Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Filière Systèmes industriels

Orientation Power and Control

Diplôme 2007

Matteo Belometti

Réglage par pièce
(gestion du chauffage et des stores)

Professeur	Fariba Bützberger
Expert	André Rotzetta

Sion, le 23 novembre 2007

HES-SO Valais

SI	TV	EE	IG	EST
X	X	X	X	

Données du travail de diplôme
Daten der Diplomarbeit

FO.0.2.02.07.DB
DD/18/05/2006

Filière / ***Studiengang*** : Systèmes industriels

Confidentiel / ***Vertraulich*** ☐

Etudiant / ***Student***	**Année scolaire /** ***Schuljahr***	**No TD /** ***Nr. DA***
Matteo Belometti	2006/07	SI/2007/17

Proposé par / ***vorgeschlagen von***	**Lieu d'exécution /** ***Ausführungsort***
HES-SO Valais, UPC	HES-SO Valais, DSI **Expert /** ***Experte*** André Rotzetta

Titre / ***Titel*****:**

Réglage par pièce (gestion du chauffage et des stores)

Description / ***Beschreibung*****:**

Conception et réalisation d'un système de réglage de température et de gestion des stores d'une pièce habitable afin d'améliorer le confort des habitants et d'optimiser la consommation d'énergie.

Les grandeurs de contre-réaction sont la température intérieure et extérieure de la pièce en question, la présence dans la pièce et le rayonnement solaire. Pour une installation aisée du système, on propose une communication sans fil (RF) entre les différents capteurs et les actuateurs.

L'équipement déployé pour le système de régulation est composé de :

- Une carte d'émission, placée à l'intérieure de la pièce contenant le capteur de température intérieure, le capteur de présence, ainsi que l'émetteur RF.
- Une carte d'émission placée à l'extérieure de la pièce contenant le capteur de température extérieure, le capteur de rayonnement solaire, ainsi que l'émetteur RF.
- Une Carte de réception ou le module de commande et de régulation, intégrant principalement le microcontrôleur et le récepteur RF, ainsi que l'interface utilisateur, pour l'introduction des consignes et le taux d'occupation du local selon une grille horaire.
- Actuateurs : servo-vannes électriques sur les radiateurs + servo-moteurs électriques sur les stores.

Objectifs / ***Ziele:***

- Test des différentes cartes réalisées pendant le projet de semestre.
- Test de la communication RF entre les émetteurs et le récepteur.
- Implémentation des algorithmes de commande et de régulation sur le micro-contrôleur.
- Le suivi expérimental sera effectué au collège 'Les Creusets' à Sion, sur 2 salles de classes identiques et voisines, avec même orientation et fréquentation. Des data-loggers et des compteurs de chaleur sont à disposition pour comparer les 2 salles selon les 2 critères : le confort des utilisateurs et la consommation d'énergie.

Signature ou visa / ***Unterschrift oder Visum***	**Délais /** ***Termine***
Resp. de l'orientation power and control ..... Professeur/*Dozent*: Fariba Bützberger ..... Etudiant/*Student*:	Attribution du thème / Ausgabe des Auftrags: 03.09.2007 Remise du rapport / Abgabe des Schlussberichts: 23.11.2007 Exposition publique / Ausstellung Diplomarbeiten: 30.11.2007 Défenses orales / Mündliche Verfechtungen Semaine 49

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats:* .....

Réglage par pièce (gestion du chauffage et des stores)

Raum-Regelung (Steuerung von Heizung und Rolladen)

Objectif Le but de ce travail de diplôme est de concevoir un système de gestion du chauffage et des stores pièce par pièce, qui permette d'optimiser la consommation énergétique du chauffage en prenant en considération des facteurs tels que température, présence dans la pièce et rayonnement solaire. Le système agit sur des servovanne thermostatique et sur les moteurs des stores.

Résultats Les tests effectués permettent d'affirmer que le système de régulation conçu est fonctionnel et apporte des bénéfices en termes d'économie d'énergie et de précision.

Mots-clés Régulation, chauffage, température, PID, microcontrôleur, consommation énergétique, transmission RF, émetteur, récepteur

Ziel *Das Ziel dieser Diplomarbeit ist die Entwicklung eines Heizungs- und Rolladen-Steuersystems, das das Optimieren des Energieverbrauches ermöglicht. Es berücksichtigt Faktoren wie die Temperatur, diePräsenz von Personen in Raum und dieSonnenstrahlung. Das System wirkt auf Thermoventile und Rolladenmotoren.*

Resultate *Die durchgeführten Tests zeigen dass, das entwickelte System funktional ist, und eine Abnahme des Energieverbrauchs ermöglicht, und die Genauigkeit der Regelung verbessert.*

Schlüsselwörter *Regelung, Heizung, Temperatur, PID, Mikrocontroller, Energieverbrauch, RF Übertragung, Sender, Empfänger*

Table des matières

1 Introduction

1.1 Présentation du projet

Il s'agit de concevoir et réaliser un système de réglage de température et gestion des stores « local par local » qui soit attentif aux besoins et souhaits des habitants.

La partie hardware a déjà été réalisée pendant le projet de semestre qui a précédé le travail de diplôme. Il s'agit maintenant d'étudier et implanter la partie software qui permettra la régulation du chauffage.

Le système prend en compte les facteurs météorologiques et les apports de chaleur gratuits : il met en œuvre un microcontrôleur relié à des sondes de température intérieures et extérieures, de présence dans la pièce et de rayonnement solaire.

Ces sondes sont reliées au régulateur par le biais d'une transmission sans fil.

Le régulateur intervient sur des vannes thermostatiques qui règlent les radiateurs, ainsi que sur les servomoteurs qui gèrent l'ouverture des stores.

Le système a la structure suivante :

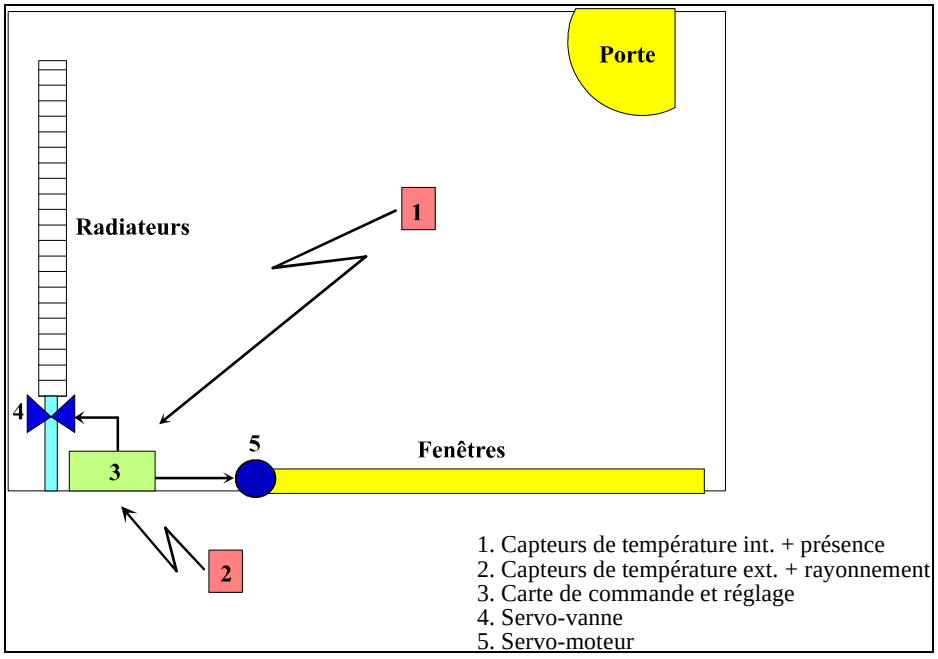


Figure 1: vue d'ensemble du système

Il sera donc composé de 3 cartes électroniques et 2 actuateurs, comme décrit dans la figure1.

Le réglage effectué permettra de :

- Améliorer le confort
- économiser et donc optimiser l'énergie consommée

1.2 Problème et état de l'art

À l'état actuel, les systèmes de réglage du chauffage les plus répandus utilisent des vannes thermostatiques : ce qui permet d'obtenir un confort satisfaisant mais un gaspillage d'énergie pas négligeable.

Dans des tels systèmes, la sonde de température étant placée près du radiateur, la température n'est pas représentative de celle du local; de plus, sans un effet prédictif du régulateur, ces vannes coupent trop tard, ce qui génère des dépassements de la consigne.

Il existe également sur le marché des systèmes de réglage (par exemple chez Siemens) qui permettent d'introduire des consignes variables dans le temps.
Il s'agit souvent d'un régulateur P ou « tout ou rien », permettant de mieux repartir l'énergie dans le bâtiment, mais ils sont loin d'être optimales.

De plus, ces systèmes ne prennent pas en compte les apports d'énergie gratuits tels que les rayons du soleil et la chaleur du corps des personnes humaines.

1.3 Objectifs principaux

Les objectifs de ce travail de diplôme sont :

- Test des cartes réalisées pendant le projet de semestre et de la communication radio entre les émetteurs et le récepteur RF.

- Conception de la partie software du système; c'est-à-dire la programmation des microcontrôleurs des cartes émetteur, qui s'occupent de l'acquisition des données de l'environnement, et la conception et implantation d'un algorithme de réglage.

- L'exécution de tests pratiques du système dans deux salles voisines identiques et de même orientation. Des dataloggers et des compteurs de chaleur permettront la comparaison entre les deux salles en mesurant le confort des utilisateurs et la consommation énergétique. Une analyse des données sera nécessaire afin de démontrer l'efficacité du système et le bénéfice apporté.

1.4 Cahier des charges

Le système de réglage possède les entrées suivantes :

- Température externe
- Température interne
- Ensoleillement
- Présence dans la pièce

Les sorties sont :
- Ouverture servovanne thermostatique
- Ouverture/fermeture stores

Le prototype est composé de 3 cartes électroniques (voir figure 1).

Le réglage à effectuer sur le chauffage est du type PID, ce qui permet d'éviter toute erreur statique et d'avoir un effet prédictif de l'erreur.

Pour la gestion des stores on va concevoir un régulateur tout ou rien qui ouvre ou ferme totalement les stores selon des critères à définir, notamment en utilisant les données de rayonnement solaire et de présence dans la pièce.

Les étapes à suivre dans ce projet sont les suivantes :

- Test de la communication RF entre les cartes
- Etude de l'acquisition des données
- Conception et implantation de l'algorithme de réglage
- Tests du système
- Analyse des résultats obtenus et optimisation du réglage

2 Description Hardware

2.1 Introduction

Comme déjà mentionné le système est composé de 3 cartes : 2 émetteurs et 1 récepteur (régulateur), qui communiquent par ondes radio.
Ces cartes ont été développées pendant le projet de semestre précédant ce travail de diplôme.
Les 2 émetteurs sont alimentés par pile et le régulateur est alimenté directement par le réseau 230 V.

2.2 Schémas blocs

1) Carte émetteur interne

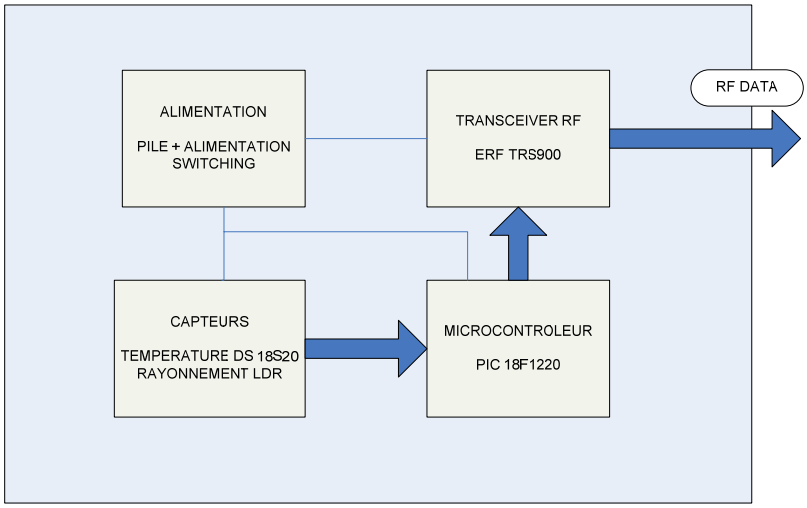


Figure 2: Carte émetteur interne

Cette carte s'occupe de détecter la présence de personnes dans la pièce et de mesurer la température ambiante; ensuite elle envoie ces données à la carte régulateur par transmission RF. Elle sera donc placée à l'intérieur de la pièce à régler.

2) Carte émetteur externe

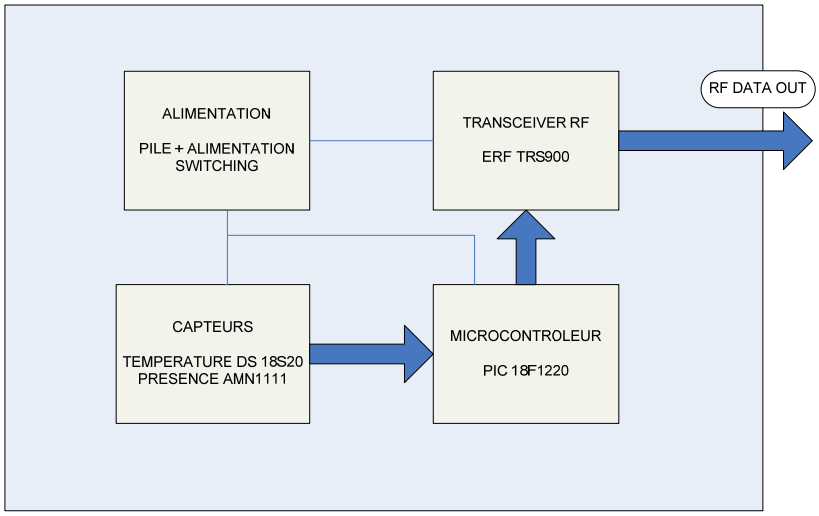


Figure 3: Carte émetteur externe

Cette carte s'occupe de détecter la présence des rayons solaires et de mesurer la température extérieure ; ensuite elle envoie ces données à la carte régulateur par transmission RF. Elle sera donc placée à l'extérieur.

3) Carte régulateur

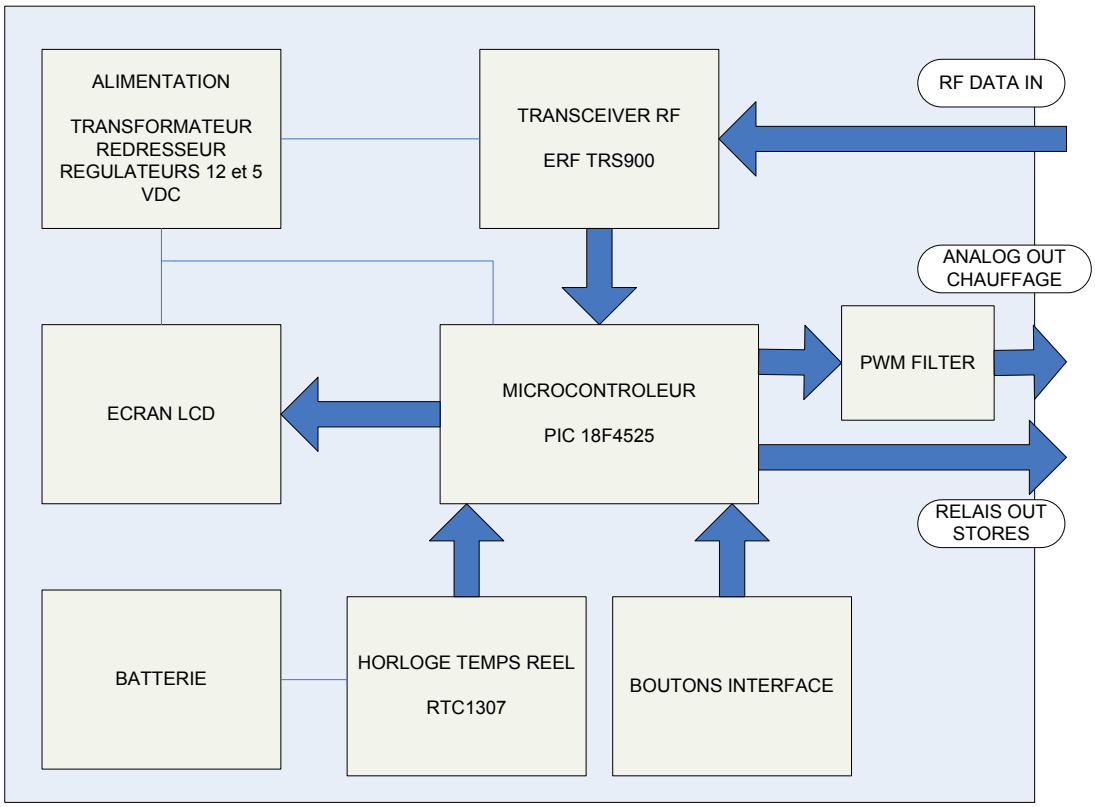


Figure 4: Carte régulation

Cette carte reçoit les données envoyées par les 2 autres cartes et contient le régulateur qui agit sur les radiateurs et les stores afin d'optimiser la consommation énergétique et de maximiser le confort. De plus, une interface permet à l'utilisateur de modifier certains paramètres tels que consigne et horaires de chauffage souhaités.

2.3 Description des principaux composants utilisé

2.3.1 Introduction

Le choix des composants a été effectué pendant le projet de semestre, donc je ne reviendrais pas sur les raisons des choix effectués, ni sur les possibilités existantes.
Je vais me limiter ici à une brève description des composants présents sur les cartes.
La programmation des éléments sera abordée dans le chapitre 5.
Les datasheets des composants se trouvent sur le CD en annexe 9.

2.3.2 Capteur de température:

Il s'agit d'un thermomètre digital de Dallas, le DS18S20, qui est très utilisé dans des applications de contrôle de température.
La température est fournie sur 8 bit, la résolution est de 0.5 °C et la plage de mesure va de -55°C à +125°C.

Voici un' image du capteur ainsi que son schéma interne :

Figure 5: Capteur DS18S20

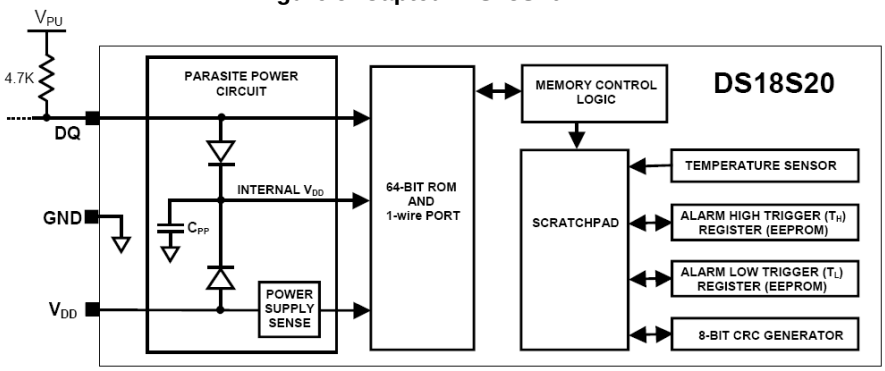


Figure 6: schéma interne DS18S20

Le temps de conversion maximale est de 750 ms et les données sont stockées dans un registre dénommé scratchpad, ou il faut aller les récupérer lors de la programmation.

SCRATCHPAD (Power-up State)		
byte 0	Temperature LSB (AAh	} (85°C)
byte 1	Temperature MSB (00h)	} (85°C)
byte 2	T_H Register or User Byte 1*	
byte 3	T_L Register or User Byte 2*	
byte 4	Reserved (FFh)	
byte 5	Reserved (FFh)	
byte 6	COUNT REMAIN (0Ch)	
byte 7	COUNT PER °C (10h)	
byte 8	CRC*	

*Power-up state depends on value(s) stored in EEPROM

Figure 7: registres du DS18S20

Une résolution meilleure, au millième de degré, est également réalisable avec ce capteur: pour l'utiliser il faut aller lire tous les registres du scratchpad et, en utilisant les valeurs des regisres, effectuer le calcul suivant :

$$Temperature = temp\_LSB - 0.25 + \frac{COUNT\_REMAIN - COUNT\_PER\_°C}{COUNT\_PER\_°C}$$

Le capteur communique avec le PIC à l'aide d'une entrée/sortie digitale ; le montage d'une résistance de pullup de 4,7 $k\Omega$ est nécessaire sur le bus de données.

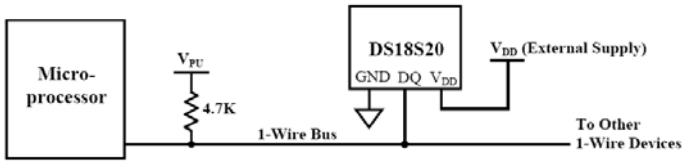


Figure 8: schéma montage DS18S20

Caractéristiques :

Température d'utilisation	-55 à 125°C
Tension d'alimentation	3 à 5.5 V
Consommation standby	500 nA
Consommation ON	4 mA
Temps de conversion température	750 us

2.3.3 Capteur de présence

Le capteur de présence utilisé est le AMN11111 de la maison Matsushita.
Il s'agit d'un capteur de mouvement infrarouge qu'on utilise pour détecter la présence de personnes dans une pièce.
Il possède une sortie digitale 0-5V.

Il a l'avantage d'être assez compact et d'intégrer un amplificateur du signal de sortie, ce qui permet une simple utilisation, c'est-à-dire de le relier directement à une entrée digitale du PIC.

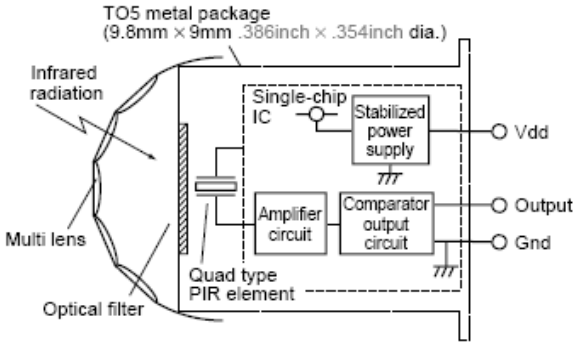


Figure 9: schéma interne capteur de présence

Caractéristiques :

Température d'utilisation	-20 à 60°C
Tension d'alimentation	3 à 6 V
Tension de sortie	Vdd / Vdd-5V
Consommation standby	100 uA
Consommation ON	270 uA
Temps de stabilisation	7 sec

La figure suivante montre les angles de détection du capteur qui sont 50° sur l'axe horizontal et 41° sur l'axe vertical:

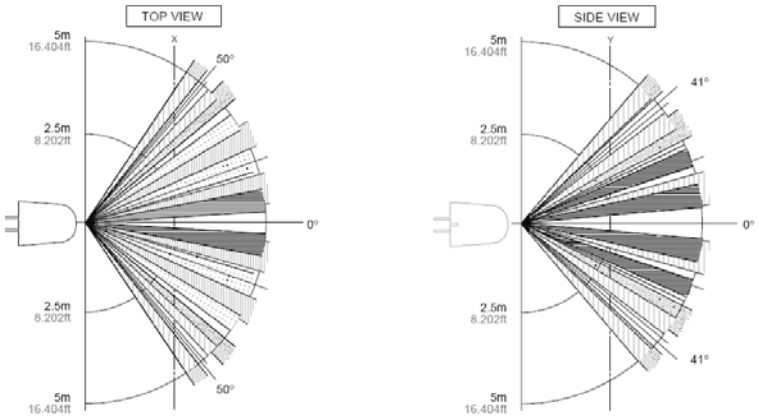


Figure 10: Angles de détection du AMN 11111

La section X-Y montre la zone de détection maximale d'environ 7.5x5.5 mètres ; les carrés représentent la projection des 16 lentilles de détection de la tête du capteur. Tout objet ayant une température différente de celle de l' « arrière-plan » est détecté.

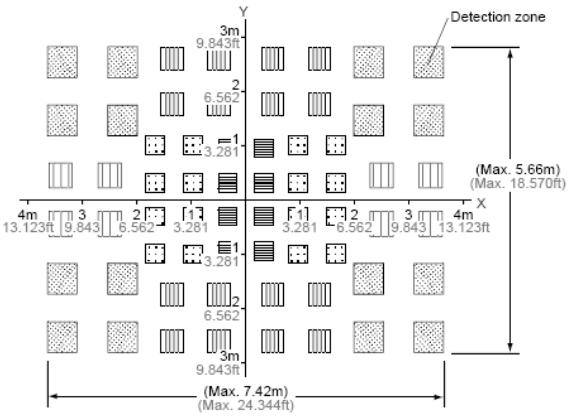


Figure 11: Zone maximale de détection du AMN 11111

2.3.4 Capteur de rayonnement solaire

Il s'agit d'une simple photorésistance, c'est-à-dire une résistance qui diminue sa valeur en fonction de l'augmentation de la lumière.
Le modèle est le A9060-14 de Perkin Elmer

Les valeurs caractéristiques sont :

- résistance dans le noir : 5MΩ
- résistance à 100 lux : 15kΩ
- centre du spectre : 600nm

Pour relier la mesure de lumière au PIC, un diviseur de tension a été effectué ; la tension à l'entrée du PIC est ainsi un signal digital qui varie entre 0 et 5 V.

Le PIC effectue ensuite la conversion analogique/digitale.

La lumière se mesure en Lux : ceci correspond à des lumens par mètre carré.
La lumière du soleil varie entre 32000 et 100000 lux, et dans une pièce bien illuminée on a 400-500 lux.

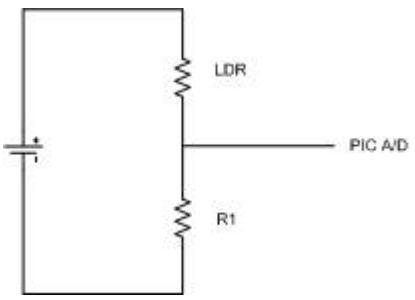


Figure 12: schéma utilisation LDR

La résistance R1 a été calculée de façon à avoir une plage de mesure la plus grande possible; avec R1= 1kOhm on obtient comme tension de sortie:

Tension d'alimentation	5 VDC
Plein soleil (rayonnement direct)	4.2-4.5 V
Ombre (dans une pièce)	1.7-2 V
Foncé (nuit)	0-0.2 V

Cela donne une consommation en courant maximale d'environ 4 mA en présence de soleil.
La photorésistance est alimentée seulement pendant quelques secondes lors de la conversion analogique de sa valeur ; ceci permet d'économiser la batterie de la carte émetteur.

L'alimentation du capteur est coupée à l'aide d'un mosfet piloté par le microcontrôleur, qui l'allume seulement en cas de besoin :

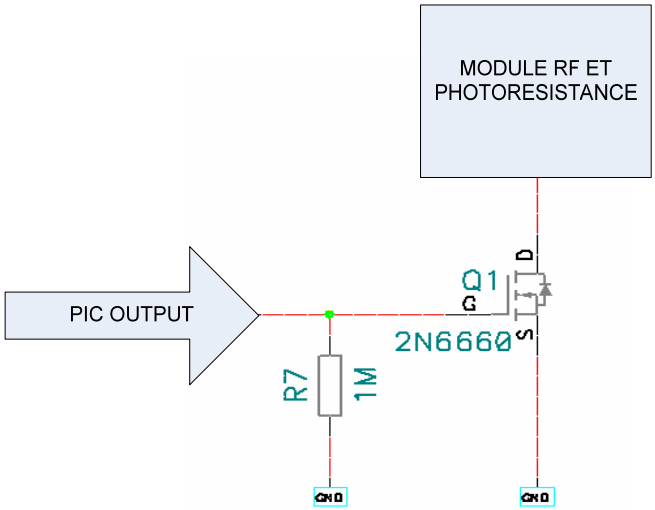


Figure 13: Coupure alimentation photorésistance et modules radio

2.3.5 Transmission RF

On utilise un module radio FM de la maison Easy Radio : il s'agit d'un module transceiver avec software embarqué, donc d'utilisation très facile, sans se soucier du protocole RF.
Ces modules communiquent avec un microprocesseur en utilisant une interface série du type UART (Universal Asynchronus Receiver Transmitter).
On ne nécessite d'aucun driver car les niveaux d'entrée/sortie du module sont déjà adaptés à cette interface.
Le module utilisé est le ER900TRS.

Figure 14: Easy radio transceiver

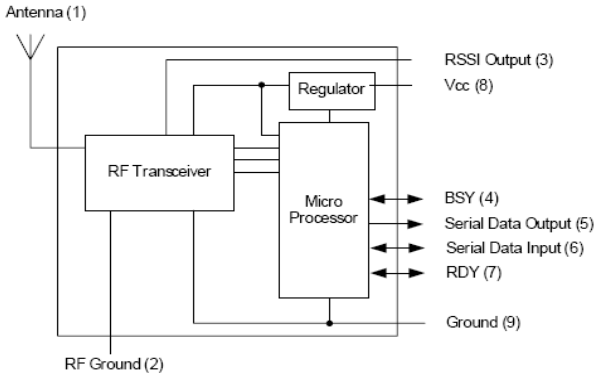


Figure 15: schéma interne module radio

Plusieurs paramètres peuvent être réglés :

- Baud rate
- Puissance du signal
- Choix des canaux et des fréquences de transmission correspondants

Les caractéristiques électriques de ces modules sont les suivantes :

Température d'utilisation	-40 à 85°C
Tension d'alimentation	3.6 / 5 V
Consommation standby	120 uA
Consommation ON	23 mA
Puissance du signal	-5 / +10 dBm
Fréquence	868 MHz

Les modules RF présents sur les cartes émetteurs seront alimentés seulement pendant les périodes d'envoi des données, afin d'augmenter la durée de la batterie d'alimentation. Cela est fait de la même façon que pour le capteur de rayonnement (voir page précédante).

2.3.6 PIC sur les cartes émetteurs

Le microcontrôleur utilisé sur les cartes émetteur est le PIC18F1220.
Ce microcontrôleur a les caractéristiques suivantes :

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	ECCP (PWM)	EUSART	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					
PIC18F1220	4K	2048	256	256	16	7	1	Y	1/3

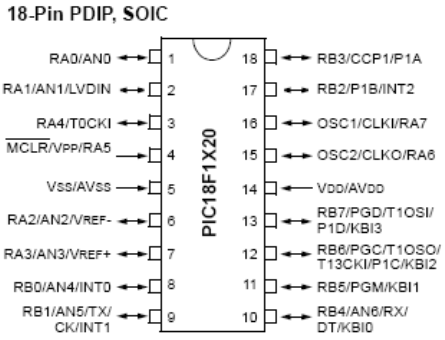


Figure 16: Diagramme des pins et caractéristiques PIC 18F1220

Les « périphériques » utilisées dans ce PIC sont l'interface sérielle UART et le module de conversion Analogique/Digitale.

2.3.7 PIC sur la carte régulateur

Sur la carte régulateur il y a un PIC18F4525.
Ce microcontrôleur a les caractéristiques suivantes :

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP/ ECCP (PWM)	MSSP		EUSART	Comp.	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI™	Master I^2C™			
PIC18F4525	48K	24576	3986	1024	36	13	1/1	Y	Y	1	2	1/3

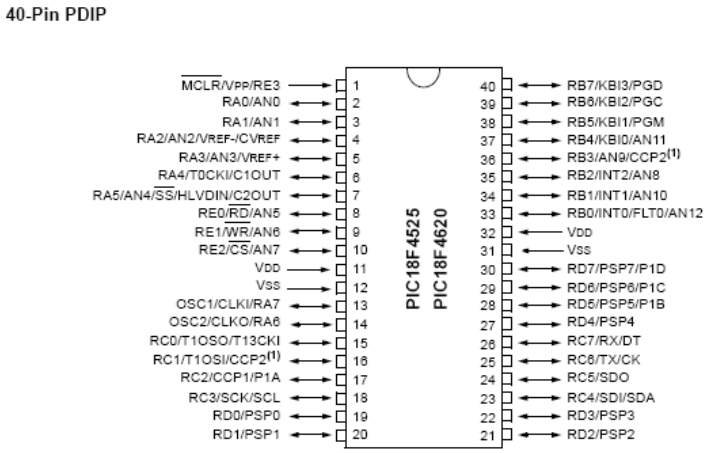


Figure 17: Diagramme des pins et caractéristiques PIC 18F4525

Les « périphériques » utilisées dans ce PIC sont : l'interface sérielle UART, le module CCP (PWM) et le module SPI (I2C).

2.3.8 Horloge temps réel

Pour ce système de réglage on a besoin de connaître en tout moment la date et l'heure actuelle, afin de déterminer les horaires de coupure du chauffage et de choisir des consignes différentes en fonction du temps. Ceci est fait par une horloge temps réel. L'horloge temps réel utilisé est le DS1307 de Dallas.

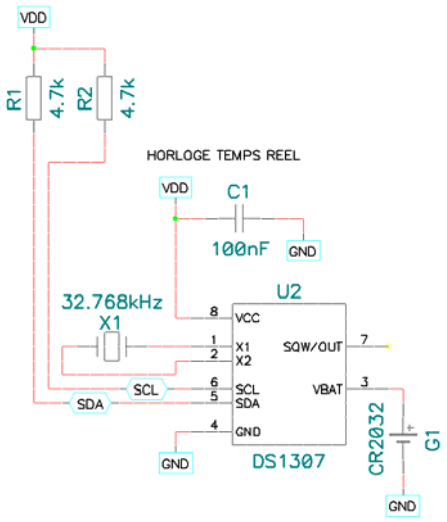


Figure 18: horloge temps réel DS1307

Cet horloge utilise le bus I2C pour communiquer avec le PIC (pins SDA et SCL).
Un quartz de 32.768 kHz fournit le signal de clock à l'horloge, et une pile 3V (G1 sur le schéma) permet le fonctionnement de l'horloge aussi en absence d'alimentation de la carte.

2.3.9 Commande stores

Pour commander les stores on utilise deux contact de relais (montée – descente), qui seront montés en parallèle avec la commande manuelle.

Chaque relais est piloté par une sortie digitale du PIC amplifiée par un driver ULN2803. Une diode roue libre est déjà présente dans le circuit intégré.

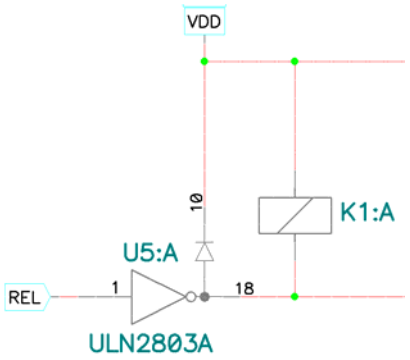


Figure 19: relais commande stores

2.3.10 Servovannes

La servovanne utilisée est le modèle ABN-DDC-Alpha de Danfoss.
Elle est alimentée en 24 VAC et l'ouverture est commandée par une tension 0-10 V DC.

