TIPE - FOUR DE REFUSION
DOSSIER

par Nicolas Viennot

...:: INTRODUCTION ::...

Nous sommes entourés d'équipements électroniques dont la réalisation est un long processus. L'une des étapes de la réalisation de cartes électronique est la fixation des composants (soudage). Plusieurs méthodes existent dont la refusion. Le sujet de mon TIPE est la conception et la réalisation d'un four de refusion du cahier des charges au réglage de l'asservissement en passant par la réalisation des cartes électroniques. La réalisation d'un tel projet nécessite plusieurs étapes:

- La définition du besoin, du cahier des charges.
- Recherche des solutions technologiques, choix des composants.
- Ebauche du schéma électronique, prototypage, début du développement logiciel.
- Réalisation du circuit électronique, développement logiciel.
- Réalisation de l'asservissement, expérimentation.

Je détaillerai toutes ces étapes dans ce dossier.

...:: LE SOMMAIRE ::...

LA PARTIE MATÉRIELLE

...:: L'ÉTUDE DU SYSTÈME ::...

Le cahier des charges

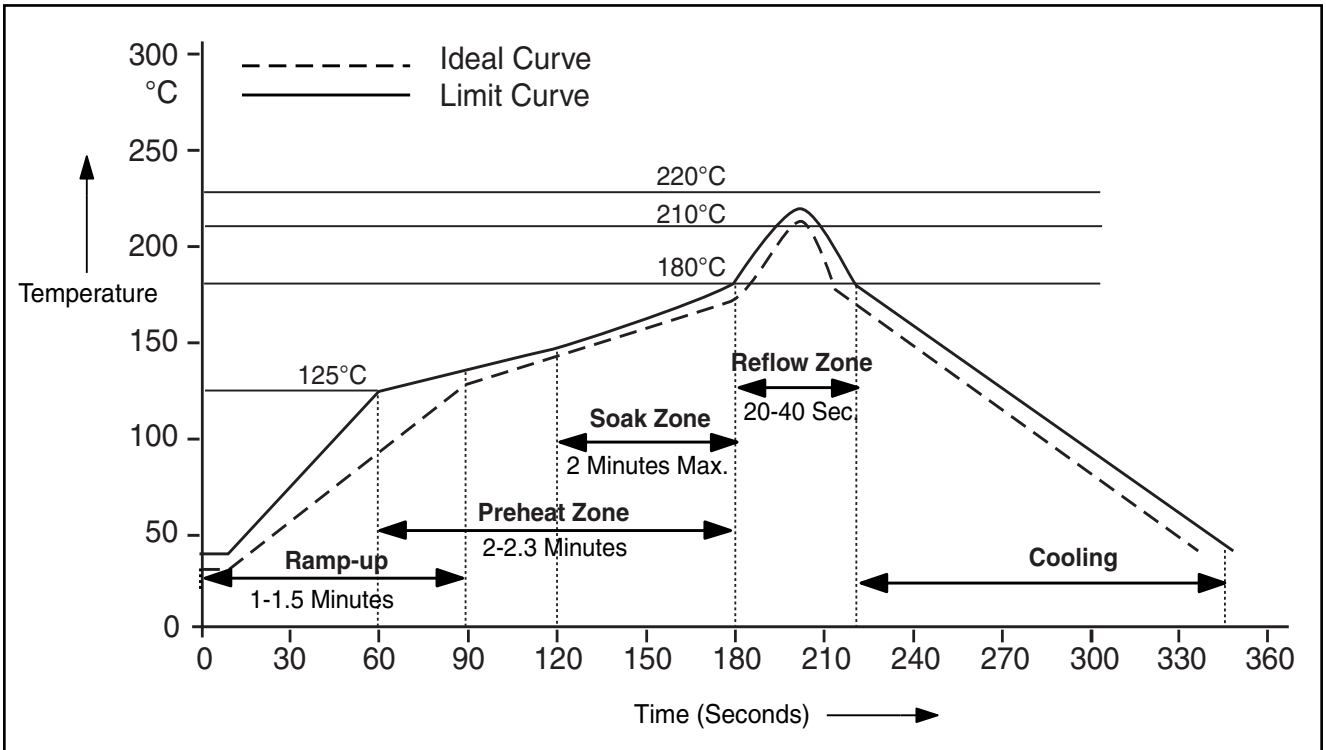
Dans mon cahier des charges, je définie les impératifs (pour que le système réponde au besoin) et les critères qui feront que le système soit le plus agréable à utiliser (esthétique, etc...). Ces derniers ne doivent pas être négligés. Cette séparation permet de définir clairement les priorités de développement qui sont essentielles pour optimiser le temps ainsi que le budget alloué à chaque partie.

Les Imperatifs

Le four doit pouvoir accepter les cartes de 20x10cm et leur faire subir un profil de ce type:
Il existe beaucoup de profils. En effet, chaque pâte à braser possède ses propres caractéristique, de plus, il faut ajuster le profil suivant les propriétés thermique du circuit imprimé. En clair, Le four doit être capable de monter en température à 3°/sec jusqu'à 350° (il faut toujours prévoir de la marge pour ce genre de variable). Il doit être également capable de descendre en température à au moins 3°/sec pour ne pas endommager les composants à haute température.
Le contrôleur (ce qui pilote le four, c'est à dire toute l'électronique) doit être séparé du four. Ce qui permet de posséder plusieurs fours/appareils à asservir et un seul contrôleur, donc permet de faire des économies, ce qui le rend idéal pour les petites entreprises. Ce choix qui implique un paramétrage dynamique de(s) capteur(s).
Le contrôleur doit être absolument autonome et ne doit pas déprendre d'un PC pour son bon fonctionnement. Cependant un PC doit pouvoir être branché pour récupérer des données de temperature, pour tracer des courbes par exemple. Donc la partie puissance doit être isolée galvaniquement de la partie commande pour protéger le PC.
Le système doit être le plus sûr possible.

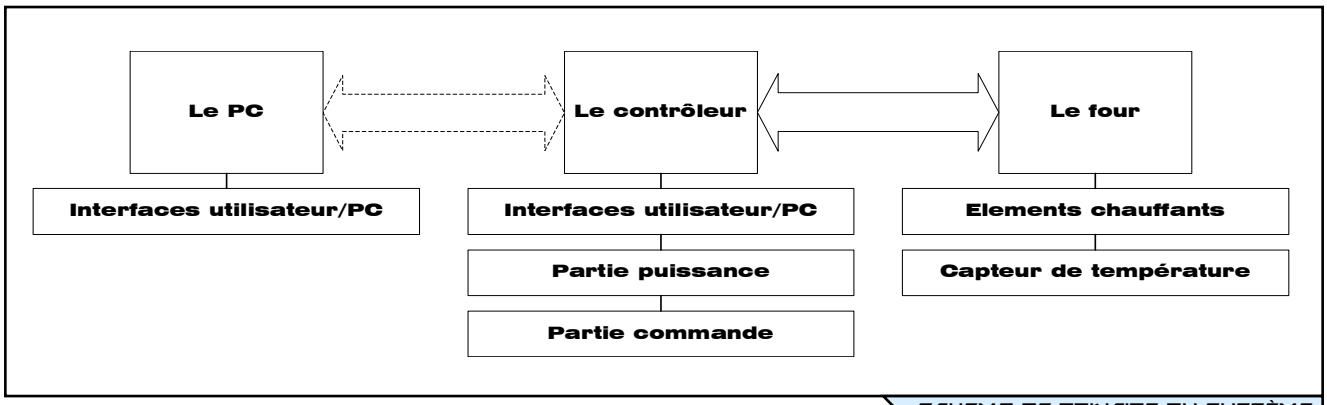
Les 'plus'

L'utilisation du four doit se faire le plus souplement possible: la mise en marche doit se faire très facilement, les interfaces doivent être le plus intuitives possible, etc...
Le contrôleur doit pouvoir gérer différents types de capteur de température afin de rendre le système plus flexible.
Le système doit être très facilement mis à jour, maintenable, ce qui assure sa durée de vie.
L'esthétique doit être soignée, que ce soit pour le contrôleur ou au niveau des interfaces des logiciels développés.



EXEMPLE DE PROFIL (LX)

...:: LES SOLUTIONS TECHNOLOGIQUES ::...



SCHEMA DE PRINCIPE DU SYSTÈME

Le choix des solutions technologiques est une étape longue, en effet il faut se documenter, réfléchir à la meilleure solution parmi tant d'autres, contacter les distributeurs, entreprises, comparer différents compostants, revoir à la baisse certains critères, etc...

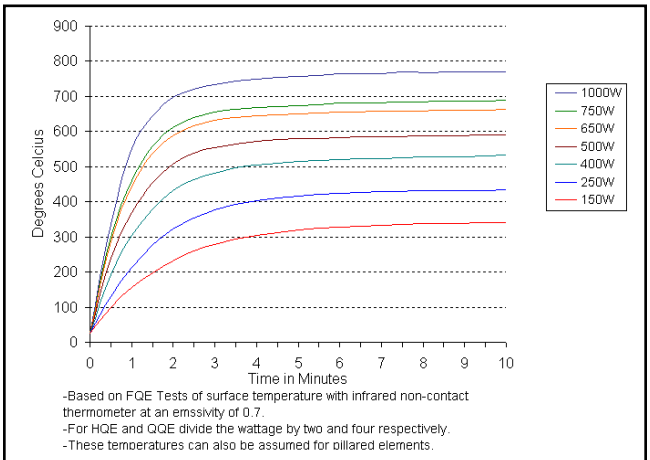
Les elements chauffants

Il y a 3 moyens de chauffe:

• la conduction: ce qui revient à poser la surface à chauffer sur une plaque chauffante, solution complètement inappropriée (le circuit brulerait).

• la convection: la surface est chauffé essentiellement par l'air réchauffé par les éléments chauffants, les temps de réponse en température sont longs, ce qui implique d'avoir différentes zones où le circuit evolue sur un convoyeur, donc cette solution est inappropriée.

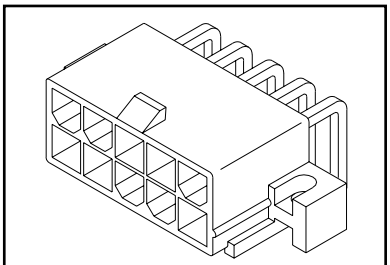
• le rayonnement: la surface à chauffer est exposée à un rayonnement infrarouge, toute la puissance consommée est restituée instantanément et presque intégralement à la surface. J'ai choisi cette solution avec 2 panneaux 'Full Quartz Element' de Ceramicx Ireland Ltd de 1000W chacun que j'ai fixé dans un petit four d'une capacité de 7 litres. D'après la courbe fournie par Ceramicx [CE], ils fournissent un très bon temps de réponse, ce qui est vérifié experimentalement. J'ai câblé l'intérieur du four avec du



COURBE TEMPÉRATURE/TEMPS DES CERAMICX

LES ÉLÉMENTS CHAUFFANTS EN FONCTIONNEMENT

câble 16A isolé en téflon relié par des cosses (on ne peut pas souder à l'étain pour les raisons de température de fonctionnement). La connection au contrôleur se fait avec un connecteur de puissance MiniFit Jr de Molex [MO]. Pour alimenter les éléments chauffants depuis le contrôleur, j'utilise un circuit à base de TRIAC qui convient parfaitement en régime alternatif. Le fonctionnement du TRIAC est analogue à celui d'une chasse d'eau: on le rend passant par une impulsion sur la gâchette et il le reste tant qu'un courant le traverse (jusqu'au prochain passage à zero de la tension du secteur). J'ai choisis un BTA140B-600 de Philips Semiconductors [PS] dont le courant admissible maximal est de 25A. Il faut prévoir de la place sur le circuit pour le dissipateur thermique. L'impulsion sera émise par le microcontrôleur. Pour réaliser l'isolation galvanique, il faut utiliser un opto-triac pour déclencher la gâchette. J'ai choisi le MOC3021. Le montage utilisé provient d'une note d'application de Fairchild



CONNECTEUR MINIFIT_JR

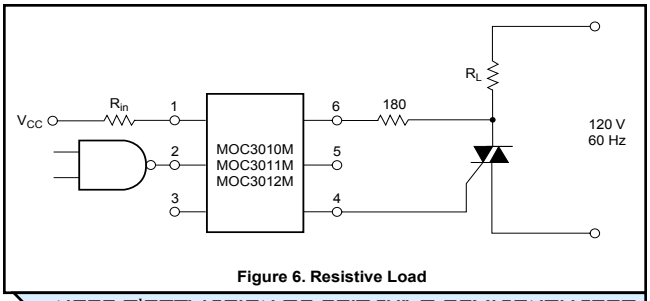
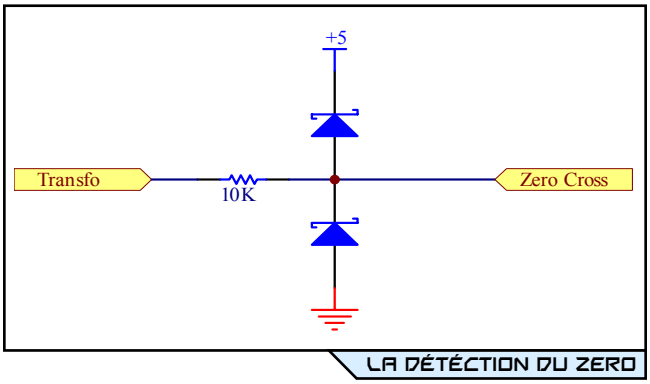


Figure 6. Resistive Load

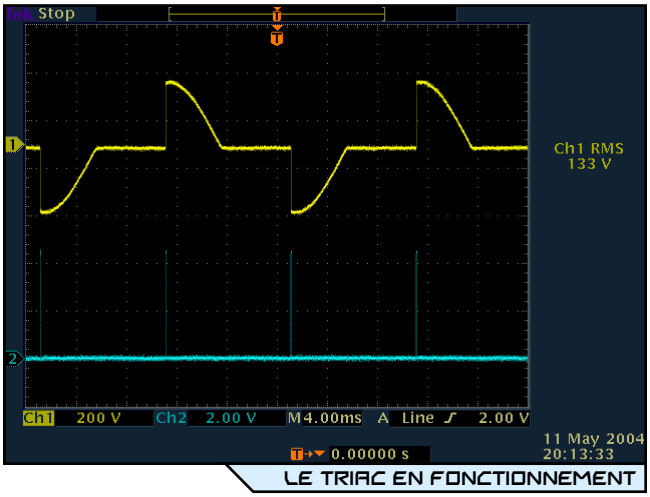
NOTE D'APPLICAION DE FAIRCHILD SEMICONDUCTOR

Semiconductor [FS]. Pour que le microcontrôleur sache quand déclencher le TRIAC, il doit connaître les passages à zero de la tension du secteur, pour cela j'utilise la 2ème sortie du transformateur (pour respecter l'isolation galvanique) qui délivre une tension alternative sinusoïdale de 10V à -10V donc j'ai mis des diodes Schottky (tension de seuil = 0,2V) pour



LA DÉTÉCTION DU ZERO

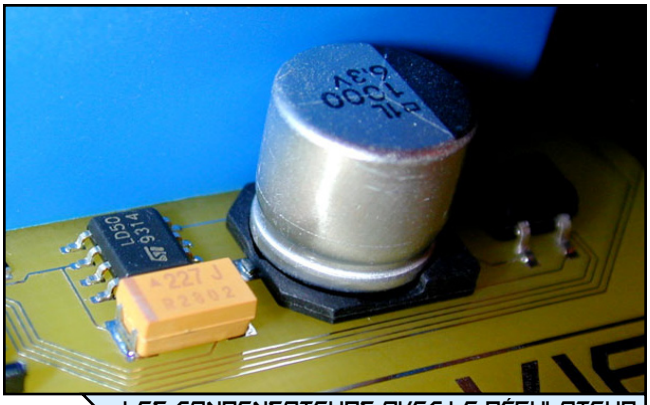
la limiter à 0V-5V. Ci-dessous une capture d'écran de mon oscilloscope montrant la tension aux bornes des éléments chauffants (CH1) et les impulsions du microcontrôleur (CH2) sur l'opto-triac.



LE TRIAC EN FONCTIONNEMENT

L'alimentation

Le four est directement branché au secteur, et le contrôleur est branché au four. On utilise le redressement double alternance. J'ai choisi un transformateur double sortie 2x6V - 2.5VA (la première sortie etant utilisée pour l'alimentation et l'autre pour la détection du zéro), un pont de diode classique, un condensateur de filtrage 1000µF et un régulateur 5V 500mA: ST LD1117D50 [ST1]. Avec cette configuration, on peut consommer 300mA sans avoir de perturbation gênante sur la ligne 5V, ce qui est suffisant pour ce que le circuit pourrait consommer.



LES CONDENSATEURS AVEC LE RÉGULATEUR

Le capteur de temperature

On veut mesurer la température dans le four sur une plage de 0° à 250°. Il existe 4 types de capteur de température [M1]:

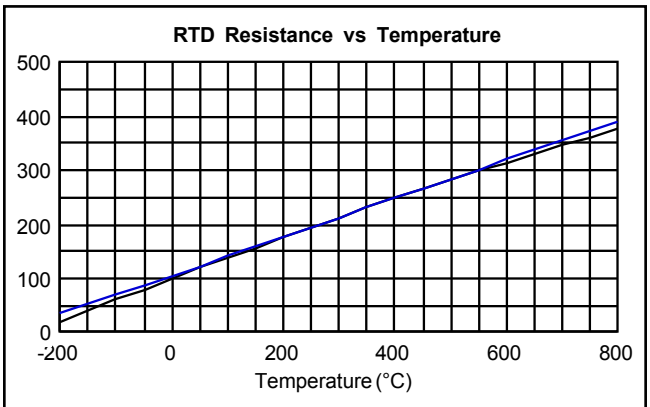
• Les thermocouples: Ils ne nécessitent pas d'alimentation, balayent une large plage de température (0-1500°C) et sont très robustes. Le capteur génère une fem (qui est très petite) en fonction de la température (non linéairement).

• Les diodes/circuits intégrées: Ils sont très facilement implémentable: leur sorties sont très variées, analogique, série numérique, logique, modulation à largeur d'impulsion...

• Les thermistances: Elles sont peu chères, mais supporte une plage de -50°C à 150°C, la sortie n'est pas linéaire [BT].

• Les RTDs: Elles sont très précises, mais relativement chères, sont très linéaire. Elles sont disponibles dans une grande variété de package. Le métal platine est utilisé le plus couramment. Les 2 grandeurs d'une RTD sont: la résistance de base qui est la résistance à 0°C (100Ω, 1000Ω, ...) et le coefficient de température (0,00385Ω/Ω/°C pour le platine). Le principe: quand la température augmente, les électrons se dispersent, et la vitesse moyenne diminue [PY].

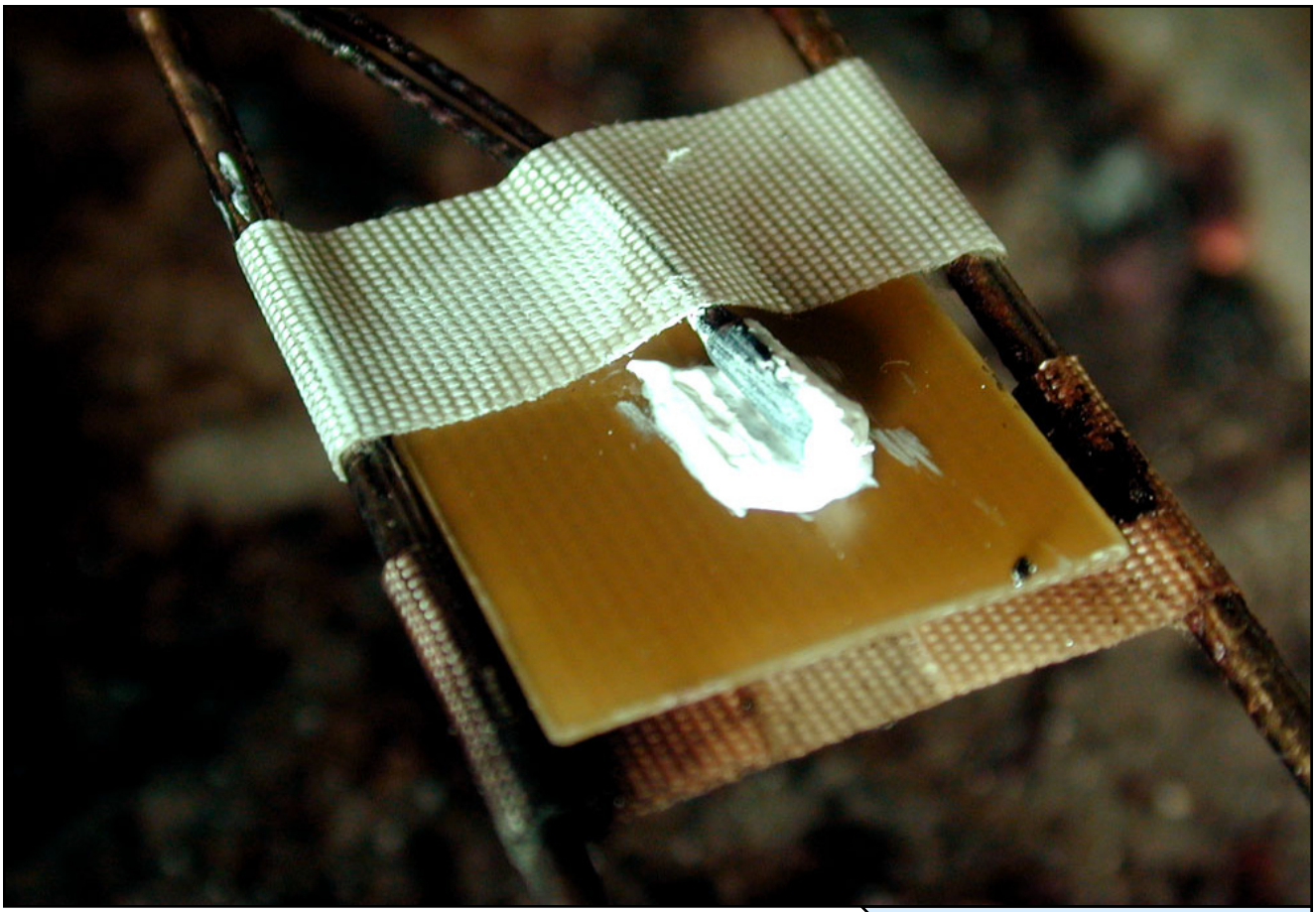
J'ai choisi un capteur RTD Honeywell HEL-707-U-0-12-00 [HW] (appelé aussi capteur à résistance de platine, ou sonde PT100,...). La résistance est fonction de la température: $R=R_0(1+\alpha_1 T+\alpha_2 T+...+\alpha_n T)$, mais on peut utiliser la relation $R=R_0(1+\alpha T)$ compte tenu



COURBE CARACTÉRISTIQUE D'UNE PT100 [NS]

Attribute	RTD	Thermocouple	Thermistor	Silicon IC
Temperature Range	-200 to 850°C	-184 to 1260°C	-55 to +150°C	-55 to +125°C
Temperature (t) Accuracy	Class B = ±[0.012 + (0.0019 | t |) -6x10^{-7}t^2]	Greater of ±2.2°C or ±0.75%	Various, ±0.5 to 5°C	Various, ±0.5 to 3°C
Output Signal	≈ 0.00385 Ω/Ω/°C	Voltage (40 µ/°C)	≈ 4% ΔR/Δt for 0°C ≤ t ≤ 70°C	Analog, Serial, Logic, Duty Cycle
Linearity	Excellent	Fair	Poor	Good
Precision	Excellent	Fair	Poor	Fair
Durability	Good, Wire wound prone to open-circuit vibration failures	Good at lower temps., poor at high temps., open-circuit vibration failures	Good, Power Specification is derated with temperature	Excellent
Thermal Response Time	Fast (function of probe material)	Fast (function of probe material)	Moderate	Slow
Cost	Wire wound - High, Thin film - Moderate	Low	Low	Moderate
Package Options	Many	Many	Many	Limited, IC packages
Interface Issues	Small ΔR/Δt	Cold junction compensation, Small ΔV	Non-linear resistance	Sensor is located on PCB

COMPARAISON DES CAPTEURS DE TEMPÉRATURE [M2]



LA SONDE RTD SUR SON SUPPORT

de la très bonne linéarité du capteur. Pour mesurer cette résistance, il suffit d'injecter un courant constant et mesurer la tension à ses bornes. Si le courant d'excitation est trop important, le capteur s'échauffe, et la mesure devient inexacte. Pour une PT100 (résistance de base de 100Ω), on injecte un courant d'environ 10mA. La résistance des fils doit être prise en compte pour une précision optimale, c'est pour cela qu'il existe des sondes 3 ou 4 fils. J'ai choisi une sonde PT1000 3fils. J'ai fixé la sonde sur un morceau de circuit imprimé avec du scotch en fibre de verre et un peu de pâte thermique. Expérimentalement, la valeur de la température lue est précise à 5°C près et le temps de réponse est < 2secs. J'ai utilisé 2 moyens de contrôle: un thermomètre infrarouge et de la pâte à braser Sn63Pb37 qui se liquéfie à 183°C.

