
# Documentation Tensiomètre

2008

# Sommaire

# Table des figures

# I. Le tensiomètre

Le tensiomètre Emulsar doit permettre l'observation de gouttes d'émulsion dans le but de déterminer certaines propriétés des gouttes par un traitement informatique. L'observation des gouttes est effectuée par une caméra à microscope.

L'électronique du tensiomètre doit permettre :

- D'injecter un liquide dans une solution grâce à une seringue contrôlée par un moteur pas à pas
- De réguler la température de cette solution grâce à un module Peltier.
- Le tensiomètre est piloté par un ordinateur via le port USB. Toutes les trames USB sont reçues par la carte microcontrôleur équipée d'un C8051F340 de Silabs.

Voici un schéma du tensiomètre et de ses différents éléments.

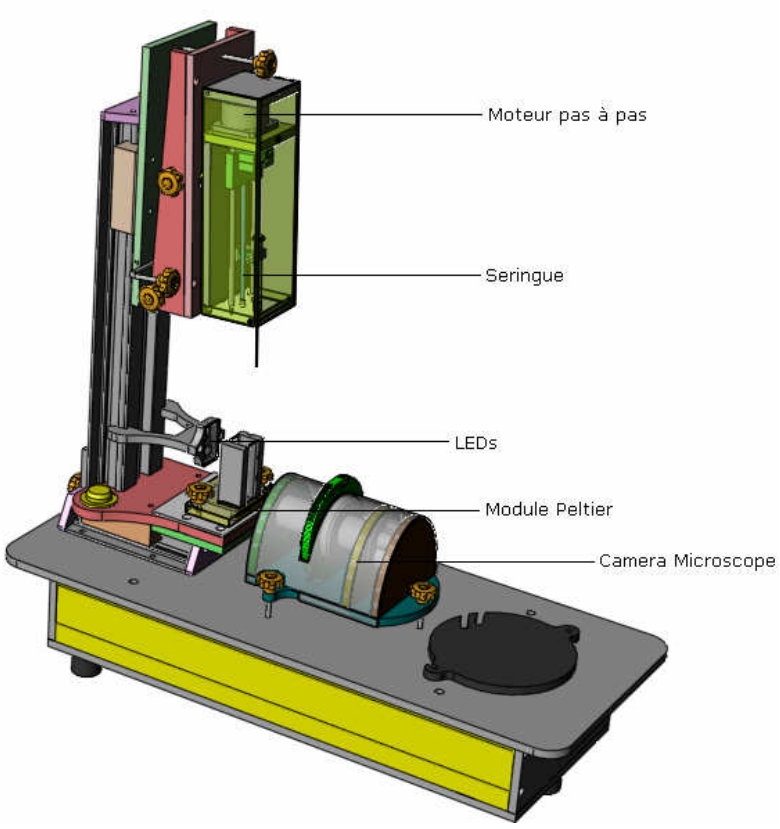



fig 1. Image du tensiomètre

Contrairement à la version de 2007, chaque fonction est réalisée par une carte électronique différente de manière à pouvoir réutiliser dans l'avenir les cartes pour des applications similaires. Dans le cas du tensiomètre, une carte est dédiée à la régulation de température par le module Peltier, une carte sert au contrôle du moteur pas à pas pour monter et descendre la seringue, une carte permet d'ajuster la luminosité des leds de manière à avoir une image exploitable et la dernière carte contient le microcontrôleur.

Voici un schéma résumant l'ensemble des cartes électroniques faisant partie du tensiomètre et les communications existantes entre ces cartes :

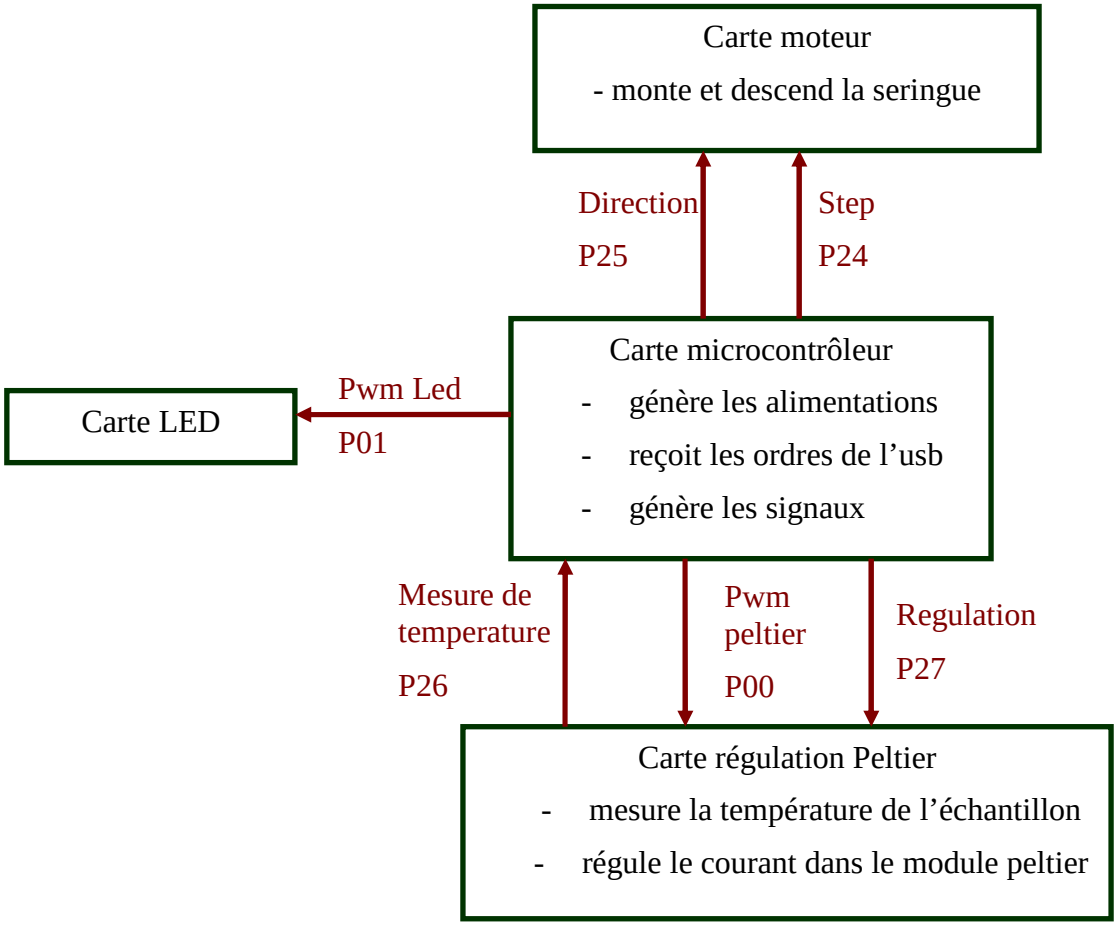



fig 2. Schéma de l'architecture électronique du tensiomètre

L'ensemble des schémas électroniques et des typons des cartes électroniques du tensiomètre sont disponibles en annexe.