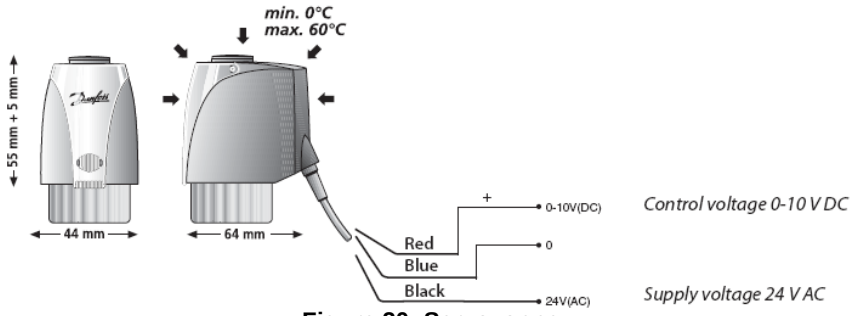


Figure 20: Servovanne

Elle a les caractéristiques suivantes :

Tension d'alimentation	24 V AC
Signal de commande	0-10 VDC
Puissance consommée	1.5 W
Puissance de réglage	90 N
Température maximale ambiante	50°C

Sa caractéristique de sortie (mm d'ouverture en fonction de la tension de commande appliquée) est linéaire, et il y a une marge d'adaptation de 0.5 V :

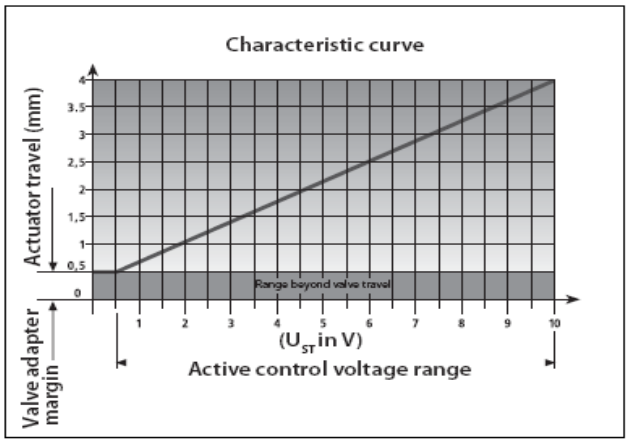


Figure 21: Caractéristique servovanne

2.3.11 Interface homme-machine

Une interface de dialogue avec l'utilisateur a été prévue, afin qu'il soit possible de modifier des données tels que consigne et horaires de coupure du chauffage.

L'interface est composée d'un écran LCD à 2 lignes avec 16 caractères par ligne, et de 4 boutons poussoir.

Voici la disposition des éléments sur la carte régulateur :

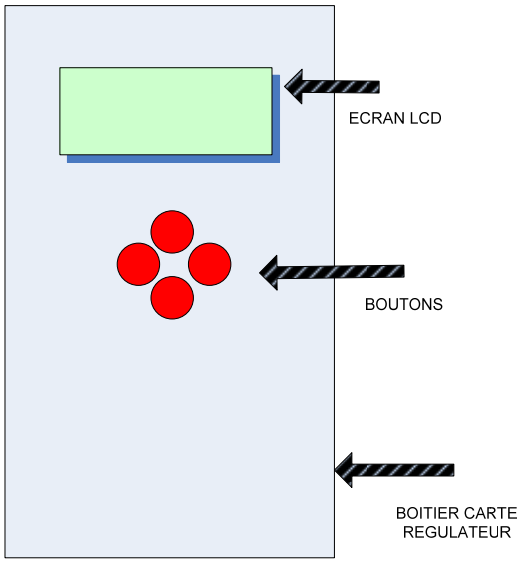


Figure 22: Interface avec l'utilisateur

L'écran LCD est le modèle EA W162N3LED de Electronic Assembly, et il a une consommation de 4 mA au maximum.

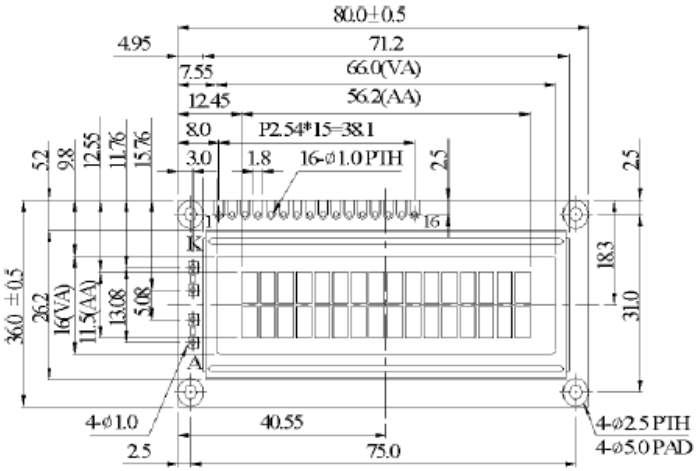


Figure 23: Ecran LCD 2 lignes

2.4 Description des cartes développées

2.4.1 Cartes émetteurs

Les 2 émetteurs sont presque identiques : la seule chose qui change est le fait d'avoir un capteur de présence à la place du capteur de lumière.
Un seul schéma a pourtant été crée en laissant la possibilité de choisir, à l'aide d'un jumper, quels capteurs utiliser.

La carte se compose de 4 parties principales :

- Alimentation :

La pile 3.6 V alimente le circuit et l'élévateur switching MAX756 élève la tension à 5V.
Une LED s'allume lorsque la pile est déchargée.

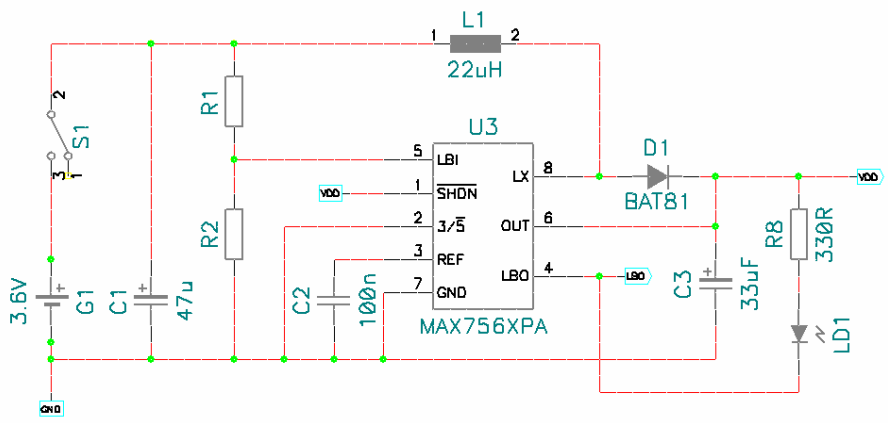


Figure 24: Alimentation carte émetteur

- Capteurs :

Deux capteurs par carte envoient leurs informations au PIC. Sur la gauche de la figure suivante, le capteur de température ; sur la droite, le capteur de présence, dont la sortie commande un transistor, et le capteur de rayonnement.

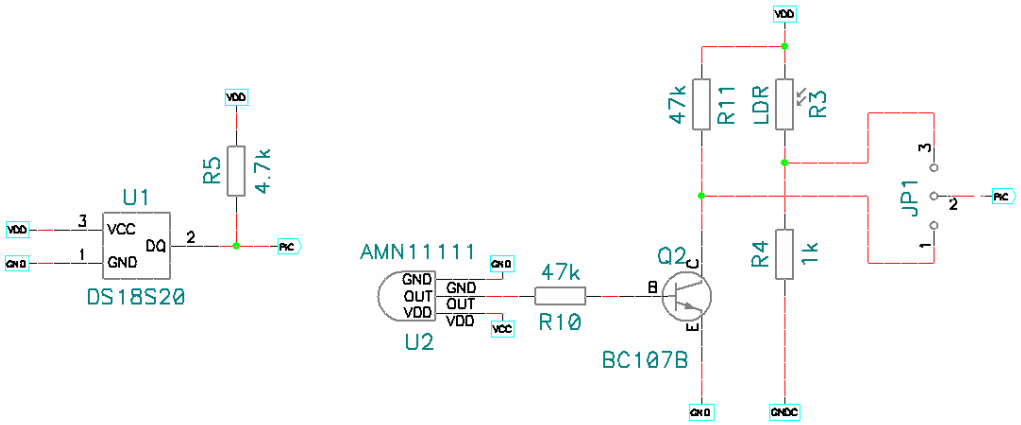


Figure 25: Capteurs sur cartes émetteur

- Elaboration des données :

Le PIC18F1220 transforme les données reçues par les capteurs et les envoie au module RF.

- Envoi des données :

Le module ER900TRS envoie les données reçues par le PIC à la carte régulateur

Voici quelques images des cartes avec les boitiers de protection:

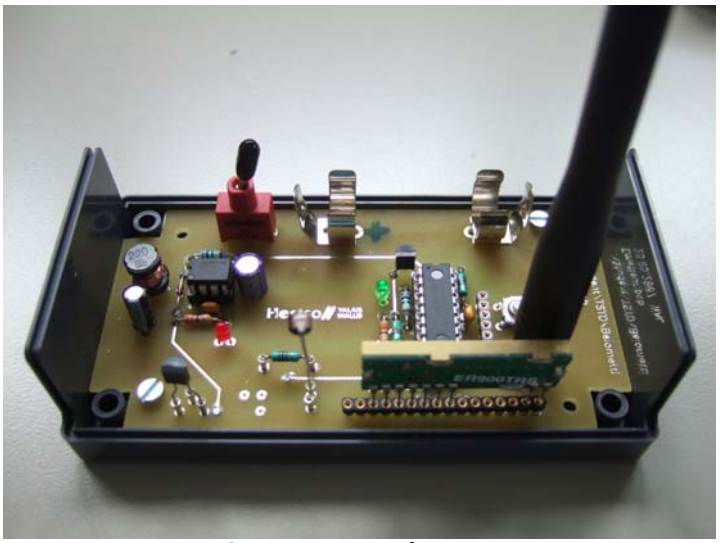


Figure 26: Carte émetteur

Figure 27: Cartes émetteur externe et interne

Les schémas électriques complets se trouvent en annexe 1.

2.4.2 Carte régulateur

Cette carte est composée de 4 parties principales :

- Alimentation :

Un transformateur 230/12V suivi de 2 régulateurs de tension fournissent les tensions d'alimentation de 5 et 12 V.

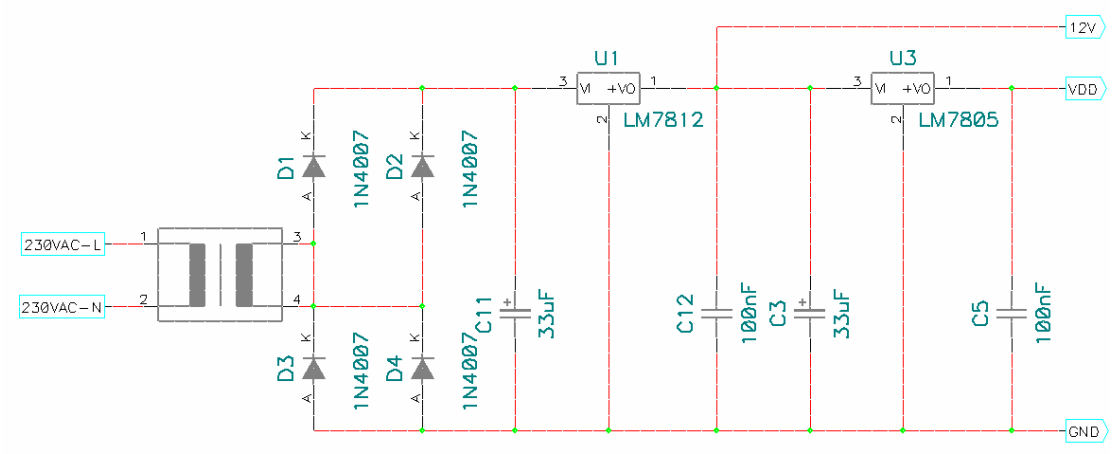


Figure 28: Alimentation carte régulateur

- Régulateur :

Le PIC 18F4525 contient l'algorithme de régulation du chauffage ainsi que la gestion des stores et de l'interface homme machine

- Réception des données :

Le module radio ER900TRS reçoit les données provenant des cartes capteurs et les transmet au PIC

- Commande des actuateurs :

La commande des servovannes s'effectue avec un signal analogique 0-10 V provenant d'une sortie PWM du PIC qui est ensuite filtrée à la bonne fréquence de coupure.
Suite au filtre, un suiveur de tension permet de découpler la charge, notamment les servovannes.

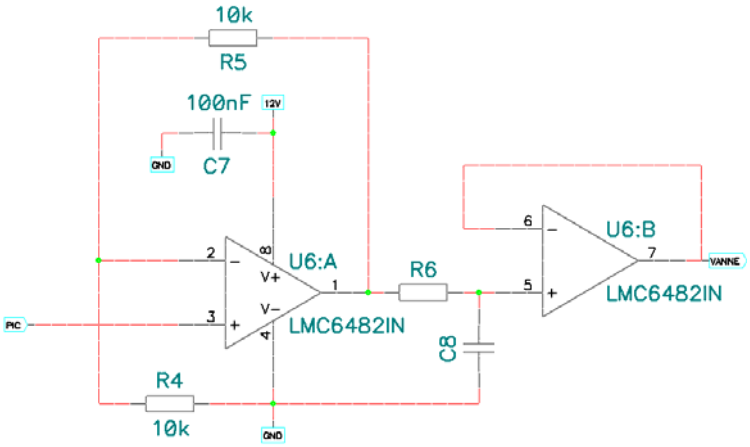


Figure 29: commande servovanne

L'amplificateur opérationnel LM 6482 transforme le niveau la sortie PWM du PIC 0-5V en 0-10V (R4 et R5 génèrent un gain de 2).
Ensuite R6 et C8 filtrent le signal PWM afin d'obtenir un signal analogique 0-10V : le dimensionnement de ces 2 élément s'effectue à partir de la fréquence de coupure :

$$f_c = \frac{1}{2 * \pi * R * C}$$

La fréquence du signal PWM généré étant 2kHz, la fréquence de coupure a été choisie une décade plus bas, à 200Hz . En choisissant un condensateur de 47nF :

$$200 = \frac{1}{2 * \pi * R6 * 47nF}$$

$$\Rightarrow R6 \cong 16.9K\Omega \Rightarrow 15K\Omega$$

Pour ce qui concerne la commande des stores, deux relais gèrent l'actionnement (montée/déscente)

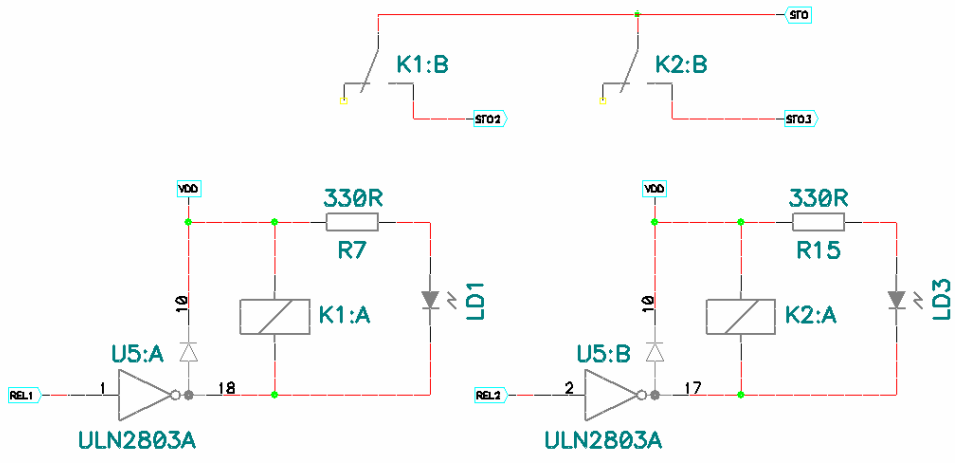


Figure 30: Relais commande stores

- Interface homme-machine :

L'écran LCD et 4 boutons présents sur cette carte, permettent à l'utilisateur de varier certaines grandeurs. Les boutons sont reliés au PIC comme entrées digitales et l'écran LCD occupe plusieurs sorties digitales.

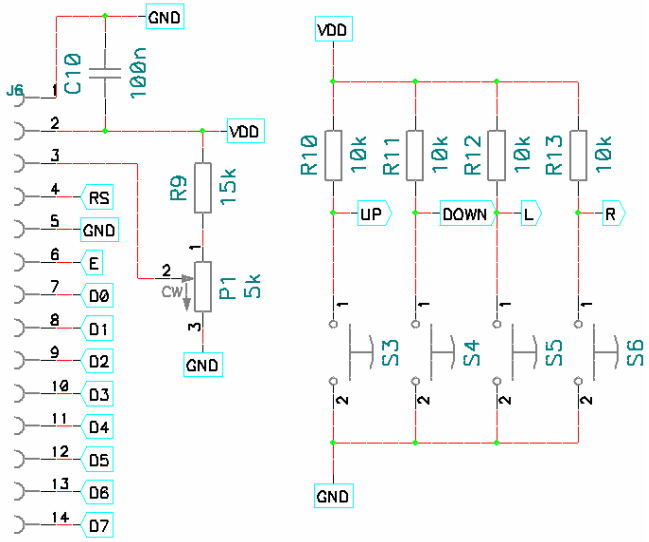


Figure 31: Interface d'interaction

Le schéma électrique se trouve en annexe 2.

Voici la carte régulateur : à remarquer que dans ces images il s'agit du premier prototype, avant la fabrication du prototype final.

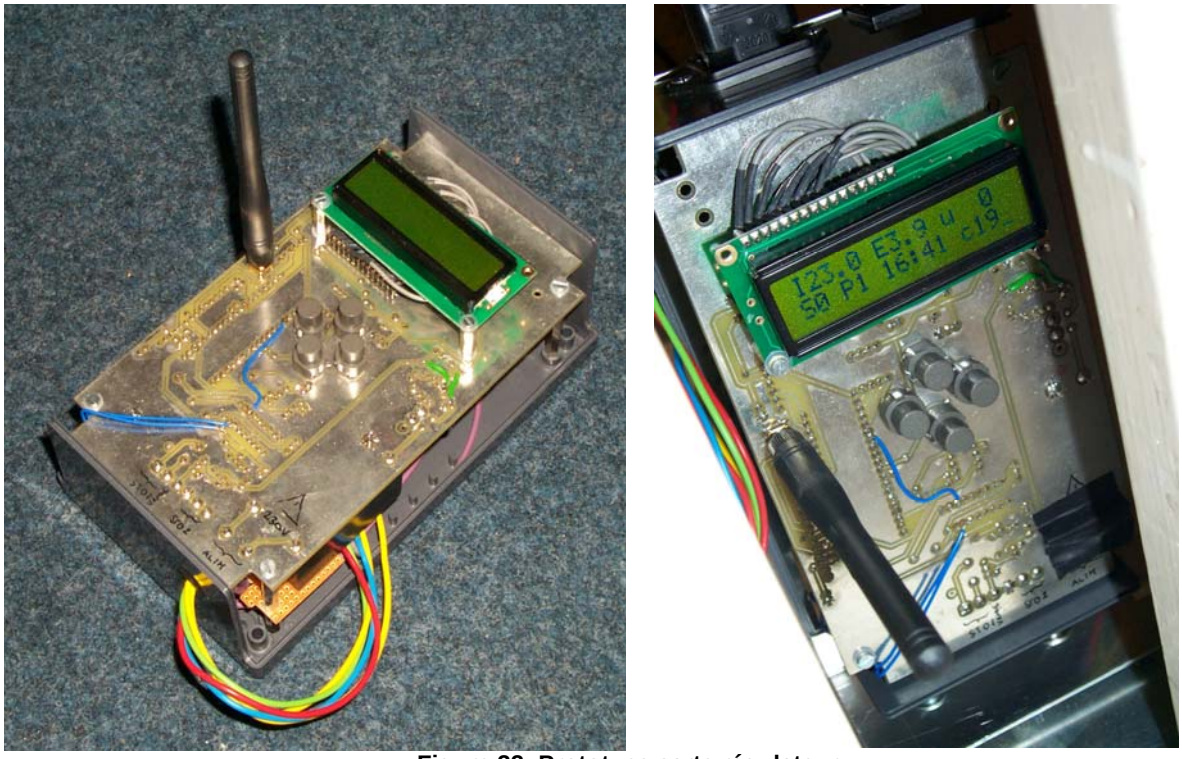


Figure 32: Prototype carte régulateur

2.5 Consommation des cartes

Surtout pour les cartes émetteurs, qui sont alimentées par pile, il est important de calculer la consommation, afin de pouvoir estimer la durée de la pile.
Pour ce qui concerne la carte régulateur, al consommation n'est pas critique, étant donné qu'elle est alimentée par le réseau 230V.

2.5.1 Carte émetteur externe

élément	*Consommation « ON »*	*Consommation « OFF »*
PIC	1.8mA	400 uA
Module RF	23 mA	0
Capteur lumière	2 mA	0
Capteur température	1.5 mA	0.75 uA
Alimentation	60 uA	-
Autres (LEDs,…)	~ 10 mA	0
Total	~ 40 mA	~ 0.5 mA

En estimant une durée d'utilisation de 5 secondes chaque 5 minutes (modalité ON), et un fonctionnement en standby (modalité OFF) pour tout le reste du temps, on peut estimer la durée de la batterie, qui a une charge de 2300mAh.

Consommation journalière à l'état ON :

$$24\frac{h}{jour} * 12\frac{utilisations}{h} * \frac{5}{3600}\frac{h}{utilisation} * 40mA = 16\frac{mAh}{jour}$$

Consommation journalière à l'état Off :

$$(24-1)\frac{h}{jour} * 0.5mA = 12.5\frac{mAh}{jour}$$

Ça donne une durée de vie estimée de la pile de :

$$\frac{2300mAh}{(16+12.5)\frac{mAh}{jour}} \cong 80\,jours$$

2.5.2 Carte émetteur interne

élément	*Consommation « ON »*	*Consommation « OFF »*
PIC	1.8mA	400 uA
Module RF	23 mA	0
Capteur présence	270 uA	100 uA
Capteur température	1.5 mA	0.75 uA
Alimentation	60 uA	-
Autres (LEDs,…)	~ 10 mA	0
Total	~ 40 mA	~ 0.5 mA

Le calcul de la durée de la batterie effectué ci-dessus est valable aussi pour cette carte, vu que la consommation des 2 cartes est presqu'identique.

2.5.3 Carte régulateur

élément	*Consommation « ON »*	*Consommation « OFF »*
PIC	5mA	400 uA
Module RF	19 mA	0
Relais stores	80 mA	N.A.
Alimentation	60 uA	0
Ecran LCD	1.8 mA	N.A.
Autres (LEDs,…)	~ 30 mA	0
Total	~ 150 mA	~ 0.5 mA

2.6 Coût des cartes

Le prix global du prototype (composants des 3 cartes, boitiers compris) est de 590 Fr.
Une bonne partie, environ 275 Fr, regarde le matériel pour la transmission radio.
La liste des prix se trouve en annexe 3.

3 Communication RF

Les modules radio utilisés, ER900TRS, comme déjà dit, sont des modules transceiver qui fonctionnent sur une fréquence de 868MHz.
Ils contiennent un microcontrôleur embarqué qui s'occupe du protocole de communication ; de plus, les niveaux d'entrée/sortie sont adaptés aux standards de l'interface série USART.

Les modules USART(Universal Synchronous-Asynchronous Receiver Transmitter) sont des composants utilisés pour transformer des données parallèles en données à échanger de façon sérielle : dans les ordinateurs ils gèrent les communication séries RS232 ; dans grand nombre de microcontrôleurs ils sont aussi intégrés.
Ils peuvent créer des communications synchrones ou asynchrones ; dans ce projet une communication asynchrone est effectuée.

Les trames envoyées/reçues par UART sont constituées de la façon suivante :

Figure 33: Trame USART

- 1 bit de start (niveau bas)
- 8 bits de données
- 1 bit de parité facultatif
- 1 bit de stop (niveu haut)

N.B. : Le bus au repos se trouve à l'état haut

L'utilisation des modules RF est donc très simple : pour la transmission il suffit de leur envoyer au biais de l'interface UART du PIC les données à envoyer, stockées d'abord dans le buffer de transmission ; pour la réception par contre il faut aller lire les messages reçus dans le buffer de réception de l'interface UART. Le module intégré s'occupe de gérer tout le reste.

Voici deux images explicatives de l'UART intégrée dans le PIC :

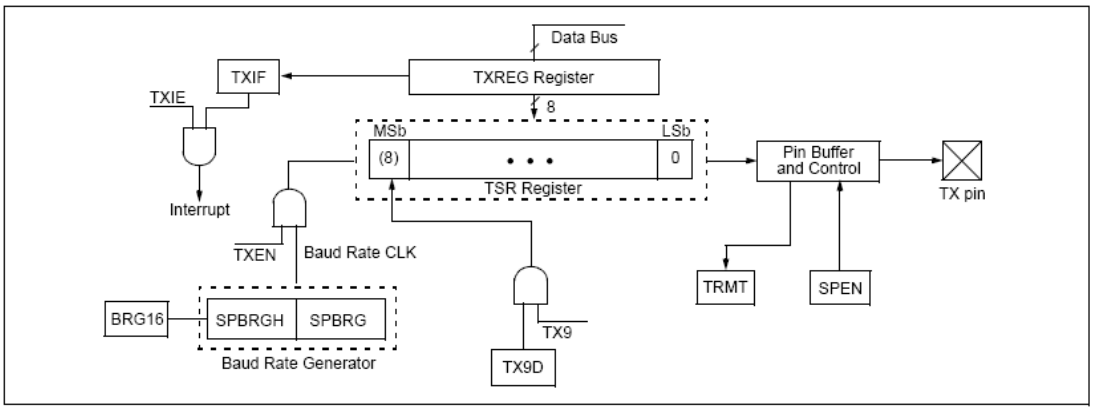


Figure 34: Transmission USART

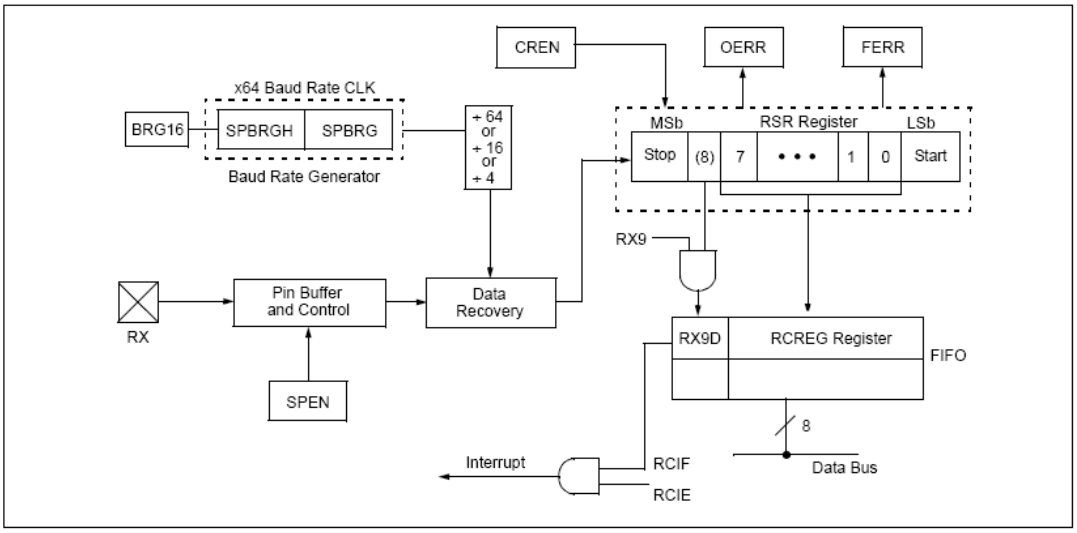


Figure 35: Réception USART

L'interface USART du PIC doit être configurée et la démarche à suivre est la suivante :

1. Configurer les pins d'entrée/sortie
2. Activer le module : bit SPEN=1
3. Configurer la transmission (registre TXSTA)
4. Configurer la réception (registre RCSTA)
5. Choisir le baudrate (registre BAUDCTL)

La partie du programme qui s'occupe de la configuration de l'interface UART est la suivante :

```
// UART INITIALISATION

SPBRG=12;                   // SELECT BAUD RATE
BRG16=0;                    // BAUDRATE ON 8 BIT
BRGH=0;
SYNC=0                      // ASYNCHRONOUS MODE

TRISB1=0;                   // SET UART PINS
TRISB4=1;

TXEN=1;                     // ENABLE TRANSMISSION
CREN=1;                     // ENABLE RECEPTION
SPEN=1;                     // ENABLE UART MODULE

GIE = 1;                    // ENABLE GLOBAL INTERRUPTS
PEIE = 1;

TXIE=0;                     // DON'T USE TRANSMISSION INTERRUPT
RCIE=1;                     // ENABLE RECEPTION INTERRUPT AND CLEAR FLAG
RCIF=0;
```

Le baudrate est la vitesse de communication, en kbits/sec.
J'ai choisi une valeur de 2400Kb/sec ; cette valeur se paramètre en utilisant le registre SPBRG (Fosc= 2 MHz):

$$BR = \frac{Fosc}{64(SPBRG+1)}$$

$$2.4*10^3 = \frac{2*10^6}{64(SPBRG+1)} \Rightarrow SPBRG = 12$$

Ceci donne une erreur de -0.16% sur la fréquence.

La réception et de la transmission de messages peuvent être gérées de deux façons :

- en utilisant les interruptions, c'est-à-dire que à chaque transmission/réception une interruption est générée afin d'effectuer l'opération demandée
- en contrôlant le flag de réception qui indique qu'un message a été reçu, pour la partie transmission, et en plaçant dans le registre de transmission les messages à envoyer, pour la partie transmission

Ci-dessous la méthode utilisée est illustrée :

- Réception : lorsqu'un message est reçu, l'interruption de réception et activée : il faut alors aller lire le registre de réception RCREG, qui contient les données reçues :

```
void interrupt uart(void)
{
   if(RCIF == 1)                        // TEST RECEPTION FLAG
   {
         receivedMessage = RCREG;       // GET CHARACTER FROM UART
         RCIF = 0;                      // RESET INTERRUPT
   }
}
```

- Transmission : sans utiliser l'interruption de transmission, on charge dans le registre TXREG les messages à envoyer, avant d'avoir attendu que le module RF et l'interface UART soient prêts. Au moment où le registre contient des données, celles-ci sont envoyées :

```
#define   RFBUSY PORTB & 0x03          // BUSY PIN DEFINITION

while(RFBUSY)                          // WAIT UNTIL RF MODULE IS
READY
{}
while(TRMT == 0)                       // WAIT UNTIL UART IS READY
{}
TXREG='a';                             // SEND AN 'A' CHARACTER
```

Voici un exemple de trame envoyée par le PIC vers le module RF :
(il s'agit du caractère 't')

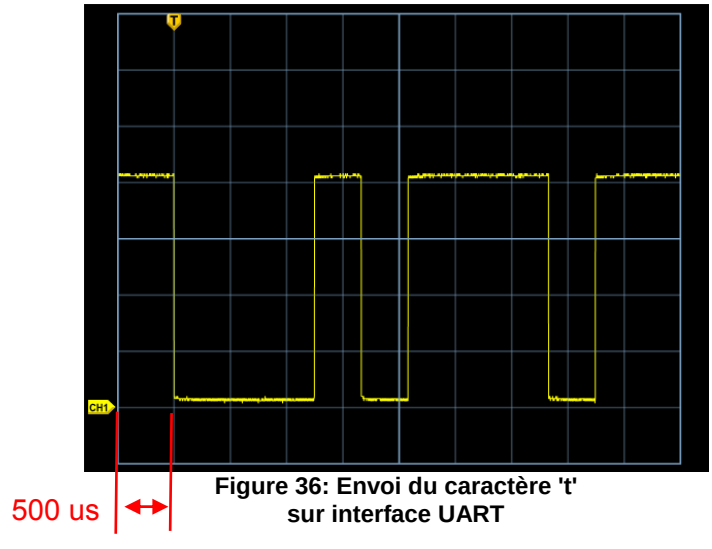


Figure 36: Envoi du caractère 't' sur interface UART

Le code ASCII du caractère ‘t’ est 116 en valeur décimale, ce qui correspond à une valeur binaire de 01110100. En analysant l’image ci- dessus, on peut constater que la valeur envoyée est 0001011101.
La trame commence avec un bit de start et se termine avec un bit de stop ; le caractère est envoyé en partant du LSB :

$$'t' = 01110100 \Rightarrow inversé... \Rightarrow startbit + 00101110 + stopbit \Rightarrow 0001011101$$

On peut aussi déterminer que la durée d’un bit vaut 420 μs ; ceci donne un baudrate effectif de

$$BR = \frac{1}{Tbit} = 2381\frac{kbit}{s}$$

Et une erreur de

$$erreur = \frac{2381 - 2400}{2400} = -0.79\%$$

Cette précision suffit à éviter la présence d’erreurs de transmission.

4 Stratégie de régulation et commande

4.1 Utilisation des grandeurs de contre-réaction

Les quatre grandeurs de contre-réaction sont utilisées bien évidemment pour les 2 réglages à effectuer.

- *Température externe* : dans un premier temps elle ne va pas être utilisée, mais elle peut être utilisée pour déterminer l'heure de démarrage du chauffage ou bien pour modifier les consignes
- *Température interne* : c'est la grandeur à régler avec le régulateur PID
- *Présence* : la présence de personnes dans la pièce a une double utilisation, d'abord elle sert à déterminer le mode de fonctionnement des stores (manuel/automatique) et en plus on l'utilise pour couper le chauffage en absence d'occupants dans la pièce
- *Rayonnement solaire* : cette grandeur permet de déterminer, suite à la détermination d'un seuil, de définir la position des stores

4.2 Régulation du chauffage

4.2.1 Choix de la consigne

La régulation est effectuée à l'aide d'un régulateur numérique PID avec horaires de coupure; sa conception sera traitée dans le chapitre suivant.

