nbmesure)*0.0638+0.0626)+0.4)*1000;

v48-=v36;
v36-=v24;
v24-=v12;


//affichage des tension sur le lcd

n1=v48/1000;
sprintf(tampon,"%3d",n1); // convertit un int en chaine de caractere
lcd_gotoxy(9,3);
lcd_puts(tampon);
lcd_putsf(",");
```

```
n1=abs((v48/10))%100;
sprintf(tampon,"%2d",n1);
lcd_puts(tampon);
lcd_putsf("V");

n1=v36/1000;
sprintf(tampon,"%3d",n1); // convertit un int en chaine de caractere
lcd_gotoxy(0,3);
lcd_puts(tampon);
lcd_putsf(",");
n1=abs((v36/10))%100;
sprintf(tampon,"%2d",n1);
lcd_puts(tampon);
lcd_putsf("V");

n1=v24/1000;
sprintf(tampon,"%3d",n1); // convertit un int en chaine de caractere
lcd_gotoxy(9,1);
lcd_puts(tampon);
lcd_putsf(",");
n1=abs((v24/10))%100;
sprintf(tampon,"%2d",n1);
lcd_puts(tampon);
lcd_putsf("V");


n1=v12/1000;
sprintf(tampon,"%3d",n1); // convertit un int en chaine de caractere
lcd_gotoxy(0,1);
lcd_puts(tampon);
lcd_putsf(",");
n1=abs((v12/10))%100;
sprintf(tampon,"%2d",n1);
lcd_puts(tampon);
lcd_putsf("V");


sprintf(tampon,"%5d",v48); // conversion tension chaîne de caractère
lcd_gotoxy(0,3);
lcd_puts(tampon);

sprintf(tampon,"%5d",v36); // conversion tension chaîne de caractère
lcd_gotoxy(0,1);
lcd_puts(tampon);

sprintf(tampon,"%5d",v24); // conversion tension chaîne de caractère
lcd_gotoxy(10,1);
lcd_puts(tampon);

sprintf(tampon,"%5d",v12); // conversion tension chaîne de caractère
lcd_gotoxy(10,3);
lcd_puts(tampon);


lcd_gotoxy(10,2); lcd_putsf("BAT1 :");
```

```
lcd_gotoxy(10,0); lcd_putsf("BAT2 :");

lcd_gotoxy( 0,2); lcd_putsf(" BAT4 :");
lcd_gotoxy( 0,0); lcd_putsf(" BAT3 :");


lcd_gotoxy(1,0); lcd_putsf("BAT1 :");
lcd_gotoxy(10,0); lcd_putsf("BAT2 :");

lcd_gotoxy( 9,2); lcd_putsf(" BAT4 :");
lcd_gotoxy( 0,2); lcd_putsf(" BAT3 :");