# II. Le microcontrôleur

## 1. Spécifications

La carte microcontrôleur est la carte centrale du tensiomètre car elle reçoit par USB les commandes de l'utilisateur et détermine alors les ordres à envoyer sur ses ports d'entrée-sortie pour les différents modules connectés.

Le microcontrôleur utilisé est un C8051F340 de chez Silabs. C'est un microcontrôleur 48 pins qui présente l'avantage d'avoir un module USB2 intégré ce qui facilite la transmission avec le pc. Le microcontrôleur est programmé en C.

Etant donné que cette carte doit être réutilisable pour d'autres applications, il est nécessaire que l'ensemble des ports d'entrée-sortie du microcontrôleur soit accessible.

De plus, la carte doit être capable de fonctionner pour une grande plage de tension d'entrée. Enfin, c'est elle qui génère l'ensemble des alimentations des cartes auxquelles elle est connectée.

## 2. Hardware

Afin de minimiser la taille de la carte (60,5 * 80,5 $mm^2$), les ports d'entrée-sortie sont accessibles à travers un port IDE 40 pins soudé horizontalement sur le bord de la carte. Ainsi, 20 pins sont sur le top et 20 sur le bottom. Etant donné que le microcontrôleur CMS est soudé sur le top, ceci implique d'avoir 20 vias pour les ports d'entrée sortie.

La programmation du microcontrôleur se fait directement sur la carte mais l'alimentation disponible sur le connecteur de programmation n'est pas utilisée.

La carte est prévue pour être alimentée directement sur une prise secteur à travers un transformateur 12V ou 24V. Elle dispose donc d'un pont redresseur de diodes et de trois régulateurs à découpage servant à générer le 12V, le 5V et le 3,3V logique. Ces régulateurs sont des régulateurs de chez RECOM R-78x-05 ou x représente la valeur de la tension de sortie. Ces régulateurs assez onéreux présentent l'avantage d'avoir les mêmes branchements que les régulateurs linéaires LM78x équivalents. On peut donc utiliser un régulateur linéaire si la différence de tension entre entrée et sortie n'est pas trop importante et que le courant de sortie est faible.

La carte peut aussi être alimentée par l'intermédiaire de l'USB. Dans ce cas là, les seules alimentations logiques générées sont le 3,3V et le 5V. Le choix entre USB et alimentation secteur se fait à l'aide de deux cavaliers.

# III. Le module Seringue

Il permet d'injecter ou d'aspirer du liquide. L'utilisateur choisi les paramètres suivants :

- Le volume de liquide à injecter.

- Le diamètre de la seringue utilisée.

- La vitesse à laquelle le piston de la seringue doit se déplacer.

- Le nombre d'allers-retours que doit effectuer le piston.

- Le sens du piston.

Il est actionné par un moteur pas à pas.

Suivant les données qu'a choisies l'utilisateur, le microcontrôleur reçoit par le port USB les différentes variables dont il a besoin pour fonctionner :

- Le nombre de pas que le moteur doit effectuer (calculé grâce au volume à injecter et au diamètre de la seringue).

- La vitesse du piston.

- Le nombre de cycles à effectuer.

- Le sens du piston.

Le contrôle du moteur pas à pas est expliqué dans une autre documentation. Le moteur est contrôlé en micro pas à l'aide d'une driver de pont en H A3986 qui gère la génération des PWM servant au contrôle du moteur pas à pas pour des tensions allant de 12 à 50V.

# IV. Le module Peltier

## 1. Principe

Il permet de réguler la température de la solution à une valeur fixée par l'utilisateur (entre 10°C et 80°C).

Un module Peltier fonctionne par effet Peltier : si on fait passer un courant électrique dans des matériaux conducteurs de nature différentes liés pas deux jonctions, l'une des jonctions se refroidit pendant que l'autre chauffe. Un module Peltier est capable de chauffer ou de refroidir suivant le sens du courant qui le traverse.

Une sonde PT1000 est utilisée pour faire la mesure de température et un régulateur de type PID permet d'atteindre et de se stabiliser à la température de consigne fixée par l'utilisateur.

## 2. Hardware

Le module Peltier est utilisé pour le tensiomètre pour refroidir ou chauffer un échantillon liquide ce qui demande une puissance importante. La carte électronique est donc prévue pour supporter des courants allant jusqu'à 10A.

Le Peltier est commandé par une PWM générée par le microcontrôleur qui vient en entrée d'un pont en H. Etant donné qu'il est très rare d'avoir des ponts en H complets pouvant fournir plus de 3A, le pont complet est réalisé par deux drivers de demi-pont IR2104, l'un ayant en entrée la pwm directement, l'autre ayant la pwm inversée par une porte inverseuse.

Ces drivers présentent l'avantage de gérer automatiquement les temps morts pour éviter d'avoir des connections alimentation-masse. De plus, les MOS ne sont pas intégrés aux drivers, on peut donc avoir des courants très importants en sortie. Les MOS utilisés sont des MOSFET à canal N FQA32N20C qui ont une très faible résistance drain-source et donc dissipent moins de puissance. Cependant, étant donné le courant important qui traverse le module Peltier, la dissipation de chaleur demeure importante et nécessite d'avoir recours à un dissipateur placé sur les MOS.

Les modules Peltier ne sont pas faits pour être commandés par une PWM car ils s'usent progressivement avec les changements d'états rapides. C'est la raison pour laquelle une première carte a été réalisée avec des filtres LC sur chaque demi-pont pour transformer la PWM en signal continu.

Voici la structure du filtre :

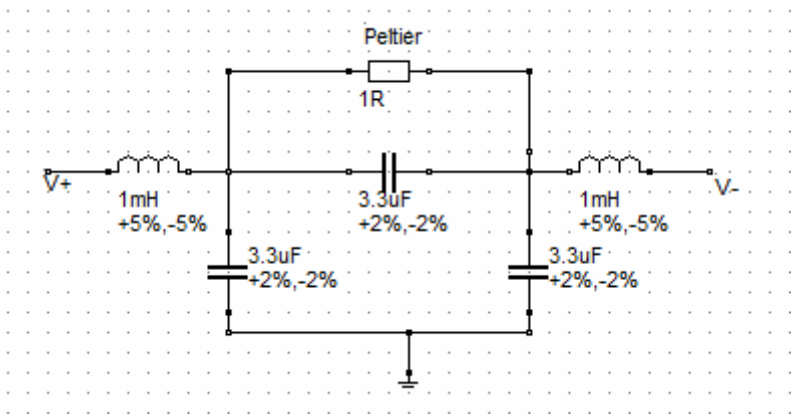



fig.3 : Structure du filtre lisseur

Le module Peltier est assimilé à un composant résistif.

La fonction de transfert obtenue est :

```
Transfer function:
        1.01e005
------------------------
s^2 + 144.3 s + 1.01e005
```

Le fichiel excel lisseur.xps permet d'obtenir les caratéristiques du filtre pour différentes valeurs des composants L et C et le fichier Matlab Calcul_filtre permet d'obtenir le tracé du Bode du filtre pour différentes valeurs de L et C.

