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

Étude & Réalisation : Régulation du courant moteur du kart

Julien Giovannangeli
Michaël Lanoë
Groupe TP : S2
Promotion 2005-2007

Enseignants : Thierry LEQUEU
Sophie LAURENCEAU

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

Étude & Réalisation : Régulation du courant moteur du kart

Julien Giovannangeli
Michaël Lanoë
Groupe TP : S2
Promotion 2005-2007

Enseignants : Thierry LEQUEU
Sophie LAURENCEAU

Sommaire

INTRODUCTION

Dans le cadre des travaux d'étude et réalisation du semestre quatre, les élèves ont en main un projet technique qui doit relever de l'informatique, qu'ils doivent réaliser de la théorie à la pratique.

Nous avons choisi de réaliser la régulation du courant moteur du kart GEII. Elle devra en fonction de la pression sur la pédale d'accélération réguler un courant plus ou moins fort dans le moteur.

Pour commencer, nous présenterons donc notre projet plus en détail, et son cahier des charges.

Ensuite on expliquera toute la partie électronique et automatique qui traite la pédale d'accélération, le capteur de courant et l'identification du moteur.

Pour finir nous allons expliquer le programme informatique qui nous permettra d'effectuer la régulation du courant.

1/ Présentation du projet

Cette partie vise à présenter plus en détail le projet et d'en exposer son cahier des charges.

1.1/ But du projet

Pour le kart de l'IUT on a besoin que le courant moteur soit proportionnel à la pression exercée sur la pédale d'accélération et qu'il ne soit pas trop grand.

Pour cela on à choisi de faire une régulation de celui-ci à l'aide du micro contrôleur ATmega8535, on va donc mettre en oeuvre le logiciel AVRstudio pour programmer celui-ci.

Pour que la régulation soit assurée nous allons aussi utilisé un hacheur que l'on commandera grâce à une fonction du microcontrôleur (PWM).

Avant de pouvoir programmer on a du établir le cahier des charges qui suit.

1.2/ Cahier des charges

Dans ce cahier des charges nous avons établi les étapes et les limites à tenir durant ce projet.

Tout d'abord nous expliquerons les différents systèmes présent dans notre projet puis nous définirons les limites de notre programmation et les matériels utilisés.

Capteur de courant:

Pour pouvoir réguler le courant moteur nous allons devoir mettre en oeuvre le capteur de courant présent sur le Kart. Celui-ci nous délivrera une tension qui sera image du courant, elle sera comprise entre -4V et +4V nous devons donc adapter cette tension à celle du micro contrôleur qui est de 0/+5V.

Pour résoudre ce problème nous avons décidé d'ajouter un montage qui sera placé dans la boucle de retour de notre régulation de courant, ce montage sera expliqué dans la suite de ce rapport.

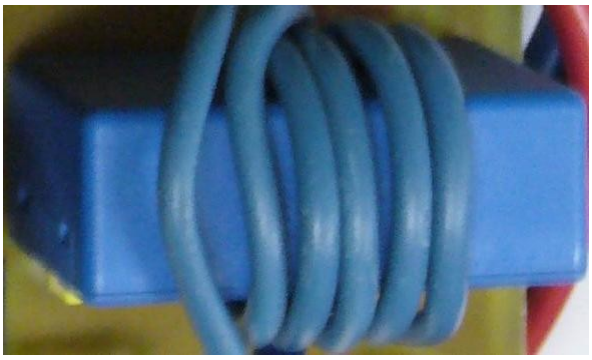


Illustration 1:Capteur de courant

Micro controleur:

Le choix du micro contrôleur utilisé est le microcontrôleur ATmegaS8535 qui était présent dans le magasin, de plus des cartes de test avaient déjà été mises en oeuvre. Les pages essentielles à notre projet du datasheet seront placées en annexe.

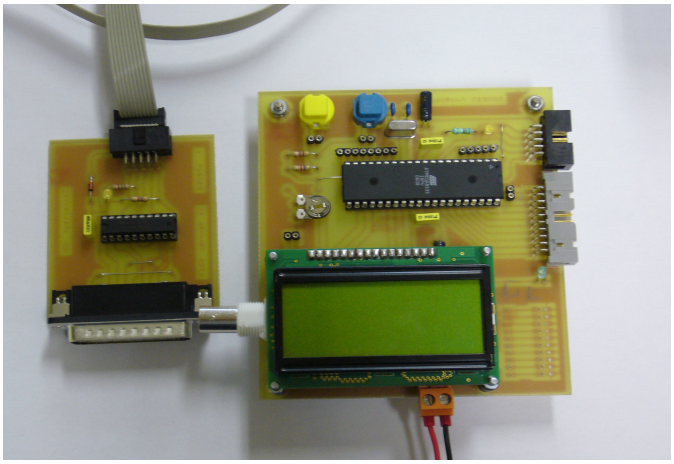


Illustration 2: Carte micro-contrôleur fournie par M Lequeu

La programmation de celui-ci se fera par le logiciel AVR studio qui est le plus approprié.

Durant notre programmation nous devrons utiliser différentes fonctions de ce micro contrôleur telles que:

- le CAN (Convertisseur Analogique Numérique) pour numériser les tension récupérées et pour pouvoir les traiter;
- le PWM (Pulse Width Modulation) nous permettra de générer à l'aide de la programmation un signal créneau dont le rapport cyclique sera variable selon l'erreur entre la tension d'entrée et de sortie, et ce rapport cyclique.
- Les routines d'interruptions qui feront office de calculateur.

Schéma du régulateur:

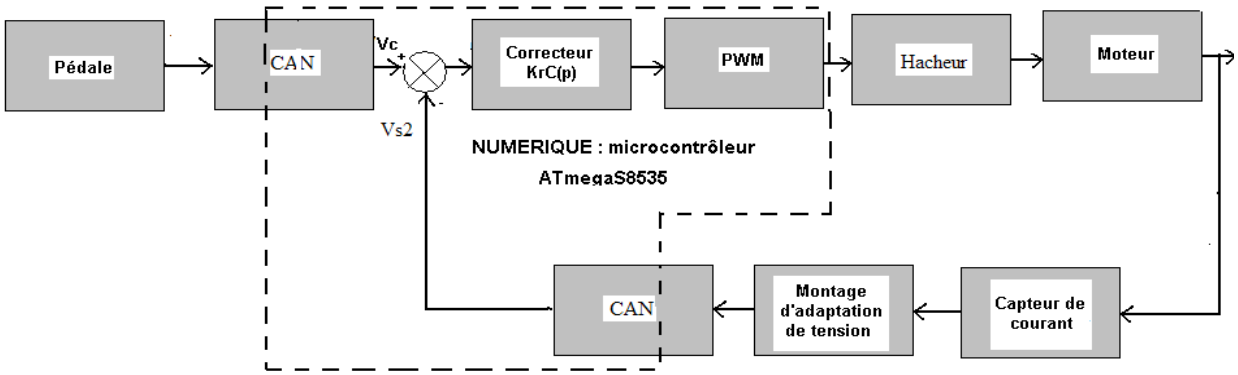


Illustration 3: Schéma du régulateur

La tension Vc est issue de la tension générée par pédale d'accélération du kart(par un pont diviseur de tension), suivie du CAN qui numérise celle-ci (Vc= tension de consigne), ensuite on a le comparateur qui va être programmé dans le microcontrôleur comme un calculateur, il calcul l'erreur entre Vc et Vs2[1] (ε=Vc-Vs2). Après la comparaison on récupère donc l'erreur

1 Vs2 dans la partie électronique correspond à Vm dans la programmation.

qui est corrigée par une action P, PI, PD ou encore PID. L'erreur étant corrigée on la traite puis on génère un PWM image de celle-ci, on a appelé cette tension Valpha. Valpha est envoyé directement dans le Hacheur (commande du transistor) et occupera donc le rôle de commande comme par exemple une commande MLI. Pour finir, dans la chaîne d'action on connecte le hacheur au moteur qui aura donc un courant moteur normalement régulé. Dans la chaîne de retour on retrouve le capteur de courant moteur qui nous délivre une tension comprise entre -4V et 4V et que l'on adapte pour obtenir une tension comprise entre 0V et 5 V que l'on pourra numériser à l'aide du CAN (Vs2) pour pouvoir faire la comparaison avec Vc.

Le montage d'adaptation de tension vous sera expliqué dans la suite.

Diagramme sagittale

Il définit les différentes fonctions à réaliser.

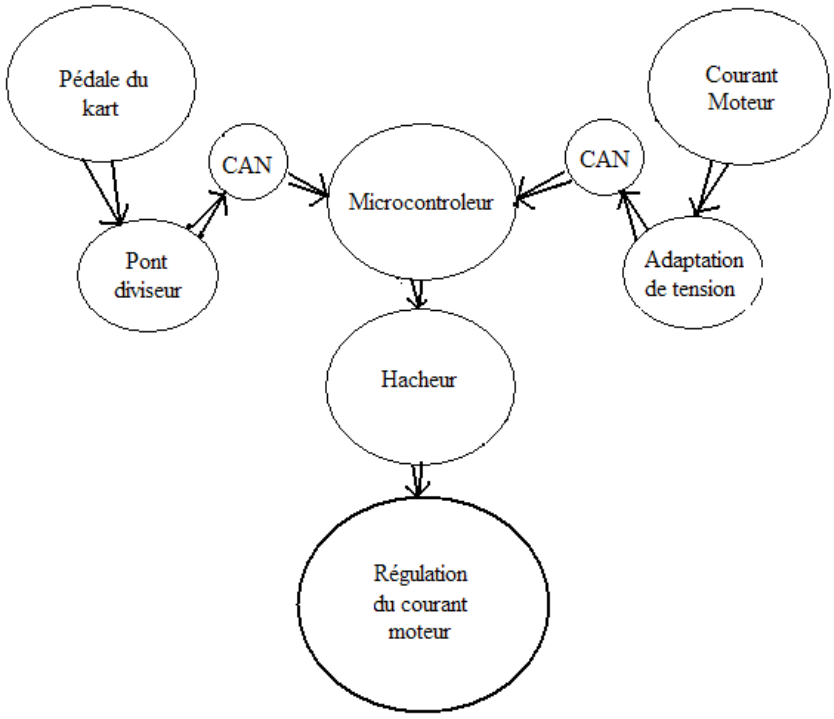


Illustration 4: Diagramme sagittale

2/ Partie électronique/automatique

Pour que nous puissions utiliser notre système, nous devons tout d'abord faire une étude électronique du projet. Nous allons vous expliquer le montage relatif à la pédale d'accélération du Kart, le montage relatif à l'adaptation de tension du capteur de courant et pour finir l'aspect automatique du projet qui est l'identification du système en boucle ouverte.

2.1/ Montage relatif à la pédale d'accélération du kart

La pédale du kart est reliée à l'aide d'un câble à une résistance variable variant de 0 à 5kΩ. Pour pouvoir récupérer une tension comprise entre 0 et 5V (plage de tension admise par le microcontrôleur) on doit utiliser un pont diviseur en conséquence.

Schéma

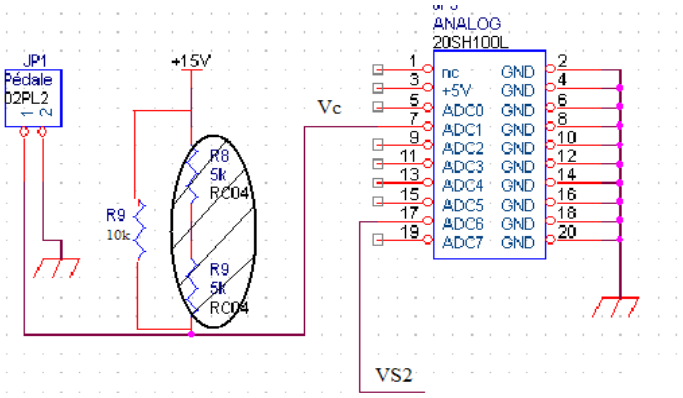


Illustration 5: Pont diviseur avec résistance variable

Sur ce schéma on retrouve au bornier appelé « Pédale » (JP1) où l'on connecte la résistance variable reliée à la pédale. Au début du projet, nous avions décidé d'utiliser deux résistances de 5kΩ qui par leurs valeurs nous auraient permis d'avoir entre 0 et 5V pour la tension Vc; mais après être allé au magasin on s'est rendu compte que ces résistances n'existaient pas. On a donc fini par opter pour une résistance de 10kΩ ce qui revient tout à fait au même.

Pour finir on voit que Vc est connecté à l'entrée analogique ADC1 du microcontrôleur et ceci car la programmation a été faite de cette façon. La tension Vs2 présente sur ce schéma vient d'un autre montage qui est celui de l'adaptation de tension qui est expliqué dans la suite.

