Université François Rabelais de Tours

Institut Universitaire de Technologie de Tours

Département Génie Électrique et Informatique Industrielle

PROJET TUTORÉ

Afficheur pour e-Kart

Stan KOCKEN
Grégoire DOLBEAU
2ème Année – Q2
Promotion 2006/2008

Enseignants :
Thierry LEQUEU
Inès DA COSTA

Université François Rabelais de Tours

Institut Universitaire de Technologie de Tours

Département Génie Électrique et Informatique Industrielle

Afficheur pour e-Kart

Stan KOCKEN
Grégoire DOLBEAU
2ème Année – Q2
Promotion 2006/2008

Enseignants :
Thierry LEQUEU
Inès DA COSTA

Table des matières

Introduction

Le principal objectif des instituts universitaires de technologie, et notamment des départements génie électrique et informatique industrielle, est de préparer leurs étudiants aux fonctions de technicien supérieur. Pour cela, nous, étudiants, disposons d'un enseignement visant à nous préparer à une insertion professionnelle imminente. Par conséquent, à travers la matière d'Études et Réalisations, nous sommes chargés de réaliser un projet tutoré. Ce dernier va nous permettre de nous familiariser à la conduite de projets, ce qui constituera une bonne préparation à nos futures expériences professionnelles.

L'objet de notre travail en Études et Réalisations, ce semestre, est axé sur l'affichage de diverses informations sur un afficheur LCD pour un karting électrique, tel que la vitesse, la température moteur, la tension de la batterie et autres. Ce rapport, à l'aide de différentes parties, va permettre de mieux comprendre le projet et les analyses techniques que nous avons réalisées pour le mener à bien.

1. Cahier des charges

1.1. Présentation

Le projet qui nous était confié lors de ce 4ème semestre répond à un besoin du club karting de notre I.U.T.. En effet, il nous a été demandé de continuer le projet de nos camarades du précédent semestre afin de réaliser toute la partie programmation d'un afficheur pour les kartings électrique de l'I.U.T.. Ainsi, suite à la finalisation de notre projet, un utilisateur de ce karting sera en mesure d'accéder en permanence à de nombreuses informations sur son véhicule afin de l'exploiter de manière optimale.

Le projet doit comporter les fonctions suivantes:

- Affichage de la vitesse: ceci permet à l'utilisateur de savoir de manière précise à quelle vitesse en km/h il roule sur telle ou telle partie du circuit.
- Affichage de la température du moteur: c'est un des paramètres les plus importants. En effet, avec cette information, l'utilisateur pourra voir en permanence s'il effectue une charge de travail trop importante par rapport à ce que le moteur peut supporter.
- Affichage de la tension batterie: comme sur tout appareil électrique à batterie, il est judicieux de savoir en temps réel, quel en est l'état afin de choisir le bon moment pour la recharger et ne pas se laisser surprendre en pleine course.
- Faire une fonction BOOST: à l'image du NOS[1] en course automobile, cette fonction doit apporter une augmentation du courant qui accélère fortement la rotation du moteur et donc d'avoir un gain en vitesse important. Ce gain en courant doit être contrôlé dans le temps car si cette fonction est activée trop longtemps, il y a alors un risque d'endommagement du moteur.

Notre travail consiste donc à programmer cet afficheur en utilisant un microcontrôleur de type « AtMega 8535 » afin de donner suite au projet du 3ème semestre de nos camarades qui ont réalisé l'étude et la conception de cette carte.

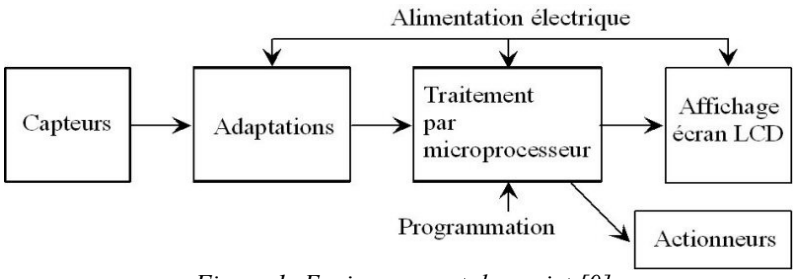


Figure 1: Environnement du projet [0]

1 NOS: Nitrous Oxide Systems, qui est du protoxyde d'azote utilisé en course automobile pour favoriser la combustion du moteur grâce à son taux d'oxygène plus élevé que celui de l'air. Il a pour effet d'augmenter de manière significative la puissance du moteur (de 30% à 100% suivant le cas).

1.2. Base du projet

La carte sur laquelle nous avons travaillé est un projet réalisé par d'autres étudiants le semestre précédent. Nous avons utilisé la deuxième version de cet afficheur LCD, qui a pour spécificités :

- Afficheur LCD (4 lignes de 16 caractères) relié par une liaison série à la carte microcontrôleur.
- Une alimentation continue qui peut être située entre 36 et 75V.
- 2 timers
- 8 entrées analogiques (dont 2 pour le courant et 1 pour la température)

1.3. Logiciel

Afin de réaliser notre afficheur, nous avions accès à un logiciel nommé « CodeVisionAVR studio », qui permet de programmer en langage C puis de compiler et de transférer directement vers le microcontrôleur.

Ce logiciel a la particularité d'avoir une interface graphique dès le lancement du programme, qui nous demande quel est notre matériel et de quoi nous aurons besoin. Ainsi, il génère le code source de base directement.

