GRANDGIRARD Mickaël
GRANDGIRARD Vincent

Enseignant responsable: Éric CHRETIEN

Licence professionnelle Véhicules Électronique et Gestion des Automates

PROJET TUTORE

Système programmé de cycle de décharge batterie

IUT Belfort-Montbéliard

Année 2010-2011

Sommaire

Annexe

Introduction:

Tout d' abord, l'objectif de ce projet tutoré consiste à créer un système permettant de proposer différents types de mode de décharges de batterie. En effet nous allons utiliser deux capteurs LEM (un courant et l'autre tension) pour visualiser la décharge de la batterie.

Description du sujet:

Pour connaître les caractéristiques d'une batterie, il est important de la tester pour différents modes de décharges.

- Recherche sur les protocoles de décharges
- Recherche sur le type charges à utiliser
- Recherche sur les valeurs limites de tests

L' utilisateur à l'aide d'une interface graphique doit pouvoir programmer différents types de décharge batterie avec un contrôle des grandeurs caractéristiques de la batterie.

Cahier des charges:

- Le but étant de connaître les différents cycles de décharges des batteries.

- Alimentation utilisée:
  Capteurs: +15/-15V
  Carte :+12/+5V

- Utilisation d' un capteur courant LEM LA 50-P pour mesuré l'intensité traversant la charge résistive (résistance, varistance).

- Utilisation d' un capteur tension LEM LV 25-P pour mesurer la tension aux bornes de la batterie.

- Intégration du système sur une carte électronique.

- Définir les contraintes d'intégrations.

Analyse de la valeur:

***Réalisation d'un système programmé de cycle de décharge batterie*.**

Orientation de l'action:

Définition de l'action:

L'action vise à améliorer la satisfaction du client en concevant un système programmé dans le cadre d'un projet.

Objectif de l'action:

A qui rend-elle service ?

Elle rend service à tous les utilisateurs qui ont des appareils possédant un système avec des batteries.

Sur quoi agit-elle ?

Elle agit sur les appareils ayant un système avec des batteries.

Dans quel but ?

Elle vise à :

- Augmenter les satisfactions du client;
- Renseigner au niveau de la capacité de la batterie (le temps restant avant de la recharger);
- Mettre en pratique les compétences acquises des élèves de la licence professionnelle VEGA.

Gestion de la qualité:

Qui ?

Le demandeur: l'IUT de Belfort.
Les responsables du projet : Vincent GRANDGIRARD et Mickaël GRANDGIRARD.

Où ?

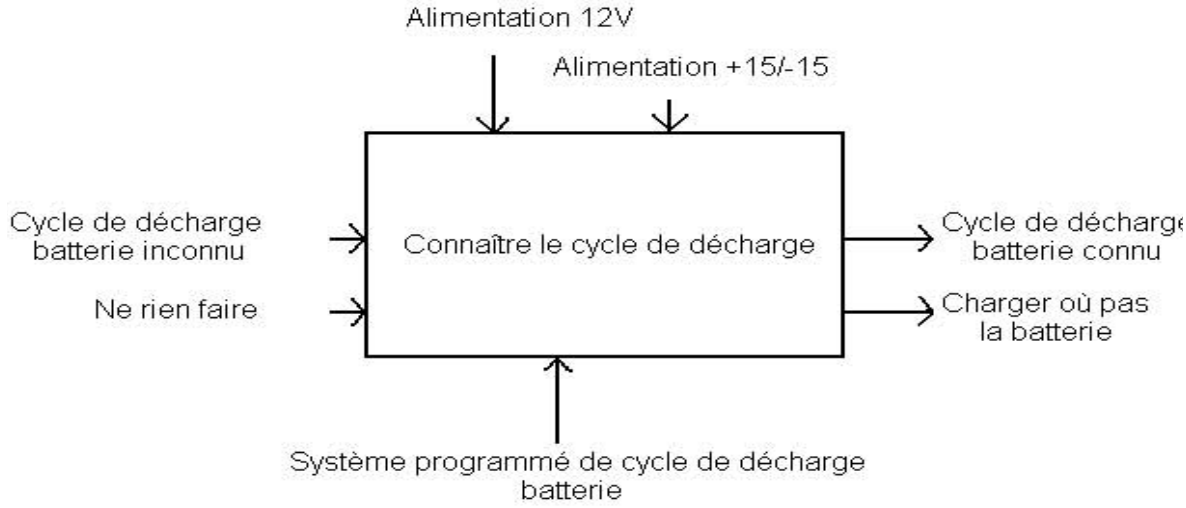
A l'IUT de Belfort en Franche-Comté (90)

Quand ?

Le projet devra être terminé à temps pour sa présentation en mars 2011.

Analyse fonctionnelle:

SADT:



« Bête à corne »:

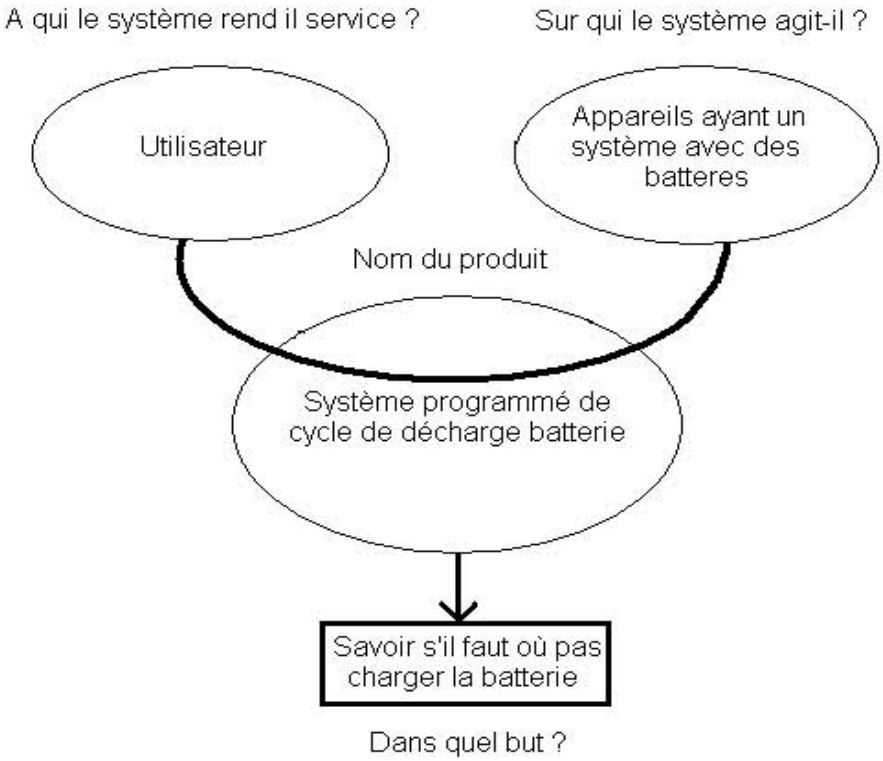
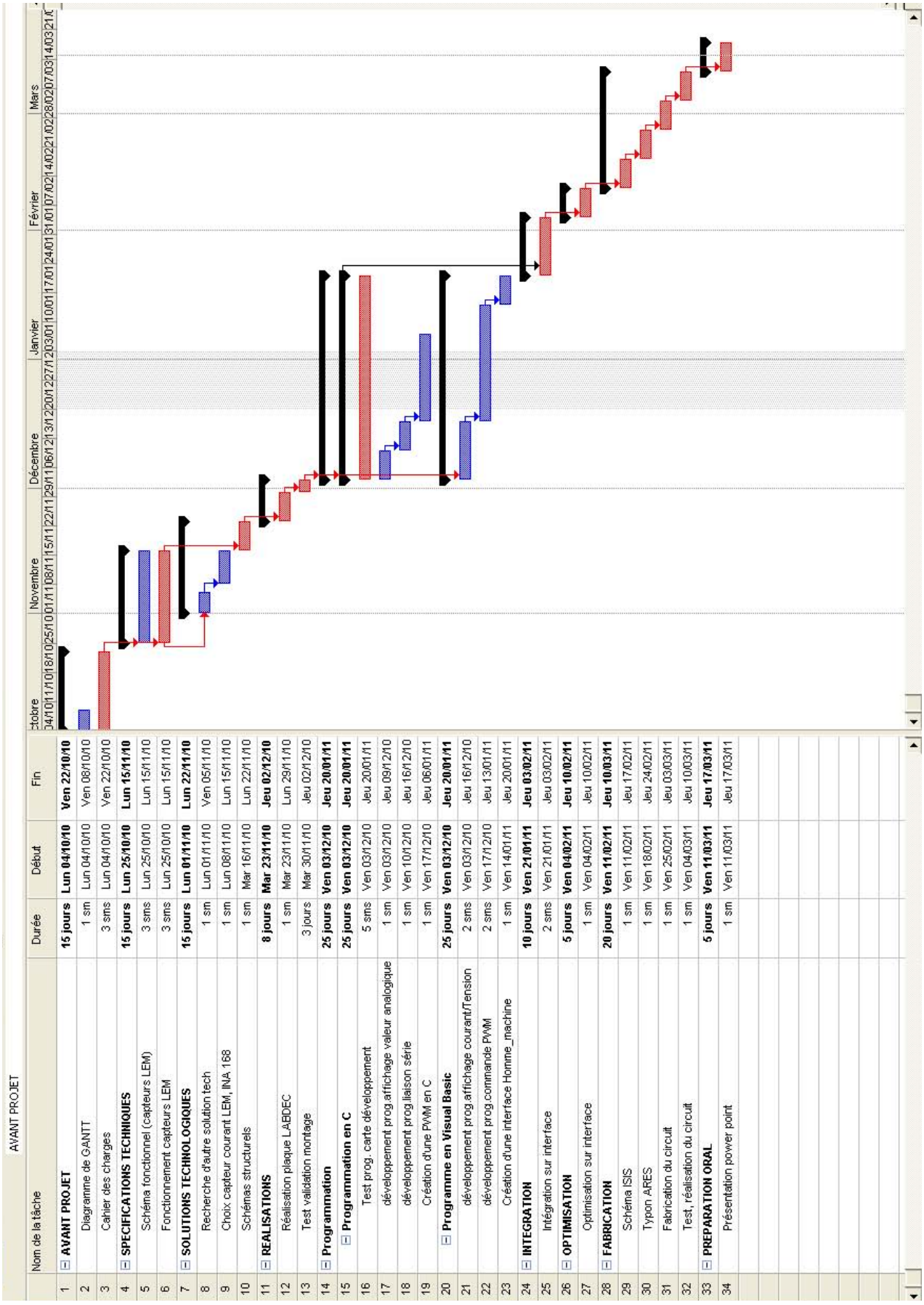


Diagramme de GANTT:



Développement de programme:

Présentation du matériel utilisé:

Tout d'abord, pour réaliser les différentes fonctions de tout les programmes, nous avons eu besoin d'une carte comportant un potentiomètre, de deux afficheurs 7 segments, d'un micro-contrôleur (16F887), d'un port RS232, d'un port d' alimentation, d' un bouton poussoir et des Switches.

Fonctions des composants:

- Potentiomètre : Il permet de faire varier les valeurs de 0 à 50 c'est-à-dire de 0 à 5V,
- Afficheurs 7 segments : Il permettent d'afficher les valeurs numériques en temps réel,
- Micro-contrôleur : Il permet de faire les fonctions que l'on désire en lui insérant un programme,
- Port RS232 : Il permet de transférer le programme du PC vers le micro-contrôleur,
- Port d'alimentation : Il permet d'alimenter la carte en 12 V,
- Bouton poussoir : Il permet de charger le programme dans le micro-contrôleur,
- Switches : permettant de faire varier les valeurs numériques.

Remarque:

Pour faire évoluer les valeurs numériques lues sur les afficheurs 7 segments, nous avons établie une liaison entre le micro-contrôleur et ces derniers à l'aide de fils.

Programmation en langage C:

Réalisation d'un programme pour afficher « bonjour »:

Tout d'abord, notre objectif était de réaliser un programme permettant d'afficher « Bonjour » sur l'hyperterminal de l'ordinateur.

Programme:

```
#include <16F887.h>                          //Choix du micro-contrôleur utilisé pour le projet
#fuses HS ,NOWDT,NOPROTECT,NOLVP//Configuration global
#use delay (clock=20000000)                  //Configuration de l'horloge interne
#use rs232 (baud=19200 ,xmit=PIN_C6, rcv=PIN_C7)    //Configuration du port série

void main() {
  while (TRUE)
  {
  printf ("\r\n");            //On fait un retour chariot et un saut de ligne
   printf ("bonjour");        //Toutes les secondes on écrit « bonjour »
  delay_ms(1000);
  }
}
```

Conclusion:

Le programme est bien envoyé dans le PIC 16F887 et on obtient bien le message « bonjour » sur le PC (écran sur le logiciel en C appelé Tiny Bootloader sur le terminal).

Création d'un voltmètre:

Ensuite, notre but était de réaliser un programme permettant de faire un voltmètre variant de 0 à 5V. Puis d'afficher les valeurs de ce dernier sur l'hyperterminal de l'ordinateur.

Programme:

```
#include <16F887.h>                    //Choix du micro-contrôleur utilisé pour le projet
#device ADC=10                         // Affichage des valeurs sur les afficheurs 7 segments
#fuses HS ,NOWDT ,NOPROTECT ,NOLVP     //Configuration global
#use delay (clock=20000000)            //Configuration de l'horloge interne
#use rs232 (baud=19200 ,xmit=PIN_C6,rcv=PIN_C7)    //Configuration du port série
unsigned int16 R, N, V;                //Variables déclarées comme un entier non signé
int resultat;                          //Variable déclarée comme un entier signé
unsigned int BCD (VALEUR)              //Déclaration de variables paramètres non signé
{
unsigned int unite,dizaine,sortie;     //Déclaration de variables paramètres non signé
dizaine=R/10;
unite=R-(dizaine*10);
dizaine=dizaine<<4;                    // Opérations pour afficher les dizaines et les unités
sortie=dizaine+unite;
return(sortie);
}

void main()
 {
setup_adc (ADC_CLOCK_DIV_32);    // Configuration de l'horloge interne / 32
setup_adc_ports(sAN0);           // Configuration entrée A0 analogique
while (TRUE)
{
set_adc_channel(0);      // Lecture sur le canal 0
N=read_adc();            //  « N » est une variable qui contient le résultat de la conversion
R=51 *N;                 // Opérations pour afficher 50 valeurs « 0 à 50 »
R=R/1023;
resultat=BCD(V);         // Conversion du résultat de binaire à décimale
output_d(resultat);      // Activation du port d
printf ("\r\n");         //On fait un retour chariot et un saut de ligne
printf ("%d",resultat);  //Toutes les 100 ms on affiche le résultat
 delay_ms(100);

  }
}
```

Conclusion:

Le programme est bien envoyé dans le PIC 16F887 et les valeurs des afficheurs 7 segments évoluent en temps réel sur l'hyperterminal de l'ordinateur.