Le principe de la régulation est le suivant : en se basant sur la présence de personnes dans la pièce et sur l'horaire, la consigne de température interne est fixée à 22°C en cas de présence, à 21°C après 10 minutes d'absence de personnes dans la salle et est réduite à 16°C pendant les horaires de coupure. Voir figure suivante.
Les horaires de coupure sont : de 11h30 à 12h30 pour la pause de midi, de 17h00 à 06h00 pour la nuit.

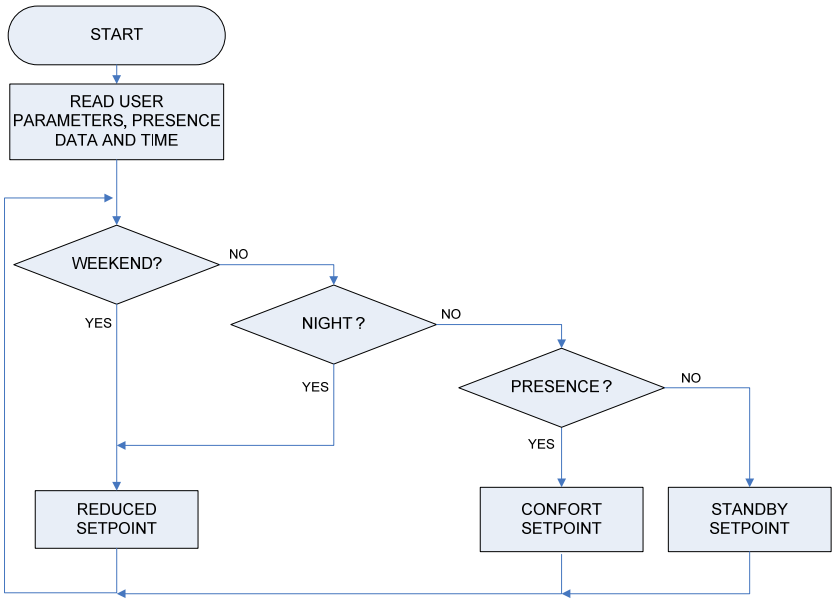


Figure 37: Schéma de principe du choix de la consigne de température

On peut améliorer ce concept en introduisant la température externe, qui pourrait déterminer par exemple l'horaire de démarrage du chauffage matinal : plus la température est faible, plus le démarrage est effectué tôt.

4.2.2 Processus de réglage

L'algorithme de réglage est traité chaque période d'échantillonnage, c'est-à-dire 5 minutes ; ce qui a été jugé suffisant, vu la lenteur du système.
Une interruption est lancée à chaque période d'échantillonnage (figure de gauche), pendant le reste du temps le contrôleur reste en attente de nouveaux données à recevoir ou bien met à jour les éventuels paramètres modifiés par l'utilisateur (figure de droite).

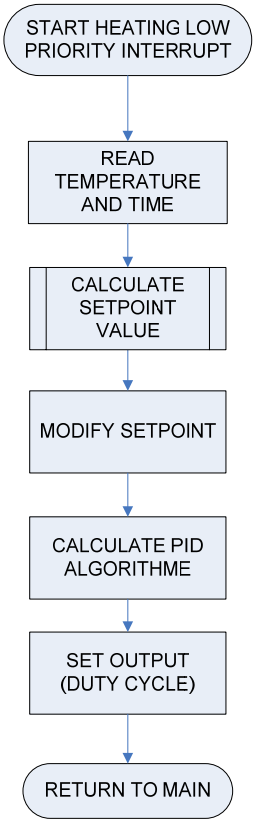


Figure 38: Processus de réglage

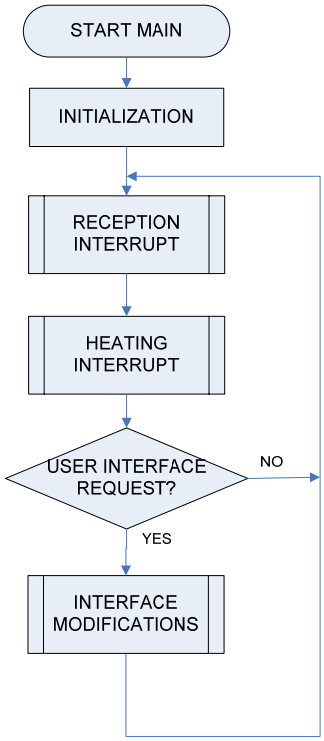


Figure 39: Programme principale

Une interruption est également déclenchée lors de la réception d'une message, qui est ensuite décodée :

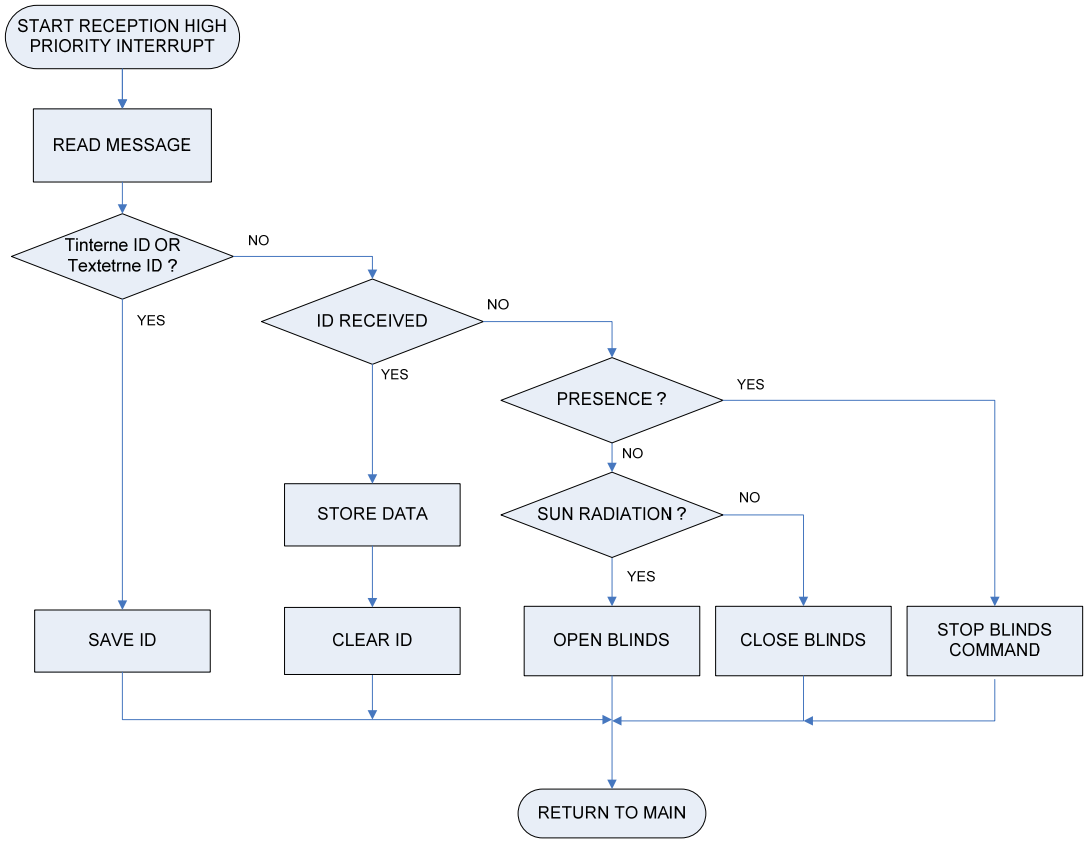


Figure 40: Interruption réception message

Les paquets de données envoyés sont structurés sur 4 caractères, par exemple :

T / 8 / 234 / S

Ces caractères ont les significations suivantes :

- Le premier caractère identifie l'émetteur (ID):

  T = émetteur externe, donc température externe
  t = émetteur interne, donc température interne

- Le deuxième caractère représente le byte de poids fort de la mesure de température (MSB)

- Le troisième caractère représente le byte de poids faible de la mesure de température (LSB)

  Le « décodage » de la température se fait avec la relation suivante :

$$Température = \frac{MSB * 256 + LSB}{100}$$

- Le quatrième byte représente les données de rayonnement solaire et présence et peut prendre les valeurs :

S	= rayonnement solaire	(émetteur externe)
s	= pas de rayonnement solaire	(émetteur externe)
P	= présence dans la pièce	(émetteur interne)
p	= pièce pas occupée	(émetteur interne)

Le paquet de l'exemple ci-dessus a donc la signification suivante :

- Température externe
- Valeur de la température : $Température = \frac{8*256+234}{100} = 22.82°C$
- Présence de rayonnement solaire

On utilise ensuite les données reçues pour effectuer les opérations décrites dans le diagramme de flux ci-dessus.

4.3 Commande des stores

La commande des stores se fait par un régulateur à deux états : stores ouvertes / stores fermées, car il n'est pas possible de connaître les positions intermédiaires.

Le concept de la gestion des stores est représenté par le diagramme de flux suivant :

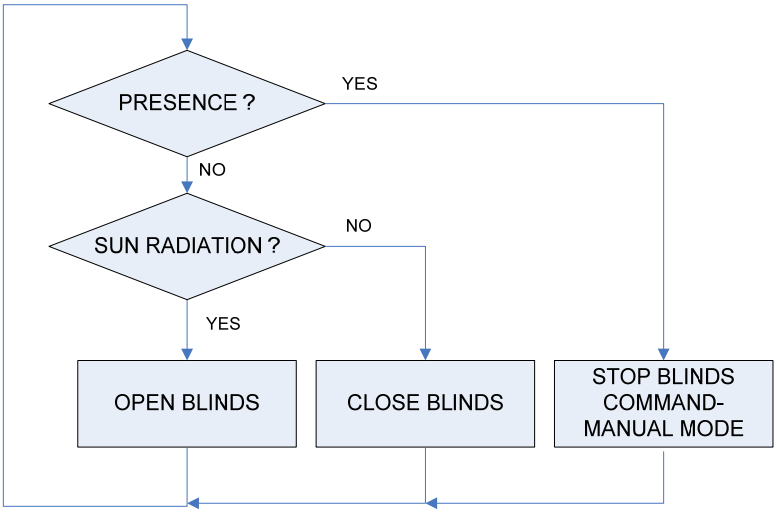


Figure 41: Schéma de principe de la commande des stores

Voici en quelques mots le principe : si des personnes se trouvent dans la pièce, la commande de stores est uniquement manuelle, donc gérée par les occupants.
Quand la pièce n'est pas occupée, le système passe en modalité automatique : si un rayonnement solaire est détecté, on monte les stores afin de profiter de cet apport passif gratuit de chaleur ; sinon on baisse les stores afin d'éviter des déperditions de chaleur par les fenêtres.

La position des stores est mémorisée afin de ne pas agir plusieurs fois dans le même sens (ouverture ou fermeture).
L'information sur le rayonnement est envoyée chaque 5 minutes, la présence est envoyée à chaque fois qu'il se vérifie un changement de valeur.

Pou plus de détail voir le chapitre « programmation du système».

5 Programmation du système

5.1 Introduction

Dans ce chapitre je vais expliquer comment se passe l'acquisition des informations des capteurs, leur gestion et comment ont été programmés les éléments présents sur les cartes, qui exploitent souvent des interfaces spéciaux des PICs.

La partie communication radio a déjà été traitée dans le chapitre précédant, ici on explique brièvement l'acquisition des données par la voie analogique et digitale, ainsi que la commande PWM, la gestion de l'horloge temps réel et la gestion de l'interface utilisateur.

L'environnement de programmation des microcontrôleurs PIC est MPLAB IDE ; il s'agit d'un logiciel de Microchip

Les PICs sont programmés à l'aide du débuggeur/programmeur ICD2, qui supporte quasiment tous les types de PIC.

Dans MPLAB il est possible de choisir entre plusieurs compilateurs ; celui que j'ai utilisé est le HITECH PICC18. Selon le compilateur utilisé, la syntaxe du langage de programmation diffère légèrement.

Le langage de programmation utilisé est le C ; les programmes se trouvent en annexe 4 et 5.

5.2 Acquisition des données et utilisation des capteurs

5.2.1 Conversion A/D et seuil du capteur

Pour le rayonnement solaire, on nécessite du module de conversion A/D du PIC, afin de convertir le signal fournit par la photorésistance.

Pour effectuer une conversion analogique/digitale, il faut suivre la démarche suivante :

1. Configuration du module A/D
   - Sélection pin analogiques (registre ADCON1)
   - Sélection du canal et allumage du module (registre ADCON0)
   - Sélection temps d'acquisition et horloge de conversion (registre ADCON2)
2. Configuration des interruptions, si désiré
   - Mettre à 1 les pins ADIE et GIE, mettre à 0 ADIF
3. Démarrage de la conversion
   - Mettre à 1 le pin GO/DONE
4. Détection de la fin de la conversion
   - Attendre que GO/DONE =0 ou
   - Attendre l'interruption, ADIF=1
5. Lecture des valeurs
   - Lire les registres ADRESL et ADRESH, qui donnent la valeur digitale de la conversion

Voici la partie de programme qui configure le module de conversion A/D :

```
//ADCONV INIT

VCFG1=0;              // VOLTAGE REFERENCES SELECTION: VDD/VSS
VCFG0=0;

CHS2=0;               // CHANNEL SELECTION: CHANNEL 0
CHS1=0;
CHS0=0;

PCFG6=1;              // PINS SETUP: ONLY CHANNEL 1 AS ANALOG
PCFG5=1;
PCFG4=1;
PCFG3=1;
PCFG2=1;
PCFG1=1;
PCFG0=0;

ADRESL=0;             // RESET AD CONVERSION VALUES
ADRESH=0;
ADFM=1;               // RESULTS RIGHT JUSTIFIED

ACQT2=1;              // ACQUISITION TIME (20 TAD)
ACQT1=1;
ACQT0=1;

ADCS2=1;              // CLOCK SOURCE SELECT (Fosc/64)
ADCS1=1;
ADCS0=0;
ADON=1;               // TURN ON AD MODULE
```

Dans mon programme je n'utilise pas les interruptions du module A/D, mais je lance la conversion et j'attends le résultat en contrôlant le bit GO/DONE.
Le calcul du temps d'acquisition minimum se fait de la façon suivante:

$$Tacq = Amplifier\_setting\_time + Holding\_capacitor\_setting\_time + Temperature\_coeff$$

$$Tacq = Tamp + Tc + Tcoeff$$

$$Tacq > 5\mu s + 9.6\mu s + \alpha * \Delta T = 12.9\mu s$$

N.B. : $\alpha = 0.02\,\mu s/°C$

N'ayant pas besoin d'une grande rapidité, le temps de conversion maximale a été choisi ; même principe pour le temps d'acquisition.

$$Tad = 64Tosc$$

$$Tacq = 20Tad$$

Il ne faut pas oublier qu'entre une conversion et l'autre il est impératif d'attendre au moins 2 Tad.

La valeur de la conversion est ensuite récupérée afin d'être envoyée au régulateur:

```
void adconv()
{
    GODONE=1;                          // START CONVERSION
    while(GODONE==1){}                 // WAIT FOR END OF CONVERSION
    mesureL=ADRESL;                    // READ AD VALUE REGISTERS
    mesure_H=ADRESH;
    mesureH=mesure_H<<8;
    mesure=(mesureH + mesureL);   // LIGHT MEASURE BETWEEN 0 AND 65535
```

}

Pour la gestion des stores, il a fallu déterminer un seuil qui permette de détecter la présence du soleil.
L'émetteur a donc été placé dans son support (voir chapitre « tests »), dans l'orientation qu'il aura pendant les tests du système, et les valeurs de rayonnement solaires ont été détectées dans les moments clé : le coucher et la levée du soleil. La mesure a été commencée à 14h de l'après midi, interrompue vers 18h et reprise à 8h.
Le résultat du test est le suivant :

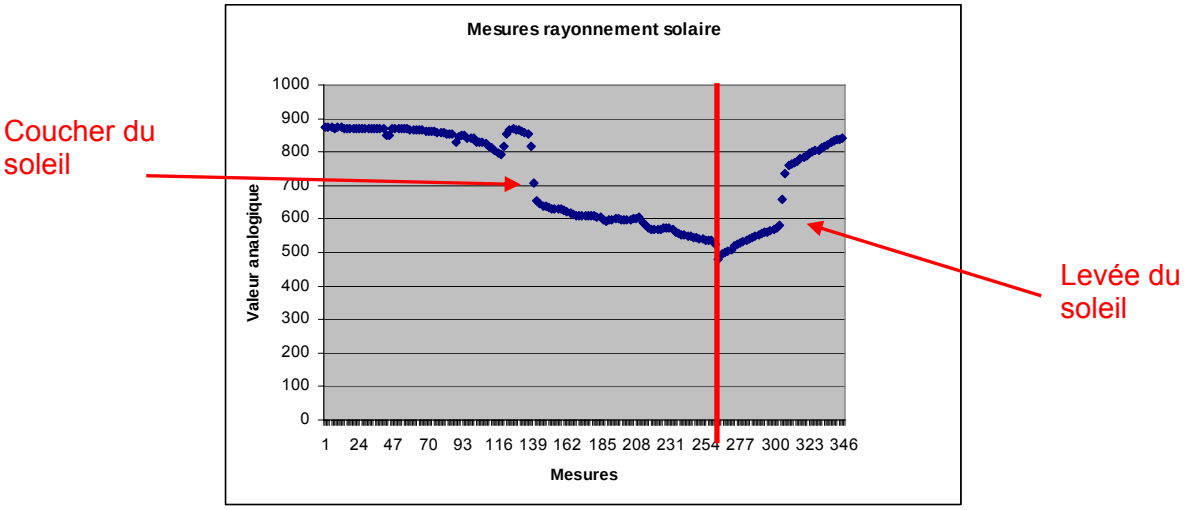


Figure 42: Détermination seuil capteur de rayonnement – Une mesure chaque minute

On peut constater que l'arrivée du soleil ainsi que son coucher sont bien visibles.
La ligne rouge indique où les mesures ont été interrompues la nuit et reprises le matin.
Le même test a été effectué pendant une journée nuageuse, à partir de 8h :

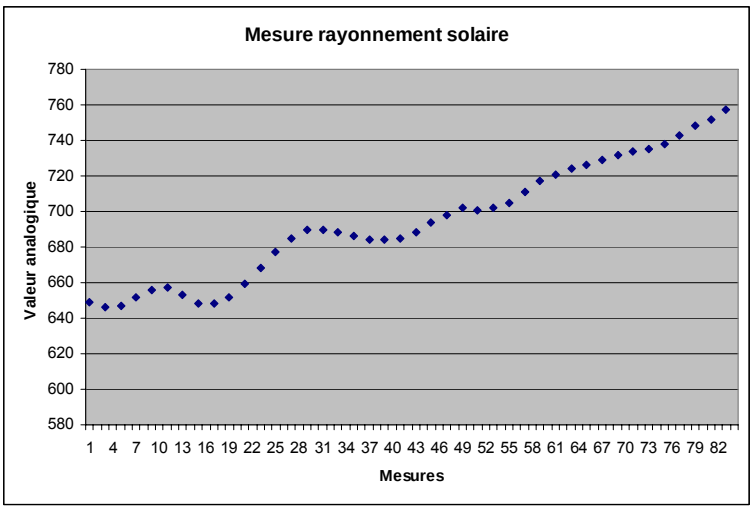


Figure 43: Rayonnement solaire pendant une journée nuageuse – Une mesure chaque minute

La lumière est évidemment plus faible sans soleil. Vers la fin de la période d'acquisition le soleil commencer à percer les nuages (zone mesure 70) : c'est pour cette raison que la valeur de la mesure augmente…

On va donc placer le seuil de présence du soleil à une valeur de la conversion analogique de :

$$SEUILSOLEIL = 750$$

5.2.2 Présence dans la pièce

Pour la commande des stores on considère aussi la présence de personnes dans la pièce : cette information est donnée par le capteur de présence AMN11111, qui fournit un signal digital.

A chaque fois que la présence d'un corps est détectée, un message est envoyé ; jusqu'au prochain cycle d'échantillonnage ce message est gardé.
Seulement si, une fois le temps écoulé, il n'y a pas de présence, un signal est envoyé au régulateur, et permet de passer en modalité de commande automatique.

```
// PRESENCE DETECTION (IN MAIN PROGRAM)

while(1)
{
    presence=PRES_SENSOR;

    if(presence!=presence_old && presence==0)
    {
          if(prescounter==0)            // FIRST EDGE DETECTED
          {
                GIE=0;                  // DISABLE INTERRUPT
                SHUTDOWN=1;             // TURN ON RADIO MODULE
                RFREADY=0;              // READY TO SEND

                createFrame('P');       // SEND PRESENCE MESSAGE

                SHUTDOWN=0;             // TURN OFF RADIO MODULE
                GIE=1;                  // ENABLE INTERRUPTS
          }
          prescounter=1;


    if(send==1)                         // SAMPLING TIME (5 MINUTES)
    {
          if(prescounter==0)
                createFrame('p');       // SEND NO PRESENCE MESSAGE
          prescounter=0;
    }

    presence_old=presence;              // STORE OLD SENSOR VALUE
```

5.2.3 Capteur de température - Bus 1-Wire

1-Wire (aussi connu sous le nom de bus Dallas ou OneWire) est un bus conçu par Dallas Semiconductor qui permet de connecter (en série, parallèle ou en étoile) des composants avec seulement deux fils (un fil de données et un fil de terre).
De nature similaire à I²C, il présente cependant des vitesses de transmission et un coût inférieurs. Il est généralement utilisé pour des thermomètres ou autres instruments de mesure météorologiques.

Le capteur de température DS18S20 utilise le protocole 1-Wire : il faut respecter ses procédures pour pouvoir accéder au capteur et effectuer la mesure.

Voici en quelques mots son fonctionnement : après une séquence d'initialisation du capteur, il faut écrire une commande ROM au capteur, suivie d'une commande d'opération (par ex. conversion température).
Ensuite il faut aller lire les données fournies par le capteur.
En schématisant le processus :

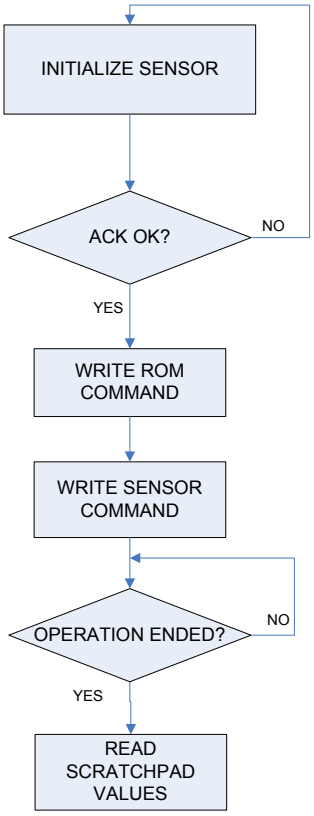


Figure 44: Procédure d'utilisation du DS18S20

Pour de plus amples informations et des détails, voir la datasheet du capteur.

5.2.4 Commande servovanne : module PWM

La commande de la servovanne, qui se fait en 0-10 VDC, est obtenue à partir d'un signal PWM dont le duty cycle détermine la valeur analogique.
Pour générer ce signal il faut utiliser le module CCP (Compare, Capture, PWM) du PIC : il s'agit d'un périphérique qui peut fonctionner en 3 modalités différentes.

En modalité PWM, la valeur du registre CCPR1L et comparée avec la valeur du timer 2 : chaque fois que les deux valeurs sont égales, le signal de sortie est mis à 1 et ensuite remis à 0 à la fin de la période.

Pour l'utilisation en modalité PWM, la configuration se fait de la manière suivante :

1. Sélection modalité PWM (registre CCP1CON)
2. Activation CCP pin : TRISC2
3. Définition de la période PWM (registre PR2)
4. Définition du duty cycle : registres CCPR1L et CCP1CON<5:4> (sur10 bits)
5. Allumage du timer2 et choix du prescaler (registre T2CON)

Voici le code correspondant à l'initialisation:

```
//PWM MODULE INIT

CCP1CON=12;                    // ENABLE PWM MODE
CCPR1L=0;                      // DUTY CYCLE INIT TO 0%
TRISC2=0;                      // PWM PORT AS OUTPUT
PR2=24;                        // PWM FREQUENCY DEFINITION (2 kHz)
T2CON=4;                       // TIMER2 ON AND PRESCALER = 1
```

Les paramètres ont été calculés selon la datasheet du PIC :

$$Tpwm = (PR2+1)*4*Tosc*TMR2\,prescaler$$
$$\Rightarrow 0.5ms = (PR2+1)*4*500ns*1 \Rightarrow PR2 = 24$$

$$Duty = (CCPR1L:CCP1CON<5:4>)*Tosc*TMR2\,prescaler$$

Dans le programme, le duty cycle est défini par le régulateur PID.
Le signal de commande (u) est converti en signal PWM en variant le duty cycle ; ceci est effectué lors de l'exécution de l'algorithme du régulateur PID :

```
duty=(int)u;                   // PWM DUTY CYCLE (0-100%)
CCPR1L= duty>>2;               // 8MSB OF DUTY CYCLE VALUE
CCP1CON &= 0b11001111;
CCP1CON|=((duty & 0x03))<<4;  // 2 LSB OF DUTY CYCLE VALUE
```

Voici le signal sorti sur le port TRISC2 avec une période de 2 kHz et un duty cycle de 40% :

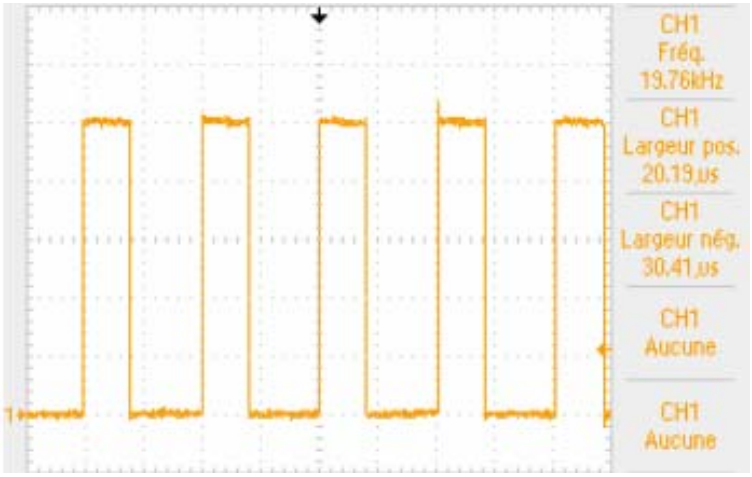


Figure 45: Signal PWM à la sortie du PIC – 25 us par division

Les mesures indiquent une fréquence de 19.76 kHz, une durée de Ton de 20.19 us et un Toff de 30.41 us.

$$Ton\_théorique = \frac{1}{Fpwm} * duty = 20\mu s$$
$$Toff\_théorique = \frac{1}{Fpwm} * duty = 30\mu s$$

5.2.5 Horloge temps réel - Bus I2C

L'horloge temps réel DS1307 possède une interface de communication sérielle I2C.

I²C (pour Inter Integrated Circuit Bus) est le nom du bus historique, développé par Philips pour les applications de domotique et d'électronique domestique au début des années 1980, notamment pour permettre de relier facilement à un microprocesseur les différents circuits d'une télévision moderne.

Le but était donc d'économiser des I/O en mettant plusieurs composants sur un même bus. Ce bus est une interface de communication sérielle sur deux fils : SDA (Serial Data) et SCL(Serial Clock).

Je ne vais pas rentrer dans les détails de fonctionnement de ce bus, mais je me limite au fait que les données sont échangées de façon sérielle sur le pin SDA au rythme de l'horloge sur la pin SCL.

L'utilisation de l'horloge temps réel DS1307 nécessite donc l'utilisation de cette interface de communication.

Pour aller écrire des valeurs dans ce composant, il a fallu créer des routines spécifiques en utilisant le module SPI (Serial Peripherical Interface) du PIC.
Pour cela je me suis basé sur la datasheet du PIC.
Voici les routines à utiliser :

```
void i2cInit(void)
{
    TRISC3 = 1 ;                    // SDA AND SCL PINS AS INPUT
    TRISC4 = 1 ;

    SSPCON1 = 0x38 ;                // PIC AS MASTER
    SSPCON2 = 0x00 ;

    // TRANSMISSION SPEED

    SSPADD = 4 ;                    // Fi2c = Fosc/(4*(SSPADD+1)) = 100 kHz
    CKE = 0 ;                       // INPUT LEVELS FOR I2C
    SMP = 1 ;                       // UNABLE SLEW RATE
    PSPIF = 0 ;                     // CLEARSSPIF INTERRUPT FLAG
    BCLIF = 0 ;                     // CLEAR BUS COLLISION FLAG

    DelayMs(10) ;
}

void i2cWaitForIdle(void)
{
    while ((SSPCON2 & 0x1F) | RW) ;     // WAIT FOR IDLE AND NOT WRITING
}
void i2cStart(void)
{
    i2cWaitForIdle() ;
    SEN =1 ;                            // SEND START
    while (SEN) ;
}
void i2cRepStart(void)
{
    i2cWaitForIdle() ;
    RSEN = 1 ;                          // SEND REPEATED START
```

```
    while (RSEN) ;
}
void i2cStop(void)
{
    i2cWaitForIdle() ;
    PEN = 1;                            // SEND STOP
    while (PEN) ;
}
unsigned char i2cRead(unsigned char ack)     // READ ACTUAL TIME DATA
{
    unsigned char data ;
    i2cWaitForIdle() ;
    RCEN = 1 ;
    i2cWaitForIdle() ;
    data = SSPBUF ;
    i2cWaitForIdle() ;
    if (ack)    ACKDT = 0 ;             // ACK
    else        ACKDT = 1 ;             // NO ACK
    ACKEN = 1 ;                         // SEND ACKNOWLEDGE SEQUENCE
    return(data) ;
}
unsigned char i2cWrite(unsigned char data)   // WRITE TIME DATA
{
    i2cWaitForIdle() ;
    SSPBUF = data ;
    return(!ACKSTAT) ;                  // RETURN '1' IF TRANSMISSION OK
}
```

Dans la fonction d'initialisation j'ai défini la fréquence utilisée à 100 kHz:

$$Fi2c = \mathrm{Fosc}/(4*(\mathrm{SSPADD}+1))$$

5.3 Interface homme-machine

L'interface homme machine, c'est à dire l'écran LCD et les 4 boutons de direction, sont utilisés pour permettre à l'utilisateur de rentrer les valeurs de consigne du chauffage, les horaires de coupure ou de consigne réduite.

Voici de suite un schéma qui illustre les différentes options :

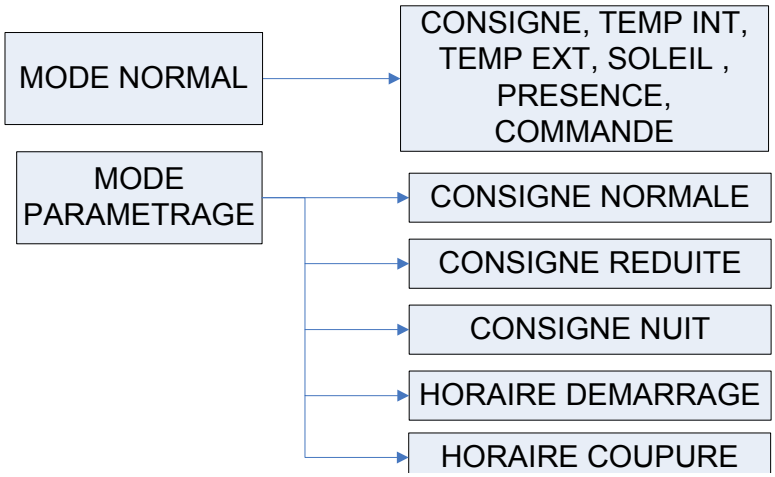
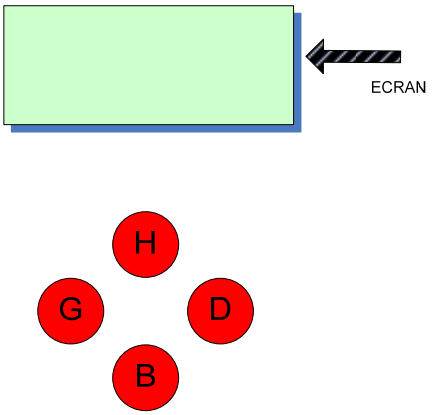


Figure 46: Schéma menu interface homme-machine

La disposition des boutons est la suivante :



Ils ont été dénommés : H=Haut B=Bas D=Droite G=Gauche

L'utilisation de l'interface suit un principe assez simple : chaque menu permet de modifier une valeur ; pour entrer dans la modalité « mode paramétrage » et les parcourir, il faut appuyer le bouton D. Les boutons H et B permettent ensuite de augmenter, respectivement diminuer la grandeur affichée à l'écran.
Pour sortir du menu et passer à l'affichage standard, « mode normal », on utilisé le bouton G.

Pour la gestion de l'écran LCD, j'ai utilisé le logiciel Application Maestro de Microchip, qui génère automatiquement le code nécessaire, après lui avoir communiqué les paramètres nécessaires.
Il génère les fichiers XLCD.c et XLCD.h, qui contiennent plusieurs fonctions utiles pour le fonctionnement de l'écran LCD. (Voir Annexe X.)
Les principales fonctions utilisées sont les suivantes :

Fonction	But	Paramètres à passer
XLCDInit()	Initialisation de l'écran	-
XLCDClear()	Réinitialisation de l'écran	-
XLCDL1home()	Place le curseur au début de la première ligne	-
XLCDL2home()	Place le curseur au début de la deuxième ligne	-
XLCDPut()	Affiche un caractère	Caractère à visualiser
XLCDPutRomString()	Affiche une chaine de caractères	Chaine à visualiser

Pour ce qui concerne les 4 boutons, une détection de flanc montant est effectuée dans la boucle principale :

```
while(1)                                        // INFINITY LOOP
{
      up_actual= UP_BUTTON;                     // UPDATE BUTTONS STATUS
      down_actual= DOWN_BUTTON;
      left_actual=LEFT_BUTTON;
      right_actual=RIGHT_BUTTON;

      if(up_actual!=up_old && up_actual==0)     // POSITIVE EDGES DETECTION
        {…}
      if(down_actual!=down_old && down_actual==0)
        {…}
      if(left_actual!=left_old && left_actual==0)
        {…}
      if(right_actual!=right_old && right_actual==0)
        {…}

      {…}

      up_old=up_actual;                         // STORE OLD BUTTON VALUES
      down_old=down_actual;
      left_old=left_actual;
      right_old=right_actual;
}
```

6 Conception du régulateur et de la commande

6.1.1 Description du régulateur

Le régulateur qui a été implanté est un PID traditionnel, qui est largement utilisé dans le domaine de la régulation.
Ce régulateur est composé de trois termes : Proportionnel, Intégrateur et Dérivateur.
Le terme P est en réalité un gain ; il permet de s'approcher de la consigne mais une erreur statique est toujours présente.
Le terme I intègre l'erreur (consigne – mesure) et élimine l'erreur statique ; il augmente aussi la vitesse du régulateur, mais crée des dépassements si sa valeur et mal-ajustée.
Le terme D dérive l'erreur : il a un effet prédictif et son désavantage est le ralentissement du régulateur si sa valeur et mal-ajustée.