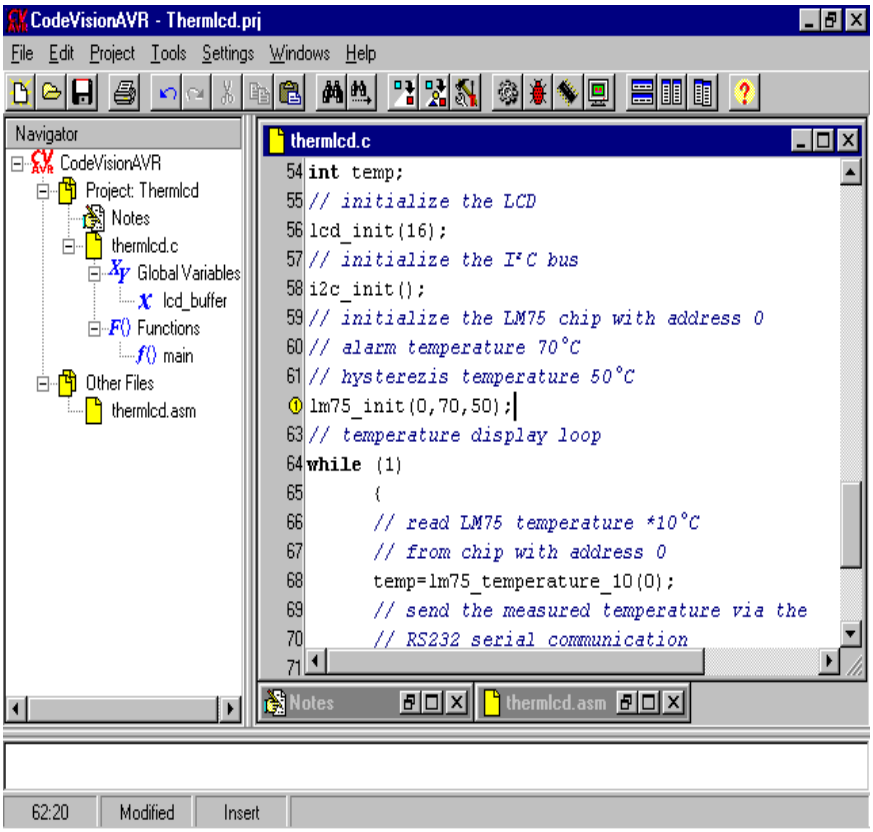


Illustration 1: Visualisation du logiciel AVR[0]

2. Principe de fonctionnement

2.1. Introduction

À l'image d'autres programmes, nous avons décidé de faire une mini-présentation au lancement de notre afficheur afin d'en rappeler le nom du projet, les noms des créateurs et la date de création du projet.

Code

```
// Initialisation du module LCD
lcd_init(16);

// On place notre texte « visuKart » à l'adresse 0,0
lcd_gotoxy(0,0);
lcd_putsf("      VisuKart      ");
lcd_gotoxy(0,1);
lcd_putsf("Stan K      Greg D");
lcd_gotoxy(0,2);
lcd_putsf(" le 19/03/2008   ");

lcd_gotoxy(0,3);
lcd_putsf("           .           ");
// Une petite attente pour faire penser à un chargement
delay_ms(400);

lcd_gotoxy(0,3);
lcd_putsf("           ..          ");
delay_ms(400);

lcd_gotoxy(0,3);
lcd_putsf("          ...          ");
delay_ms(400);

lcd_gotoxy(0,3);
lcd_putsf("          ...          ");
delay_ms(400);
```

2.2. Tension batterie

Il nous a été demandé que notre afficheur puisse indiquer la tension des batteries du karting, afin de savoir si l'on a une consommation normale en tension, et de savoir approximativement, quand il sera nécessaire de les recharger.

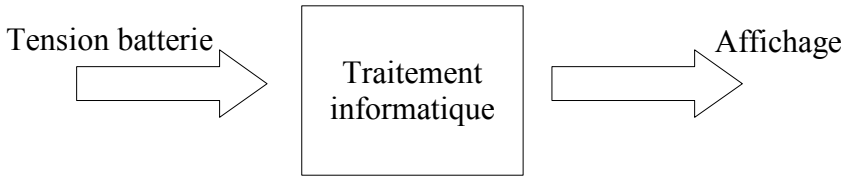


Figure 2: Schéma fonctionnel de l'affichage de la tension batterie[1]

Afin de réaliser cette partie, nous avons cherché à étudier la fonction de transfert entre l'entrée en tension d'alimentation et celle mesurée par le microcontrôleur. Cela nous a heurté à

un premier problème: la fonction n'était pas linéaire. On peut le voir ici avec la droite théorique qui n'apporte pas des résultats satisfaisants.

Valeur S	Voltage E	ax+b
835	65,5	62,92
804	60,3	59,95
768	55,2	56,51
719	49,9	51,81
663	45	46,45
598	40,1	40,23
526	35,2	33,33
	A=	0,1
	B=	-17,04

Illustration 2: Tableau de mesures tension batterie avec diode zener[1]

Après une discussion avec M. Lequeu, nous avons pu voir qu'une diode zener était sûrement l'élément perturbateur de notre fonction de transfert. Nous l'avons donc supprimé et refait des tests, qui ont cette fois été meilleurs.

Sortie	Tension alim	ax+b
1000	66,11	66,06
896	59	59,12
830	54,75	54,72
753	49,6	49,58
615	40,44	40,38
535	35	35,04
	a =	0,07
	b =	-0,64

Illustration 3: Tableau de mesures tension batterie sans diode zener[1]

On a donc pu codé:

Code

```
// Partie batterie


    // Lecture de la tension batterie sur l'entrée ADC7
i=read_adc(7);
    // Conversion de la tension lu en tension batterie
conversion = (float)((float)i*0.0666915732-0.6358820244)*10;
    // Préparation pour l'affichage
sprintf(tampon,"%2i.%i",(int)(conversion/10), (int)(conversion-(int)(conversion/10)*10));
    // On se déplace vers la position désirée
lcd_gotoxy(2,1);
    // On affiche la valeur de la tension batterie
lcd_puts(tampon);
```

2.3. Accélération

L'accélération est mesurée grâce à une résistance variable produite par la pédale d'accélérateur qui, en fonction de la pression de l'utilisateur, renvoie grâce à un montage sur notre carte une tension image.