//remise à zero des variables
n=0;
n1=0;
v12=0;
v24=0;
v36=0;
v48=0;
}
}
}
```

4.3. Annexe 3 : schéma pattes ATmega et tableau des pattes du LCD

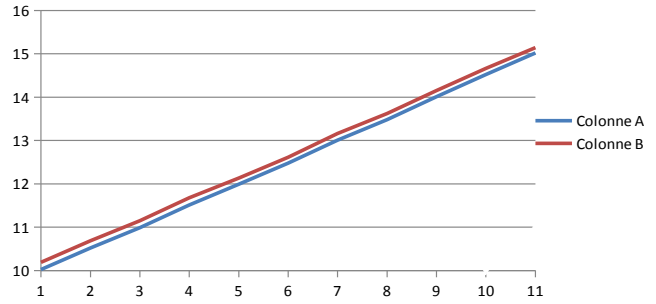


Interface pin connection

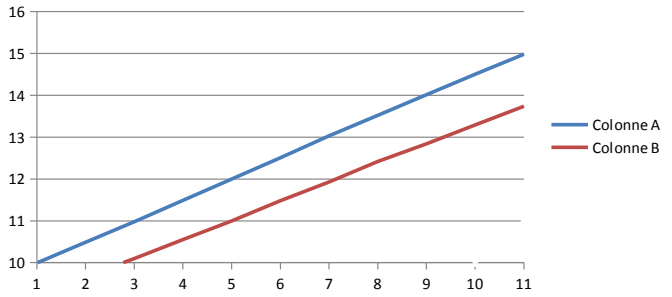
PIN NO.	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
SYMBOL	V_{SS}	V_{DD}	V_0	RS	R/$\overline{W}$	E	DB0	DB1
PIN NO.	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
SYMBOL	DB2	DB3	DB4	DB5	DB6	DB7	A/(V_{EL})	K/(V_{EL})

4.4. Annexe 4 : Tableaux et graphiques des valeurs de tensions du groupe précédent.

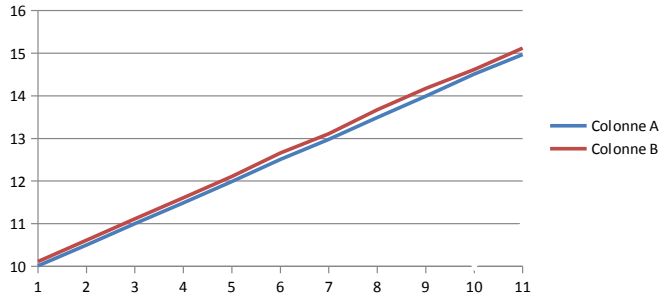
Tension mesurée batterie 12V (en V)	Tension affichée batterie 12V (en V)	Ecart de la batterie 12V (en V)
10,02	10,19	0,17
10,52	10,69	0,17
10,99	11,15	0,16
11,51	11,68	0,17
11,99	12,13	0,14
12,48	12,61	0,13
13,01	13,16	0,15
13,48	13,62	0,14
14,01	14,15	0,14
14,52	14,66	0,14
15,02	15,14	0,12



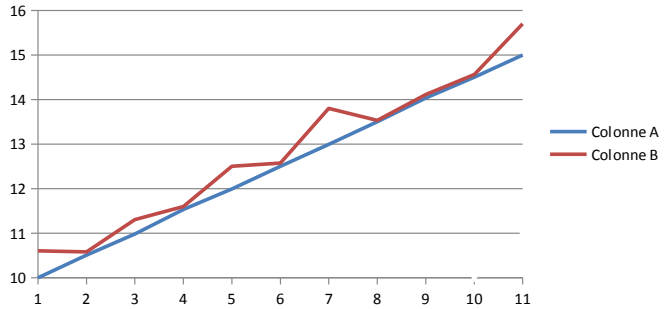
Tension mesurée batterie 24V (en V)	Tension affichée batterie 24V (en V)	Ecart de la batterie 24V (en V)