Un grand défaut de ce régulateur c'est qu'il est fonctionnel de manière optimale pour des systèmes linéaires ; ce qui n'est pas le cas de notre système à régler.

Pour améliorer son comportement il faudrait ajuster ses paramètres en fonction des paramètres du système dynamique, et mettant en œuvre dans ce cas un réglage adaptatif.

6.2 Régulation du chauffage

6.2.1 Boucle de réglage

La boucle de réglage numérique standard a la forme suivante :

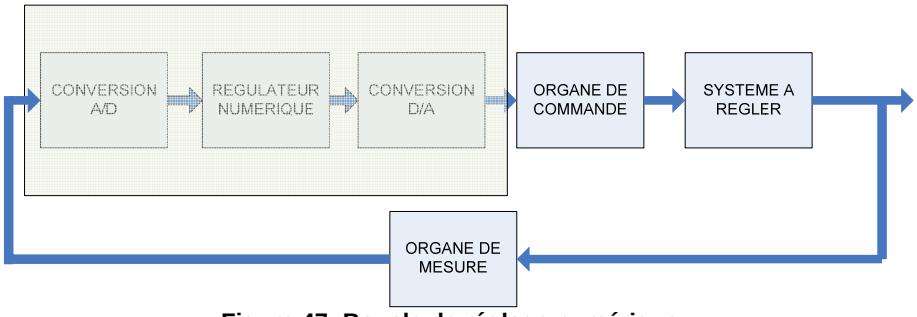


Figure 47: Boucle de réglage numérique

Dans notre cas spécifique, la boucle devient :

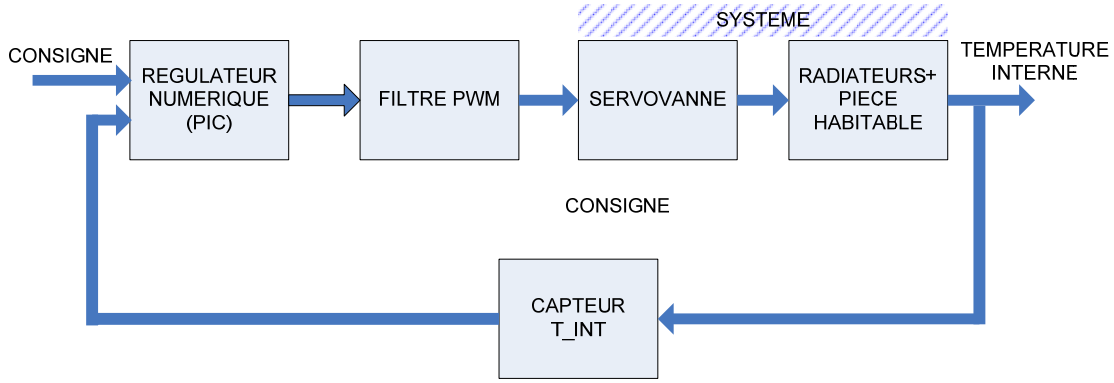


Figure 48: Boucle de réglage pour le système à considérer

Le régulateur reçoit la mesure de température interne ainsi que la consigne.
Il fournit un signal PWM dont le duty-cycle représente la valeur analogique du signal 0-10V qui attaque la servovanne. Ce signal PWM est ensuite filtré et la vanne commande l'ouverture du radiateur.

Afin de dimensionner le régulateur, le premier pas est la modélisation de la salle de tests, qui est le système qu'on veut régler.

6.2.2 Modélisation du système à régler

Afin de modéliser le système où les tests ont été effectués, plusieurs sauts indiciels en boucle ouverte ont été effectués : les vannes des radiateurs ont été ouvertes au maximum et la courbe de la température interne a été relevée.

Une modélisation du système exacte et valable pour tout le domaine de fonctionnement est assez difficile à effectuer, car en réalité il y des facteurs externes qui influencent son comportement : la position des stores, la présence ou absence de soleil, la température externe et la température de l'eau des radiateurs.

J'ai donc effectué plusieurs réponses indicielles en utilisant la plus significative pour effectuer la modélisation.

Voici l'allure de la réponse indicielle :

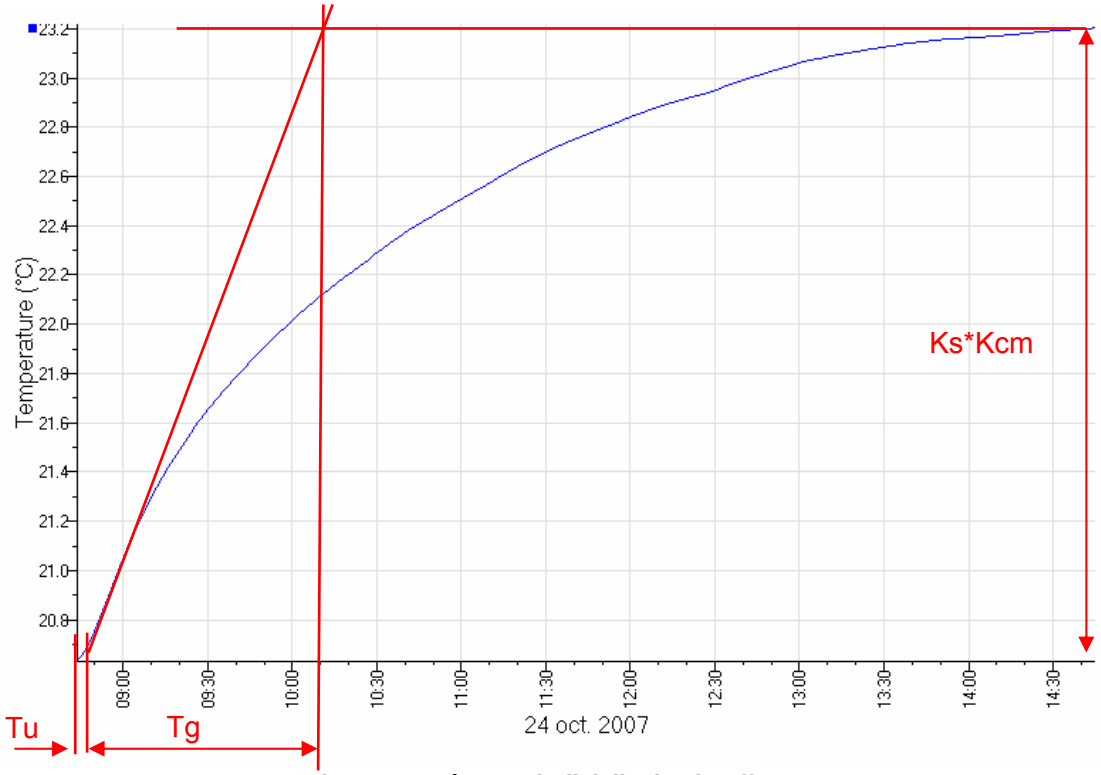


Figure 49: Réponse indicielle du chauffage

On remarque que la fonction de transfert du chauffage est de deuxième ordre (PT2), qui a la forme générale suivante :

$$F(s) = \frac{Ks * Kcm}{(1+sT_1)*(1+sT_2)}$$

A partir de la courbe il est possible de déterminer de façon expérimentale la fonction de transfert qui modélise le système (réf. Cours de MCR1, chapitre 6.5)

Je reporte ici les résultats obtenus avec la réponse indicielle la plus représentative.

On mesure Tu , Tg et le gain global $Ks * Kcm$ (servovanne+radiateurs+pièce) :

$Tu = 300s$
$Tg = 4080s$
$$Ks * Kcm = \frac{\Delta Out}{\Delta In} = \frac{(22.2 - 20.7.)°C}{10V} = 0.15$$

Le tableau suivant (cours MCR1) donne des rapport pour la modélisation de systèmes de deuxième ordre : de plus, si le rapport est inférieur à 9.65, le système a une ordre supérieur à 2. A l'aide de ce tableau, on calcule T_1 et T_2 :

$\mu = \frac{T_2}{T_1}$	$\frac{T_g}{T_1}$	$\frac{T_g}{T_u}$
0.1	1.29	20.09
0.2	1.50	13.97
0.3	1.68	11.91
0.4	1.84	10.91
0.5	2.00	10.36
0.6	2.15	10.03
0.7	2.30	9.83
0.8	2.44	9.72
0.9	2.58	9.66
0.99	2.70	9.65
1.11	2.87	9.66
1.2	2.99	9.70
2	4.00	10.36
3	5.20	11.50
4	6.35	12.73
5	7.48	13.97
6	8.59	15.22
7	9.68	16.45
8	10.77	17.67
9	11.85	18.89
10	12.92	20.09

$$\frac{Tg}{Tu} = 13.6 \Rightarrow 1^{ere}\, colonne$$
$$\frac{Tg}{T_1} = 7.48 \Rightarrow T_1 = 2720$$
$$\frac{T_2}{T_1} = 0.5 \Rightarrow T_2 = 545$$

La fonction de transfert du système est donc la suivante :

$$F(s) = \frac{0.15}{(1+2720s)*(1+545s)}$$

Avec les autres réponses indicielles, j'ai obtenu les résultats suivants :

	conditions	Soleil	Prés.	Tu	Tg	T1	T2	K
Rép. 1	Text=15°C, matin	Oui	Non	180	3360	2552	283.5	0.185
Rép. 2	Text=10°C, matin	Non	Non	180	5280	3814	381.4	0.115
Rép. 3	Text=10°C, après midi	Non	Non	300	4080	2720	545.5	0.150

Etant donné que la caractéristique de la salle varie selon des conditions extérieures, il faudrait déterminer plusieurs jeux de paramètres qui seront appliqués selon les conditions environnementales (température externe, soleil).
Cela n'a pas été effectué pendant mon projet.
La solution optimale serait d'effectuer un réglage auto-adaptatif qui modélise le système en temps réel et calcule les paramètres du régulateur.

Il est à remarquer aussi que la modélisation de la salle a été effectuée le 24 octobre, bien avant les tests, quand le climat était beaucoup plus doux ; le dimensionnement du régulateur ne sera donc pas tout à fait optimal, mais permettra quand même d'effectuer une régulation d'un niveau acceptable.

6.2.3 Dimensionnement du régulateur

Le dimensionnement du régulateur consiste à trouver une combinaison optimale des 3 paramètres P,I et D; il existe plusieurs méthodes.

6.2.3.1 Méthode pseudo-continue

Le dimensionnement du régulateur numérique a été effectué dans un premier temps selon la méthode pseudo-continue : on ne peut utiliser cette méthode que si les constantes de temps dominantes du système à régler sont au moins 2 fois plus grandes de la période d'échantillonnage.

J'ai utilisé le critère méplat, car le système n'a pas un comportement intégrateur.

La fonction de transfert du PID en pseudo-continu est la suivante :

$$G_R(s) = \frac{(1+sTn)(1+sTv)}{sTi}$$

La fonction de transfert de l'organe de commande (filtre du signal PWM) est la suivante :

$$G_{cm}(s) = \frac{1}{1+sTp} = \frac{1}{1+0.0005s}$$

$$\text{N.B : } Tp = \frac{1}{f_{PWM}}$$

La fonction de transfert du système, d'après la modélisation, est la suivante (servovanne et radiateurs) :

$$G_s(s) = \frac{Ks * Kcm}{(1+sT_1)*(1+sT_2)} = \frac{0.15}{(1+2720s)*(1+545s)}$$

D'après le critère Meplat on calcule les éléments du régulateur :

Tn est déterminé par compensation du pôle du système :

$$Tn = Tc = 2720$$

Tv compense la deuxième constante de temps du système :

$$Tv = Tc = 545$$

Ti se calcule selon la formule :

$$Ti = 2 * Kcm * Ks * Tpe = 90$$

Il est à noter que Tp devient Tpe pour les régulateurs numériques :

$$Tpe = Tp + Tr + K * h \cong 300s$$

K vaut 1 pour un PID
Tr est le temps de calcul, qui vaut quelques μs , donc négligeable
h est la période d'échantillonnage, qui a été choisie à 5 minutes, c.-à-d. 300 s

La fonction de transfert du PID en pseudo-continu est donc la suivante :

$$G_R(s) = \frac{(1+2720s)(1+545s)}{90s}$$

A l'aide des formules de conversion des gains analogiques en numérique, il est possible de déterminer les paramètres du régulateur numérique :

$$Kp = \frac{Tn + Tv - h}{Ti} = 32.9$$

$$Ki = \frac{h}{Ti} = 3.33$$

$$Kd = \frac{Tn * Tv}{h * Ti} - \frac{2(Tn + Tv) - h}{4Ti} = 33.6$$

6.2.3.2 Vérification par Matlab

Le bon dimensionnement du régulateur a été vérifié par simulation à l'aide de Matlab Simulink.
Le schéma bloc du système est le suivant :

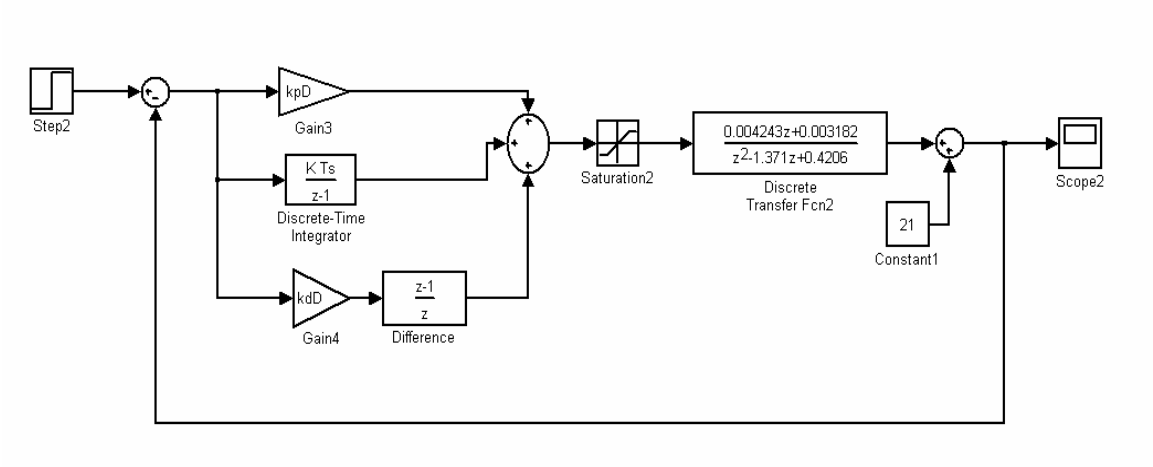


Figure 50: Schéma bloc simulation

En imposant un saut de 1°C, et en rentrant les paramètres calculés selon la méthode pseudo-continue ; en analysant le comportement du régulateur j'ai constaté qu'ils n'étaient pas optimaux, mais ils généraient une oscillation permanente.
Dans le premier graphique le saut indiciel ainsi que la réponse sont affichées ; sur le deuxième on visualise le signal de commande 0-10 V :

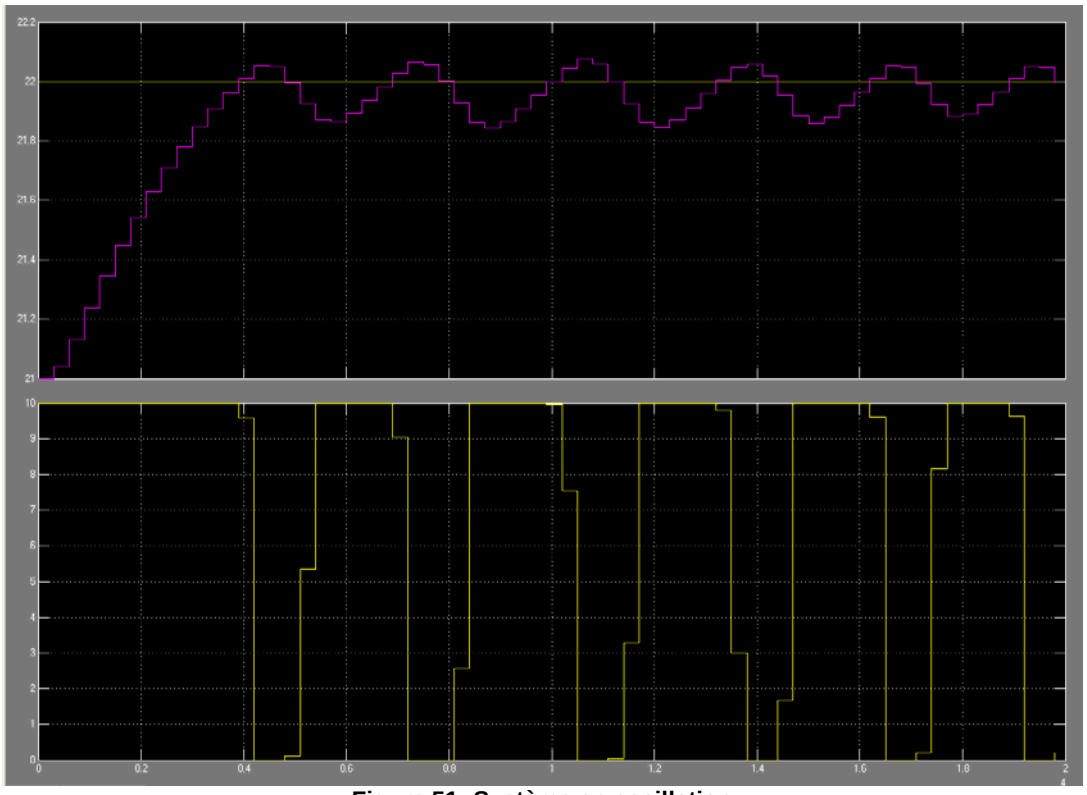


Figure 51: Système en oscillation

En modifiant les paramètres j'ai constaté que le terme intégrateur était 1000 fois trop grand par rapport à une valeur optimale.
J'ai donc décidé de dimensionner le régulateur PID en utilisant la méthode de dimensionnement dans le domaine digitale (dimensionnement en Z – cft Cours de MCR2, chapitre 7.2) :

6.2.3.3 Dimensionnement en 'Z'

La fonction de transfert en 's' du système est évidemment la même qu'avant:

$$F(s) = \frac{0.15}{(1+2720s)*(1+545s)}$$

La fonction 'c2d' de Matlab permet de passer dans le domaine numérique en calculant la transformée en Z :

$$F(z) = \frac{0.003677z + 0.002955}{z^2 - 1.474z + 0.5186}$$

La fonction de transfert numérique d'un régulateur PID à la forme suivante :

$$G(z) = Kp * \frac{(1+Td)*z^2 + \left(\frac{h}{Ti} - 1 - 2Td\right)*z + Td}{z*(z-1)}$$

En modifiant son écriture on obtient :

$$G(z) = Kp * \frac{z^2 + \frac{\left(\frac{h}{Ti} - 1 - 2Td\right)}{(1+Td)} * z + \frac{Td}{(1+Td)}}{\frac{z*(z-1)}{(1+Td)}}$$

Td et Ti se calculent en compensant les pôles du système :

$$\frac{\left(\frac{h}{Ti} - 1 - 2Td\right)}{(1+Td)} = -1.474$$

$$\frac{Td}{(1+Td)} = 0.5186$$

$$\Rightarrow Td = 1.077$$

$$\Rightarrow Ti = 3238$$

En utilisant la fonction 'rlocus' de Matlab, on trace le lieu des pôles de la fonction $F(s)*G(s)$, avec $Kp = 1$, qui devient :

$$F(z) = \frac{(0.003677z + 0.002955)*(1+Td)}{z*(z-1)}$$

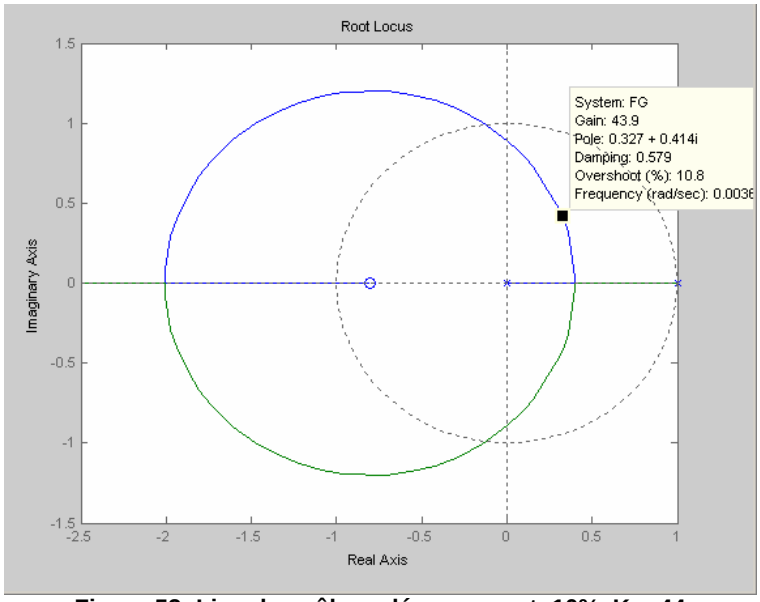


Figure 52: Lieu des pôles - dépassement=10%, Kp=44

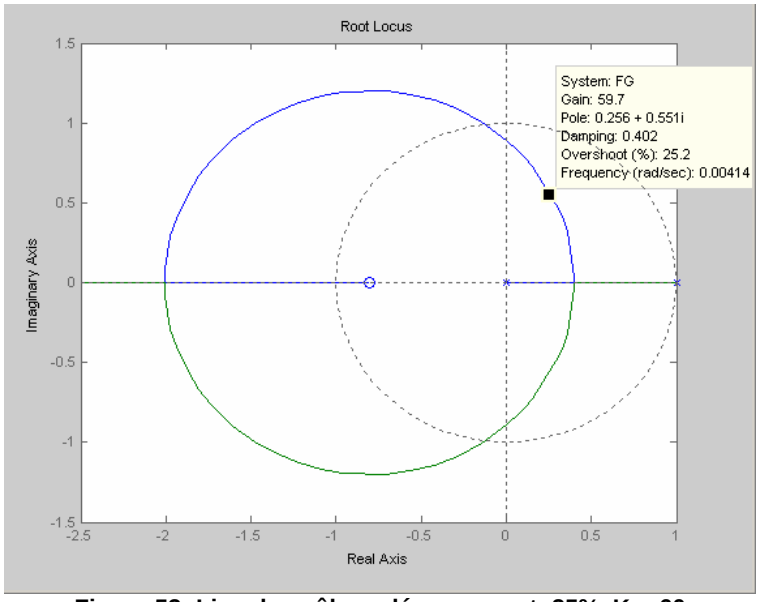


Figure 53: Lieu des pôles - dépassement=25%, Kp=60

Cela impose un gain Kp maximal de 150 afin d'avoir un système stable en boucle fermée. J'ai donc enfin dimensionné les paramètres du PID en considérant un dépassement de 25% :

$$Kp = 60$$

$$Ki = \frac{Kp}{Ti} = 0.018$$

$$Kd = Kp * Td = 65$$

La simulation du système en boucle fermée donne le résultat ci-dessous.
Dans le premier graphique le saut indiciel ainsi que la réponse sont affichées ; sur le deuxième on visualise le signal de commande 0-10 V :

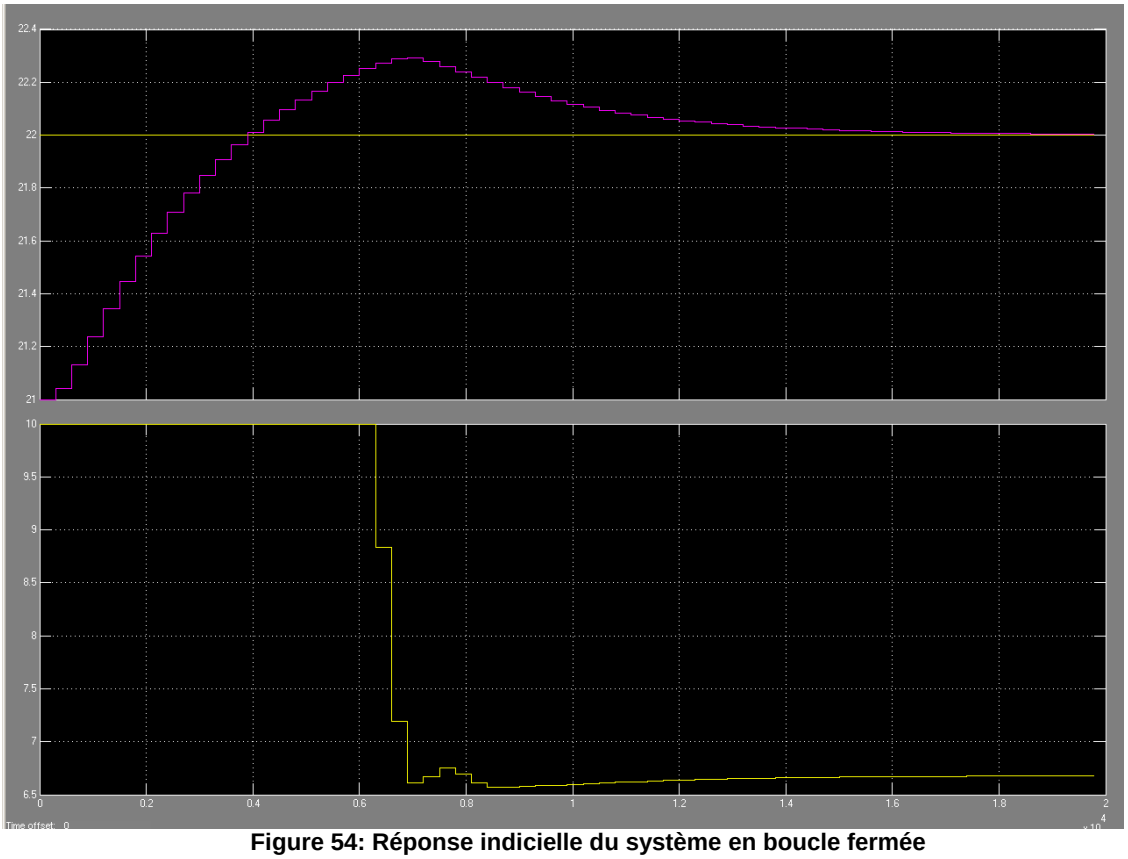


Figure 54: Réponse indicielle du système en boucle fermée

Avec les paramètres calculés par dimensionnement en Z, le régulateur a un bon comportement, mais il se produit un important dépassement de la consigne : en diminuant légèrement le terme intégrateur, on obtient un résultat satisfaisant :

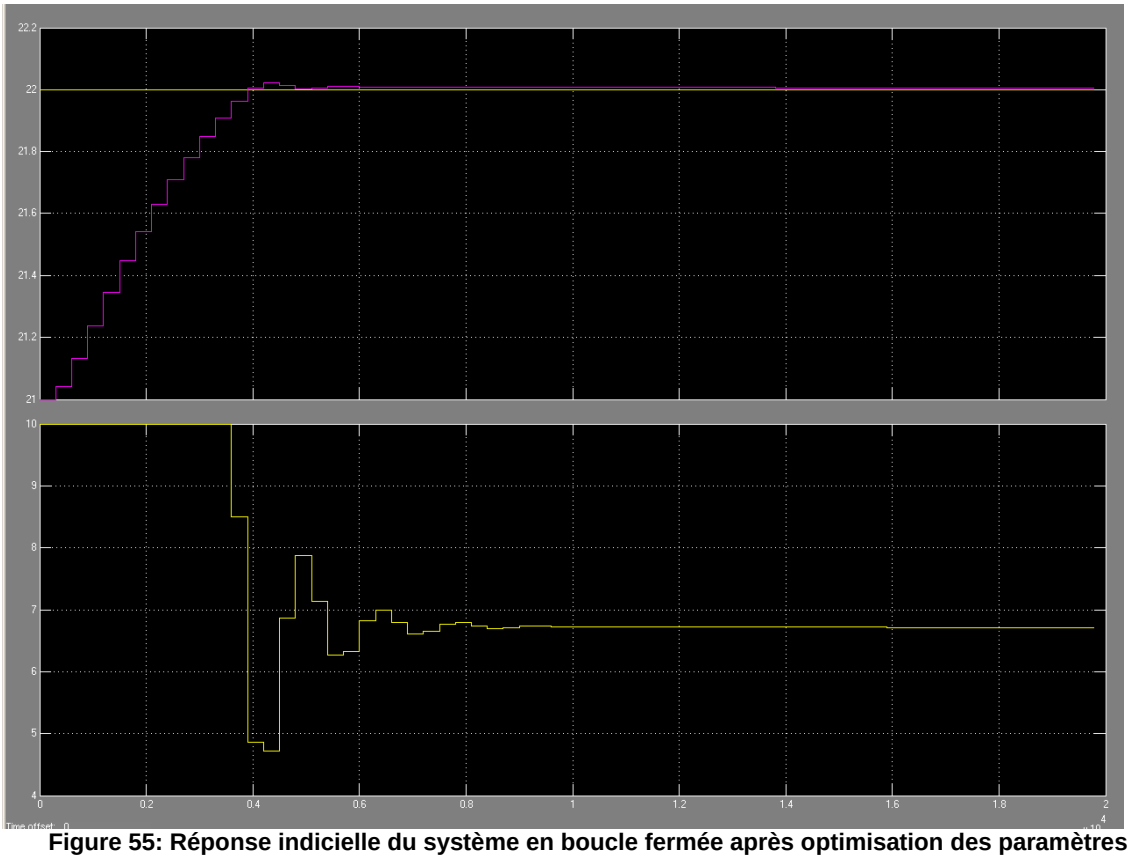


Figure 55: Réponse indicielle du système en boucle fermée après optimisation des paramètres

On peut observer un temps de régulation en boucle fermée d'environ 4000 s (1heure et 6 minutes), et l'absence d'erreur statique ; il reste un très petit dépassement qui n'est pas nocif pour la régulation du système.

6.2.4 Implémentation du régulateur

La structure du régulateur est donc la suivante :

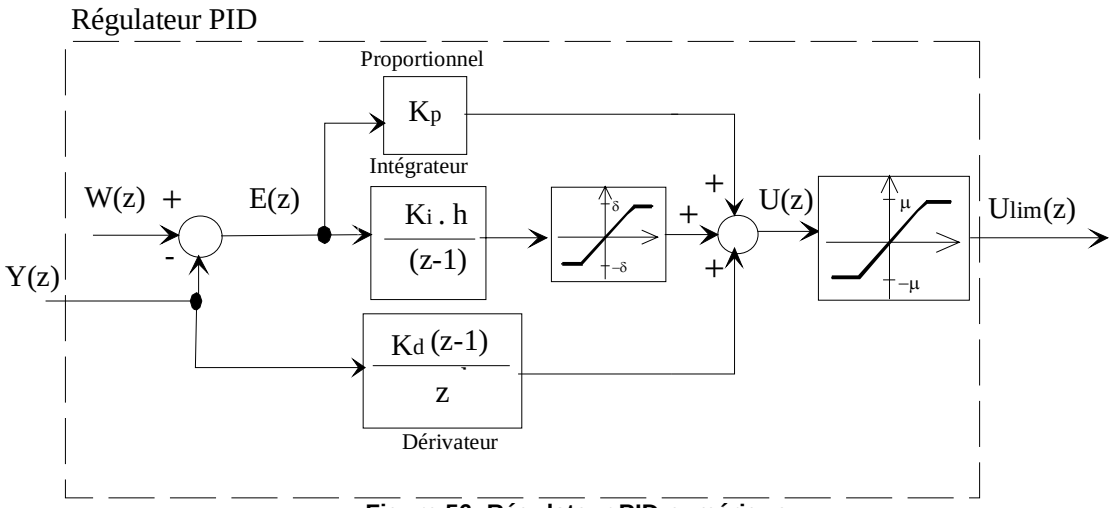


Figure 56: Régulateur PID numérique

On peut reconnaître les 3 éléments du régulateur : P,I et D avec leur fonction de transfert numérique ; suivis de la limitation de la commande selon les limites physiques du système (signal de commande 0-10V).
L'intégrateur est suivi d'un bloc antireset windup : en limitant la sortie on peut générer des commandes très grandes ; cela implique qu'il faudrait beaucoup de temps pour « vider » l'intégrateur après un phénomène transitoire, ce qui pourrait générer des dépassements et des oscillations.