2.2/ Montage d'adaptation de tension du capteur de courant

Pour adapter la tension que nous délivre le capteur de courant -4/+4V à la tension du microcontrôleur, 0/+5, on doit faire un montage que nous avons appelé le montage d'adaptation de tension. Tout d'abord on exposera le schéma puis nous expliquerons les différentes parties de celui-ci.

Schéma

Le schéma a été fait à l'aide du logiciel Orcad et plus exactement sous Capture:

Ce montage comprend deux montages montés en cascade qui vous sont expliqués dans la suite.

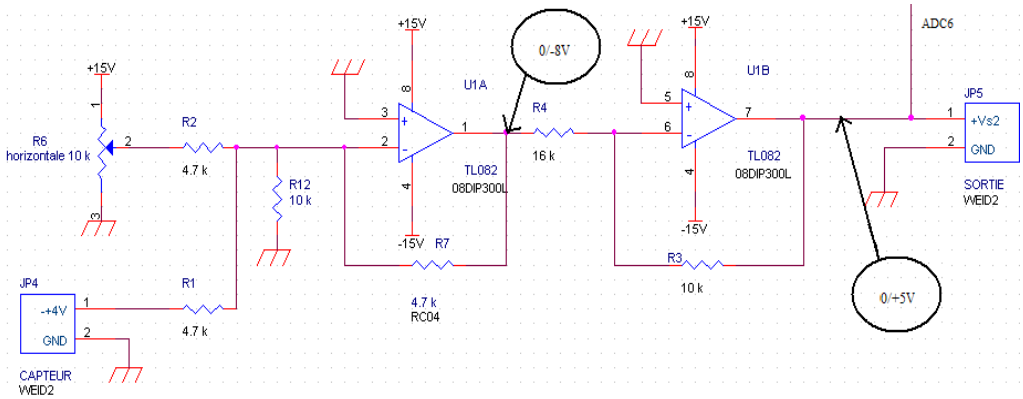


Illustration 6: Montage d'adaptation de tension

Principe de fonctionnement

L'entrée de ce montage est la tension comprise entre -4/+4V délivrée par le capteur de courant, et la sortie est la tension image de celle-ci comprise entre 0/+5V.

Pour cela on a utilisé deux montages misent en cascade:

-un montage qui ajoute un offset de +4V (inverseur donc -4V);

-un montage qui supprime le surplus de tension soit 3V (inverseur).

- Nous allons donc commencer par expliquer le montage d'ajout de l'offset:

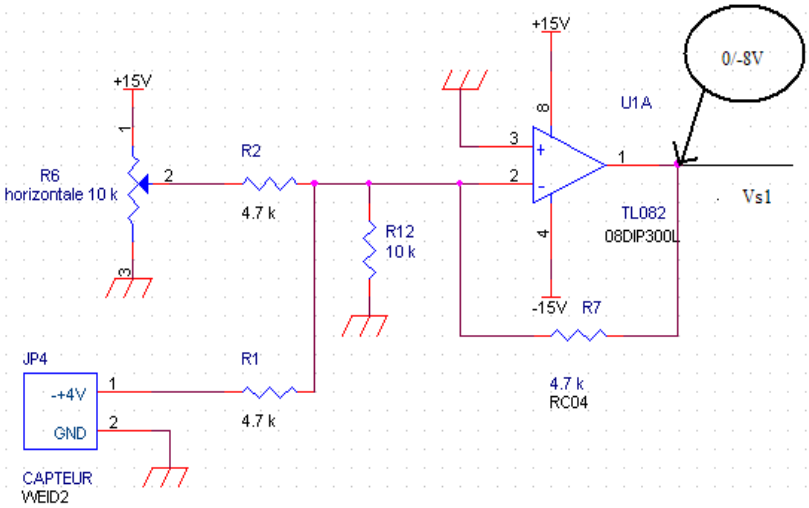


Illustration 7: Schéma d'ajout d'un offset de +4V

On peut voir sur ce schéma que l'on utilise un montage additionneur inverseur à base d'AOP. Il a donc pour entrée (sur l'entrée - de l'AOP) la tension image du capteur de courant comprise entre -4V et +4V et la tension que l'on souhaite ajouter (4V) qui se réglera à l'aide d'une résistance ajustable. L'inversion est dû à la boucle de contre réaction connectée à l'entrée « - », on retrouve donc en sortie de ce montage, soit à Vs1, une tension comprise entre -8V et 0V. On peut remarquer que même après une seconde inversion on aura un surplus de 3V, c'est pour cette raison que nous avons mit en cascade un second montage.

Sur ce montage, le choix des résistance ont été fait suite à des tests préliminaires qui sont expliqué dans la suite.

Suppression du surplus de 3V:

Pour supprimer ce surplus de tension on à décidé d'ajouter un montage inverseur ayant une amplification de -5/8:

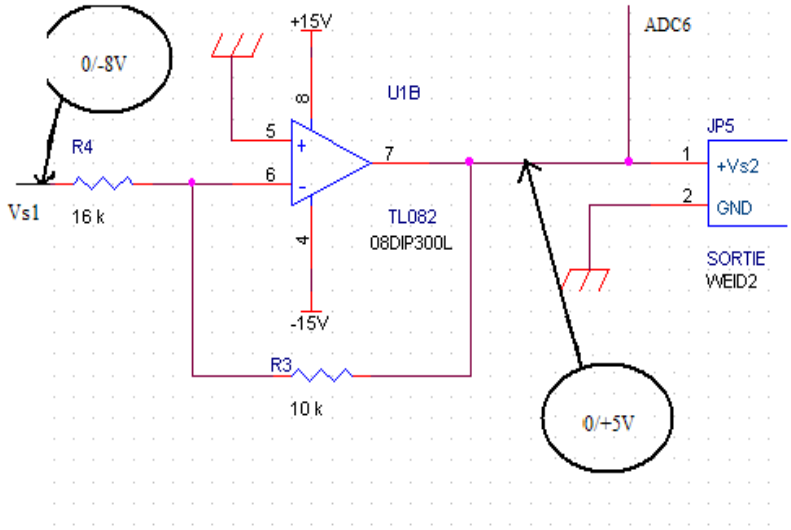


Illustration 8: Montage inverseur

Ce montage est un montage à base d'AOP, le choix des résistances ne fut pas très complexe car il suffit que le rapport de -R3/R4 soit égale à -5/8 (-10/16=-5/8).

Le bornier JP5 est similaire à un point test, car pendant les tests il a fallu que l'on relève le signal de Vs2. Cette sortie est aussi relié à l'entrée analogique ADC6 du microcontrôleur car c'est la tension Vm utilisée en programmation pour faire la comparaison avec Vc.

Pour conclure sur ce montage, il est simple d'utilisation et on verra par la suite les résultats des tests effectués.

2.3/ Tests sur l'adaptation de tension

Tests préliminaires:

Durant l'élaboration nous avons dut faire des tests pour faire un choix judicieux de résistances et pour vérifier le bon fonctionnement du montage. Pour cela nous avons utilisé une plaque test et fait les connections nécessaires à notre montage pour son bon fonctionnement.

Tout d'abord on a choisi de tester avec les résistances dont voici les valeurs R1=R2=R7=1 kΩ, on a pu voir que l'on avait pas la même tension en sortie qu'en entrée du montage mais la tension de sortie n'est pas vraiment de 0/+5V, elle est un peu inférieur on a donc essayé avec R1=R2=R7=4,7 kΩ, cette fois-ci la tension est plus précise on a donc choisi ces valeurs.

Tests finals:

Après avoir gravé la carte correspondant au schéma on a retesté le montage et en avons capturer les oscillogrammes suivant:

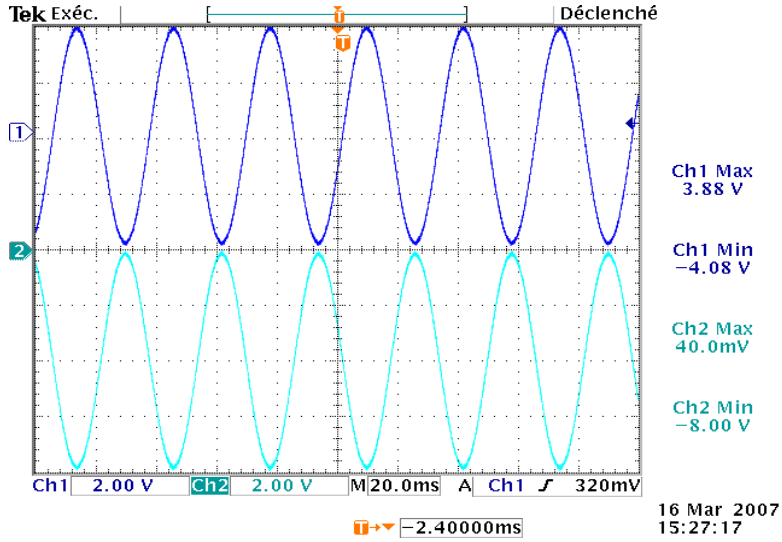


Illustration 9: Ajout d'offset de -4V

On peut voir que la tension sur la voie 1 (bleue marine), qui est notre tension issue du capteur est de -4V/+4V. La tension sur la voie 2 (bleue ciel), est donc celle d'entrée à laquelle on a ajouté un offset de -4V (« - » dû à l'inversion); on voit donc que le premier montage fonctionne et que l'on a en sortie de celui-ci du 0/-8V, ensuite on a vérifié, si il y avait en sortie du montage mit en cascade une tension de 0/+5V:

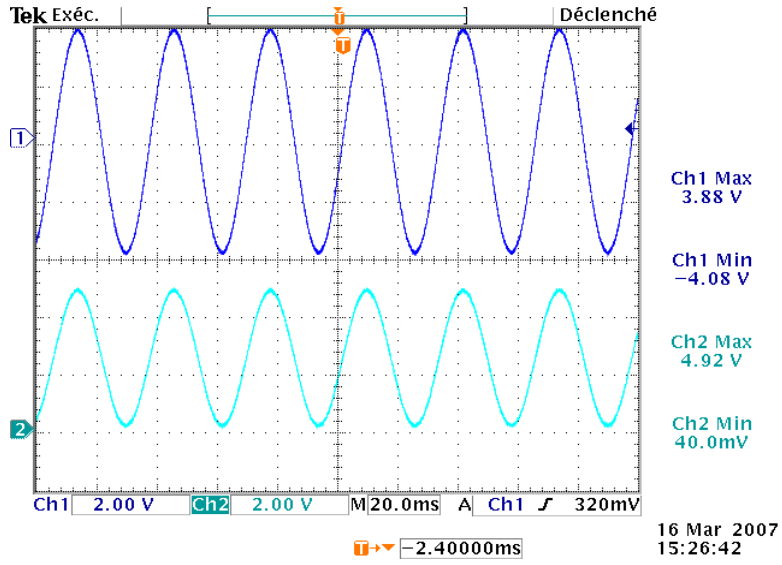


Illustration 10: Suppression du surplus de tension

La tension de la voie 1 est toujours celle d'entrée, alors que sur la voie 2 on a la tension de sortie du montage complet, qui est donc la tension d'entrée modifiée par l'ajout d'offset et le montage qui supprime le surplus de 3V, c'est à dire le montage inverseur d'amplification -5/8.

La carte électronique d'adaptation de l'alimentation fonctionne donc bien et est prêt à être intégré à notre régulation.

2.4/ Schéma globale

Voici le schéma globale qui à été confectionné sur Orcad Capture et qui rassemble le montage relatif à la pédale, le montage d'adaptation de tension ainsi que le bornier 2x10 broches que l'on utilise pour ce connecter au micro contrôleur, ce composant n'étant pas dans les bibliothèque d'Orcad nous avons utilisé celui fait par M.Lequeu.