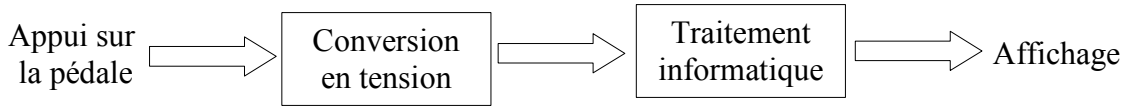


Figure 3: Schéma fonctionnel de l'affichage de l'accélération[1]

Nous avons réalisé des tests afin de voir la valeur mesurée par le microcontrôleur en fonction de l'accélération et nous avons pu ainsi remarqué que celle-ci était simplement 6 fois plus grande, ce qui nous a amené à:

Code

```
// Partie accélérateur


    // Lecture de la tension image de l'accélérateur sur l'entrée ADC2
i=read_adc(2);
    // Préparation pour l'affichage
sprintf(tampon,"%3d",(int)(i/6));
    // On se déplace vers la position désirée
lcd_gotoxy(12,0);
    // On affiche la valeur de l'accélération
lcd_puts(tampon);
```

2.4. Température

La carte, sur laquelle nous avons travaillé, est équipée d'un bornier sur lequel l'on branche directement un capteur LM75 et qui convertit la température en une tension image.

Figure 4: Schéma fonctionnel de l'affichage de la température[1]

Un programme « type » pour cette application est fourni sur le site de M. Lequeu, ce qui nous a permis de recopier simplement le code afin de l'exploiter dans notre programme. Tout d'abord, une initialisation est nécessaire dans le programme:

Code

```
// LM75 Temperature Sensor initialization
// thyst: 35°C
// tos: 40°C
// O.S. polarity: 0
lm75_init(7,35,40,0);
```

Puis une partie dans la boucle « while(1) » pour l'affichage:

Code

```
// Partie temperature


    // Lecture de la tension image de l'accélérateur sur l'entrée ADC2
temp=lm75_temperature_10(0);
    // Préparation pour l'affichage
sprintf(tampon,"%2i.%1u",temp/10,temp%10);
    // On se déplace vers la position désirée
lcd_gotoxy(10,1);
    // On affiche la valeur de l'accélération
lcd_puts(tampon);
```

2.5. Compteur de vitesse

Cette fonction est la principale fonction attendue sur notre afficheur. Mais ce n'est pas la partie qui a été traitée en priorité à cause de sa complexité. En effet, afin de mettre en œuvre la mesure de vitesse, une étude plus longue a été nécessaire afin de comprendre comment nous pouvions avec nos outils, réaliser une telle chose.

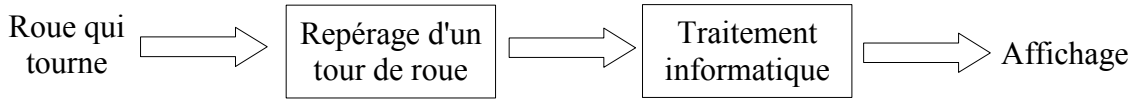


Figure 5: Schéma fonctionnel du compteur de vitesse[1]

Afin de réaliser le repérage d'un tour de roue, comme montré sur le schéma ci-dessus, on place sur l'arbre moteur un aimant, puis sur un socle juste au-dessus, un interrupteur électromagnétique (enclenché par le passage de l'aimant).

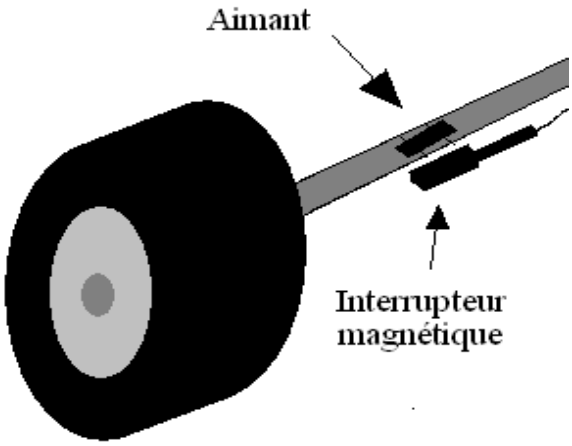


Illustration 4: Fixation du compteur d'impulsion[1]

La première idée qui nous est venue, était de compter le nombre d'impulsions qu'il y avait en un temps donné. Par exemple, combien d'impulsions il y avait en 1 seconde. Or, cette méthode n'aurait pas eu une précision suffisante à basse vitesse car le nombre d'impulsions aurait été trop faible pour être significatif.

Après discussion, nous avons opté pour une seconde méthode, qui consiste à compter le temps entre deux impulsions et trouver une formule permettant de transformer ce temps en une vitesse en km/h. Nous avons relevé les informations suivantes:

Rayon de la roue = 13,5cm = 0,135m

Périmètre = Pi * 2 * rayon = 0,84823 m/tr

Notre karting effectue donc 0,84823 mètre à chaque tour de roue, et donc il se passe 0,84823 mètre entre chaque pulsation.

- À 100km/h (= 27,78 m/s), il faudrait t secondes pour effectuer un tour de roue, où t=d/v donc t = 0,84823 / 27,78 = 0,03053384 seconde
- À 10km/h (= 2,778 m/s), il faudrait t secondes pour effectuer un tour de roue, où t=d/v donc t = 0,84823 / 2,778 = 0,30533837 seconde

Nous devons utiliser un timer ainsi qu'une interruption pour effectuer cette tâche afin de ne pas bloquer le programme pendant le comptage du temps. Les timers disponibles fonctionnent avec une horloge de 16MHz, sur laquelle est appliquée des diviseurs de fréquence. De plus, le timer est capable de compter sur 16 bits, soit de 0 à 65535. Ainsi, nous avions le choix entre plusieurs fréquences de comptage. Nous avons pris une fréquence de 62,5kHz afin de pouvoir compter de 0,000016 seconde (16µs) à 1,04856 seconde. Toutes les 16µs, le timer incrémente automatiquement une variable nommée TCNT1 qui peut compter de 0 à 65535 également. Nous avons décidé de relever cette valeur de TCNT1 à chaque tour de roue puis de remettre ce compteur à 0. Ainsi, la vitesse est directement proportionnelle à ce nombre.

vit = nombre d'impulsions entre chaque passage de roue

- À 100km/h (= 27,78 m/s), t = 0,03053384 secondes et donc

vit = t / (16us) = 1908 impulsions

- À 10km/h (= 2,778 m/s), t = 0,30533837 secondes et donc

vit = t / (16us) = 19083 impulsions

Vitesse = vit * a + b

100 = 1908 * a + b donc a = - 0.005333333

10 = 19083 * a +b et b = 110

On a donc pu programmer ceci, en déclarant la fonction d'interruption comme ceci:

Code

```
int vit;
interrupt [EXT_INT1] void ext_int1_isr(void)
{
// Place your code here
if((TCNT1>0)&&(TCNT1<5155))
{
vit = TCNT1;
}
TCNT1 = 0x00;
}
```

Et en paramétrant le timer et l'interruption comme ceci:

Code

```
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 62,500 kHz
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x04;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: On
// INT1 Mode: Falling Edge
// INT2: Off
GICR|=0xC0;
MCUCR=0x0A;
MCUCSR=0x00;
GIFR=0xC0;
```

Tous ces paramètres ont été fournis automatiquement par un calcul du logiciel AVR studio en fonction de nos besoins. Il ne nous reste ensuite qu'à l'afficher avec cette fonction:

Code

```
// Partie vitesse
        sprintf(tampon,"%3i",(int)((float)vit*(float)(-0.005333333)+(float)110));
        lcd_gotoxy(0,0);
        lcd_puts(tampon);
```

2.6. Chronomètre

Il nous restait à programmer la fonction BOOST, mais nous savions déjà que cette fonction ne serait pas fonctionnelle car nous n'avions pas de contrôle de courant moteur, et donc de l'accélération, avec notre afficheur. Donc, si nous avions fait cette fonction, elle n'aurait servi qu'à un affichage inutile. Mais, comme il nous restait quatre semaines avant la fin de la remise du rapport, nous avons décidé de remplacer cette fonction (avec l'accord de M. Lequeu) par une autre: le chronomètre.

Le but étant de pouvoir, lors de l'appui sur un bouton, déclencher un comptage du temps et de pouvoir l'arrêter avec un appui sur ce même bouton. Lorsque le chronomètre est arrêté et que l'utilisateur appuie de nouveau sur le bouton, le comptage recommence de 0 pour un nouveau cycle.

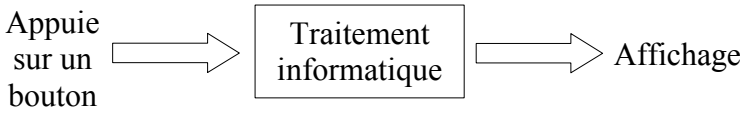


Figure 6: Schéma fonctionnel du chronomètre[1]

Afin de réaliser cela, nous avons utilisé une nouvelle fois un timer, mais sur 8 bits cette fois-ci. Le choix de la fréquence a été fait cette fois par soucis de simplicité, car nous voulions compter avec une précision en millisecondes. Et notre choix s'est une nouvelle fois tourné vers un comptage en 62,5kHz afin d'avoir une incrémentation toutes les 16µs. Comme nous avons un compteur de 8 bits, alors il y a 255 incrémentations avant de saturer, puis on règle notre programme afin qu'il émette une interruption lors de cette saturation. Ainsi, on a une interruption toutes les 255*16µs, soit toutes les 4ms. On considère alors que l'on compte une seconde lorsque l'on a plus de 244 interruptions.

Une simple incrémentation de variable permet ensuite d'avoir les minutes (lorsque la variable secondes dépasse 59) et une variable heure (lorsque la variable minutes dépasse 59 également).

Une deuxième interruption doit être créée lorsque l'utilisateur appuie sur le bouton afin d'arrêter de compter ou de remettre à 0 et de lancer le comptage, suivant le mode précédent.

Lors de la programmation de cet outil, on a pu remarquer que l'on avait un phénomène de « rebond ». En effet, lors de l'appui sur notre bouton, celui-ci agissait comme si l'on appuyait plusieurs fois en très peu de temps, et donc le fonctionnement était très aléatoire. Afin de résoudre ce soucis, nous avons décidé de bloquer l'interruption « appui sur le bouton » pendant une durée de 400ms (ce qui correspond à 100 fois la période de l'interruption temps vue plus haut).

Pour commencer, on déclare les variables globales utilisées par nos interruptions:

Code

```
        // Comptage de ms, mais en réalité on compte le nombre de 4ms
int chrono_ms=0;
        // Comptage des secondes
int chrono_s=0;
int chrono_m=0;
        // Choix du mode, si on compte ou si on est arrêté, içi on est arrêté
int mode_ch = 0;
        // Variable pour le comptage des 400ms à attendre
int int_att = 0;
```

Ensuite, on a la fonction interruption pour le temps:

Code

```
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
        // Si l'on est dans le mode 1, alors on incrémente le compteur
if(mode_ch==1)
    chrono_ms ++;
        // Si l'on vient d'appuyer sur le bouton alors int_att est à 0, et on l'incrémente
if(int_att<110)
        int_att ++;
        // Lorsque l'on dépasse 244 alors on a une seconde de passé
if(chrono_ms>=244)
{
        // On remet le compteur de ms à 0
chrono_ms = 0;
        // Et on incrémente celui des secondes
chrono_s ++;
}
}
```

Puis l'interruption pour l'appui sur un bouton:

Code

```
interrupt [EXT_INT0] void ext_int0_isr(void)
{
        // Si l'on a attendu suffisamment longtemps depuis le dernier déclenchement du bouton
if(int_att>100)
{
        // Si le mode précédent est celui de comptage
        if(mode_ch==1)
        {
        // On change le mode en arrêt du compteur
                mode_ch=0;
        // On initialise le compteur qui dit que l'on vient d'appuyer
                int_att=0;
        }
        else
        {
        // On change le mode en comptage
                mode_ch=1;
        // On initialise toutes les variables de temps
                chrono_ms=0;
                chrono_s=0;
                chrono_m=0;
        // On initialise le compteur de temps
                TCNT0=0x00;
        // On initialise le compteur qui dit que l'on vient d'appuyer
                int_att=0;
        }
}
}
```

On place ensuite dans la boucle « while(1) » la partie affichage à l'écran:

Code

```
// Partie chronomètre
        // Si le nombre de secondes est supérieur ou égal à 60
         if ( chrono_s>=60 )
         {
        // On remet les secondes à 0
                chrono_s  =  00;
        // On incrémente les minutes
                chrono_m  ++;
         }
        // On prépare l'affichage du chronomètre avec une division par 2.5 du nombre de ms afin
d'avoir le nombre de ms réél
         sprintf ( tampon,"Chrono:  %2i:%2i:%2i",chrono_m,chrono_s, ( int ) ( ( float ) chrono_ms/2.5 ) );
        // On place le pointeur en position 0,3 et on affichage la phrase précédente
         lcd_gotoxy ( 0,3 );
         lcd_puts ( tampon );
```

2.7. Intensité moteur et batterie

La carte fournie dispose de deux capteurs de courant et comme il nous restait deux semaines, nous avons décidé de faire une mesure de l'intensité du courant dans le moteur et dans la batterie.