Le pseudo-code d'un PID numérique avec antireset windup est le suivant :

```
Read (y)
e := W - y ;

u' := u^i + (Kp+Ki+Kd)*e - Kd e^old ;
if  u'> + ulim  then  u := ulim
elseif u'<- ulim  then u := -ulim
else u :=u' ;
Output (u)

elim :=e-(u'-u)/ (K^p +K^i +K^d )

u^i = u^i + Ki*elim ;

e^old := e ;
```

Voici le code utilisé pour effectuer la régulation :

```
/*********************************************************************
 *  FUNCTION: PID
 *  GOAL: This function makes a PID regulation on heating
 *  COMMENTS: Initialize parameter on the top of the program
 *********************************************************************/
void PID(void)
{
    consigne()                                          // CHOSE SETPOINT

    erreur = consigne - tempInt;                        // ERROR CALCULATION

    u_prim=ui+(Kp+Ki+Kd)*erreur-Kd*erreur_old;          // REGUL. ALGORITHM

    if(u_prim<0)                                        // OUTPUT LIMITATION
          u=0;
    else if (u_prim>100)
          u=100;
    else
          u=u_prim;


    duty=u;                                 // DUTY CYCLE OF OUTPUT
    CCP1CON &= 0b11001111;                  // DUTY CYCLE CALCULATION
    CCP1CON|=((duty & 0x03))<<4;            // 2 LSB OF DUTY
    CCPR1L= duty>>2;                        // 8 MSB OF DUTY

    erreur_lim=erreur-(u_prim-u)/(Kp+Ki+Kd);        // ANTIRESET WINDUP
    ui=ui+Ki*erreur_lim;
    erreur_old=erreur;

}
```

Remarquez que la consigne est déterminée avant chaque exécution de l'algorithme selon des critères basés sur l'horaire et la présence dans la pièce à régler.
Voir le chapitre 5 pour une explication sur le choix de la consigne.
Voici le code correspondant :

```
/*********************************************************************
 *  FUNCTION:   CONSIGNE
 *  GOAL:       This function calculate normal and reduced setpoints
 *  COMMENTS:   Setpoint as interface choice
 *********************************************************************/

void consigne()
{
    if(Jour>5 ||  Heure<start_time || Heure>=stop_time
    || ((Heure>=11 && Minute>=30) && (Heure<12 && Minute<30)))
          consigne_PID=cons_red;              // REDUCED SETPOINT
    else
    {
          if(presenceCounter>2)   // MORE THAN 10 MIN WITHOUT PRESENCE
                consigne_PID=cons_standby;    // STANDBY SETPOINT
          else
                consignePID= cons_norm;       // CONFORT SETPOINT
    }
}
```

6.3 Commande des stores

Pour les stores, il s'agit d'implanter un régulateur à deux états (tout ou rien).
Ce régulateur travaille selon le principe présenté dans le diagramme de flux du chapitre X.
Voici le code correspondant :

```
/*********************************************************************
 *  FUNCTION:   BLINDS
 *  GOAL:       This function manage blinds command
 *  COMMENTS:   Up/down on auto mode; stop in manual mode.
 *              Mode defined by motion sensor
 *********************************************************************/

void blinds()
{
    if(manualMode==0)                   // AUTOMATIC MODE
    {
          if(storesUp==1 && up==0)      // OPEN BLINDS
          {
                dir_up=1;
                storesUp=0;
                TMR1L=0xCC;
                TMR1H=0xCC;
                TMR1ON=1;               // START TEMPORIZATION
                RELAIS_DOWN=0;          // COMMAND RELAIS
                RELAIS_UP =1;
                up=0;
                down=0;
          }
          if(storesDown==1 && down==0)  // CLOSE BLINDS
          {
                dir_down=1;
                storesDown=0;
                TMR1L=0xCC;
                TMR1H=0xCC;
                TMR1ON=1;               // START TEMPORIZATION
                RELAIS_UP=0;            // COMMAND RELAIS
                RELAIS_DOWN=1;
                down=0;
                up=0;
          }
    }
    else
    {
          TMR1ON=0;                     // MANUAL MODE: STOP BLINDS
          RELAIS_UP =0;
          RELAIS_DOWN =0;
    }
}
```

Chaque 5 minutes l'acquisition de la valeur du rayonnement solaire et de présence, appelle la fonction blinds, qui gère la commande des stores :

```
if(receivedMessage=='s')                // NO SUN RADIATION
{
    storesDown=1;                       // BLINDS DOWN
    blinds();                           // MODIFY BLINDS STATUS
}
if(receivedMessage=='S')                // SUN RADIATION
{
    if(presence==0) storesUp=1;         // BLINDS UP IF AUTO MODE
```

```
    blinds();                           // MODIFY BLINDS STATUS
}
if(receivedMessage=='p')                // NO PRESENCE
{
    manualMode=0;                       // SWITCH TO AUTOMATIC MODE
    blinds();                           // MODIFY BLINDS STATUS
}
if(receivedMessage=='P')                // PRESENCE
{
    manualMode=1;                       // SWITCH TO MANUAL MODE
    blinds();                           // MODIFY BLINDS STATUS
}
```

L'interruption du timer1 met à jour l'état après chaque actionnement :

```
if(TMR1IF==1)
{
    TMR1ON=0;                           // STOP TIMER 1
    if(dir_down==1)
    {
          down=1;                       // ACK BLINDS CLOSED
          up=0;
          dir_down=0;
    }
    if(dir_up==1)                       // ACK BLINDS OPENED
    {
          down=0;
          up=1;
          dir_up=0;
    }
    RA4=0;                              // END BLINDS PULSE
    RB0=0;
    TMR1IF=0;                           // CLEAR FLAG
}
```

La variable manualMode est modifiée selon la présence ou pas de personnes dans la pièce.
Une temporisation de quelques secondes est démarrée pour effectuer une impulsion assez longue sur la commande des stores (Timer 1).
Des variables d'état (up et down) sont utilisées afin de savoir en quelle position se trouvent les stores avant d'envoyer une commande.

7 Tests

7.1 Introduction et concept général

Suite à la conception du système de réglage, l'exécution de tests de performance s'impose.

Le déroulement idéal de ces tests est la comparaison de 2 salles sur des périodes de plusieurs semaines: une salle équipée avec le régulateur développé pendant ce projet et l'autre réglée par le chauffage central et des vannes thermostatiques.

Afin d'effectuer des tests qui puissent fournir des résultats fiables, plusieurs semaines seraient nécessaires, afin d'inverser à chaque semaine les 2 salles et quantifier ainsi l'apport réel du système de réglage ; c'est-à-dire « filtrer » les différences de consommation du chauffage dues à des facteurs externes qui diffèrent pour les deux salles. Il s'agit donc d'une méthode statistique qui permettrait de déterminer le gain minimal et maximal effectif apporté par le système de réglage.

Malheureusement le temps à disposition n'étant pas suffisant, je ne peux fournir que des résultats pas totalement exacts et fiables.
Les facteurs externes ont été donc estimés selon des calculs qui permettent de les quantifier et vérifier l'apport réel du régulateur, en essayant d'avoir une approximation la plus proche possible de la réalité. (Voir chapitre 8)

7.2 Environnement et préparation salles

Après avoir abandonnée l'idée initiale de faire les tests au lycée des Creusets, il a été décidé d'utiliser les salles A112 et A113 dans le bâtiment A de notre école.
Il s'agit de 2 salles de clase de dimensions identiques, placées une à coté de l'autre dans la même orientation (sud-est).
Voici un schéma des salles :

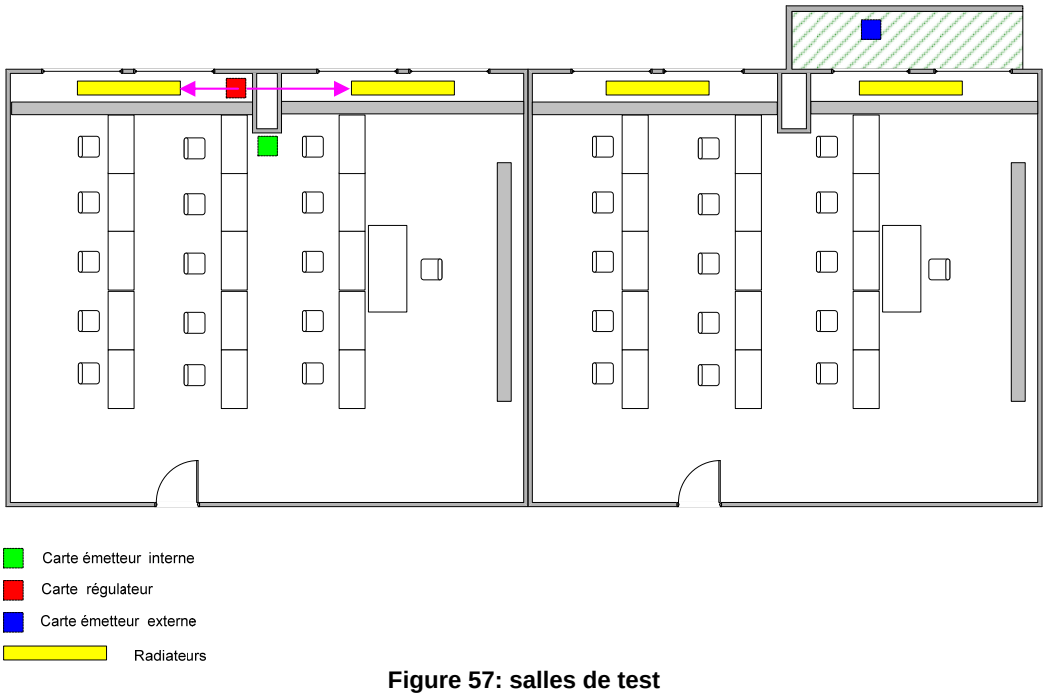


Figure 57: salles de test

Dans chaque salle on trouve 2 radiateurs et 2 stores qui seront commandés en parallèle.

Figure 58: Salles de test

Figure 59: Servovanne sur un radiateur

7.3 Fonctionnement du chauffage - réglage effectué par l'école

Le chauffage de l'école fonctionne de la façon suivante : une chaudière centrale chauffe l'eau pour tout le bâtiment ; un capteur de température interne de référence se trouve dans un couloir au sous sol et un capteur de température externe globale permet de déterminer l'horaire de mise en marche de la chaudière. La température de l'eau qui passe dans les radiateurs varie entre 70 et 90°C.

La consigne journalière est de 21°C et la nuit, à partir de 17h, cette consigne descend à 16°C.

Sur chaque radiateur il y a une vanne thermostatique avec une consigne fixe.

Une mesure de la température interne dans les 2 salles à permis de déterminer la consigne imposée par les vannes thermostatiques, qui est d'environ 22°C.

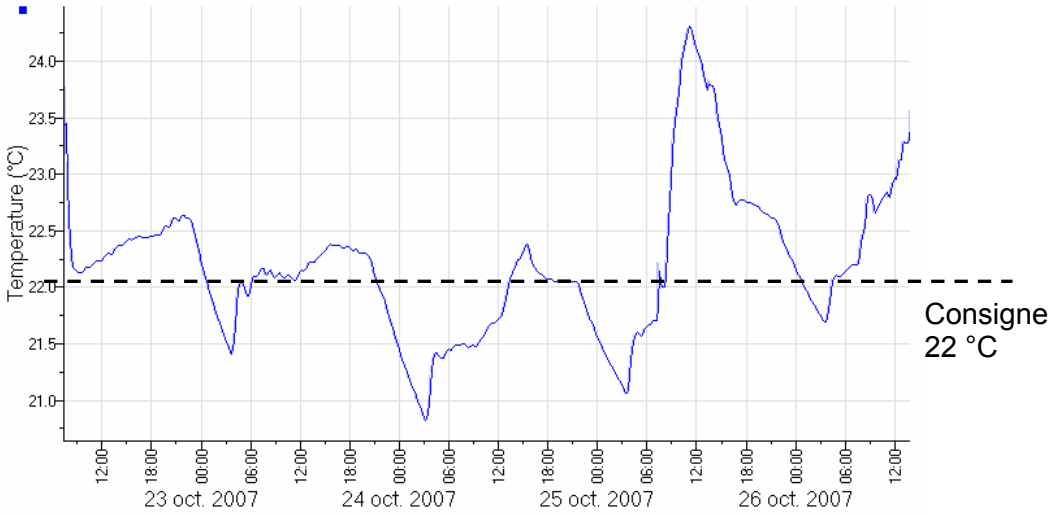


Figure 60: Température dans une salle avec régulation normale

C'est pour cette raison que la consigne journalière de mon système a été choisie à 22°C aussi.

7.4 Positionnement des capteurs

Les capteurs doivent être placés dans une position la plus représentative possible.

Les capteurs de température internes sont placés sur la colonne qui est au milieu des fenêtres (voir figure à la page précédante), à une hauteur de 1,5m environ, afin de pouvoir mesurer la température ressentie par les occupants et pouvoir facilement détecter la présence de personnes dans la pièce.

Figure 61: Capteurs internes et datalogger

Les capteurs externes sont placés, dans une « cage » en aluminium afin de les protéger contre les intempéries, sur la petite terrasse externe à la salle A112, à une distance d'environ 1 m du bâtiment, de façon à ne pas influencer la température (voir figure à la page suivante).

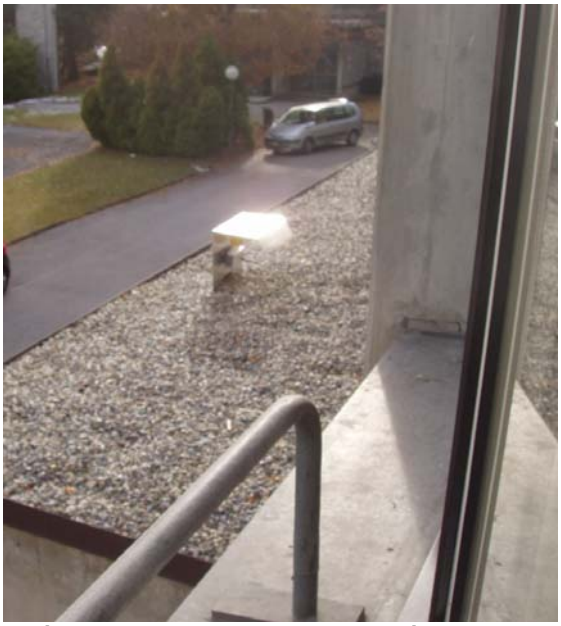


Figure 62: Capteurs externes sur la terrasse

Le seuil de présence de rayonnement solaire a été déterminé expérimentalement (Voir chapitre 5). Des essais ont permis de vérifier qu'en présence de soleil la température mesurée est légèrement supérieure à la température ambiante externe réelle.

7.5 Instrumentation de mesure

Pour effectuer les mesures nécessaires, des dataloggers de tension et température interne/externe ont été utilisés.

Figure 63: Datalogger de température externe (gauche) et interne (droite)

Il s'agit de dataloggers de Tinytag couplés avec un software qui permet la configuration de ces instruments et l'analyse des données, ainsi que l'exportation sous Excel.

Ces dataloggers ont été placés bien évidemment aux mêmes endroits que les capteurs correspondants.

Les grandeurs mesurées sont :

- Température externe
- Température interne
- Température des salles voisines
- Tension sur la servovanne

De plus, pour la mesure de l'énergie consommée, sur chaque radiateur un compteur de chaleur a été monté : il mesure cette énergie en se basant sur les températures d'aller et retour du radiateur ainsi que le débit d'eau qu'y circule.

$$Q_{consommée} = \int m * c * (T_{aller} - T_{retour}) * dt$$

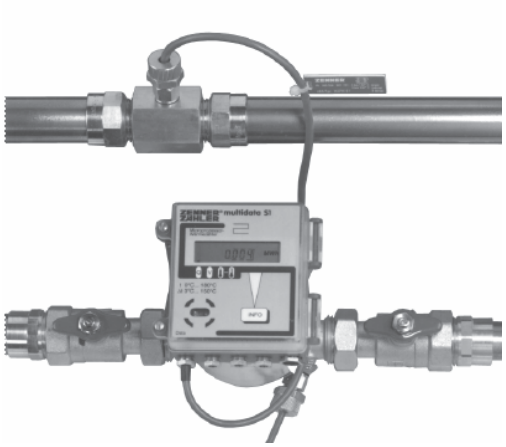


Figure 64: Compteur de chaleur avec affichage numérique

Cette énergie sera utile pour la mesure de la consommation énergétique et pour la création de la signature énergétique.

7.6 Déroulement des tests

Je n'ai pu effectuer que 2 semaines de tests à cause de plusieurs facteur : en premier lieu le montage des compteurs de chaleur n'a été effectué qu'un début du mois de novembre ; de plus le climat était assez doux et il n'était pas possible d'effectuer des tests significatif car le chauffage était au minimum (ou même éteint) la plupart du temps.

7.7 Mesures

7.7.1 Tension sur les vannes

A partir des données fournies par le datalogger, on peut voir que le chauffage est enclenché essentiellement le matin, et le reste de la journée il est éteint.
De plus, pendant le weekend, l'abaissement de la consigne fait en sorte que la vanne reste tout le temps fermée.

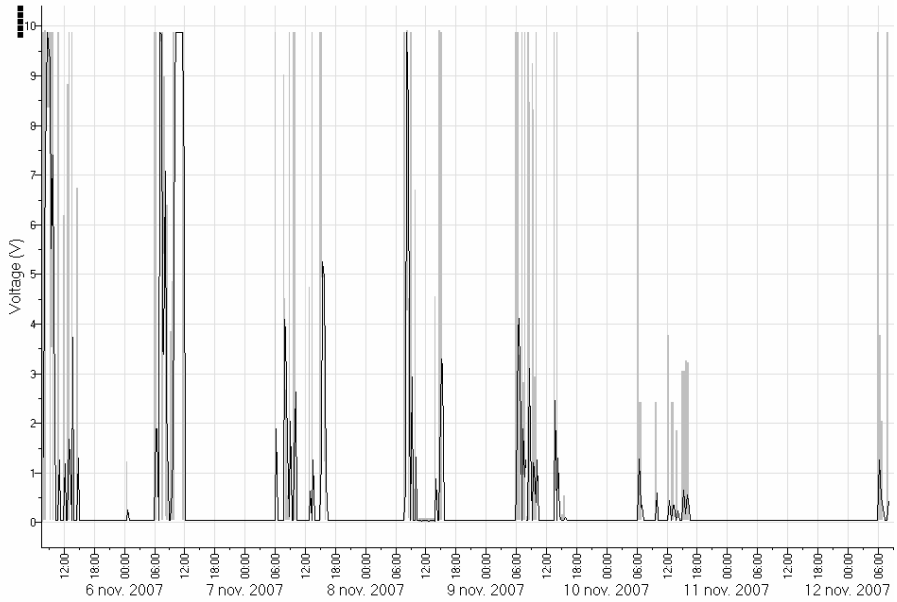


Figure 65: Tension sur la vanne

Sur le graphique qui suit, on voit très bien que la vanne est ouverte à 6h00, heure d'enclenchement, jusqu'à l'atteinte de la consigne « réduite » (21.5°C dans ce cas); ensuite, vers 8h00, quand la présence de personnes dans la pièce est détectée, le changement de consigne génère une deuxième période d'ouverture de la vanne, pour atteindre la consigne « normale » de 22°C.

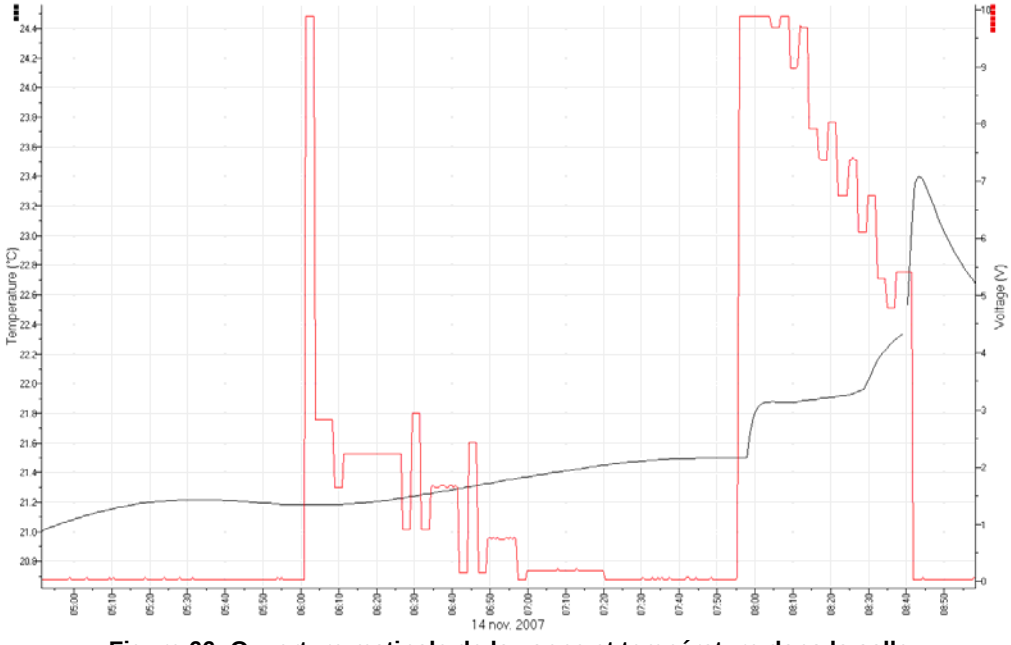


Figure 66: Ouverture matinale de la vanne et température dans la salle

On remarque en premier lieu que la température dans la salle augmente déjà à partir de 5h, car le chauffage central es déjà en marche et les autres salles sont déjà chauffées : la conduction thermique à travers les parois fait en sorte que même dans la salle avec le régulateur la température monte.

Le matin, le but est d'atteindre la consigne à 8h00 ; lors de cette mesure, la température externe était d'environ 5°C, et on voit qu'en démarrant le chauffage à 6h00, la consigne est atteinte bien avant 8h00. Afin d'améliorer cela, une bonne méthode serait d'utiliser la température externe pour déterminer l'horaire de démarrage matinal, afin d'éviter un réchauffement inutile de la salle avant qu'il soit nécessaire.

7.7.2 Température dans les pièces

7.7.2.1 Apport solaire

Le graphique ci-dessous représente l'allure de la température dans une salle pendant 2 journées consécutives dont la première sans soleil et le deuxième avec du soleil pendant la matinée.
Une bonne gestion des stores permet donc d'économiser sur le chauffage.

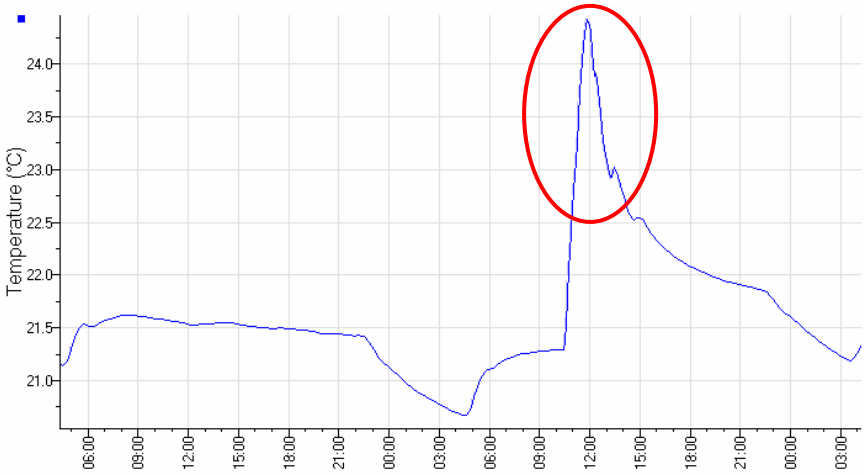


Figure 67: Température salle de tests sur 2 jours

Sur l'image suivante, qui reporte la température dans les deux salles pendant la même journée, on peut voir l'effet de la commande des stores : le pique de température dans la salle réglée par le PID est causé par l'ouverture des stores lorsque le soleil apparaît. Cette chaleur emmagasinée permet ensuite d'économiser de l'énergie ensuite.

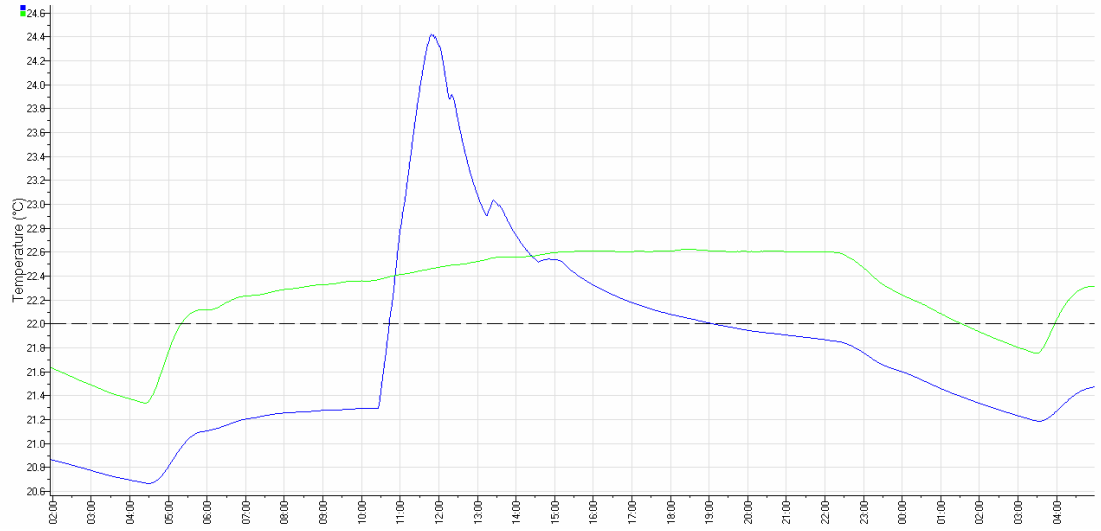


Figure 68: Comparaison des salles sur une journée

7.7.2.2 Occupation salles

L'occupation des salles joue un rôle très important dans les apports énergétiques gratuits(gain internes).
Voici dans le graphique suivant la comparaison entre un dimanche, où il n'y avait personne dans la pièce, et un lundi avec des cours qui avaient lieu dans la salle pendant toute la journée. Pendant les 2 jours le ciel était nuageux.

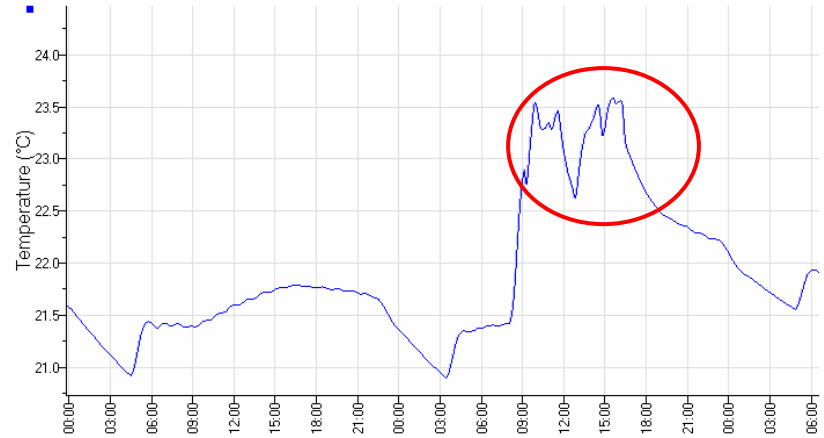


Figure 69: Influence de l'occupation dans une salle

On remarquera la grandeur de l'apport des personnes, qui n'est pas du tout négligeable. Un autre aspect important c'est le pique négatif qui a eu lieu vers 12h : cela est causé par la somme de plusieurs facteurs : plus de personnes dans la pièce, porte et/ou fenêtres ouvertes.

7.7.2.3 Comparaison des 2 salles

Le graphique ci-dessous permet de constater que pendant la journée la consigne est souvent dépassée à cause des apports internes et des gains solaires

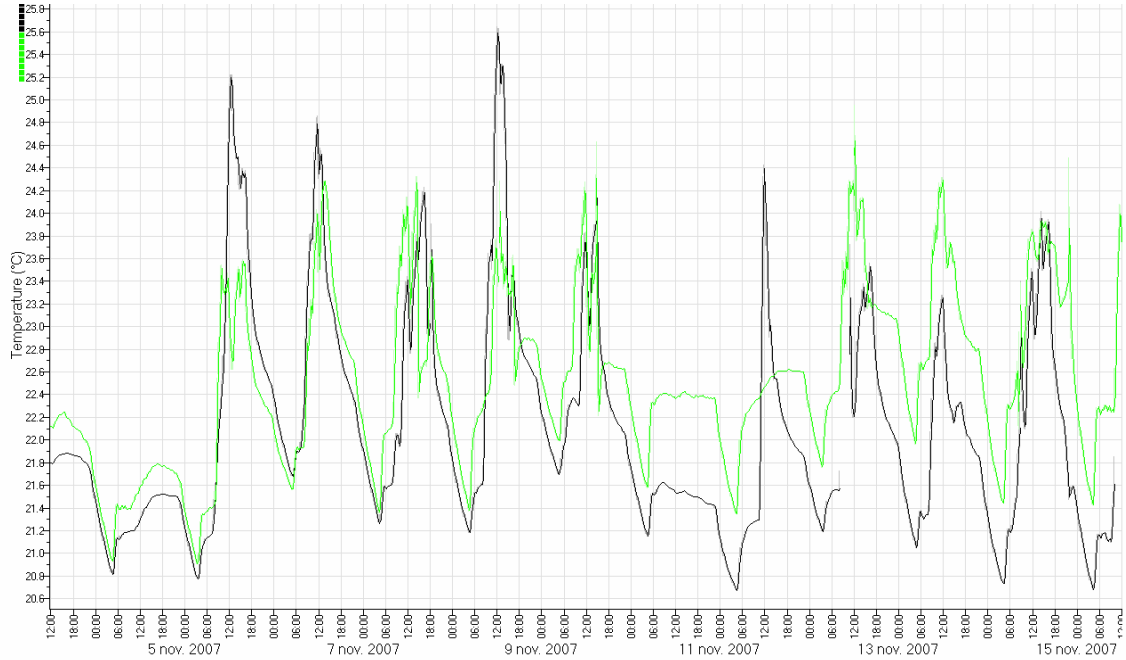


Figure 70: Température salles de test sur 10 jours

On remarque une différence assez importante de 2-3 °C entre les jours dans lesquels il y a des apports passifs et ceux où ces apports sont minimes, pendant les weekends et les journées sans soleil par exemple.

La salle équipée du régulateur PID de plus, possède une température moyennement plus basse : ceci est dû essentiellement à la régulation effectuée par l'école génère des dépassements de la consigne plus importants, et à la consigne réduite de 21 °C en cas de salle pas occupée, qui permet d'économiser de l'énergie.

La température moyenne dans les 2 salles sur la période de tests a été de 22.1 °C dans la salle de test du régulateur PID, et de 22.5 dans l'autre salle.

7.7.3 Température externe

Dans ce graphique on remarque que la température en début du mois de novembre était encore assez douce ; ce qui implique une très petite consommation énergétique et en conséquence il n'est pas possible de tester précisément l'efficacité du régulateur, car les facteurs externes (apports/pertes) sont très importantes par rapport à la consommation énergétique.

En présence de soleil on atteignait des pointes de 18-19 °C en présence de soleil ; la nuit la température minimale a été supérieure à 2-3 °C jusqu'au 6-7 novembre.
La moyenne sur toute la période est de 8.9°C.

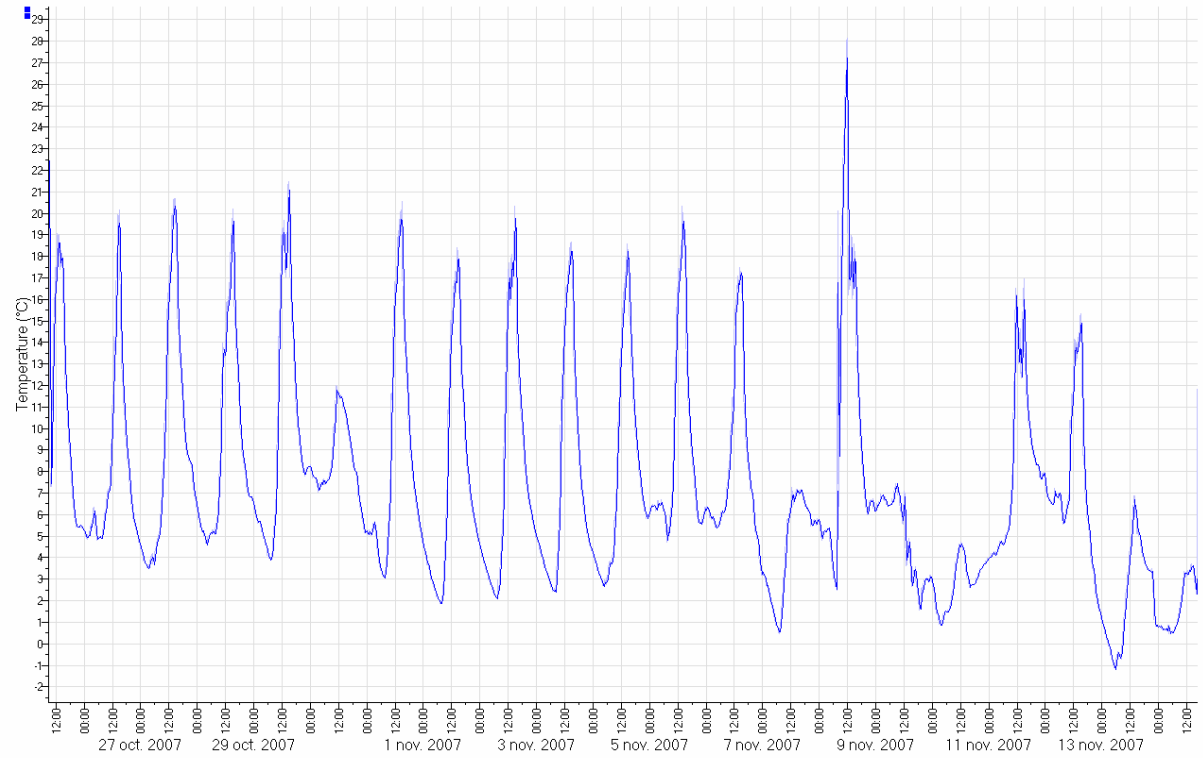


Figure 71: Température externe

Il est intéressant de remarquer que la température maximale est atteinte systématiquement à 15h chaque jour, et la minimale vers 8h, en cas de ciel dégagé.

8 Analyse des résultats

8.1 Critères et outils de comparaison

Le système a été évalué selon principalement 2 critères : l'économie d'énergie et le confort.

8.1.1 Confort thermique

Le confort thermique représente l'aisance à être dans un lieu en fonction de la température qu'y règne.
Les occupants d'une pièce désirent que la température soit la plus stable possible.

Le confort se mesure donc en quantifiant et en sommant les écarts entre la température imposée (consigne) et celle réelle (mesure) : on distingue l'écart positif, l'écart négatif et l'écart type, qui est la somme des 2 écarts.

Écart type	:	$\lvert T_{\text{int}} - consigne \rvert$	
Ecart positif	:	$T_{\text{int}} - consigne$	si $T_{\text{int}} > consigne$
Ecart négatif	:	$consigne - T_{\text{int}}$	si $T_{\text{int}} < consigne$

8.1.2 Energie consommée et signature énergétique

La Signature énergétique est une méthode d'analyse du comportement d'un élément (souvent un bâtiment) du point de vue de la consommation énergétique.
Cette méthode peut également être appliquée à une pièce.
Elle consiste à relever périodiquement la consommation d'énergie et les conditions météorologiques ainsi que la température interne.
Ces données sont ensuite reportées sur un graphique représentant la consommation énergétique en fonction des conditions météorologiques, notamment la température externe, souvent chiffrée en degrés-jour (DJ) ou degrés-heure (DH).