9,99	9,2	-0,79
10,49	9,65	-0,84
10,98	10,1	-0,88
11,49	10,55	-0,94
12	11	-1
12,51	11,48	-1,03
13,03	11,93	-1,1
13,52	12,42	-1,1
14,01	12,84	-1,17
14,5	13,29	-1,21
14,98	13,74	-1,24



Tension mesurée batterie 36V (en V)	Tension affichée batterie 36V (en V)	Ecart de la batterie 36V (en V)
10,01	10,11	0,1
10,5	10,61	0,11
11	11,11	0,11
11,49	11,61	0,12
11,99	12,11	0,12
12,51	12,66	0,15
12,98	13,11	0,13
13,49	13,67	0,18
13,99	14,17	0,18
14,51	14,62	0,11
14,97	15,12	0,15

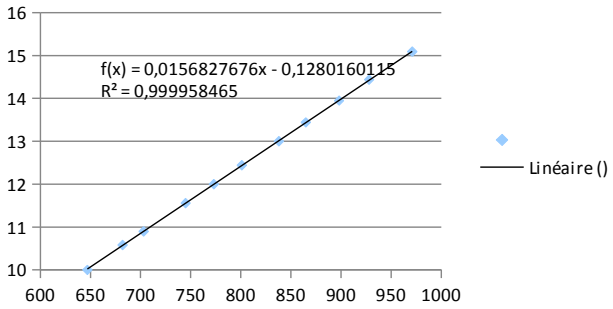


Tension mesurée batterie 48V (en V)	Tension affichée batterie 48V (en V)	Ecart de la batterie 48V (en V)
10	10,6	0,6
10,51	10,58	0,07
10,98	11,3	0,32
11,53	11,6	0,07
11,99	12,5	0,51
12,5	12,57	0,07
12,99	13,8	0,81
13,5	13,53	0,03
14,03	14,11	0,08
14,5	14,56	0,06
15	15,7	0,7

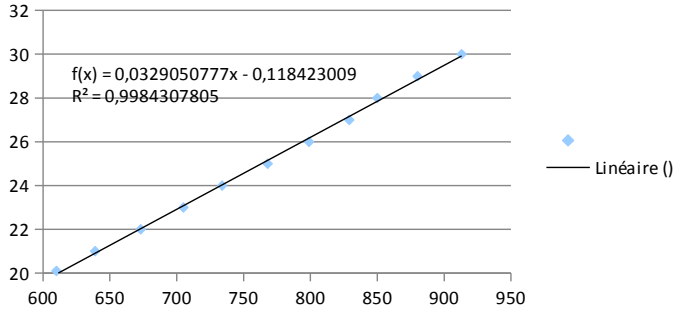


4.5. Annexe 5 : Tableaux et graphiques des tests des différentes batteries

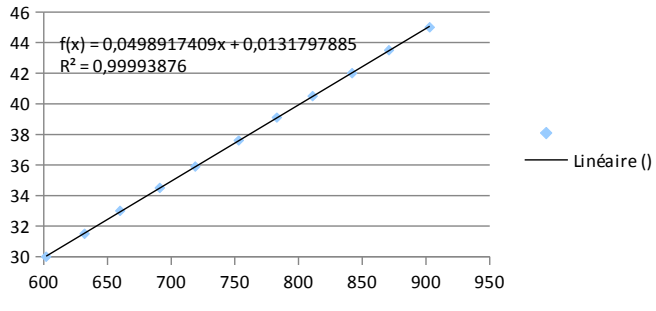
Tension mesurée (V) Batterie 1	Valeur numérique affichée Batterie 1