Le composant utilisé pour l'AOP est un composant qui contient deux AOP intégrés (Annexe 2: T082)ce qui nous a permis de minimiser la place prise sur la carte

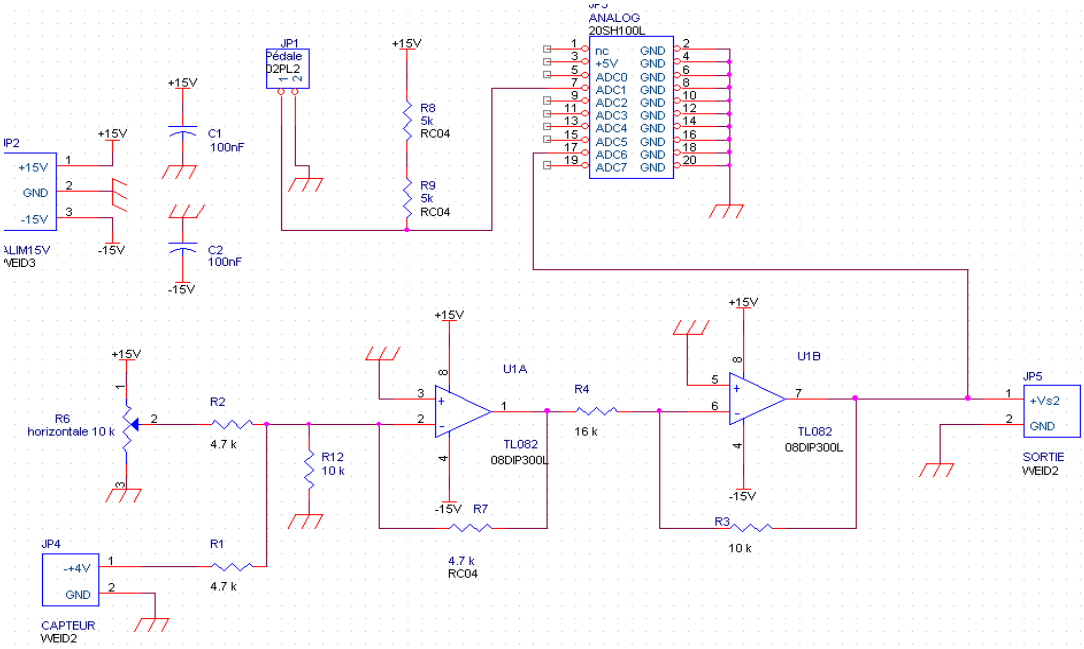


Illustration 11: Schéma globale de la carte électronique

Ce schéma nous a permis de router le typon à l'aide de Orcad Layout. C'est un outil qui prend en compte ce qui est fait sous Capture grâce à la Netlist. Voici le typon de notre carte:

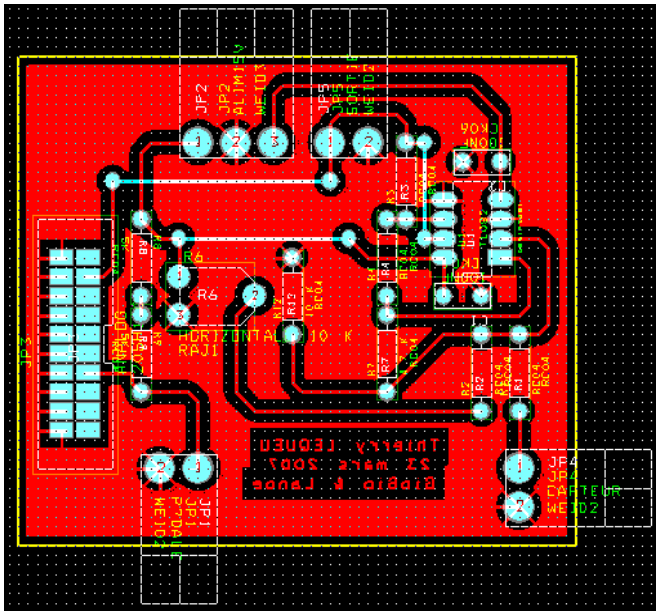


Illustration 12: Typon de notre carte

Après avoir imprimé le typon on a gravé la carte et soudé les composants à implanter:

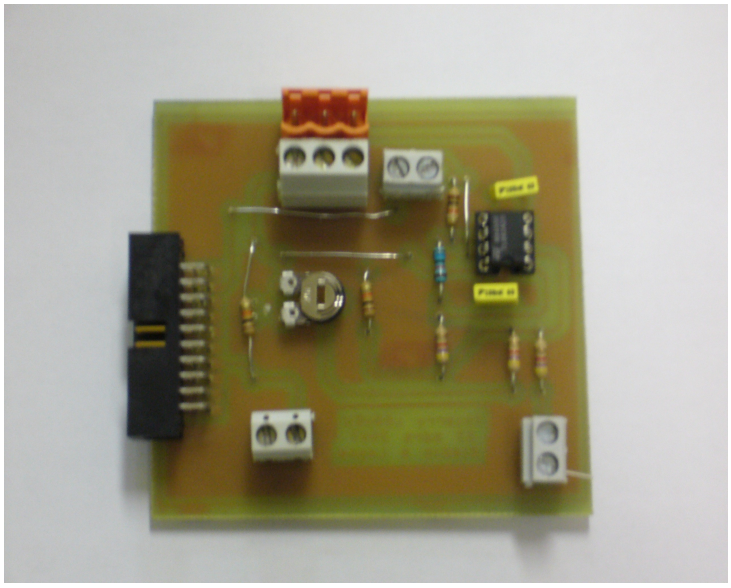


Illustration 13: Photographie de notre carte

2.5/ Automatique: Identification du moteur

La régulation du courant moteur est établi grâce à l'automatique. On doit tout d'abord relever la réponse indicielle du système à un échelon de tension.

Nous n'avons pas eu le temps d'étudier tout le système, mais on a relevé la réponse d'un bloc comprenant « correcteur (ici P) + PWM + Hacheur + Moteur + Capteur de courant » soit:

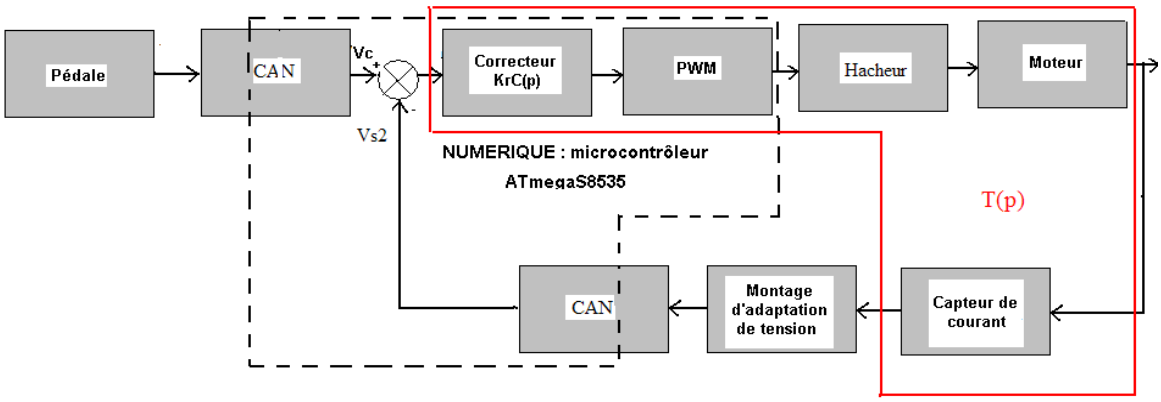


Illustration 14: synoptique montrant le sytème à identifier

On a relevé la réponse indicielle de la partie du système entourée en rouge , la voici:

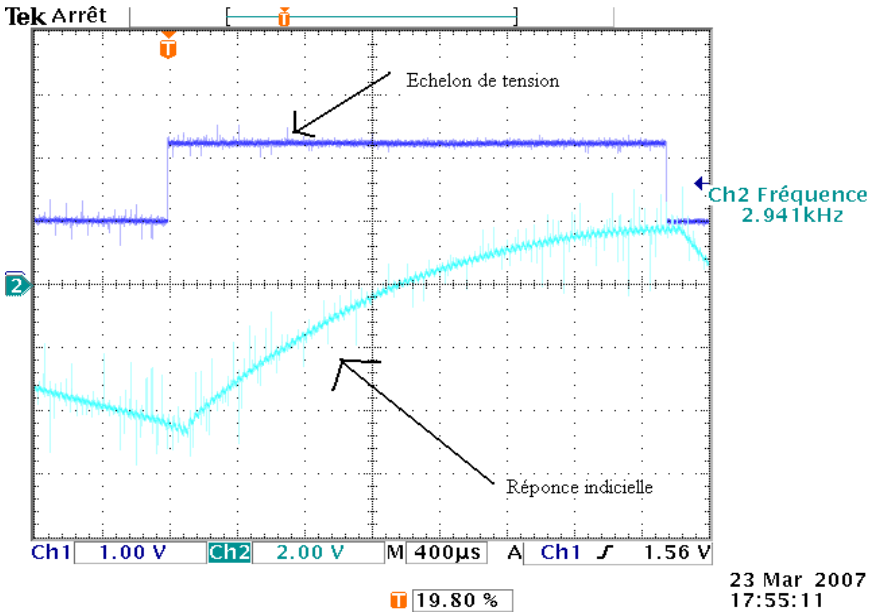


Illustration 15: Tension image du courant du moteur

Pour identifier un système il existe plusieurs méthodes appelées « Méthodes déterministes d'identification », il en existe beaucoup, telles que : Broïda, Zeigler & Nichols, ou encore Strejc qui sont utilisées en option automatique.

Dans notre cas on a décidé d'utiliser la méthode de Broïda, pour cela on à besoin de quelques paramètres à relever sur la réponse indicielle:

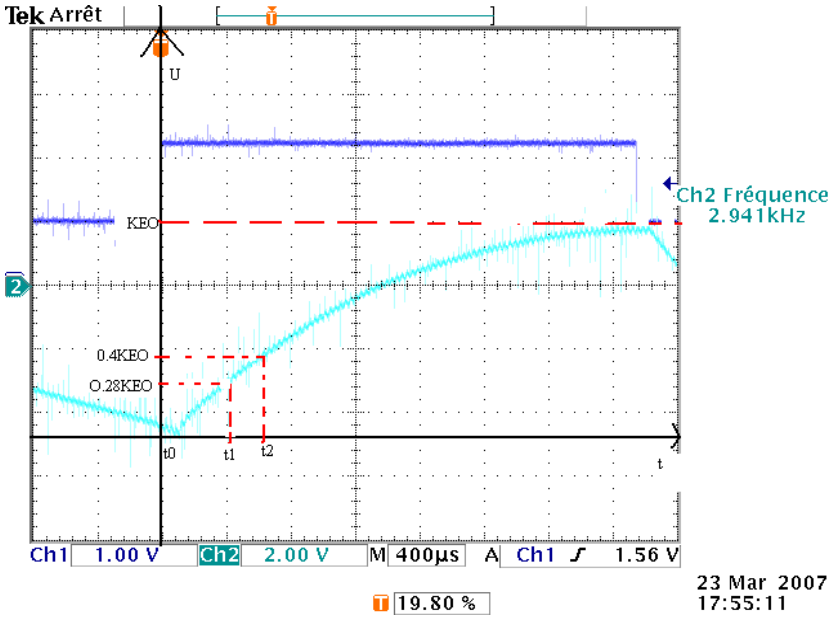


Illustration 16: méthodologie de Broïda

Tout d'abord il faut identifier la valeur finale de la réponse KE0:

→ ici KE0= 6,6V avec E0=1V

→ alors

K = 6,6 V

Ensuite on doit déterminer les temps t1 et t2:

- Calcul de t1:

Pour calculer t1 il faut calculer 0,28KE0 et faire l'ordonnée à l'origine pour trouver le temps t1 correspondant sachant qu'un carreau sur le relevé représente 400 µs:

→ 0,28KE0 = 1,848 V

→ alors

t1= 457 µs

- Calcul de t2:

De la même manière on trouve t2 avec 0,40KEO:

→ 0,40KE0 = 2,64 V

→ alors

t2= 628 µs

Maintenant que l'on a les paramètres indispensables à la résolution, on peut trouver les paramètre qui sont présent dans la fonction de transfert de la méthode de broïda, r (le retard) et ζ (tau la constante de temps du système):

$$T(p) = K \times e^{-rp} / (1 + \zeta p)$$

- Calcul de r:

→ r = 2,8(t1-t0) – 1,8 (t2-t0)

→ ici t0 = 0 s

→ donc

r= 150 µs

- Calcul de ζ:

→ ζ= 5,5(t2-t1)

→ donc

ζ= 940 µs

On a déterminé les paramètres, la fonction de transfert du système est :

$$T(p) = 6{,}6 \times e^{-0{,}00015 p} / (1 + 0{,}00094\, p)$$

Pour finir, on a vérifié si ce modèle était bon à l'aide du logiciel ASL4 développé par le LMP (Laboratoire de Micro-électronique de Puissance), en voici quelques points qui justifient que le modèle utilisé nous permet de trouver la même allure de courbe en théorie qu'en pratique:

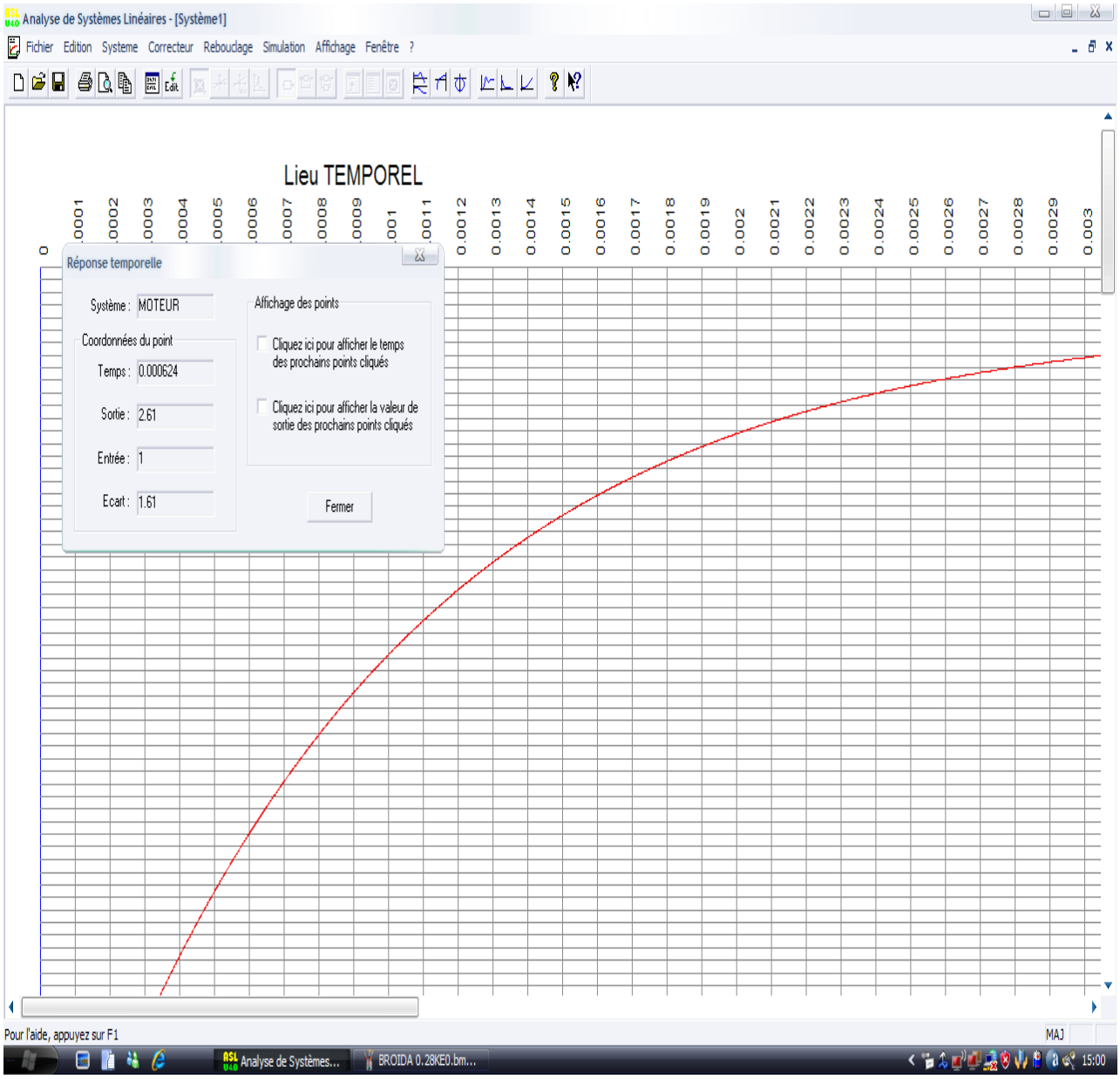


Illustration 17: Simulation du moteur sur ASL : confirmation du temps t2

Voici le point qui est concordant avec le temps t2

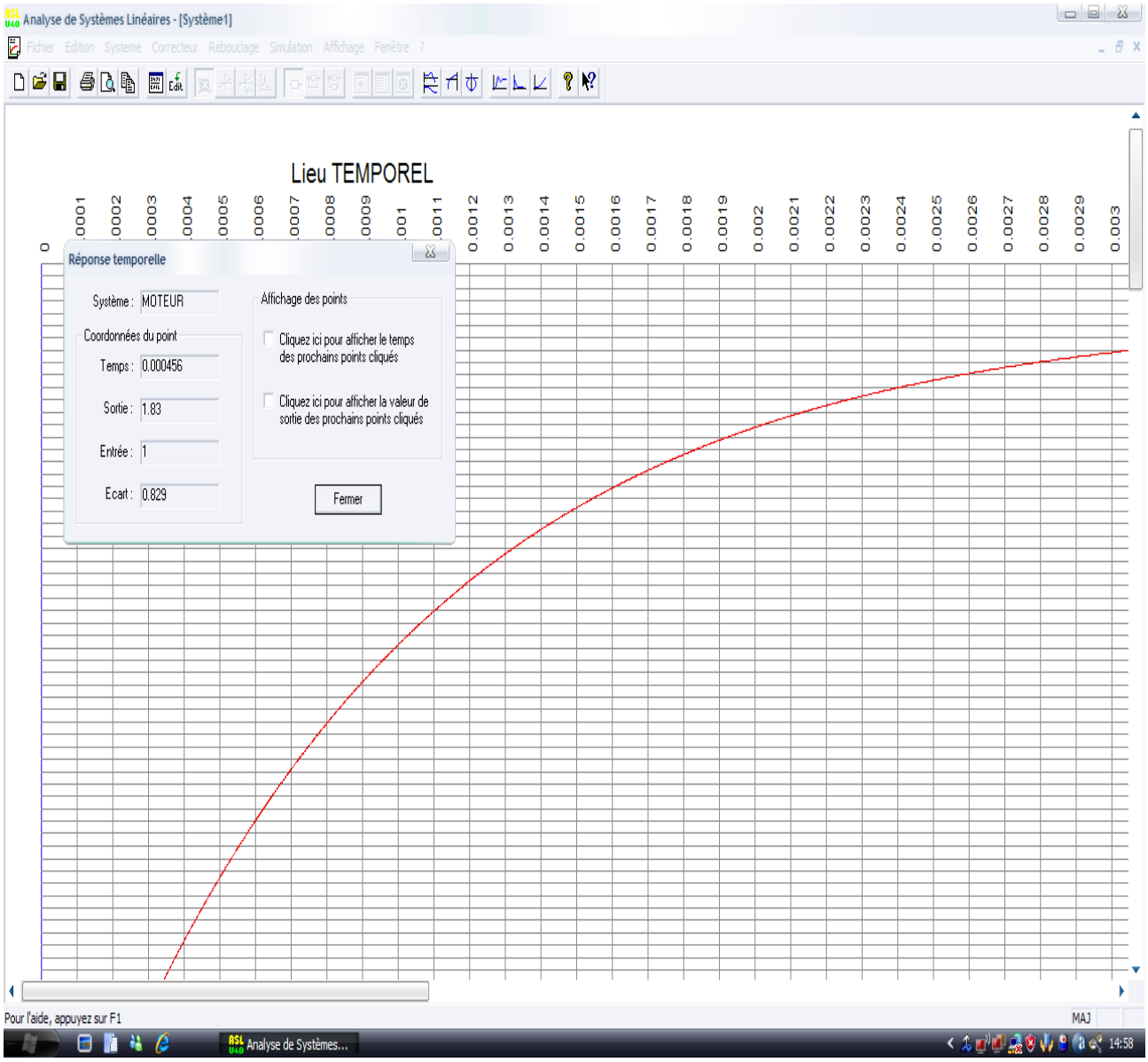


Illustration 18: Simulation du moteur sur ASL : confirmation du temps t1

Ici on retrouve un point identique à celui de t1.

Pour conclure sur la totalité de cette partie, il fut très intéressant de lier électronique et informatique, surtout d'utiliser des méthodes apprises en cours d'automatique.

Maintenant on va lier ceci à l'informatique qui fait l'objet de la partie suivante.

3/ Programmation

3.1/ Introduction : mise en place et objectif de la programmation

Le cadre de notre projet était orienté sur un point de vue plus informatique par rapport au semestre précédent. Nous devons réguler notre système grâce à un micro-contrôleur ATMEL® ATmégaS8535. Un logiciel d'interface est fournit avec, il permet de programmer et de compiler l'ensemble dans le composant électronique. Le logiciel s'appelle Code Vision AVR. Cependant, la programmation sur un tel environnement est nouveau pour nous. Nous n'avions eu aucune formations sur ce software. Nous avons dû alors nous familiariser pendant quelques séances avant que nous puissions commencer à réaliser quelques opérations nécessaires à la réalisation du programme final. Ainsi, nous avons découpé notre travail selon les fonctions nécessaires au programme final.

Le but de notre projet est de réguler le fonctionnement d'un moteur selon la pression de l'utilisateur sur la pédale de l'accélérateur.

Dans la synoptique de notre système, l'élément nommé hacheur est essentiel dans le bon fonctionnement du dispositif.

L'étude approfondie de ce composant électronique ne fait pas partie de notre étude.

Cependant, il permet une transition entre la partie commande (micro-contrôleur) et la partie opérative (moteur). Le moteur ne peut pas être alimenté par une sortie TTL.

Le but du micro-contrôleur est alors d'envoyer un signal en créneau positif avec une fréquence et un rapport cyclique notée α. Néanmoins, l'obtention de celui-ci n'est pas immédiat, il faut passer par différentes opérations.

Nous comparons les deux valeurs numériques respectives aux deux tensions en entrée du micro-contrôleur.

Soit Vc, la tension de consigne, elle provient de la pression sur la pédale de l'accélérateur.

Soit Vm, la tension de mesure, elle provient de la sortie du montage Adaptation de tension.

C'est la tension de sortie de la chaîne de retour.

Nous effectuons une différence entre ces deux tensions, nous notons $\boldsymbol{\varepsilon}$ le résultat. Il s'agit d'une tension écart entre Vc et Vm. Le but de la régulation étant le fait que ε soit égale à 0 en régime permanent.

Nous traiterons alors cette tension ε par différentes opérations. Ces opérations ont été étudiées dans la matière automatique en analogique.

Dans notre étude, nous devons effectuer une correction en numérique à l'aide du micro-contrôleur.

Les opérations seront des actions P, PI, PD ou PID mixte.

Une grandeur numérique respective à une tension notée Vα sera le résultat des différentes opérations sur la grandeur ε .

La notation sera respectée plus ou moins dans les lignes de codes du programme, les caractères spéciaux n'étant pas pris par le logiciel de programmation.

Afin de mieux comprendre les diverses opérations, nous exposons un schéma montrant les différentes opérations pouvant être réalisées par le micro-contrôleur.

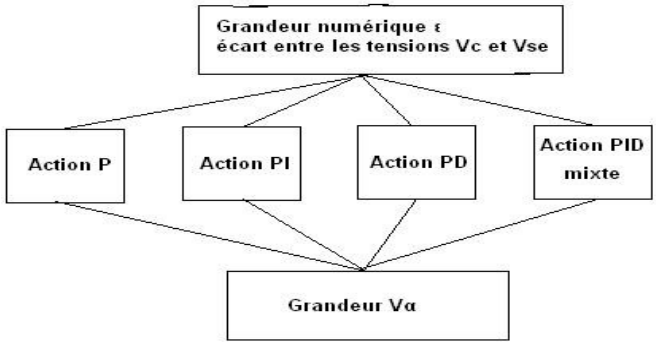


Illustration 19: Différentes actions de corrections

Nous utilisons le générateur de programme automatique ou CodeWizard AVR : cela nous permet de paramétrer différentes parties du micro-contrôleur. L'avantage de ce générateur est qu'il nous offre les lignes de programmes relatives au paramètres désirés.