Cependant, dans cette configuration, la partie haute du demi-pont n'est jamais passante car la capacité de bootstrap ne parvient pas à se charger à une valeur de tension suffisante. Pour utiliser les filtres prévus initialement, il faudrait donc rajouter sur chaque demi-pont des circuits élévateurs de tension de manière à garantir une tension grille-source permettant au MOS de la partie haute du pont d'être passant.

La mesure de température est effectuée par une sonde PT1000, c'est une sonde de platine qui présente la particularité d'avoir une résistance de 1000 Ohms à 0°C. Le platine présente l'avantage d'avoir une résistance qui varie très linéairement en fonction de la température, on peut donc par la mesure de cette résistance, trouver aisément la température correspondante. Cette résistance est placée dans un pont de Wheatstone dans lequel les autres résistances valent 1000 Ohms.

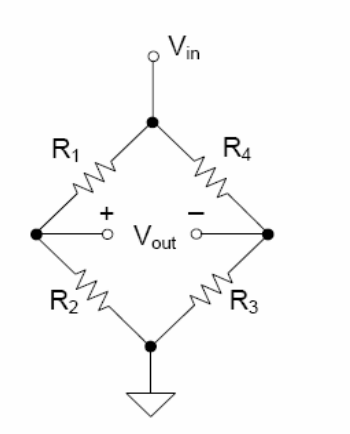



fig.4 : Structure de Wheatstone

On peut démontrer que pour ce pont, on a :

$$V_{out} = \frac{R_2 R_4 - R_1 R_3}{(R_1 + R_2)(R_3 + R_4)} V_{in}$$

En plaçant la résistance de platine à la place de R3 et en considérant que les autres résistances valent toutes R et que la sonde vaut R + Δ. On obtient :

$$V_{out} = \frac{\Delta}{4R + 2\Delta} V_{in}$$

où la tension Vout est la tension entre les deux points milieux du pont.

L'intérêt d'utiliser un pont de Wheatstone est qu'il réduit l'erreur commise sur la mesure de résistance. Une autre possibilité aurait été de créer un générateur de courant stable à l'aide d'un amplificateur opérationnel.

Les sondes PT100 sont plus courantes et donc plus faciles à trouver que les sondes PT1000 mais l'utilisation d'une PT1000 permet de réduire par 10 le courant passant dans le pont de Wheatstone comparativement à une PT100 et donc d'avoir un courant de l'ordre du mA dans chaque branche du pont et ainsi d'éviter un échauffement de la sonde qui fausserait la mesure.

Les résistances utilisées sont des résistances de précision à 0.1% et qui présentent un faible coefficient de variation avec la température.

En sortie du pont, un amplificateur d'instrumentation AD623 permet de faire la mesure de la tension Vout et de l'amplifier de manière à utiliser la dynamique maximale de 3,3V. La valeur d'amplification dépend uniquement du choix de la valeur de résistance Rg.

# V. Le module Caméra

Afin de correctement distinguer le contour des gouttes filmées par la caméra, il est nécessaire d'éclairer l'échantillon. Cet éclairage est réalisé par 2 leds de puissance de type Luxeon Star de couleur blanche. Bien que les courants mis en jeu ne soient pas très importants, le réglage de la luminosité est effectué par un signal pwm et un demi pont. Les leds sont directement alimentées par le 3,3V logique car le courant consommé est faible.

Le module Caméra et le programme de traitement d'image sont traités plus en détail dans une autre documentation.

# VI. Le Firmware

Le firmware du microcontrôleur permet de recevoir les données par l'USB et de piloter les différents périphériques. A chaque périphérique utilisé est associé un fichier spécifique dans le projet. Ainsi, on peut aisément réutiliser ces fichiers pour d'autres applications.

Le main du programme ne permet que d'initialiser l'ensemble des périphériques et de recevoir les messages en USB par scrutation. Il serait plus propre de modifier le code pour que la réception des messages USB entraîne une interruption et ainsi éviter d'utiliser inutilement le microcontrôleur.

Voici l'architecture des fichiers du projet tensiomètre :

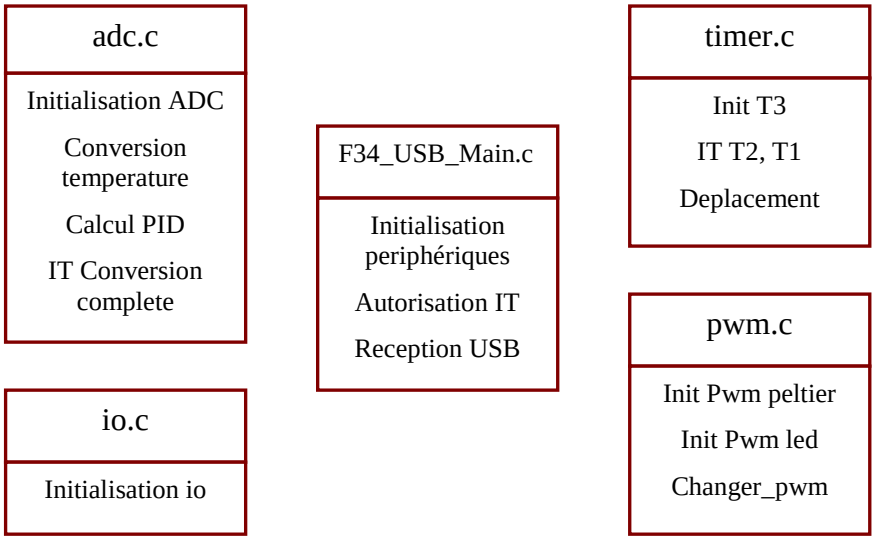



fig.5 : Architecture des fichiers du projet tensiomètre

Le fichier main a pour but d'initialiser l'ensemble des périphériques utilisés puis d'autoriser les interruptions. Après cela, le programme entre en boucle infinie en scrutant l'arrivée de nouveaux messages sur l'USB et en attendant la levée des interruptions.

Le fichier io.c contient uniquement la fonction d'initialisation de l'ensemble des ports d'entrée-sortie du microcontrôleur.

La régulation de température utilise le Timer 3, l'adc et une pwm ; le timer est utilisé pour limiter le nombre de mesures effectuées par l'adc. A chaque débordement du timer, l'adc convertit la valeur de sa tension d'entrée. Une fois la conversion terminée, l'interruption de l'adc est traitée. La valeur obtenue est traduite en température par la fonction Conversion_Temperature et elle sert au calcul de la nouvelle consigne pour la fonction Calcul_PID. Une fois la nouvelle consigne déterminée, la valeur de pwm du peltier est modifiée par la fonction Changer_Pwm.

Le contrôle du moteur utilise les timers 1 et 2. L'ensemble du code associé au moteur se trouve dans les interruptions des timers 1 et 2 et dans la fonction de Deplacement. Ce code est expliqué en détail dans une autre documentation.