Remarque:

Les valeurs des afficheurs 7 segments sont des valeurs hexadécimales et les valeurs lues sur l'hyperterminal sont des valeurs décimales. Donc lorsque nous lisons la valeur « 80 » sur l'hyperterminal cela veut dire que les afficheurs 7 segments affichent la valeur « 50 ». On à représenté tous les résultats dans un tableau.

Tableau:

Valeurs des afficheurs 7 segments (Hexadécimal)	0	1	2	3	4	5	6	7	8	9	10
Valeurs lues sur l'hyperterminal (Décimales)	0	1	2	3	4	5	6	7	8	9	16

11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
17	18	19	20	21	22	23	24	25	32	33	34	35	36	37	38	39	40	41	48

31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
49	50	51	52	53	54	55	56	57	64	65	66	67	68	69	70	71	72	73	80

Exemple:

Si nous lisons la valeur « 38 » sur les afficheurs 7 segments cela signifie que nous lirons sur l'hyperterminal de l'ordinateur la valeur « 56 » (voir ci-dessous). Pour obtenir un résultat en volt il faut diviser la valeur affichée par les afficheurs 7 segments par 10. Donc pour cet exemple la valeur serait de 3,8V.

Variation de valeurs numériques:

Enfin, nous avons réalisé un autre programme pour faire varier des valeurs numériques à l'aide de Switches.

Programme:

```
#include <16F877.h>                          //Choix du micro-contrôleur utilisé pour le projet
#fuses HS ,NOWDT ,NOPROTECT ,NOLVP           //Configuration global
#use delay (clock = 20000000)                //Configuration de l'horloge interne
#use rs232 (baud=9600 ,xmit=PIN_C6, rcv=PIN_C7)   //Configuration du port série

unsigned int valeur;                         //Variables déclarées comme un entier non signé

void main()
{
  while (TRUE)
  {
   valeur=input_b();                         // Lecture de la valeur sur le port b
   printf ("%d",valeur);                     // Envoi de données ( Entier signé) toutes les secondes
   delay_ms(1000);
   }
}
```

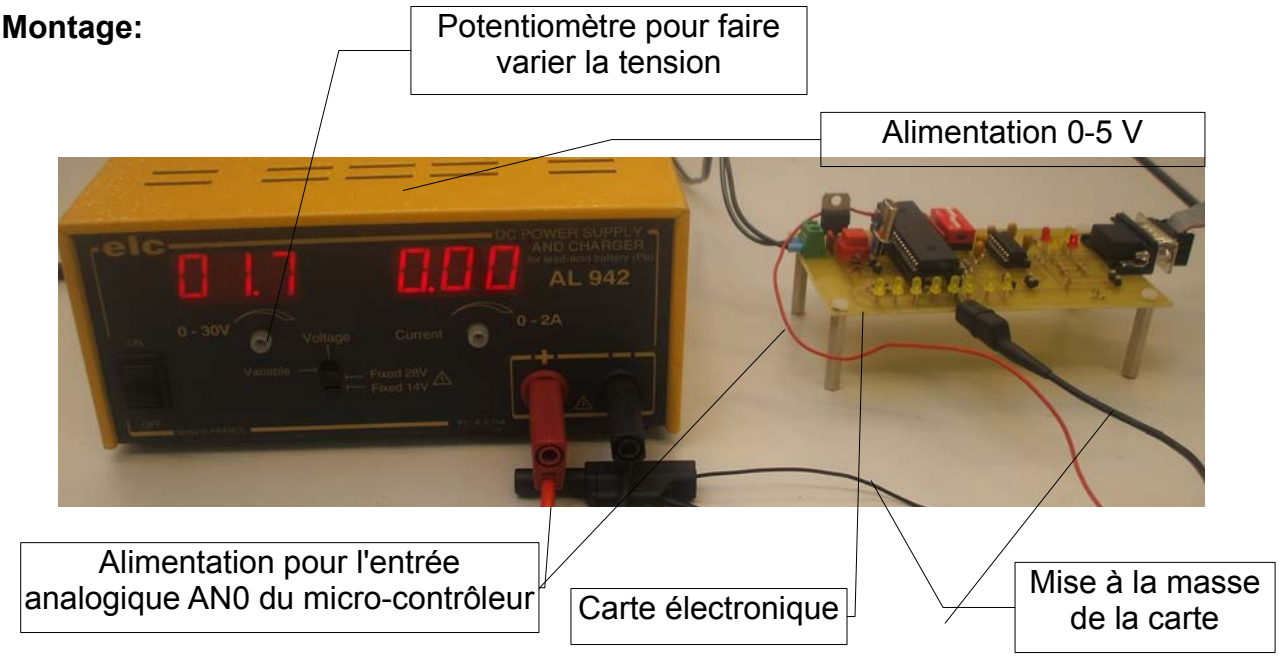
Conclusion:

Le programme est bien envoyé dans le PIC 16F877 et les valeurs évoluent bien en temps réel sur l'hyperterminal de l'ordinateur.

Variation de valeurs numériques en fonction d' une alimentation:

Dans un premier temps, nous avons réalisé un programme permettant de faire varier les valeurs numériques avec une alimentation (voir montage ci-dessous).

Montage:



Programme:

```
#include <16F877.h>                          //Choix du micro-contrôleur utilisé pour le projet
#fuses HS ,NOWDT ,NOPROTECT ,NOLVP           //Configuration globale
#use delay (clock = 20000000)                //Configuration de l'horloge interne
#use rs232 (baud=9600 ,xmit=PIN_C6, rcv=PIN_C7)   //Configuration du port série

unsigned int valeur;                         //Variables déclarées comme un entier non signé

void main ()
  {
setup_adc (ADC_CLOCK_INTERNAL);    //Configuration de l'horloge interne pour la fréquence de conversion
setup_adc_ports (ALL_ANALOG);      //Configuration pour l'utilisation de toutes les entrées analogiques
while (true)
  {
  (printf ("%u", valeur));      // Envoi de données ( Entier non signé)
  set_adc_channel(0);           // Lecture sur le canal 0
  delay_ms(1000);               // Changement de valeurs toutes les secondes
  valeur=read_adc();            // La variable valeur contient le résultat de la conversion
  }
  }
```

Conclusion:

Le programme est bien envoyé dans le PIC 16F877 et les valeurs évoluent bien en temps réel sur l'hyperterminal de l'ordinateur.

Remarque:

Il faut faire varier l'alimentation entre 0 et 5 V car le circuit ne supporte pas de fortes tensions.

<u>Création d'une PWM:</u>

Dans un second temps, nous avons réalisé un programme permettant de créer une PWM afin de commander la décharge de la batterie.

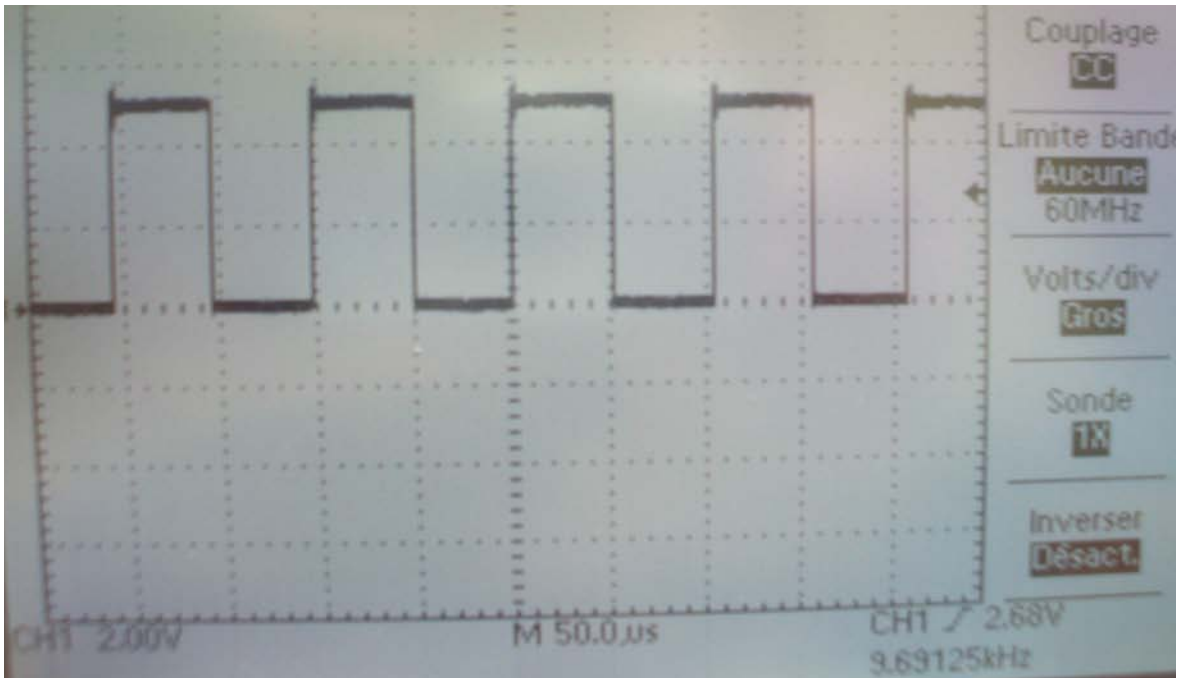
```
#include <16F877.h>                          //Choix du micro-contrôleur utilisé pour le projet
#fuses HS,NOWDT,NOPROTECT,NOLVP              //Configuration globale
#use delay(clock=20000000)                   //Configuration de l'horloge interne

void main()  {

  setup_ccp1 (CCP_PWM);                      //Premier canal, broche RC2 PIN (17)
  setup_timer_2(T2_DIV_BY_4,128,1);          //Activation PWM Timer 2 sur ccp1

    while (TRUE)  {
      set_pwm1_duty(64);                     //Valeur du rapport cyclique
    }
}
```

Conclusion:

Le programme est bien envoyé dans le PIC 16F877 et on observe bien une PWM sur l'oscilloscope.

Exemple:

Ci-dessous on visualise une PWM de 50%, c'est-à-dire que l'état de la PWM est à 1 ou 0 durant une demi période.



Programmation en Visual Basic (VB6,0):

Réalisation d'un programme pour afficher « bonjour »:

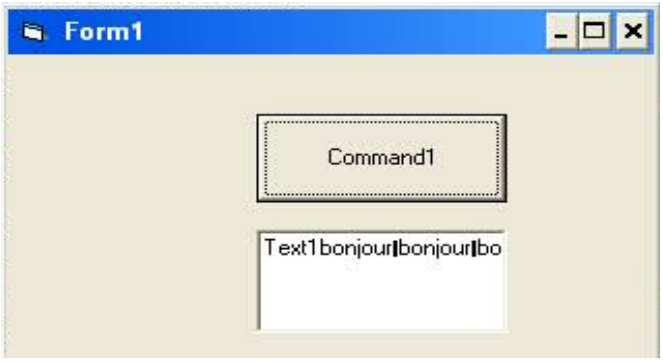
Tout d'abord, notre objectif était de réaliser un programme permettant d'afficher « Bonjour » sur l'hyperterminal du logiciel Visual Basic (VB6,0).

Programme:

```
Private Sub Command1_Click()
MSComm1.CommPort = 1
 //19200 bauds, pas de parité, 8 bits de données et 1 bit d'arrêt.
MSComm1.Settings = "19200,N,8,1"
 //Indique au contrôle qu'il doit lire la totalité du tampon si la propriété Input est utilisée.
MSComm1.InputLen = 0
 //Ouvre le port.
MSComm1.PortOpen = True
 //Envoie la commande Attention au modem.
MSComm1.Output = "ATV1Q0" & Chr$(13)
 //Vérifie que le modem répond "bonjour"
 //Attend le retour des données vers le port série.
 Dim buffer As String
 Do
 DoEvents
buffer = buffer & MSComm1.Input
 Text1.Text = Text1.Text & buffer
 Loop Until InStr(buffer, "bonjour" & vbCrLf)
 //Lit les données composant la réponse "bonjour" au niveau du port série.
 //Ferme le port série.
 MSComm1.PortOpen = False
 End Sub
```

Conclusion:

Le programme à bien été executé car on obtient bien le message « bonjour » sur l'hyperterminal du logiciel Visual Basic (VB6,0)

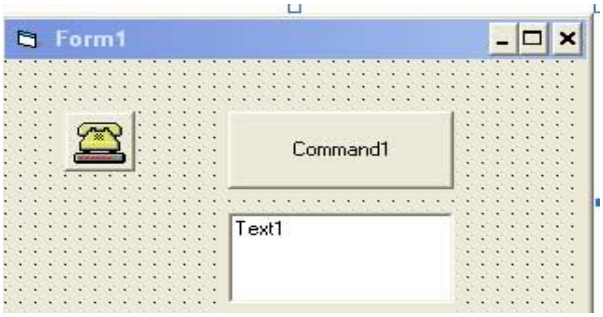


Remarque:

Pour faire dérouler le programme sur ce logiciel, il faut placer les éléments nécessaires afin de voir si ce que l'on affiche est correct par rapport à ce que l'on désire.