Toutefois, il manquait sur la carte toutes les résistances qui permettent de convertir la mesure de courant. Sachant que le capteur délivre une tension variant entre -4 et +4V (image de l'intensité de -200 à +200A), il fallait calculer les résistances afin d'obtenir en sortie une tension de 0 à 5V. Pour cela, nous avons dû calculer les résistances nécessaires pour faire un diviseur de tension de +15V à +4V, puis mettre en place un montage sommateur afin de ramener la tension dans une plage de 0 à +8V avec un certain gain permettant de ramener cette tension entre 0 et +5V.

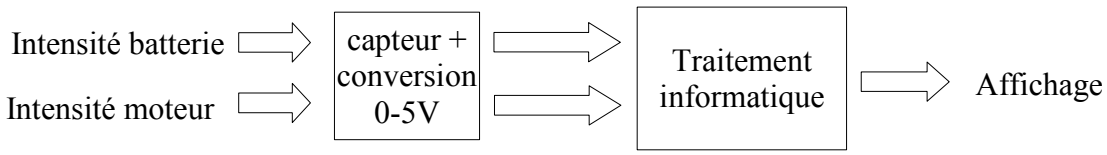


Figure 7: Schéma fonctionnel de l'intensité moteur et batterie[1]

Un schéma complet du montage de conversion en 0-5V est fourni en annexe.

Dans le schéma tel que celui proposé dans le datasheet de la carte utilisée, la résistance variable R36 a été remplacé par deux résistances ($R36_1$ et $R36_2$).

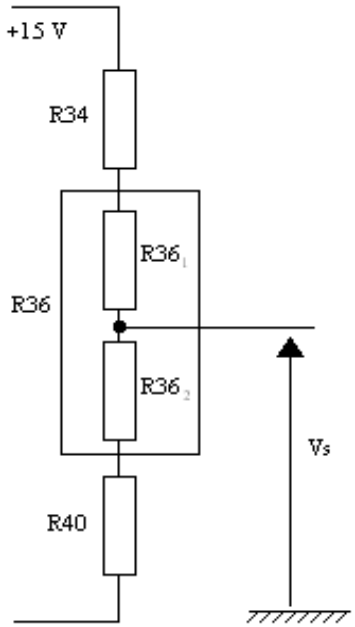


Illustration 5: Diviseur de tension pour la conversion 0-5V[1]

Vs = 15 * ($R36_2$ + R40) / ($R36_1$ +R34 + $R36_2$ + R40)
$R36_1$ + $R36_2$ = 10kΩ

- On veut Vs=3,5V si $R36_1$ = 0kΩ et $R36_2$ = 10kΩ

d'où 3,5V = 15*(10k + R40) / (R34 + 10k + R40)
d'où R40 = 3,5/11,5 * R1 + 10k

- On veut Vs=4,5V si $R36_1$ = 10kΩ et $R36_2$ = 0kΩ

d'où 3,5V = 15*(R40) / (R34 + 10k + R40)
d'où R40 = 4,5/10,5 * R1 + 45000/10,5

Donc **R34** = 40000 * 161 / (7*20) = **46kΩ**
et d'où **R40 = 24kΩ**

Pour le deuxième point de mesure, nous n'avions qu'un potentiomètre variable de 50kΩ. On a donc dû modifier les valeurs de R24 et R30 pour avoir: **R24 = 230kΩ** et **R30 = 120kΩ**.

Cependant, nous n'avons pas pu obtenir l'effet souhaité car un diviseur de courant n'est valable seulement si aucun courant n'est débité, or pour nous ce n'est pas le cas. Ainsi, nous avons relevé que V+= (VLEM +4)/3. Et donc, si VLEM = 4V alors V+ = 2,6667V (au lieu de 8V) et si VLEM = -4V alors V+ = 0V. Afin d'avoir notre sortie en 0-5V, nous devons donc avoir un gain de 2,6667 / 5.

Sans détailler les calculs ici car ce n'était pas le sujet de notre projet, on trouve **R33=100kΩ** et **R39=118kΩ** avec **R35 = 50kΩ** variable.

L'entrée mesure de notre microcontrôleur varie de 0 à 5V et donne une sortie entre 0 et 1023. Nous avons 200A pour 5V et -200A pour 0V, donc 200A pour 1023 et -200 pour 0. Un calcul simple permet d'obtenir que la sortie = ((entrée (0-5V)/1023)*200)-200 afin d'avoir la sortie en ampère en fonction de l'entrée en volts (entre 0 et 5V).

Ainsi, on a pu produire le code suivant:

Code

```
//  Partie intensité batterie

        // On lit la tension d'entrée
          i=read_adc(0);
        // On prépare l'affichage des ampères avec notre fonction précédente
          sprintf(tampon,"%3i",(int)((float)i*400./1023.)-200);
        // Et on affiche le tout
          lcd_gotoxy(3,2);
          lcd_puts(tampon);


// Partie intensité moteur

        // Même programme pour l'intensité moteur
          i=read_adc(1);
          sprintf(tampon,"%3i",(int)((float)i*400./1023.)-200);
          lcd_gotoxy(12,2);
      lcd_puts(tampon);
```

3. Planning

Afin d'estimer la durée du projet et d'orienter notre travail convenablement, nous avons mis en place un planning des objectifs à réaliser dès le début du projet. Ensuite, en fin de projet, nous avons dressé le planning réel afin de réaliser une comparaison.