Une deuxième pwm est utilisée pour contrôler le niveau de luminosité des leds. Après avoir été initialisée, cette pwm est modifiée directement dans le main sur réception d'un message par USB.

Les autres fichiers présents dans le projet servent au fonctionnement de l'USB, le code provient des fichiers exemples de Silabs.

Etant donné l'espace mémoire ram offert par le microcontrôleur et le nombre important de variables en particulier le nombre important de float utilisé pour la régulation de température, toutes les variables hormis celles liées à l'USB sont placées dans la xram.

# 1. Initialisation des différents périphériques

La première fonction appelée dans le main Init_Device permet l'initialisation de l'ensemble des périphériques. La première instruction réalisée consiste à désactiver le watchdog pour éviter d'avoir le microcontrôleur qui reset pour une raison inconnue. On peut rallonger le temps initial du watchdog en modifiant le fichier startup.

Par la suite, les initialisations de l'USB et de l'oscillateur sont faites par les fonctions Sysclk_Init et Usb0_Init. On utilise l'horloge interne du microcontrôleur à sa fréquence maximum de 46MHz

## a. IO_Init

La fonction IO_Init initialise l'ensemble des ports d'entrée-sortie et active la crossbar.

On utilise P00 et P01 pour les pwms du Peltier et des leds.

P24 et P25 sont configurés en sortie et vont servir au contrôle du moteur.

P26 est configuré en entrée analogique et va être utilisé pour la mesure de température et P27 va permettre d'activer ou non la régulation de température en venant en entrée de la patte SD des drivers IR2104

```
void Port_Init(void)
{
   P2MDIN   = 0xBF;
   P2MDOUT  = 0x0C;
   XBR1 = 0x42;
}
```

### b. Timer3_Init

Cette fonction sert à la configuration du timer 3 qui va être utilisé dans la régulation de température. Chaque interruption de ce timer entraîne le lancement de la conversion de l'ADC.

On utilise ce timer de manière à ralentir au maximum la mesure d'échantillon car les variations de température sont lentes. On utilise le timer 16 bits et on utilise la fréquence d'oscillation la plus faible SysClk/48.

On obtient une fréquence de l'ordre de 14Hz.

```
void Timer3_Init(void)
{
  TMR3CN   = 0x04;
  CKCON  = 0x02;
  TMR3L    = 0xff;
  TMR3H    = 0xff;
}
```

### c. Adc_Init

Le convertisseur analogique numérique est utilisé pour faire la conversion de la tension de sortie du capteur de température.

L'adc 10 bits est configuré en right adjusted. C'est-à-dire que le résultat se trouve dans les 2 bits de poids faible de ADC0H et dans ADC0L. L'adc convertit la tension se trouvant entre P26 et la masse.

La conversion est effectuée sur débordement de T3 et on utilise le mode track and hold c'est-à-dire que l'adc attend trois périodes au début de la conversion pour avoir une valeur stabilisée.

```
void Adc_Init(void)
{
        AMX0P = 0x05;                   // Entrée positivie = P2.6
        AMX0N = 0x1F;                   // Entrée négative = GND
        ADC0CN  = 0xC5;
}
```

### d. PCA_Init

Le PCA est un périphérique du microcontrôleur qui peut remplir différentes fonctions. Dans notre cas, il est utilisé pour générer les pwm du Peltier et des Leds.

Dans le cas du Peltier, afin d'avoir une précision plus importante, on utilise le mode pwm 16 bits alors que pour les leds, 256 valeurs possibles étant largement suffisant, on utilise le mode 8 bits.

Les interruptions générées par ce module à chaque période ou à chaque changement d'état ne sont pas gérées.

```
void PCA_Init (void)
{
        PCA0CPM0 = 0xC2;
        PCA0CN = 0x00;
        PCA0MD = 0x02;
        // Initialisation de la pwm peltier a 50%
        PCA0CPL0 = 0x00;
        PCA0CPH0 = 0x80;
        PCA0CPM1 = 0x42;
        PCA0MD &= ~0x40;
        // Initialisation de la pwm led a 0%
        PCA0CPL1 = 0x00;
        PCA0CPH1 = 0x00;
        PCA0CPL4  = 0x00;
    }
```

Une fois l'ensemble des périphériques initialisés, la fonction Enable_IT permet d'autoriser les interruptions de l'ensemble des périphériques.

Enfin, la fonction Start_Device lance les timers et les pwms.

## 2. La réception des données

La communication USB s'effectue grâce à l'envoi de trames de 8 octets. Ces trames sont définies dans le fichier principal du projet.

Pour envoyer une trame au microcontrôleur, il suffit de placer les données dans IN_PACKET et les données reçues se trouvent dans OUT_PACKET.

Etant donné la diversité des messages utilisés pour le tensiomètre. Un octet dans chaque trame est utilisé pour identifier le type de message, les autres contiennent les données à transmettre.

Voici un tableau récapitulatif de l'ensemble des trames échangées.

| 0 (Flag) | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 = Pas de message | | | | | | | |
| 2 = Peltier | | | | Température désirée | | | |
| 3 = Régulation off | | | | | | | |
| 4 = led | | Intensité lumineuse | | | | | |
| 5 = paramètres pid | | Proportionnel | | Intégral | | Dérivé | |
| 6 = Monter | | | | | | | |
| 7 = Descendre | | | | | | | |
| 8 = Moteur | | Nombre de pas | | | Sens | Nombre de cycles | Vitesse |

fig.6 : Trames USB échangées par le microcontrôleur et l'ordinateur central

Le nombre de pas du moteur, la température cible et les paramètres du pid sont sur 2 octets car un seul ne suffit pas à stocker la valeur. Le premier octet contient chaque fois le poids faible et le deuxième le poids fort.

Lors de l'envoi d'une température de consigne, la régulation de température est immédiatement activée et elle peut être désactivée par l'utilisateur à tout moment.

Les options « monter » et « descendre » signifient qu'on déplace le moteur jusqu'à avoir un changement d'état de l'un des détecteurs de contact. Les détecteurs de contact ne sont pas gérés dans le code.

# VII. Pilotage des périphériques

## 1. Le pilotage du moteur

L'ensemble du code relatif au déplacement du moteur est expliqué dans une autre documentation.

## 2. Le pilotage du module Peltier

La régulation de température est effectuée à l'aide d'un correcteur de type Proportionnel-Intégral-Dérivé. Les deux grosses difficultés avec ce type de régulation sont que la température évolue très lentement en comparaison de la vitesse de fonctionnement du microcontrôleur et qu'on ne connaît pas à priori la valeur de pwm à laquelle va se stabiliser le système pour une consigne donnée. En effet, en fonction de l'échantillon analysé et de la température extérieure, la valeur de pwm permettant d'atteindre la consigne peut varier, on ne peut donc pas avoir une courbe permettant de savoir pour une température donnée à quelle valeur va se stabiliser la pwm.