Les points résultants, un pour chaque mesure, sont ensuite comparés avec une droite de référence qui représente la signature énergétique de l'élément à analyser ; c.-à-d. la dépense énergétique de la pièce.

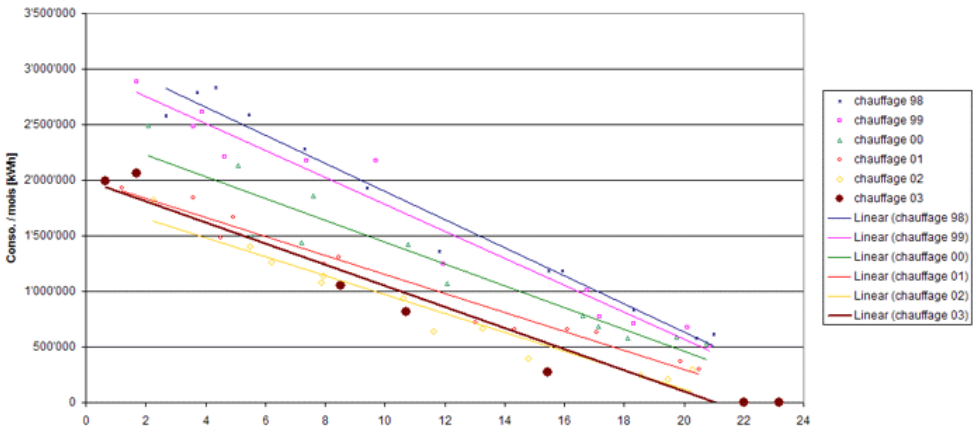


Figure 72: Exemple de signature énergétique d'un bâtiment sur plusieurs années

Un point au dessus de la ligne signifie une consommation excessive, par contre un point en dessous de celle-ci indique une consommation faible. Les points qui s'écartent de la ligne doivent être analysés en essayant de trouver la cause de cet écart, qui parfois peut être simplement dû au climat.

Une analyse attentive permettra donc ensuite d'optimiser la consommation énergétique en apportant des améliorations au système de chauffage.
Le but est de « situer » les mesures le plus bas possible sur le graphique. Evidemment ce but est contré par le confort souhaité pour les habitants, qui a tendance à pousser la courbe vers le haut.

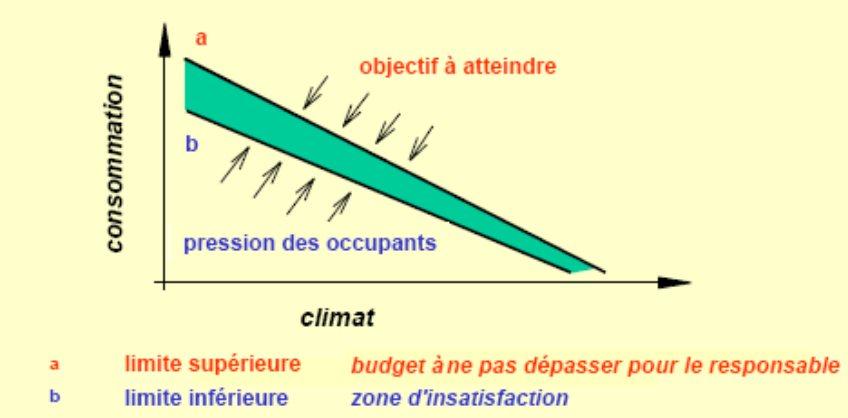


Figure 73: Signature énergétique

Cette méthode est itérative et chaque période permet de surveiller le bon fonctionnement de l'installation de chauffage et apporter des éventuelles modifications. Voici la démarche schématisée :

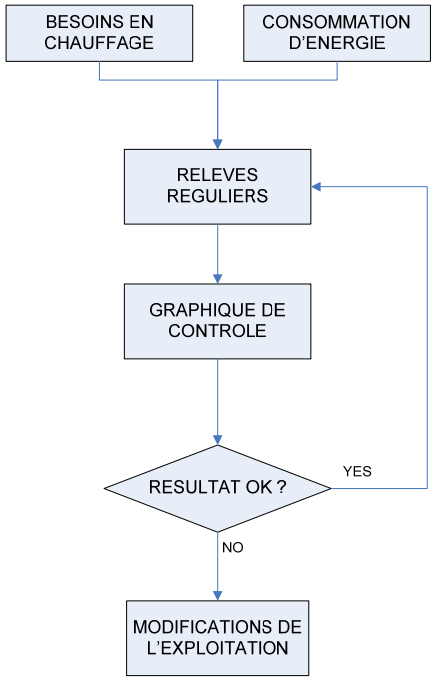


Figure 74: Démarche d'utilisation de la signature énergétique

Dans notre cas la signature énergétique permettra de comparer le réglage effectué par l'école et celui effectué par le régulateur PID.

Degrés-heure

Les degrés-heure est une unité souvent utilisée pour la signature énergétique : il mesure les besoin de chauffage et se calcule en mesurant l'écart entre la température externe et celle interne sur une heure.
Dans la pratique, cette référence (température moyenne interne) est estimée à 18°C pour les pièces habitées.
Un degré heure correspond donc à une différence d'un degré sur un laps de temps d'une heure. Les apports gratuits sont estimés à environ 2-3 °C supplémentaires.

$$DH = 18°C - T_{ext\_moyenne\_horaire}$$

Degrés-jour

Le degré-jour est comparable au degré-heure à la différence près de la période de temps considérée qui est d'un jour et pas d'une heure.
Un degré-jour correspond approximativement à 24 degrés-heure.

Etablissement de la référence

La signature est propre au local, elle dépend des caractéristiques de celui-ci. Dans la pratique cette référence est crée avec les mesures de la première année d'analyse. On peut quand même calculer la droite de référence théorique des besoins de chauffage de la pièce au moyen d'un bilan thermique :

$$E = \int P(t)dt = \int \left[H\left(T_{\text{int}} - T_{ext}\right) - Pa - Ae*Is\right]* dt$$

H	$[W/K]$	Besoins thermiques bruts spécifiques au local (pertes par l'enveloppe)
Ae	$[m^2]$	Surface efficace de captage solaire
Is	$[W/m^2]$	Rayonnement solaire
Pa	$[W]$	Puissance d'éléments annexes au système

Les apports solaires ($Ae*Is$), ainsi que les apports internes sont déjà pris en compte dans la référence de 18°C pour le calcul des degrés-heure.
Le coefficient H dépend de la surface en contact avec l'air extérieur et se calcule en déterminant un coefficient qui détermine la densité des besoins thermiques des murs extérieurs totale de ces surfaces :

$$k_{mur}\left[\frac{W}{m^2K}\right] = \frac{1}{\frac{1}{h_{\text{int}}} + \frac{1}{h_{ext}} + \frac{d_{mur}}{\lambda_{mur}}}$$

h_{int} et h_{ext}	$[W/m^2*K]$	coefficients de transmission de chaleur par convection
d_{mur}	$[m]$	épaisseur des parois
λ_{mur}	$[W/m*K]$	conductivité thermique des parois

$$H = \sum k_i * A_i$$

La puissance des éléments annexes au système est la puissance des radiateurs, et des appareils qui apportent de la chaleur au local.

Sur le graphe suivant j'ai représenté la signature énergétique de référence pour les 2 salles de test.

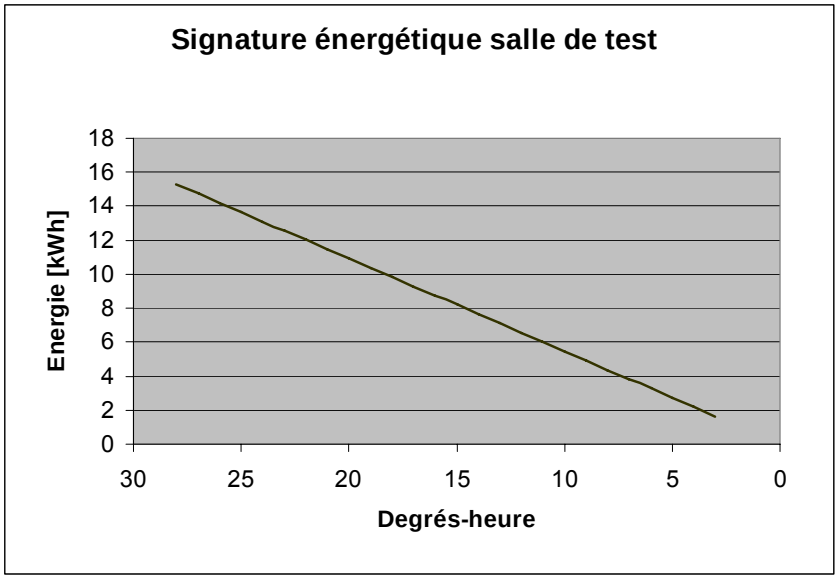


Figure 75: Signature énergétique de référence salles de test

Remarque : Dans ce projet, la relève des données a été effectuée chaque jour.

8.1.3 Consommation d'énergie

Quelques remarques s'imposent au sujet de l'énergie consommée par le chauffage :
En premier lieu, l'énergie est l'intégrale de la puissance thermique instantanée dégagée, sur un certain laps de temps. Elle représente les pertes de la pièce vers l'environnement extérieur. Ces pertes sont principalement les surfaces d'échange avec l'extérieur (fenêtres) et avec les pièces voisines (murs).

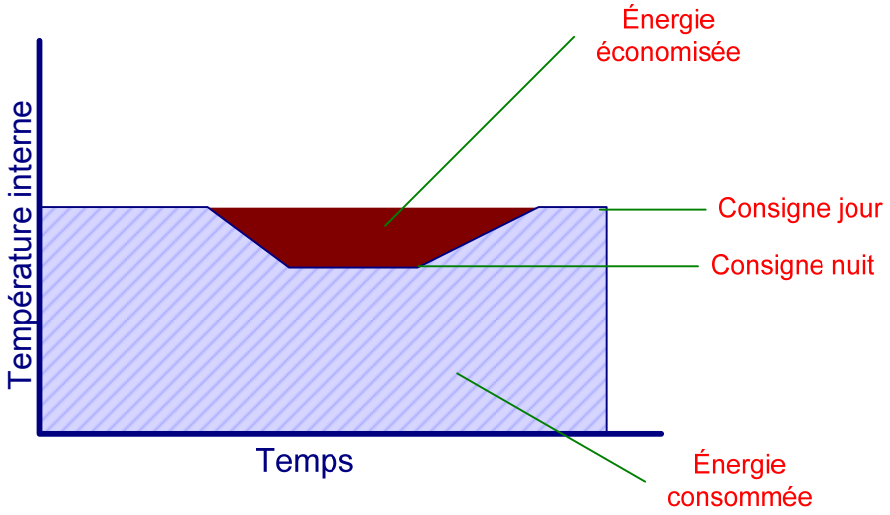


Figure 76: Consommation d'énergie

Sur ce graphique on peut voir le gain énergétique qu'apporte une réduction de consigne pendant la nuit : il faut donc baisser la consigne le plus possible pour économiser de l'énergie.

C'est dans cette direction que va le régulateur PID avec heures de coupure.

Au niveau du bâtiment, étant donné que les pertes sont proportionnelles à la surface d'échange et aux coefficients de convection thermique, il faut essayer de construire avec des matériaux avec des bons coefficients.

8.2 Facteurs d'influence externe

Dans un bâtiment il se passe plusieurs phénomènes thermiques : des échanges avec l'environnement extérieur à travers l'enveloppe et entre les locaux, des gains internes et des apports passifs.

Pour effectuer un bilan thermique on doit tenir compte de tous ces facteurs.
Il s'agit notamment de :

- gain passifs : rayonnement solaire
- gains internes : personnes, appareils électriques
- pertes par l'enveloppe : échanges avec l'extérieur à travers murs et fenêtres
- pertes par conduction : échanges avec les locaux voisins
- pertes par aération : ouverture des fenêtres, des portes
-

Pour ce qui concerne les tests effectués, il est nécessaire d'estimer toute différence entre les deux pièces comparées, afin d'approximer le gain énergétique réel apporté par le système de gestion par pièce.

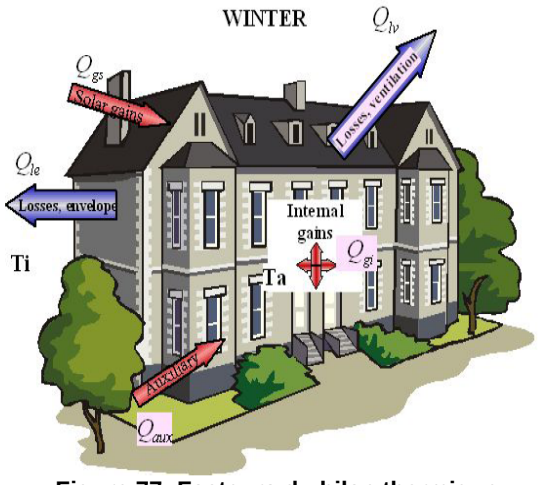


Figure 77: Facteurs du bilan thermique

Certains facteurs tels que les échanges de chaleur avec l'extérieur par les fenêtres, ne sont pas importants car on peut estimer qu'ils affectent de la même façon, ou presque, les 2 salles ; par contre certains autres facteurs qui diffèrent selon les conditions d'utilisation des salles, sont à évaluer :

1. Présence de personnes dans la pièce : le corps humain à repos fournit une puissance calorifique d'environ 60-80W, ce qui implique, avec la présence de 15 personnes dans la salle, un apport énergétique d'environ 1.2 kWh pendant 2 heures de cours, ce qui n'est de loin pas négligeable.

2. Ouverture de portes et fenêtres : un débit d'air de température différente à celle de la salle va y rentrer et modifier la température interne ; cet air a 2 effets : le refroidissement de l'air dans la pièce ainsi que le refroidissement des murs sur une certaine profondeur liée au temps. Le refroidissement de l'air ambiant par rapport à celui des murs peut être négligé en raison de la masse volumique de l'air qui est beaucoup plus petite que celle du béton.

   On peut estimer ces pertes comme il suit : la profondeur de propagation de la chaleur dans un mur en fonction du temps s'estime avec la formule :

$$\partial = \sqrt{\frac{\lambda_{mur} * t}{c_{parois} * \rho_{parois}}}$$

∂	$[m]$	Profondeur de propagation de l'onde de chaleur pendant le temps t
c_{parois}	$[m]$	chaleur massique du béton
λ_{mur}	$[W/m*K]$	conductivité thermique des parois
ρ_{parois}	$[kg/m^3]$	masse volumique du béton

L'énergie perdue se calcule en sommant toutes les surfaces (sol, plafond et murs) de la salle :

$$Q = m * c * \Delta T = \sum m_{parois} * c_{parois} * (Tin - Text) = \sum C_{salle} * (Tin - Text)$$

Où C_{salle} est la capacité calorifique des parois :

$$C_{salle} = m_{parois} * c_{parois} = A_{tot} * \partial = A_{tot} * \sqrt{\frac{\lambda_{mur} * t}{c_{parois} * \rho_{parois}}} * \rho_{parois} * c_{parois} =$$

$$= A_{tot} * \sqrt{\lambda_{mur} * \rho_{parois} * c_{parois} * t}$$

3. Température des salles voisines : des échanges de chaleur se produisent à travers les parois des salles en cas d'une différence de température entre salles voisines. Ceci peut être quantifié de la façon suivante :

$$Q = \frac{\Delta T}{Rth} * A * \Delta t$$

$$Rth = \frac{1}{h_1} + \frac{1}{h_2} + \frac{d_{mur}}{\lambda_{mur}}$$

A	$[m^2]$	surface des parois
d_{mur}	$[m]$	épaisseur des parois = 0.014
λ_{mur}	$[W/m*K]$	conductivité thermique des parois = 0.44
h_1 et h_2	$[W/m^2*K]$	coeff. de transmission de chaleur par convection= 8 et 20

4. Les échanges de chaleur avec l'extérieur ont quand même été quantifiés afin de rendre les résultats plus précis : les déperditions à travers les fenêtres se calculent de façon semblable à la transmission de chaleur à travers les murs, car il s'agit du même phénomène

$$Q = \frac{\Delta T}{Rth} * A * \Delta t$$

$$Rth = \frac{2}{h_1} + \frac{d_{fenetre}}{\lambda_{verre}}$$

A	$[m^2]$	surface d'échange avec l'extérieur
$d_{fenetres}$	$[m]$	épaisseur des fenêtres
$\lambda_{fenetres}$	$[W/m*K]$	conductivité thermique des fenêtres = 0.91
h_1	$[W/m^2*K]$	coefficient de transmission de chaleur par convection = 8

5. Appareils internes : l'illumination de la pièce et les appareils qui s'y trouvent (par exemple projecteurs), génèrent, de même que les occupants, des apports d'énergie.

La différence de gain solaire par rapport à l'état des stores (ouvertes/fermées) n'est pas prise en compte, car cette différence est due essentiellement à la régulation des stores mêmes ; ce qui fait partie du le gain apporté par le système ; de plus ce gain est aussi dépendant de la façon d'utiliser la salle par les occupants.

On s'aperçoit immédiatement qu'avec tous ces facteurs en jeu, il est difficile d'effectuer des calculs précis.

8.3 Analyse et discussion des résultats obtenus

8.3.1 Ecarts de température

Le premier critère de comparaison, le confort, a été mesuré en sommant les écarts de température sur une journée.

Il est très difficile de comparer les 2 salles du point de vue du confort et de déterminer quelle est la contribution du système de réglage.

Cette mesure n'a été effectuée que pendant des journées sans gains passifs, notamment sans occupation des salles et sans apport solaire, car dans ces cas des importants écarts positifs se présentent et vont fausser les mesures.

Voici les résultats et la comparaison des 2 systèmes pendant une journée sans soleil et sans occupation des salles, pendant une période de 5 heures.

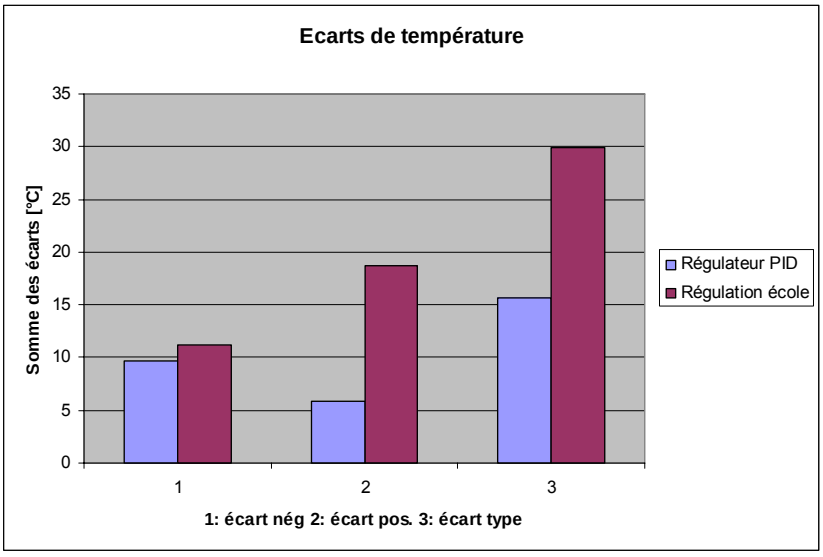


Figure 78: Ecarts de température (par rapport à 22°C)

On remarque qu'avec le régulateur PID on obtient une précision supérieure d'environ 40% qu'avec la régulation effectuée par l'école.

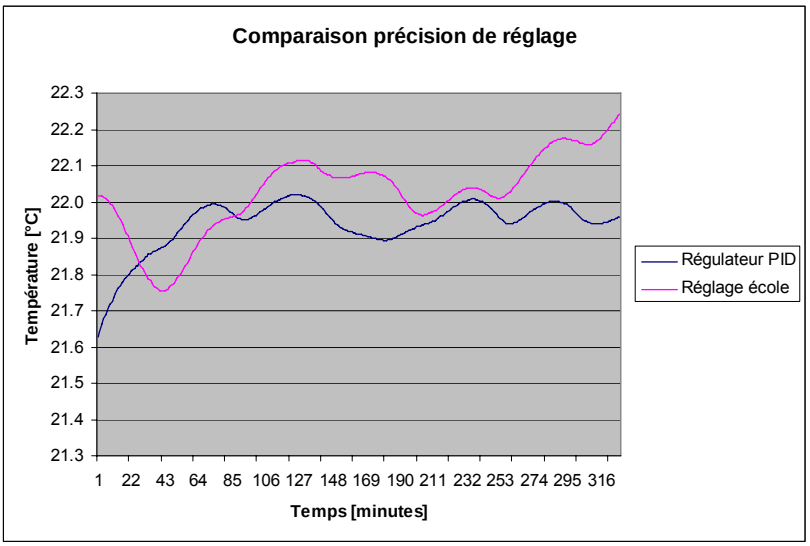


Figure 79: Température dans les 2 salles – début de la mesure : 6h

La vanne thermostatique génère des dépassements de la consigne plus importants que le PID ; cela se traduit en un gaspillage d'énergie.
Comme mentionné dans le chapitre 8, à cause des dépassements de la consigne, la moyenne de température interne dans la salle régulée par l'école, sur 2 semaines, est d'environ 0.4 °C.

Le graphique ci-dessous montre l'allure de la température dans les 2 salles pendant une journée sans occupation et pas ensoleillée.
La courbe en bleu représente la salle réglée par le PID (A112 – courbe rouge) ; la consigne était de 21°C car il n'y avait pas de présence. La courbe verte représente la température dans l'autre salle (A113 – courbe noire).
On peut remarquer plusieurs aspects :

- le chauffage central démarre à 4h du matin : la salle A113 est tout de suite chauffée et il se vérifie un réchauffement de la salle réglée A112, à cause de la conduction thermique des parois.
- le chauffage dans la salle A112 démarre à 6h, selon les paramètres choisis
- La précision du régulateur PID est de loin meilleure de celle de la vanne thermostatique

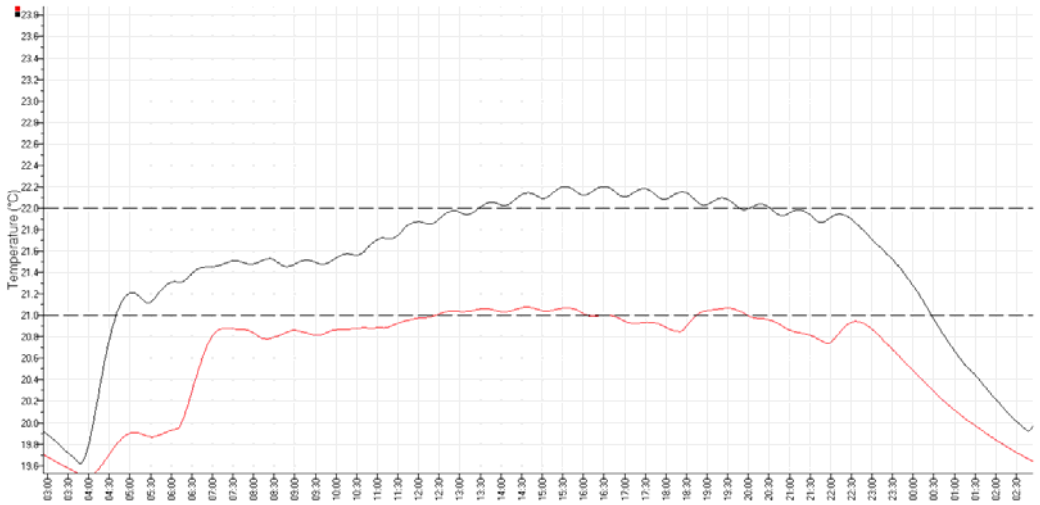


Figure 80: Comparaison précision de réglage température, samedi 18 novembre

8.3.2 Consommation d'énergie

La consommation énergétique des radiateurs indiquée par les compteurs de chaleur, corrigée avec les facteurs d'influence, permet d'estimer le gain d'énergie effectivement apporté par le régulateur.

$$Q_{effective} = Q_{compteur} + Q_{personnes} - Q_{echanges\_externes} - Q_{aération} \text{ (Pour chaque salle)}$$

$$\Delta Q = \frac{Q_{effective\ -salle\_normale} - Q_{effective\ -salle\_test}}{Q_{effective\ -salle\_normale}}$$

En analysant les résultats obtenus, j'ai remarqué plusieurs aspects, par rapports aux phénomènes décrits au chapitre 8.2 :

Tout d'abord les transmissions de chaleur à travers les murs, en particulier vers les salles voisines, ont une influence assez importante : la salle à côté de celle réglée par le PID étant en moyenne plus chaude que celle à côté de la salle pas réglée ; les pertes par conduction pour les deux salles comparées sont donc très différentes.

Les portes ont aussi une influence : si on laisse une porte ouverte, ceci va générer beaucoup plus de pertes que si on la laisse fermée. Le même discours est valable pour les fenêtres. Cependant, il est impossible de déterminer l'état des portes, donc ce facteur n'a pas été pris en ligne de compte. Pour ce qui concerne les fenêtres, on peut estimer les temps d'ouverture en analysant la mesure de température, notamment les chutes brusques de température.

Bref, pour conclure, encore un fois, les résultats numériques sont à prendre avec des pincettes ; mais on peut affirmer qu'un gain réel est apporté par le réglage effectué.

Pour l'analyse des données, j'ai rapporté les mesures sur un tableau en effectuant des moyennes horaires de température et en calculant les pertes et apports susmentionnés. Ceci m'a permis d'avoir des valeurs plus « propres».

Sur la page suivante on trouve un exemple de mesures sur une journée.

		A112	A111	A113	A114		Salle régulateur PID (A112)						Salle régulation normale (A113)							
Date	Horaire	Temp. interne salle test	Temp. salle à côté	Temp. interne salle normale	Temp. salle à côté	Temp. externe	Pertes murs	Pertes fenetres	Apport personnes	Pertes aération	Energie compteur	Total pertes	Pertes murs	Pertes fenetres	Apport personnes	Pertes aération	Energie compteur	Total pertes	% écon. "brut"	Tot. Econ.
	[-]	[°C]	[°C]	[°C]	[°C]	[°C]	[Wh]	[Wh]	[Wh]	[Wh]	[Wh]	[Wh]	[Wh]	[Wh]	[Wh]	[Wh]	[Wh]	[Wh]	[%]	[%]
7.11	08.00	21.8	21.3	22.7	20.3	1.0	-13.91	400.40	0.00				97.37	417.73	600.00					
7.11	09.00	22.0	21.2	23.5	20.7	3.0	-24.34	365.75	0.00				132.15	394.63	600.00					
7.11	10.00	22.4	21.4	23.8	20.5	5.0	-13.91	334.95	600.00				146.06	361.90	600.00					
7.11	11.00	23.3	22.0	23.9	20.9	6.0	24.34	333.03	600.00				107.81	344.58	600.00					
7.11	12.00	23.1	21.4	23.4	20.4	7.0	48.69	309.93	0.00				97.37	315.70	0.00					
7.11	13.00	23.2	21.6	23.9	18.7	7.0	31.30	311.85	600.00				187.79	325.33	600.00					
7.11	14.00	23.7	21.8	23.3	20.0	7.0	79.98	321.48	600.00				83.46	313.78	600.00					
7.11	15.00	23.9	22.0	22.6	20.5	7.0	111.3	325.33	600.00				10.43	300.30	0.00					
7.11	16.00	23.8	22.5	22.7	20.5	7.0	83.46	323.40	600.00				20.87	302.23	0.00					
7.11	17.00	22.9	23.0	23.1	20.5	6.0	-10.43	325.33	0.00				79.98	329.18	600.00					
7.11	18.00	22.9	22.0	23.5	20.5	6.0	10.43	325.33	0.00				107.81	336.88	600.00					
7.11	19.00	22.5	22.0	22.9	20.5	6.0	3.48	317.63	0.00				79.98	325.33	600.00					
7.11	20.00	22.3	22.0	22.6	20.5	6.0	0.00	313.78	0.00				66.07	319.55	0.00					
7.11	21.00	22.2	22.0	22.6	20.5	5.5	-6.96	321.48	0.00				69.55	329.18	0.00					
7.11	22.00	22.1	21.0	22.5	20.5	5.5	24.34	319.55	0.00				66.07	327.25	0.00					
7.11	23.00	21.9	21.0	22.2	20.2	5.5	20.87	315.70	0.00				62.60	321.48	0.00					
8.11	00.00	21.7	21.0	22.0	19.9	5.5	13.91	311.85	0.00				66.07	317.63	0.00					
8.11	01.00	21.5	21.0	21.8	19.7	5.0	6.96	317.63	0.00				66.07	323.40	0.00					
8.11	02.00	21.4	21.0	21.7	19.5	5.0	3.48	315.70	0.00				69.55	321.48	0.00					
8.11	03.00	21.3	21.0	21.5	19.4	5.0	3.48	313.78	0.00				62.60	317.63	0.00					
8.11	04.00	21.2	21.0	21.9	19.4	5.0	-17.39	311.85	0.00				93.90	325.33	0.00					
8.11	05.00	21.4	22.0	21.4	20.1	5.0	-20.87	315.70	0.00				27.82	315.70	0.00					
8.11	06.00	21.5	22.0	22.1	20.1	4.0	-38.25	336.88	0.00				73.03	348.43	0.00					
8.11	07.00	21.6	22.5	22.2	20.1	3.0	-52.16	358.05	0.00				76.51	369.60	0.00					
		22.3	21.7	22.7	20.2	5.3	267.78	7846.30	3600.00	0.00	1830.20	4514.08	1950.93	8004.2	5400.00	0.00	2290.70	4555	20.10	38.42

Voici un tableau récapitulatif des résultats obtenus :

mesure	énergie salle normale	énergie salle PID	Gain « brut »	Gain « net »
[jour]	[Wh]	[Wh]	[%]	[%]
1	1709.50	1502.40	12.11	-10.86
2	2290.70	1830.20	20.10	38.42
3	2280.00	3019.20	-32.42	34.95
4	1383.90	363.00	73.77	48.05
5	3789.60	2621.40	30.83	46.37
6	3607.90	3787.00	-4.96	-37.77
7	2140.10	934.70	56.32	56.32
Totale			10%	6%
Moyenne	-	-	22%	25%

Les mesures ci-dessus sont les plus significatives.

Dans ce tableau j'ai calculé : un gain « brut » qui correspond à la différence d'énergie consommée par les 2 salles ; un gain « net » qui correspond au gain d'énergie adapté avec les apports gratuits estimés et les pertes externes.

Dans la mesure 3, l'énergie consommée est de 30% plus importantes pour le régulateur PID : cela est explicable par une occupation des salles très différentes ce jour-là ; en effet en réalité un gain réel d'environ 35 % a été apporté.
La mesure 4 représente un weekend : on voit que l'énergie économisée monte à presque 50%, car avec l'ouverture des stores pendant la journée de samedi, on a emmagasiné assez d'énergie pour maintenir la température jusqu'à lundi.

Parfois, à cause de problèmes de calcul et d'estimation, j'ai obtenu des résultats incorrects, forcément parce que quelques facteurs n'avaient pas été pris en compte.

En totale, sur la période de test de 2 semaines, le régulateur PID a permis d'économiser environ le 10% d'énergie consommée par les radiateurs, c'est-à-dire un total de 3kWh sur 31kWh totales consommés.

Pour ce qui concerne la moyenne, on remarque un gain moyen d'environ 20% d'énergie.

8.3.3 Signature énergétique

La signature énergétique nous permet de comparer la consommation des 2 salles en fonction des degrés-heure, c'est-à-dire lu climat.

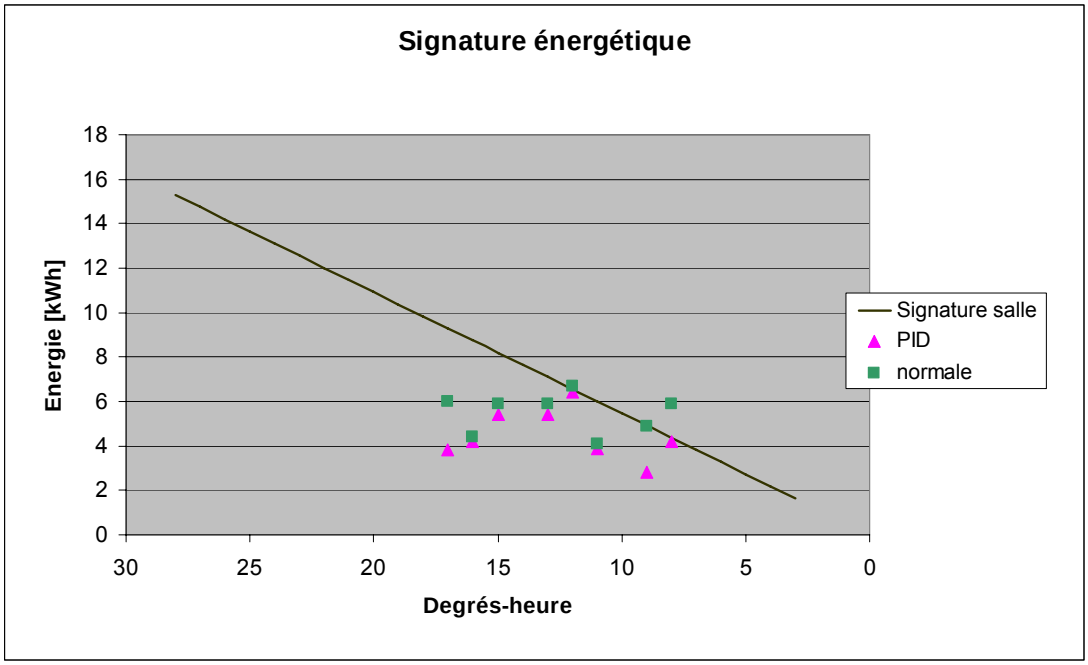


Figure 81: Signature énergétique

On remarque bien que les points du régulateur PID sont plus en bas de ceux de la vanne thermostatique : ceci confirme la plus faible consommation d'énergie du PID.

9 Conclusion, commentaires, améliorations

9.1 Points forts du projet

Par rapport aux projets effectués au sein de l'école dans le passé, sur le thème de la régulation du chauffage, le fait de gérer les stores joue un rôle très important car, en dehors des heures d'occupation, permet de minimiser les pertes vers l'extérieur et de maximiser l'apport solaire.

Par rapport à un système traditionnel, le régulateur PID permet d'avoir une précision majeure, donc des écarts mineurs.