Exemple:

Pour afficher le message « bonjour » il faut placer sur la face ci-dessous un bouton poussoir (Command1). Ceci permettra de démarrer la lecture du programme. Ensuite il faut mettre une zone de texte « Text1 » pour afficher « Bonjour ». Ceci permet de voir immédiatement si cela correspond à ce que l'on veut ou pas. Enfin « le téléphone » permet de communiquer avec la liaison série RS232.



Variation de valeurs numériques:

Ensuite, notre but était de réaliser un programme permettant de faire varier des valeurs numériques. Puis afficher les valeurs de ce dernier sur l'hyperterminal du logiciel Visual Basic (VB6,0).

Programme:

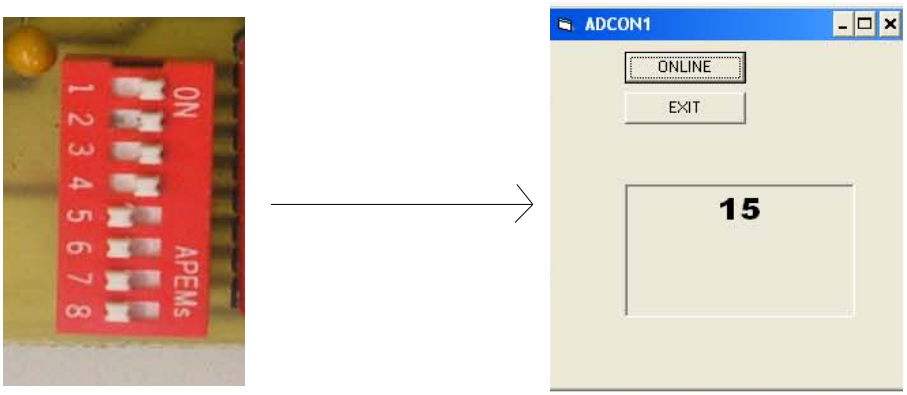
```
Private gv As Integer
Private a As String
Private Sub Command1_Click()
//Permet de démarrer le programme en cliquant sur « Command1 »
If MSComm3.PortOpen = False Then
MSComm3.PortOpen = True
//Ouvre le port.
Timer1.Enabled = True
Command1.Caption = "ONLINE"
//Lorsque l'on appui sur le bouton « commande » il y a écrit « ONLINE »
Else
MSComm3.PortOpen = False
Timer1.Enabled = False
Command1.Caption = "START"
//Lorsque l'on appui sur le bouton « commande » il y a écrit « START »
End If
End Sub
Private Sub Command2_Click() 'EXIT
MSComm3.Settings = "9600,N,8,1"
//9600 bauds, pas de parité, 8 bits de données et 1 bit d'arrêt.
If MSComm3.PortOpen = True Then
Timer1.Enabled = False
```

```
MSComm3.PortOpen = False
End If
End
End Sub
Private Sub Port_Click(Index As Integer)
MSComm3.CommPort = Index + 1
End Sub

Private Sub Timer1_Timer()
// Permet d'obtenir des données du PIC
a = MSComm3.Input
// Permet de retirer des données numériques
gv = Val(Left$(a, 3))
// Permet de vérifier que les données sont valides
If Len(a) > 0 Then
// Les données sont vraies donc on peut les exploiter
DSP.Caption = a
a = 0
End If
End Sub
```

Conclusion:

Le programme à bien été exécuté car on obtient bien l'évolution des valeurs numériques sur l'hyperterminal du logiciel Visual Basic (VB6,0) lorsque l'on agit sur les Switches.

Exemple:



Ici on à réglé sur les Switches la valeur « 15 » et on à observé sur l'hyperterminal du logiciel Visual Basic (VB6,0) si on lisait la même valeur (voir ci-dessous).

NB:

Les valeurs des Swiches sont binaire:

- pour mettre un Switche à 0 il faut le mettre du côté où il y a les nombres (OFF),
- pour mettre un Switche à 1 il faut le mettre du côté où il y a le « ON ».

Ensuite la somme des Switches peut varier de 0 à 255.

Voici la composition des différentes valeurs des Switches:

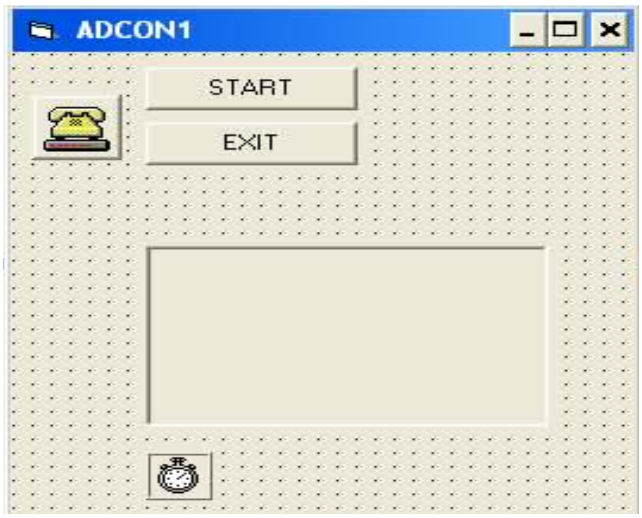
N° Switches	1	2	3	4	5	6	7	8
Puissance de 2	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
Valeur numériques	1	2	4	8	16	32	64	128

Enfin, pour obtenir la valeur « 15 » on fait la somme des Switches de façon à avoir le résultat égal à 15.

Donc on obtient en binaire: 0000 1111 (régler les Switches)
C'est-à-dire: 2^0+2^1+2^2+2^3 = 15

Remarque:

Pour visualiser l'évolution des valeurs numériques, on a besoin de placer sur la face avant (voir ci-dessous) différents éléments. En effet « le téléphone » permet de communiquer avec la liaison série RS232. Ensuite « l'horloge » va permettre de faire évoluer les valeurs à une certaine fréquence. On remarque aussi que l'on a un bouton poussoir « START » pour démarrer le programme et un autre pour le quitter « EXIT ».
Enfin nous avons placé une zone (zone creuse) pour afficher l'évolution des valeurs numériques.



<u>Variation de valeurs numériques en fonction d' une alimentation:</u>

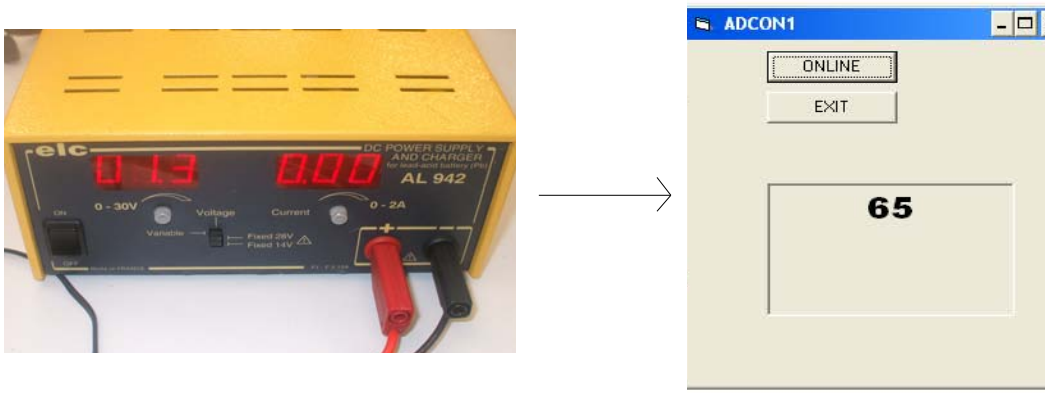
Enfin nous avons utilisé le même programme que précédemment pour faire varier les valeurs numériques à l'aide d'une alimentation. Puis d'afficher les valeurs de ce dernier sur l'hyperterminal du logiciel Visual Basic (VB6,0).

Conclusion:

Le programme à bien été exécuté car on obtient bien l'évolution des valeurs numériques sur l'hyperterminal du logiciel Visual Basic (VB6,0) lorsque l'on agit sur le bouton de l'alimentation.

Exemple:

Lorsque l'on à une tension de 1,3V, la valeur numérique que l'on lit sur l'hyperterminal du logiciel Visual Basic (VB6,0) est de « 65 » (voir ci-dessous).



Remarque:

Il faut faire varier l'alimentation entre 0 et 5 V car le circuit que l'on avait à notre disposition ne supportait pas de fortes tensions.

Développement sur la carte d'acquisition:

Visualisation du courant et de la tension:

Tout d'abord, via la liaison série RS232 nous avons programmé le micro-contrôleur (16F877) de façon à obtenir la valeur de la tension aux bornes de la batterie. Mais aussi de visualiser la valeur de l'intensité à travers la charge sur visual basic 6,0. .

Remarque:

Nous avons besoin de deux programmes pour réaliser cette fonction; un en langage c qui permet d'envoyer et de recevoir des informations au micro-contrôleur. L'autre qui permet d'afficher les valeurs de la tension et du courant via la liaison RS232.

Programme en langage C:

```
#include <16F877.h>                          //Choix du micro-contrôleur utilisé pour le projet
#fuses HS ,NOWDT ,NOPROTECT ,NOLVP           //Configuration global
#use delay (clock = 20000000)                //Configuration de l'horloge interne
#use rs232 (baud=9600 ,xmit=PIN_C6, rcv=PIN_C7)   //Configuration du port série

float tension,courant;                       // Pour avoir des valeurs à virgule (flottants)
float variable_float;

void main ()
  {
  setup_adc (ADC_CLOCK_DIV_2);       // Horloge interne / 2
  setup_adc_ports (RA0_ANALOG);      // RA0 est une entrée analogique

while (true)
  {

  set_adc_channel(0);              // Lecture sur le canal 0
  delay_ms(10);                    // Changement de valeur toutes les 10 ms pour la tension
  tension=read_adc()/21.25;        // La variable tension contient le résultat de la conversion
  set_adc_channel(1);              // Lecture sur le canal 1
  delay_ms(10);                    // Changement de valeur toutes les 10 ms pour la courant
  courant=read_adc()/85;           // La variable courant contient le résultat de la conversion
  (printf ("%f", tension));        // Envoi de données (Réel) pour la tension
  printf("\n");                    // On fait un saut de ligne
  (printf ("%f", courant));        // Envoi de données (Réel) pour la courant
  printf("\r\n");                  // On fait un retour chariot et un saut de ligne
  }
 }
```

Programme en visual basic 6,0:

```
Private gv As Integer
Private a As String
Private Sub Command1_Click()
// Permet de démarrer le programme en cliquant sur « Command1 »
If MSComm2.PortOpen = False Then
MSComm2.PortOpen = True
// Ouvre le port
Timer1.Enabled = True
Caption = "COURANT – TENSION"
// Message permettant d'afficher sur l'onglet de l'interface
Command1.Caption = "ONLINE"
// Lorsque l'on appui sur le bouton « Command1 » il y a écrit « ONLINE »
Else
MSComm2.PortOpen = False
Timer1.Enabled = False
Command1.Caption = "START"
// Lorsque l'on appui sur le bouton « Command1 » il y a écrit « START »
End If
End Sub
Private Sub Command2_Click() 'EXIT
MSComm2.Settings = "9600,N,8,1"
// 9600 bauds, pas de parité, 8 bits de données et 1 bit d'arrêt
If MSComm2.PortOpen = True Then
Timer1.Enabled = False
MSComm2.PortOpen = False
End If
End
End Sub
Private Sub Port_Click(Index As Integer)
MSComm2.CommPort = Index + 1
End Sub
Private Sub Timer1_Timer()
// Permet d'obtenir des données du PIC
a = MSComm2.Input
// Permet de retirer des données numériques
gv = Val(Left$(a, 3))
// Permet de vérifier que les données sont valides
If Len(a) > 0 Then
// Les données sont vraies donc on peut les exploiter
DSP.Caption = a
a = 0
End If
End Sub
```

Conclusion:

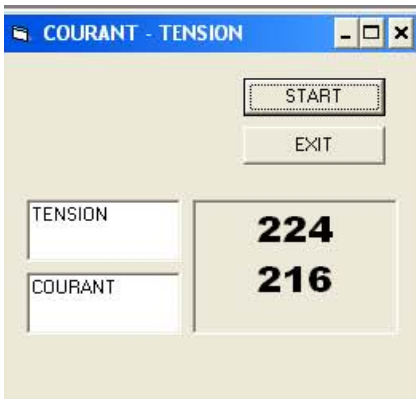
Le programme à bien été exécuté car on obtient bien l'évolution des valeurs numériques sur l'hyperterminal du logiciel Visual Basic (VB6,0) lorsque l'on agit sur le bouton de l'alimentation.

Remarque:

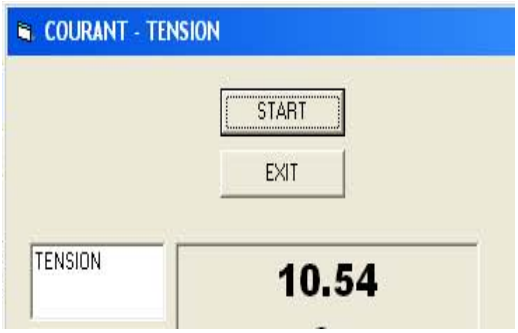
Au départ nous arrivions bien à faire évoluer les valeurs numériques mais ces dernières variaient de 0 à 255. Ceci n'était donc pas très explicite et nous avons amélioré le programme de façon à avoir des valeurs adéquats (tension de 0 à 12V et le courant de 0 à 1 A).

Exemple:

Avant l'amélioration nous avions ces valeurs (voir ci-dessous).



Après l'amélioration nous avions des valeurs pratiquement identiques à celles que l'on lisaient au niveau de l'alimentation(voir ci-dessous).



Transferts de données et visualisation de ces dernières sur l'oscilloscope:

Ensuite nous avons réalisé un programme permettant de transférer des données afin de piloter la charge. Pour vérifier cette fonction on à visualisé le résultat sur l'oscilloscope sous forme d' une PWM.