Caractéristique de la Résistance en fonction de la Température

Comportement réel

```
> A := alpha * ( 1 + delta/100 ):
  B := -alpha * delta/100^2:
  R[Réel] := R[0] * ( 1 + A*T + B*T^2 );
```

$$R_{Réel} := R_0 \left(1 + a \left(1 + \frac{1}{100} \delta \right) T - \frac{1}{10000} a \delta T^2 \right)$$

Comportement linéarisé

```
> R[Linéaire] := R[0] * ( 1 + alpha*T );
```

$$R_{Linéaire} := R_0 (1 + a T)$$

Constantes de la sonde utilisée fournies par le constructeur

```
> R[0] := 1000;
  alpha := 0.00375;
  delta := 1.605;
```

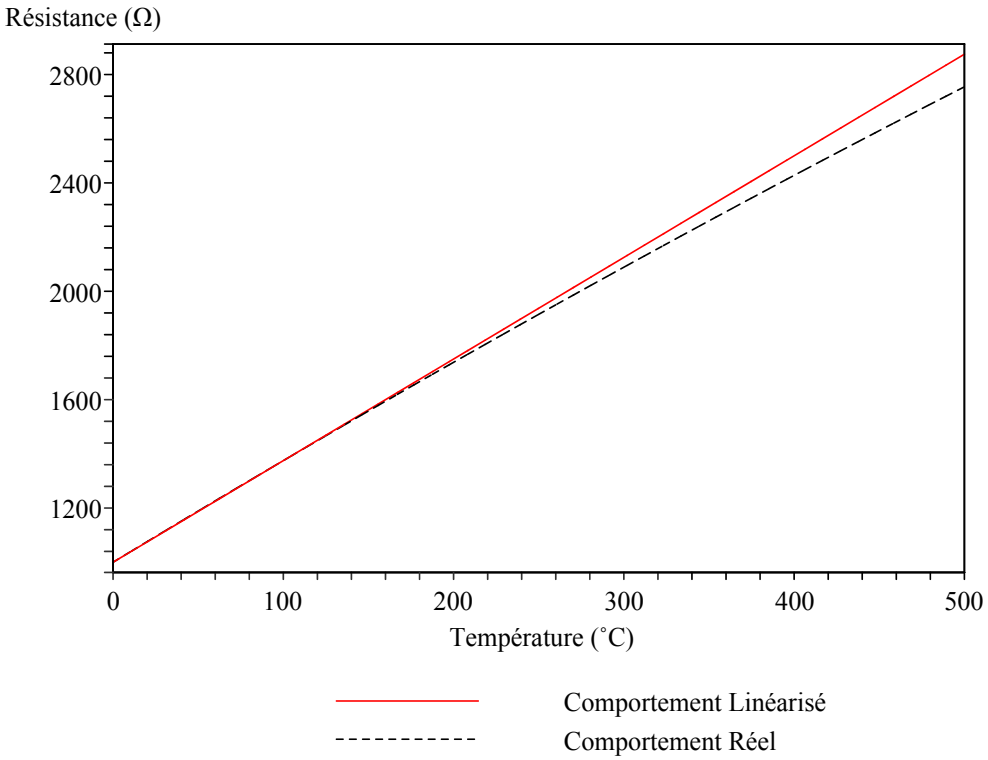
$$R_0 := 1000$$

$$a := 0.00375$$

$$\delta := 1.605$$

La Résistance en fonction de la Température

```
> plot( [ R[Linéaire], R[Réel] ], T=0..500, legend = [ "Comportement Linéarisé", "Comportement Réel" ],
  linestyle = [ 1, 3 ], color = [ red, black ], axes = BOXED, font = [ TIMES, ROMAN, 12 ] );
```



Erreur commise en haute température (180°C - 1675Ω)

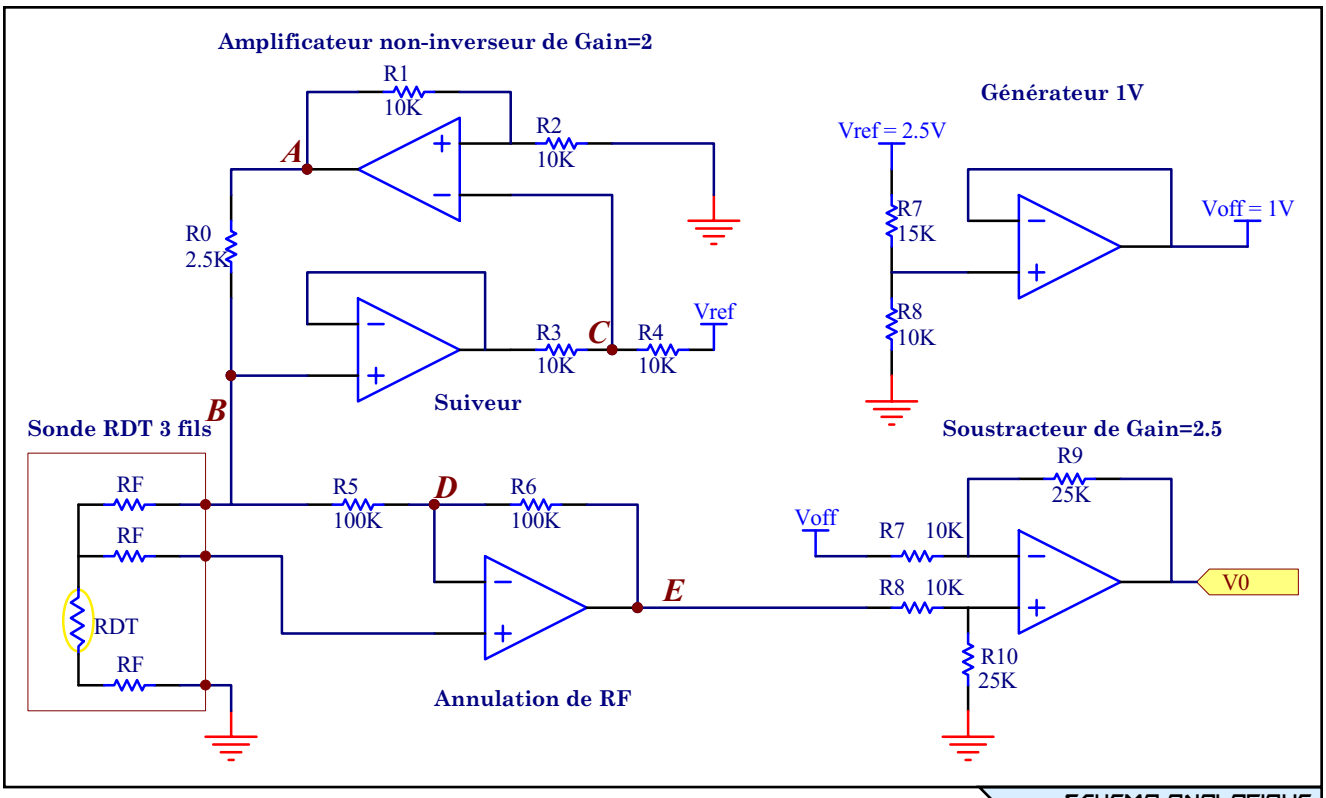
```
> T[Réel][1675] := solve( R[Réel]=1675, T )[1];
  T[Linéaire][1675] := solve( R[Linéaire]=1675, T );
```

$$T_{Réel_{1675}} := 182.4128214$$

$$T_{Linéaire_{1675}} := 180.$$

```
> Erreur[1675] := T[Réel][1675] - T[Linéaire][1675];
```

$$Erreur_{1675} := 2.4128214$$



SCHEMA ANALOGIQUE

$$V_A = V_C\,(1+\frac{R_1}{R_2}) \quad \Rightarrow \quad V_A = 2V_C$$

$$V_{ref} - V_C = V_C - V_B \quad \Rightarrow \quad V_B = 2V_C - V_{ref}$$

$$I_{RDT} = \frac{V_A - V_B}{R_0} = \frac{V_{ref}}{R_0} \quad \Rightarrow \quad I_{RDT} = 1mA$$

$$V_B = V_{RDT} + 2V_{RF}$$

$$V_D = V_{RDT} + V_{RF}$$

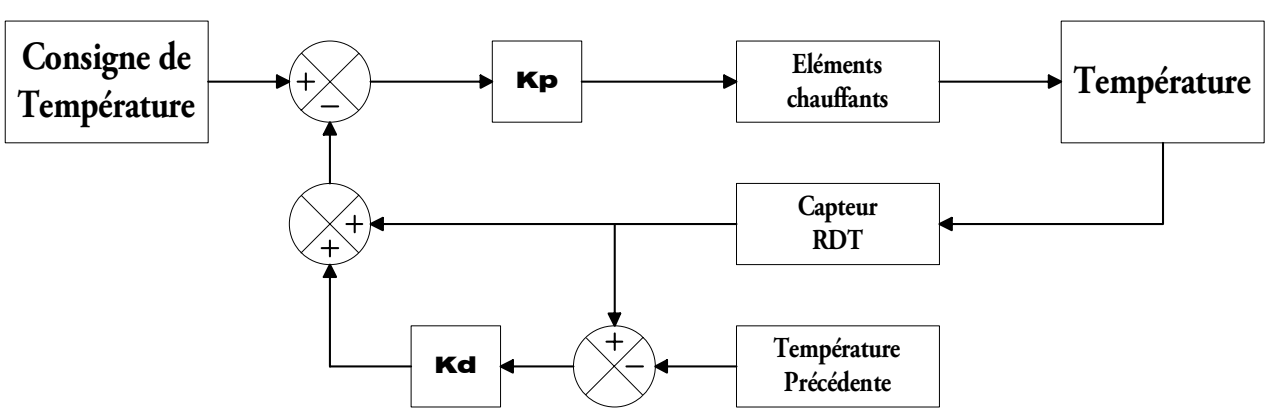
$$V_{R5} = V_{R6} \quad \Rightarrow \quad V_B - V_D = V_D - V_E$$

$$V_E = 2V_D - V_B \quad \Rightarrow \quad V_E = V_{RDT}$$

$$T \in [0°C, 250°C] \quad \Rightarrow \quad V_{RDT} \in [1V, 2V]$$

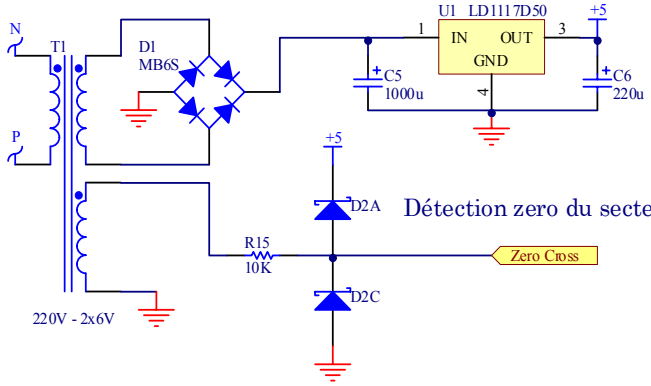
$$V_0 = (V_E - V_{off})\frac{R_{10}}{R_8} \quad \Rightarrow \quad V_0 = 2{,}5 \times (V_{RDT} - 1V)$$

EQUATIONS



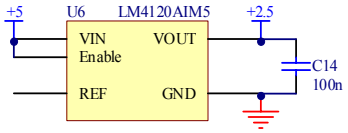
LE SCHEMA BLOC DE L'ASSERVISSEMENT

Transformateur + pont de diode



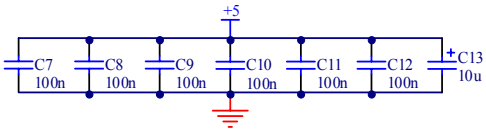
Régulateur 5V

Régulateur 2.5V



Détection zero du secteur

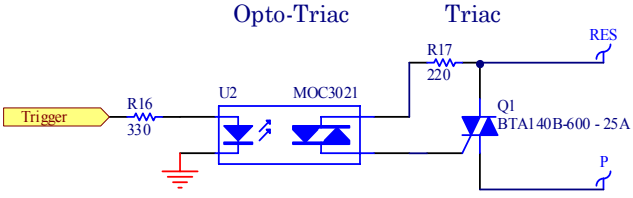
Condensateurs de découplage



Etage d'Alimentation

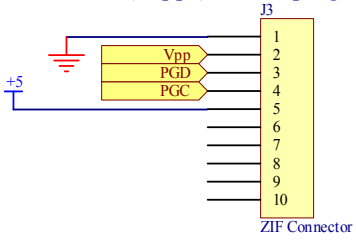
Opto-Triac

Triac

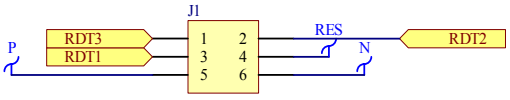


'Pre-actionneur' des éléments chauffants

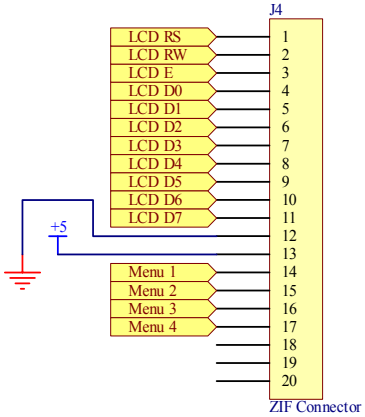
Connecteur (nappe) vers le programmateur



Connecteur de puissance vers le four

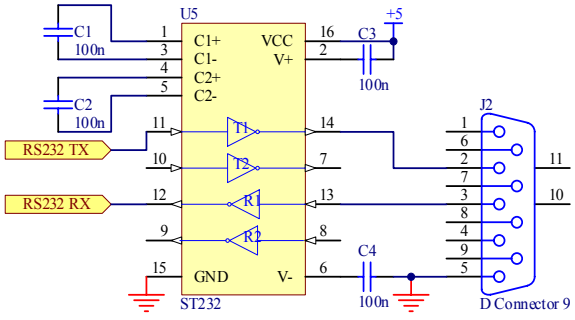


Connecteur (nappe) vers l'étage supérieur



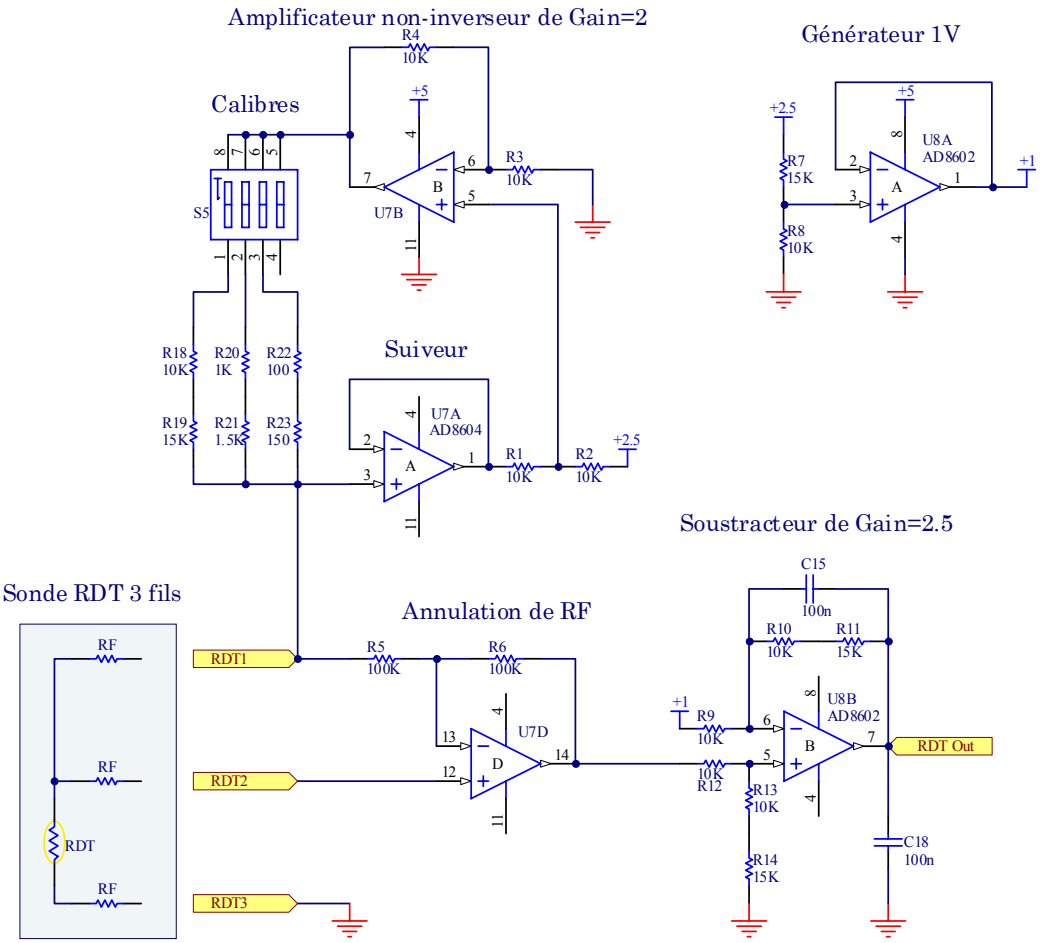
ST232 (TTL -> RS232)

Port COM

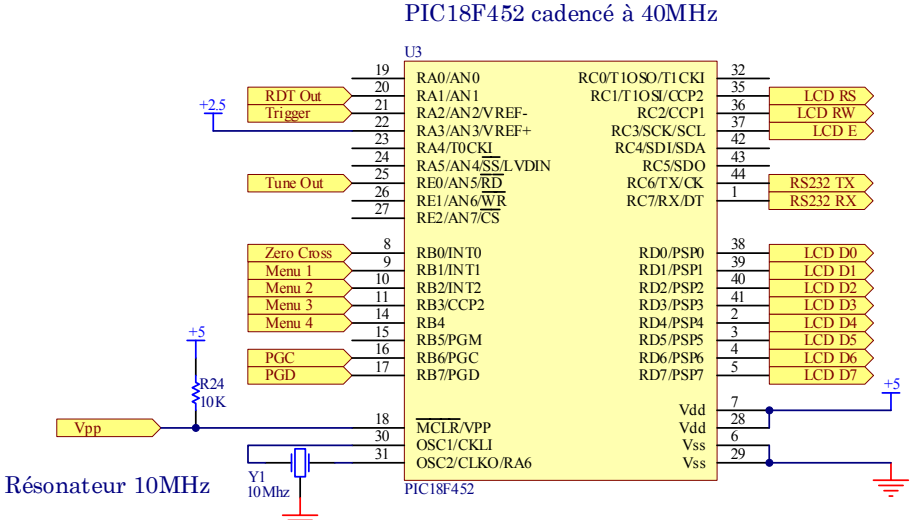


Connecteurs

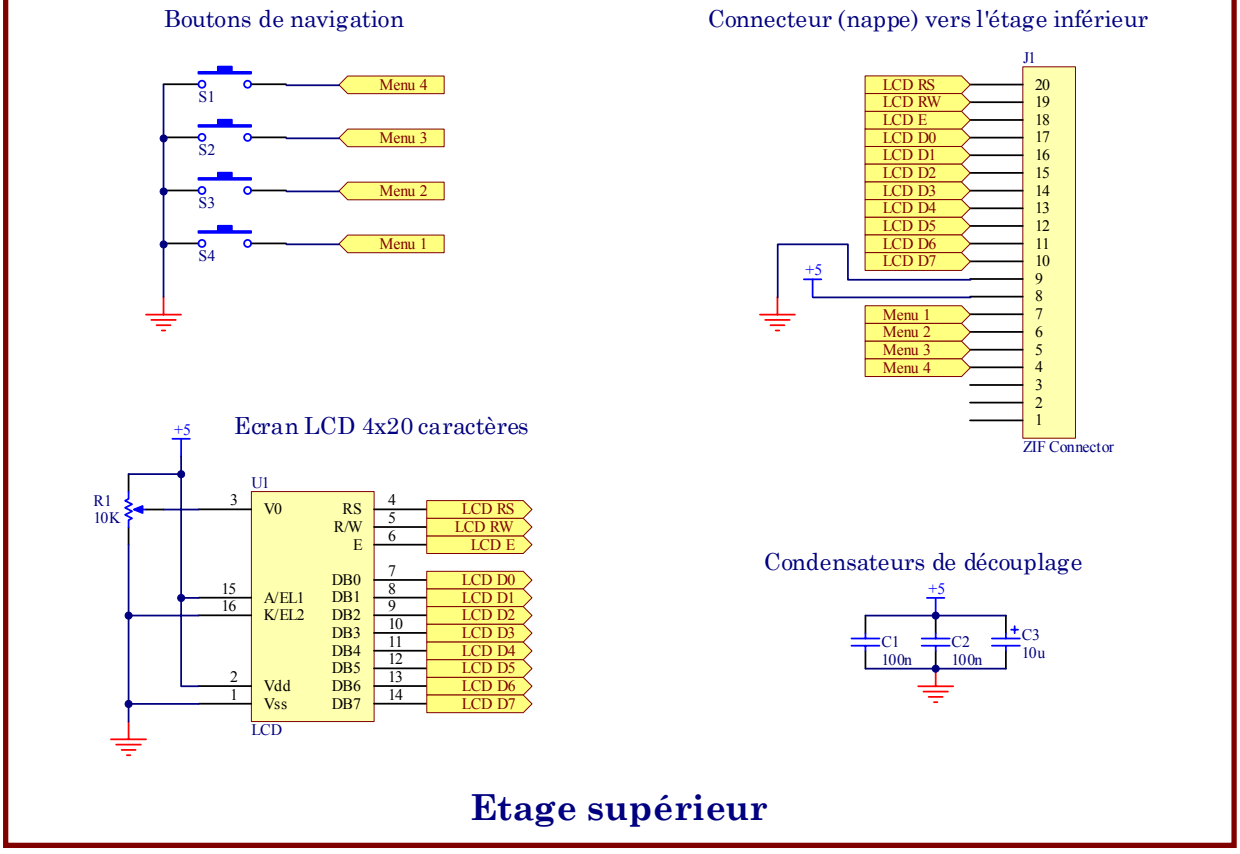
Générateur de courant



Etage Analogique



Microcontrôleur



SCHEMA ÉLECTRONIQUE (3)

L'interface RS232

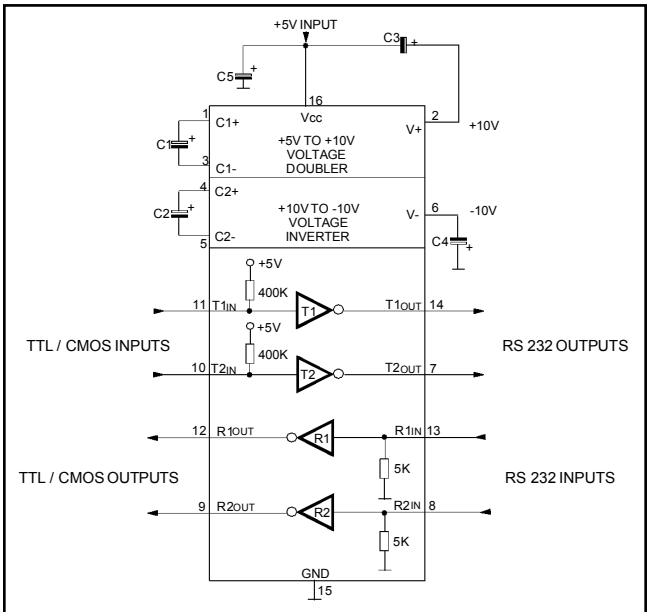
Pour communiquer avec un PC, j'utilise le protocole RS232 (H=-12V, L=12V, alors que la logique TTL est H=5V, L=0V). Pour cela il faut utiliser un driver de ligne comme le ST232 [ST2] et ses 4 condensateurs de 100nF. La liaison se fait à 19200baud avec CRC pour assurer l'intégritée des données [M3].