9.2 Améliorations futures

Au niveau hardware le prototype est fonctionnel est fiable ; si on décide d'industrialiser ce système de régulation il faudra par contre optimiser le tout en cherchant des composants plus économiques et qui consomment le moins d'énergie possible, pour garantir une longue durée de la batterie d'alimentation des cartes émetteurs.

Pour ce qui concerne le réglage, comme on l'a déjà dit auparavant, le défaut du PID est le fait qu'il est dimensionné pour un point de fonctionnement, donc il n'effectue pas une régulation optimale sur toute la plage d'utilisation du système.
Il serait donc bonne chose d'implanter un régulateur auto-adaptatif qui calcule les paramètres en fonction des conditions d'utilisation.

L'adaptation du régulateur selon les conditions météorologiques, ainsi que la prévision de l'occupation des salles, devraient permettre une grande amélioration du réglage.

La mesure de température externe pourrait être intégrée dans la stratégie de régulation de plusieurs façons : par exemple pour déterminer l'horaire de démarrage du chauffage ou pour différencier les consignes afin de minimiser les gaspillages d'énergie.

Avant de changer la stratégie de réglage, il serait nécessaire d'effectuer des tests sur plusieurs semaines de façon statistique, comme expliqué dans le chapitre 8.1 : de cette façon on pourra déterminer avec précision les bénéfices apportés par la régulation.

9.3 Conclusion technique

9.3.1 Tests des cartes

Les cartes développées pendant le projet de semestre présentaient quelques problèmes de fonctionnalité.
Elles ont donc subi des modifications et maintenant le bon fonctionnement est assuré.

9.3.2 Programmation

La programmation du système a pris quelques semaines ; d'abord pour décider une stratégie et ensuite pour écrire le code et tester le bon fonctionnement.
A l'état actuelle tout problème a été résolu.

9.3.3 Régulation

Le seul régulateur implémenté et testé est un PID avec horaires de coupure.
Les horaires de coupures, donc des consignes réduites, permettent une bonne économie sur le chauffage.

9.3.4 Tests

Je rappelle encore une fois au lecteur que les tests effectués ont permis d'avoir des chiffres approximatives, et ne permettent pas de déterminer avec précision le gain apporté par le système de réglage.

Le fait d'utiliser des salles occupées génère beaucoup plus d'inégalités dans les salles de test et donc des difficultés dans les calculs, faisant diminuer la fiabilité des résultats.

Toutefois on peut dire que le réglage effectué apporte effectivement un gain dans la consommation énergétique : ce chiffre est en moyenne 20 %.

Il est également possible d'affirmer que les écarts de température sont moins importants par rapport à la régulation effectuée par l'école, car les dépassements de la consigne sont beaucoup moins importants.

L'essai effectué montre que, par rapport à la régulation des vannes thermostatiques, on atteint une précision supérieure d'environ 40% avec le régulateur PID.

9.4 Conclusion générale

Les objectifs imposés dans ce travail de diplôme ont été atteints : les cartes ont d'abord été testées et les modifications nécessaires au bon fonctionnement ont été ensuite apportées ; la communication RF entre les émetteurs et le récepteur-régulateur a aussi été testée, avec des bons résultats.

Ensuite l'algorithme de réglage a été défini, conçu, implémenté et testé ; la réalisation d'un suivi expérimental pendant les deux dernières semaines du projet a ensuite permis de vérifier que le système apporte bien un gain.

La durée du projet n'a par contre pas permis de quantifier avec sureté le réel apport du système.

10 Remerciements

Pour conclure, je souhaite remercier d'abord ma responsable de projet madame F. Butzberger et les personnes qui m'ont aidé pendant mon travail de diplôme, pour leur disponibilité; notamment C. Truffer, A.Vaccari, M. Bonvin et l'atelier d'électronique.

Sion, le 23.11.2007 Matteo Belometti

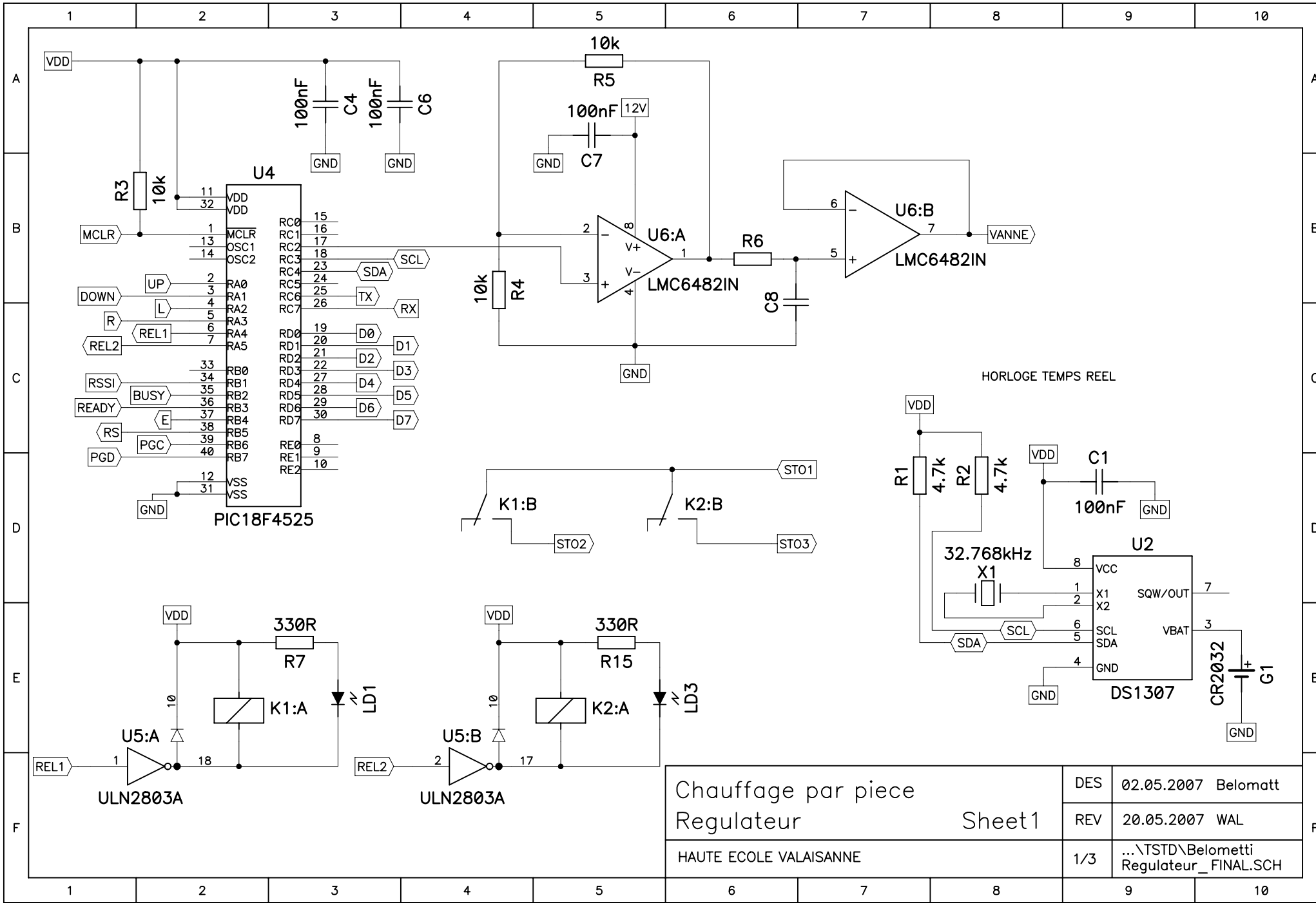
11 Bibliographie

- J.M. Marcuard, F. Butzberger, *cours de Mesure Commande Régulation 1ère partie*, Chapitre 6, Comportement temporel des systèmes, v.1.02, 10.2007
- J.M. Marcuard, F. Butzberger, *cours de Mesure Commande Régulation 2ème partie*, Chapitre 3, Réglage numérique, v.1.02, 10.2007
- M. Bonvin, *cours de Physique, chapitre 6 « thermodynamique »*, HEVS, 2006
- Rapport du travail de diplôme de J. Bonnard : Réglage d'une servovanne thermostatique, 2001
- Rapport du travail de diplôme de C. Monnet : Commande et réglage d'une servovanne thermostatique,1999

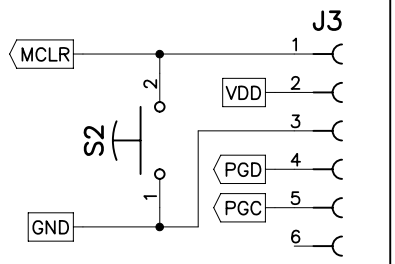
12 Annexes

Annexe 1	:	Schéma électrique et layout PCB cartes émetteur
Annexe 2	:	Schéma électrique et layout PCB carte régulateur
Annexe 3	:	Prix des cartes
Annexe 4	:	Programme émetteur interne et diagramme de flux
Annexe 5	:	Programme émetteur externe et diagramme de flux
Annexe 6	:	Programme régulateur et diagramme de flux
Annexe 7	:	CD avec datasheets et description du contenu

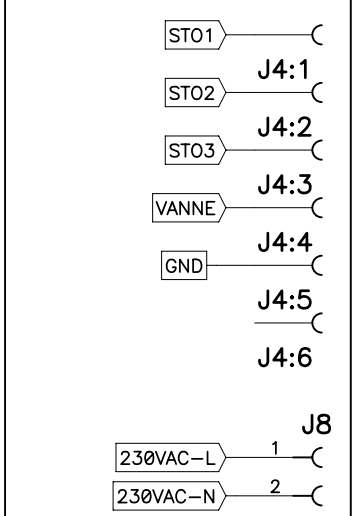
ANNEXE 1


VDD
100nF
C4
100nF
C6
GND
GND
10k
R5
100nF
12V
C7
GND
R3
10k
U4
MCLR
UP
DOWN
L
R
REL1
REL2
RSSI
BUSY
READY
E
RS
PGC
PGD
SCL
SDA
TX
RX
D0
D1
D2
D3
D4
D5
D6
D7
PIC18F4525
10k
R4
U6:A
LMC6482IN
R6
C8
U6:B
LMC6482IN
VANNE
GND
HORLOGE TEMPS REEL
R1
4.7k
R2
4.7k
C1
100nF
32.768kHz
X1
U2
DS1307
CR2032
G1
STO1
STO2
STO3
K1:B
K2:B
330R
R7
K1:A
LD1
U5:A
ULN2803A
REL1
330R
R15
K2:A
LD3
U5:B
ULN2803A
REL2
Chauffage par piece
Regulateur
Sheet1
HAUTE ECOLE VALAISANNE
DES
02.05.2007 Belomatt
REV
20.05.2007 WAL
1/3
...\TSTD\Belometti
Regulateur_ FINAL.SCH

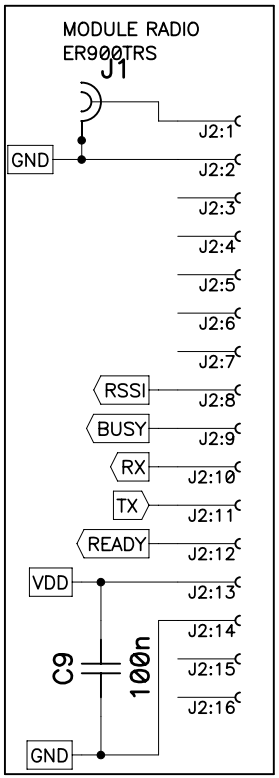
PRISE ICD2



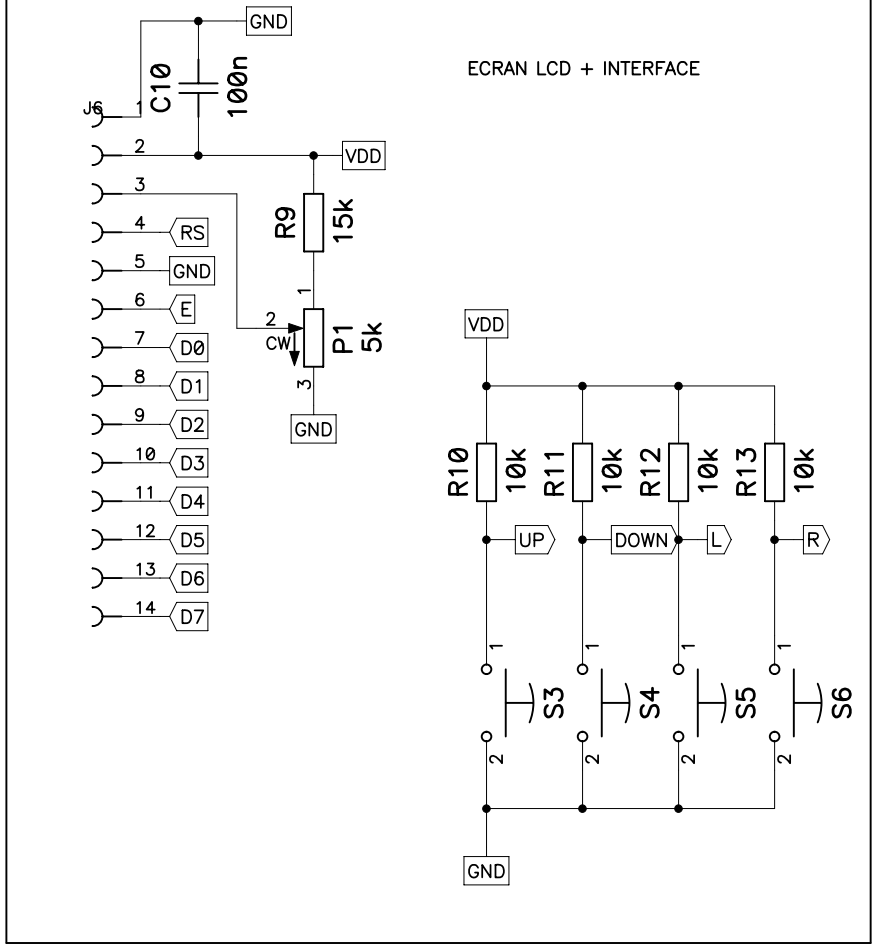
BORNIER I/O



MODULE RADIO
ER900TRS

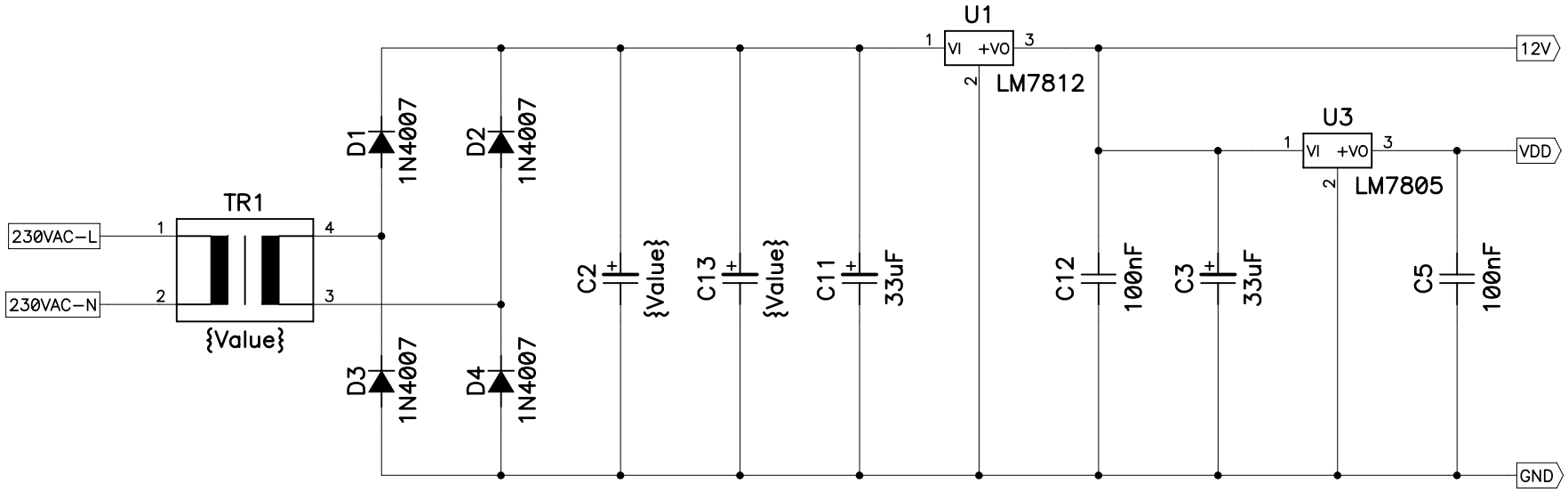


ECRAN LCD + INTERFACE

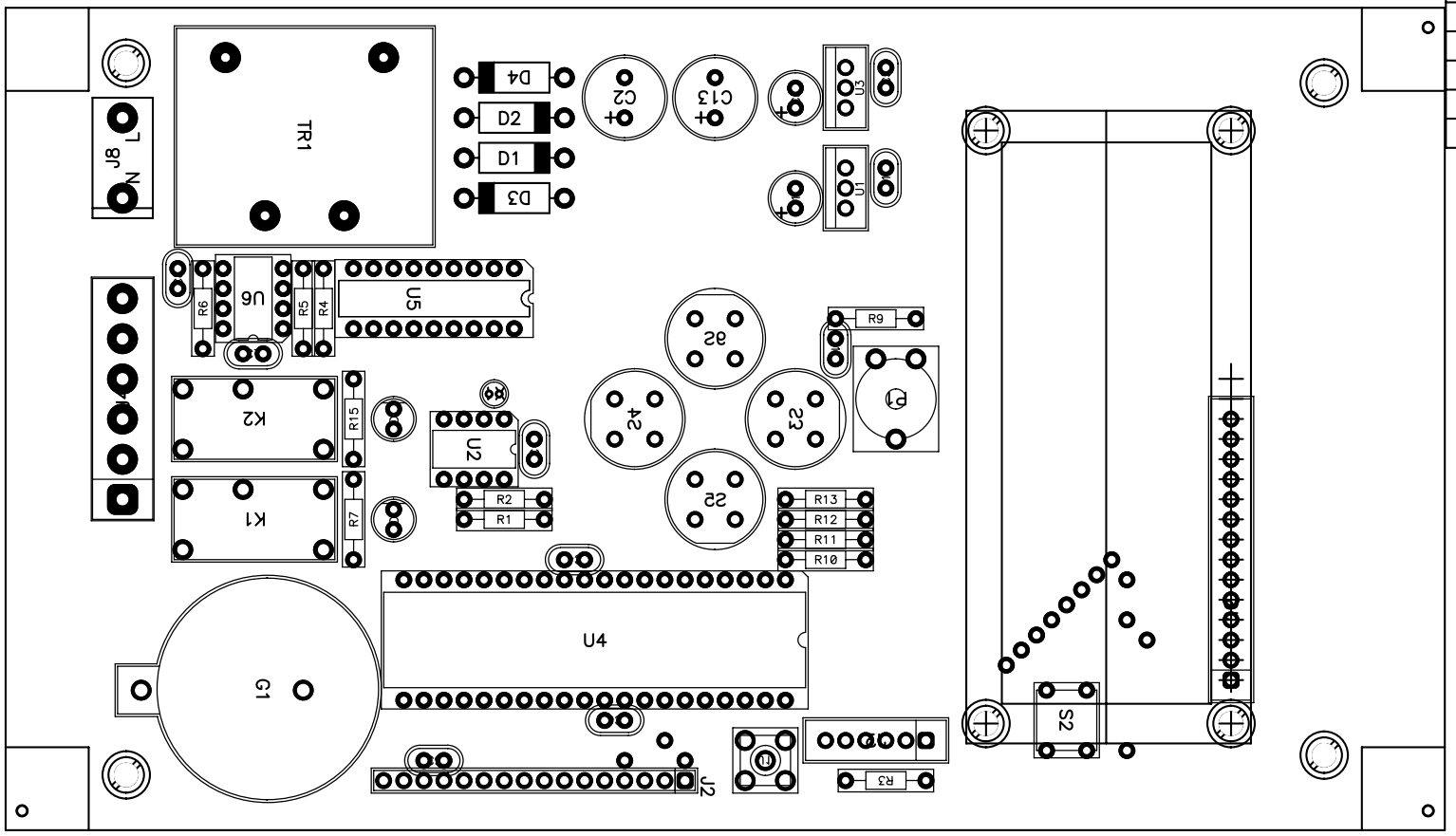


Chauffage par piece Regulateur Sheet2	DES	02.05.2007 Belomatt
	REV	20.05.2007 WAL
HAUTE ECOLE VALAISANNE	2/3	...\TSTD\Belometti Regulateur_ FINAL.SCH

ALIMENTATION 12VDC 5 VDC



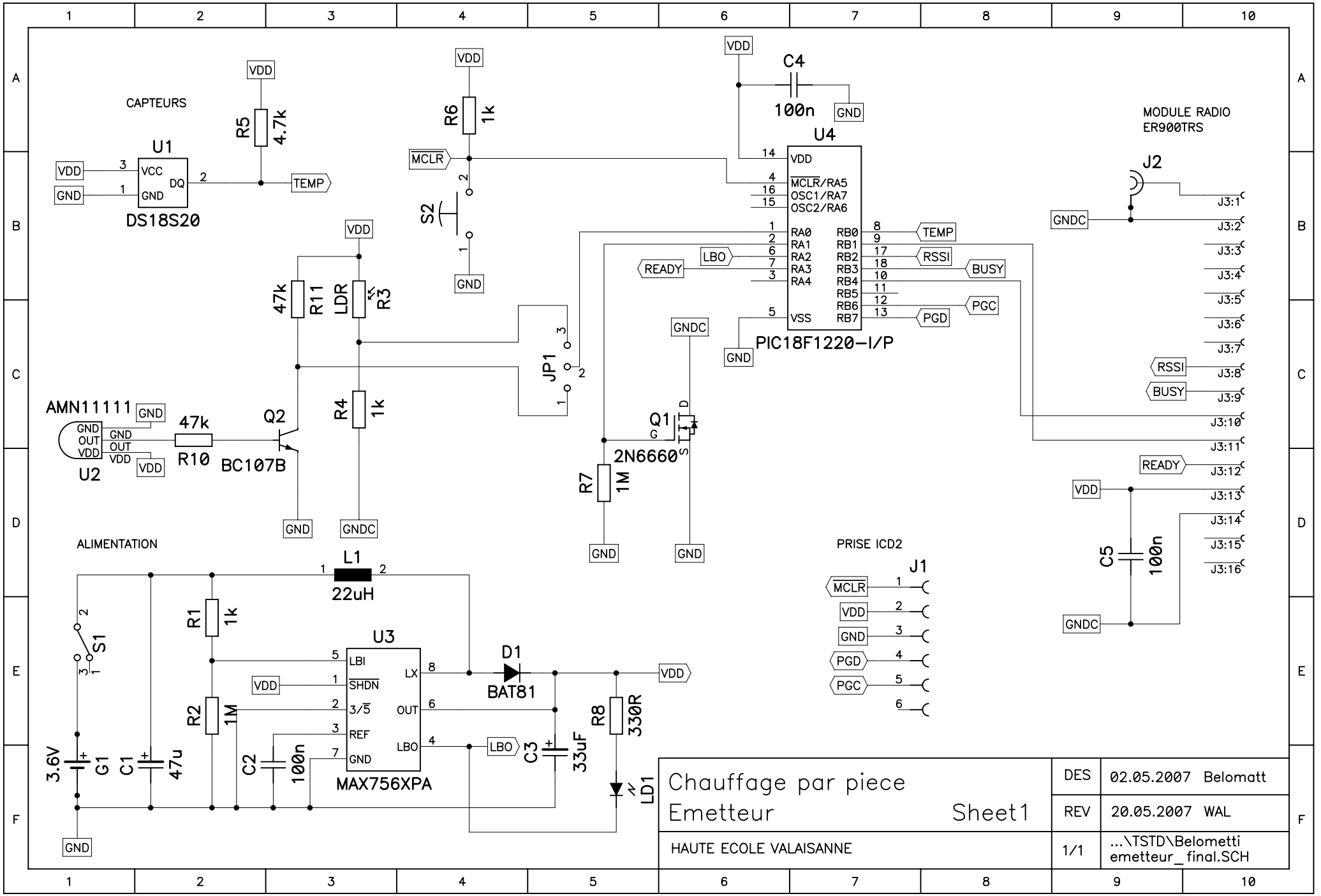
Chauffage par piece Regulateur Sheet3	DES	02.05.2007 Belomatt
	REV	20.05.2007 WAL
HAUTE ECOLE VALAISANNE	3/3	...\TSTD\Belometti Regulateur_ FINAL.SCH

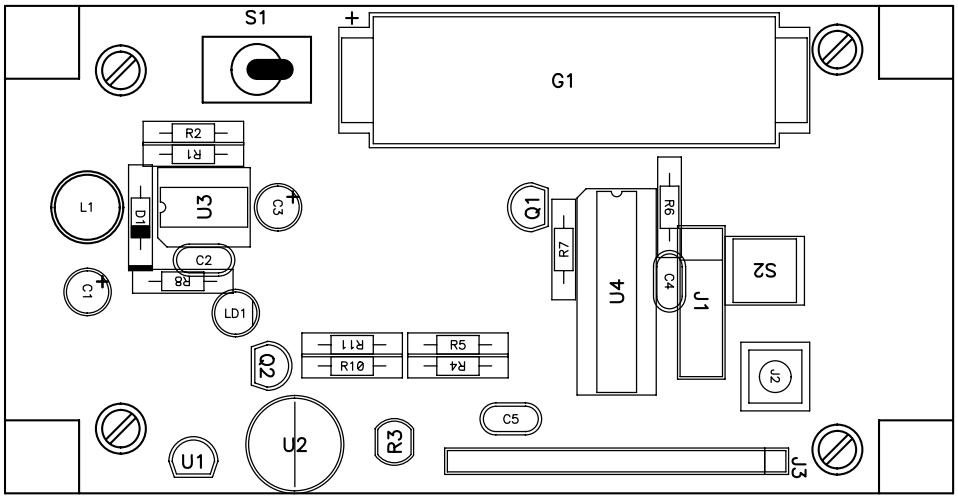


Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.300	+	2	Yes
0.800	×	134	Yes
0.888	Y	3	Yes
0.900	⊤	15	Yes
1.000	⧖	56	Yes
1.100	⋈	6	Yes
1.200	⊡	8	Yes
1.300	◈	13	Yes
1.400	⊠	8	Yes
1.500	◈	7	Yes
3.000	⊠	8	Yes

Chauffage par piece Regulateur	DES	21.11.2007 WAL
	REV	V1.0
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\Belometti Regulateur_ FINAL.pcb	

ANNEXE 2


CAPTEURS
U1
DS18S20
VCC
DQ
GND
R5
4.7k
TEMP
R6
1k
MCLR
S2
C4
100n
U4
PIC18F1220-I/P
RA0
RA1
RA2
RA3
RA4
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
OSC1/RA7
OSC2/RA6
MCLR/RA5
LBO
READY
RSSI
BUSY
PGC
PGD
MODULE RADIO
ER900TRS
J2
J3:1
J3:2
J3:3
J3:4
J3:5
J3:6
J3:7
J3:8
J3:9
J3:10
J3:11
J3:12
J3:13
J3:14
J3:15
J3:16
47k
R11
LDR
R3
JP1
R4
1k
AMN11111
U2
R10
BC107B
Q2
Q1
2N6660
R7
1M
ALIMENTATION
L1
22uH
S1
R1
U3
MAX756XPA
LBI
SHDN
REF
LX
OUT
D1
BAT81
R2
C2
100n
C3
33uF
R8
330R
LD1
3.6V
G1
C1
47u
PRISE ICD2
J1
C5
Chauffage par piece
Emetteur
Sheet1
HAUTE ECOLE VALAISANNE
DES
02.05.2007 Belomatt
REV
20.05.2007 WAL
1/1
...\TSTD\Belometti
emetteur_ final.SCH



Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.800	+	76	Yes
0.888	×	3	Yes
0.900	Y	7	Yes
1.000	⊤	24	Yes
1.500	⧖	5	Yes
1.800	⋈	3	Yes
1.900	⊡	4	Yes
3.000	◈	4	Yes

Chauffage par piece Emetteur	DES	20.11.2007 WAL
	REV	V1.0
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\Belometti emetteur_ final.pcb	

ANNEXE 3

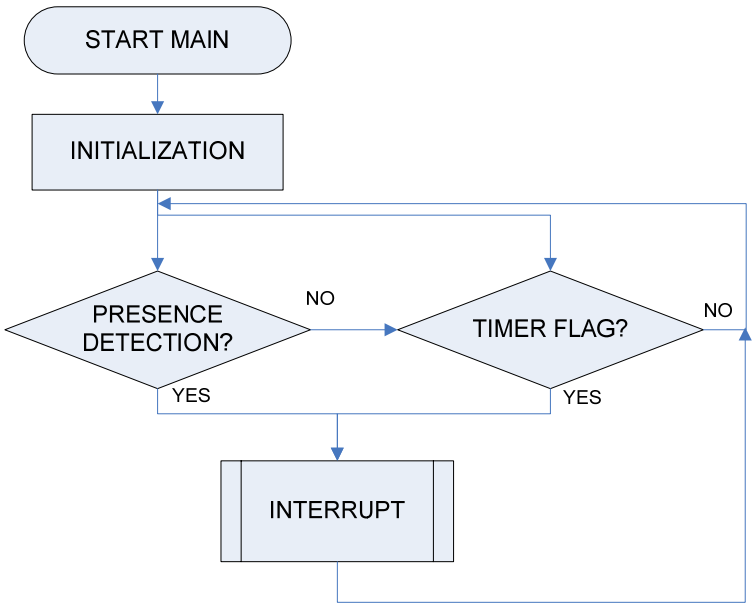
Prix composants cartes

Qté	Description	type	Fournisseur	Num d'article	Prix unitaire	Prix total
2	Capteur temperature	DS18S20	farnell	3330060	SFr. 8.74	SFr. 17.48
1	Photorésistance	NSL4960	farnell	3168335	SFr. 1.90	SFr. 1.90
1	Capteur de présence	AMN11111	farnell	4160253	SFr. 27.50	SFr. 27.50
2	Microcontroleur	PIC18F1220	farnell	9761977	SFr. 9.00	SFr. 18.00
1	Microcontroleur	PIC18F4525	distrelec	64 65 43	SFr. 17.40	SFr. 17.40
2	Elevateur switching	MAX756	distrelec	64 98 88	SFr. 10.65	SFr. 21.30
3	Transceiver RF	ER-TRS900	wirelessworld	-	SFr. 59.90	SFr. 179.70
3	Antenne	ER-ANT-SMA	wirelessworld	-	SFr. 28.10	SFr. 84.30
3	Connecteur SMA antenne	Tyco electronics	farnell	1248990	SFr. 3.64	SFr. 10.92
2	Pile 3.6V Lithium	Sonnenschein	distrelec	97 25 41	SFr. 13.00	SFr. 26.00
4	Porte pile	clips	distrelec	97 01 82	SFr. 1.72	SFr. 6.88
1	Porte pile	K106	distrelec	97 01 70	SFr. 3.77	SFr. 3.77
1	Pile bouton 3V	CR2032	distrelec	97 05 08	SFr. 1.94	SFr. 1.94
2	Mosfet	2N6660	distrelec	61 23 06	SFr. 6.90	SFr. 13.80
1	Transistor	BC107B	distrelec	61 03 02	SFr. 1.29	SFr. 1.29
1	Horloge temps réel	DS1307	farnell	1188042	SFr. 6.70	SFr. 6.70
1	Transformateur 230/12V		distrelec	35 24 68	SFr. 8.93	SFr. 8.93
1	Régulateur de tension 12 V	LM7812	distrelec	64 08 86	SFr. 1.60	SFr. 1.60
1	Régulateur de tension 5 V	LM7805	distrelec	64 08 81	SFr. 1.50	SFr. 1.50
4	Bouton poussoir	TPC11CG	distrelec	20 01 05	SFr. 3.20	SFr. 12.80
1	Ampli op	LM6482	distrelec	64 23 82	SFr. 2.70	SFr. 2.70
2	Relais pour print	série 32	distrelec	40 05 38	SFr. 4.20	SFr. 8.40
1	Quartz 32.768kHz	NTF-3238-E	distrelec	64 48 42	SFr. 2.69	SFr. 2.69
21	Résistance 0.25W	-	distrelec	-	SFr. 0.17	SFr. 3.57
11	Condensateur 100nF	-	distrelec	82 18 46	SFr. 0.50	SFr. 5.50
1	Condensateur 47uF	-	distrelec	80 04 62	SFr. 0.30	SFr. 0.30
2	Bobine 22uH	-	distrelec	35 06 76	SFr. 0.90	SFr. 1.80
3	Condensateur 33uF	-	distrelec	80 04 87	SFr. 0.43	SFr. 1.29
4	Bornier 16 pin	-	distrelec	12 03 24	SFr. 1.40	SFr. 5.60
1	Bornier à vis	-	distrelec	14 03 44	SFr. 3.23	SFr. 3.23
2	Diode Shottky	BAT85	distrelec	60 30 47	SFr. 0.59	SFr. 1.18
4	Diode	1N4007	distrelec	60 00 99	SFr. 0.22	SFr. 0.88
1	Driver pour relais	ULN2803	distrelec	64 69 92	SFr. 2.04	SFr. 2.04
2	Interrupteur	-	distrelec	20 21 62	SFr. 2.50	SFr. 5.00
1	Interrupteur	-	distrelec	20 31 60	SFr. 1.80	SFr. 1.80
1	Prise 230 V	-	distrelec	11 53 06	SFr. 4.09	SFr. 4.09
1	Boitier régulateur	-	distrelec	30 00 92	SFr. 21.30	SFr. 21.30
2	Boitier émetteur	-	distrelec	30 00 88	SFr. 14.50	SFr. 29.00
1	Ecran LCD	W162-N3LED	distrelec	66 13 34	SFr. 30.40	SFr. 30.40
TOTALE						SFr. 594.48