Programme en langage c:

```
#include <16F877.h>                          //Choix du micro-contrôleur utilisé pour le projet
#fuses HS ,NOWDT ,NOPROTECT ,NOLVP           //Configuration global
#use delay (clock = 20000000)                //Configuration de l'horloge interne
#use rs232 (baud=9600 ,xmit=PIN_C6, rcv=PIN_C7)   //Configuration du port série

int commande;                                //Variable déclarée comme un entier signé

float tension;
float courant;                               //Pour avoir des valeurs à virgule (flottants)
float variable_float;

void main ()
  {
  setup_ccp1 (CCP_PWM);                      //premier canal, broche RC2 PIN (17)
  setup_timer_2(T2_DIV_BY_4,128,1);          //Activation PWM Timer 2 sur ccp1
  setup_adc (ADC_CLOCK_DIV_2);               // Horloge interne / 2
  setup_adc_ports (RA0_ANALOG);              // RA0 est une entrée analogique

while (true)
  {
  set_adc_channel(0);              // Lecture sur le canal 0
  delay_ms(9);                     // Changement de valeur toutes les 9 ms pour la tension
  tension=read_adc()/21.25;        // La variable tension contient le résultat de la conversion
  set_adc_channel(1);              // Lecture sur le canal 1
  delay_ms(9);                     // Changement de valeur toutes les 9 ms pour la courant
  courant=read_adc()/85;           // La variable courant contient le résultat de la conversion
  (printf ("%f", tension));        // Envoi de données (Réel) pour la tension
  printf("\n");                    // On fait un saut de ligne
  (printf ("%f", courant));        // Envoi de données (Réel) pour la courant
  printf("\n");                    // On fait un saut de ligne
  if (kbhit())                     // Test de réception d'un message
    {
    commande=getchar();            // La variable commande est de type caractère
    }                              // Attente de la lecture d'un caractère via le port série
  set_pwm1_duty(commande);         //Valeur du rapport cyclique
    }
  }
```

Programme en visual basic 6,0:

```
Private Sub Command1_Click()
MSComm1.CommPort = 1
//9600 bauds, pas de parité, 8 bits de données et 1 bit d'arrêt.
```

```
MSComm1.Settings = "9600,N,8,1"
//Indique au contrôle qu'il doit lire la totalité du tampon si la propriété Input est utilisée.
MSComm1.InputLen = 0
//Ouvre le port.
MSComm1.PortOpen = True
Command1.Caption = "envoi"
//Lorsque l'on appui sur le bouton « commande » il y a écrit « envoi »
MSComm1.Output = Chr(99)
//Envoie la commande Attention au modem.
//MSComm1.Output = "ATV1Q0" & Chr$(13) ' Vérifie que le modem répond "99"
//Attend le retour des données vers le port série.
Dim buffer As String
MSComm1.PortOpen = False
// Permet  de retirer des données numériques
gv = Val(Left$(a, 3))
// Vérifie que les données sont valides
If Len(a) > 2 Then
// Les données sont vraies donc on peut les exploiter
gauge.Value = 128 – gv
// Jauge permettant de faire évoluer les valeurs de 0 à 128
DSP.Caption = a
a = RS232.Input
End If
End Sub
```

Conclusion:

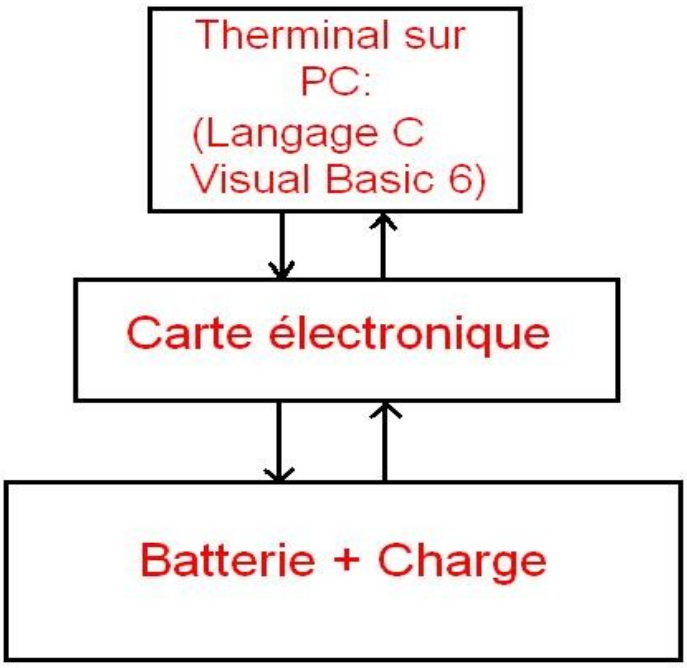
Le programme est bien envoyé dans le PIC 16F877 et on observe bien une PWM sur l'oscilloscope lorsque l'on fait varier les valeurs.

Remarque:

Nous avons eu un problème pour transférer les données à partir du VB6,0 via la liaison RS232. Donc nous avons essayé de transférer des données d'un PC à un autre grâce à une liaison RS232 inversé et on à pu comprendre le disfonctionnement.

Présentation de la carte d'acquisition permettant d'étudier la décharge d'une batterie:

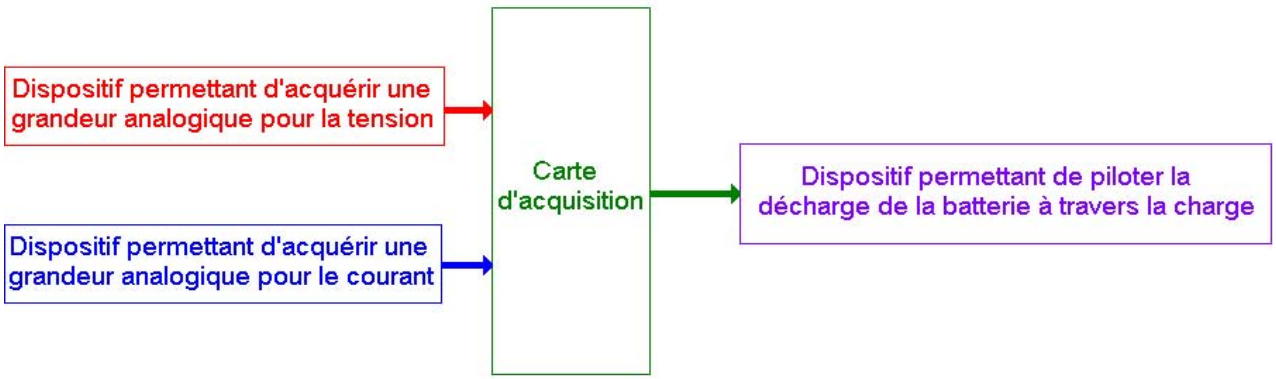
Synoptique du dispositif :



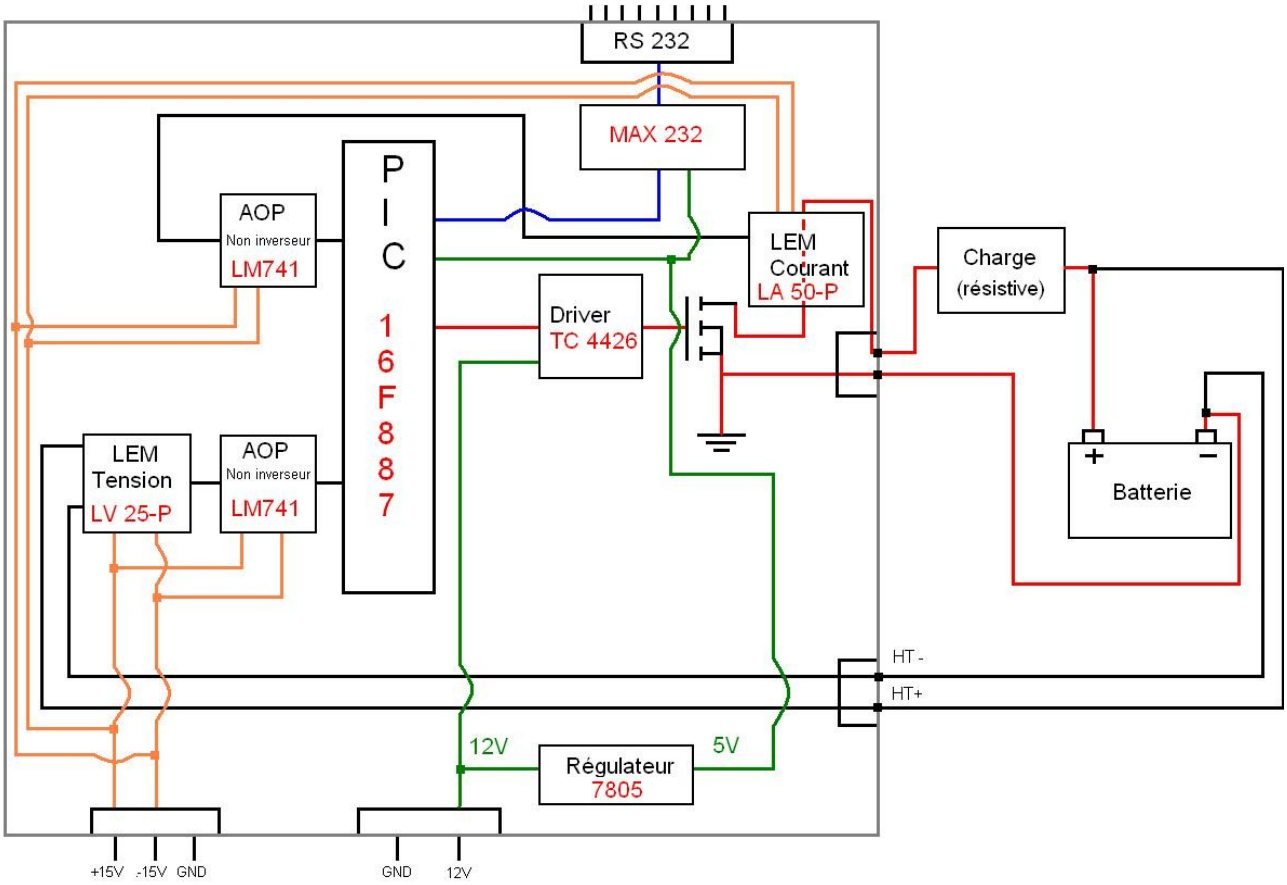
Ce dispositif permet d'étudier des cycles de décharge de la batterie à travers la charge (charge résistive fixe), pour cela on prendra un rhéostat.

Pour l'étude on va récupérer le courant et la tension délivrée par la batterie à travers la charge résistive.

Synoptique du fonctionnement:



Synoptique de la carte d'acquisition:



Fonctionnement de la carte:

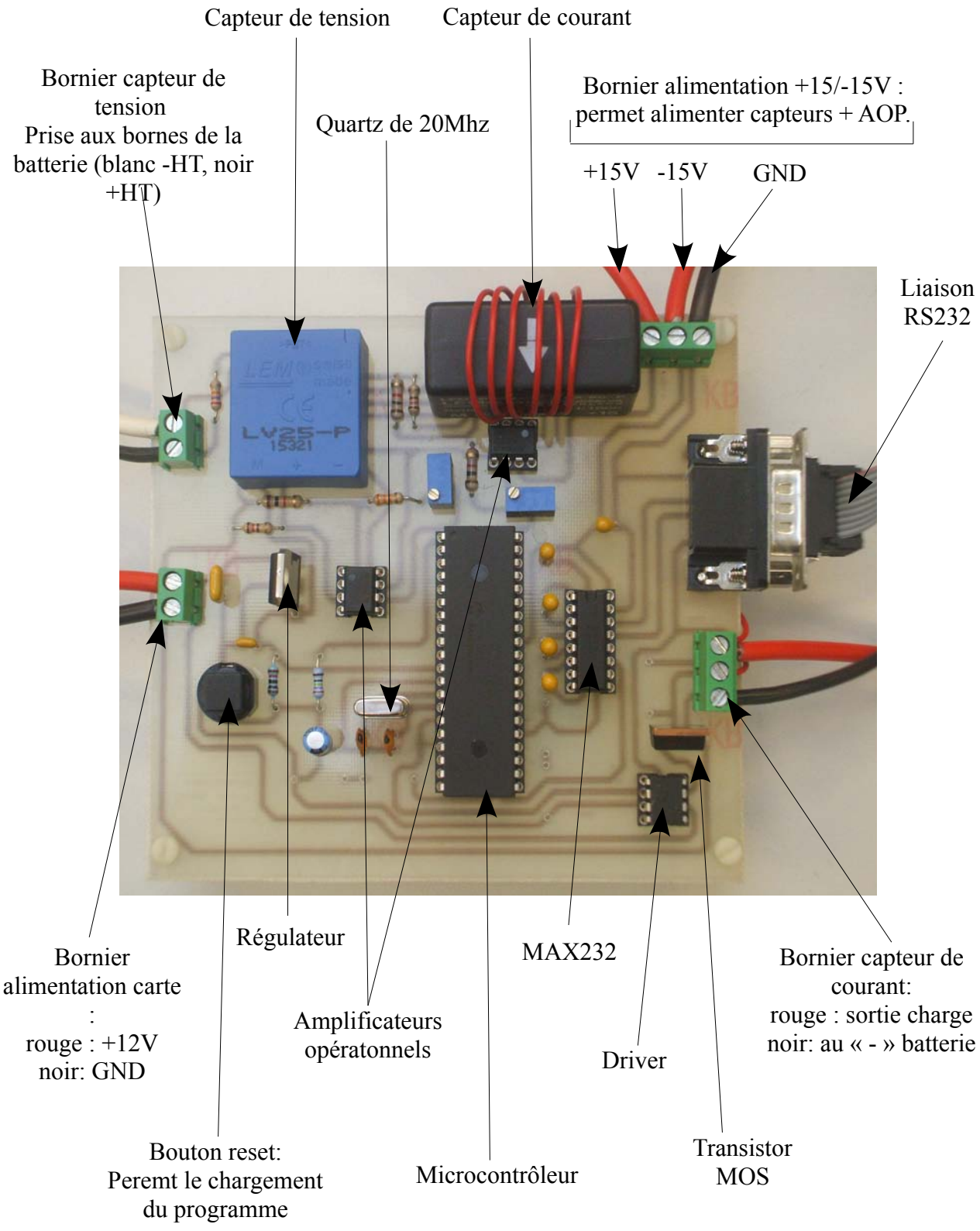
Cette carte d'acquisition permet à la fois de prendre deux grandeurs analogiques c'est-à-dire la tension et le courant délivré par la batterie (circuit en « noir »), et de piloter la décharge de la batterie à travers la résistance (circuit en « rouge »).