L'interface utilisateur

Le choix est large, j'ai choisi un afficheur alphanumérique rétroécléré bleu avec des lettres blanches 4x20 caractères. Ainsi que 4 boutons pour se déplacer facilement dans les menus pour des raisons de confort.

Le microcontroleur

Le choix est également très large. J'ai choisi le PIC18F452, microcontrôleur 8bits, cadencé à 40Mhz. La programmation de ce dernier se fait directement sur le circuit imprimé ce qui facilite le développement du firmware.

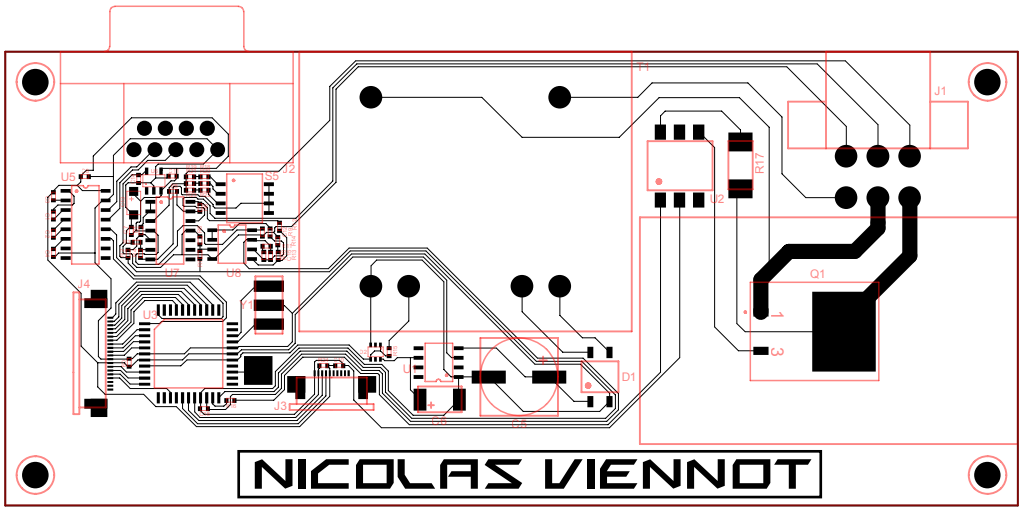


LE DRIVER DE LIGNE

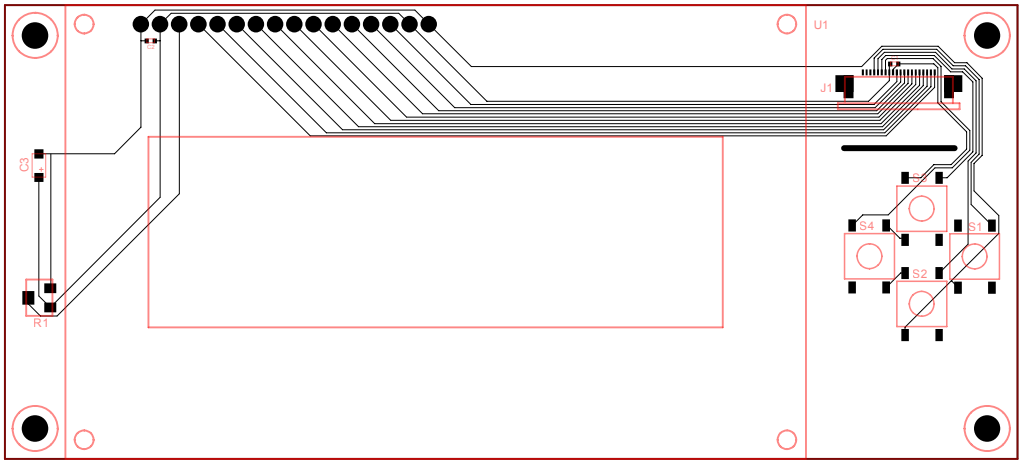
...:: LA RÉALISATION ::...

Les differentes etapes

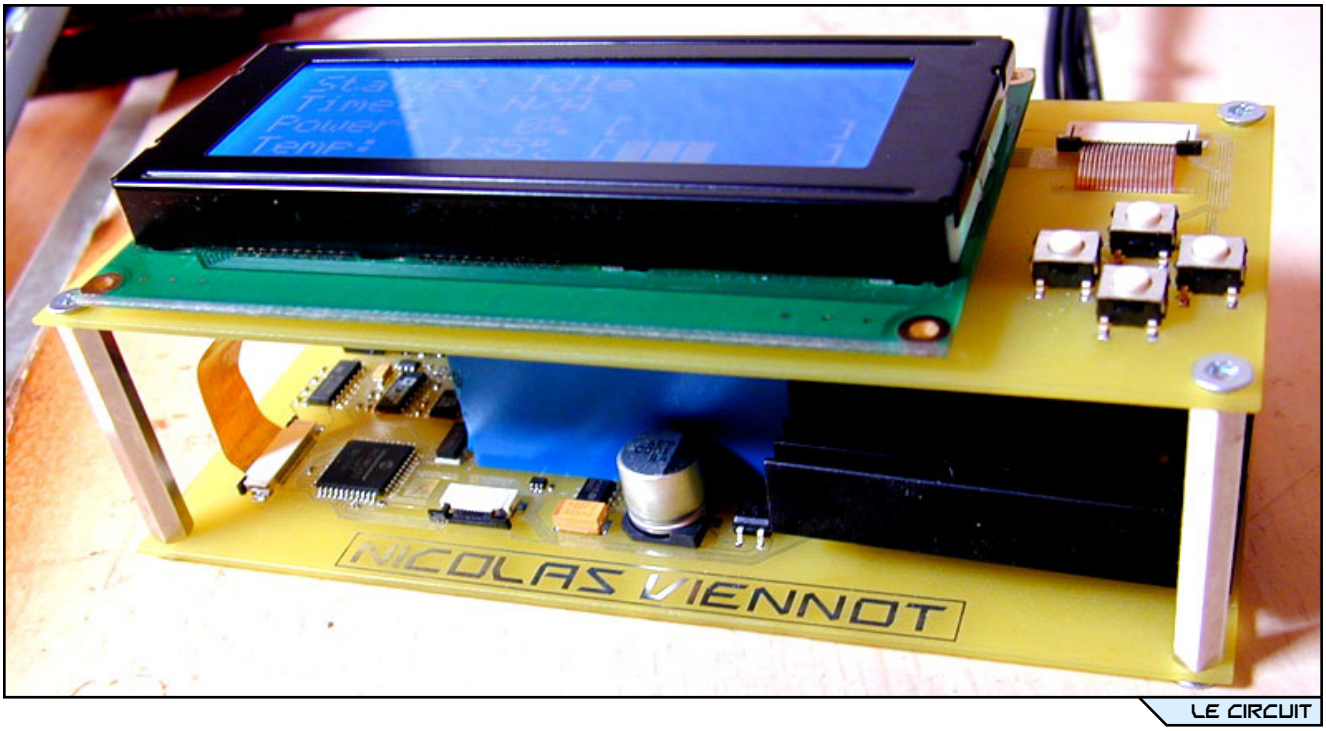
Pour réaliser un circuit imprimé, on passe généralement par 8 étapes. Tout d'abord, on imprime le typon sur un transparent (faire attention à l'échelle), puis on insole la plaquette cuivrée présensibilisée positive avec une insoleuse (on peut aussi le faire avec des lampes UV qui sont bien moins chères. On peut également transformer un vieux scanner en insoleuse avec des néon UV) et attention au sens du typon. Après l'insolation, il faut enlever le présensibilisé qui a été exposé aux UVs, pour cela on utilise du revelateur qui n'est rien d'autre que de la soude. Une fois le cuivre à enlever mis à nu, on procède à la gravue, traditionnellement fait avec du très salissant perchlorure de fer chaud. Ce procédé dure environ 5 minutes. Ensuite, il faut enlever le présensibilisé, pour cela on utilise de l'acétone ou de l'alcool à bruler. Ensuite, vient le perçage. Lorsque l'on pèrce avec des forêts HSS sur de l'époxy, ceux-ci s'usent très vite (environ 50 trous) et finissent par provoquer des cratères. Les forêts en carbure de tungstène sont adaptés et quasiment inusables mais ils sont très cassants (rien qu'en les faisant tomber par terre, ils cassent) et nécéssitent une vitesse de rotation de 30 000 tr/min. Ensuite, pour avoir un circuit protégé contre les oxydations et facile à souder, on plonge le circuit dans un bain d'étain à froid. Puis on soude les composants. Un petit fer à souder est obligatoire pour les composants CMS (Composants Montés en Surface). Ensuite vient la phase du test (moment critique de l'opération) ...



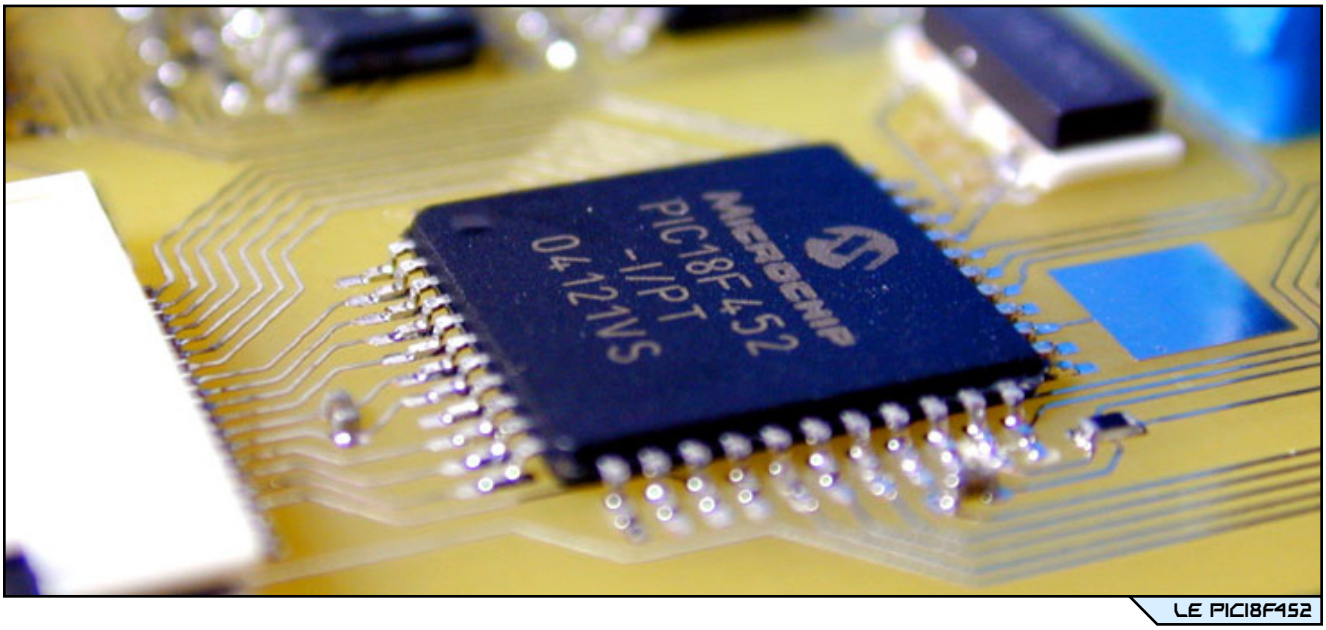
TYPON DE L'ÉTAGE INFÉRIEUR. ECHELLE 1:1



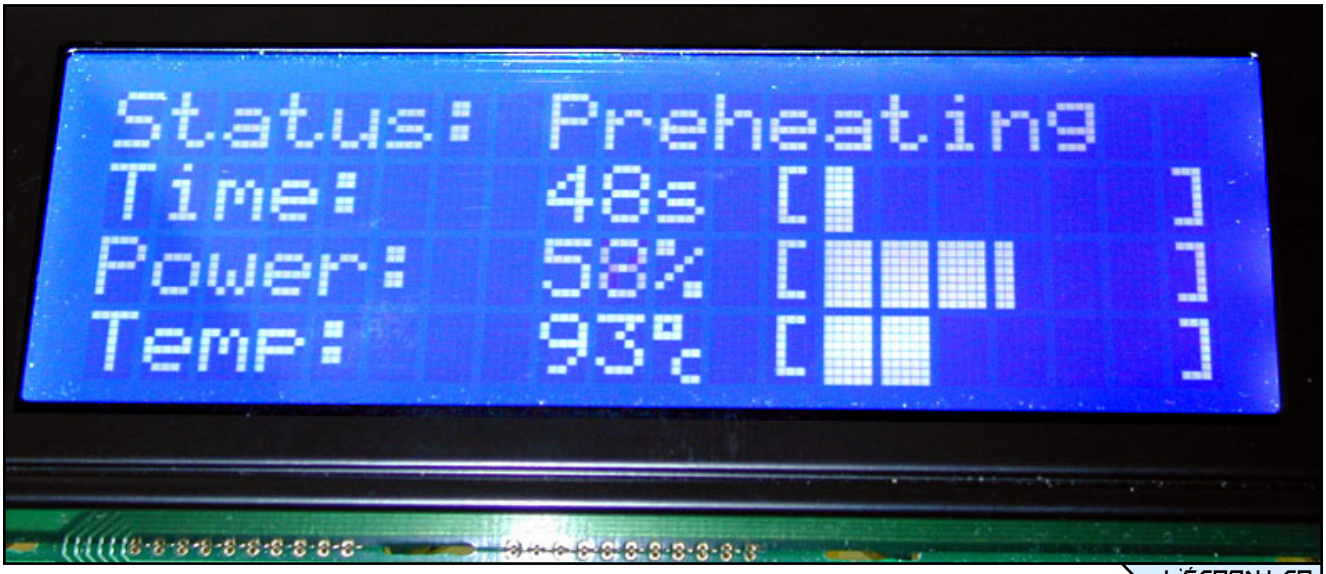
TYPON DE L'ÉTAGE SUPÉRIEUR. ECHELLE 1:1



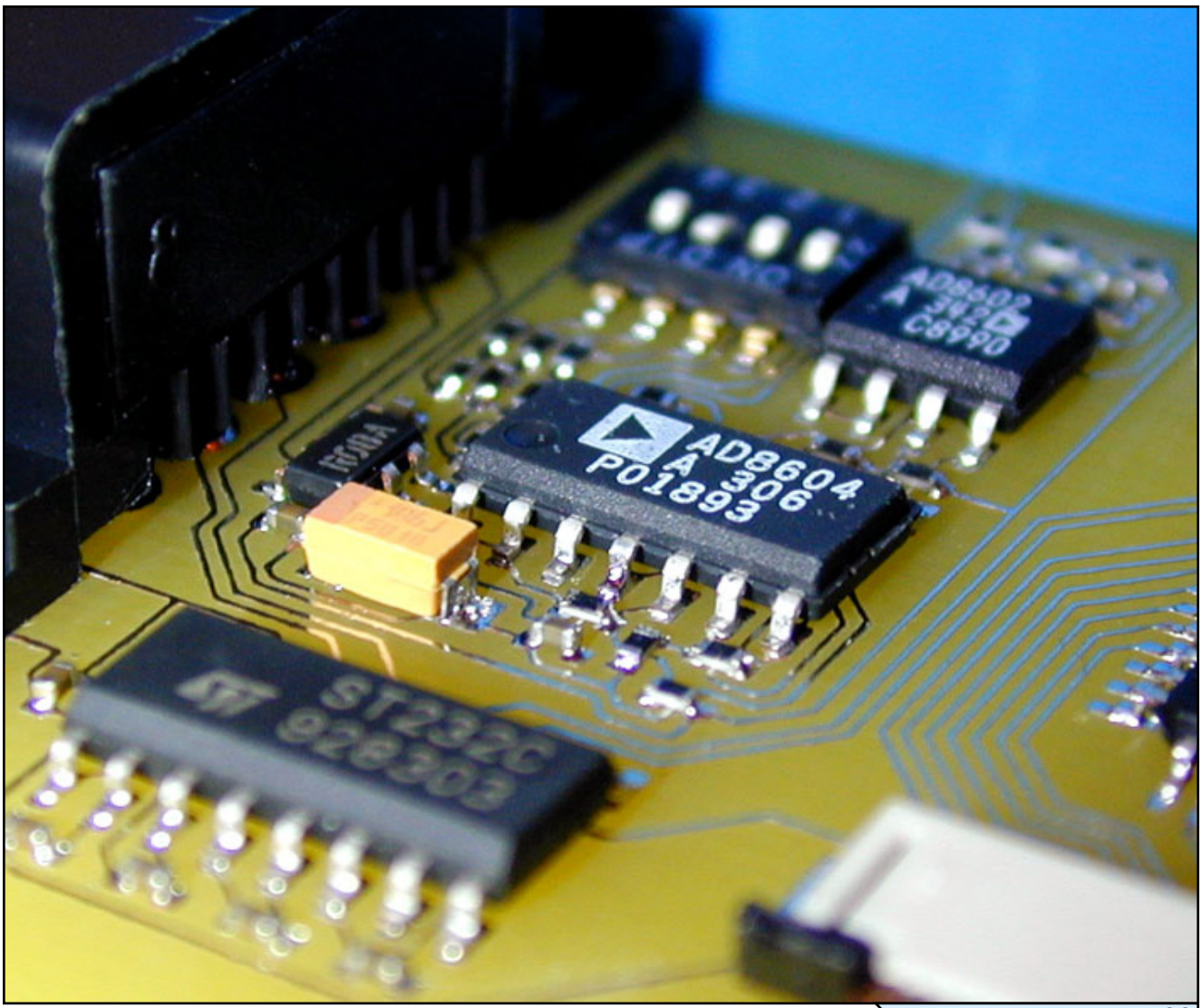
LE CIRCUIT



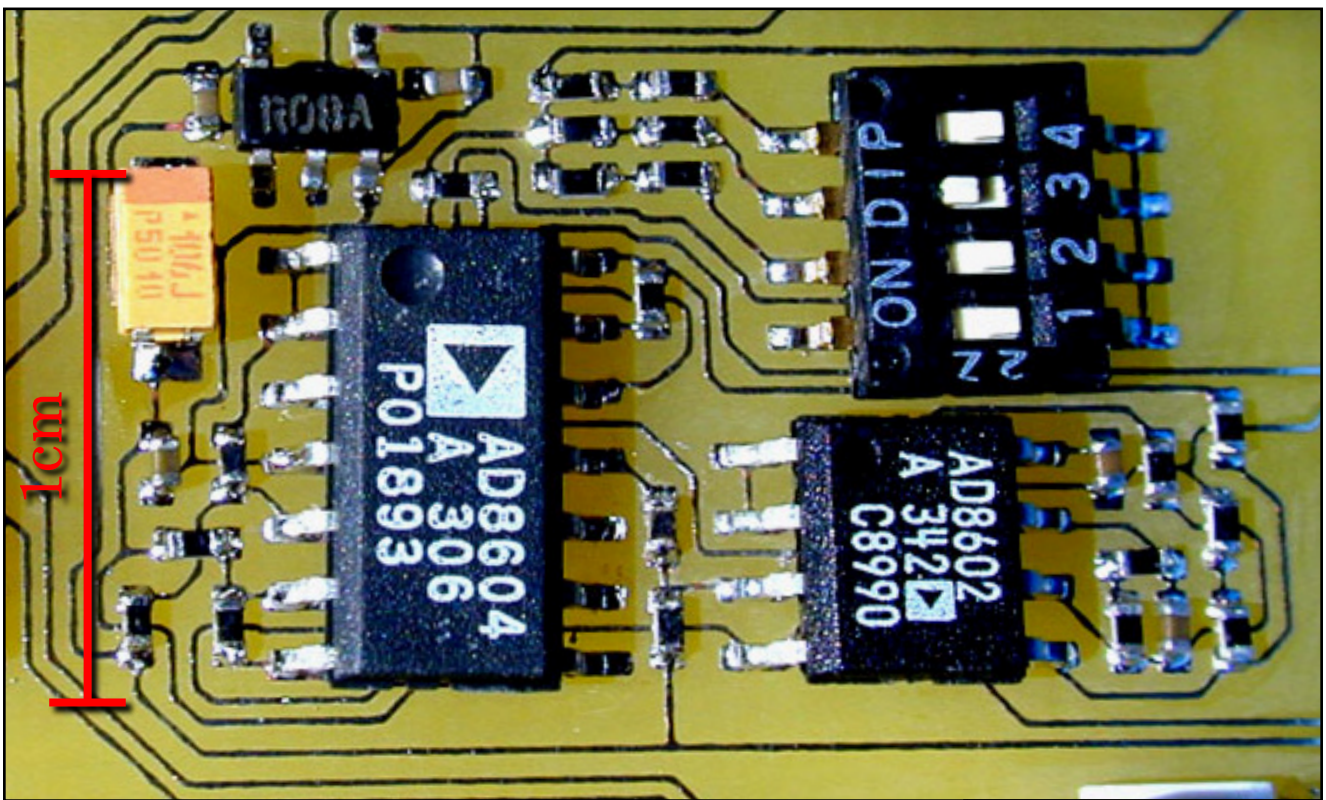
LE PIC18F452



L'ÉCRAN LCD



LA PARTIE ANALOGIQUE (1)