Il faut donc en permanence modifier la valeur de pwm de consigne en fonction de l'erreur qu'on observe et de l'évolution de cette dernière.

L'ensemble de la régulation de température est effectué dans l'interruption de l'adc qui survient à chaque fois qu'une conversion est terminée.

Les principales étapes de cette régulation sont :

- la conversion de la tension en température
- le calcul du pid
- le changement de la valeur de pwm
- l'adaptation de la valeur de consigne

Initialement, la pwm de consigne est fixée à 50% quelque soit la valeur de température que l'on souhaite atteindre. Cette valeur est réaffectée chaque fois qu'un nouvel ordre de régulation est envoyé.

### a. Conversion tension-température

Une fois la tension de sortie de l'amplificateur d'instrumentation convertit par l'adc, il est nécessaire de la traiter pour remonter à la température correspondante.

Dans un premier temps, on traduit la valeur convertit en valeur de tension correspondante. Le résultat de la conversion se trouve dans les deux bits de poids faible de ADCOH et dans ADC0L. Etant donné que sur 10 bits, on a 1024 possibilités et que le microcontrôleur est alimenté en 3,3V, on divise par 1024 et on multiplie par 3,3 pour se ramener à la valeur de tension.

```
tension_ampli=(double)(((ADC0H&0x03)*256.0+ADC0L)/1024.0)*3.3;
```

Par la suite, on divise le résultat par la valeur d'amplification de l'amplificateur d'instrumentation pour obtenir la tension différentielle entre les deux branches du pont de Wheatstone.

```
tension = tension_ampli/6.881;
```

Pour déterminer la valeur correspondante de la résistance de platine, on utilise la formule $Vout = \frac{\Delta}{4*R+2*\Delta}Vin$ qui nous donne $\Delta = \frac{4*R*Vout}{Vin-2*Vout}$ avec R = 1000 et Vin la tension d'alimentation du pont.

```
resistance = 1000.0 + 4.0*tension*1000.0/(3.3-2.0*tension);
```

Une fois la valeur de la résistance de platine déterminée, on calcule la valeur équivalente de température par une régression linéaire d'ordre 3 obtenue sur Matlab avec une erreur entre 0 et 80°C inférieure à 0,005°C.

```
return  ((resistance*resistance*resistance*7.4456e-10)+(resistance*resistance*7.4599e-
6)+resistance*0.2387-246.9315);
```

Pour ralentir encore le processus et pour diminuer l'importance de mauvaises mesures, la température est calculée en faisant la moyenne de dix mesures de température consécutives.

### b. Calcul PID

Le correcteur PID discret présente l'avantage de pouvoir être utilisé pour la régulation de tous types de systèmes à condition d'adapter les coefficients du correcteur aux constantes de temps du système.

Dans un premier temps, on calcule l'erreur courante entre la consigne et la température mesurée par la PT1000. Cette erreur est l'erreur proportionnelle.

```
erreur = (float)(consigne - temperature);
```

On détermine ensuite l'erreur intégrale en sommant l'ensemble des erreurs mais en ajoutant un coefficient permettant de donner moins d'importance aux erreurs antérieures.

```
erreur_int[1]=erreur_int[0]+erreur;
erreur_int[0]=erreur;
integral_err = (integral_err +  erreur)/2;
```

L’erreur dérivée est obtenue par simple soustraction des deux dernières erreurs. On ne divise pas par la période d’échantillonnage pour éviter un calcul inutile mais normalement toutes ces erreurs devraient être divisées par la période d’échantillonnage.

```
erreur_der[1]=erreur-erreur_der[0];
erreur_der[0]=erreur;
derivee_err=erreur_der[1];
```

La valeur de correction de pwm est ensuite juste une combinaison linéaire des erreurs avec leur coefficient associé.

```
correcteur = correct_proportionnel*erreur + correct_integral*integral_err + correct_derivee*derivee_err;
```

La principale difficulté est en réalité de choisir les coefficients du correcteur car on ne dispose pas de modélisation du système qui présente de nombreuses non-linéarités et qui est donc difficile à modéliser correctement.

Cependant, on sait que les constantes de temps du système sont très grandes, ceci implique d’avoir un coefficient dérivé très important pour éviter les fluctuations autour du point d’équilibre.

Le coefficient proportionnel permet de rendre le système plus rapide. En effet, plus il est grand, plus le système va converger rapidement vers la consigne. Le coefficient intégral permet d’éliminer l’erreur statique, c'est-à-dire qu’il assure que la valeur de stabilisation du système sera la valeur de consigne ; il n’a pas besoin d’être très élevé dans notre cas. Le coefficient dérivé enfin permet d’anticiper l’arrivée sur la valeur de consigne afin d’éviter de dépasser de manière trop importante et ensuite d’osciller autour du point d’équilibre.

Ces coefficients sont définis et initialisés dans le main du programme, ils peuvent être modifiés par l’utilisateur par l’envoi d’une trame USB.

### c. Changement PWM

Une fois la valeur du correcteur déterminée, la valeur de pwm est modifiée. Cependant, à tout moment, la valeur de pwm est la somme de la valeur déterminée par le correcteur et de la pwm de consigne ramenée sur 1000. En effet, il est plus simple de travailler avec des valeurs variant entre 0 et 1000 plutôt qu’entre 0 et 65536. La fonction Changer_pwm permet donc d’adapter les échelles de manière à utiliser toutes les possibilités du microcontrôleur.

La somme pwm + pwm_consigne est bornée entre 50 et 995 de manière à éviter d’avoir une tension continue aux bornes du Peltier et donc de chauffer démesurément les MOS.

```
temp = pwm * 30000.0/ 1000.0;
pwm = (long int)temp;
PCA0CPL0 = pwm & 0x00FF;
PCA0CPH0 = (pwm >> 8) & 0xFF;
```

En réalité, étant donné que la carte de régulation de température altère le signal pwm, on arrive à la pleine échelle avec une pwm de l'ordre 50% donc on adapte l'échelle uniquement entre 0 et 30000.

### d. Adaptation de la consigne

L'adaptation de la valeur de consigne de pwm est la partie la plus délicate de la régulation. La principale difficulté provient du fait qu'on ne sait pas vers quelle valeur le système va se stabiliser pour rester à la valeur de consigne de température.

On part donc d'une consigne initiale de 50% soit 500. Par la suite, on va environ toutes les 5s mesurée la valeur de l'erreur puis on va observer la variation de cette erreur sur 30s. Si cette erreur varie beaucoup, cela démontre qu'on est loin de la valeur de température désirée donc on ne modifie pas trop la consigne. Par contre, si l'erreur varie peu, cela veut dire qu'on tend à se stabiliser mais qu'on n'a pas atteint la valeur de température de consigne, il est donc nécessaire d'adapter la pwm de consigne à une valeur proche de la valeur de pwm moyenne sur ces 15s.

La variable erreur_temperature contient les erreurs accumulées sur les 30s et la variable erreur_stable contient la différence de la moyenne des 3 premières erreurs et de la moyenne des 3 dernières. La variable save_pwm va permettre de faire la moyenne de la pwm sur 15s.