On peut constater sur la carte les différentes alimentations utilisées:

- Circuit représenté en « orange » permet d'alimenter les deux capteurs (Tension/Courant) et d'alimenter également les deux amplificateurs opérationnel en +15V et -15V,
- Circuit représenté en « vert » permet d'alimenter en 12V le driver et grâce au régulateur 7805 d'alimenter en 5V le micro-contrôleur et le MAX 232 (composant qui permet de communiquer entre l'interface Homme-Machine et le micro-contrôleur.

Pour la communication entre l'interface Homme-Machine et le micro-contrôleur est réalisé par une liaison série RS232 circuit en « bleu ».

Emplacement des différents composant de la carte:



Liste des composants et leur caractéristique:

- PIC 16F887 : permet de visualiser deux grandeurs analogiques (tension/courant) sur une plage de 0 à 5V et de commander la charge. La visualisation et la commande se fera sur l'interface Homme-Machine.

- LA 50-P: Capteur de courant LEM permet de lire en temps réel l'image du courant en tension lors de l'étude faite lors de la décharge de la batterie.

- LV 25-P: Capteur de tension LEM permet de lire en temps réel la tension lors de l'étude faite lors de la décharge de la batterie.

- LM741: Amplificateur Opérationnel non inverseur permet d'amplifier les grandeurs analogiques des capteurs pour respecter la plage de 0 à 5V sur les entrées analogiques du micro-contrôleur.

- 7805: Régulateur permettant de transformer la tension de 12V en une tension de 5V.

- TC 4426: Driver permet de piloter le transistor MOS c'est-à-dire que lorsque le micro-contrôleur envoie une tension sur son entrée il va plus ou moins piloter le transistor en envoyant une tension qui peut varier de 0 à 12V. Attention ce driver possède une entrée inverseuse donc par exemple, lorsque que le PIC envoie 0V sur l'entrée inverseuse, le driver pilote le transistor en envoyant une tension de 12V.

- MAX232: permet de transmettre (TX=PIN 11) de la liaison série des données au micro-contrôleur et de recevoir (RX=PIN12) du micro-contrôleur à la liaison série des données.

- Liaison RS232: permet de faire la liaison entre le MAX232 et l'interface Homme-Machine. Cet interface Homme-Machine permet de visualiser en temps réel la tension et le courant lors de la décharge de la batterie et de commander plus ou moins vite la décharge de la batterie.

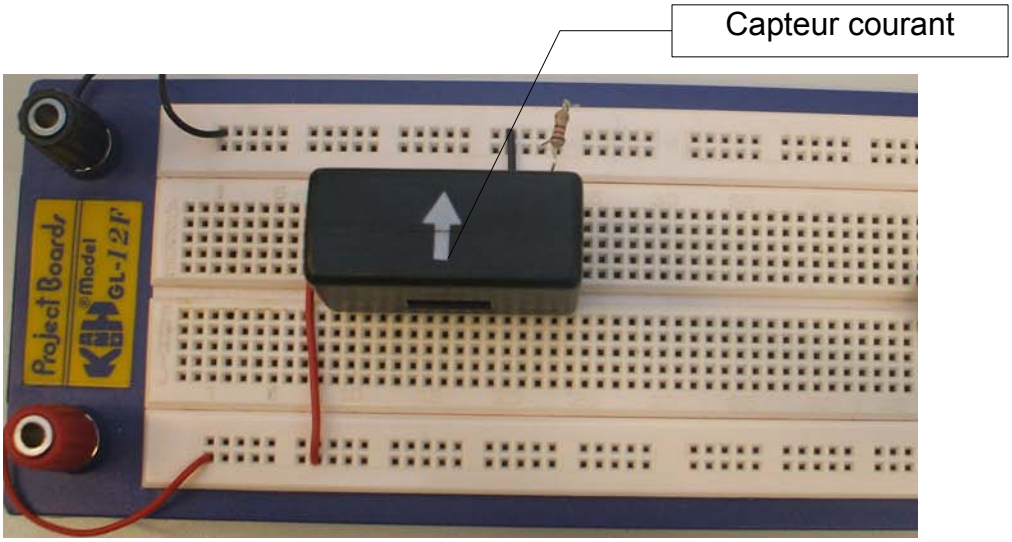
Pour plus d'information technique voir en annexe la documentation technique des différents composants.

Dimensionnement :

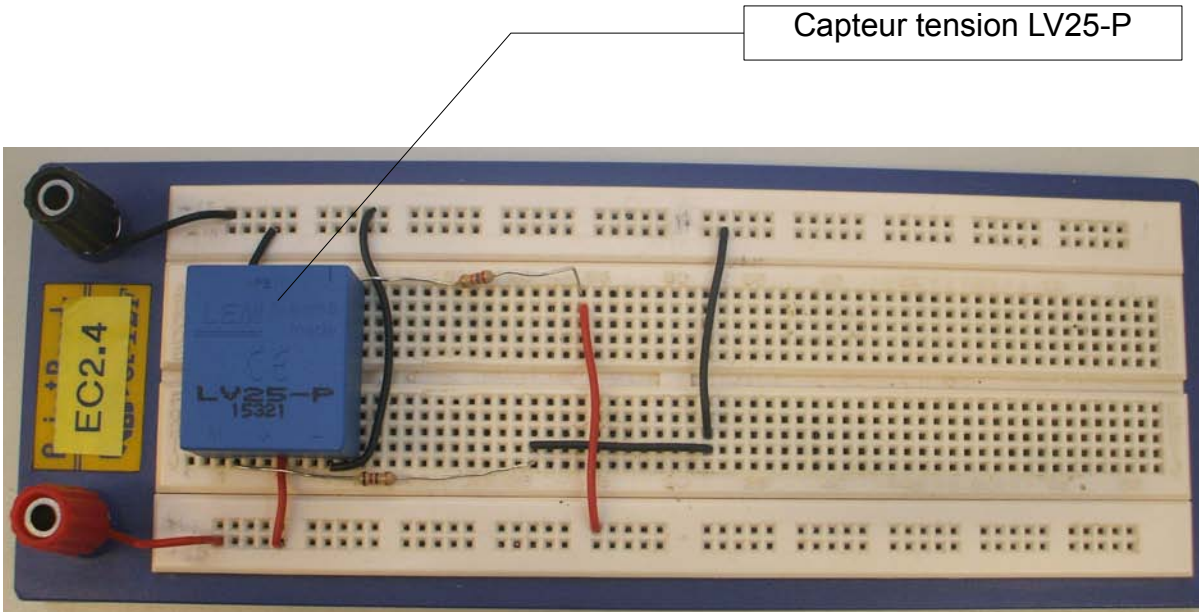
Le capteur de tension et le capteur de courant nous ont été fournis par notre professeur responsable du projet.

Donc pour respecter une plage de 0 à 5V sur le micro-contrôleur on à procédé dans un premier temps aux tests de ces deux derniers :

Implantation du capteur courant:



Implantation du capteur tension:



Tests et mesures des capteurs:

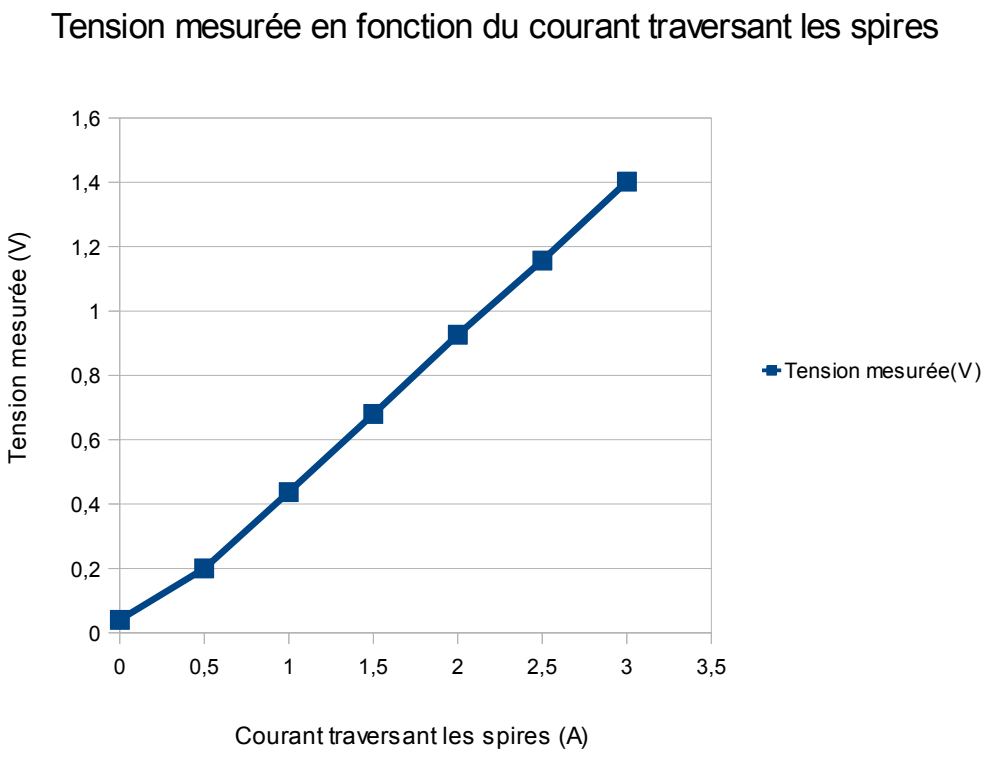
Tests et mesures du capteur de courant :

Pour le test du capteur de courant ont c'est fixé une mesure minimum de 0 A et une mesure maximum de 3 A.

Tableau de mesure:

Courant traversant les spires (A)	Mesure résistance / masse (V)
0	0,04
0,5	0,2
1	0,44
1,5	0,68
2	0,93
2,5	1,16
3	1,4

Caractéristique du capteur courant:



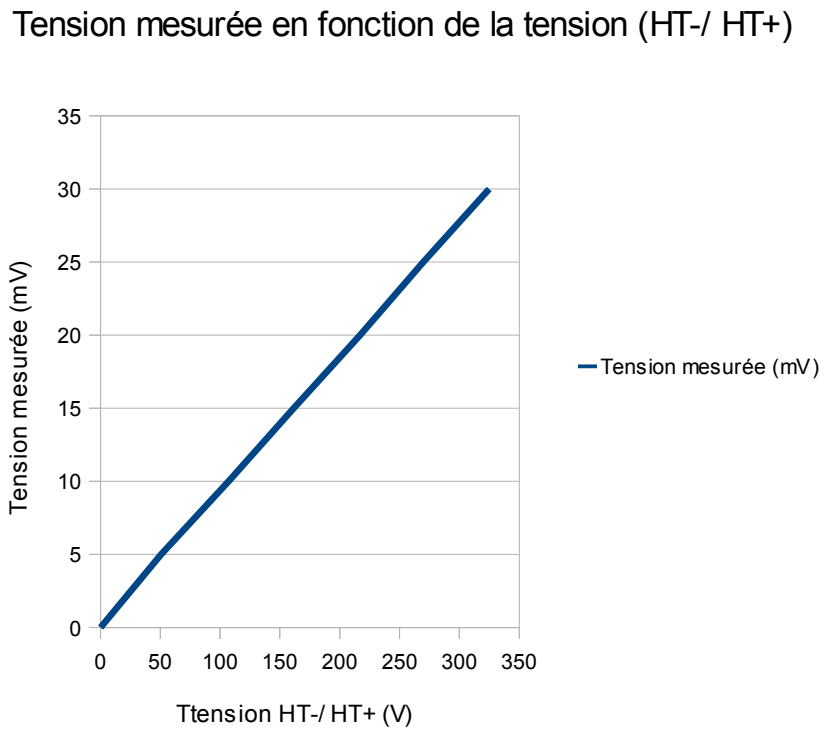
Tests et mesures du capteur de tension :

Pour le test du capteur de tension ont c'est fixé une mesure minimum de 0 V et une mesure maximum de 30V.

Tableau de mesure:

Tension HT-/ HT+ (V)	Mesure résistance / masse (mV)
0	0
5	50,7
10	107,2
15	161,6
20	217
25	270
30	325

Caractéristique du capteur tension:



Conclusion des tests:

Une fois les tests et les mesures relevées ont peut constater que les capteurs fonctionnent bien car on voit au cour de ces tests qu'il y a bien proportionnalité entre la tension mesurée et le courant traversant les spires pour le capteur de courant et qu'il y a bien proportionnalité entre la tension mesurée la tension prise aux bornes de la batterie pour le capteur de tension.

Par contre les valeurs mesurées des deux capteurs restent faible pour les exploiter:

Capteur de courant:

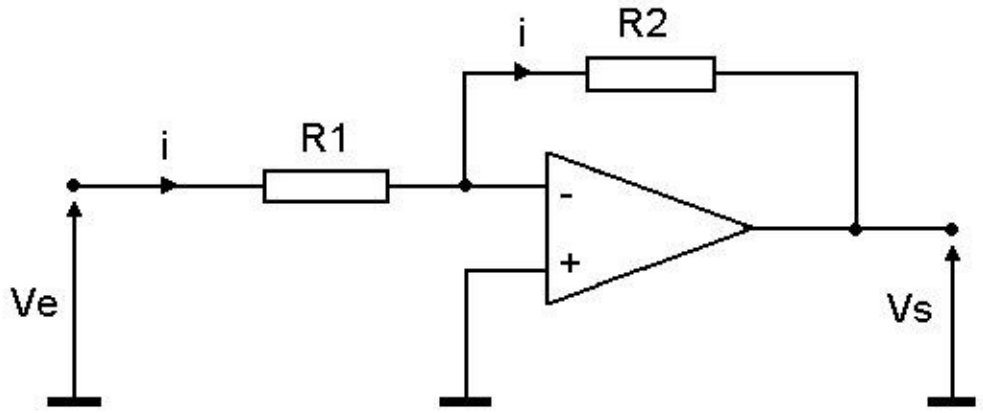
On mesure 0V pour 0A et on mesure 1,4V pour 3A, donc cela ne respect pas la plage 0 à 5V c'est-à-dire pour 0V ont mesure 0A et pour 5V on mesure 3A.