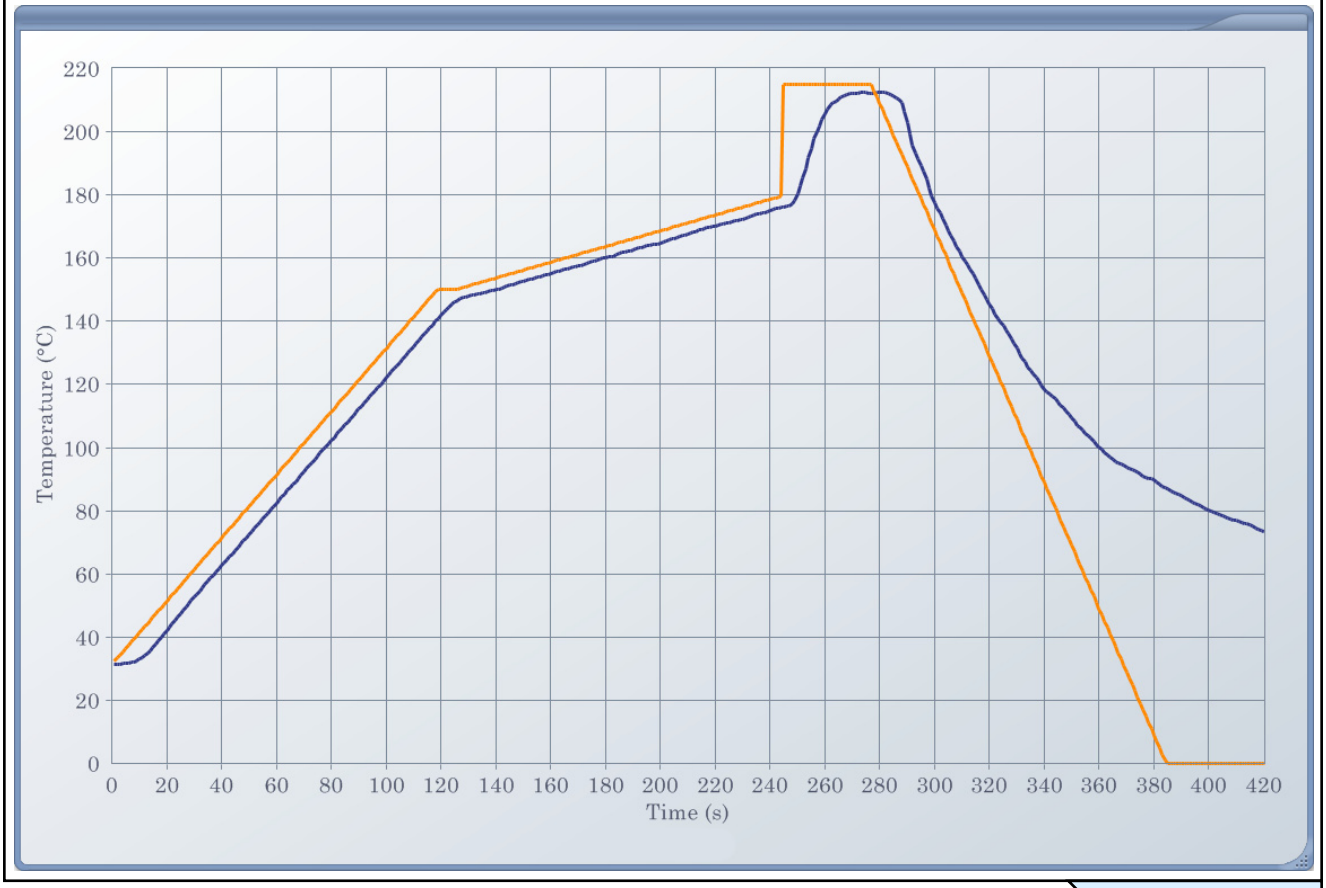


LA PARTIE ANALOGIQUE (2)

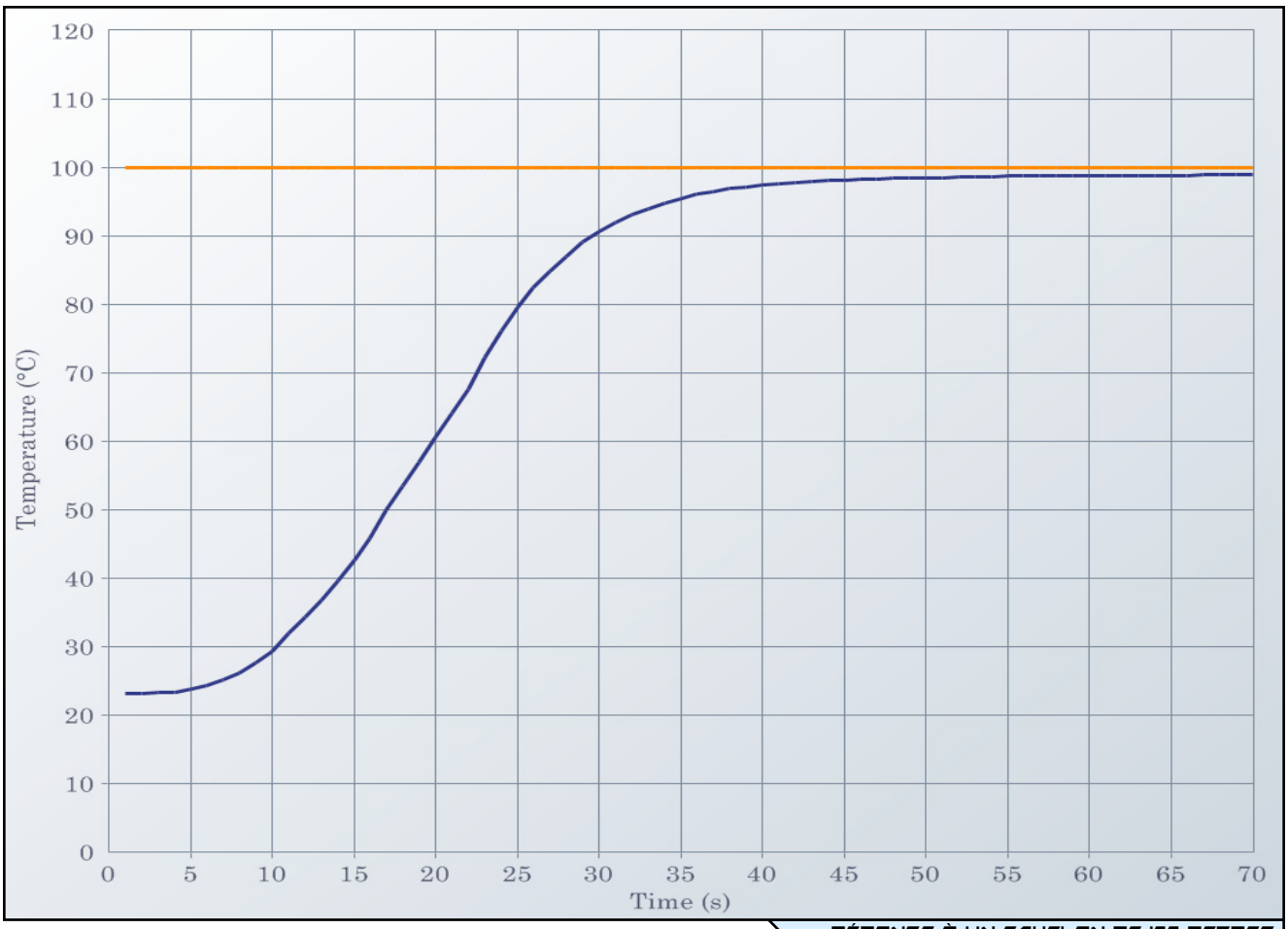
...:: LES RÉSULTATS EXPÉRIMENTAUX ::...

Interpretation des courbes

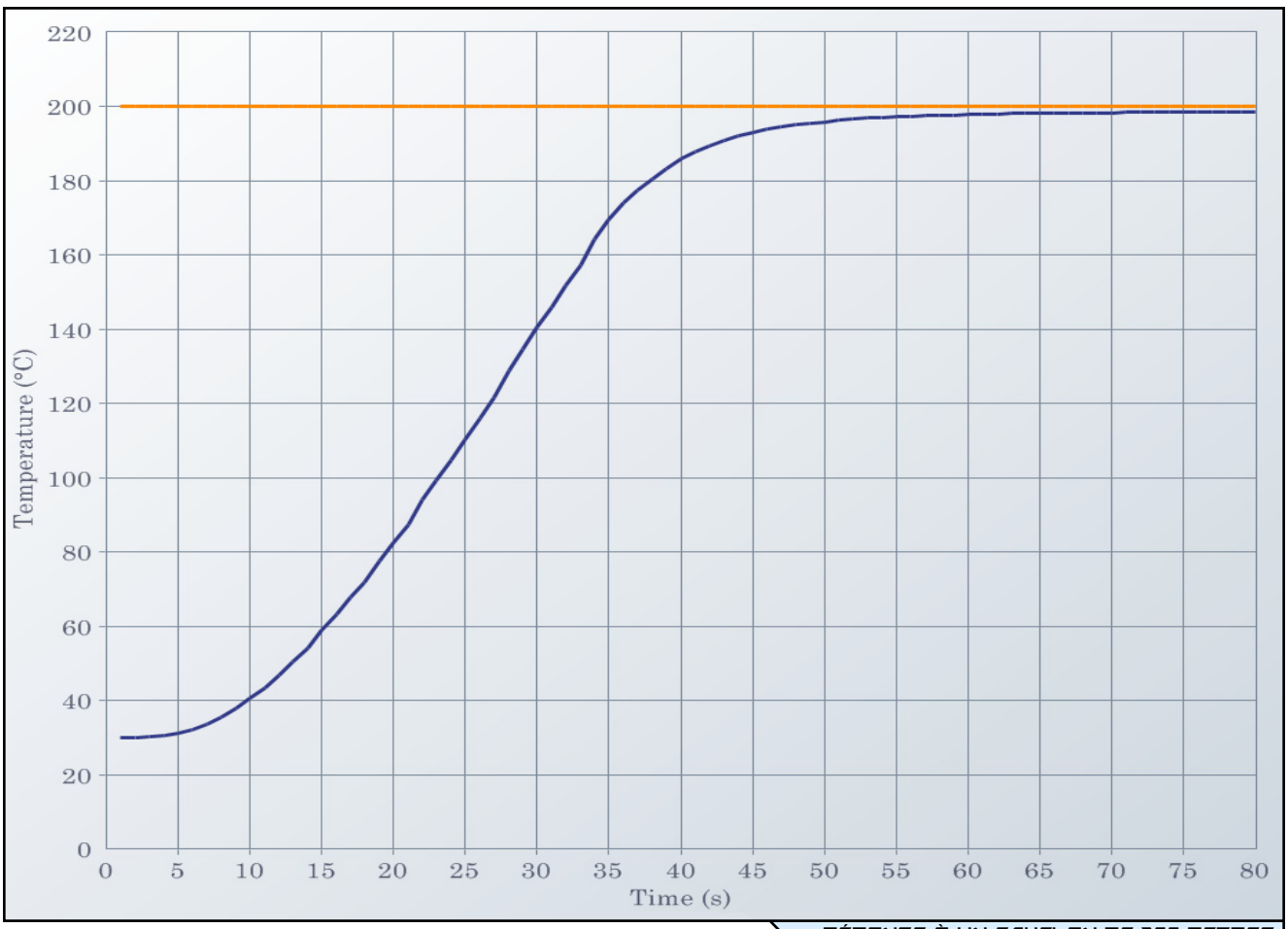
J'ai développé un logiciel PC qui trace les courbes de température en temps réel. En orange, on a la courbe de consigne et en bleu la courbe de température réelle.
L'asservissement fonctionne correctement: le profil obtenu est conforme au profil imposé dans le cahier des charges. Cependant, le majeur problème est le refroidissement: le système n'étant pas équipé de bouche d'aération, la descente en température ne peut être assurée automatiquement, il faut donc ouvrir la porte du four manuellement.
De plus sur les courbes en réponse à un échelon, on remarque un très bon temps de réponse avec une montée à 6°C/sec pour celle a 200°C. On a une erreur statique de d'environ 2°C. Pour y remedier, il faudrait utiliser une composante de type intégrale dans la boucle de l'asservissement.



PROFIL OBTENU



RÉPONSE À UN ECHELON DE 100 DEGRES



RÉPONSE À UN ECHELON DE 200 DEGRES

...:: LA NOMENCLATURE ::...

Produit	Constructeur	Ref. Constructeur	Prix unitée	Quantitée	Prix global	Ref. Distributeur	Distributeur
Element Chauffant infrarouge	Ceramicx Ireland Ltd	Full Quartz Element	42,71 €	2	85,42 €	376-2571	RS
Four	Alfredo	XU440W	39.99 €	1	39.99 €	N/A	Darty
Sonde RDT	Honeywell	HEL-707-U-0-12-00	23,46 €	1	23,46 €	293-8654	RS
Cosses Male 0.25"	N/A	N/A	0,029 €	8	0.232 €	433-129	RS
Cosses Femelle 0.25"	N/A	N/A	0,033 €	8	0.264 €	433-113	RS
Connecteur Puissance Male	Molex	39-01-2060	0,44 €	2	0,88 €	172-8951	RS
Connecteur Puissance Femelle	Molex	39-01-2061	0,66 €	1	0,66 €	172-9077	RS
Connecteur Puissance Femelle	Molex	1068	1,34 €	1	1,34 €	215-5843	RS
Contact Male HCS AWG 16	Molex	44478-3112 Mâles	0,363 €	4	1,45 €	324-8003	RS
Contact Femelle HCS AWG 16	Molex	44476-3112 Fem.	0,363 €	8	2,902 €	324-8025	RS
Contact Male AWG 24-18	Molex	39-00-0041 Mâles	0,225 €	2	0,450 €	172-9140	RS
Contact Femelle AWG 24-18	Molex	39-00-0039 Fem	0,173 €	4	0,691 €	172-9134	RS
gaine fibre de verre 1m	Croylek	2000402	0,68 €	1	0,68 €	398-745	RS
cable teflon 1.5mm² 1m	Alpha Wire	KZ04-07	1,037 €	1	1,037 €	365-919	RS
cable souple 1.0mm² 3m	Brand-Rex	N/A	0,1532 €	3	0,4596 €	356-729	RS
cable souple 0.5mm² 6m	Brand-Rex	N/A	0,0913 €	6	0,5478 €	356-448	RS
Plaque Epoxy 135x60	Kelan	141304	1,93 €	1	3,86 €	699421	Farnell
Etain Sn62Pb36Ag2 creme 1g	Multicore	SN62RA10BAS86-25G	0,76 €	1	0,76 €	149968	Farnell
Etain Sn62Pb36Ag2 fil 1g	Multicore	SMART26 250G REEL	0,19 €	1	0,19 €	419503	Farnell
Entretoises métalliques	Richco	HTSB-M3-35-5-1	0,229 €	4	0,916 €	280-8929	RS
Vis M3 X 6 tête fraisée Torx	N/A	N/A	0,025 €	8	0,202 €	449-6900	RS
Dissipateur thermique	Seifert	KL 224/25,4SW	1,44 €	1	1,44 €	218-2249	RS
Feuille thermique 1cm²	Bergquist	SILPAD400AC	0,037 €	1	0,037 €	169-2458	RS
Barrete Secable pas 2.54mm	Tyco/AMP	3-103321-6	0,051 €	16	0.813 €	973117	Farnell
LCD 4x20 caracteres	Crystalfontz	CFAH2004A-TMI-JP	23,85 €	1	23,85 €	N/A	CF
Port serie femelle sub-D9	ITT Cannon	ADE-9S-1A2N-A197	2,51 €	1	2,51 €	446-585	RS
Transformateur 2x220V/12V	Dagnall	D3008	4,93 €	1	4,93 €	159359	Farnell
Regulateur 5V 800mA	STmicroelectronic	LD1117D50	1,14 €	1	1,14 €	3529927	Farnell
Reference 2.5V ±0.2%	National semi	LM4120AIM5-2.5	1,90 €	1	1,90 €	3327097	Farnell
OptoTriac	Fairchild semi	MOC3021	1,45 €	1	1,45 €	885654	Farnell
Triac	Philips	BTA140B-600	3,28 €	1	3,28 €	936315	Farnell
Pont de diodes	Vishay General semi	MB6S	1,01 €	1	1,01 €	505821	Farnell
Commutateur/calibres	Multicomp	MCNHDS-04	2,17 €	1	2,17 €	566755	Farnell
Driver TTL.RS232	STmicroelectronic	ST232CD	1,34 €	1	1,34 €	3025019	Farnell
AmpliOp pack x4	Analog devices	AD8604AR	3,55 €	1	3,55 €	3884223	Farnell
AmpliOp pack x2	Analog devices	AD8602AR	2,67 €	1	2,67 €	3884211	Farnell
Bouton Poussoir	Omron	B3S-1000	0,66 €	4	2.64 €	177807	Farnell
Potentiometre 10K	Vishay sfernice	TSM4YJ 10K 10%	2,15 €	1	2,15 €	3065194	Farnell
Diodes Schottky x3	Central semi	CMKSH-3T	N/A	1	N/A	N/A	N/A
Microcontrolleur 40Mhz	Microchip	PIC18F452-I/PT	7,22 €	1	7,22 €	N/A	Microchip
Resonateur 10Mhz	AVX	PBRC-10.00HR	2,34 €	1	2,34 €	648164	Farnell
Canons metal 0.8MM	Grosvenor	CSS031 BAILS 0,8MM	0,061 €	9	0,549 €	463929	Farnell
Connecteur 0.5MM 10 voies	Molex	52745-1090	1,76 €	1	1,76 €	3078309	Farnell
Connecteur 0.5MM 20 voies	Molex	52745-2090	2,80 €	2	5,60 €	3078334	Farnell
Nappe 20 voies, 0.5mm	Multicomp	AXO-00008	2,21 €	1	2,21 €	3295035	Farnell
Condensateur 1000µF 10V	Nichicon	UUD1A102MNR1GS	0,78 €	1	0,78 €	3450211	Farnell
Condensateur 220µF 6V	AVX	TAJC227K006R	1,44 €	1	1,44 €	197087	Farnell
Condensateur 10µF 6V	AVX	TAJA106K006R	0,28 €	2	0,56 €	197014	Farnell
Condensateur 100nF 16V	AVX	CM05Y5V104Z76AH	0,034 €	15	0,510 €	316581	Farnell
Resistance ±5% 220R 220V	Phycomp	2322 762 60221	0,20 €	1	0,20 €	3258026	Farnell
Resistance ±5% 10K	Phycomp	2322 705 70103	0,007 €	2	0,02 €	195157	Farnell
Resistance ±5% 330	Phycomp	2322 705 70331	0,007 €	2	0,02 €	194979	Farnell
Resistance ±1% 100	Phycomp	RC-32H-100R-1P5	0,0308 €	1	0,0308 €	3603209	Farnell
Resistance ±1% 150	Phycomp	RC-32H-150R-1P5	0,0308 €	1	0,0308 €	3603222	Farnell
Resistance ±1% 1K	Phycomp	RC-32H-1K-1P5	0,0308 €	1	0,0308 €	3603325	Farnell
Resistance ±1% 1,5K	Phycomp	RC-32H-1K5-1P5	0,0308 €	1	0,0308 €	3603349	Farnell
Resistance ±1% 10K	Phycomp	RC-32H-10K-1P5	0,0308 €	10	0,3080 €	3603441	Farnell
Resistance ±1% 15K	Phycomp	RC-32H-15K-1P5	0,0308 €	4	0,1232 €	3603465	Farnell
Resistance ±1% 100K	Phycomp	RC-32H-100K-1P5	0,0308 €	2	0,0616 €	3603568	Farnell

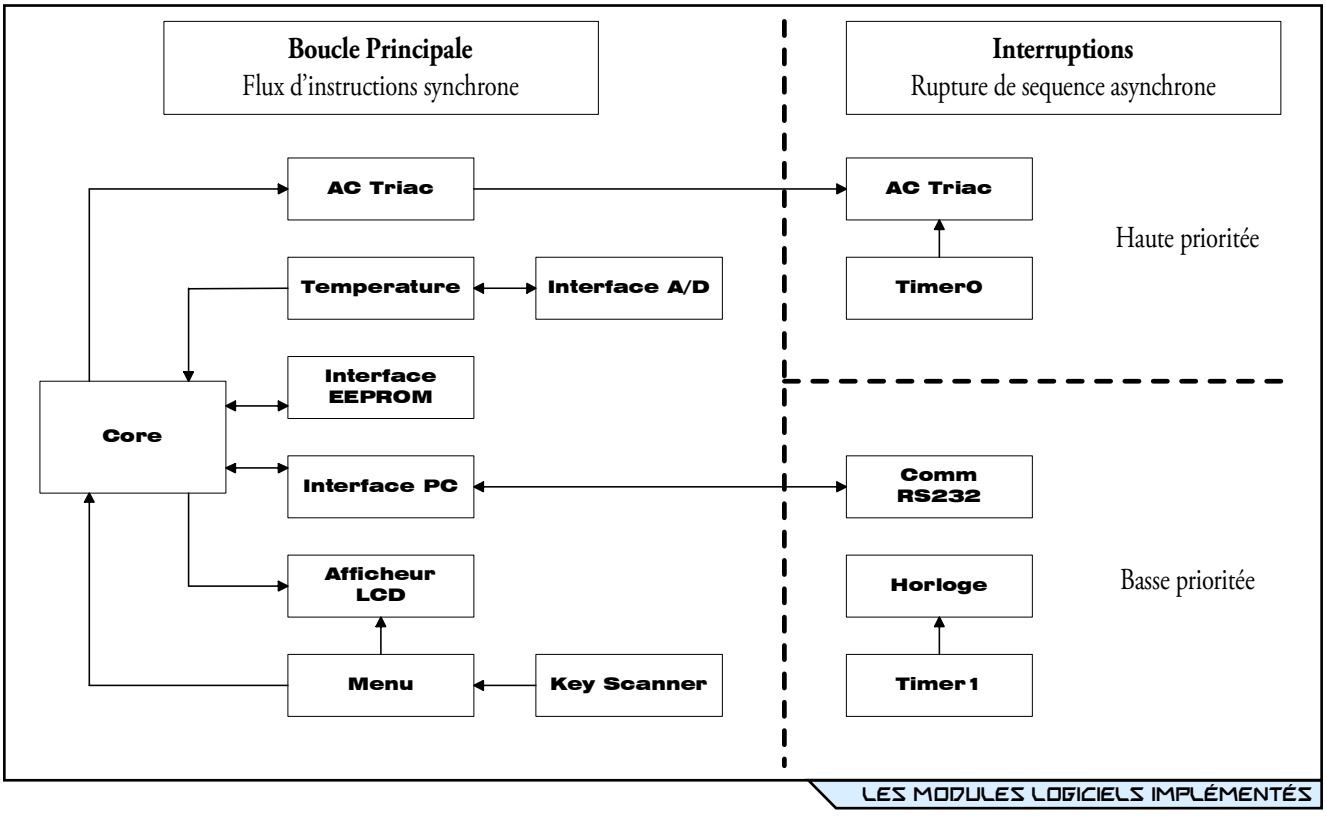
Total: 247,1 €

LA PARTIE LOGICIELLE

...:: DESCRIPTION GÉNÉRALE ::...

L'implementation

J'ai programmé le microcontrôleur en language C, nettement plus souple que l'assembleur.
La difficultée réside à traiter toutes les tâches simultanement (lancer une impulsion sur le Triac, mettre à jour l'écran, numeriser la tension de la sonde RDT).



