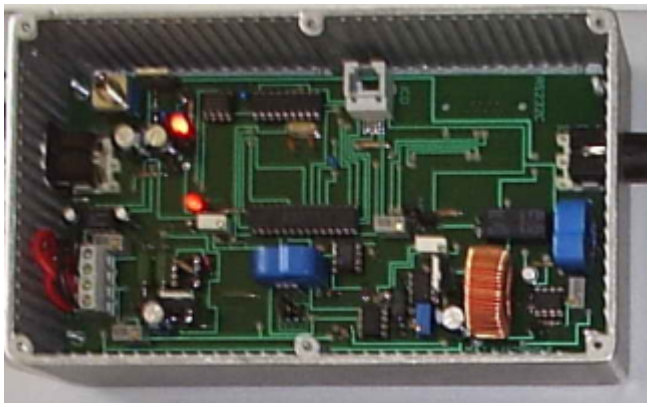
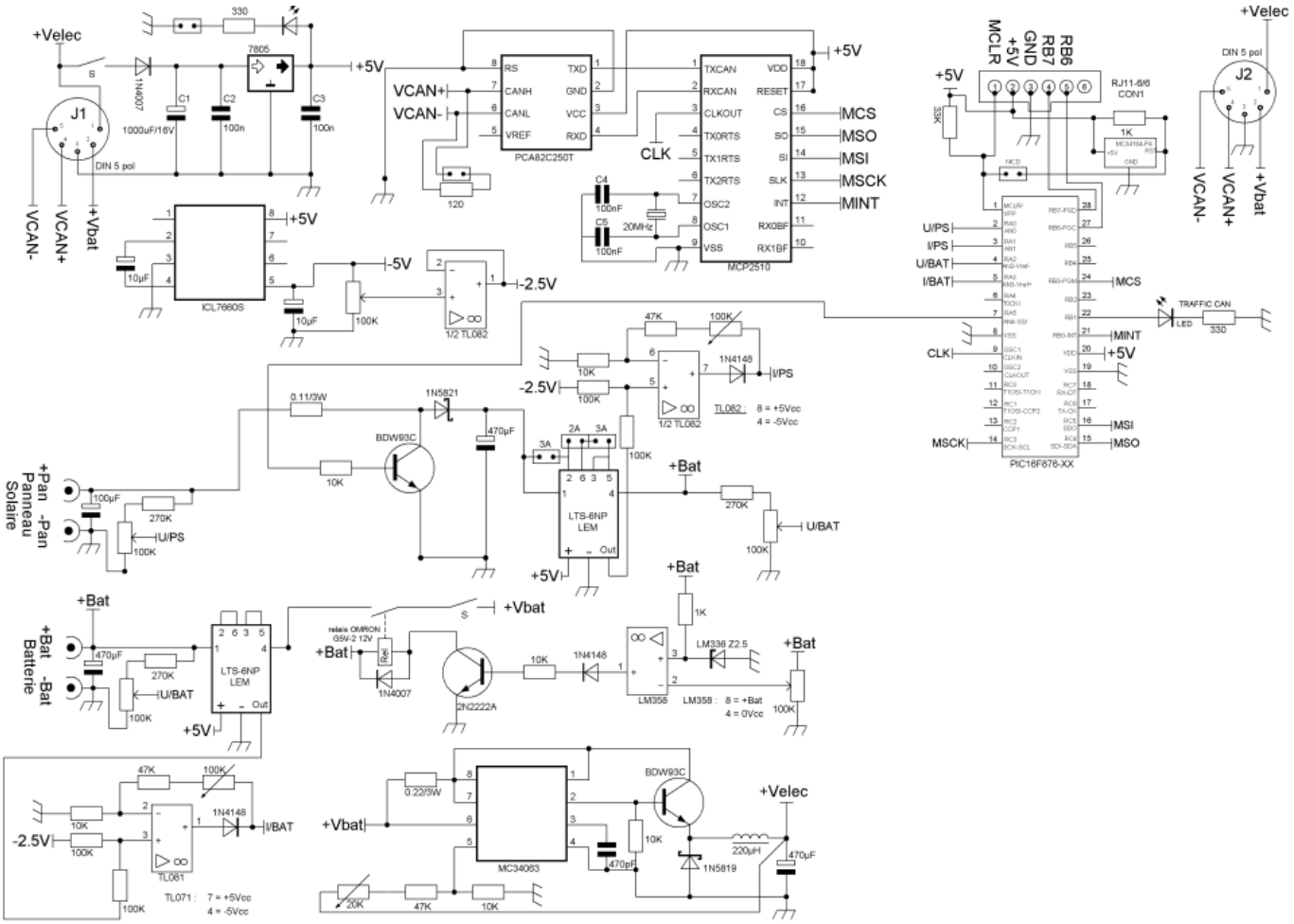
Présentation du module énergie.