Illustration 6: Planning prévisionnel et réel[1]

On peut remarquer une différence relativement importante entre ces deux plannings. Cette différence s'explique par le fait que le logiciel est doté d'une interface qui permet de tout configurer très facilement et nous avons pu ainsi commencer très vite l'étude et la programmation des différentes fonctions. De plus, nous avons rajouté des fonctions au programme demandé (comme la fonction chronomètre, intensité batterie et moteur, affichage de l'accélération), ce qui a également modifié le planning prévisionnel.

4. Nomenclature

Notre projet étant essentiellement constitué de programmation, nous n'avons eu aucune commande à effectuer. Par conséquent, le coût a été réduit à 0 euro. Cependant, nous avons tout de même récupéré le projet de nos camarades qui ont, eux, dépensé de l'argent afin de réaliser la carte que nous avons utilisé. Par conséquent, même si notre partie du projet n'a rien coûté, le coût total du projet est quand même présent.

De plus, la carte n'avait pas été finalisée, puisqu'il manquait quelques résistances sur la carte fournie, que nous avons donc dû rajouter. Ce qui a pour conséquence d'augmenter le prix de notre projet de quelques centimes d'euros.

5. Tests et validation

On se propose lors de cette partie « tests et validation » de tester une partie du montage afin de savoir si l'on est en conformité avec ce que l'on voudrait obtenir.

Comme notre projet repose sur de la programmation, nous n'avons pas beaucoup de tests possibles. Cependant, on a tout de même réalisé une partie de calculs et de soudure afin de réaliser la conversion -4/+4V en 0-5V. On va donc montrer ici, le test de ce montage.

Objectif du test :

Il s'agit de tester la conversion en tension qui, après avoir reçue une tension image de la batterie entre -4 et +4V, nous donne une tension image de la précédente entre 0 et 5V.

Préparation du test :

Le connecteur JP10 peut être simulé par une alimentation -4/+4V branchée sur la borne 3 et 4 de ce connecteur.

Il est nécessaire de brancher l'alimentation +15V/0V/-15V afin d'alimenter correctement le circuit.

Schéma de mesures :

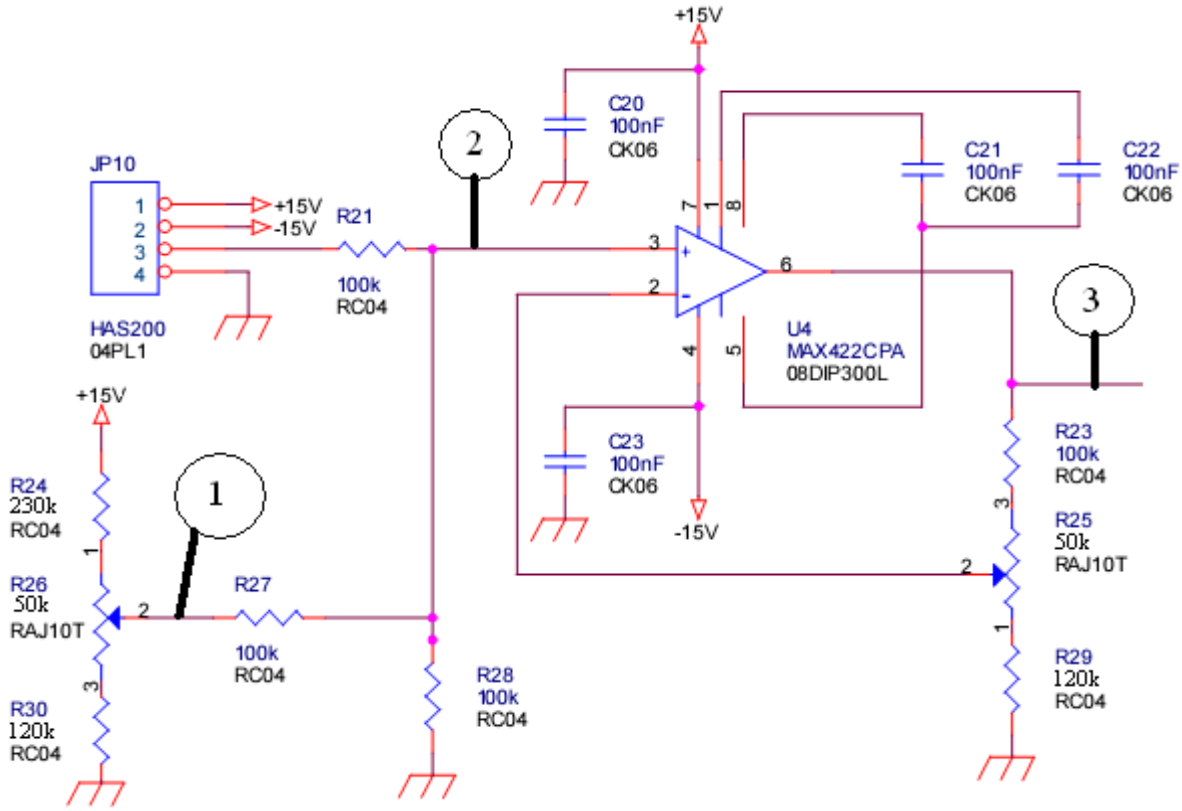


Illustration 7: Schéma de tests[1]

Procédure de test :

Appareil de mesures nécessaire : un multimètre.

Alimenter le circuit en +15/0/-15V et brancher l'alimentation -4/+4V sur le connecteur JP10 aux bornes 3 et 4. Se placer à -4V.

Régler le potentiomètre R26 de manière à avoir +4V au point test 1.

Maintenant régler le potentiomètre R25 de manière à avoir 0V au point test 3.

Procédure de test :

Placer votre alimentation à +4V. Vous devriez lire au point test 1: +4V et au point test 3: +5V.

Rapport de test:

Résultat du test:

Accepté: ☒

Refusé: ☐

Compte-rendu:

La tension lue au point test 3 est bien +5V, on a donc bien réalisé notre convertisseur -4/+4V en 0/+5V pour être exploité convenablement par notre microcontrôleur.