LES MODULES LOGICIELS IMPLÉMENTÉS

...:: LA SOURCE DU MICROCONTROLEUR ::...

```
/*****************************************************************************
 *   Reflow Oven Firmware                                                    *
 *                                                                           *
 *   Copyright (C) 2004 by Nicolas Viennot                                   *
 *   nicolas.viennot@free.fr                                                 *
 *                                                                           *
 *   This program is free software; you can redistribute it and/or modify    *
 *   it under the terms of the GNU General Public License as published by    *
 *   the Free Software Foundation; either version 2 of the License, or       *
 *   (at your option) any later version.                                     *
 *                                                                           *
 *   This program is distributed in the hope that it will be useful,         *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of          *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the           *
 *   GNU General Public License for more details.                            *
 *                                                                           *
 *   You should have received a copy of the GNU General Public License       *
 *   along with this program; if not, write to the                           *
 *   Free Software Foundation, Inc.,                                         *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.               *

 *****************************************************************************/
```

ac.h

```
#ifndef __AC_H
#define __AC_H

// using Timer0 and Timer1


// pin defines
#define AC_TRIGGER RA2
#define AC_TRIGGER_DIR TRISA2

#define AC_ZERO RB0
#define AC_ZERO_DIR TRISB0
#define AC_ZERO_INTE INT0IE
#define AC_ZERO_INTF INT0IF

#define AC_SYNCH_OUT RE0
#define AC_SYNCH_OUT_DIR TRISE0

#define AC_SAMPLE 32

#define AC_ZERO_INCERTITUDE 3

void ACInit();
void ACSetPower( double power );
void ACMeasurePeriod();
void ACEnableTuning( char enable );

extern unsigned int ACLookUp[];

#define DUMP_COMPENSATION 4
#define DumpTMR0ToACTemp()\
{\
        asm( "movf _TMR0L, W");\
        asm( "movwf _ACTemp");\
        asm( "movf _TMR0H, W");\
        asm( "movwf (_ACTemp) + 1");\
}

#define DumpACTempToTMR0()\
{\
        asm( "movf (_ACTemp) + 1, W");\
        asm( "movwf _TMR0H");\
        asm( "movf _ACTemp, W");\
        asm( "movwf _TMR0L");\
}

#define DumpACTempToTMR1()\
{\
        asm( "movf (_ACTemp) + 1, W");\
        asm( "movwf _TMR1H");\
        asm( "movf _ACTemp, W");\
        asm( "movwf _TMR1L");\
}


#endif
```

ac.c

```
#include "always.h"
#include "ac.h"
#include "lcd.h"
#include "adc.h"
#include "eeprom.h"

#include <string.h>
#include <stdio.h>
#include <math.h>

near unsigned int ACTemp;
near unsigned int ACHalfPeriod;
near unsigned int ACTrigger = 0;
near bit ACDisableTrigger;
bit ACTunning;
```

```
void ACInit()
{
        AC_ZERO_DIR = 1;
        asm("nop");
        AC_TRIGGER_DIR = 0;
        asm("nop");
        AC_TRIGGER = 0;
        asm("nop");
        AC_SYNCH_OUT_DIR = 1;

        T0CON  = 0b10000010;   // 1:8 prescaler, 16bits
        TMR0IP = 1;            // high priority
        TMR0IE = 1;            // timer0 interrupt

        T1CON  = 0b10110001;   // 1:8 prescaler
        TMR1IP = 1;            // high priority

        AC_ZERO_INTE = 1;      // interrupt on zero cross pin

        ACMeasurePeriod();

        ACSetPower(0.0);
}

void ACEnableTuning( char enable )
{
        ACTunning = enable;
        if( enable ) AC_SYNCH_OUT_DIR = 0;
        else AC_SYNCH_OUT_DIR = 1;
}

void ACMeasurePeriod()
{
        unsigned long Avg = 0;
        char loop = AC_SAMPLE;

        sprintf( LCDLines[0], "Reflow Oven" );
        sprintf( LCDLines[1], "Version: %s", VERSION );
        strcpy( LCDLines[2], "AC Initializing..." );
        LCDUpdate();

        ACTemp = 0;

        // sync with timer0
        AC_ZERO_INTF = 0;
        while( !AC_ZERO_INTF );

        TMR0H = 0;
        TMR0L = 3;

        while( loop-- )
        {
                AC_ZERO_INTF = 0;
                while( !AC_ZERO_INTF );

                DumpTMR0ToACTemp();

                TMR0H = 0;
                TMR0L = 3 + DUMP_COMPENSATION;

                Avg += ACTemp;
        }

        ACHalfPeriod = Avg / ( AC_SAMPLE*2 );

        strcpy( LCDLines[2], "AC Initialized." );
        sprintf( LCDLines[3], "Frequence: %.3fHz", 1000.0/(((double)ACHalfPeriod)*8.0*100e-6*2.0));

        LCDUpdate();
        DelayS( 3 );
}

void ACSetPower( double power )
{
        unsigned long temp;
```

```
        // incertitude threshold
        if( power > 100 - AC_ZERO_INCERTITUDE )
        {
                ACDisableTrigger = 1;
                AC_TRIGGER = 1;
                return;
        }
        if( power < AC_ZERO_INCERTITUDE )
        {
                ACDisableTrigger = 1;
                AC_TRIGGER = 0;
                return;
        }
        power = (100.0 - power)/100.0;
        power = asin( sqrt( power ))*0.55+0.08;

        temp = (unsigned long)(power * (double)ACHalfPeriod);

        ACTrigger = temp;
        AC_SYNCH_OUT_DIR = 1;
        ACDisableTrigger = 0;
}

void interrupt HighPriorityISR()
{
        if( TMR0IF )
        {
                char temp;
                TMR0IF = 0;

                if( !ACDisableTrigger )
                {
                        AC_TRIGGER = 1;
                        // delay
                        for( temp = 100; temp != 0; temp-- )
                                asm("nop");
                        AC_TRIGGER = 0;
                }
        }

        else if( AC_ZERO_INTF )
        {
                AC_ZERO_INTF = 0;
                INTEDG0 = !INTEDG0;

                ACTemp = -( ACTrigger );
                DumpACTempToTMR0();
        }
}
```

adc.h

```
#ifndef __ADC_H
#define __ADC_H

#define ADC_SAMPLE 255

void ADCInit();
void ADCSelectChannel( char channel );
unsigned int ADCRead();

#endif
```

adc.c

```
#include "always.h"
#include "adc.h"

void ADCInit()
{
        ADCON1 = 0b11000101;            // Vref+ = AN3
                                        // Vref- = Vss
                                        // AN0 & AN1 analog mode
}

void ADCSelectChannel( char channel )
{
        channel &= 0x07;
```

```
        ADCON0 = 0b10000001 | ( channel << 3 );      // Conversion Clock: Fosc/64
}

unsigned int ADCRead()
{
        unsigned char loop = ADC_SAMPLE;
        unsigned long result = 0;

        while( loop-- )
        {
                DelayUs(15); // Wait Tacq
                GODONE = 1;
                while( GODONE );
                result += ADRESL | ( ADRESH << 8 );
        }

        result /= ADC_SAMPLE;

        return result & 0xFFFF;
}
```

always.h

```
#ifndef __ALWAYS_H
#define __ALWAYS_H

#define PIC_CLK         40000000
#define VERSION "0.9.1205"

#include <pic18.h>
#include "delay.h"

#endif
```

characters.h

```
#ifndef __CHARACTERS_H
#define __CHARACTERS_H

extern const char CharBars[];

#define CHAR_BARS_0                     0x20
#define CHAR_BARS_1                     0x08
#define CHAR_BARS_2                     0x01
#define CHAR_BARS_3                     0x02
#define CHAR_BARS_4                     0x03
#define CHAR_BARS_5                     0xFF

#define CHAR_DEG                        0x04
#define CHAR_PERS                       0x05

#define CHAR_ARROW_UP                   0x06
#define CHAR_ARROW_LEFT                 0x06
#define CHAR_ARROW_DOWN                 0x07
#define CHAR_ARROW_RIGHT                0x07

#define CHAR_STRAIGHT_ARROW_RIGHT       0x7E
#define CHAR_STRAIGHT_ARROW_LEFT        0x7F

#endif __CHARACTERS_H
```

cmdhandler.h

```
#ifndef __CMDHANDLER_H
#define __CMDHANDLER_H

void ProcessRX();

void UpdateStatus();
void RunStarted();

#define CMD_SETPOWER                    0x00
#define CMD_UPDATE_STATUS               0x01
#define  CMD_RUN_STARTED                0x02

#endif
```

cmdhandler.c

```
#include "always.h"
#include "cmdhandler.h"
#include "comm.h"
#include "ac.h"
#include "core.h"
#include <string.h>
#include <stdio.h>

void ProcessRX()
{
        switch( RXBuffer[0] )
        {
                case CMD_SETPOWER:
                        ACSetPower( (double)RXBuffer[1]*100.0/255.0 );
                break;
        }
}

void RunStarted()
{
        TXLength = 9;
        TXBuffer[0] = CMD_RUN_STARTED;
        BeginTX();
}

void UpdateStatus()
{
        union {
                double d;
                long l;
        } temp;

        temp.d = CoreTempActual;

        TXLength = 11;
        TXBuffer[0] = CMD_UPDATE_STATUS;

        TXBuffer[1] = temp.l & 0xFF;
        TXBuffer[2] = (temp.l >> 8 )& 0xFF;
        TXBuffer[3] = (temp.l >> 16 )& 0xFF;
        TXBuffer[4] = (temp.l >> 24 )& 0xFF;

        temp.d = CoreTempSet;

        TXBuffer[5] = temp.l & 0xFF;
        TXBuffer[6] = (temp.l >> 8 )& 0xFF;
        TXBuffer[7] = (temp.l >> 16 )& 0xFF;
        TXBuffer[8] = (temp.l >> 24 )& 0xFF;

        TXBuffer[9] = CoreTime & 0xFF;
        TXBuffer[10] = ( CoreTime >> 8 ) & 0xFF;

        BeginTX();
}
```

comm.h

```
#ifndef __COMM_H
#define __COMM_H

#define COMM_BUFFER_SIZE 15

void CommInit();
void InterruptRX();
void InterruptTX();

void CheckRX();
#define WaitTX() { while( TXIE || !TRMT ) asm("clrwdt"); }
#define BeginTX() { WaitTX(); TXIE = 1; }

extern bit CommCancelRX;      // reset state (used to timeout the reception)
extern near char RXLength, TXLength;
extern char RXBuffer[ COMM_BUFFER_SIZE ];
extern char TXBuffer[ COMM_BUFFER_SIZE ];

#endif
```

comm.c

```
#include "always.h"
#include "comm.h"
#include "cmdhandler.h"

const char Header[] = { 0x13, 0x37 };
char RXBuffer[ COMM_BUFFER_SIZE ];
char TXBuffer[ COMM_BUFFER_SIZE ];
near char RXLength, TXLength;
bit CommCancelRX = 0;
bit CommNewRX;

void CommInit()
{
        RCSTA = 0b10010000;
        TXSTA = 0b00100100;    // High Speed
                               // Transmission enabled

        SPBRG = 129;           // 40e6/(16*(129+1)) =  19230bps

        RCIE = 1;              // enable low priority interrupts
        TXIE = 0;
        RCIP = 0;
        TXIP = 0;
}

unsigned int UpdateCRC( unsigned int crc, char data )
{
        crc = (unsigned char)(crc >> 8) | (crc << 8);
        crc ^= data;
        crc ^= (unsigned char)(crc & 0xff) >> 4;
        crc ^= (crc << 8) << 4;
        crc ^= ((crc & 0xff) << 4) << 1;
        return crc;
}

void CheckRX()
{
        if( CommNewRX )
        {
                CommNewRX = 0;
                ProcessRX();
                CREN = 1;
        }
}

#include "ac.h"
void InterruptRX()
{
        near static char offset;
        static unsigned int CRC;
        static bit headerReceived = 0;
        char RX;

        // handle overflow error
        if( OERR )
        {
                  // reset connection
                CREN = 0;
                CREN = 1;
                CommCancelRX = 1;
                return;
        }

        if( CommCancelRX )
        {       // reset state
                CommCancelRX = 0;
                headerReceived = 0;
                offset = 0;
        }

        RX = RCREG;
        if( !headerReceived )  // new reception
        {

                if( offset == sizeof( Header ) )
```

```
                {
                        RXLength = RX;
                        if( RXLength > COMM_BUFFER_SIZE )
                                CommCancelRX = 1;

                        CRC = UpdateCRC( 0, RX );
                        headerReceived = 1;

                        // reset ptr for data
                        offset = 0;
                }

                else
                {
                        // valid header ?
                        if( Header[ offset ] != RX )
                                CommCancelRX = 1;
                        offset++;
                }
        }

        else
        {
                if( offset == RXLength )
                {
                        if( CRC & 0xFF != RX  )
                                CommCancelRX = 1;
                }

                else if( offset == RXLength + 1 )
                {
                        if( CRC >> 8 == RX )
                        {
                                CommNewRX = 1;

                                // disable reception
                                CREN = 0;
                        }
                        CommCancelRX = 1;
                }

                else
                {
                        RXBuffer[ offset ] = RX;
                        CRC = UpdateCRC( CRC, RX );
                }

                offset++;
        }
}

void InterruptTX()
{
        static bit headerTransmitted = 0;
        static near offset = 0;
        static unsigned int CRC;

        // header not transmitted yet
        if( !headerTransmitted )
        {
                if( offset == sizeof( Header ))
                {
                        TXREG = TXLength;
                        CRC = UpdateCRC( 0, TXLength );
                        headerTransmitted = 1;
                        offset = 0;
                }
                else
                {
                        TXREG = Header[ offset ];
                        offset++;
                }
        }

        else
        {
                // end of data
```

```
                if( offset == TXLength )
                {
                        TXREG = CRC & 0xFF;
                        offset++;
                }

                else if( offset == TXLength + 1 )
                {
                        TXREG = CRC >> 8;
                        headerTransmitted = 0;
                        offset = 0;
                        TXIE = 0;
                }

                else
                {
                        // Send data
                        TXREG = TXBuffer[ offset ];
                        CRC = UpdateCRC( CRC, TXBuffer[ offset ] );
                        offset++;
                }
        }
}
```

core.h

```
#ifndef __CORE_H
#define __CORE_H

// status
#define CORE_IDLE                               0
#define CORE_CONSTANT_POWER                     1
#define CORE_CONSTANT_TEMP                      2
#define CORE_PREHEATING                         3
#define CORE_DRYING                             4
#define CORE_HEATING                            5
#define CORE_REFLOW                             6
#define CORE_COOLING                            7


// typdef
typedef struct CoreProfile_
{
        double PreHeatSlope;
        double DryingTemp;
        int DryingTime;
        double SoakTemp;
        double ReflowTemp;
        int ReflowTime;
        double CoolingSlope;
} CoreProfile;

typedef struct CoreCoefficientsStruct_
{
        double Kp;
        double Kd;
} CoreCoefficientsStruct;

// maxVar
#define CORE_MAX_POWER                  100.0
#define CORE_MAX_TEMPERATURE            260
#define CORE_MAX_GRAD_TEMPERATURE       2

#define CORE_GRAD_SAMPLES               5
#define CORE_TEMP_SAMPLES               5

#define CORE_TEMP_THRESHOLD             4.0

extern char CoreStatus;
extern volatile int CoreTime;
extern int CoreMaxTime;
extern double CorePower;
extern double CoreTempActual;
extern double CoreTempGrad;
extern double CoreTempSet;
extern double CoreTempError;

extern CoreCoefficientsStruct CoreCoefficients;
```

```
extern CoreProfile CoreCurrentProfile;

#define CORE_TICKS_INTERVAL           1000 // ms
extern bit CoreUpdateTicksFlag;

// functions
void CoreUpdateTicks();
void CoreDisplayStatus();
void CoreUpdateTemperature( double temperature );
void CoreStart( char mode );
void CoreStop();

#endif
```

core.c

```
#include "core.h"
#include "temperature.h"
#include "ac.h"
#include "menu.h"
#include "lcd.h"
#include "timer.h"
#include "characters.h"
#include "cmdhandler.h"
#include <stdio.h>
#include <string.h>

char CoreStatus = CORE_IDLE;

volatile int CoreTime;
int CoreMaxTime = 400;
double CorePower;
double CoreTempActual;
double CoreTempGrad;
double CoreTempSet;
double CoreTempError;
bit CoreUpdateTicksFlag;

CoreCoefficientsStruct CoreCoefficients = { 12.0, 6.0 };
CoreProfile CoreCurrentProfile = { 1.0, 150.0, 120, 180.0, 215.0, 10, -2.0 };

void CoreDisplayStatus()
{
        if( !MenuBusy )
        {
                if( MenuStatus == MENU_STATUS_GENERAL )
                {
                        const char* status;

                        if( CoreStatus == CORE_IDLE ) status = "Idle";
                        else if( CoreStatus == CORE_CONSTANT_POWER ) status = "Cte Power";
                        else if( CoreStatus == CORE_CONSTANT_TEMP ) status = "Cte Temp";
                        else if( CoreStatus == CORE_PREHEATING ) status = "Preheating";
                        else if( CoreStatus == CORE_DRYING ) status = "Drying";
                        else if( CoreStatus == CORE_HEATING ) status = "Heating";
                        else if( CoreStatus == CORE_REFLOW ) status = "Reflow";
                        else if( CoreStatus == CORE_COOLING ) status = "Cooling";

                        sprintf( LCDLines[0], "Status: %s", status );

                        if( CoreStatus > CORE_CONSTANT_TEMP )
                                sprintf( LCDLines[1], "Time: %4ds %8s", CoreTime,\
                                        MenuGetBar( CoreTime, CoreMaxTime, 8 ));
                        else if( CoreStatus == CORE_IDLE ) strcpy( LCDLines[1], "Time:   N/A" );
                        else sprintf( LCDLines[1], "Time: %4ds", CoreTime );

                        sprintf( LCDLines[2], "Power: %3d%% %8s", (int)CorePower,\
                                MenuGetBar( (int)CorePower, (int)CORE_MAX_POWER, 8 ));

                        sprintf( LCDLines[3], "Temp: %4d%c %8s",      (int)CoreTempActual,\
                                CHAR_DEG, MenuGetBar( (int)CoreTempActual, CORE_MAX_TEMPERATURE, 8 ));
                }


                else if( MenuStatus == MENU_STATUS_CONTROL )
                {
                        strcpy( LCDLines[0], "Not Implemented" );
                        LCDLines[1][0] = 0;
```

```
                        LCDLines[2][0] = 0;
                        LCDLines[3][0] = 0;
                        /*
                        sprintf( LCDLines[0], "P: %7d%% %8s", (int)CoreControl.P,\
                                MenuGetBar( (int)CoreControl.P, (int)CORE_MAX_POWER, 8 ));

                        sprintf( LCDLines[1], "I: %7d%% %8s", (int)CoreControl.I,\
                                MenuGetBar( (int)CoreControl.I, (int)CORE_MAX_POWER, 8 ));

                        sprintf( LCDLines[2], "D: %7d%% %8s", (int)CoreControl.D,\
                                MenuGetBar( (int)CoreControl.D, (int)CORE_MAX_POWER, 8 ));

                        sprintf( LCDLines[3], "Power: %3d%% %8s", (int)CorePower,\
                                MenuGetBar( (int)CorePower, (int)CORE_MAX_POWER, 8 ));
                        */
                }

                else if( MenuStatus == MENU_STATUS_TEMP )
                {
                        sprintf( LCDLines[0], "TSet: %4d%c %8s", (int)CoreTempSet,\
                                CHAR_DEG, MenuGetBar( (int)CoreTempSet, CORE_MAX_TEMPERATURE, 8 ));

                        sprintf( LCDLines[1], "Temp: %4d%c %8s", (int)CoreTempActual,\
                                CHAR_DEG, MenuGetBar( (int)CoreTempActual, CORE_MAX_TEMPERATURE, 8 ));

                        sprintf( LCDLines[2], "TErr: %4d%c %8s",(int)CoreTempError,\
                                CHAR_DEG, MenuGetBar( (int)CoreTempError, CORE_MAX_TEMPERATURE, 8 ));

                        if( CoreTempGrad < -9.9 )
                                sprintf( LCDLines[3], "Tgrd:%4.0f%c%c %8s", CoreTempGrad,\
                                CHAR_DEG, CHAR_PERS, MenuGetBar( (long)(CoreTempGrad*1000.0),\
                                CORE_MAX_GRAD_TEMPERATURE*1000, 8 ));
                        else sprintf( LCDLines[3], "Tgrd:%4.1f%c%c %8s", CoreTempGrad,\
                                CHAR_DEG, CHAR_PERS, MenuGetBar( (long)(CoreTempGrad*1000.0),\
                                CORE_MAX_GRAD_TEMPERATURE*1000, 8 ));
                }

                LCDUpdate();
        }
}

void CoreUpdateVars()
{
        double CoreTempEstimation = CoreTempActual + CoreTempGrad * CoreCoefficients.Kd;
        double CoreTempErrorEstimation = CoreTempSet - CoreTempEstimation;
        double temp;
        CoreTempError = CoreTempSet - CoreTempActual;

        if( CoreTempErrorEstimation < 0.0 )
                temp = 0.0;

        else
        {
                temp = CoreCoefficients.Kp * CoreTempErrorEstimation;

                if( temp > CORE_MAX_POWER )
                        temp = CORE_MAX_POWER;
        }

        CorePower = temp;
        ACSetPower( CorePower );
}

void CoreUpdateTicks()
{
        static int oldTime;
        static double startingTemp;

        if( CoreStatus > CORE_CONSTANT_TEMP )
{

        //CoreTime++; done in timer.c

        // update CoreTempSet
        switch( CoreStatus )
        {
                case CORE_PREHEATING:
```

```
                        if( CoreTime <= 1 )
                                startingTemp = CoreTempActual;

                        // adjust Tset
                        CoreTempSet = startingTemp + CoreTime * CoreCurrentProfile.PreHeatSlope;
                        if( CoreTempSet > CoreCurrentProfile.DryingTemp )
                                CoreTempSet = CoreCurrentProfile.DryingTemp;

                        // check temp
                        if( CoreTempActual >= CoreCurrentProfile.DryingTemp - CORE_TEMP_THRESHOLD )
                        {
                                oldTime = CoreTime;
                                CoreStatus++;
                        }
                        break;

                case CORE_DRYING:
                        // adjust Tset
                        CoreTempSet = CoreCurrentProfile.DryingTemp + (CoreTime - oldTime) * \
                                (( CoreCurrentProfile.SoakTemp - CoreCurrentProfile.DryingTemp )/\
                                (double)CoreCurrentProfile.DryingTime);

                        if( CoreTempSet > CoreCurrentProfile.SoakTemp )
                                CoreTempSet = CoreCurrentProfile.SoakTemp;

                        // check temp
                        if( CoreTempActual >= CoreCurrentProfile.SoakTemp - CORE_TEMP_THRESHOLD )
                        {
                                oldTime = CoreTime;
                                CoreStatus++;
                        }
                        break;

                case CORE_HEATING:
                        // adjust Tset
                        CoreTempSet = CoreCurrentProfile.ReflowTemp;

                        // check temp
                        if( CoreTempActual >= CoreCurrentProfile.ReflowTemp - CORE_TEMP_THRESHOLD )
                        {
                                oldTime = CoreTime;
                                CoreStatus++;
                        }
                        break;

                case CORE_REFLOW:
                        // adjust Tset
                        CoreTempSet = CoreCurrentProfile.ReflowTemp;

                        // check time
                        if( CoreTime - oldTime >= CoreCurrentProfile.ReflowTime )
                        {
                                oldTime = CoreTime;
                                CoreStatus++;
                        }
                        break;

                case CORE_COOLING:
                        // adjust Tset
                        CoreTempSet = CoreCurrentProfile.ReflowTemp + ((CoreTime - oldTime) * CoreCurrent-
Profile.CoolingSlope);

                        if( CoreTempSet < 0.0 )
                                CoreTempSet = 0.0;

                        if( CoreTempActual <= 70.0 )
                                CoreStop();

                        break;
        }
        }
                UpdateStatus();
}

void CoreUpdateTemperature( double temperature )
{
        static double tempActualAvg[ CORE_TEMP_SAMPLES ];
```

```
        static char actualAvgIndex;
        char i;

        if( ++actualAvgIndex >= CORE_TEMP_SAMPLES )
                actualAvgIndex = 0;

        tempActualAvg[ actualAvgIndex ] = temperature;

        // sample temp
        temperature = 0.0;
        for( i = 0; i < CORE_TEMP_SAMPLES; i++ )
                temperature += tempActualAvg[ i ];
        CoreTempActual = temperature/(double)CORE_TEMP_SAMPLES;


        // update CoreTempGrad
        temperature = tempActualAvg[ actualAvgIndex ];
        temperature -= actualAvgIndex == CORE_TEMP_SAMPLES-1 ? tempActualAvg[ 0 ] :\
                       tempActualAvg[ actualAvgIndex + 1 ];

        CoreTempGrad   =   temperature*1000.0/(double)((CORE_TEMP_SAMPLES-1)*TEMP_READ_INTERVAL*TEMP_SAM-
PLE);


        if( CoreStatus >= CORE_CONSTANT_TEMP )
                CoreUpdateVars();

        CoreDisplayStatus();
}

void CoreStart( char mode )
{
        CoreStatus = mode;
        if( CoreStatus == CORE_CONSTANT_POWER )
                ACSetPower( CorePower );

        CoreTime = 0;
        TimerResetCore();
        CoreUpdateTicks();
        RunStarted();
}

void CoreStop()
{
        CoreStatus = CORE_IDLE;
        CoreDisplayStatus();
}
```