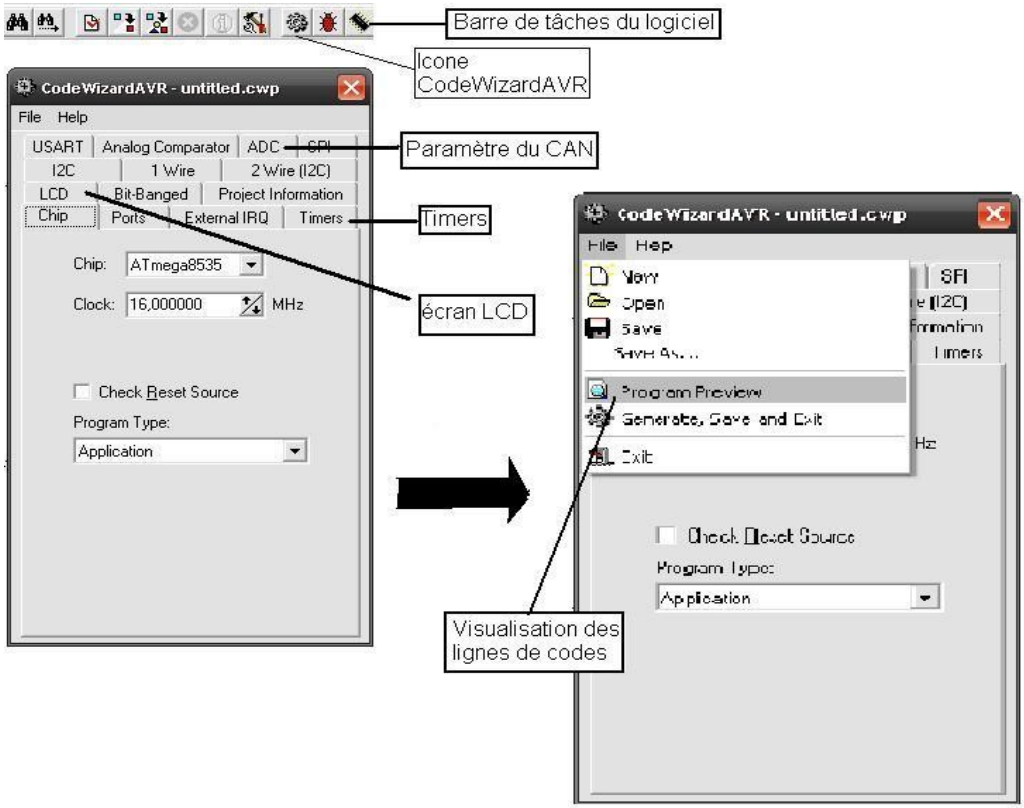


Illustration 20: Interface Code Wizard

3.2/ Mise en place de la conversion analogique-numérique.

Le synoptique nous montre le fait que deux tensions rentrent dans le micro-contrôleur. Il s'agit de grandeurs analogiques, il faut donc pouvoir grâce à un convertisseur intégré au micro-contrôleur les transformer en grandeur numérique. La mise en place d'un CAN sera donc réalisée dans le cadre de notre projet.

Le CAN est disposé sur 10bits. Les valeurs numériques résultantes d'une conversion seront comprises entre 0 et 1023. Nous diviserons les grandeurs afin qu'elle soit comprise entre 0 et 255.

Conversion d'une seule tension

Afin de comprendre la conversion, nous avons réalisé dans un premier temps la conversion sur une seule tension. Nous avons utilisé un programme 'voltemétre.c' développé par un étudiant de l'IUT l'année dernière. Celui-ci permet d'acquérir une tension et de l'afficher sur un écran LCD. Dans la suite des séances, nous n'avons pas utilisé ce programme afin de convertir deux tensions. Par conséquent, nous ne feront pas une étude exhaustive sur 'voltemétre.c' .

```
/*Déclaration des bibliotèques*/
#include<mega8535.h>
#include<lcd.h>
#include<delay.h>
#include<stdio.h>
/*Prototypes de fonctions*/
interrupt[ADC_INT]void adc_isr(void);
void brdInit(void);
/*Déclaration des variables*/
int tension;
unsigned char tampon[20];
/*programme principal*/
void main (void)
{
      brdInit();
      #asm("sei")
      ADCSRA|=0x40;                  //on lance une seule fois la conversion
      while(1)
      {
      }
}

/*Définitions des fonctions*/
void brdInit(void)
{
         #asm
         .equ __lcd_port=0x15
         #endasm
         lcd_init(16);

         lcd_clear(); // cette fonction permet d'effacer l'écran LCD.

         ADMUX=0b01000000;
         ADCSRA=0b10101110;
         SFIOR=0x00;
}
interrupt[ADC_INT]void adc_isr(void)
{
            lcd_gotoxy(0,0);
            lcd_putsf("   Tension 1   ");


            tension=ADCW; // permet de ne pas avoir de calcul a faire
                        // il met le résultat de 2x8 bits bout a bout
            sprintf(tampon,"V=%5d",tension);
            lcd_gotoxy(0,1);
            lcd_puts(tampon);                  //on affiche
            delay_ms(100);
}
```

Ce programme met en place deux fonctions dont les prototypes sont :

Prototype de la fonction :

void brdInit(void);

Valeur de retour : aucune valeurs de retour.

Paramètres : aucun paramètres passés par la fonction.

Description : cette fonction initialise l'écran LCD et le convertisseur.

Prototype de la fonction :

interrupt[ADC_INT]void adc_isr(void);

C'est une fonction de type interruption

Valeur de retour : la fonction retourne la valeur numérique convertie par le CAN.

Paramètres : aucun paramètres passés par la fonction.

Description : cette fonction retourne la valeur numérique d'une tension par l'intermédiaire d'une interruption interne au micro-contrôleur à intervalle régulié, nous effectuons une interruption et nous récoltons la valeur de la tension. Nous aurons l'opportunité de reparler d'interruptions dans la suite du rapport de projet.

Des tests sur ce programme ont été réalisés pour confirmer son bon fonctionnement, une carte électronique avec des potentiomètres avait été réalisée par le professeur. Cette carte est reliée au micro-contrôleur au niveau des pattes destinées au conversion analogique-numérique. En agissant sur un potentiomètre, nous faisons varier la tension envoyée sur l'organe de commande, cette variation est alors observable grâce à une lecture sur l'écran LCD.

Conversion de deux tensions

Dans un premier temps, nous avons voulu réutiliser le programme précédent afin de réaliser cette opération. Nous n'avons pas réussi à convertir deux tensions.

Finalement, nous avons utilisé le générateur de programme automatique : CodeWizardAVR.

Nous avons aussi utilisé le Datasheet du micro-contrôleur afin de régler le CAN.

Initialisation du CAN

Le générateur de programme nous a fourni les lignes de programme afin d'initialiser le convertisseur. L'initialisation du CAN consiste essentiellement à paramétrer les différents registres du micro-contrôleur relatif au CAN.

Sachant que l'initialisation est effectuée automatique, nous expliquerons les différentes lignes du cadre suivant brièvement grâce au datasheet du constructeur.

```
#define ADC_VREF_TYPE 0x00
[suite du programme ... ]
// 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: None
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x84; // soit ADCRA = 0b10000100 (en binaire)
SFIOR&=0xEF; // soit SFIOR&= 0b11101111
```

ADMUX est un registre de 16 bits permettant de sélectionner une tension de référence pour le CAN, c'est à dire l'entrée du micro-contrôleur sur laquelle nous lirons les grandeurs analogiques.

Dans notre cas, le registre ADMUX est égale à 0x00 (grandeur hexadécimale), tous les bits sont mis à 0. Par conséquent, les tensions seront lues sur l'entrée ADCO du CAN, soit la patte numéro 40 du micro-contrôleur

SFIOR est un registre de 16 bits permettant de sélectionner la source de l'auto trigger du CAN.

D'après le datasheet (*page 223/321*), la source du trigger est le timer 1.

ADCRA est le registre A de 16 bits permettant de contrôler et définir les paramètres du CAN.

Nous avions choisi différents paramètres du CAN dans le CodeWizardAVR, nous les retrouvons dans les valeurs des bits du registre.

D'après le datasheet (*page 221/321*), le bit de poids fort étant mis à 1 signifie que nous activons le CAN.

Nous avons aussi choisi la fréquence de l'horloge à laquelle les conversions seront effectuées.

D'après le datasheet (*page 222/321*), le bit 2 étant mis à 1 et les derniers bits mis à 0, cela signifie que nous divisons par 16 l'horloge propre au micro-contrôleur $16.10^6/16=1000$kHz.

Nous retrouvons bien la valeur mis en commentaire par le Code Wizard AVR.

Finalement, une fonction a été réalisée par le générateur de programme automatique.

Prototype de la fonction :

unsigned int read_adc(unsigned char adc_input);

Valeur de retour : valeur numérique, entier non signé

Paramètres : numéro de l'entrée du CAN.

Définition de la fonction :

```
/*Définition des fonctions*/

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;
}
```

Description de la fonction :

Dans un premier temps, la conversion est permise par l'écriture de 0x40 sur le registre A du CAN (d'après datasheet, *page 221/321*).

Ensuite, tant que la conversion n'est pas terminée, la fonction autorise la conversion et retourne la valeur numérique de la tension sur les 16 bits du registre consacrés à sauvegarder les conversions (ADCW).

Afin de mieux comprendre la fonction, voici l'ordinogramme :

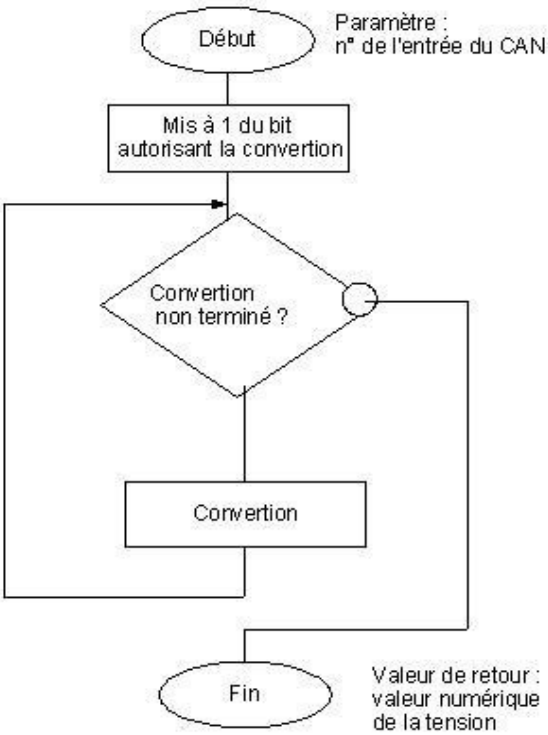


Illustration 21: Ordinogramme de la fonction "CAN"

Nous avons alors effectué des tests afin de contrôler le bon fonctionnement de la conversion analogique-numérique sur deux tensions.

Nous avions à notre disposition le même dispositif que les tests sur 'voltemétre.c'.

Nous affichons les deux valeurs numériques relatives aux tensions sur un écran LCD. Les tests effectuées confirment le bon fonctionnement du programme.

Nous exposerons essentiellement la boucle infinie assurant l'affichage des grandeurs numériques. Le paramétrage de l'écran LCD ne sera pas indiqué.

```
/*Définition des bibliothèques
#include <mega8535.h>
#include <lcd.h>
#include<delay.h>
#include<stdio.h>
#define ADC_VREF_TYPE 0x00

/*Prototype des fonctions*/
unsigned int read_adc(unsigned char adc_input);

/*Déclaration des variables*/
        int tension1;
        int tension2;
        unsigned char tampon [20];
        unsigned char tampon1 [20];

/*programme principal*/
        void main (void)
{

/*initialisation des entrées/sorties*/
//dispensable pour la compréhension du programme vis à vis des tests

/*Initialisation du CAN*/
//lignes de code déjà étudiées

/*Initialisation de l'écran LCD*/
////dispensable pour la compréhension du programme vis à vis des tests

/*Boucle infinit*/
while (1)
    {
    lcd_gotoxy(0,0);
    lcd_putsf("  Tension n°1  ");
    lcd_gotoxy(0,2);
    lcd_putsf("  Tension n°2  ");
    // Place your code here
     tension1=read_adc(1)/4;
     tension2=read_adc (6)/4;
     sprintf(tampon,"Vc=%5d",tension1);
     sprintf(tampon1,"Vse=%5d",tension2);
     lcd_gotoxy(0,1);
     lcd_puts(tampon);                          //on affiche
           lcd_gotoxy(0,3);
     lcd_puts(tampon1);    //on affiche
      delay_ms(10);
    };
}
```

Suite du programme testant le fonctionnement de la conversion analogique-numérique de deux tensions :

```
/*Définition des fonctions*/

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;
}
```

Description du programme :

Nous affichons sur l'écran LCD une légende pour les tensions que nous convertissons.

Ensuite, nous plaçons dans deux variables déclarées au préalable la valeur numérique obtenue après appel de la fonction permettant la conversion.

Ensuite, nous créons deux chaînes de caractères avec la fonction 'sprintf' où nous plaçons les deux valeurs numériques. Nous diviserons par quatre la valeur convertie afin d'être dans l'intervalle [0;255].

Finalement, nous affichons sur l'écran LCD.