Le module énergie permet, à partir du panneau solaire, de charger la batterie et de fournir l'énergie nécessaire aux différents modules du système.

Le module énergie dialogue avec le module de contrôle au travers du bus CAN.



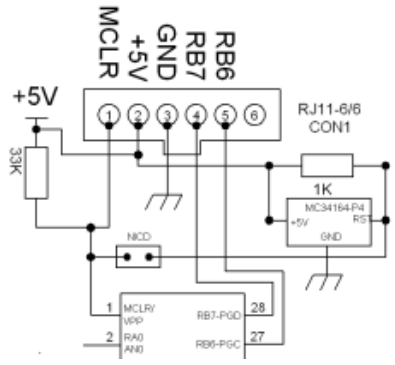
On donne le schéma structurel de ce module :



Gestion de la carte.

Le pilotage de la carte est confié à un microcontrôleur PIC16F876. La programmation in situ du PIC est assurée depuis le connecteur RJ11. Elle utilise les broches RB6 et RB7 du PIC. La broche MCLR permet, en mode programmation d'appliquer la tension de programmation.

L'horloge du processeur est fournie par la sortie SLK du circuit MCP2510 qui fournit un signal de fréquence 2,5 MHz (la fréquence du quartz divisée par 8).



La broche MCLR permet également d'assurer un RESET matériel.

Le superviseur d'alimentation MC34164 est activé en mode normal par la mise en place d'un cavalier sur le connecteur NICD. Il permet de produire un RESET matériel dans le cas ou la tension d'alimentation devient trop faible.

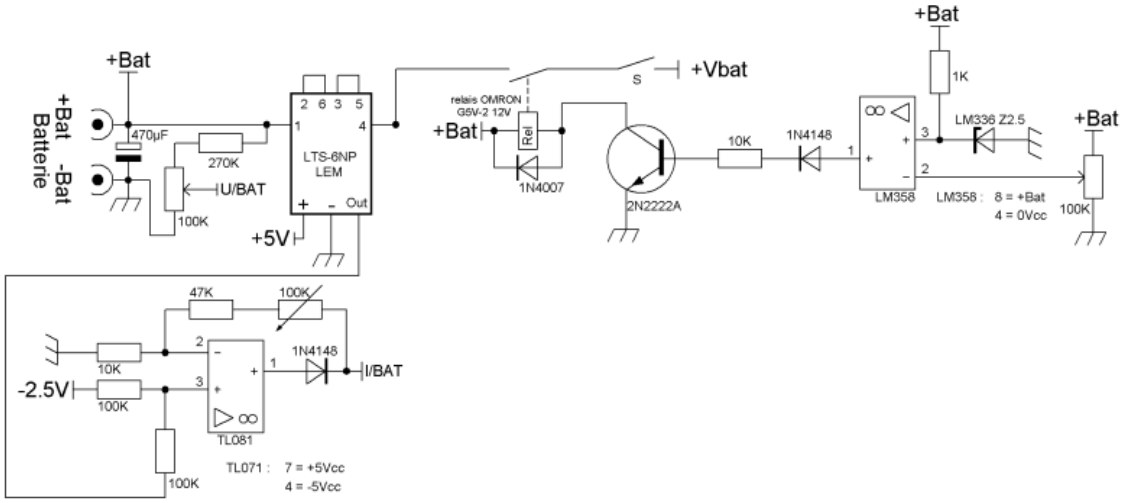
Lors de la programmation ou la mise au point d'un programme, ce cavalier est à enlevé. Le microcontrôleur fonctionne à une fréquence de 4 MHz obtenue grâce à un quartz.

Alimentation du système à partir de la batterie.

L'alimentation de l'ensemble du système est obtenue depuis la batterie par la tension +Bat. Cette tension est mesurée par le PIC après division avec le signal *U/BAT*.