Une fois ces erreurs déterminées, on borne la valeur d'erreur stable dans le cas ou elle est très faible.

Par la suite, on adapte la valeur de la pwm de consigne en distinguant le cas ou la moyenne de la pwm est proche de la valeur de consigne et le cas ou on a un gros écart. Plus l'erreur_stable sur les 30 dernières secondes est faible, plus on va donner de l'importance à la valeur moyenne de pwm.

A la fin, on décale les erreurs de température da manière à recommencer un cycle. Chaque 15s, la valeur de pwm de consigne est modifiée en fonction de l'évolution du système. A l'initialisation, les valeurs d'erreurs de température sont placées à des valeurs très importantes pour que l'erreur soit colossale et donc éviter ainsi un changement de la valeur de consigne dès les premières 15s car le système répond très lentement.

```
erreur_temperature[compteur_nb_erreur] = consigne - temperature;
save_pwm[compteur_nb_erreur - 3] = pwm;
compteur_nb_erreur++;
```

```
        if(compteur_nb_erreur == 6)
        {
        save_pwm[3]=(save_pwm[0]+save_pwm[1]+save_pwm[2])/3;
        erreur_stable = ((erreur_temperature[0] + erreur_temperature[1] + erreur_temperature[2])/3.0 - (erreur_temperature[3] + erreur_temperature[4] + erreur_temperature[5])/3.0);

        if (( erreur_stable >= 0.0)  && ( erreur_stable <= 0.025))
        { erreur_stable = 0.025; }
        else if (( erreur_stable <= 0.0)  && ( erreur_stable >= -0.025)
        { erreur_stable = -0.025; }

        if ((save_pwm[3]-pwm_consigne)>=10||(save_pwm[3]-pwm_consigne)<=-10)
        {
                if (erreur_stable >= 0)
                {
                        pwm_consigne = pwm_consigne + (save_pwm[3] - pwm_consigne)/(40.0* erreur_stable);
                }
                else
                {
                        pwm_consigne = pwm_consigne - (save_pwm[3] - pwm_consigne)/(40.0* erreur_stable);
                }
        }

        else if ((save_pwm[3] - pwm_consigne)<0)
        {       pwm_consigne = pwm_consigne - 10;       }
        else if ((save_pwm[3] - pwm_consigne)>0)
        {       pwm_consigne = pwm_consigne + 10;       }

        compteur_nb_erreur = 3;
        erreur_temperature[0] = erreur_temperature[3];
        erreur_temperature[1] = erreur_temperature[4];
```

```
erreur_temperature[2] = erreur_temperature[5];
}
```

## 3. Le pilotage des LEDs

Le pilotage des LEDs est très simple. Le rapport cyclique placé dans PCA0PH1 correspond à la valeur indiquée par l'utilisateur est reçue par l'USB.

```
else if (Mode == 4) // Changement de la luminosité des leds
{
    PCA0CPH1 = OUT_PACKET[2];
}
```

Etant donné que la pwm est très rapide par rapport à l'œil humain, les leds paraissent au maximum de leur luminosité dès qu'on atteint un rapport cyclique de l'ordre de 40%. Pour utiliser ce code, il faut donc envoyer des valeurs par USB comprises entre 0 et 100 de manière à avoir une échelle utilisée de manière optimum.