6. Fiche de suivi de projet

Voici la fiche de suivi de projet qui nous accompagne depuis la première séance d'Études et Réalisation et sur laquelle sont notés les événements relatifs au déroulement du projet :

Intitulé du projet	Étudiants
Afficheur pour e-Kart	Stan KOCKEN Grégoire DOLBEAU

Date	Commentaires	Commandes
23/01/08	Prise en main du sujet et premier test du logiciel AVR	
01/02/08	Étude et programmation de la visualisation de l'accélération	
06/02/08	Étude et programmation de la tension batterie	
13/02/08	Fin de la programmation de la tension batterie et étude et programmation de la fonction température	
27/02/08	Étude de la visualisation de la vitesse	
05/03/08	Programmation de la visualisation de la vitesse	
12/03/08	Étude et programmation du chronomètre	
19/03/08	Étude et soudage des composants pour la fonction de visualisation des intensités moteur et batterie	
26/03/08	Tests du fonctionnement complet de l'afficheur	

Conclusion

Grâce à ce projet que nous avons réussi à mettre en œuvre, nous avons pu voir une mise en application directe de la programmation enseignée à l'I.U.T. d'une manière plus abstraite jusque là. Cela nous a également fait comprendre les enjeux d'une programmation dans un système temps réel, où le temps de calcul pour réaliser nos fonctions n'était pas un paramètre négligeable et qu'il fallait optimiser notre code si l'on voulait le rendre fonctionnel dans un environnement concret. Le fait que notre projet va être utile pour le club de Karting de l'I.U.T. a été pour nous, une motivation supplémentaire de faire un travail soigné et clair afin que nos successeurs puissent, sans grandes difficultés, réutiliser notre travail afin d'en améliorer l'utilisation.

Résumé

Savoir construire et piloter un karting électrique c'est bien, mais pouvoir, en même temps qu'on le conduit, visualiser les principaux paramètres du karting, ça apporte un plus !

Ainsi, nous devions réaliser un afficheur pour le karting, qui soit capable d'afficher la température moteur, la charge de la batterie, la vitesse en km/h ainsi qu'une fonction BOOST.

Notre travail fut plus rapide par rapport à ce que nous espérions, ce qui nous a permis d'ajouter de nombreux éléments à notre afficheur. On peut par exemple citer un chronomètre, un ampèremètre pour le moteur et la batterie ainsi qu'un affichage de l'accélération. N'ayant pas de contrôle de courant sur notre carte, nous avons décidé, en accord avec le professeur, de retirer la partie « BOOST » qui ne serait que factice et donc non-opérationnelle.

Une partie inattendue de notre projet a été celle de devoir faire un peu d'électronique afin de calculer les résistances nécessaires pour réaliser un diviseur de tension et un sommateur avec gain réglable par potentiomètre. Ceci s'explique par le fait qu'il n'était pas prévu au départ, que nous réalisions la partie « mesure de courants » pour le moteur et la batterie mais que celle-ci a tout de même été effectuée grâce au temps restant.

On a donc réussi à mener à bien notre projet initialement demandé (mise à part la fonction BOOST) et même, à rajouter des fonctions à notre afficheur afin de le rendre encore plus utile.

[240 Mots]

Index des illustrations

Index des figures

Bibliographie

0: Thierry Lequeu, , 2008, http://www.thierry-lequeu.fr/data/DATA363.HTM

1: Stan Kocken, Document ou image personnel, 2008

Annexe

Programme complet:

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.7f Evaluation
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com

Project :
Version :
Date    : 23/01/2008
Author  : Freeware, for evaluation and non-commercial use only
Company :
Comments:


Chip type           : ATmega8535
Program type        : Application
Clock frequency     : 16,000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 128
*****************************************************/
#include <mega8535.h>
// I2C Bus functions
#asm
    .equ __i2c_port=0x18 ;PORTB
    .equ __sda_bit=0
    .equ __scl_bit=1
#endasm
#include <i2c.h>

#include<delay.h>
#include<stdio.h>

// LM75 Temperature Sensor functions
#include <lm75.h>


// Alphanumeric LCD Module functions
#asm
    .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>

#define ADC_VREF_TYPE 0x00

int vit;
int chrono_ms=0;
int chrono_s=0;
int chrono_m=0;
int mode_ch = 0;
int int_att = 0;


interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here
if (mode_ch==1)
    chrono_ms ++;

if (int_att<110)
            int_att ++;
if (chrono_ms>=244)
{
```

```
chrono_ms = 0;
chrono_s ++;
}
}


// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here
if (int_att>100)
{
        if (mode_ch==1)
        {
                mode_ch=0;
                int_att=0;
        }
        else
        {
                mode_ch=1;
                chrono_ms=0;
                chrono_s=0;
                chrono_m=0;
                TCNT0=0x00;
                int_att=0;
        }
}
}


// External Interrupt 1 service routine
interrupt [EXT_INT1] void ext_int1_isr(void)
{
// Place your code here
if((TCNT1>0)&&(TCNT1<5155))
{
vit = TCNT1;
}
TCNT1 = 0x00;