Afin de mieux comprendre la fonction, voici l'ordinogramme :

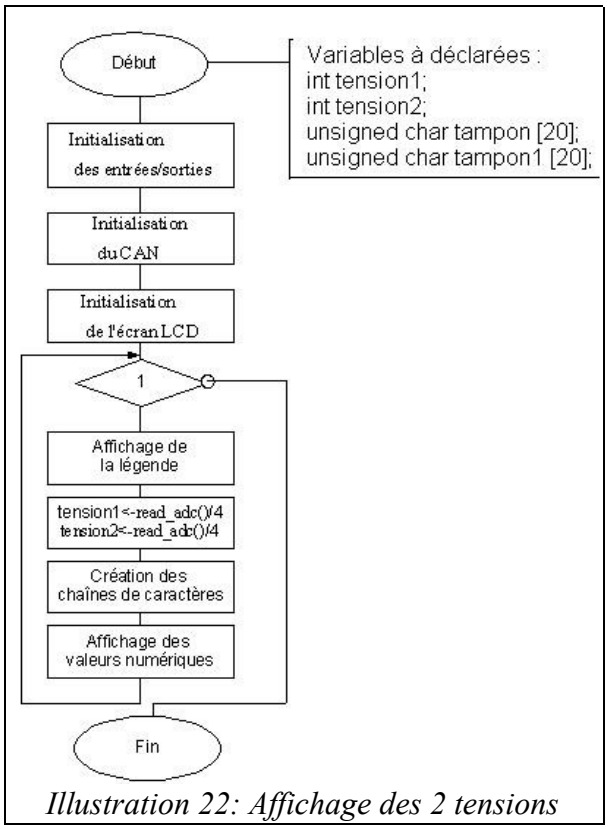


Illustration 22: Affichage des 2 tensions

3.3/ Mise en place d'une routine d'interruption

Le principe de l'interruption est très utilisé en informatique industrielle. En effet, les interruptions permettent d'effectuer des opérations à un intervalle régulier. Nous verrons le réglage des intervalles dans cette partie.

Le générateur de programme automatique permet de configurer trois Timers disponibles sur le micro-contrôleur. La configuration de différents registres internes au composant permet de régler TIMER 0, TIMER 1 et TIMER 2. Tout comme le Convertisseur Analogique Numérique, nous utiliserons le Code Wizard afin de paramétrer l'interruption.

Nous utiliserons TIMER 0.

a) Configuration de l'interruption

Afin d'illustrer notre configuration, voici une impression écran du Code Wizard.

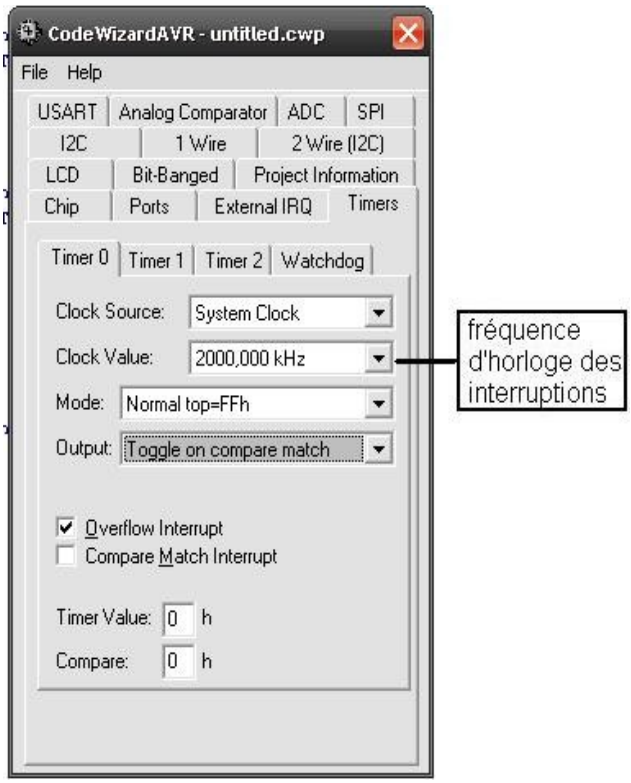


Illustration 23: Configuration Code Wizard des interruptions

Nous avons choisi d'effectuer des interruptions sur fronts montants et descendants, afin d'avoir des séquences de calcul mises à jour plus régulièrement. Nous aurions pu choisir un seul type de front.

Nous avons défini une fréquence d'horloge égale à 2000 Mhz.

La configuration de la sortie "Toggle on compare match" ne sera pas expliqué dans ce rapport. Des explications n'apporteraient rien d'essentiels au projet.

Nous avons alors effectué une pré-visualisation du nouveau programme avec la configuration du TIMER 0.

Afin de mieux visualiser le principe des interruptions, nous présentons un schéma.

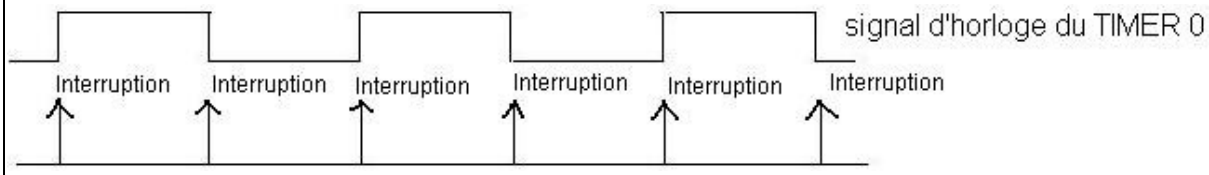


Illustration 24: Interruptions

La fréquence d'interruption peut être aussi appelée fréquence d'échantillonnage. C'est un terme que nous retrouvons dans la littérature du filtrage numérique.

Nous avons utilisé la pré-visualisation de programme afin de récolter les lignes de programmes permettant de configurer le TIMER 0.

```
// Clock source: System Clock
// Clock value: 2000,000 kHz
// Mode: Normal top=FFh
// OC0 output: Toggle on compare match
TCCR0=0x12;
TCNT0=0x00;
OCR0=0x00;
```

```
// Timer/Counter 0 initialization// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=0 State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x08;
```

Dans ces deux cadres, nous pouvons visualiser les modifications apportées lorsque nous paramétrons notre TIMER.

Nous utiliserons le datasheet du micro-contrôleur afin d'expliquer chaque ligne.

Les modifications observables se font sur l'état du registre TCCR0 et le registre de direction de données DDRB.

Nous utiliserons le datasheet du micro-contrôleur afin d'expliquer ces différents changements. Le registre TCCR0 a pris comme valeur TCCR0=0x12. Cela signifie que les bits 1 et 4 du registre ont été mis à 1.

D'après la *page 83 sur 321* du datasheet, le fait de mettre le bit 4 à 1 implique le fait que la sortie OC0 prédomine sur l'ensemble des entrées/sorties. Néanmoins la programmation d'ATMEL oblige leurs déclarations. Toutes les entrées doivent avoir leur registre de direction DDR mis à 0, les sorties doivent avoir leur registre de direction DDR mis à 1.

La ligne de configuration DDRB = 0x08 signifie que le PIN relatif à OC0 a été déclaré comme une sortie.

Programmation de la routine d'interruption

Le générateur de programme automatique nous fournit aussi une fonction dont le prototype est :

interrupt [TIM0_OVF] void timer0_ovf_isr(void);

C'est une fonction de type interruption utilisant le TIMER 0 rentré en configuration du Code Wizard. Aucun paramètre n'est passé par la fonction.

Nous avons décidé d'utiliser la définition de cette fonction en incorporant l'ensemble des lignes de codes permettant de convertir les tensions, traitements sur les grandeurs numériques. Ces actions se feront régulièrement grâce aux interruptions.

Dans un premier temps, nous avons juste placé deux lignes de codes.

Définition de la fonction :

```
interrupt [TIM0_OVF] void
timer0_ovf_isr(void)
{
PORTD.0=1;
PORTD.0=0;
}
```

Nous plaçons alternativement le PIN 0 du PORT D à 1 et 0.

Nous effectuons des tests, nous présentons alors cet oscillogramme illustrant les interruptions

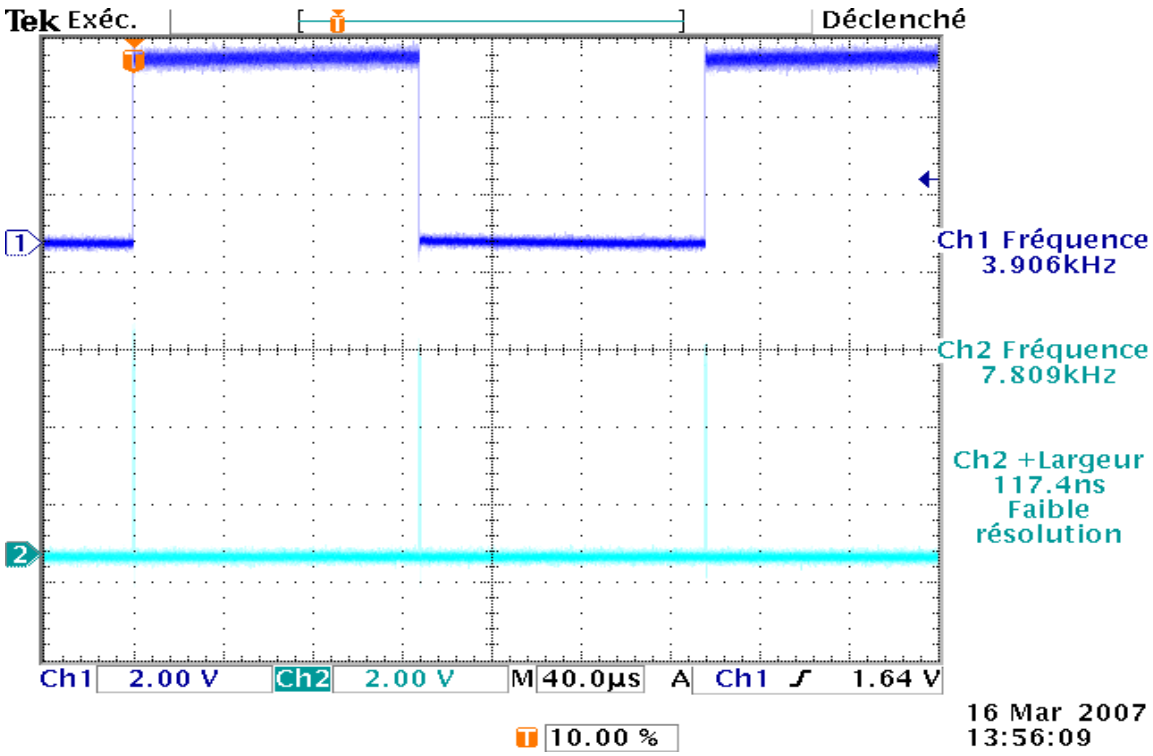


Illustration 25: Oscillogramme de l'interruption

Le signal bleue (au-dessus) est le signal de sortie du TIMER 0.

Le signal vert (en-dessous) est le signal relatif aux interruptions.

Les pics observés sur le signal vert correspondent aux deux lignes de code rentrées dans la fonction routine d'interruption.

Nous pouvons remarqué que la fréquence d'échantillonnage est égale au double à la fréquence de sortie de l'OVERFLOW.

Les pics relatifs durent 128ns, c'est le temps que le micro-contrôleur exécute les deux lignes de code. Quand on effectuera les actions (conversion, filtrage), nous devons s'arranger pour que la durée des pics soit suffisante.

La fréquence d'échantillonnage est égale à 7,809 kHz. Nous avions choisi une fréquence d'interruption dans le code Wizard égale à 2MHz. Le TIMER 0 est effectué sous 8 bits.

Or 2MHz/255=7,809kHZ. Ainsi, nous retrouvons la configuration du code Wizard pendant ce test.

Ensuite, nous avons fait appel à la fonction permettant la conversion analogique numérique dans la routine d'interruption :

```
/*Définition des fonctions*/
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;
}
```

De nouvelles opérations ont été ajoutées, nous devrions obtenir une largeur d'impulsion plus importante.