Capteur de tension:

On mesure 0V pour 0V et on mesure 130mV pour 12V, donc cela ne respect pas la plage 0 à 5V c'est-à-dire pour 0V ont mesure 0V et pour 5V on mesure 12V.

Donc dans un second temps il faut amplifier ces grandeurs analogiques pour respecter la plage 0 à 5V, donc pour cela on va pour chacun des capteurs mettre en place un montage amplificateur opérationnel non inverseurs.

Montage Amplificateur Opérationnel non inverseur:



Calculs:

<u>Pour le montage du capteur de tension:</u>

Ve min = 0mV Vs min = 0V
Ve max = 130mV Vs max = 5V
Av = Vs/Ve = 1+R2/R1 R2 = Rf+Rv (Rf = R fixe & Rv = R variable).
Av = Vs max / Ve max
Av = 5 / 0,130
Av= 38,46

Av = 1 +(R2 / R1) **Ici on prend R1 fixe à 1kohm.**

(Av -1) = R2 / 1
(Av -1) = R2
(38,46 -1) = 37,46

On sait que R2 = Rf+Rv donc on fixe **Rf à 33kohm.**
Donc:

(37,46-33) = 4,46

Donc **Rv en théorie devra être réglé à 4,46kohm.**

Pour le montage du capteur de courant:

Ve min = 40mV Vs min = 0V
Ve max = 1,402V Vs max = 5V

Av = Vs/Ve = 1+R2/R1 R2 = Rf+Rv (Rf = R fixe & Rv = R variable).

Av = Vs max / Ve max
Av = 5 / 1,402
Av= 3,56

Av = 1 +(R2 / R1) **Ici on prend R1 fixe à 1kohm.**
(Av -1) = R2 / 1
(Av -1) = R2
(3,56 -1) = 2,56

On sait que R2 = Rf+Rv donc on fixe **Rf à 1kohm.**

Donc:

(2,56-1) = 1,56

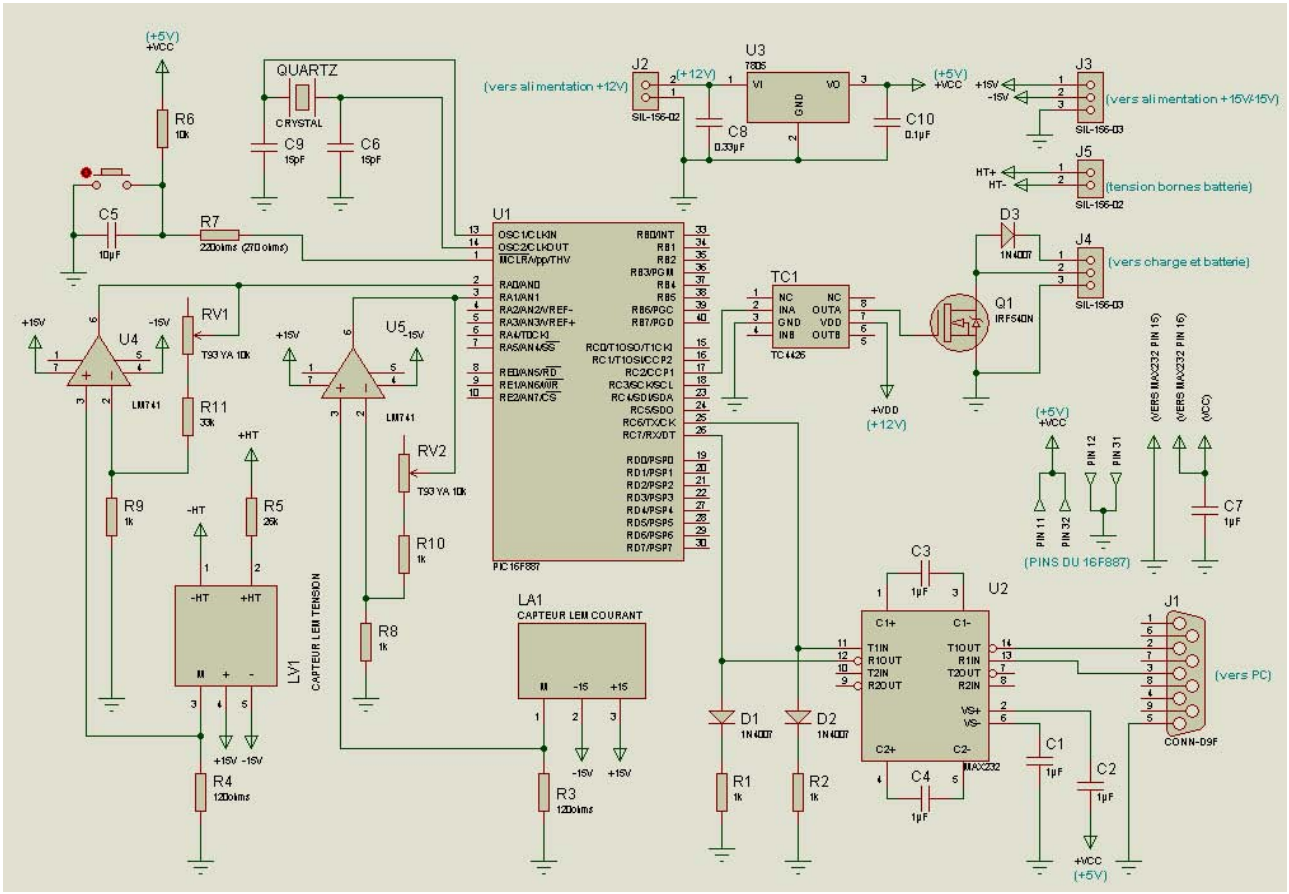
Donc **Rv en théorie devra être réglé à 1,56kohm.**

Conclusion:

Avec les calculs théoriques on respect bien les deux grandeurs analogiques que l'on souhaitent obtenir sur les deux entrées analogiques du micro-contrôleur c'est-à-dire avoir une plage de 0 à 5V.

Schématisation:

Schématisation sur ISIS:



Nomenclature:

Bill Of Materials For Projet_tutoré.DSN

Design Title	:	Projet_tutoré.DSN
Author	:	
Revision	:	
Design Created	:	jeudi 20 janvier 2011
Design Last Modified	:	jeudi 3 février 2011
Total Parts In Design	:	41

11 Resistors

Quantity:	References	Value
5	R1, R2, R8-R10	1k
2	R3, R4	120ohms
1	R5	26k
1	R6	10k
1	R7	220ohms (270 ohms)

1	R11	33k

10 Capacitors

Quantity:	References	Value
5	C1-C4, C7	1µF
1	C5	10µF
2	C6, C9	15pF
1	C8	0.33µF
1	C10	0.1µF

5 Integrated Circuits

Quantity:	References	Value
1	U1	PIC16F887
1	U2	MAX232
1	U3	7805
2	U4, U5	LM741

1 Transistors

Quantity:	References	Value
1	Q1	IRF540N

3 Diodes

Quantity:	References	Value
3	D1-D3	1N4007

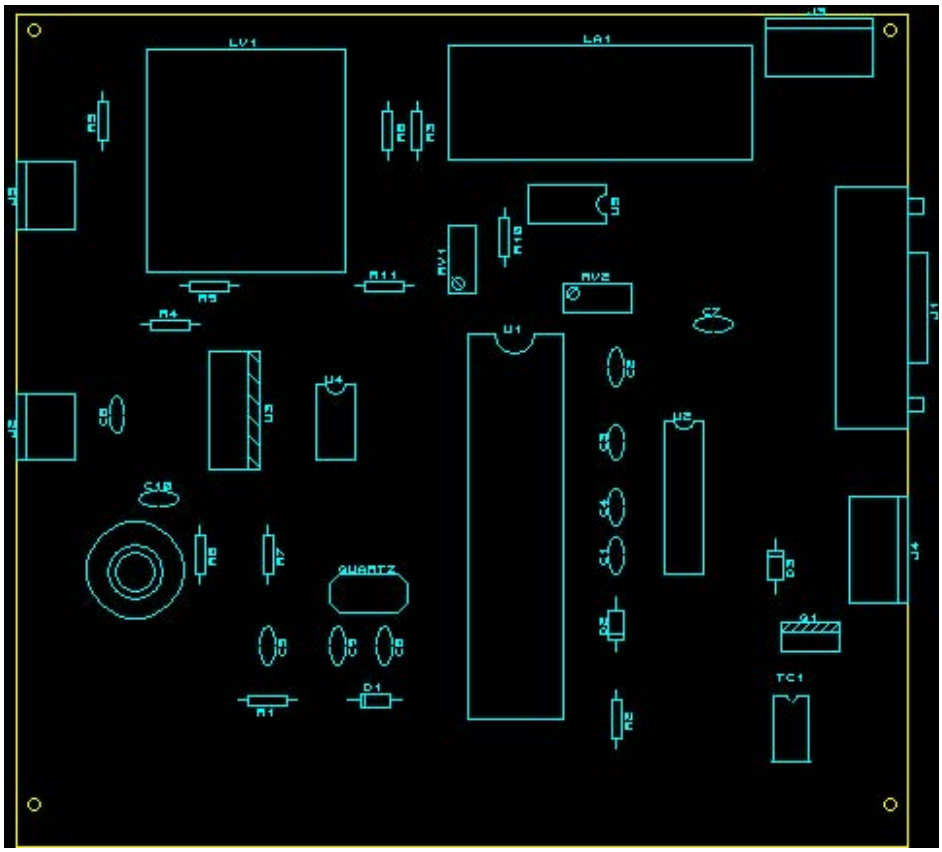
11 Miscellaneous

Quantity:	References	Value
1	J1	CONN-D9F
2	J2, J5	SIL-156-02
2	J3, J4	SIL-156-03
1	LA1	CAPTEUR LEM COURANT
1	LV1	CAPTEUR LEM TENSION
1	QUARTZ	CRYSTAL
2	RV1, RV2	T93 YA 10k
1	TC1	TC4426

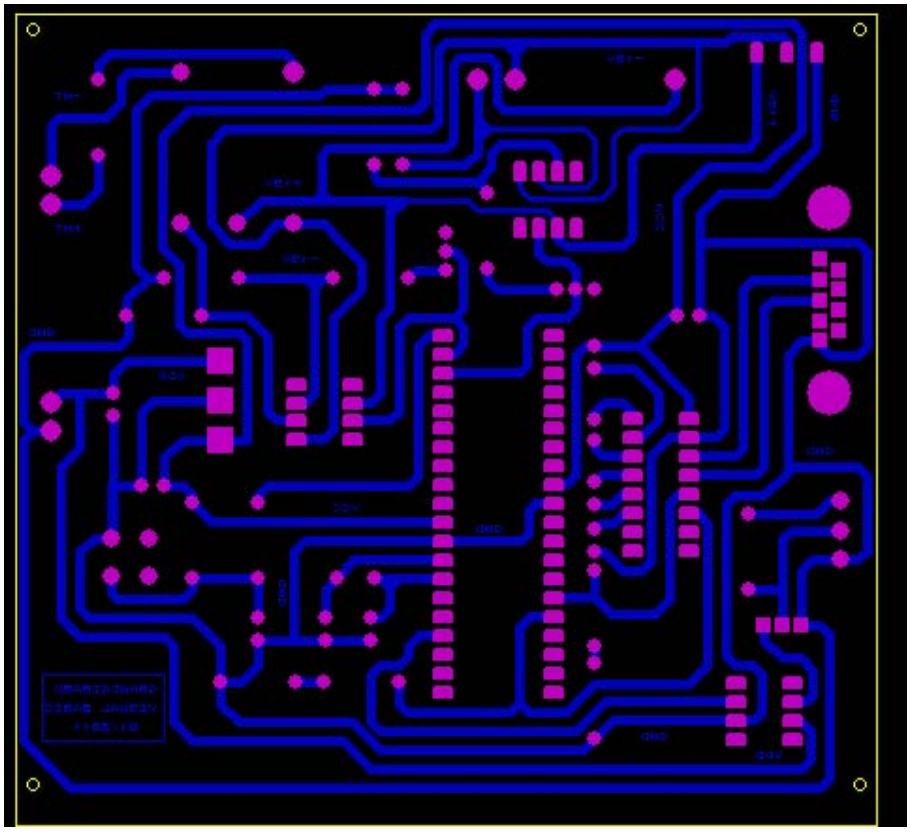
vendredi 11 mars 2011 16:26:30

Schématisations sur ARES:

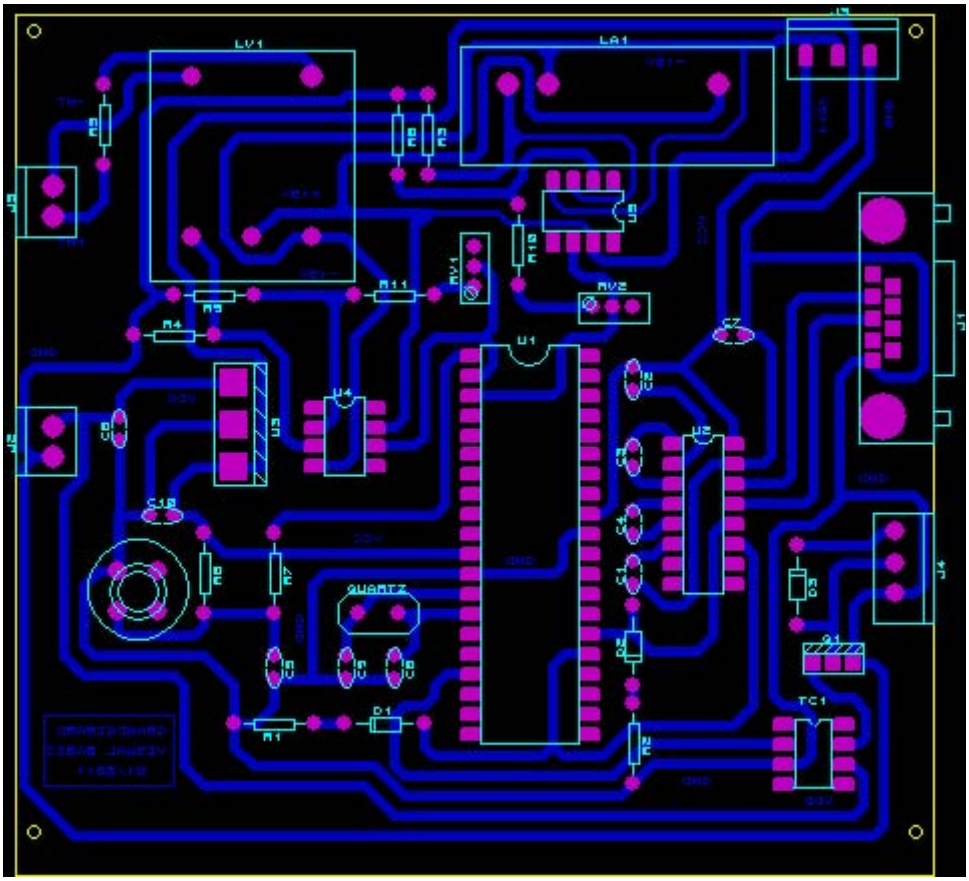
Schématisation emplacement composants:



Schématisation des pistes:



Schématisation piste + composants:



Après la réalisation de la carte d'acquisition permettant d'étudier le cycle de décharge d'une batterie, à travers une charge résistive fixe, on procède à différents test.