delay.h

```
#ifndef __DELAY_H
#define __DELAY_H

#include "always.h"

extern unsigned char delayus_variable;

#if PIC_CLK == 4000000
        #define DelayDivisor 4
        #define WaitFor1Us asm("nop")
        #define Jumpback asm("goto $ - 4") //on PIC18F, it is asm("goto $ - 4") and on PIC16F core it is
asm("goto $ - 2")
#elif PIC_CLK == 8000000
        #define DelayDivisor 2
        #define WaitFor1Us asm("nop")
        #define Jumpback asm("goto $ - 4")
#elif PIC_CLK == 10000000
        #define DelayDivisor 2
        #define WaitFor1Us asm("nop"); asm("nop")
        #define Jumpback asm("goto $ - 6")
#elif PIC_CLK == 16000000
        #define DelayDivisor 1
        #define WaitFor1Us asm("nop")
        #define Jumpback asm("goto $ - 4")
#elif PIC_CLK == 20000000
        #define DelayDivisor 1
        #define WaitFor1Us asm("nop"); asm("nop")
        #define Jumpback asm("goto $ - 6")
```

```
#elif PIC_CLK == 32000000
        #define DelayDivisor 1
        #define WaitFor1Us asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop")
        #define Jumpback asm("goto $ - 12")
#elif PIC_CLK == 40000000
        #define DelayDivisor 1
        #define  WaitFor1Us  asm("nop");  asm("nop");  asm("nop");  asm("nop");  asm("nop");  asm("nop");
asm("nop");
        #define Jumpback asm("goto $ - 16")
#else
        #error delay.h - please define PIC_CLK correctly
#endif

#define DelayUs(x) { \
                        delayus_variable=(unsigned char)(x/DelayDivisor); \
                        asm("movlb (_delayus_variable) >> 8"); \
                        WaitFor1Us; } \
                        asm("decfsz (_delayus_variable)&0ffh,f"); \
                        Jumpback;

#define LOOP_CYCLES_CHAR        9               //how many cycles per loop, optimizations on
#define timeout_char_us(x)      (long)(((x)/LOOP_CYCLES_CHAR)*(PIC_CLK/1000000/4))

#define LOOP_CYCLES_INT         16              //how many cycles per loop, optimizations on
#define timeout_int_us(x)       (long)(((x)/LOOP_CYCLES_INT)*(PIC_CLK/1000000/4))

//if lo byte is zero, faster initialization by 1 instrucion
#define timeout_int_lobyte_zero_us(x)  (long)(((x)/LOOP_CYCLES_INT)*(PIC_CLK/4.0)&0xFF00)

//function prototypes
void DelayBigUs(unsigned int cnt);
void DelayMs(unsigned char cnt);
void DelayMs_interrupt(unsigned char cnt);
void DelayBigMs(unsigned int cnt);
void DelayS(unsigned char cnt);

#endif
```

delay.c

```
#ifndef __DELAY_C
#define __DELAY_C

#include <pic18.h>
#include "always.h"

unsigned char delayus_variable;

#include        "delay.h"

void DelayBigUs(unsigned int cnt)
{
        unsigned char  i;

        i = (unsigned char)(cnt>>8);
        while(i>=1)
        {
                i--;
                DelayUs(253);
                CLRWDT();
        }
        DelayUs((unsigned char)(cnt & 0xFF));
}

void DelayMs(unsigned char cnt)
{
        unsigned char  i;
        do {
                i = 4;
                do {
                        DelayUs(250);
                        CLRWDT();
                } while(--i);
        } while(--cnt);
}
```

```
//this copy is for the interrupt function
void DelayMs_interrupt(unsigned char cnt)
{
        unsigned char  i;
        do {
                i = 4;
                do {
                        DelayUs(250);
                } while(--i);
        } while(--cnt);
}

void DelayBigMs(unsigned int cnt)
{
        unsigned char  i;
        do {
                i = 4;
                do {
                        DelayUs(250);
                        CLRWDT();
                } while(--i);
        } while(--cnt);
}

void DelayS(unsigned char cnt)
{
        unsigned char i;
        do {
                i = 4;
                do {
                        DelayMs(250);
                        CLRWDT();
                } while(--i);
        } while(--cnt);
}

#endif
```

eeprom.h

```
#ifndef __EEPROM_H
#define __EEPROM_H

#define EE_CORECOEFFICIENTS            0x00
#define EE_COREPROFILES                0x80

void EEWriteChar( unsigned int addr, char value );
void EEWriteInt( int addr, int value );
void EEWriteLong( int addr, long value );
void EEWriteDouble( int addr, double value );
char EEReadChar( char addr );
int EEReadInt( char addr );
long EEReadLong( char addr );
double ReadDouble( char addr );

#endif
```

eeprom.c

```
#include <pic18.h>
#include "eeprom.h"

void EEWriteChar( unsigned int addr, char value )
{
        eeprom_write( addr, value );
}

void EEWriteInt( int addr, int value )
{
        eeprom_write( addr, value & 0xFF );
        addr++;
        eeprom_write( addr, ( value >> 8 ) & 0xFF );
}

void EEWriteLong( int addr, long value )
{
        eeprom_write( addr, value & 0xFF );
```

```
        addr++;
        eeprom_write( addr, ( value >> 8 ) & 0xFF );
        addr++;
        eeprom_write( addr, ( value >> 16 ) & 0xFF );
        addr++;
        eeprom_write( addr, ( value >> 24 ) & 0xFF );
}

void EEWriteDouble( int addr, double value )
{
        union
        {
                double d;
                long l;
        } val;
        val.d = value;

        eeprom_write( addr, val.l & 0xFF );
        addr++;
        eeprom_write( addr, ( val.l >> 8 ) & 0xFF );
        addr++;
        eeprom_write( addr, ( val.l >> 16 ) & 0xFF );
        addr++;
        eeprom_write( addr, ( val.l >> 24 ) & 0xFF );
}

char EEReadChar( char addr )
{
        return eeprom_read( addr );
}

int EEReadInt( char addr )
{
        int val;
        val = eeprom_read( addr + 1 ) << 8;
        val |= eeprom_read( addr );
        return val;
}

long EEReadLong( char addr )
{
        long val;
        val = eeprom_read( addr );
        val |= eeprom_read( addr + 1 ) << 8;
        val |= ((long)eeprom_read( addr + 2 )) << 16;
        val |= ((long)eeprom_read( addr + 3 )) << 24;
        return val;
}

double ReadDouble( char addr )
{
        union
        {
                double d;
                long l;
        } val;

        val.l = eeprom_read( addr );
        val.l |= eeprom_read( addr + 1 ) << 8;
        val.l |= ((long)eeprom_read( addr + 2 )) << 16;
        val.l |= ((long)eeprom_read( addr + 3 )) << 24;

        return val.d;
}
```

lcd.h

```
#ifndef __LCD_H
#define __LCD_H

#define LCD_RS                  RC1
#define LCD_RS_DIR              TRISC1
#define LCD_RW                  RC2
#define LCD_RW_DIR              TRISC2
#define LCD_E                   RC3
#define LCD_E_DIR               TRISC3
#define LCD_DATA                PORTD
#define LCD_DATA_DIR            TRISD
```

```
void LCDInit();
void LCDSendData( char data );
void LCDClearDisplay();
void LCDUpdateLine( char line );
void LCDUpdate();
void LCDClearLines();

void LCDMapDegC();
void LCDMapDegF();
void LCDMapVerticalArrows();
void LCDMapHorizontalArrows();

#define LCD_LEN                 20

extern char LCDLines[4][ LCD_LEN + 1 ];

#define LCDSetDDAddress( x )\
{\
         LCD_RS = 0;\
         LCDSendData( 0x80 | (x) );\
}

#define LCDSetCGAddress( x )\
{\
         LCD_RS = 0;\
         LCDSendData( 0x40 | (x) );\
}

#define LCDSendDataNoWait( x )\
{\
        LCD_E = 1;\
        LCD_DATA = (x);\
        asm("nop");\
        asm("nop");\
        LCD_E = 0;\
}

#define LCDClearDisplay()\
{\
        LCD_RS = 0;\
        LCDSendData( 0b00000001 );\
}

#endif
```

lcd.c

```
#include "always.h"
#include "characters.h"
#include "lcd.h"
#include <string.h>

char LCDLines[4][ LCD_LEN + 1 ];

// custom chars
const char CharBars[] = { CHAR_BARS_0, CHAR_BARS_1, CHAR_BARS_2,\
                          CHAR_BARS_3, CHAR_BARS_4, CHAR_BARS_5 };
const char LCDMapBars1d[] =          { 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10 };
const char LCDMapBars2d[] =          { 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18 };
const char LCDMapBars3d[] =          { 0x1C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C };
const char LCDMapBars4d[] =          { 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E };
const char LCDMapDegFd[] =           { 0x1C, 0x14, 0x1C, 0x00, 0x07, 0x04, 0x06, 0x04 };
const char LCDMapDegCd[] =           { 0x1C, 0x14, 0x1C, 0x00, 0x03, 0x04, 0x04, 0x03 };
const char LCDMapPerSd[] =           { 0x01, 0x0D, 0x01, 0x0E, 0x10, 0x0C, 0x02, 0x1C };
const char LCDMapArrowUpd[] =        { 0x00, 0x00, 0x00, 0x04, 0x0E, 0x1F, 0x00, 0x00 };
const char LCDMapArrowDownd[] =      { 0x00, 0x00, 0x1F, 0x0E, 0x04, 0x00, 0x00, 0x00 };
const char LCDMapArrowRightd[] =     { 0x00, 0x10, 0x18, 0x1C, 0x18, 0x10, 0x00, 0x00 };
const char LCDMapArrowLeftd[] =      { 0x00, 0x01, 0x03, 0x07, 0x03, 0x01, 0x00, 0x00 };

void LCDMapDefFont();

void LCDInit()
{
        LCD_RS_DIR = 0;
        LCD_RW_DIR = 0;
        LCD_E_DIR = 0;
```

```
        LCD_DATA_DIR = 0;

        LCD_RS = 0;
        LCD_RW = 0;

        LCDSendDataNoWait( 0b00111000 );
        DelayMs( 5 );
        LCDSendDataNoWait( 0b00111000 );
        DelayUs( 150 );
        LCDSendDataNoWait( 0b00111000 );

        LCDSendData( 0b00111000 );
        LCDSendData( 0b00001100 );
        LCDSendData( 0b00000001 );
        LCDSendData( 0b00000110 );

        LCDMapDefFont();
        LCDMapDegC();
        LCDClearDisplay();
}

void LCDSendData( char data )
{
        static bit oldRS;
        unsigned int timeOut = -1;

        oldRS = LCD_RS;
        LCD_RS = 0;
        LCD_DATA_DIR = 0xFF;
        LCD_RW = 1;

        while( timeOut-- )
        {
                LCD_E = 1;
                // propagation delay
                DelayUs(50);
                if(!( LCD_DATA & 0x80 )) break;
                LCD_E = 0;
        }

        LCD_RW = 0;
        LCD_DATA_DIR = 0;
        LCD_RS = oldRS;
        LCDSendDataNoWait( data );
}

void LCDUpdateLine( char line )
{
        char i;
        char data;
        static bit fillLine;

        if( line == 0 )
        {
                LCDSetDDAddress( 0x00 );
        }

        else if( line == 1 )
        {
                LCDSetDDAddress( 0x40 );
        }

        else if( line == 2 )
        {
                LCDSetDDAddress( 0x14 );
        }

        else if( line == 3 )
        {
                LCDSetDDAddress( 0x54 );
        }

        LCD_RS = 1;
        fillLine = 0;
        for( i = 0; i < LCD_LEN; i++ )
        {
                if( fillLine )
```

```
                        data = ' ';
                else
                {
                        data = LCDLines[ line ][ i ];
                        if( data == 0 )
                        {
                                fillLine = 1;
                                data = ' ';
                        }
                }

                LCDSendData( data );
        }
}

void LCDUpdate()
{
        LCDUpdateLine( 0 );
        LCDUpdateLine( 1 );
        LCDUpdateLine( 2 );
        LCDUpdateLine( 3 );
}

void LCDClearLines()
{
        LCDLines[0][0] = 0;
        LCDLines[1][0] = 0;
        LCDLines[2][0] = 0;
        LCDLines[3][0] = 0;
}

void LCDSendCharPattern( char addr, const char pattern[] )
{
        char loop;
        LCDSetCGAddress( addr << 3 );

        LCD_RS = 1;

        for( loop = 0; loop < 8; loop++ )
                LCDSendData( pattern[ loop ] );
}

void LCDMapDefFont()
{
        LCDSendCharPattern( CHAR_BARS_1, LCDMapBars1d );
        LCDSendCharPattern( CHAR_BARS_2, LCDMapBars2d );
        LCDSendCharPattern( CHAR_BARS_3, LCDMapBars3d );
        LCDSendCharPattern( CHAR_BARS_4, LCDMapBars4d );
        LCDSendCharPattern( CHAR_PERS, LCDMapPerSd );
}

void LCDMapDegC()
{
        LCDSendCharPattern( CHAR_DEG, LCDMapDegCd );
}

void LCDMapDegF()
{
        LCDSendCharPattern( CHAR_DEG, LCDMapDegFd );
}

void LCDMapVerticalArrows()
{
        LCDSendCharPattern( CHAR_ARROW_UP, LCDMapArrowUpd );
        LCDSendCharPattern( CHAR_ARROW_DOWN, LCDMapArrowDownd );
}

void LCDMapHorizontalArrows()
{
        LCDSendCharPattern( CHAR_ARROW_LEFT, LCDMapArrowLeftd );
        LCDSendCharPattern( CHAR_ARROW_RIGHT, LCDMapArrowRightd );
}
```

main.c

```
#include "always.h"
#include "lcd.h"
#include "menu.h"
```

```
#include "timer.h"
#include "core.h"
#include "comm.h"
#include "adc.h"
#include "ac.h"
#include "temperature.h"

void Init()
{
        LCDInit();
        MenuInit();
        TimerInit();
        ADCInit();
        ACInit();
        CommInit();

        IPEN = 1;       // enable all interrupts
        GIEL = 1;
        GIEH = 1;
}

int main()
{
        Init();

        while( 1 )
        {
                // ~14ms
                if( TempReadFlag )
                {
                        TempReadFlag = 0;
                        TempRead();
                        continue;
                }

                // ~7ms when updating LCD
                if( MenuReadButtonFlag )
                {
                        MenuReadButtonFlag = 0;
                        MenuReadButtons();
                        continue;
                }

                // ~60us in profile mode
                if( CoreUpdateTicksFlag )
                {
                        CoreUpdateTicksFlag = 0;
                        CoreUpdateTicks();
                        continue;
                }

                CheckRX();
        }
}


// HighPriorityISR() handled in ac.c
void interrupt low_priority LowPriorityISR()
{
        if( TMR3IF )
        {
                TMR3IF = 0;
                InterruptTimer3();
                return;
        }

        else if( TXIF && TXIE )
                InterruptTX();

        else if( RCIF )
                InterruptRX();
}
```

menu.h

```
#ifndef __MENU_H
#define __MENU_H
```

```
#define MENU_BUTTON_LEFT              0x01
#define MENU_BUTTON_RIGHT             0x02
#define MENU_BUTTON_UP                0x04
#define MENU_BUTTON_DOWN              0x08
#define MENU_HOLD_DOWN                0x10
#define MENU_REFRESH                  0x20
#define MENU_ENTER                    0x40
#define MENU_OK                       0x80

#define MENU_BUTTON_LEFT_PIN          RB1
#define MENU_BUTTON_UP_PIN            RB2
#define MENU_BUTTON_DOWN_PIN          RB3
#define MENU_BUTTON_RIGHT_PIN         RB4
#define MENU_BUTTON_LEFT_PIN_DIR      TRISB1
#define MENU_BUTTON_RIGHT_PIN_DIR     TRISB2
#define MENU_BUTTON_UP_PIN_DIR        TRISB3
#define MENU_BUTTON_DOWN_PIN_DIR      TRISB4

#define MENU_READ_INTERVAL            5       // (ms)
#define MENU_BUTTON_THRESHOLD         1
#define MENU_HOLD_DOWN_THRESHOLD      80
#define MENU_HOLD_DOWN_REPEAT         20
#define MENU_HOLD_DOWN_MULTIPLIER     5
#define MENU_BLINK_INTERVAL           60

#define MENU_STATUS_GENERAL           0x00
#define MENU_STATUS_CONTROL           0x01
#define MENU_STATUS_TEMP              0x02

#define VTYPE_SIGNED_CHAR             0x11
#define VTYPE_SIGNED_INT              0x21
#define VTYPE_SIGNED_LONG             0x41
#define VTYPE_SIGNED_DOUBLE           0x45
#define VTYPE_UNSIGNED_CHAR           0x12
#define VTYPE_UNSIGNED_INT            0x22
#define VTYPE_UNSIGNED_DOUBLE         0x44
#define VTYPE_SIGNED                  0x01
#define VTYPE_UNSIGNED                0x02
#define VTYPE_REAL                    0x04
#define VTYPE_CHAR                    0x10
#define VTYPE_INT                     0x20
#define VTYPE_LONG                    0x40

typedef struct VariableItem_
{
        char type;
        union pVar_
        {
                long* l;
                int* i;
                char* c;
                double* d;
        } pVar;

        signed long varMax;
} VariableItem;

typedef void (MenuProc)( char arg );

#define MENU_ITEM_TYPE_VARIABLE       1
#define MENU_ITEM_TYPE_MENUPROC       2
#define MENU_ITEM_TYPE_CHILDS         3

typedef struct MenuItem_
{
        char type;
        const char* name;
        union event_
        {
                const MenuProc* mProc;
                const struct MenuItem_* const * childs;
        } event;

        const VariableItem* varItem;
} MenuItem;

extern bit MenuReadButtonFlag;
```

```
extern bit MenuBusy;
extern char MenuStatus;

void MenuReadButtons();
void MenuInit();
void MenuButtonEvent( char button );
void MenuReset();
void MenuBack();
void MenuRefresh();

char* MenuGetBar( long index, long range, char strLen /* >= 3 */);

#endif
```