}

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}

#include<stdio.h>
#include<delay.h>


// Declare your global variables here

void main(void)
{
// Declare your local variables here
unsigned int i;
int temp;
float conversion;
unsigned char tampon[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=0xF0;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 62,500 kHz
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x04;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 62,500 kHz
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x04;
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: On
```

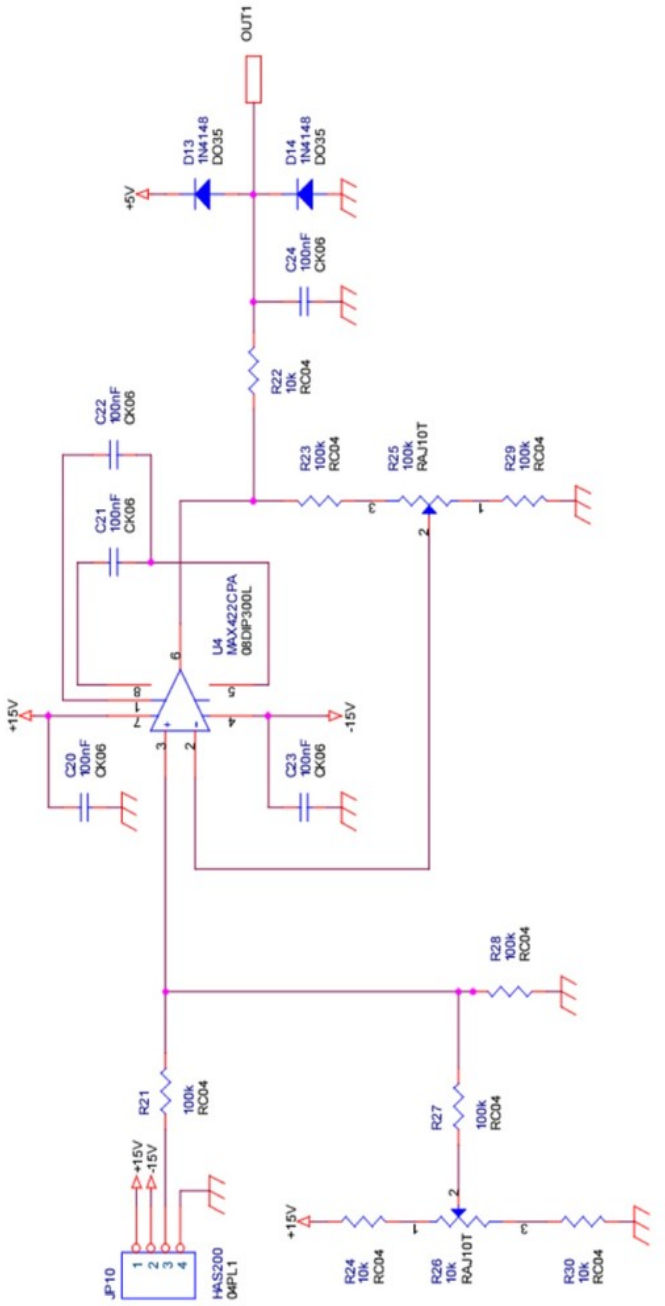
```
// INT0 Mode: Falling Edge
// INT1: On
// INT1 Mode: Falling Edge
// INT2: Off
GICR|=0xC0;
MCUCR=0x0A;
MCUCSR=0x00;
GIFR=0xC0;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;

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

// I2C Bus initialization
i2c_init();

// LM75 Temperature Sensor initialization
// thyst: 35¤C
// tos: 40¤C
// O.S. polarity: 0
lm75_init(7,35,40,0);

// ADC initialization
// ADC Clock frequency: 1000,000 kHz
// ADC Voltage Reference: AREF pin
// ADC High Speed Mode: Off
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x84;
SFIOR&=0xEF;

// Global enable interrupts
#asm("sei")


// LCD module initialization
lcd_init(16);

lcd_gotoxy(0,0);
lcd_putsf("      VisuKart      ");
lcd_gotoxy(0,1);
lcd_putsf("Stan  K      Greg  D");
lcd_gotoxy(0,2);
lcd_putsf("  le  19/03/2008    ");

lcd_gotoxy(0,3);
lcd_putsf("           .          ");
delay_ms(400);

lcd_gotoxy(0,3);
lcd_putsf("           ..         ");
delay_ms(400);

lcd_gotoxy(0,3);
lcd_putsf("          ...         ");
delay_ms(400);

lcd_gotoxy(0,3);
lcd_putsf("         .....        ");
delay_ms(400);
/*
lcd_gotoxy(0,0);
lcd_putsf("    VisuKart    ");*/
lcd_gotoxy(0,0);
lcd_putsf("      kmh  Acc:    %");
lcd_gotoxy(0,1);
```

```
lcd_putsf("B:      V  T:      \xdfC");
lcd_gotoxy(0,2);
lcd_putsf("Ib:     A    Im:     A");
lcd_gotoxy(0,3);
lcd_putsf("                    ");

while (1)
      {
          // Partie batterie
          i=read_adc(7); // Tension batt
          conversion = (float)((float)i*0.0666915732-0.6358820244)*10;
          sprintf(tampon,"%2i.%i",(int)(conversion/10),(int)(conversion-(int)(conversion/10)*10));
          lcd_gotoxy(2,1);
          lcd_puts(tampon); //on affiche la valeur de i.


          // Partie accelerateur
          i=read_adc(2);
          sprintf(tampon,"%3d",(int)(i/6));
          lcd_gotoxy(12,0);
          lcd_puts(tampon);


          // Partie temp□rature
          temp=lm75_temperature_10(0);
          sprintf(tampon,"%2i.%1u",temp/10,temp%10);
          lcd_gotoxy(10,1);
          lcd_puts(tampon);


          // Partie vitesse
          sprintf(tampon,"%3i",(int)((float)vit*(float)(-0.005333333)+(float)110));
          lcd_gotoxy(0,0);
          lcd_puts(tampon);


         // Partie chrono
         if(chrono_s>=60)
         {
                  chrono_s = 00;
                  chrono_m ++;
         }
         sprintf(tampon,"Chrono:  %2i:%2i:%2i",chrono_m,chrono_s,(int)((float)chrono_ms/2.5));
         lcd_gotoxy(0,3);
         lcd_puts(tampon);


          // Partie intensite batterie
          i=read_adc(0); // Tension image batt
          sprintf(tampon,"%3i",(int)((float)i*400./1023.)-200);        // 1023 correspond □ 5V (200A) et 0 □ 0V (-200A)
          lcd_gotoxy(3,2);
          lcd_puts(tampon); //on affiche la valeur de i.


          // Partie intensite moteur
          i=read_adc(1); // Tension image mot
          sprintf(tampon,"%3i",(int)((float)i*400./1023.)-200);
          lcd_gotoxy(12,2);
          lcd_puts(tampon); //on affiche la valeur de i.


         delay_ms(100);
      };
}
```

Convertisseur -4/+4V en 0/+5V

pour la mesure de courant

Extrait de la documentation « Afficheur LCD version 2 »



(Les valeurs des résistances sont données à titre indicatif et ne correspondent pas à leurs véritable valeurs qui sont, elles, données dans le rapport)

Datasheet complète à l'adresse:

http://www.thierry-lequeu.fr/data/AFF-LCD2.pdf