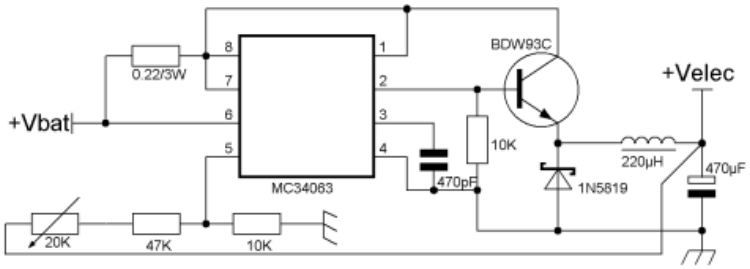
Le courant débité par la batterie est mesuré par le PIC grâce au signal *I/BAT* obtenu après amplification de la tension délivrée par le capteur LTS-6NP.

La tension d'alimentation *+Vbat* du bus système est égale à la tension *+Bat*. Mais quand la tension +Bat devient inférieur au seuil de décharge profonde de la batterie, il convient de ne plus alimenter le système. Le comparateur LM358 compare la tension +Bat à une tension de référence et n'alimente le relais que si *+Bat* est suffisante.

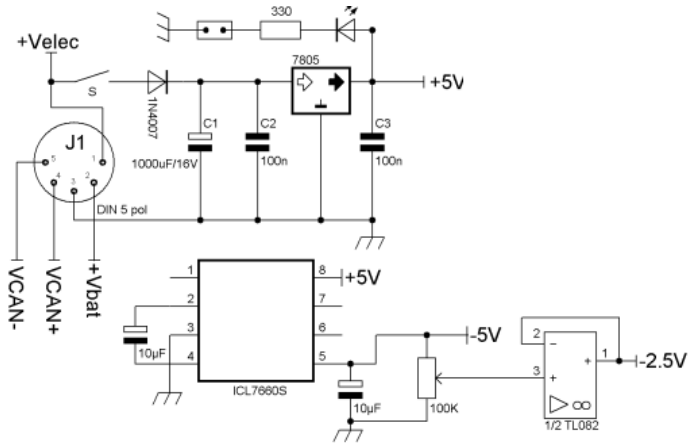


La tension *+Velec* du bus système est obtenue grâce à un convertisseur abaisseur à découpage depuis la tension *+Vbat*.

En effet, les différents modules comportant des régulateurs linéaires pour fournir la tension de +5V nécessaires, il est intéressant de fournir aux modules une tension de l'ordre de +8V plutôt que 13V afin de réduire les pertes inhérentes à la régulation linéaire.



Présentation du module énergie.



La tension *+Velec* de 8V permet grâce au régulateur 7805 de fournir la tension de 5V nécessaire aux différents circuits de la carte.

Une tension de -2,5V nécessaire à la mesure des courants est obtenue par les circuits ICL7660S et TL082.

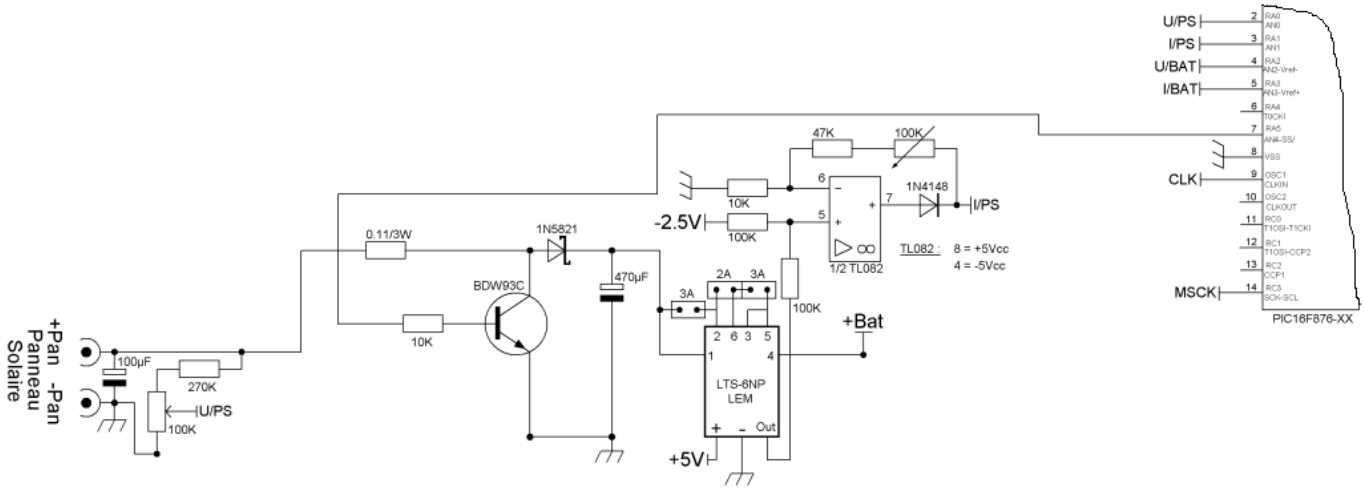
La LED indicateur de présence tension peut être désactivée par le cavalier prévu à cet effet afin de diminuer la consommation.