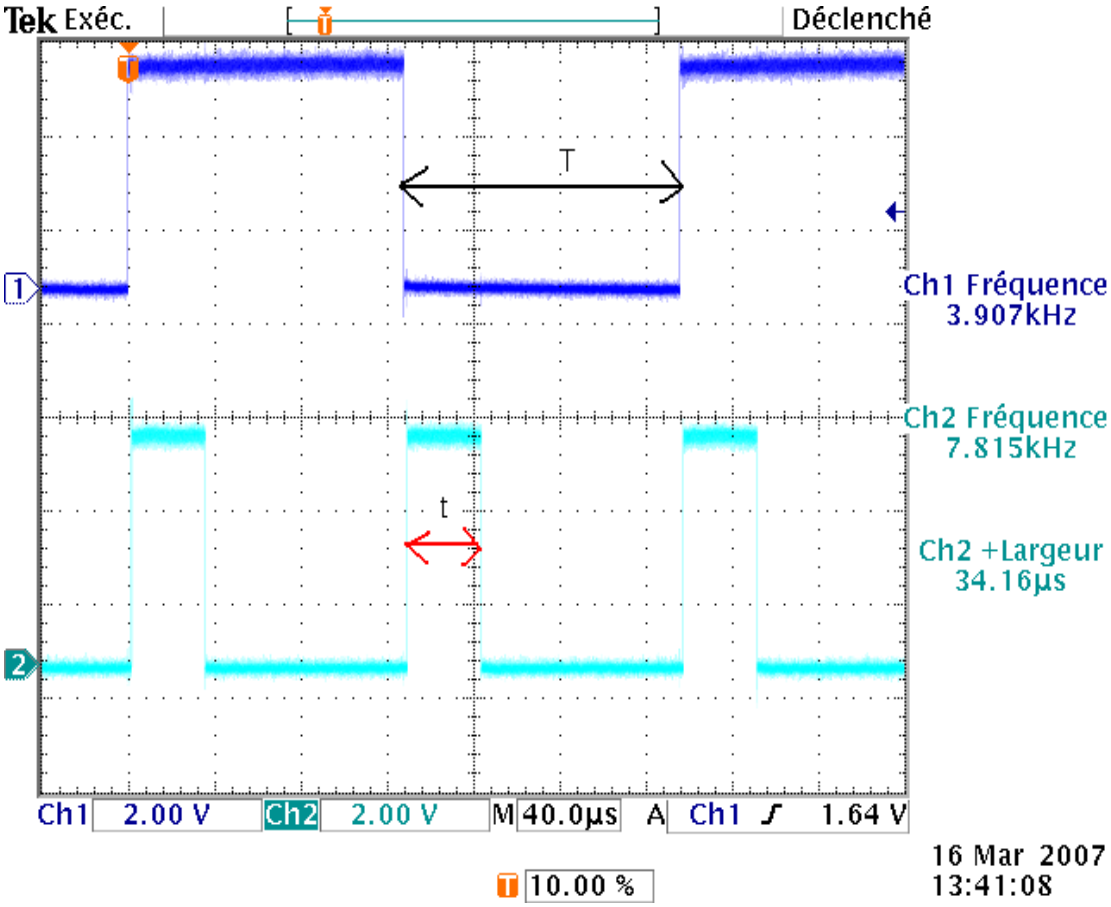


Illustration 26: Conversion de 2 tensions dans la routine d'interruption

Soit T le temps séparant deux types de front du signal de sortie du TIMER 0.

Soit t la largeur d'impulsion relative à une routine d'interruption dont le contenu est la conversion de deux tensions.

Le fait de convertir deux tensions demande une largeur d'impulsion relative aux interruptions beaucoup plus importantes.

Nous avons aussi testé avec d'autres lignes de codes pour voir l'influence sur le temps des interruptions. Nous pouvons en conclure que seules les conversions agissent réellement sur la durée. La seule condition dans l'utilisation des interruptions est que : t<T.

La commande du hacheur s'effectue avec une fonction propre au micro-contrôleur ATMEL nommée PWM. Nous l'exposerons dans la partie suivante.

3.4/ La fonction PWM

Le hacheur doit être commandé par un signal en créneau dont les largeurs d'impulsion doivent varier selon le rapport cyclique désiré pour commander le moteur. En électronique,

les signaux dont la largeur d'impulsion en fonction du temps sont appelés signal MLI (Modulation de Largeur d'Impulsion). Nous voulons recréer cela avec le micro-contrôleur.

Ainsi, nous avons utilisé la fonction PWM : Pulse Width Modulation, modulation de largeur d'impulsion. C'est une fonction utilisable avec le micro-contrôleur et configurable avec les différents TIMER. Nous avons décidé d'activer la fonction grâce au TIMER 1.

Configuration du TIMER 1.

Tout comme les fonctions déjà utilisées , le mode PWM possède une configuration entièrement réalisable avec le Code Wizard.

Voici l'impression écran des paramètres du générateur de programme automatique :

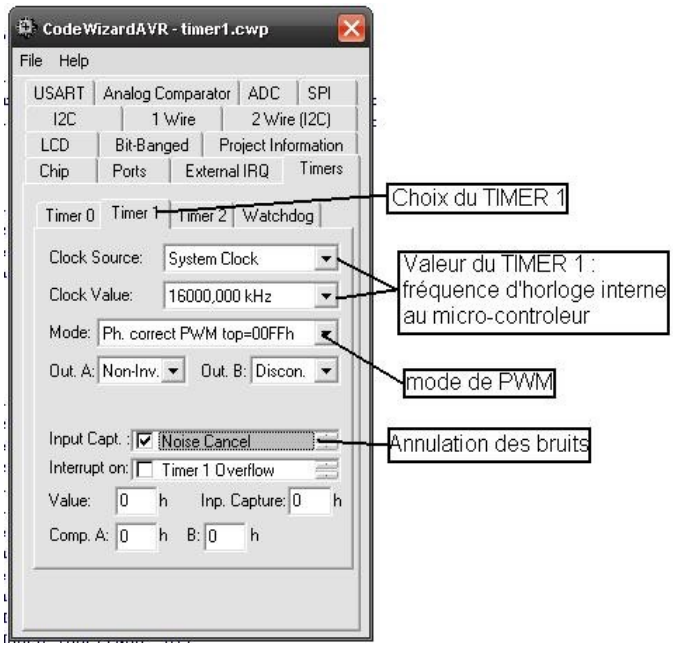


Illustration 27: Configuration TIMER 1

Nous utilisons encore la fonction 'Program preview' fournit par Code Wizard afin de visualiser les configurations du registre relatives au mode PWM désiré.

Ainsi nous obtenons ces lignes de code que nous allons expliquer à l'aide du datasheet :

```
// Timer/Counter 1 initialization                TCCR1A=0x81;
// Clock source: System Clock                    TCCR1B=0x81;
// Clock value: 16000,000 kHz                    TCNT1H=0x00;
// Mode: Ph. correct PWM top=00FFh               TCNT1L=0x00;
// OC1A output: Non-Inv.                         ICR1H=0x00;
// OC1B output: Discon.                          ICR1L=0x00;
// Noise Canceler: On                            OCR1AH=0x00;
// Input Capture on Falling Edge                 OCR1AL=0x40;
// Timer 1 Overflow Interrupt: Off               OCR1BH=0x00;
// Input Capture Interrupt: Off                  OCR1BL=0x00;
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;
```

Une étude du datasheet nous permet d'affirmer que la sortie du TIMER 1 en mode PWM est le registre OCR1AL. Nous placerons donc le résultat du traitement des tensions Vc et Vm dans ce registre afin de créer notre signal modulé en largeur d'impulsion. Selon l'écart et le type d'action, nous pourrons observer la variation de la largeur d'impulsion.

De plus, nous avons configuré la sortie du PWM (OCR1AL) comme sortie du micro-contrôleur. Le PIN relatif à cette sortie est placé sur le PORTD. Nous avons décidé de configurer l'ensemble des PIN du PORT D comme sortie. Ainsi, nous trouvons dans le registre de direction du PORT :

DDRD = 0xFF

Ensuite, nous étudions les valeurs des registres TCCR1B et TCCR1A qui ont pour valeur hexadécimale 0x81.

D'après le datasheet *pages 110 à 112 sur 32*1 expliquant TCCR1A , les bits 0 et 7 sont mis à 1.

Le fait de mettre à 1 le bit 7 a pour même conséquence la mise à 1 du bit relatif au PIN OC0. La conséquence de la mise à 1 du bit 0 est de pouvoir modifier la forme du signal.

L'ensemble des combinaisons est rassemblé dans le tableau de la page 112 sur 321.

D'après le datasheet *pages 113 sur 321* expliquant le registre TCCR1B, les bits 0 et 7 sont aussi mis à 1.Mettre à 1 le bit 7 nous permet d'annuler le bruit de l'entrée du PWM. Le bit 0 permet de sélectionner la valeur de l'horloge du TIMER 1. Le tableau de la *page 113 sur 321* récapitule l'ensemble des combinaisons de valeur d'horloge : division par différentes valeurs de la fréquence d'horloge interne au micro-contrôleur. Nous observons que nous divisons par 1 cette horloge.

L'étude du registre TIMSK s'effectue avec la *page 115 sur 321* du datasheet. Le bit 0 est mis à 1.

Dans l'étude de ces registres, nous pouvons en déduire que nous sommes conforme à nos configurations rentrées dans le Code Wizard.

Traitement des tensions

Nous avons converti deux tensions nommées Vc et Vm. En Automatique analogique, ces deux tensions sont comparées à l'aide d'un comparateur dont le symbole est :

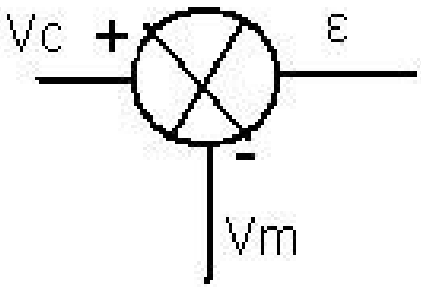


Illustration 28: Symbole du comparateur

Nous noterons ε la comparaison entre la tension de consigne Vc et la tension de mesure Vm.

Dans le cadre de notre projet, nous effectuerons ces mêmes opérations d'un point de vue numérique.

Les valeurs numériques relatives à Vc et Vm varient entre 0 et 255.

Nous avons décidé que ε aurait la même plage de variation. Nous avons donc limité les valeurs négatives et les valeurs supérieures à 255.

Dans un premier temps, nous avons effectué une simple action proportionnelle à la valeur numérique d'epsilon. Ce traitement sera effectué dans la routine d'interruption du TIMER 0.

Nous obtenons alors comme définition de fonction de la routine :

```
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here
PORTD.0=1;
Vc=read_adc(1)/4;
Vm=read_adc (6)/4;
Epsilon=Vc-Vm;
VAlpha=GAIN*Epsilon;
if (Valpha<=0) // élimination des valeurs négatives Valpha=0;
//résultantes de la comparaison
if(VAlpha>=255) // élimination des valeurs supérieures à
Valpha=255;         //255
if (VAlpha>=0 && VAlpha<=255 )
{OCR1AL=VAlpha;} // nous rentrons la valeur dans le
                  //registre de sortie du PWM
}
```

Afin de mieux comprendre cette fonction, nous exposons un ordinogramme :

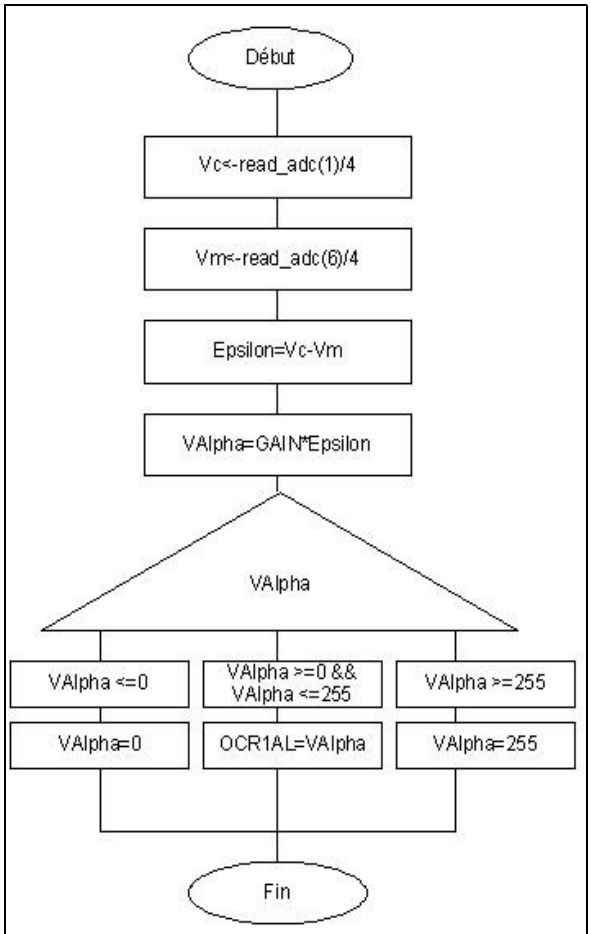


Illustration 29: Traitement des 2 tensions dans la routine d'interruption

Vous trouverez en annexe le listing de la programmation.

3.5/ Relation entre le PC, le hacheur et le moteur.

Nous n'avons pas pu effectué tous les tests escomptés à la relation entre les trois organes.