Tests de la carte d'acquisition:

Tests de continuité:

Tout d'abord on procède aux tests de continuité c'est-à-dire on vérifie la continuité des pistes pour voir si il n'y a pas de coupure entre les différents composants présent sur la carte.

Conclusion du test de continuité:

Toutes les pistes possèdent une très bonne continuité il n'y a pas de coupure, on peut alors passer aux tests au niveau des alimentations.

Tests des alimentations:

Ensuite on procède aux tests des différentes alimentations par précaution on retire le micro-contrôleur, les Amplificateur Opérationnel, le driver et le MAX232.

Ceux-ci étant fait ont vérifie:

- l'alimentation 12V pour alimenter le driver (PIN 6 = VDD) & le régulateur (PIN 1 = VI);
- l'alimentation 5V (sortie du régulateur PIN 3 = VO) pour alimenter le driver (PIN 2 = +VCC), le micro-contrôleur (PIN 1 = /MCLR/Vpp/THV);
- l'alimentation +15V/-15V permet d'alimenter le capteur de tension (+15V sur PIN 2 = +HT, -15V sur PIN 1 = -HT), le capteur de courant (+15V sur PIN 3 = +15, -15V sur PIN 2 = -15), les deux amplificateurs opérationnels (+15V sur PIN 7 = VCC, -15V sur PIN 4 = VEE).

Conclusion du test des alimentations:

Les différentes alimentations présentes sur la carte permettant d'alimenter les composants ont été validées.

Test de la liaison série RS232:

Pour cela on remet en place le micro-contrôleur, le MAX232, le driver et les deux amplificateurs opérationnels.

Ici nous avons essayé dans un premier temps de charger et de transférer un programme dans le micro-contrôleur, et dans un second temps récupérer les deux valeurs analogiques dans le micro-contrôleur.

Problème rencontré:

On arrive pas à transférer le programme lorsqu'on charge avec le Tiny BootLoader et que l'on appuie sur le bouton reset de la carte.

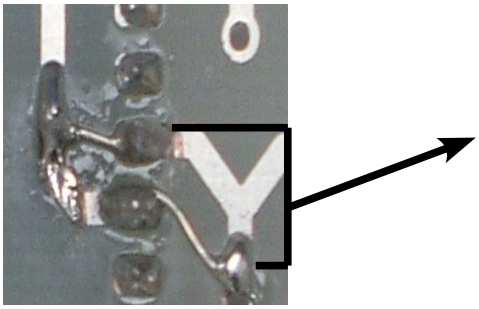
Solutions:

- Vérifier les différents paramètres permettant de transférer le programme au micro-contrôleur (vitesse de transfert, ports utilisés...);
- Vérifier complétement sur le schéma les pistes permettant la transmission des données (PIN 25 = TX au niveau du PIC, PIN 11 = TX au niveau du MAX232) et la réception des données (PIN 26 = RX au niveau du PIC, PIN 12 = RX au niveau du MAX232);
- Vérifier sur le schéma les pistes permettant de faire le reset sur la carte.

Conclusion de la liaison série RS232:

D'après les solutions proposé on a pu constater une inversion entre la transmission « TX » et la réception « RX » entre le MAX232 et le micro-contrôleur.

Photo de la modification:



Inversion des pistes de façon à obtenir une liaison « TX » et « RX » entre le PIC et le MAX232.

Test du capteur de tension et du capteur de courant:

D'après les calculs théoriques on règle la résistance variable dans chacun des montages amplificateur opérationnel c'est-à-dire on règle la résistance variable à 4,46kohms pour le montage concernant le capteur de tension. Ensuite, on règle la résistance variable à 1,56kohms pour le montage concernant le capteur de courant.

Pour le capteur de courant:

On à relevé sur l'entrée analogique (AN0) du micro-contrôleur la tension en fonction du courant débité par la batterie à travers la charge, c'est-à-dire lorsque le capteur mesure 0A ont lit 0V sur l'entrée analogique, et lorsque le capteur mesure 3A ont lit 5V sur l'entrée analogique.

Conclusion:

D'après les mesures de tension effectuées au niveau de l'entrée analogique (AN0) en fonction du courant débité par la batterie à travers la charge, on respect bien la plage 0 à 5V.

Pour le capteur de tension:

On à relevé sur l'entrée analogique (AN1) du micro-contrôleur la tension en fonction de la tension mesuré aux bornes de la batterie, c'est-à-dire lorsque le capteur mesure 0V ont lit 0V sur l'entrée analogique, et lorsque le capteur mesure 12V ont lit 5V sur l'entrée analogique.

Problème rencontré:

Lorsque on mesure la tension sur l'entrée analogique (AN1) on ne retrouve pas la plage 0 à 5V c'est-à-dire on mesure 0V pour une mesure du capteur 0V mais on mesure 2,5V pour une mesure du capteur 5V.

Solution:

- Vérifier des calculs au niveau du montage amplificateur opérationnel pour le capteur de tension.

Conclusion:

Après vérification des calculs on à pu constater qu'il y avait une erreur au niveau du dimensionnement d'une résistance. Après cette modification on respect bien cette fois-ci la plage 0 à 5V.

Photo de la modification:

Ici on à remplacé la résistance R11 de 12kohms par une résistance de 33kohms.

Tests pour commander la décharge de la batterie:

Pour cela on envoie une PWM depuis l'interface homme-Machine du visual Basic qui varie de 0% à 100% c'est-à-dire on envoie une commande permettant de décharger plus ou moins vite la batterie à travers la charge afin d'étudier son comportement.

Ici on va prendre plusieurs mesures de tension :

- Sur la sortie CCP1 = PIN 17 du micro-contrôleur;
- Sur l'entrée inverseuse INA = PIN 2 du driver;
- Sur la sortie OUTA = PIN 7 du driver;
- Sur la broche « G »(Gate) du transistor MOS.

En principe:

- lorsqu'on envoie une commande de 0% on envoie du 5V au niveau de la PIN 17 du micro-contrôleur, c'est-à-dire sur l'entrée inverseuse du driver on à 5V (PIN 2) et en sortie (PIN 7) du driver on à 0V. Donc ici on ne pilote pas le transistor c'est-à-dire on ne commande pas la décharge de la batterie.

- lorsqu'on envoie une commande de 50% on envoie du 2,5V au niveau de la PIN 17 du micro-contrôleur, c'est-à-dire sur l'entrée inverseuse du driver on à 2,5V (PIN 2) et en sortie (PIN 7) du driver on à 6V. Donc ici on pilote le transistor c'est-à-dire on commande la décharge de la batterie à 50%.

- lorsqu'on envoie une commande de 100% on envoie du 0V au niveau de la PIN 17 du micro-contrôleur, c'est-à-dire sur l'entrée inverseuse du driver on à 0V (PIN 2) et en sortie (PIN 7) du driver on à 12V. Donc ici on pilote le transistor c'est-à-dire on commande la décharge de la batterie à 100%.

Problème rencontré:

Lorsqu'on commande la PWM on à un problème au niveau de l'interface c'est-à-dire au niveau du programme en Visual Basic lorsqu'on pilote à 0% on envoie « 0 » et lorsqu'on pilote à 100% on envoie « 128 ».

Le véritable problème est lorsqu'on pilote depuis le programme en langage C on arrive à visualiser avec un oscilloscope la PWM sur le transistor MOS, mais par contre on arrive pas à visualiser ceux-ci si l'on pilote depuis l'interface du Visual Basic.

Solutions:

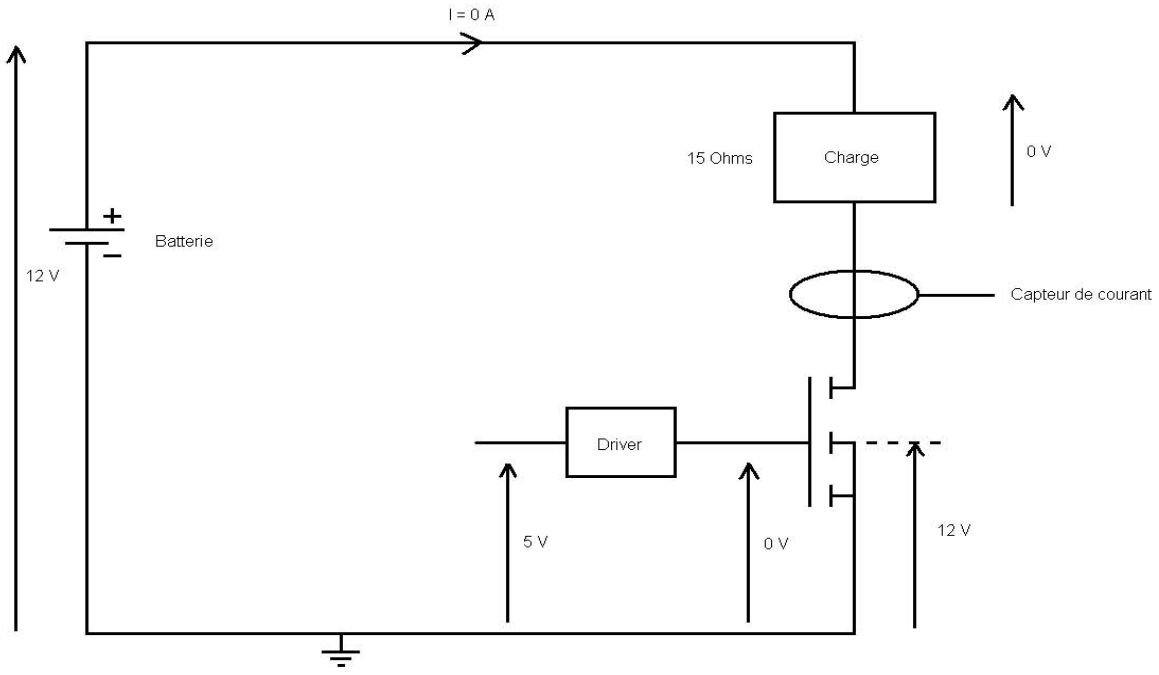
- Vérifier le programme en Visual Basic;
- Vérifier les vitesses de transfert des deux programme en C et en Visual Basic.
- Vérifier le codage c'est-à-dire soit en décimale, ou en ASCII,..

Conclusion:

Pour piloter la décharge de la batterie à travers la charge depuis l'interface du Visual Basic, il fallait trouver une fonction permettant de transformer le codage ASCII en décimal . Après cette modification apportée on peut piloter la décharge de la batterie à travers la charge depuis l'interface.

<u>Relever des tests pour une commande de 0%, de 50% et de 100%:</u>

Pour une commande de la PWM à 0%:

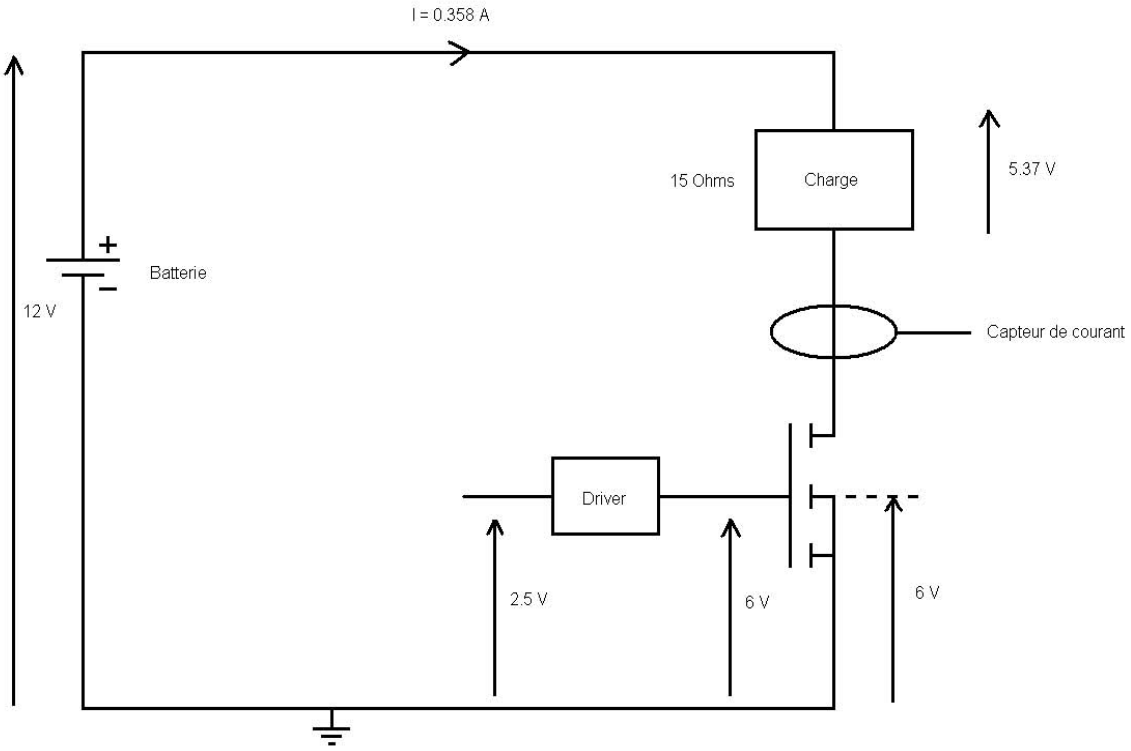


Ici on ne pilote pas la décharge de la batterie à travers la charge car aux bornes de la charge on à une tension de 0V donc avec U=RI on à:

I=U/R R=charge fixe à 15ohms pour nos test.
I=0/15
I=0A

On à 0A donc pas de décharge de la batterie.