Un interrupteur S permet de mettre la majorité des circuits de la carte hors tension. Cette interrupteur peut être omis sur certaines versions (on a alors des « straps » et le module et continuellement sous tension).

Charge de la batterie depuis le panneau solaire.

La tension fournie par le panneau solaire est mesurée par le PIC grâce au signal U/PS. La batterie est chargée au travers de la résistance de 0,11 ohms, de la diode faible seuil 1N5821 et du capteur LTS-6NP (impédance interne très faible).
La batterie ne se charge que si la tension délivrée par le panneau est supérieure à celle disponible à ses bornes.

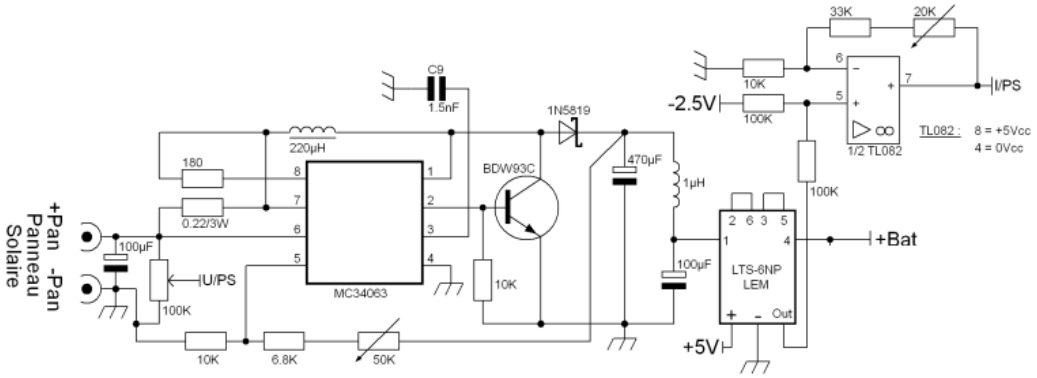
Le courant débité par le panneau est mesuré par le PIC grâce au signal *I/PS* obtenu après amplification de la tension délivrée par le capteur LTS-6NP.



La sortie RA5 du PIC va faire conduire le transistor BDW93C si la tension de la batterie dépasse 13,7V afin de limiter sa charge. Cela ne pose pas de problème de limitation de courant car l'impédance de sortie du panneau est assez élevée.

Attention, sur la version éducation cette commande est désactivée, afin que l'on puisse utiliser une alimentation continue en lieu et place du panneau. Il faudra, dans ce cas, s'assurer que la tension de l'alimentation reste inférieure à 13,7V afin de préserver la batterie.

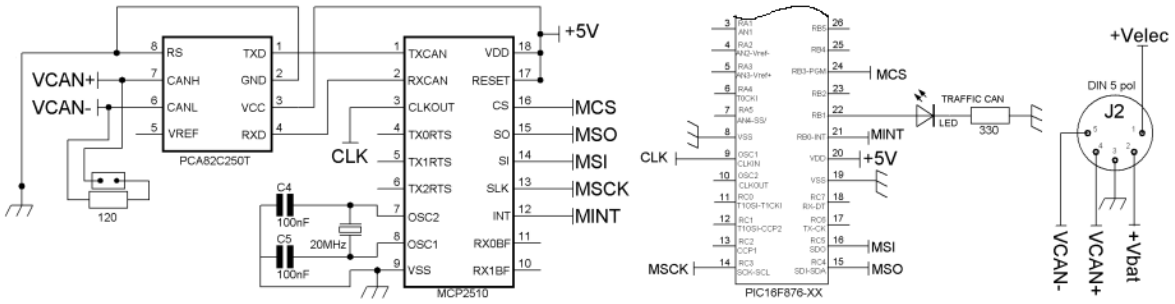
Remarque : Dans les systèmes utilisant un panneau solaire de puissance plus importante, on peut remplacer le circuit de charge de la batterie par le sous-ensemble suivant (le circuit imprimé est prévu également pour cette configuration).