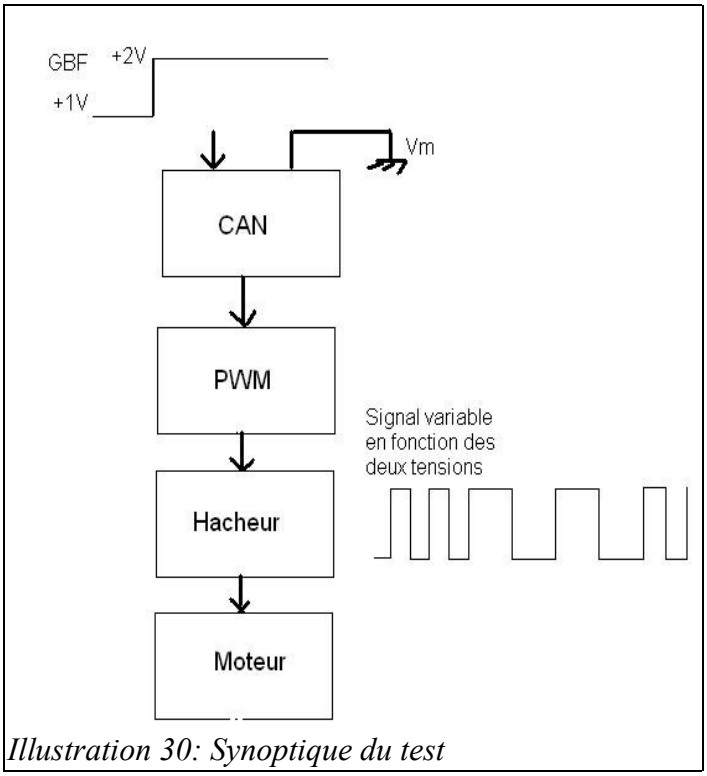


Illustration 30: Synoptique du test

Nous avons réussi à réaliser un seul test. Nous avons simulé un signal échelon compris entre +1 et +2 V à l'aide d'un GBF, il représente la tension de consigne Vc. Nous avons simulé en boucle ouverte, la tension de mesure est nulle.

Nous avons alors effectué des tests grâce à un oscilloscope.

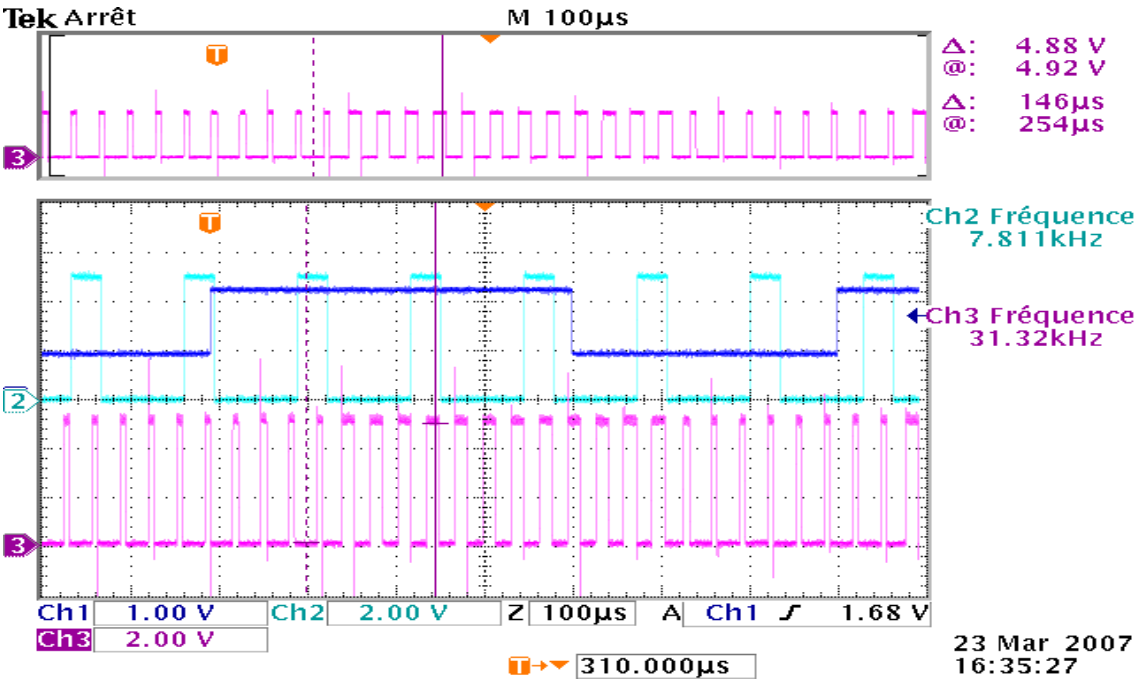


Illustration 31: Relation entre Hacheur et Micro controleur

Le signal bleu foncé est le signal du GBF

Le signal vert est le signal relatif au routine d'interruption : acquisition des tensions et traitements.

Le signal rose est le signal envoyé sur le hacheur.

Observation :

Le changement de rapport cyclique s'effectue après un changement de front sur le signal du GBF et le passage d'une impulsion relative à la routine d'interruption.

CONCLUSION

Durant les séances d'étude et réalisation du semestre quatre nous avons pu mettre en place différentes notions étudiées lors de notre cursus à l'IUT. Ainsi des matières telles que l'électronique, l'informatique, et l'automatique on été appliquées pour la réalisation de notre projet.

Tout d'abord nous avons fait de l'électronique, pour la conception de la carte d'adaptation de tension. Nous avons adapté la tension de sortie du capteur de courant placé sur le hacheur à la tension d'entrée du microcontrôleur. Nous avons aussi mis en œuvre sur cette carte le montage pont diviseur de tension relatif à la pédale d'accélération du Kart.

Ensuite on a lié l'électronique à l'automatique pour pouvoir trouver la fonction de transfert du système, c'est-à-dire l'identification du système. Ceci nous a permis de mettre en œuvre les méthodes d'identification abordées en cours d'automatique.

Finalement, nous avons développé un programme réalisant le calculateur. Ce dernier acquiert les deux tensions analogiques et les numérises (CAN). Il effectue aussi le traitement de ces deux grandeurs numériques (Correction). Nous avons alors développé un générateur d'impulsions modulables à l'aide du Timer1. Le filtrage numérique est basé sur l'échantillonnage. Donc nous avons mis en place une routine d'interruption à l'aide du Timer0.

Nous n'avons pas réussi à identifier le système de façon complète: Choix du meilleur correcteur. Néanmoins le travail qui reste à réaliser est moindre.

Ce projet nous a apporté beaucoup de connaissances sur la programmation du microcontrôleur et l'environnement du logiciel associé.

Index des illustrations

Les illustrations 1, 2 et 13ont été réalisées avec notre appareil photo numérique.

Les illustrations 3, 4, 19, 21, 22, 24, 28, 29 et 30 sont des illustrations réalisées par nos soins avec PAINT.

Les illustrations 5, 6, 7 ,9, 10, 11, 14 sont des impressions écrans du schéma de notre carte électronique modifiées avec PAINT.

Les illustrations 15, 16, 25, 26 et 31 ont été obtenues grâce à la copie sur disquette de relevés oscillographiques. Elles ont pu être modifiées sur PAINT.

Les illustrations 17 et 18 sont des impressions écran de la simulation effectuée sur ASL.

Les illustrations 19, 20, 23 et 27 sont des impressions écran du Code Wizard de nos différentes configurations sur Code Vision AVR.

Bibliographie

- Cours dInformatique de M.Petit, Lycée Leonard de Vinci, Amboise.

  Cours de terminale S option Science de l'ingénieur. Code Vision AVR Studio
- Bivliothèque ORCAD de Monsieur Thierry LEQUEU.
- Logiciel ASL 4 développé par le LMP.
- Datasheet du micro contrôleur AtmegaS8535 (321 pages) fournie sur format papier par Monsieur Thierry LEQUEU à l'ensemble du groupe.
- Datasheet de l'Amplificateur Opérationnel :

  http://pdf1.alldatasheet.com/datasheetpdf/view/25385/STMICROELECTRONICS/TL082A.html
- Information sur la programmation sur le logiciel Code Vision AVR :

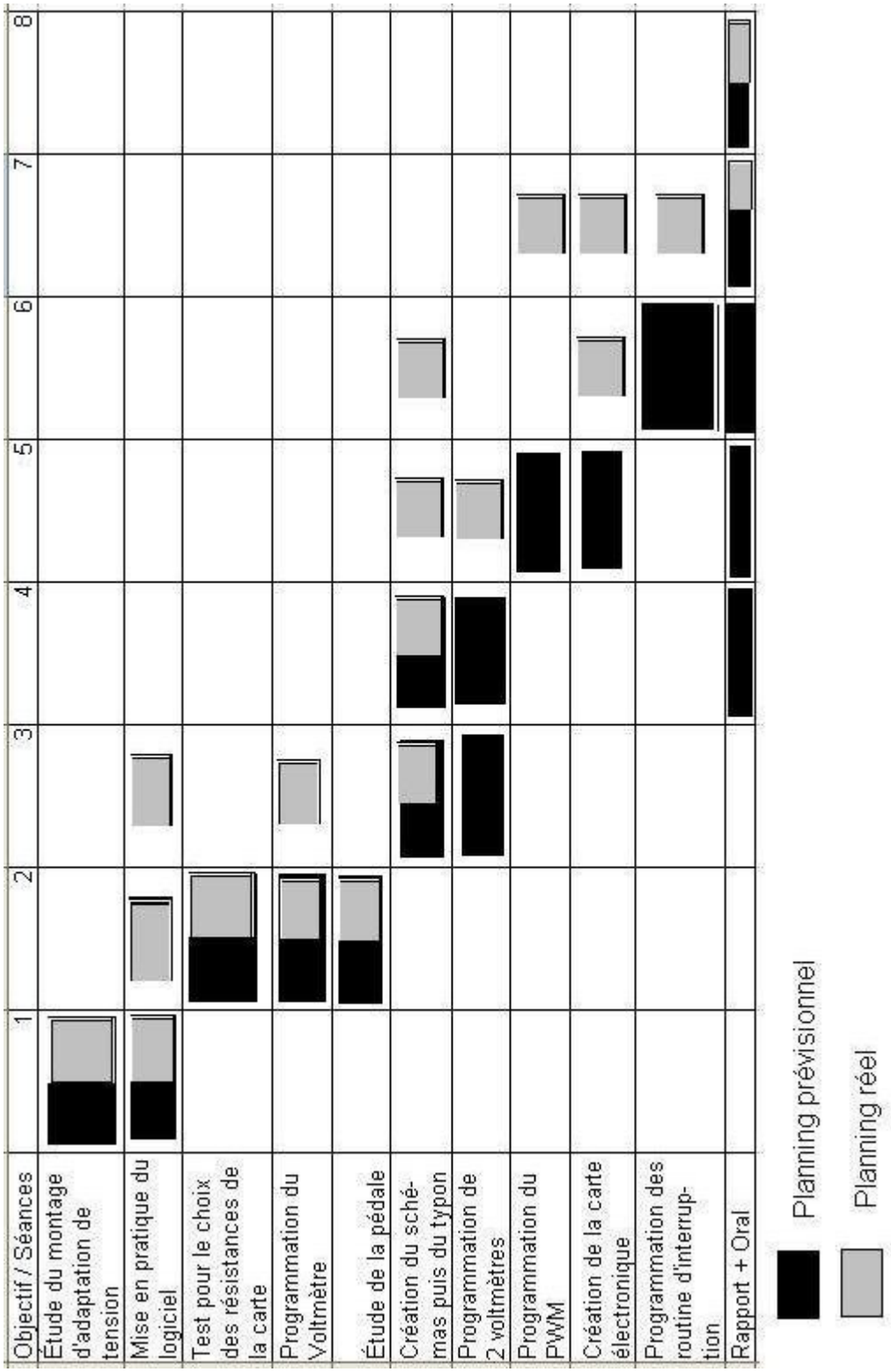
  http://www.ac-orleans-tours.fr/sti-gel/MICROCONTROLEUR/AVR.HTM

  consulté le 19 janvier 2007.

  http://www.micrelec.fr/lyctech/DevAVR/PRESENTATION.HTM#top

  consulté le 19 janvier 2007.

Planning


Objectif / Séances
1
2
3
4
5
6
7
8
Étude du montage d'adaptation de tension
Mise en pratique du logiciel
Test pour le choix des résistances de la carte
Programmation du Voltmètre
Étude de la pédale
Création du schémas puis du typon
Programmation de 2 voltmètres
Programmation du PWM
Création de la carte électronique
Programmation des routine d'interruption
Rapport + Oral
Planning prévisionnel
Planning réel

ANNEXES

ANNEXE 1 : Listing du programme final.

ANNEXE 2 : Datasheet de l'amplificateur opérationnel TL082.

ANNEXE 3 : Pages référant les explications de registre des différents registre du ATmégaS8535.

ANNEXE 1

Listing du programme final.

ANNEXE 2

Datasheet de l'amplificateur opérationnel TL082.

ANNEXE 3

Pages référant les explications de registre des différents registre du ATmégaS8535