Pour une commande de la PWM à 50%:

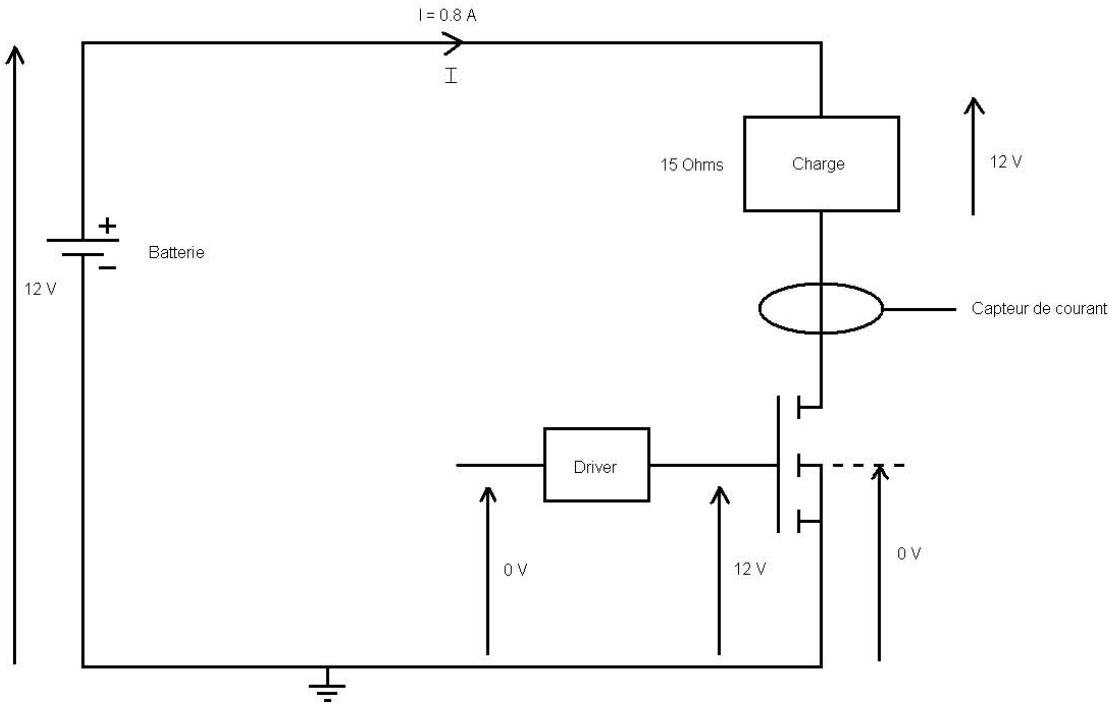


Ici on pilote la décharge de la batterie à travers la charge à 50% car aux bornes de la charge on à une tension de 5,37V donc avec U=RI on à:

I=U/R R=charge fixe à 15ohms pour nos test.
I=5,37/15
I=0,358A

On à 0,358A donc on à une de décharge de la batterie de 50%.

Pour une commande de la PWM à 100%:



Ici on pilote la décharge de la batterie à travers la charge à 100% car aux bornes de la charge on à une tension de 12V donc avec U=RI on à:

I=U/R R=charge fixe à 15ohms pour nos test.
I=12/15
I=0,8A

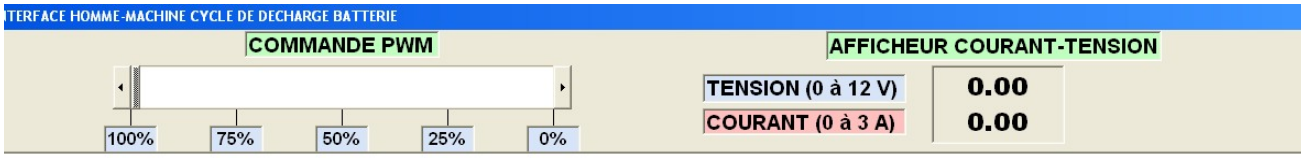
On à 0,8A donc on à une de décharge de la batterie de 100%.

Enfin après avoir validé les tests et les programmes développés sur la carte d'acquisition, nous avons réfléchi sur l'aspect ergonomique mais aussi pratique au niveau de l'interface Homme-Machine.

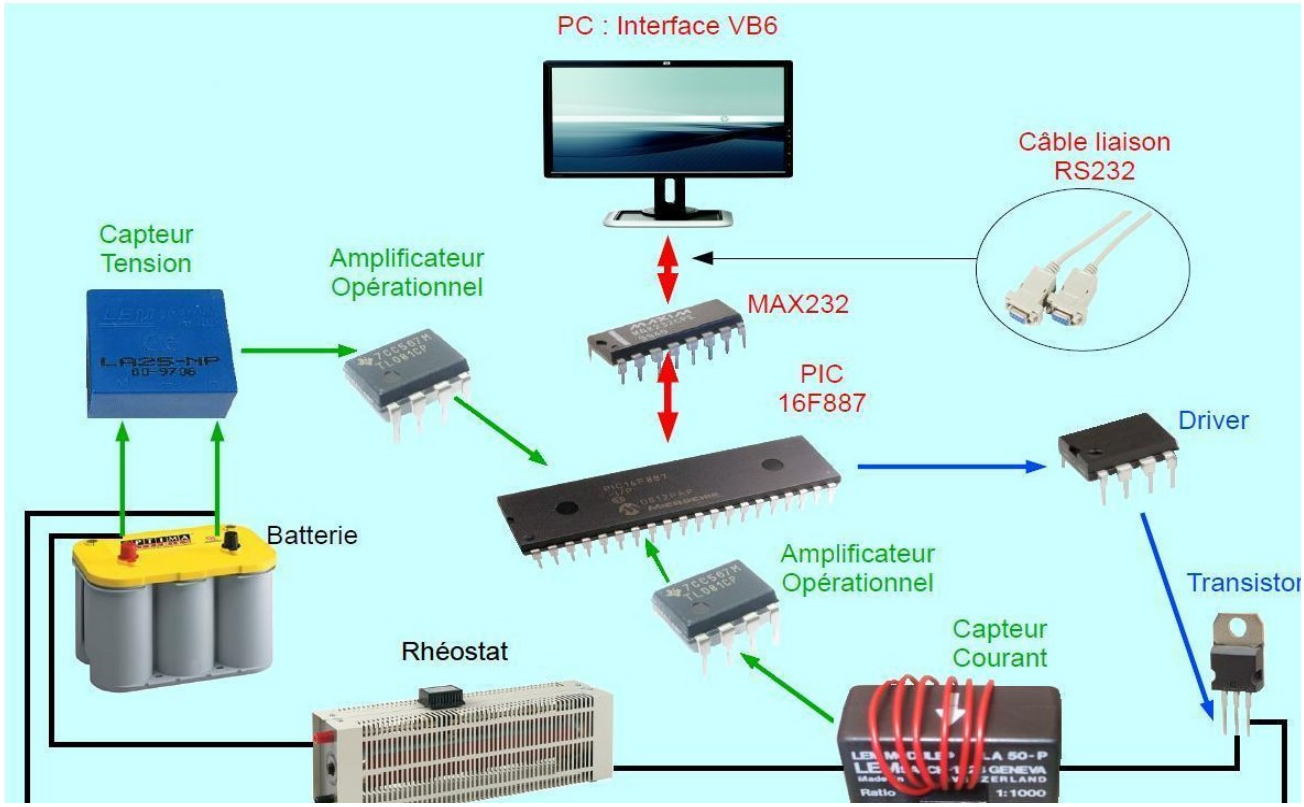
Optimisation de l'interface Homme-Machine:

Dans un premier temps, nous avons pensé qu'il était nécessaire de réaliser une jauge pour piloter la charge au lieu de retaper à chaque fois une donnée. Dans un second temps, pour visualiser en temps réel l'évolution de la tension et du courant nous avons crée un afficheur. Puis dans un troisième temps, nous avons réalisé un synoptique de l'ensemble du produit afin de voir le fonctionnement de ce dernier (voir ci-après).

Interface « HOMME-MACHINE CYCLE DE DECHARGE BATTERIE » final:



Synoptique: étude de cycle de décharge d'une batterie



Enfin pour clôturer le projet nous avons eu l'intention de rendre disponible le produit de manière simple. C'est-à-dire pouvoir l'utilisé à tout moment et sur n'importe quel ordinateur. Pour cela on à réalisé un exécutable pour que l'utilisateur travail de manière efficace et rapide avec le produit.

Conclusion:

Tout d'abord, nous avons pris du plaisir à réaliser ce projet et nous sommes content qu'il puisse fonctionner correctement comme il était prévu dans le cahier des charges. Ce projet technique nous à également permis de constater qu'il n'était pas facile de résoudre les problèmes que nous avons rencontrés.

Ensuite, la réalisation de notre projet à fait intervenir de la conception et de la gestion de plusieurs paramètres. Il nous à aussi permis d'utiliser nos bases en électronique et en informatique apprises lors de notre formation.

Enfin, nous avons pu voir un nouveau logiciel « Visual Basic » (VB6,0) qui nous permis de réaliser certains objectifs au niveau de la programmation.

Remerciements:

Nous tenons à remercier les personnes sans qui ce travail n'aurait pu aboutir:

- Mr CHRETIEN, professeur en automatisme et responsable de ce projet, pour ses informations pertinentes et ses judicieux conseils;

- Mr BURGUNDER, informaticien, pour ses multiples interventions sur l'ordinateur afin d'installer le logiciel « Visual Basic » (VB6,0);

- Ainsi qu'à toutes les personnes, professeurs et élèves ayant contribués à l'élaboration de ce projet.

Annexe

Bibliographies:

- www.fairchildsemi.com
- www.irf.com
- www.motorola.com
- www.lem.com
- www.microchip.com
- www.ti.com
- www.scienceprog.com
- www.vishay.com
- documentations VEGA

Programmes définitif du produit :

Programme en Visual Basic 6.0:

Création d'un jauge:

Afin de rendre le produit plus pratique pour l'utilisateur, nous avons réalisé une jauge « Scrollbar ». En effet cette fonction permet très facilement de faire varier la charge.

```
Private gv As Integer
Private a As String
Private Sub Hscroll1_Change()
// Permet de faire évoluer des valeurs
If MSComm2.PortOpen = False Then
MSComm2.PortOpen = True
MSComm2.Output = Chr(HScroll1)
// Permet d'envoyer sur la liaison série les valeurs du scrollbar
MsgBox HScroll1.Value / 1
Timer1.Enabled = True
Caption = "INTERFACE HOMME-MACHINE CYCLE DE DECHARGE BATTERIE"
// Message permettant d'afficher sur l'onglet de l'interface
Else
MSComm2.PortOpen = False
Timer1.Enabled = False
End If
End Sub
Private Sub Timer1_Timer()
// Permet d'obtenir des données du PIC
a = MSComm2.Input
// Permet de retirer des données numériques
gv = Val(Left$(a, 3))
// Permet de vérifier que les données sont valides
If Len(a) > 0 Then
// Les données sont vraies donc on peut les exploiter
DSP.Caption = a
a = 0
End If
End Sub
```

Programme en langage c:

```
#fuses HS ,NOWDT ,NOPROTECT ,NOLVP              //Configuration global
#use delay (clock = 20000000)                   //Configuration de l'horloge interne
#use rs232 (baud=9600 ,xmit=PIN_C6, rcv=PIN_C7) //Configuration du port série

int commande;                                   //Variable déclarée comme un entier signé

float tension;
float courant;                                  //Pour avoir des valeurs à virgule (flottants)
float variable_float;
```

```
void main ()
  {
  setup_ccp1 (CCP_PWM);                    //premier canal, broche RC2 PIN (17)
  setup_timer_2(T2_DIV_BY_4,128,1);        //Activation PWM Timer 2 sur ccp1
  setup_adc (ADC_CLOCK_DIV_2);             // Horloge interne / 2
  setup_adc_ports (RA0_ANALOG);            // RA0 est une entrée analogique

while (true)
  {
  set_adc_channel(0);              // Lecture sur le canal 0
  delay_ms(9);                     // Changement de valeur toutes les 9 ms pour la tension
  tension=read_adc()/21.25;        // La variable tension contient le résultat de la conversion
  set_adc_channel(1);              // Lecture sur le canal 1
  delay_ms(9);                     // Changement de valeur toutes les 9 ms pour la courant
  courant=read_adc()/85;           // La variable courant contient le résultat de la conversion
  (printf ("%f", tension));        // Envoi de données (Réel) pour la tension
  printf("\n");                    // On fait un saut de ligne
  (printf ("%f", courant));        // Envoi de données (Réel) pour la courant
  printf("\n");                    // On fait un saut de ligne
  if (kbhit())                     // Test de réception d'un message
    {
    commande=getchar();            // La variable commande est de type caractère
    }                              // Attente de la lecture d'un caractère via le port série
  set_pwm1_duty(commande);         //Valeur du rapport cyclique
    }
  }
```