menu.c

```
#include "always.h"
#include "lcd.h"
#include "characters.h"
#include "menu.h"
#include "core.h"
#include "eeprom.h"
#include "ac.h"

#include <string.h>
#include <stdio.h>

bit MenuRefreshFlag;
bit MenuCatchButtonEvent;
bit MenuReadButtonFlag;
bit MenuBlinkToggle;
bit MenuBusy;
bit MenuBackFlag;
char MenuStatus = MENU_STATUS_GENERAL;
const MenuItem * currentMenuItem;

void MenuInternalError();
void MenuDisplayMsg( const char* msg );

////////////////////////////////////////////////////////////////////////////
// Menu Events
////////////////////////////////////////////////////////////////////////////
void mStatusGeneral( char arg )
{
        MenuStatus = MENU_STATUS_GENERAL;
        MenuDisplayMsg( "General Selected." );
}

void mStatusControl( char arg )
{
        MenuStatus = MENU_STATUS_CONTROL;
        MenuDisplayMsg( "Control Selected." );
}

void mStatusTemp( char arg )
{
        MenuStatus = MENU_STATUS_TEMP;
        MenuDisplayMsg( "Temp Selected." );
}

void mSetTemp( char arg )
{
        if( arg == MENU_OK )
        {
                CoreStart( CORE_CONSTANT_TEMP );
                MenuDisplayMsg( "Started." );
        }
}

void mSetPower( char arg )
{
        if( arg == MENU_OK )
        {
                CoreStart( CORE_CONSTANT_POWER );
                MenuDisplayMsg( "Started." );
        }
```

```
        else ACSetPower( CorePower );
}

void mProfile( char arg )
{
        CoreStart( CORE_PREHEATING );
        MenuDisplayMsg( "Started." );
}

void mDummy( char arg )
{

}

////////////////////////////////////////////////////////////////////////////
// Menu Definition
////////////////////////////////////////////////////////////////////////////

extern const MenuItem *const mi1c[];
extern const MenuItem *const mi2c[], *const mi22c[], *const mi23c[], *const mi24c[], *const mi25c[];
extern const MenuItem *const mi3c[], *const mi21c[], *const mi221c[];

const MenuItem mi1 = { MENU_ITEM_TYPE_CHILDS, "Status", { mi1c }, NULL };
        const MenuItem mi11 = { MENU_ITEM_TYPE_MENUPROC, "General Status", { mStatusGeneral }, NULL };
        const MenuItem mi12 = { MENU_ITEM_TYPE_MENUPROC, "Control Status", { &mStatusControl }, NULL };
        const MenuItem mi13 = { MENU_ITEM_TYPE_MENUPROC, "Temperature Status",{ &mStatusTemp }, NULL };
const MenuItem* const mi1c[] = { &mi11, &mi12, &mi13, NULL };

const MenuItem mi2 = { MENU_ITEM_TYPE_CHILDS, "Settings", { mi2c }, NULL };

const MenuItem mi21 = { MENU_ITEM_TYPE_CHILDS, "Profil Settings", { mi21c }, NULL };
        const MenuItem mi211 = { MENU_ITEM_TYPE_CHILDS, "Coefficients", { NULL }, NULL };
        const MenuItem mi212 = { MENU_ITEM_TYPE_MENUPROC, "Load Preset", { NULL }, NULL };
        const MenuItem mi213 = { MENU_ITEM_TYPE_MENUPROC, "Save Preset", { NULL }, NULL };
        const MenuItem mi214 = { MENU_ITEM_TYPE_MENUPROC, "Delete Preset", { NULL }, NULL };
const MenuItem* const mi21c[] = { &mi211, &mi212, &mi213, &mi214, NULL };

const MenuItem mi22 = { MENU_ITEM_TYPE_CHILDS, "Control Settings", { mi22c }, NULL };
        const MenuItem mi221 = { MENU_ITEM_TYPE_CHILDS, "Coefficients", { mi221c }, NULL };
        const MenuItem mi2211v = { VTYPE_UNSIGNED_DOUBLE, { &CoreCoefficients.Kp }, 30 };
        const MenuItem mi2211 = { MENU_ITEM_TYPE_VARIABLE, "Kp", { NULL }, &mi2211v };
        const MenuItem mi2212v = { VTYPE_UNSIGNED_DOUBLE, { &CoreCoefficients.Kd }, 30 };
        const MenuItem mi2212 = { MENU_ITEM_TYPE_VARIABLE, "Kd", { NULL }, &mi2212v };
const MenuItem* const mi221c[] = { &mi2211, &mi2212, NULL };

        const MenuItem mi222 = { MENU_ITEM_TYPE_MENUPROC, "Load Preset", { NULL }, NULL };
        const MenuItem mi223 = { MENU_ITEM_TYPE_MENUPROC, "Save Preset", { NULL }, NULL };
        const MenuItem mi224 = { MENU_ITEM_TYPE_MENUPROC, "Delete Preset", { NULL }, NULL };
const MenuItem* const mi22c[] = { &mi221, &mi222, &mi223, &mi224, NULL };

const MenuItem mi23 = { MENU_ITEM_TYPE_CHILDS, "Langage", { mi23c }, NULL };
        const MenuItem mi231 = { MENU_ITEM_TYPE_MENUPROC, "English", { NULL }, NULL };
        const MenuItem mi232 = { MENU_ITEM_TYPE_MENUPROC, "Francais", { NULL }, NULL };
const MenuItem* const mi23c[] = { &mi231, &mi232, NULL };

const MenuItem mi24 = { MENU_ITEM_TYPE_CHILDS, "Probe Settings", { mi24c }, NULL };
        const MenuItem mi241 = { MENU_ITEM_TYPE_VARIABLE, "TCR", { NULL }, NULL };
        const MenuItem mi242 = { MENU_ITEM_TYPE_VARIABLE, "Base Resistance", { NULL }, NULL };
        const MenuItem mi243 = { MENU_ITEM_TYPE_VARIABLE, "Excitation Current", { NULL }, NULL };
const MenuItem* const mi24c[] = { &mi241, &mi242, &mi243, NULL };

const MenuItem mi25 = { MENU_ITEM_TYPE_CHILDS, "Temperature Unit", { mi25c }, NULL };
        const MenuItem mi251 = { MENU_ITEM_TYPE_MENUPROC, "Celsius", { NULL }, NULL };
        const MenuItem mi252 = { MENU_ITEM_TYPE_MENUPROC, "Farenheight", { NULL }, NULL };
        const MenuItem* const mi25c[] = { &mi251, &mi252, NULL };
const MenuItem* const mi2c[] = { &mi21, &mi22, &mi23, &mi24, &mi25, NULL };

const MenuItem mi3 = { MENU_ITEM_TYPE_CHILDS, "Run", { mi3c }, NULL };
        const MenuItem mi31 = { MENU_ITEM_TYPE_MENUPROC, "Profile", { &mProfile }, NULL };
        const VariableItem mi32v = { VTYPE_UNSIGNED_DOUBLE, { &CoreTempSet }, CORE_MAX_TEMPERATURE };
        const MenuItem mi32 = { MENU_ITEM_TYPE_VARIABLE, "Constant Temp", { &mSetTemp }, &mi32v };
        const VariableItem mi33v = { VTYPE_UNSIGNED_DOUBLE, { &CorePower }, 100 };
        const MenuItem mi33 = { MENU_ITEM_TYPE_VARIABLE, "Constant Power", { &mSetPower }, &mi33v };
const MenuItem* const mi3c[] = { &mi31, &mi32, &mi33, NULL };

const MenuItem mi4 = { MENU_ITEM_TYPE_MENUPROC, "Stop", { NULL }, NULL };
```

```
const MenuItem* const miRoot1c[] = { &mi3, &mi1, &mi2, NULL };
const MenuItem* const miRoot2c[] = { &mi4, &mi1, &mi2, NULL };

const MenuItem miRoot1 = { MENU_ITEM_TYPE_CHILDS, NULL, miRoot1c };
const MenuItem miRoot2 = { MENU_ITEM_TYPE_CHILDS, NULL, miRoot2c };

void MenuInit()
{
        MENU_BUTTON_LEFT_PIN_DIR = 1;
        MENU_BUTTON_RIGHT_PIN_DIR = 1;
        MENU_BUTTON_UP_PIN_DIR = 1;
        MENU_BUTTON_DOWN_PIN_DIR = 1;

        RBPU = 0;


        MenuReset();
}

void mVariableProc( char arg )
{
        static lineIndex = 0;
        static near char type;
        static near long oldVar;
        static near long varMax;
        static near long var;
        int added;

        // init
        if( arg & MENU_ENTER )
        {
                lineIndex = 0;
                type = currentMenuItem->varItem->type;

                if( type & VTYPE_REAL ) var = (long)(*currentMenuItem->varItem->pVar.d);
                else if( type & VTYPE_CHAR ) var = (long)(*currentMenuItem->varItem->pVar.c);
                else if( type & VTYPE_INT ) var = (long)(*currentMenuItem->varItem->pVar.i);
                else if( type & VTYPE_LONG ) var = *currentMenuItem->varItem->pVar.l;

                oldVar = var;

                varMax = currentMenuItem->varItem->varMax;

                LCDMapHorizontalArrows();
                MenuRefresh();
        }

        if( arg & MENU_REFRESH )
        {
                if( currentMenuItem->varItem == NULL || currentMenuItem->varItem->pVar.l == NULL )
                {
                        MenuInternalError();
                        return;
                }

                sprintf( LCDLines[0], "%s:", currentMenuItem->name );

                if( MenuBlinkToggle && lineIndex == 0 )
                        sprintf( LCDLines[1], "%c %ld %c", CHAR_ARROW_LEFT, var, CHAR_ARROW_RIGHT );
                else sprintf( LCDLines[1], "  %ld", var );

                strcpy( LCDLines[2], MenuGetBar( var, varMax, LCD_LEN ) );

                if( lineIndex == 0 )
                        sprintf( LCDLines[3], "%cCancel          %cOK", ' ', ' ' );
                else if( lineIndex == 1 )
                        sprintf( LCDLines[3], "%cCancel            %cOK", CHAR_STRAIGHT_ARROW_RIGHT, ' '
);
                else sprintf( LCDLines[3], "%cCancel          %cOK", ' ', CHAR_STRAIGHT_ARROW_RIGHT );

                LCDUpdate();
        }

        if( lineIndex == 0 )
        {
                if( arg & MENU_BUTTON_UP )
                        return;
```

```
		if( arg & MENU_BUTTON_DOWN )
		{
			lineIndex++;
			MenuRefresh();
			return;
		}

		if( arg & MENU_BUTTON_LEFT || arg & MENU_BUTTON_RIGHT )
		{
			added = 1;
			if( arg & MENU_BUTTON_LEFT ) added = -added;
			if( arg & MENU_HOLD_DOWN  ) added *= (MENU_HOLD_DOWN_MULTIPLIER * varMax)/128 ?\
                                                (MENU_HOLD_DOWN_MULTIPLIER * varMax)/128 : 2;

			var += added;

			// check bounds
			if( type & VTYPE_SIGNED )
			{
				if( var < -varMax ) var = -varMax;
			}
			else
			{
				if( var < 0 ) var = 0;
			}
			if( var > varMax ) var = varMax;

			/*
			if( type & VTYPE_REAL )
				*currentMenuItem->varItem->pVar.d = (double)var;
			else *currentMenuItem->varItem->pVar.l = var;
			*/

			if( type & VTYPE_REAL ) *currentMenuItem->varItem->pVar.d = (double)var;
			else if( type & VTYPE_CHAR ) *currentMenuItem->varItem->pVar.c = (char)var;
			else if( type & VTYPE_INT ) *currentMenuItem->varItem->pVar.i = (int)var;
			else if( type & VTYPE_LONG ) *currentMenuItem->varItem->pVar.l = var;

			if( currentMenuItem->event.mProc != NULL )
				(*currentMenuItem->event.mProc)( MENU_REFRESH );

			MenuRefresh();
		}
	}

	else if( lineIndex == 1 )
	{
		if( arg & MENU_BUTTON_UP )
		{
			lineIndex--;
			MenuRefresh();
			return;
		}

		if( arg & MENU_BUTTON_DOWN )
		{
			lineIndex++;
			MenuRefresh();
			return;
		}

		if( arg & MENU_BUTTON_RIGHT )
		{ // Cancel
			if( type & VTYPE_REAL )
				*currentMenuItem->varItem->pVar.d = (double)oldVar;
			else *currentMenuItem->varItem->pVar.l = oldVar;
			//MenuReset();
			MenuBack();
		}
	}

	else if( lineIndex == 2 )
	{
		if( arg & MENU_BUTTON_UP )
		{
			lineIndex--;
```

```
                        MenuRefresh();
                        return;
                }

                if( arg & MENU_BUTTON_DOWN )
                        return;

                if( arg & MENU_BUTTON_RIGHT )
                { // OK
                        if( type & VTYPE_REAL )
                                *currentMenuItem->varItem->pVar.d = (double)var;
                        else *currentMenuItem->varItem->pVar.l = var;

                        if( currentMenuItem->event.mProc == NULL )
                                //MenuReset();
                                MenuBack();
                        else (*currentMenuItem->event.mProc)( MENU_OK );
                }
        }
}

near char MenuChildIndex = 0;
near char MenuDisplayIndex = 0;
near char MenuParentIndex = 0;

void MenuRefresh()
{
        MenuRefreshFlag = 1;
        MenuReadButtonFlag = 1;
}

void MenuReset()
{
        currentMenuItem = &miRoot1;
        MenuCatchButtonEvent = 0;
        MenuChildIndex = 0;
        MenuDisplayIndex = 0;
        MenuParentIndex = 0;
        MenuBusy = 0;
}

void MenuBack()
{
        MenuBackFlag = 1;
}

void MenuDisplayMsg( const char* msg )
{
        LCDClearLines();
        strcpy( LCDLines[0], msg );
        LCDUpdate();
        DelayBigMs( 300 );
        MenuReset();
}

void MenuInternalError()
{
        strcpy( LCDLines[0], "Internal Error !" );
        MenuDisplayMsg( "Internal Error !" );
        MenuCatchButtonEvent = 0;
        return;
}

////////////////////////////////////////////////////////////////////////////
// MenuButtonEvent(): Menu handler
////////////////////////////////////////////////////////////////////////////
void MenuButtonEvent( char button )
{
        static const MenuItem* parents[ 8 ] = { NULL };
        char i;

        if( MenuCatchButtonEvent )
        {
                // redirect buttons event to the current proc
                if( currentMenuItem->type == MENU_ITEM_TYPE_VARIABLE )
                        mVariableProc( button );
```

```
        else
        {
                if( currentMenuItem->event.mProc == NULL )
                {
                        MenuInternalError();
                        return;
                }
                (*currentMenuItem->event.mProc)( button );
        }
        return;
}


switch( button & 0x0F )
{
case MENU_BUTTON_LEFT:
        if( MenuParentIndex == 0 )
        {
                MenuBusy = 0;
                CoreDisplayStatus();
                return;
        }

        MenuParentIndex--;
        MenuChildIndex = 0;
        MenuDisplayIndex = 0;
        currentMenuItem = parents[ MenuParentIndex ];

        break;

case MENU_BUTTON_RIGHT:
        if( currentMenuItem->event.childs == NULL )
        {
                MenuInternalError();
                return;
        }

        parents[ MenuParentIndex ] = currentMenuItem;
        MenuParentIndex++;
        currentMenuItem = currentMenuItem->event.childs[ MenuChildIndex ];
        MenuChildIndex = 0;
        MenuDisplayIndex = 0;

        if( currentMenuItem->type == MENU_ITEM_TYPE_CHILDS )
                break;

        MenuCatchButtonEvent = 1;
        if( currentMenuItem->type == MENU_ITEM_TYPE_VARIABLE )
                mVariableProc( MENU_ENTER );

        MenuRefresh();
        return;

case MENU_BUTTON_UP:
        if( currentMenuItem->event.childs == NULL )
        {
                MenuInternalError();
                return;
        }

        if( MenuChildIndex == 0 )
                break;

        MenuChildIndex--;
        if( MenuDisplayIndex > MenuChildIndex )
                MenuDisplayIndex = MenuChildIndex;

        break;

case MENU_BUTTON_DOWN:
        if( currentMenuItem->type == MENU_ITEM_TYPE_CHILDS )
        {
                if( currentMenuItem->event.childs == NULL )
                        break;
                if( currentMenuItem->event.childs[ MenuChildIndex+1 ] == NULL )
                        break;
```

```
                        MenuChildIndex++;
                        if( MenuDisplayIndex + 3 < MenuChildIndex )
                                MenuDisplayIndex = MenuChildIndex - 3;
                }

                break;
        }

        for( i = 0; i < 4; i++ )
                strcpy( LCDLines[ i ], " " );

        LCDLines[ MenuChildIndex - MenuDisplayIndex ][0] = CHAR_STRAIGHT_ARROW_RIGHT;

        if( currentMenuItem->event.childs == NULL )
        {
                MenuInternalError();
                return;
        }

        for( i = 0; i < 4; i++ )
        {
                if( currentMenuItem->event.childs[ MenuDisplayIndex + i ] == NULL )
                        break;
                strcpy( LCDLines[i]+1, currentMenuItem->event.childs[ MenuDisplayIndex + i ]->name );
        }

        if( MenuBlinkToggle )
        {
                // add some arrows
                for( i = strlen( LCDLines[0] ); i < LCD_LEN-1; i++ )
                        LCDLines[0][ i ] = ' ';
                LCDLines[0][ LCD_LEN - 1 ] = CHAR_ARROW_UP;
                LCDLines[0][ LCD_LEN ] = 0;

                for( i = strlen( LCDLines[3] ); i < LCD_LEN-1; i++ )
                        LCDLines[3][ i ] = ' ';
                LCDLines[3][ LCD_LEN - 1 ] = CHAR_ARROW_DOWN;
                LCDLines[3][ LCD_LEN ] = 0;
        }

        LCDMapVerticalArrows();

        LCDUpdate();
}

////////////////////////////////////////////////////////////////////////////
// MenuReadButtons(): Scan keys and send keyevent to MenuButtonEvent()
////////////////////////////////////////////////////////////////////////////

void MenuReadButtons()
{
        static char holdDown;
        static char oldButtons;
        static char incBlink = MENU_BLINK_INTERVAL;
        char currentButtons;

        if( !--incBlink )
        {
                incBlink = MENU_BLINK_INTERVAL;
                if( MenuBlinkToggle ) MenuBlinkToggle = 0;
                else MenuBlinkToggle = 1;
                MenuRefresh();
        }

        if( MenuRefreshFlag )
        {
                MenuRefreshFlag = 0;
                if( !MenuBusy )
                        return;
                MenuButtonEvent( MENU_REFRESH );
        }

        if( MenuBackFlag )
        {
                MenuBackFlag = 0;
                MenuCatchButtonEvent = 0;
                MenuButtonEvent( MENU_BUTTON_LEFT );
```

```
                return;
        }

        // scan keys
        if( !MENU_BUTTON_LEFT_PIN )
        {
                //strcpy(LCDLines[3], "Left");
                currentButtons = MENU_BUTTON_LEFT;
        }

        else if( !MENU_BUTTON_RIGHT_PIN )
        {
                //strcpy(LCDLines[3], "Right");
                currentButtons = MENU_BUTTON_RIGHT;
        }

        else if( !MENU_BUTTON_DOWN_PIN )
        {
                //strcpy(LCDLines[3], "Down");
                currentButtons = MENU_BUTTON_DOWN;
        }

        else if( !MENU_BUTTON_UP_PIN )
        {
                //strcpy(LCDLines[3], "Up");
                currentButtons = MENU_BUTTON_UP;
        }

        else
        {
                //strcpy(LCDLines[3], "");
                //LCDUpdateLine(3);
                holdDown = 0;
                return;
        }

        //LCDUpdateLine(3);

        if( currentButtons != oldButtons )
        {
                oldButtons = currentButtons;
                holdDown = 0;
                return;
        }

        holdDown++;

        if( holdDown == MENU_BUTTON_THRESHOLD+1 )
        {
                if( !MenuBusy )
                {
                        MenuBusy = 1;
                        MenuRefresh();
                        return;
                }
                MenuButtonEvent( currentButtons );
        }

        // hold down feature
        else if( holdDown == MENU_HOLD_DOWN_THRESHOLD )
        {
                holdDown = MENU_HOLD_DOWN_THRESHOLD - MENU_HOLD_DOWN_REPEAT;
                MenuButtonEvent( currentButtons | MENU_HOLD_DOWN );
        }
}


char* MenuGetBar( long index, long range, char strLen )
{
        static char bar[ LCD_LEN + 1 ] = "[";
        char bars;
        char endBar;
        signed char i;

        if( index < 0 ) index = -index;
        if( index > range ) index = range;
```

```
        strLen -= 2; // '[' ']'

        bars = ( index * strLen * 5 ) / range;
        endBar = bars/5;

        for( i = endBar - 1; i >= 0; i-- )
                bar[ i + 1 ] = CHAR_BARS_5;

        bar[ endBar + 1 ] = CharBars[ bars % 5 ];

        for( i = endBar + 1; i < strLen; i++ )
                bar[ i + 1 ] = CHAR_BARS_0;


        bar[ strLen + 1 ] = ']';
        bar[ strLen + 2 ] = '\0';

        return bar;
}
```

temperature.h

```
#ifndef __TEMPERATURE_H
#define __TEMPERATURE_H

#define TEMP_RDT_CHANNEL        1       //RA1
#define TEMP_RDT_TCR            0.00385

#define TEMP_READ_INTERVAL      30      // ms
#define TEMP_SAMPLE             8

void TempRead();
extern bit TempReadFlag;

#endif
```

temperature.c

```
#include "always.h"
#include "temperature.h"
#include "adc.h"
#include "core.h"

bit TempReadFlag;

void TempRead()
{
        static double avgTemp;
        static char sampled = TEMP_SAMPLE;

        unsigned int temp;
        ADCSelectChannel( TEMP_RDT_CHANNEL );
        temp = ADCRead();

        avgTemp += ((double)temp)/( TEMP_RDT_TCR * 1023.0 );

        if( !--sampled )
        {
                sampled = TEMP_SAMPLE;
                CoreUpdateTemperature( avgTemp /(double)TEMP_SAMPLE );
                avgTemp = 0.0;
        }
}
```

timer.h

```
#ifndef __TIMER_H
#define __TIMER_H

// using timer3

void TimerInit();
void TimerResetCore();
void InterruptTimer3();

// flags
extern bit Timer20ms;

#define DumpTMR3ToTimerTemp()\
```

```
{\
        asm( "movf _TMR3L, W");\
        asm( "movwf _TimerTemp");\
        asm( "movf _TMR3H, W");\
        asm( "movwf (_TimerTemp) + 1");\
}

#define DumpTimerTempToTMR3()\
{\
        asm( "movf (_TimerTemp) + 1, W");\
        asm( "movwf _TMR3H");\
        asm( "movf _TimerTemp, W");\
        asm( "movwf _TMR3L");\
}

#endif
```

timer.c

```
#include <pic18.h>
#include "timer.h"
#include "ac.h"
#include "menu.h"
#include "temperature.h"
#include "core.h"

near unsigned int TimerTemp;
near unsigned int TimerIncCore;
bit Timer20ms;

void TimerInit()
{
        T3CON = 0b10000001;     // prescale: 1:1
        TMR3IE = 1;
        TMR3IP = 0;             // low priority
}

void TimerResetCore()
{
        TimerIncCore = CORE_TICKS_INTERVAL;
        CoreUpdateTicksFlag = 0;
}

// triggered every 1ms
void InterruptTimer3()
{
        static char IncMenu = MENU_READ_INTERVAL;
        static char IncTemp;

        DumpTMR3ToTimerTemp();
        TimerTemp = 2*DUMP_COMPENSATION + TimerTemp - 10000; // 1ms
        DumpTimerTempToTMR3();

        if( !--IncMenu )
        {
                IncMenu = MENU_READ_INTERVAL;
                MenuReadButtonFlag = 1;
        }

        if( !--IncTemp )
        {
                IncTemp = TEMP_READ_INTERVAL;
                TempReadFlag = 1;
        }

        if( !--TimerIncCore )
        {
                CoreTime++;
                TimerIncCore = CORE_TICKS_INTERVAL;
                CoreUpdateTicksFlag = 1;
        }
}
```

...:: LA SOURCE DU LOGICIEL PC ::...

```
/*****************************************************************************
 *   Reflow Oven Firmware                                                    *
 *                                                                           *
 *   Copyright (C) 2004 by Nicolas Viennot                                   *
 *   nicolas.viennot@free.fr                                                 *
 *                                                                           *
 *   This program is free software; you can redistribute it and/or modify    *
 *   it under the terms of the GNU General Public License as published by    *
 *   the Free Software Foundation; either version 2 of the License, or       *
 *   (at your option) any later version.                                     *
 *                                                                           *
 *   This program is distributed in the hope that it will be useful,         *
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of          *
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the           *
 *   GNU General Public License for more details.                            *
 *                                                                           *
 *   You should have received a copy of the GNU General Public License       *
 *   along with this program; if not, write to the                           *
 *   Free Software Foundation, Inc.,                                         *
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.               *
 *****************************************************************************/
```

mainForm.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using SoftwareFX.ChartFX;
using SoftwareFX.ChartFX.Data;

namespace ReflowOven
{
	public class Form1 : System.Windows.Forms.Form
	{
		private OvenDelegate pRefresh;
		private Oven oven;
		private System.ComponentModel.Container components = null;
		private DateTime runStart;

		private const double TimeBase = 420;
		private const double TempBase = 260;

		private const double TempThreshold = 1;
		private const double TempDeltaForce = 2;
		private SoftwareFX.ChartFX.Chart chart;
		private DateTime lastTempWrittenTime;

		private double lastTempActualWritten = 0.0;
		private double lastTempSetWritten = 0.0;

		private void RefreshOvenStatus()
		{
			// adjust axisX
			if( oven.Time > chart.AxisX.Max )
				chart.AxisX.Max += TimeBase;

			DateTime ovenTime = new DateTime(oven.Time * 10 * 1000 );
```

```
        TimeSpan deltaWrittenTime = ovenTime - lastTempWrittenTime;
        if( lastTempActualWritten != 0.0 && \
                deltaWrittenTime.TotalSeconds < TempDeltaForce )
        {
                if( Math.Abs( lastTempActualWritten - oven.TempActual ) < TempThreshold \
                        && Math.Abs( lastTempSetWritten - oven.TempSet ) < TempThreshold)
                        return;
        }

        lastTempActualWritten = oven.TempActual;
        lastTempSetWritten = oven.TempSet;
        lastTempWrittenTime = ovenTime;


        // udpate data
        chart.OpenData(COD.Values | COD.AddPoints, 2, 1);
        chart.OpenData(COD.XValues, 2, 0);
        chart.Value[0,0] = oven.TempActual;
        chart.XValue[0,0] = oven.Time;
        chart.Value[1,0] = oven.TempSet;
        chart.XValue[1,0] = oven.Time;
        chart.CloseData(COD.XValues);
        chart.CloseData(COD.Values);
}


private void Reset_Click(object sender, System.EventArgs e)
{
        ResetCurves();
}

private void ResetCurves()
{
        chart.OpenData( COD.Remove | COD.Values, 1, 0 );
        chart.CloseData( COD.Remove | COD.Values );
        chart.OpenData( COD.Remove | COD.Values, 2, 0 );
        chart.CloseData( COD.Remove | COD.Values );

        lastTempWrittenTime = DateTime.MinValue;
        runStart = DateTime.Now;
        chart.AxisX.Max = TimeBase;
}

public Form1()
{
        InitializeComponent();
        oven = new Oven();

        pRefresh = new OvenDelegate( RefreshOvenStatus );
        oven.StatusInvalided += pRefresh;
        oven.RunStarted += new OvenDelegate( ResetCurves );

        chart.OpenData(COD.Values | COD.Remove | COD.AllocHidden, 1, 0);
        chart.CloseData(COD.Values);

        chart.OpenData(COD.Values | COD.Remove | COD.AllocHidden, 2, 0);
        chart.CloseData(COD.Values);

        //chart.Printer.Compress = true;
        chart.Printer.ForceColors = true;
        //chart.Printer.UsePrinterResolution = true;
        chart.Printer.Orientation = SoftwareFX.ChartFX.Orientation.Landscape;

        // axis setup
        chart.AxisX.Title.Text = "Time (s)";
        chart.AxisX.Step = 20;
        chart.AxisX.Min = 0;
        chart.AxisX.Max = TimeBase;
        chart.AxisX.AutoScale = false;

        chart.AxisY.Title.Text = "Temperature (°C)";
        chart.AxisY.Step = 20;
        chart.AxisY.Min = 0;
        chart.AxisY.Max = TempBase;
        chart.AxisY.AutoScale = false;

        chart.Scrollable = false;
```

```
                        // gfx toolbar
                        SoftwareFX.ChartFX.Annotation.AnnotationX annot = \
                                                new SoftwareFX.ChartFX.Annotation.AnnotationX();
                        annot.Enabled =true;
                        chart.Extensions.Add(annot);

                        ResetCurves();
                }

                protected override void Dispose( bool disposing )
                {
                        if( disposing )
                        {
                                if (components != null)
                                {
                                        components.Dispose();
                                }
                        }
                        if( oven != null )
                                oven.Dispose();
                        base.Dispose( disposing );
                }