Un élévateur de tension produit dans ce cas une tension de sortie +Bat limitée à 13,7V. La tension issue du panneau n'a pas besoin d'être supérieure à celle de la batterie pour assurer la charge.

Accès au bus CAN.

Pour l'accès au bus CAN, on utilise les éléments suivants :



Le circuit MCP2510 dialogue avec le microprocesseur de la carte à travers les liaisons suivantes :

- Un bus série constitué de MSO, MSI et l'horloge MSCK.
- Un fils de sélection de boîtier MCS.
- La sortie MINT peut être utilisé pour générer des interruptions pour le microprocesseur (non activé dans notre cas).

La liaison du circuit MCP2510 avec le bus CAN s'effectue grâce aux broches TXCAN et RXCAN (niveau TTL) et le circuit d'interface PCA82C250T qui produit les signaux du bus. Le bus utilise les signaux VCAN+, VCAN- et GND.

Le circuit MCP2510 nécessite pour rythmer les signaux du bus CAN d'un quartz fixé ici à 20 MHz. La broche CLKOUT produit un signal de 2,5 MHz (division par 8 de la fréquence du quartz) qui est utilisé pour fournir une horloge au circuit PIC.

On notera la présence de la résistance RT de 120 ohms qui permet d'assurer, si nécessaire, la terminaison du bus CAN si l'on met en place le cavalier prévu.
La LED « TRAFFIC CAN » sera commandé par le PIC pour signaler une activité sur le bus CAN.

Etude du logiciel embarqué dans le PIC.

En ce qui concerne le logiciel, on trouve tout d'abord les indications relatives à la compilation du programme pour le PIC. La conversion s'effectue sur 10 bits.

```
#include <16F876.h>
#device ICD=TRUE
#device *=16
#device ADC=10
#use delay(clock=2500000)
#fuses XT, NOPROTECT, BROWNOUT, NOWDT
#zero_ram                          //remet la ram a 0 (initialise les variables a 0)
```

Gestion du bus CAN.

Le driver logiciel du circuit MCP2510 est pris en compte par la ligne :

```
#include "can.c"                              //driver can a consulter pour bits utilises
```

L'identifiant utilisé pour ce module est déclaré :

```
/* identifiants bus can */
#define i_ren    0x301              //emission vers le module de controle
#define i_rmain  0x300              //reception depuis le module de controle
```

La commande de la LED d'activité est également déclarée :

```
#define led_on output_high(PIN_B1)
#define led_off output_low(PIN_B1)
```

Le bus CAN n'est utilisé qu'en réception par la fonction *gest_can* .

La réception est opérée par scrutation au moyen de la commande `if ( can_kbhit() )`

Si une réception est détectée, on examine l'identifiant et si c'est *i_rmain*, on envoie les données adéquates au module de contrôle avec l'identifiant *i_ren*.

La fonction *can_putd* permet de renvoyer les mesures demandées.

La fonction *gest_can* est appelée périodiquement par le programme principal.

```
void gest_can(){
   if ( can_kbhit() ){   //y a t il des donnees dans le buffer ?...
      if(can_getd(crx_id, &crxbuf[0], crx_len, rxstat)){ //...si oui lecture des donnees
         if (crx_id == i_rmain) {
            led_on;             // change la LED system
            ctxbuf[0]=en_ve;
            ctxbuf[1]=en_ie;
            ctxbuf[2]=en_is;
            ctxbuf[3]=en_vs;
            ctxbuf[4]=charge;
            ctxbuf[5]=full;
            ctxbuf[6]=empty;
            can_putd(i_ren, &ctxbuf[0], 7,1,1,0);       //reponse avec 7 octets de buffer
         }
      }
   }
}
```

Gestion de la charge.

La gestion de la charge est assurée par la fonction *gest_bat* appelée cycliquement depuis le programme principal.

On positionne les 3 indicateurs, batterie en charge, batterie vide et batterie pleine.

Si la batterie est chargée, on limite la charge.

```
void gest_bat(){
      if (en_vs > 137){
         full = 1;
         //OFF;          //limitation de la charge de la batterie
      }
      if (en_vs < 134) {
         full = 0;
         //ON;           //activation de la charge
      }
      if (en_ie > (en_is+1)){charge = 1;}
      else {charge = 0;}

      if (en_vs < 114){empty = 1;}
      else {empty = 0;}
}
```

Mesure des paramètres.