# Table des annexes

# Annexe 1 : Schéma de la carte microcontrôleur

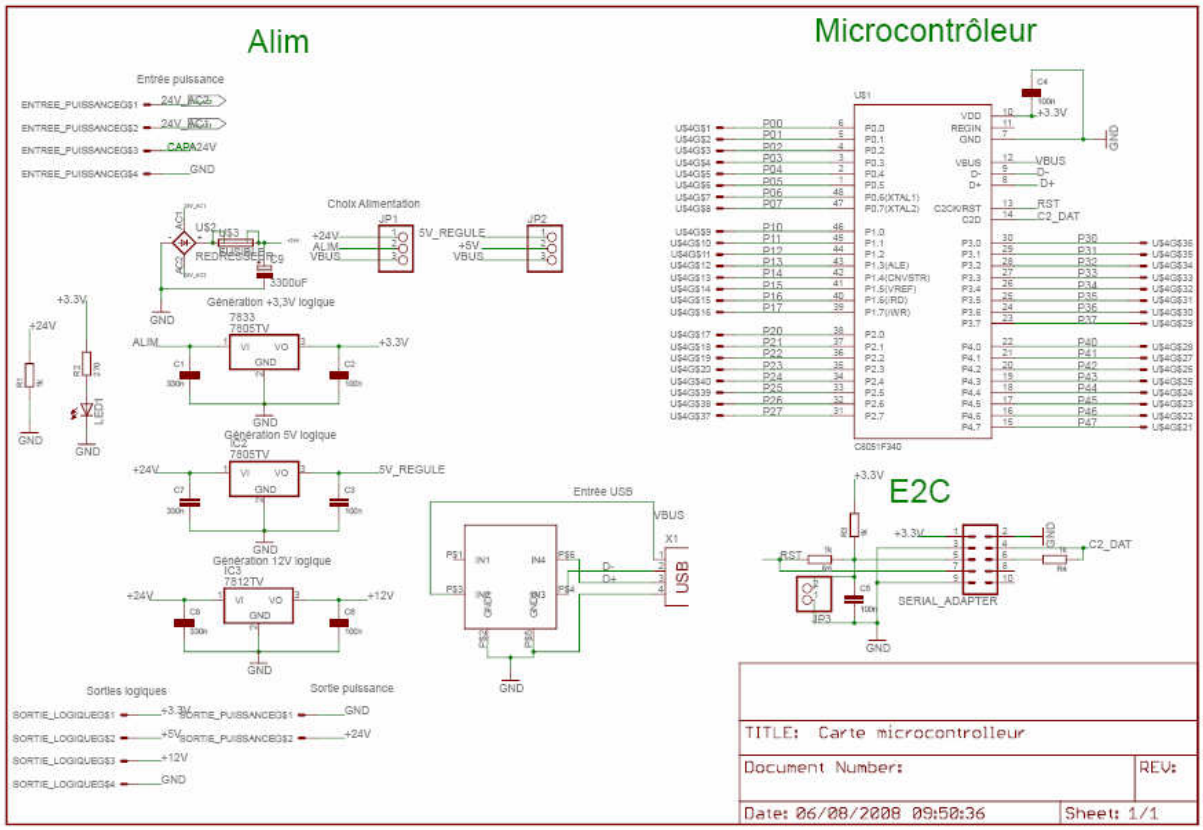



fig.7 : Schéma électronique de la carte microcontrôleur

# Annexe 2 : Typon de la carte microcontrôleur

Bottom

Top

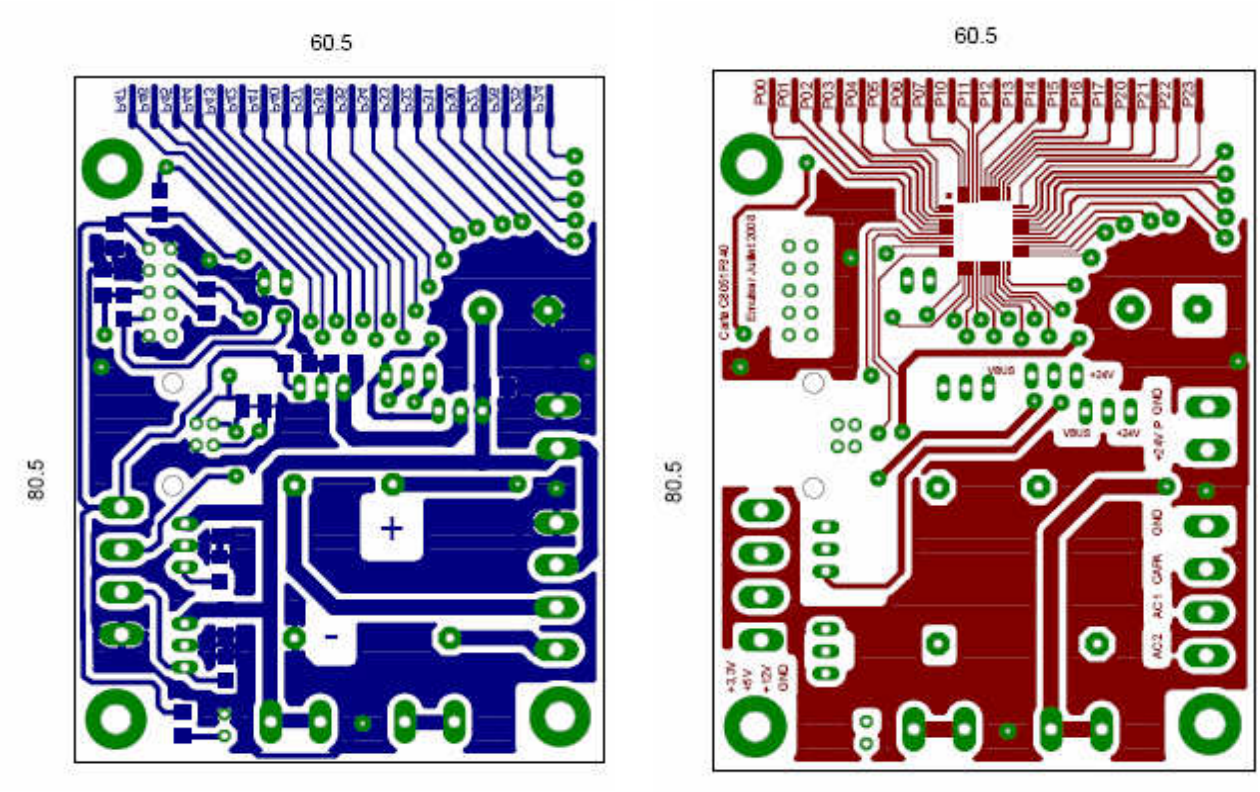



fig.8 : Typon de la carte microcontrôleur

# Annexe 3 : Schéma de la carte de régulation de température

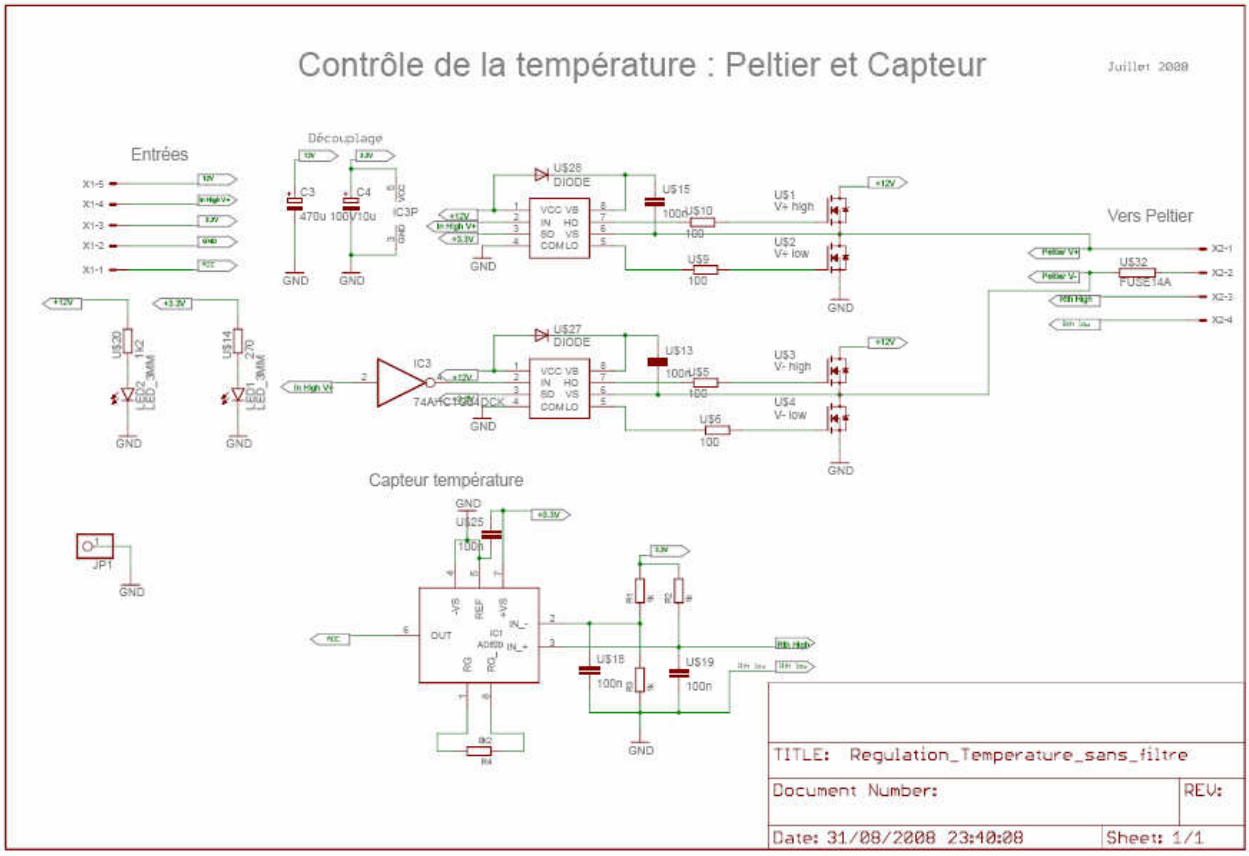
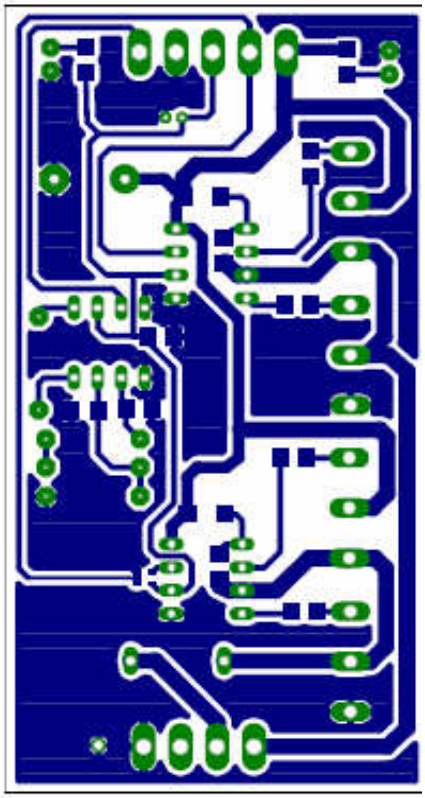