                #region Code généré par le Concepteur Windows Form

                private void InitializeComponent()
                {
                        SoftwareFX.ChartFX.Borders.GradientBackground gradientBackground1 = \
                                                new SoftwareFX.ChartFX.Borders.GradientBackground();
                        SoftwareFX.ChartFX.Borders.ImageBorder imageBorder1 = \
new SoftwareFX.ChartFX.Borders.ImageBorder(SoftwareFX.ChartFX.Borders.ImageBorderType.Butterfly);
                        SoftwareFX.ChartFX.SeriesAttributes seriesAttributes1 = \
                                                        new SoftwareFX.ChartFX.SeriesAttributes();
                        SoftwareFX.ChartFX.SeriesAttributes seriesAttributes2 = \
                                                        new SoftwareFX.ChartFX.SeriesAttributes();
                        SoftwareFX.ChartFX.TitleDockable titleDockable1 = \
                                                        new SoftwareFX.ChartFX.TitleDockable();
                        this.chart = new SoftwareFX.ChartFX.Chart();
                        this.SuspendLayout();
                        //
                        // chart
                        //
                        this.chart.AxisX.Gridlines = true;
                        this.chart.AxisX.Step = 10;
                        this.chart.AxisY.Gridlines = true;
                        this.chart.AxisY.LabelsFormat.Decimals = 0;
                        this.chart.AxisY.MinorTickMark = SoftwareFX.ChartFX.TickMark.Cross;
                        this.chart.AxisY.Step = 10;
                        gradientBackground1.Type = \
                                                SoftwareFX.ChartFX.Borders.GradientType.ForwardDiagonal;
                        this.chart.BackObject = gradientBackground1;
                        imageBorder1.Type = SoftwareFX.ChartFX.Borders.ImageBorderType.Butterfly;
                        this.chart.BorderObject = imageBorder1;
                        this.chart.DataStyle = SoftwareFX.ChartFX.DataStyle.ReadXValues;
                        this.chart.Dock = System.Windows.Forms.DockStyle.Fill;
                        this.chart.Gallery = SoftwareFX.ChartFX.Gallery.Lines;
                        this.chart.InsideColor = System.Drawing.Color.Transparent;
                        this.chart.LineWidth = 3;
                        this.chart.Location = new System.Drawing.Point(0, 0);
                        this.chart.MarkerShape = SoftwareFX.ChartFX.MarkerShape.None;
                        this.chart.Name = "chart";
                        this.chart.NSeries = 1;
                        this.chart.NValues = 20;
                        this.chart.Palette = "Nature.Sky";
                        seriesAttributes2.Color = \
        System.Drawing.Color.FromArgb(((System.Byte)(255)), ((System.Byte)(128)), ((System.Byte)(0)));
                        this.chart.Series.AddRange(new SoftwareFX.ChartFX.SeriesAttributes[] {
                                                        seriesAttributes1,
                                                        seriesAttributes2});
                        this.chart.Size = new System.Drawing.Size(928, 653);
                        this.chart.SmoothFlags = ((SoftwareFX.ChartFX.SmoothFlags)(((SoftwareFX.ChartFX.
SmoothFlags.Fill | SoftwareFX.ChartFX.SmoothFlags.Border)
                                        | SoftwareFX.ChartFX.SmoothFlags.Text)));
                        this.chart.TabIndex = 5;
                        this.chart.Titles.AddRange(\
                                        new SoftwareFX.ChartFX.TitleDockable[]{titleDockable1});
```

```
                this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                this.ClientSize = new System.Drawing.Size(928, 653);
                this.Controls.Add(this.chart);
                this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
                this.Text = "Reflow Oven";
                this.ResumeLayout(false);

        }
        #endregion

        [STAThread]
        static void Main()
        {
                try
                {
                        Application.Run(new Form1());
                }

                catch( Exception ex )
                {
                        MessageBox.Show( "Error: " + ex.Message );
                        Application.ExitThread();
                }
        }
    }
}
```

com.cs

```
using System;
using System.Threading;
using System.IO;
using System.Runtime.InteropServices;

namespace ReflowOven
{
    public class Com : Stream
    {
        #region settings & const

        private const int BufferSize = 1600;
        private const int Timeout = 50; // ms

        public enum ParityMode
        {
                Even,
                Mark,
                None,
                Odd,
                Space
        };

        #endregion

        #region members

        private Win32.OVERLAPPED ovr;
        private Win32.OVERLAPPED ovw;
        private ManualResetEvent ovrEvent = new ManualResetEvent( false );
        private ManualResetEvent ovwEvent = new ManualResetEvent( false );

        private bool init = false;
        private int portNum = 0;
        private IntPtr hCom = IntPtr.Zero;

        #endregion

        #region ctor & dtor

        public Com()
        {
                ovr.Internal = 0;
                ovr.InternalHigh = 0;
                ovr.Offset = 0;
                ovr.OffsetHigh = 0;
                ovw.Internal = 0;
                ovw.InternalHigh = 0;
                ovw.Offset = 0;
```

```
            ovw.OffsetHigh = 0;

            ovr.hEvent = ovrEvent.Handle;
            ovw.hEvent = ovwEvent.Handle;
        }

        public void Open( int portNum, int baudRate, ParityMode parity, int byteSize )
        {
            try
            {
                if( init )
                    return;

                this.portNum = portNum;

                hCom = Win32.CreateFile( "COM" + portNum.ToString(),
                    Win32.GENERIC_READ | Win32.GENERIC_WRITE,
                    0,
                    IntPtr.Zero,
                    Win32.OPEN_EXISTING,
                    Win32.FILE_FLAG_OVERLAPPED,
                    IntPtr.Zero );

                if( hCom.ToInt32() == Win32.INVALID_HANDLE_VALUE )
                    throw new IOException( \
                        "Can't open COM" + portNum.ToString() + " (create)" );

                if( Win32.SetupComm( hCom, BufferSize, BufferSize ) == 0 )
                    throw new IOException( \
                        "Can't open COM" + portNum.ToString() + " (buffer)" );

                if( Win32.PurgeComm( hCom, Win32.PURGE_TXABORT | Win32.PURGE_RXABORT | \
                        Win32.PURGE_TXCLEAR | Win32.PURGE_RXCLEAR ) == 0 )
                    throw new IOException( \
                        "Can't open COM" + portNum.ToString() + " (purge)" );

                SetDataFormat( baudRate, parity, byteSize );

                init = true;
            }

            catch
            {
                if( hCom != IntPtr.Zero )
                {
                    if( hCom.ToInt32() != Win32.INVALID_HANDLE_VALUE )
                        Win32.CloseHandle( hCom );

                    init = false;
                    hCom = IntPtr.Zero;
                }
                throw;
            }
        }

        private void SetDataFormat( int baudRate, ParityMode parity, int byteSize )
        {
            try
            {
                Win32.DCB                       DCB, DCBDump;
                Win32.COMMTIMEOUTS              cto;

                DCB.dcbLength = Marshal.SizeOf( typeof( Win32.DCB ));
                DCB.baudRate = baudRate;
                DCB.byteSize = (byte) byteSize;
                DCB.eofChar = 0;
                DCB.errorChar = 0;
                DCB.evtChar = 0;

                if( parity == ParityMode.Even )
                    DCB.parity = Win32.EVENPARITY;
                else if( parity == ParityMode.Mark )
                    DCB.parity = Win32.MARKPARITY;
                else if( parity == ParityMode.None )
                    DCB.parity = Win32.NOPARITY;
                else if( parity == ParityMode.Odd )
                    DCB.parity = Win32.ODDPARITY;
```

```
                else if( parity == ParityMode.Space )
                        DCB.parity = Win32.SPACEPARITY;
                else throw new ApplicationException();

                //DCB.bitfield = DCB.parity == 0 ? 1 : 3;
                DCB.bitfield = 1;
                DCB.stopBits = 0;
                DCB.wReserved = 0;
                DCB.wReserved1 = 0;
                DCB.xoffChar = 0;
                DCB.xoffLim = 0;
                DCB.xonChar = 0;
                DCB.xonLim = 0;

                if( Win32.SetCommState( hCom, ref DCB ) == 0)
                        throw new IOException();

                //make sure it's really set
                if( Win32.GetCommState( hCom, out DCBDump ) == 0 )
                        throw new IOException();

                if( !Win32.DCB.Equals( DCB, DCBDump ) )
                        throw new IOException();

                cto.ReadIntervalTimeout = Win32.MAXDWORD;
                cto.ReadTotalTimeoutMultiplier = 0;
                cto.ReadTotalTimeoutConstant = 0;
                cto.WriteTotalTimeoutMultiplier = 0;
                cto.WriteTotalTimeoutConstant = 0;

                if( Win32.SetCommTimeouts( hCom, ref cto ) == 0 )
                        throw new IOException();
        }

        catch
        {
                throw new IOException( \
                        "Can't set COM" + portNum.ToString() + "data format to: " +
                        byteSize.ToString() + "bits, " +
                        baudRate.ToString() + "bps, " +
                        "Parity: " + parity.ToString());
        }
}

public override void Close()
{
        if( !init )
                return;

        init = false;
        Win32.CloseHandle( hCom );
}

~Com()
{
        Close();
}


#endregion

#region Stream members

public override bool CanRead
{ get { return true; } }

public override bool CanWrite
{ get { return true; } }

public override bool CanSeek
{ get { return false; } }

public override long Length
{
        get
        {
                throw new NotSupportedException();
```

```
        }
    }

    public override long Position
    {
        get
        {
            throw new NotSupportedException();
        }
        set
        {
            throw new NotSupportedException();
        }
    }

    public override void SetLength(long value)
    {
        throw new NotSupportedException();
    }


    public override long Seek(long offset, SeekOrigin origin)
    {
        throw new NotSupportedException();
    }

    public override void Flush()
    {
        return;
    }

    public override int Read( byte[] buffer, int offset, int count )
    {
        int br;
        int ptr = offset;
        int totalbr = 0;

        while( totalbr < count )
        {
            DateTime start = DateTime.Now;
            while( true )
            {
                unsafe
                {
                    fixed( byte* pBuffer = &buffer[ ptr ] )
                    {
                        if( Win32.ReadFile( \
hCom, ( IntPtr )pBuffer, count - totalbr, out br, ref ovr ) == 0 && \
Marshal.GetLastWin32Error() != Win32.ERROR_IO_PENDING )
                            throw new IOException( \
                            "Can't access to COM" + portNum.ToString() );
                    }
                }

                if( br != 0 )
                    break;

                TimeSpan delta = DateTime.Now - start;
                if( delta.TotalMilliseconds > Timeout )
                    return totalbr;

                Thread.Sleep( 1 );
            }
            ptr += br;
            totalbr += br;
        }
        return totalbr;
    }

    public override void Write(byte[] buffer, int offset, int count)
    {
        if( !init )
            throw new IOException( "Port COM not initialized" );

        unsafe
        {
            fixed( byte* pBuffer = &buffer[ offset ] )
```

```
                        {
                                int bw = 0;
                                if( Win32.WriteFile( hCom, (IntPtr)pBuffer, count, out bw, ref ovw
) == 0 &&
                                        Marshal.GetLastWin32Error() != Win32.ERROR_IO_PENDING )
                                        throw new IOException( \
                                                "COM" + portNum.ToString() + " can't send" );
                        }
                }
        }

        #endregion

        #region Imports
        public class Win32
        {
                [StructLayout(LayoutKind.Sequential)]
                public struct OVERLAPPED
                {
                        public int Internal;
                        public int InternalHigh;
                        public int Offset;
                        public int OffsetHigh;
                        public IntPtr hEvent;
                }

                [StructLayout(LayoutKind.Sequential)]
                public struct COMMTIMEOUTS
                {
                        public int ReadIntervalTimeout;
                        public int ReadTotalTimeoutMultiplier;
                        public int ReadTotalTimeoutConstant;
                        public int WriteTotalTimeoutMultiplier;
                        public int WriteTotalTimeoutConstant;
                }


                [StructLayout(LayoutKind.Sequential)]
                public struct DCB
                {
                        public int              dcbLength;
                        public int              baudRate;
                        public int              bitfield;
                        public short            wReserved;
                        public short            xonLim;
                        public short            xoffLim;
                        public byte             byteSize;
                        public byte             parity;
                        public byte             stopBits;
                        public byte             xonChar;
                        public byte             xoffChar;
                        public byte             errorChar;
                        public byte             eofChar;
                        public byte             evtChar;
                        public short            wReserved1;
                }

                public const int MAXDWORD                       = -1;
                public const int GENERIC_READ                   = -2147483648;
                public const int GENERIC_WRITE                  = 1073741824;
                public const int OPEN_EXISTING                  = 3;
                public const int FILE_FLAG_OVERLAPPED           = 1073741824;
                public const int INVALID_HANDLE_VALUE           = -1;
                public const int PURGE_TXABORT                  = 1;
                public const int PURGE_RXABORT                  = 2;
                public const int PURGE_TXCLEAR                  = 4;
                public const int PURGE_RXCLEAR                  = 8;
                public const int ERROR_IO_INCOMPLETE            = 996;
                public const int ERROR_IO_PENDING               = 997;
                public const int NOPARITY                       = 0;
                public const int ODDPARITY                      = 1;
                public const int EVENPARITY                     = 2;
                public const int MARKPARITY                     = 3;
                public const int SPACEPARITY                    = 4;

                [ DllImport( "KERNEL32.DLL", SetLastError = true, CharSet = CharSet.Ansi ) ]
                        public static extern IntPtr CreateFile( string fileName, \
```

```
                        int desiredAccess, int shareMode, IntPtr securityAttributes,\
                        int creationDisposition, int flagsAndAttributes, IntPtr templateFile );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int SetupComm( IntPtr hFile, int dwInQueue, \
                                                                        int dwOutQueue );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int PurgeComm( IntPtr hFile, int dwFlags );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int CloseHandle( IntPtr handle );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int ReadFile ( IntPtr hFile, IntPtr lpBuffer, \
        int nNumberOfBytesToRead, out int lpNumberOfBytesRead, ref OVERLAPPED lpOverlapped );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int GetOverlappedResult(IntPtr hFile, \
                ref OVERLAPPED lpOverlapped, out int lpNumberOfBytesTransferred, int bWait );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int WriteFile( IntPtr hFile, IntPtr lpBuffer, \
        int nNumberOfBytesToWrite, out int lpNumberOfBytesWritten, ref OVERLAPPED lpOverlapped );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int SetCommTimeouts ( IntPtr hFile , \
                                                        ref COMMTIMEOUTS lpCommTimeouts );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int SetCommState ( IntPtr hCommDev , ref DCB lpDCB );

                [ DllImport( "KERNEL32.DLL", SetLastError = true ) ]
                public static extern int GetCommState ( IntPtr hCommDev , out DCB lpDCB );
        }

        #endregion
    }
}
```

oven.cs

```
using System;
using System.Threading;
using System.IO;
using System.Windows.Forms;

namespace ReflowOven
{
    public delegate void OvenDelegate();

    public class Oven : IDisposable
    {
        public OvenCmd cmd;
        public event OvenDelegate StatusInvalided;
        public event OvenDelegate RunStarted;


        #region Properties
        public float TempActual;
        public float TempSet;
        public int Time;

        #endregion

        public void TriggerStatusInvalided()
        {
            if( StatusInvalided != null )
                StatusInvalided();
        }

        public void TriggerRunStarted()
        {
            if( RunStarted != null )
                RunStarted();
        }

        #region ctor & dtor
```

```
        public Oven()
        {
                cmd = new OvenCmd( this );
        }

        public void Dispose()
        {
                cmd.Dispose();
                GC.SuppressFinalize( this );
        }

        #endregion

}

public class OvenCmd : IDisposable
{
        public enum Command
        {
                SetPower = 0x00,
                UpdateStatus = 0x01,
                RunStarted = 0x02
        }

        private Com com = new Com();
        private PacketBuilder pb;
        private Oven oven;

        #region ctor & dtor

        public OvenCmd( Oven oven )
        {
                this.oven = oven;
                com.Open( 5, 19200, Com.ParityMode.None, 8 );
                pb = new PacketBuilder( com );
                pb.PacketReceived += new ReflowOven.PacketBuilder.PacketEvent( PacketReceived );
        }

        public void Dispose()
        {
                pb.Dispose();
                com.Close();
                GC.SuppressFinalize( this );
        }

        #endregion

        private void PacketReceived( Command cmd, BinaryReader br )
        {
                if( cmd == Command.UpdateStatus )
                {
                        oven.TempActual = br.ReadSingle();
                        oven.TempSet = br.ReadSingle();
                        oven.Time = br.ReadUInt16();
                        oven.TriggerStatusInvalided();
                }

                else if( cmd == Command.RunStarted )
                {
                        oven.TriggerRunStarted();
                }
        }

        #region Commands

        public void SetPower( byte power )
        {
                pb.StartPacket( Command.SetPower );
                pb.bw.Write( power );
                pb.EndPacket();
        }

        #endregion
}

public class PacketBuilder : IDisposable
```

```
{
        private static readonly byte[] Header = { 0x13, 0x37 };
        private readonly Com com;
        private MemoryStream msw = new MemoryStream();
        public BinaryWriter bw;
        private Thread tRead;

        public delegate void PacketEvent( OvenCmd.Command cmd, BinaryReader br );
        public event PacketEvent PacketReceived;

        private bool BuildingPacket = false;

        public PacketBuilder( Com com )
        {
                this.com = com;
                bw = new BinaryWriter( msw );
                tRead = new Thread( new ThreadStart( ReadThread ) );
                tRead.Start();
        }

        public void Dispose()
        {
                tRead.Abort();
                tRead.Join();
                GC.SuppressFinalize( this );
        }


        public void StartPacket( OvenCmd.Command cmd )
        {
                if ( BuildingPacket )
                        throw new ArgumentException();
                BuildingPacket = true;
                msw.SetLength(0);

                bw.Write( Header[0] );
                bw.Write( Header[1] );
                bw.Seek( 1, SeekOrigin.Current ); // skip length
                bw.Write( (byte)cmd );
        }

        public void EndPacket()
        {
                if ( !BuildingPacket )
                        throw new ArgumentException();
                BuildingPacket = false;


                bw.Seek( 2, SeekOrigin.Begin );
                bw.Write( (byte)( msw.Length - 3 ) );

                bw.Seek( 2, SeekOrigin.Begin );

                ushort crc = 0;
                int current;

                while( (current = msw.ReadByte()) != -1 )
                {
                        crc = (ushort)( (crc >> 8) | (crc << 8) );
                        crc ^= (byte)current;
                        crc ^= (ushort)( (crc & 0xFF) >> 4 );
                        crc ^= (ushort)( crc << 12 );
                        crc ^= (ushort)((crc & 0xFF) << 5);
                }

                bw.Write( crc );

                byte[] buffer = new byte[ msw.Length ];
                msw.Position = 0;
                msw.Read( buffer, 0, (int)msw.Length );
                com.Write( buffer, 0, (int)msw.Length );
        }

        private void ReadThread()
        {
                byte[] buffer = new byte[ 0x1000 ];
                int offset = 0;
```

```
                        bool receiving = false;
                        int packetLenght = 0;
                        ushort crc = 0;

                        while( true )
                        {
                                if( !receiving )
                                {
                                        crc = 0;
                                        offset = 0;
                                }

                                if(  com.Read( buffer, offset, 1 ) == 0 )
                                {
                                        // timed-out
                                        receiving = false;
                                        continue;
                                }

                                receiving = true;
                                offset++;

                                switch( offset )
                                {
                                        case 1:
                                                if(  buffer[0] != Header[0] )
                                                        receiving = false;
                                                continue;
                                        case 2:
                                                if(  buffer[1] != Header[1] )
                                                        receiving = false;
                                                continue;
                                        case 3:
                                                packetLenght = buffer[2];
                                                if( packetLenght == 0 )
                                                        receiving = false;
                                                break;

                                        default:
                                                if( offset == packetLenght + 4 )
                                                {
                                                        if(  buffer[ offset-1 ] != (crc & 0xFF) )
                                                                receiving = false;
                                                        continue;
                                                }

                                                else if( offset == packetLenght + 5 )
                                                {
                                                        if(  buffer[ offset-1 ] == (crc >> 8) )
                                                        {
                                                                MemoryStream stream = \
                                                                        new MemoryStream( packetLenght-1 );
                                                                stream.Write( buffer, 4, packetLenght-1 );
                                                                stream.Position = 0;
                                                                PacketReceived( \
                                        ( OvenCmd.Command )buffer[ 3 ], new BinaryReader( stream ) );
                                                        }
                                                        receiving = false;
                                                        continue;
                                                }
                                                break;
                                }

                                crc = (ushort)( (crc >> 8) | (crc << 8) );
                                crc ^= buffer[ offset-1 ];
                                crc ^= (ushort)( (crc & 0xFF) >> 4 );
                                crc ^= (ushort)( crc << 12 );
                                crc ^= (ushort)((crc & 0xFF) << 5);
                        }
                }
        }
}
```

...:: CONCLUSION ::...

Réaliser ce projet de A à Z m'a couter environ 250 heures de travail intensif.
J'ai passe 4/5 du temps impartis à écrire le code source.
Ce projet m'aura permis d'en apprendre encore plus dans le monde de l'électronique, d'aborder des problèmes d'asservissement, de temps réel et surtout de mise en pratique de concepts theoriques.

...:: BIBLIOGRAPHIE ::...

[LX] Linx Technologies, Inc. TXE-***-KH Datasheet http://www.linxtechnologies.com/images/products_cat/rf_modules/kh_series/kh-rxd_manual.pdf

[CE] Ceramicx Ireland Ltd. Full Quartz Element (FQE) Heat Up Graph Over 10 Minutes http://www.ceramicx.com/Products/Quartz%20Elements/QuartzHeatUpCurveQ.htm

[MO] Molex 4.20mm (.165") Pitch Mini-Fit Jr.™ Header, Dual Row, Right Angle, with Snap-in Plastic Peg PCB Lock Datasheet http://www.newproduct.molex.com/datasheet.aspx?ProductID=24219

[PS] Philips Semiconductors Triacs BTA140B series Datasheet http://www.semiconductors.philips.com/acrobat/datasheets/BTA140B_SERIES_1.pdf

[FS] Fairchild Semiconductor MOC3041-M 6-Pin DIP 400V Zero Crossing Triac Driver Output Optocoupler Datasheet http://www.fairchildsemi.com/ds/MO%2FMOC3041-M.pdf

[ST1] STMicroelectronics LD1117D50 Low Drop Fixed And Adjustable Positive Voltage Regulators http://www.st.com/stonline/books/pdf/docs/2572.pdf

[M1] Microchip MASTERs Conference 2003 IC Thermal Circuits Solve Temperature Measurement Problems Presentation http://techtrain.microchip.com/masters2003/(icv3o045dzajeemjx3ld2u3m)/masters2003-CD/classes/727/727_TMP.pdf

[M2] Microchip AN895 - Oscillator Circuits for RTD Temperature Sensors http://ww1.microchip.com/downloads/en/AppNotes/00895a.pdf

[BT] NTC Thermistor Theory http://www.betatherm.com/indextheory.php

[PY] Resistive Temperature Detectors by PAUL Y. OH http://www.pages.drexel.edu/~pyo22/mem351-2004/lecture03/pp041-047-Rtd.pdf

[HW] HEL-700 Series Temperature Sensor, Platinum RTD, Ceramic case leadwire datasheet http://catalog.sensing.honeywell.com/printfriendly.asp?FAM=temperature&PN=HEL-707-U-0-12-00

[NS] National Semiconductor Temperature Sensor Handbook http://micro.et-inf.fho-emden.de/datenblaetter/sensor/temphb.pdf

[ST2] STMicroelectronics 5V POWERED MULTI-CHANNEL RS-232 DRIVERS AND RECEIVERS ST232 http://www.st.com/stonline/books/pdf/docs/6420.pdf

[M3] Microchip AN730 - CRC Generating and Checking http://ww1.microchip.com/downloads/en/AppNotes/00730a.pdf