10	647
10,58	682
10,9	703
11,56	745
12	773
12,44	801
13	838
13,44	865
13,95	898
14,44	928
15,09	971



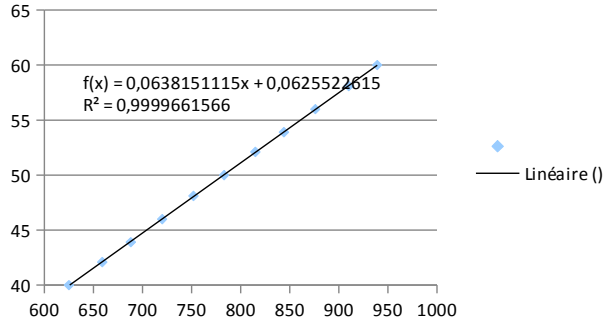
Tension mesurée (V) Batterie 2	Valeur numérique affichée Batterie 2
20,1	610
21	639
22	673
23	705
24	734
25	768
26	799
27	829
28	850
29	880
30	913



Tension mesurée (V) Batterie 3	Valeur numérique affichée Batterie 3
30	602
31,5	632
33	660
34,5	691
35,9	719
37,6	753
39,1	783
40,5	811
42	842
43,5	871
45	903



Tension mesurée (V) Batterie 4	Valeur numérique affichée Batterie 4
40	625
42,1	659
43,9	688
46	720
48,1	752
50	783
52,1	815
53,9	844
56	876
58,1	910
60	939



4.6. Annexe 6 : Tableaux représentant les mesures des tests finaux

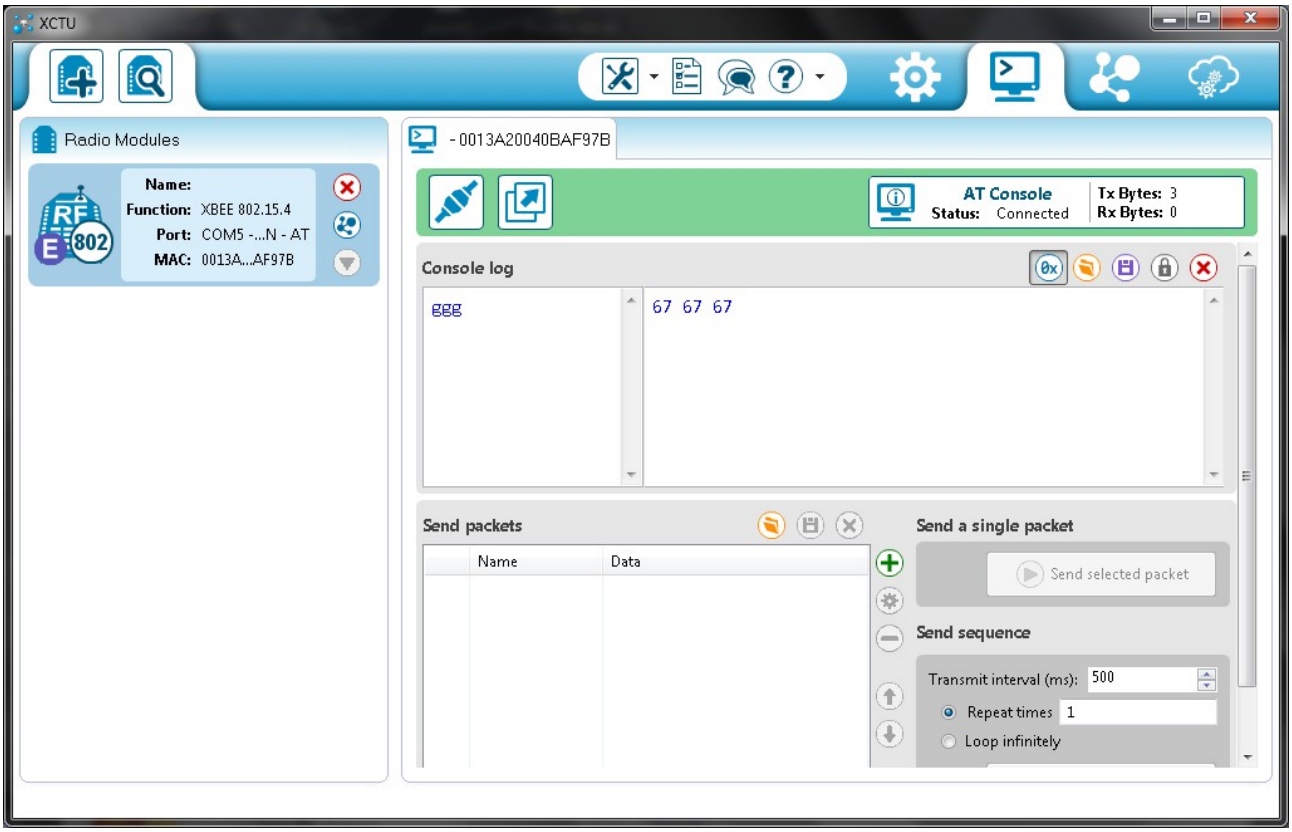
Tension mesurée (V) Batterie 1	Tension affichée (V) Batterie 1	Ecart mesuré-affiché
10,001	9,9	0,101
10,462	10,37	0,092
10,997	10,92	0,077
11,498	11,42	0,078
12,013	11,92	0,093
12,421	12,35	0,071
12,947	12,87	0,077
13,453	13,37	0,083
14,043	13,95	0,093
14,505	14,42	0,085
15,031	14,92	0,111