fig.9 : Schéma électronique de la carte de régulation de température

# Annexe 4 : Typon de la carte de régulation de température

Bottom



Top

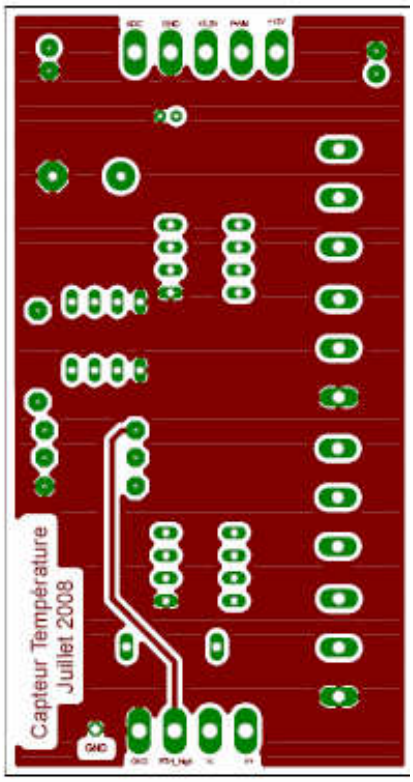



fig.10 : Typon de la carte de régulation de température

# Annexe 5 : Schéma de la carte LED

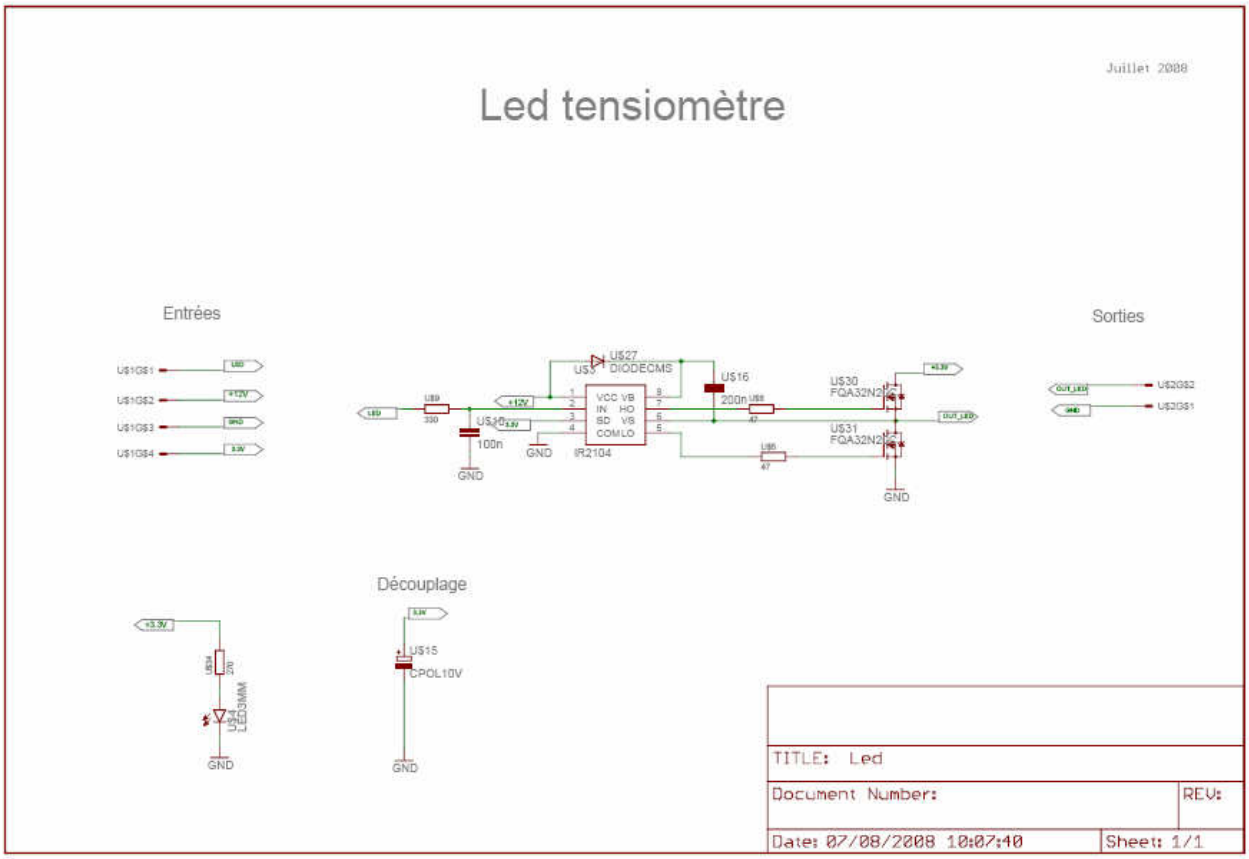
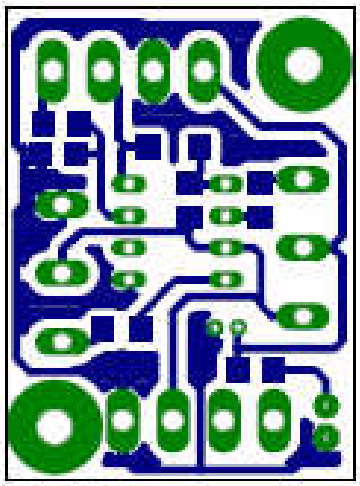



fig.11 : Schéma électronique de la carte LED

# Annexe 6 : Typon de la carte LED

Bottom



Top

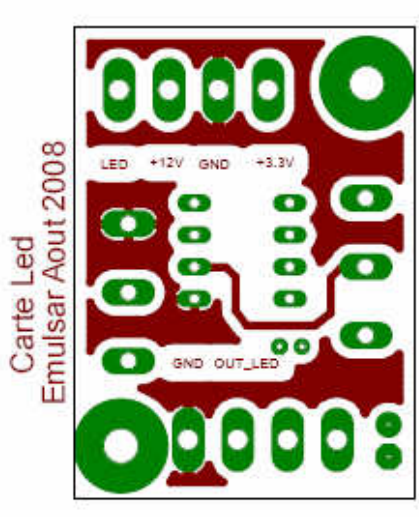



fig.12 : Typon de la carte LED