La mesure des courants et des tensions est assurée par la fonction *mesure* appelée cycliquement depuis le programme principal.

Cette fonction est activée périodiquement par le *timer* logiciel grâce à l'indicateur *mes_flag*.

On mesure les grandeurs tensions et courants que l'on place dans une table au moyen du pointeur j qui varie de 0 à 3. On lance ensuite un calcul de moyenne sur les 4 dernières valeurs pour chaque grandeur mesurée.

```
void calcul(){
   int16 somme;                    //variable pour calcul
   int i;                          //compteurs de boucle

   if (calc_flag==1){
      somme=0;
      for (i=0;i<=3;i++){somme=somme+table[i];}
      en_ve=somme/4;

      somme=0;
      for (i=4;i<=7;i++){somme=somme+table[i];}
      en_ie=somme/4;

      somme=0;
      for (i=8;i<=11;i++){somme=somme+table[i];}
      en_vs=somme/4;

      somme=0;
      for (i=12;i<=15;i++){somme=somme+table[i];}
      en_is=somme/4;

      calc_flag=0;
   }
}
```

```
void mesure(){
   int16 a;

   if (mes_flag==1){

      set_adc_channel(0);
      delay_us(100);
      a=read_adc();
      a = (a*17)/100;      //en_ve
      table[j] = a;

      set_adc_channel(1);
      delay_us(100);
      a=read_adc();
      a = (a*1)/3;       //en_ie
      table[j+4] = a;

      set_adc_channel(2);
      delay_us(100);
      a=read_adc();
      a = (a*17)/100;      //en_vs
      table[j+8] = a;

      set_adc_channel(3);
      delay_us(100);
      a=read_adc();
      a = (a*1)/3;        //en_is
      table[j+12] = a;

      j++;
      if(j>=4){j=0;}

      mes_flag=0;
      calc_flag=1;                    //indicateur pour calcul
   }
}
```

C'est l'indicateur *clac_flag* qui active la fonction *calcul*.

Le programme principal.

La déclaration des variables :

```
/* variables du systeme */
int16 en_ve;                        //tension d'entree
int16 en_vs;                        //tension de sortie
int16 en_ie;                        //courant d'entree
int16 en_is;                        //courant de sortie
int charge;                         //indicateur de charge
int full;                           //indicateur de charge complete
int empty;                          //indicateur de decharge

int j;                             //compteurs de boucle
int table[18];
```

On déclare les fonctions :

```
/* declaration des fonctions */
void init();
void gest_can();
void gest_bat();
void mesure();
void calcul();
```

On déclare les entrées/sorties :

```
#define ON output_low(PIN_A5)
#define OFF output_high(PIN_A5)
```

Reste le programme principal qui appelle le programme d'initialisation :

```
/* programme principal */
void main()
{
   init();
   while (1)
   {
      led_off;
      gest_bat();
      mesure();
      calcul();
      gest_can();
   }
}
```

Le programme d'initialisation initialise le bus CAN, la conversion analogique/numérique et le *timer*.

```
void init(){
   can_init();
   setup_adc_ports( ALL_ANALOG );
   setup_adc(ADC_CLOCK_DIV_2);
   setup_timer_2(T2_DIV_BY_16,8,5);    // parametrage du timer 2
   enable_interrupts(INT_TIMER2);      // autorisation interruption timer2
   enable_interrupts(GLOBAL);          // autorisation de toutes les interruptions (afin de prendre en compte timer2)
}
```

Gestion du temps.

Afin de ne pas rendre les fonctions bloquantes, on utilise un *timer* logiciel appelé par un *timer* matériel initialisé pour générer une interruption toutes les ms.

Les déclarations :

```
/* indicateurs des taches activees periodiquement */
int1 mes_flag;                          //indicateur pour mesure
int1 calc_flag;                         //indicateur pour calcul
int16 mst;
```

Toutes les millisecondes, la fonction ci-dessous est déclenchée :

```
/* timer des taches appele par interruption */
#int_timer2
void isr_timer2(void) {
   mst++;                               //timer qui est appele toutes les ms par une interruption du timer2
   if (mst == 500){mes_flag = 1;}
   if (mst == 500){mst = 0;}
}
```

L'indicateur *mst* sert à déclencher les mesures. Ces dernières sont activées toutes les 500 ms par l'indicateur *mes_flag*.

Dés que *mst* atteint la valeur 500 et on remet *mst* à 0.