Tension mesurée (V) Batterie 2	Tension affichée (V) Batterie 2	
10,089	9,88	0,209
10,535	10,34	0,195
11,027	10,83	0,197
11,498	11,26	0,238
11,954	11,72	0,234
12,558	12,31	0,248
13,042	12,84	0,202
13,537	13,3	0,237
14,03	13,83	0,2
14,493	14,26	0,233
14,998	14,81	0,188

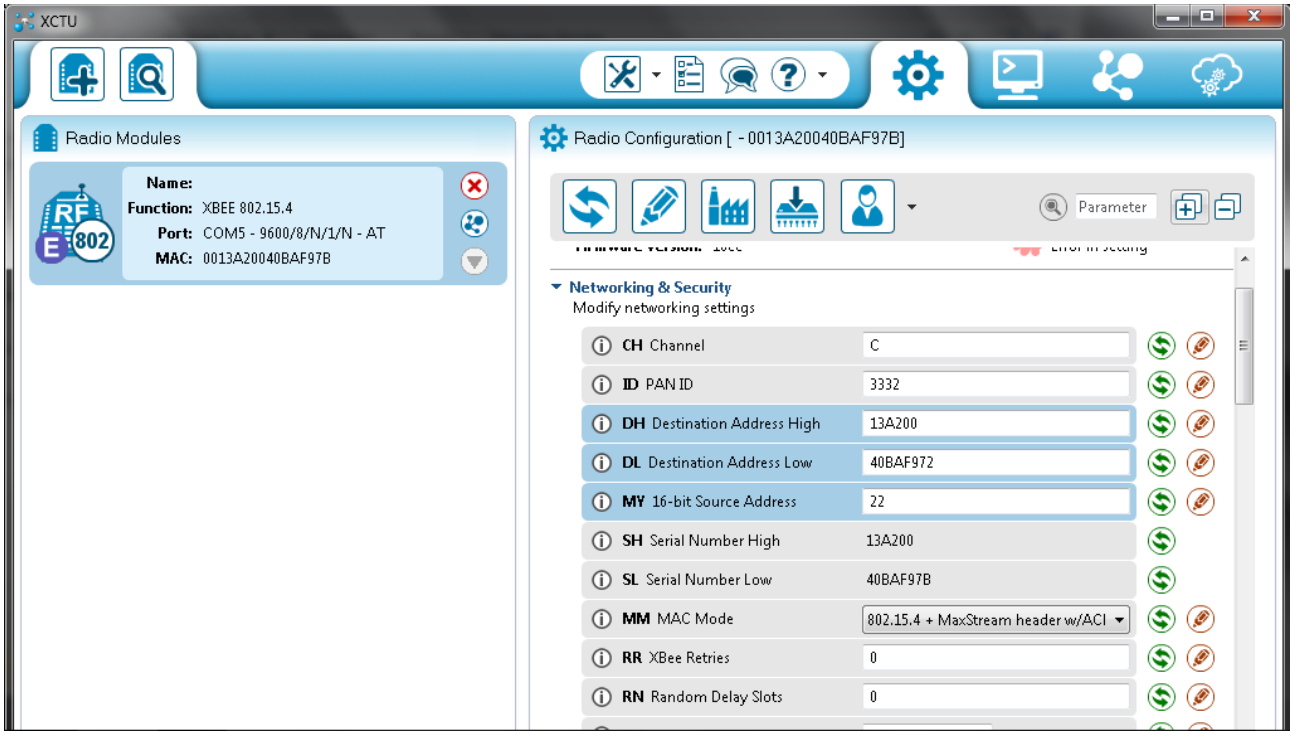
Tension mesurée (V) Batterie 3	Tension affichée (V) Batterie 3	
10,065	10,25	-0,185
10,497	10,6	-0,103
11,076	11,22	-0,144
11,49	11,67	-0,18
11,929	12,7	-0,771
12,547	12,67	-0,123
13,07	13,21	-0,14
13,538	13,66	-0,122
14,031	14,11	-0,079
14,508	14,66	-0,152
15,031	15,6	-0,569

Tension mesurée (V) Batterie 4	Tension affichée (V) Batterie 4	
10,05	9,93	0,12
10,518	10,44	0,078
10,981	10,89	0,091
11,459	11,4	0,059
12	11,92	0,08
12,495	12,43	0,065
12,999	12,88	0,119
13,496	13,34	0,156
13,995	13,9	0,095
14,507	14,41	0,097
15,04	14,92	0,12

4.7. Annexe 7 : Mode console du logiciel X-CTU



4.8. Annexe 8 : Configuration des adresses du module XBEE



4.9. Annexe 9 : Configuration des adresses du module XBEE neuf