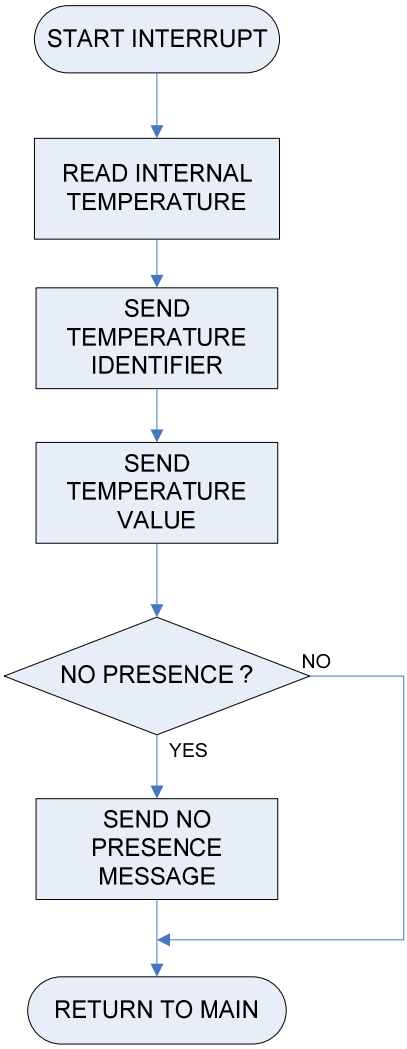
ANNEXE 4

Emetteur interne

Boucle principale



Interruption envoi données



```
1   /*****************************************************************************
2   * FILENAME  :   EMETTEUR INTERNE.c                                           *
3   * FUNCTION  :   PROGRAM FOR RADIO DATA INTERNAL EMITTER:                     *
4   *               READS SENSORS & SENDS DATA (TEMPERATURE & PRESENCE)          *
5   *----------------------------------------------------------------------------*
6   * DATE      :   23.11.07                                                     *
7   * AUTHOR    :   BELOMETTI MATTEO                                             *
8   *----------------------------------------------------------------------------*
9   * PROCESSOR :   PIC18F1220                                                   *
10  * COMPILER  :   HI-TECH PICC18 V                                             *
11  *----------------------------------------------------------------------------*
12  * REVISION  :   1.0 SEPTEMBER 2007                                           *
13  *****************************************************************************/
14
15  #include <pic18.h>
16  #include <stdlib.h>
17  #include <stdio.h>
18  #include <hitech.h>
19  #include <string.h>
20
21  /*****************************************************************************
22   * CONSTANTS DEFINITION
23   *****************************************************************************/
24
25  // PORTS DIRECTION ------------------------------------------------------------
26
27  #define PORTA_DIR       0b11100101
28  #define PORTB_DIR       0b11111111
29  #define DS_DIR          TRISB0
30
31  // I/O DEFINITION -------------------------------------------------------------
32
33  #define PRES_SENSOR     RA0
34  #define SHUTDOWN        RA1
35  #define LOW_BATT        RA2
36  #define RFREADY         RA3
37  #define DS_BUS          RB0
38  #define UART_TX         RB1
39  #define RFBUSY          RB3
40  #define UART_RX         RB4
41
42
43  /*****************************************************************************
44   * GLOBAL VARIABLES DECLARATION
45   *****************************************************************************/
46
47  // TEMPERATURE READING --------------------------------------------------------
48
49
50  int oldTemp=0;              // OLD TEMPERATURE VALUE
51  int tempDiff=0;             // TEMPERATURE ERROR
52  bit DS_DATA;                // DS18S20 DATA
53
54  unsigned char actualTemp=0; // SCRATCHPAD'S NEEDED REGISTERS BYTES
55  unsigned char signe=0;
56  unsigned char remain=0;
57  unsigned char perc=0;
58
59  int temperature=0;
60  char tosend[4];             // FRAMES TO SEND
61  char temperatureData[10];   // DS18S20 SCRATCHPAD VALUES
62
63  // PRESENCE DETECTION ---------------------------------------------------------
64
65  char presence=0;
66  char presence_old=0;
67  char presDetected=0;
68  char sendPresence=0;
69
70  // OTHERS ---------------------------------------------------------------------
71
72  char timerCount=0;          // 4 SECONDS TIMER COUNTER
73  int sendData=1;             // ALLOW TO SEND MESSAGES
74
75  /*****************************************************************************
76   * FORWARD FUNCTIONS DECLARATION
```

```
77   ********************************************************************************/
78
79   void init(void);                // GLOBAL INITIALIZATION
80   void ds_init(void);             // INITIALIZE TEMPERATURE SENSOR
81   void ds_write(char writeData);  // WRITE DATA ON TEMPERATURE SENSOR
82   void readScratch(void);         // READ TEMPERATURE SENSOR REGISTERS
83   void readProbe();               // READ TEMPERATURE VALUE
84   void createFrame(char message); // SEND DATA
85   void interrupt time(void);      // INTERRUPT ON TIMER 0
86   void Delay480(void);            // DELAYS
87   void Delay750(void);
88   void Delay100(void);
89   void Delay3(void);
90
91
92   /*******************************************************************************
93    *  FUNCTION:   INIT()
94    *  GOAL:       This function initializes all needed parameters
95    *  COMMENTS:   clock, uart, ports, timers, interrupts,
96    ********************************************************************************/
97
98   void init()
99   {
100      // CLOCK INIT ------------------------------------------------------------
101
102      IRCF2=1;                // 2 MHz INTERNAL CLOCK SOURCE
103      IRCF1=0;
104      IRCF0=1;
105
106      // PORTS INIT ------------------------------------------------------------
107
108      TRISA=PORTA_DIR;
109      TRISB=PORTA_DIR;
110      SHUTDOWN=1;
111      ADCON1=255;
112
113      // UART INIT -------------------------------------------------------------
114
115      SPBRG=12;               // BAUD RATE CALCULATION
116      BRG16=0;
117      BRGH=0;
118      SYNC=0;
119      PCFG5=1;
120      PCFG6=1;
121      SPEN=1;                 // SERIAL PORT ENABLED
122      TXEN=1;                 // TRANSMISSION ENABLED
123      CREN=0;                 // RECEPTION DISABLED
124
125      // INTERRUPTS INIT -------------------------------------------------------
126
127      GIE = 1;                // GLOBAL INTERRUPT ENABLED
128      PEIE = 1;               // PERIPHERAL INTERRUPT ENABLED
129      TMR0IE=1;               // TIMER0 INTERRUPT ENABLED
130      TMR0IF=1;               // TIMER0 FLAG SET
131
132      // TIMER 0 INIT ----------------------------------------------------------
133
134      T08BIT=0;               // 16 BITS MODE
135      T0CS=0;                 // INTERNAL CLOCK SOURCE
136      PSA=0;                  // PRECALER ASSIGNED
137      T0PS2=1;                // PRESCALER VALUE = 128
138      T0PS1=1;
139      T0PS0=0;
140      TMR0ON=1;               // START TIMER
141      TMR0H=0xC2;             // 4 SECONDS TIMING
142      TMR0L=0xF7;
143  }
144
145  /*******************************************************************************
146   *  FUNCTION:   DELAYS
147   *  GOAL:       delays generation for temperature sensor management
148   *              (1Wire protocol)
149   *  COMMENTS: -
150   ********************************************************************************/
151
152  void Delay480()
153  {
```

```
154     int time=13;
155     while(--time>0)
156         asm("nop");
157 }
158 
159 void Delay750()
160 {
161     int time=20;
162     while(--time>0)
163         asm("nop");
164 }
165 
166 void Delay100()
167 {
168     int time=2;
169     while(--time>0)
170         asm("nop");
171 }
172 
173 void Delay3()
174 {}
175 
176 /*****************************************************************************
177  *  FUNCTION:   1Wire protocol routines
178  *  GOAL:       DS1820 management
179  *  COMMENTS:   see sensor's datasheetfor information about 1Wire protocol
180  *****************************************************************************/
181 
182 // INITIALIZE DS1820 SENSOR ------------------------------------------------
183 
184 void ds_init()
185 {
186     char DS_PRESENCE=0;
187     char DS_ERROR=0;
188     do
189     {
190         DS_BUS=1;                           // PINS SETTING
191         DS_DIR=0;
192         DS_BUS=0;                           // 1) RESET PULSE ON BUS
193         Delay480();
194         DS_BUS=1;
195         DS_DIR=1;                           // 2) WAIT 4 SENSOR ANSWER
196         Delay100();
197         DS_PRESENCE=DS_BUS;
198         if(DS_PRESENCE==1)                  // 3) CHECK SENSOR PRESENCE
199             DS_ERROR=1;
200         else
201             DS_ERROR=0;
202         Delay480();
203     }
204     while(DS_ERROR==1);
205 }
206 
207 // WRITE A BYTE ON DS1820 SENSOR -------------------------------------------
208 
209 void ds_write(char writeData)
210 {
211     int count=8;
212     actualTemp=0;
213     for(count;count>0;count--)          // READ A BYTE
214     {
215         DS_DATA= writeData & 0x01;      // BEGIN WITH LSB
216         DS_DIR=0;
217         DS_BUS=0;                       // START OF WRITE SLOT
218         asm("nop");
219         DS_BUS=DS_DATA;                 // WRITE VALUE ON BUS
220         Delay100();
221         DS_BUS=1;                       // RELEASE BUS, END OF WRITE SLOT
222         writeData=writeData>>1;         // SHIFT DATA TO WRITE NEXT BIT
223         asm("nop");
224     }
225 }
226 
227 // READ DS1820 SCRATCHPAD --------------------------------------------------
228 
229 void readScratch()
230 {
```

```
231     int val=1;
232     int count=8;
233     int i=9;
234
235     ds_write(0xBE);                     // READ SCRATCHPAD COMMAND
236
237     for(i;i>0;i--)                      // BYTE COUNTER
238     {
239         temperatureData[i]=0;
240         for(count;count>0;count--)       // TO READ 8 BITS
241         {
242             DS_DIR=0;
243             DS_BUS=0;                   // START OF READ SLOT
244             asm("nop");
245             DS_DIR=1;                   // RELEASE BUS
246
247             Delay3();                   // GET CLOSE TO CENTER OF TIMESLOT
248             DS_DATA=DS_BUS;             // READ DATA BIT
249
250             temperatureData[i]=temperatureData[i] + DS_DATA*val;
251             val=val*2;
252             asm("nop");
253         }
254         asm("nop");
255         asm("nop");
256     }
257     asm("nop");
258     ds_init();
259
260     actualTemp=temperatureData[9];      // STORE DATA
261     signe=temperatureData[8];
262     remain=temperatureData[3];
263     perc=temperatureData[2];
264 }
265
266 // READ TEMPERATURE ON DS1820 ------------------------------------------
267
268 void readProbe()
269 {
270     ds_init();
271     ds_write(0xCC);                     // SKIP ROM
272     ds_write(0x44);                     // TEMPERATURE CONVERSION
273     Delay750();                         // TIME TO CONVERT TEMPERATURE
274     ds_init();
275     ds_write(0xCC);                     // SKIP ROM
276     readScratch();                      // READ TEMPERATURE VALUE
277 }
278
279 /*************************************************************************
280  *  FUNCTION:   Time interrupt
281  *  GOAL:       This function generates an interrupt every 4 seconds
282  *  COMMENTS:   15 times to have a minute
283  *************************************************************************/
284
285 void interrupt time(void)
286 {
287     if(TMR0IF == 1)
288     {
289         TMR0H=0xC2;                     // RESTART TIMER 0
290         TMR0L=0xF7;
291         TMR0IF=0;
292         timerCount++;
293         if(timerCount==15)              // 1 MINUTE TIMER
294         {
295             sendData=1;                 // ENABLE DATA ACQUISITION
296             timerCount=0;               // RESET COUNTER
297         }
298     }
299 }
300
301 /*************************************************************************
302  *  FUNCTION:   CREATEFRAME()
303  *  GOAL:       This function puts messages to send on UART transmission
304  *              register
305  *  COMMENTS:   -
306  *************************************************************************/
307
```

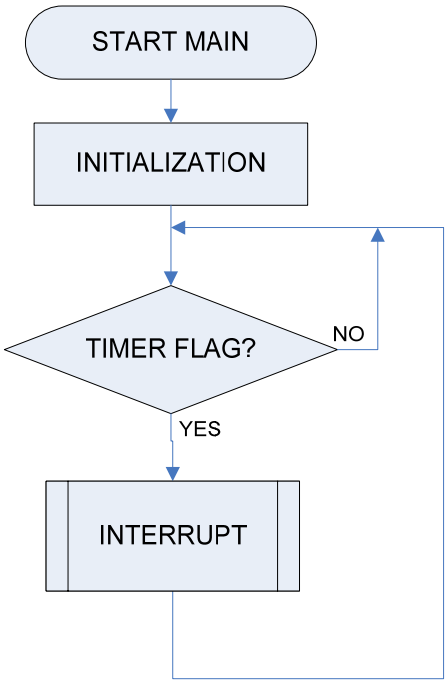
```
308 void createFrame(char message)
309 {
310     while(RFBUSY == 1);                 // WAIT RF MODULE TO BE READY
311     while(TRMT == 0);                   // WAIT UART TO BE READY
312     TXREG=message;                      // PUT MESSAGE ON TX REGISTER
313     while(RFBUSY == 1);                 // WAIT END OF TRANSMISSION
314 }
315
316 /**************************************************************************
317  *  FUNCTION:    MAIN()
318  *  GOAL:        This function is the main program: calls initialisation and
319  *               make an infinity loop
320  *  COMMENTS:    -
321  **************************************************************************/
322
323 void main()
324 {
325     int i;
326     init();                                 // GLOBAL INITIALIZATION
327
328     while(1)
329     {
330         presence=PRES_SENSOR;               // CHECK PRESENCE STATUS
331
332         if(LOW_BATT==0)                     // BATTERY VOLTAGE LOW
333         {
334             SHUTDOWN=1;                     // TURN ON RADIO MODULE
335             createFrame('B');
336             SHUTDOWN=0;                     // TURN OFF RADIO MODULE
337         }
338
339         if(presence!=presence_old && presence==0)// EDGE DETECTION
340         {
341             if(presDetected==0)             // FIRST EDGE
342             {
343                 presDetected=1;
344                 sendPresence=1;
345             }
346         }
347         if(sendData==1 || sendPresence)     // EACH MINUTE SEND DATA
348         {
349             SHUTDOWN=1;                     // TURN ON RADIO MODULE
350             RFREADY=0;                      // PIC IS READY
351
352             GIE=0;                          // DISABLE INTERRUPTS
353
354             do
355             {
356                 readProbe();                // READ TEMPERATURE
357                 if(actualTemp-oldTemp>0)    // CALCULATE DIFF BETWEEN THE LASTS MEASUREMENTS

358                     tempDiff=actualTemp-oldTemp;
359                 else
360                     tempDiff=oldTemp-actualTemp;
361                 oldTemp=actualTemp;
362             }
363             while(tempDiff!=0 || actualTemp==170);  // CHECK IF AN ERROR OR INIT OCCURRED
364
365             actualTemp=actualTemp>>1;       // TEMPERATURE HIGH RESOLUTION CALCULATION
366
    temperature=100*((float)actualTemp-0.25+(((float)perc)-((float)remain))/((float)perc));
367
368             if(signe==255)                  // IF TEMP<0, SEND 0
369                 temperature=0.0;
370
371             tosend[0]='t';                  // EXTERNAL TEMPERATURE ID
372             tosend[1]=(char)temperature;    // TEMPERATURE DATA MSB
373             tosend[2]=temperature>>8;       // TEMPERATURE DATA LSB
374
375             if(sendPresence==1)
376             {
377                 tosend[3]='P';              // SEND PRESENCE MESSAGE
378                 sendPresence=0;
379             }
380             if(sendData==1)
381             {
382                 if(presDetected==0)
```

```
383                     tosend[3]='p';          // SEND NO PRESENCE MESSAGE
384                 else
385                 {
386                     tosend[3]='P';          // SEND PRESENCE MESSAGE
387                     presDetected=0;
388                 }
389                 sendData=0;
390             }
391
392             for(i=0;i<4;i++)                // SEND MESSAGES
393                 createFrame(tosend[i]);
394
395             SHUTDOWN=0;                     // TURN OFF RADIO MODULE
396             GIE=1;                          // ENABLE INTERRUPTS
397         }
398     }
399 }
400
```

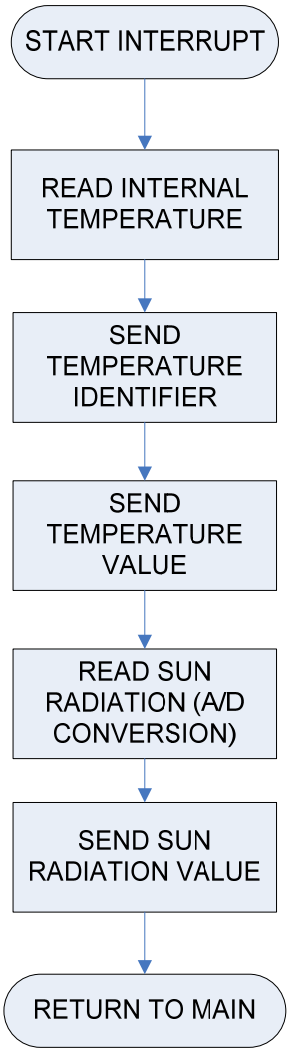
ANNEXE 5

Emetteur externe

Boucle principale



Interruption envoi données



```
1   /*****************************************************************************
2   * FILENAME  :   EMETTEUR EXTERNE.c                                           *
3   * FUNCTION  :   PROGRAM FOR RADIO DATA EXTERNAL EMITTER:                     *
4   *               READS SENSORS & SENDS DATA (TEMPERATURE & SUN RADIATION)     *
5   *----------------------------------------------------------------------------*
6   * DATE      :   23.11.07                                                     *
7   * AUTHOR    :   BELOMETTI MATTEO                                             *
8   *----------------------------------------------------------------------------*
9   * PROCESSOR :   PIC18F1220                                                   *
10  * COMPILER  :   HI-TECH PICC18 V                                             *
11  *----------------------------------------------------------------------------*
12  * REVISION  :   1.0 SEPTEMBER 2007                                           *
13  *****************************************************************************/
14
15  #include <pic18.h>
16  #include <stdlib.h>
17  #include <stdio.h>
18  #include <hitech.h>
19  #include <string.h>
20
21  /*****************************************************************************
22   * CONSTANTS DEFINITION
23   *****************************************************************************/
24
25  // PORTS DIRECTION ------------------------------------------------------------
26
27  #define PORTA_DIR       0b11100101
28  #define PORTB_DIR       0b11111111
29  #define DS_DIR          TRISB0
30
31  // I/O DEFINITION -------------------------------------------------------------
32
33  #define SUN_SENSOR      RA0
34  #define SHUTDOWN        RA1
35  #define LOW_BATT        RA2
36  #define RFREADY         RA3
37  #define DS_BUS          RB0
38  #define UART_TX         RB1
39  #define RFBUSY          RB3
40  #define UART_RX         RB4
41
42
43  /*****************************************************************************
44   * GLOBAL VARIABLES DECLARATION
45   *****************************************************************************/
46
47  // TEMPERATURE READING --------------------------------------------------------
48
49
50  int oldTemp=0;              // OLD TEMPERATURE VALUE
51  int tempDiff=0;             // TEMPERATURE ERROR
52  bit DS_DATA;                // DS18S20 DATA
53
54  unsigned char actualTemp=0; // SCRATCHPAD'S NEEDED REGISTERS BYTES
55  unsigned char signe=0;
56  unsigned char remain=0;
57  unsigned char perc=0;
58
59  int temperature=0;
60  char tosend[4];             // FRAMES TO SEND
61  char temperatureData[10];   // DS18S20 SCRATCHPAD VALUES
62
63  // SUN MEASUREMENT ------------------------------------------------------------
64
65  int mesure=0;               // TO CALCULATE SUN RADIATION VALUE
66  int mesureH=0;              // SUN RADIATION HIGH VALUE
67  int mesureL=0;              // SUN RADIATION LOW VALUE
68
69  // OTHERS ---------------------------------------------------------------------
70
71  char timerCount=0;              // 4 SECONDS TIMER COUNTER
72  int sendData=1;                 // ALLOW TO SEND MESSAGES
73
74  /*****************************************************************************
75   * FORWARD FUNCTIONS DECLARATION
76   *****************************************************************************/
```

```
77
78  void init(void);                // GLOBAL INITIALIZATION
79  void ds_init(void);             // INITIALIZE TEMPERATURE SENSOR
80  void ds_write(char writeData);  // WRITE DATA ON TEMPERATURE SENSOR
81  void readScratch(void);         // READ TEMPERATURE SENSOR REGISTERS
82  void readProbe();               // READ TEMPERATURE VALUE
83  void createFrame(void);         // SEND DATA
84  void adconv(void);              // AD CONVERSION, SUN RADIATION VALUE
85  void interrupt time(void);      // INTERRUPT ON TIMER 0
86  void Delay480(void);            // DELAYS
87  void Delay750(void);
88  void Delay100(void);
89  void Delay3(void);
90
91
92  /**************************************************************************
93   *  FUNCTION:   INIT()
94   *  GOAL:       This function initializes all needed parameters
95   *  COMMENTS:   clock, uart, ports, timers, interrupts,
96   **************************************************************************/
97
98  void init()
99  {
100     // CLOCK INIT ------------------------------------------------------
101
102     IRCF2=1;                // 2 MHz INTERNAL CLOCK SOURCE
103     IRCF1=0;
104     IRCF0=1;
105
106     // PORTS INIT ------------------------------------------------------
107
108     TRISA=PORTA_DIR;
109     TRISB=PORTA_DIR;
110     SHUTDOWN=1;
111     RA4=0;
112
113     // UART INIT -------------------------------------------------------
114
115     SPBRG=12;               // BAUD RATE CALCULATION
116     BRG16=0;
117     BRGH=0;
118     SYNC=0;
119     PCFG5=1;
120     PCFG6=1;
121     SPEN=1;                 // SERIAL PORT ENABLED
122     TXEN=1;                 // TRANSMISSION ENABLED
123     CREN=0;                 // RECEPTION DISABLED
124
125     // INTERRUPTS INIT -------------------------------------------------
126
127     GIE = 1;                // GLOBAL INTERRUPT ENABLED
128     PEIE = 1;               // PERIPHERAL INTERRUPT ENABLED
129     TMR0IE=1;               // TIMER0 INTERRUPT ENABLED
130     TMR0IF=1;               // TIMER0 FLAG SET
131
132     // TIMER 0 INIT ----------------------------------------------------
133
134     T08BIT=0;               // 16 BITS MODE
135     T0CS=0;                 // INTERNAL CLOCK SOURCE
136     PSA=0;                  // PRECALER ASSIGNED
137     T0PS2=1;                // PRESCALER VALUE = 128
138     T0PS1=1;
139     T0PS0=0;
140     TMR0ON=1;               // START TIMER
141     TMR0H=0xC2;             // 4 SECONDS TIMING
142     TMR0L=0xF7;
143
144
145     //AD CONVERSION INIT ----------------------------------------------
146
147     VCFG1=0;                // VOLTAGE REFERENCE VDD TO VSS
148     VCFG0=0;
149     CHS2=0;                 // AD CHANNEL SELECTION
150     CHS1=0;
151     CHS0=0;
152     PCFG6=1;                // DIGITAL I/O SELECTION
153     PCFG5=1;
```

```
154     PCFG4=1;
155     PCFG3=1;
156     PCFG2=1;
157     PCFG1=1;
158     PCFG0=0;
159     ADRESL=0;               // REGISTERS INIT
160     ADRESH=0;
161     ADFM=1;
162     ACQT2=1;                // ACQUISITION TIME
163     ACQT1=1;
164     ACQT0=1;
165     ADCS2=1;                // CLOCK SOURCE
166     ADCS1=1;
167     ADCS0=0;
168     ADON=1;                 // TURN ON AD CONVERSION MODULE
169
170 }
171
172 /**************************************************************************
173  *  FUNCTION:   DELAYS
174  *  GOAL:       delays generation for temperature sensor management
175  *              (1Wire protocol)
176  *  COMMENTS: -
177  **************************************************************************/
178
179 void Delay480()
180 {
181     int time=13;
182     while(--time>0)
183         asm("nop");
184 }
185
186 void Delay750()
187 {
188     int time=20;
189     while(--time>0)
190         asm("nop");
191 }
192
193 void Delay100()
194 {
195     int time=2;
196     while(--time>0)
197         asm("nop");
198 }
199
200 void Delay3()
201 {}
202
203 /**************************************************************************
204  *  FUNCTION:   1Wire protocol routines
205  *  GOAL:       DS1820 management
206  *  COMMENTS:   see sensor's datasheetfor information about 1Wire protocol
207  **************************************************************************/
208
209 // INITIALIZE DS1820 SENSOR ------------------------------------------------
210
211 void ds_init()
212 {
213     char DS_PRESENCE=0;
214     char DS_ERROR=0;
215
216         DS_BUS=1;                           // PINS SETTING
217         DS_DIR=0;
218         DS_BUS=0;                           // 1) RESET PULSE ON BUS
219         Delay480();
220         DS_BUS=1;
221         DS_DIR=1;                           // 2) WAIT 4 SENSOR ANSWER
222         Delay100();
223         DS_PRESENCE=DS_BUS;
224         if(DS_PRESENCE==1)                  // 3) CHECK SENSOR PRESENCE
225             DS_ERROR=1;
226         else
227             DS_ERROR=0;
228         Delay480();
229 }
230
```

```
231 // WRITE A BYTE ON DS1820 SENSOR ------------------------------------------
232
233 void ds_write(char writeData)
234 {
235     int count=8;
236     actualTemp=0;
237     for(count;count>0;count--)          // READ A BYTE
238     {
239         DS_DATA= writeData & 0x01;      // BEGIN WITH LSB
240         DS_DIR=0;
241         DS_BUS=0;                       // START OF WRITE SLOT
242         asm("nop");
243         DS_BUS=DS_DATA;                 // WRITE VALUE ON BUS
244         Delay100();
245         DS_BUS=1;                       // RELEASE BUS, END OF WRITE SLOT
246         writeData=writeData>>1;         // SHIFT DATA TO WRITE NEXT BIT
247         asm("nop");
248     }
249 }
250
251 // READ DS1820 SCRATCHPAD -------------------------------------------------
252
253 void readScratch()
254 {
255     int val=1;
256     int count=8;
257     int i=9;
258
259     ds_write(0xBE);                     // READ SCRATCHPAD COMAMND
260
261     for(i;i>0;i--)                      // BYTE COUNTER
262     {
263         temperatureData[i]=0;
264         for(count;count>0;count--)       // TO READ 8 BITS
265         {
266             DS_DIR=0;
267             DS_BUS=0;                   // START OF READ SLOT
268             asm("nop");
269             DS_DIR=1;                   // RELEASE BUS
270
271             Delay3();                   // GET CLOSE TO CENTER OF TIMESLOT
272             DS_DATA=DS_BUS;             // READ DATA BIT
273
274             temperatureData[i]=temperatureData[i] + DS_DATA*val;
275             val=val*2;
276             asm("nop");
277         }
278         asm("nop");
279         asm("nop");
280     }
281     asm("nop");
282     ds_init();
283
284     actualTemp=temperatureData[9];      // STORE DATA
285     signe=temperatureData[8];
286     remain=temperatureData[3];
287     perc=temperatureData[2];
288 }
289
290 // READ TEMPERATURE ON DS1820 ---------------------------------------------
291
292 void readProbe()
293 {
294     ds_init();
295     ds_write(0xCC);                     // SKIP ROM
296     ds_write(0x44);                     // TEMPERATURE CONVERSION
297     Delay750();                         // TIME TO CONVERT TEMPERATURE
298     ds_init();
299     ds_write(0xCC);                     // SKIP ROM
300     readScratch();                      // READ TEMPERTATURE VALUE
301 }
302
303 /*****************************************************************************
304  *  FUNCTION:   Time interrupt
305  *  GOAL:       This function generates an interrupt every 4 seconds
306  *  COMMENTS:   15 times to have a minute
307  *****************************************************************************/
```

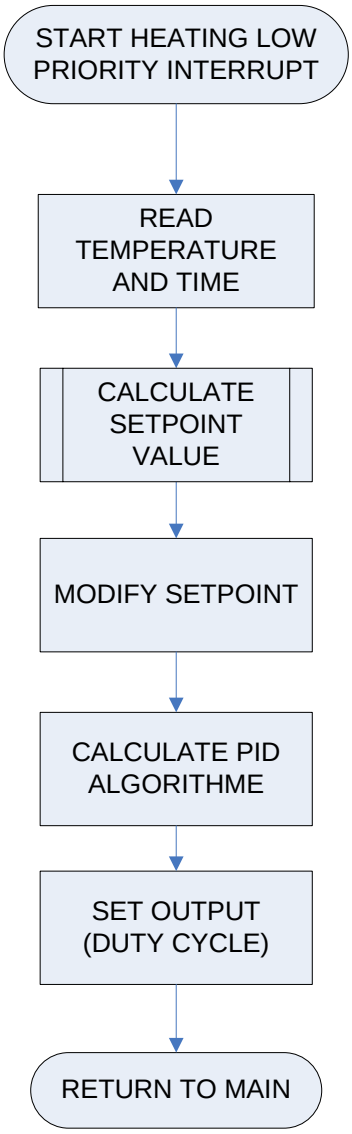
```
308
309 void interrupt time(void)
310 {
311     if(TMR0IF == 1)
312     {
313         TMR0H=0xC2;                     // RESTART TIMER 0
314         TMR0L=0xF7;
315         TMR0IF=0;
316         timerCount++;
317         if(timerCount==15)              // 1 MINUTE TIMER
318         {
319             sendData=1;                     // ENABLE DATA ACQUISITION
320             timerCount=0;               // RESET COUNTER
321         }
322     }
323 }
324
325 /*************************************************************************
326  *  FUNCTION:   CREATEFRAME()
327  *  GOAL:       This function puts messages to send on UART transmission
328  *              register
329  *  COMMENTS:   -
330  *************************************************************************/
331
332 void createFrame(char message)
333 {
334     while(RFBUSY == 1);                 // WAIT RF MODULE TO BE READY
335     while(TRMT == 0);                   // WAIT UART TO BE READY
336     TXREG=message;                      // PUT MESSAGE ON TX REGISTER
337     while(RFBUSY == 1);                 // WAIT END OF TRANSMISSION
338 }
339
340 /*************************************************************************
341  *  FUNCTION:   ADCONV()
342  *  GOAL:       This function makes AD converion of sun radiation value
343  *  COMMENTS:   -
344  *************************************************************************/
345
346 void adconv()
347 {
348     GODONE=1;                           // START AD CONVERSION
349     while(GODONE==1){}                  // WAIT 4 END OF CONVERSION
350     mesureL=ADRESL;                     // READ VALUES
351     mesureH=ADRESH;
352 }
353
354 /*************************************************************************
355  *  FUNCTION:   MAIN()
356  *  GOAL:       This function is the main program: calls initialisation and
357  *              make an infinity loop
358  *  COMMENTS:   -
359  *************************************************************************/
360
361 void main()
362 {
363     int i;
364     init();                             // GLOBAL INITIALIZATION
365
366     while(1)
367     {
368         if(LOW_BATT==0)                 // BATTERY VOLTAGE LOW
369         {
370             createFrame('B');
371         }
372
373         if(sendData==1)                 // EACH MINUTE SEND DATA
374         {
375             SHUTDOWN=1;                 // TURN ON RADIO MODULE
376             RFREADY=0;                  // PIC IS READY
377             GIE=0;                      // DISABLE INTERRUPTS
378
379             do
380             {
381                 readProbe();                // READ TEMPERATURE
382                 if(actualTemp-oldTemp>0)    // CALCULATE DIFF BETWEEN THE LASTS MEASUREMENTS
383                     tempDiff=actualTemp-oldTemp;
```

```
384                 else
385                     tempDiff=oldTemp-actualTemp;
386                 oldTemp=actualTemp;
387             }
388             while(tempDiff!=0 || actualTemp==170);  // CHECK IF AN ERROR OR INIT OCCURRED
389 
390             actualTemp=actualTemp>>1;       // TEMPERATURE HIGH RESOLUTION CALCULATION
391 
    temperature=100*((float)actualTemp-0.25+(((float)perc)-((float)remain))/((float)perc));
392 
393             if(signe==255)                  // IF TEMP<0, SEND 0
394                 temperature=0.0;
395 
396             tosend[0]='T';                  // EXTERNAL TEMPERATURE ID
397             tosend[1]=(char)temperature;    // TEMPERATURE DATA MSB
398             tosend[2]=temperature>>8;       // TEMPERATURE DATA LSB
399 
400             adconv();                       // DO AD CONVERSION
401             mesureH=mesureH<<8;
402             mesure=(mesureH+mesureL);       // CALCULATE REAL SUN VALUE
403 
404             if(mesure>750)                  // COMPARE WITH REFERENCE VALUE
405             {
406                 tosend[3]='S';              // SUN PRESENCE MESSAGE
407             }
408             else
409             {
410                 tosend[3]='s';              // NO SUN MESSAGE
411             }
412 
413             for(i=0;i<4;i++)                // SEND MESSAGES
414                 createFrame(tosend[i]);
415 
416             sendData=0;
417             SHUTDOWN=0;                 // TURN OFF RADIO MODULE
418             GIE=1;                      // ENABLE INTERRUPTS
419         }
420     }
421 }
422 
```

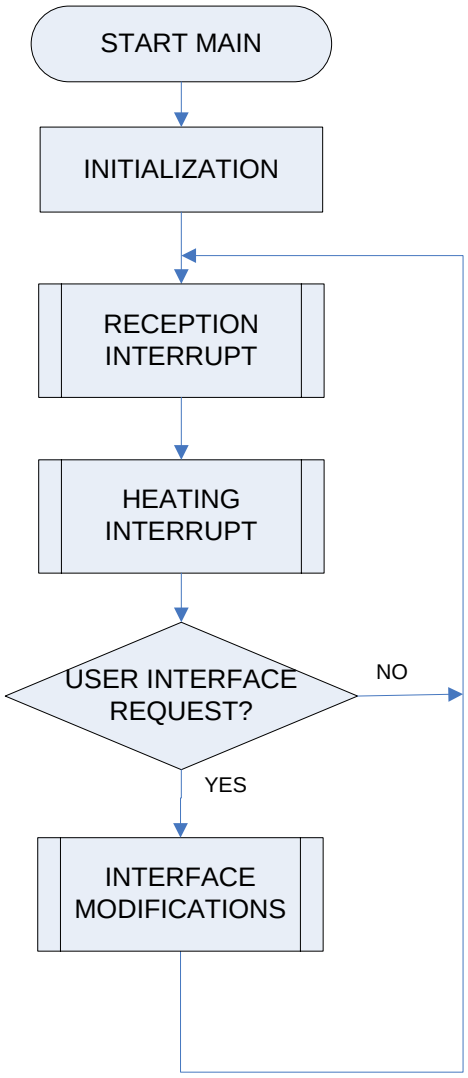
ANNEXE 6

Régulateur

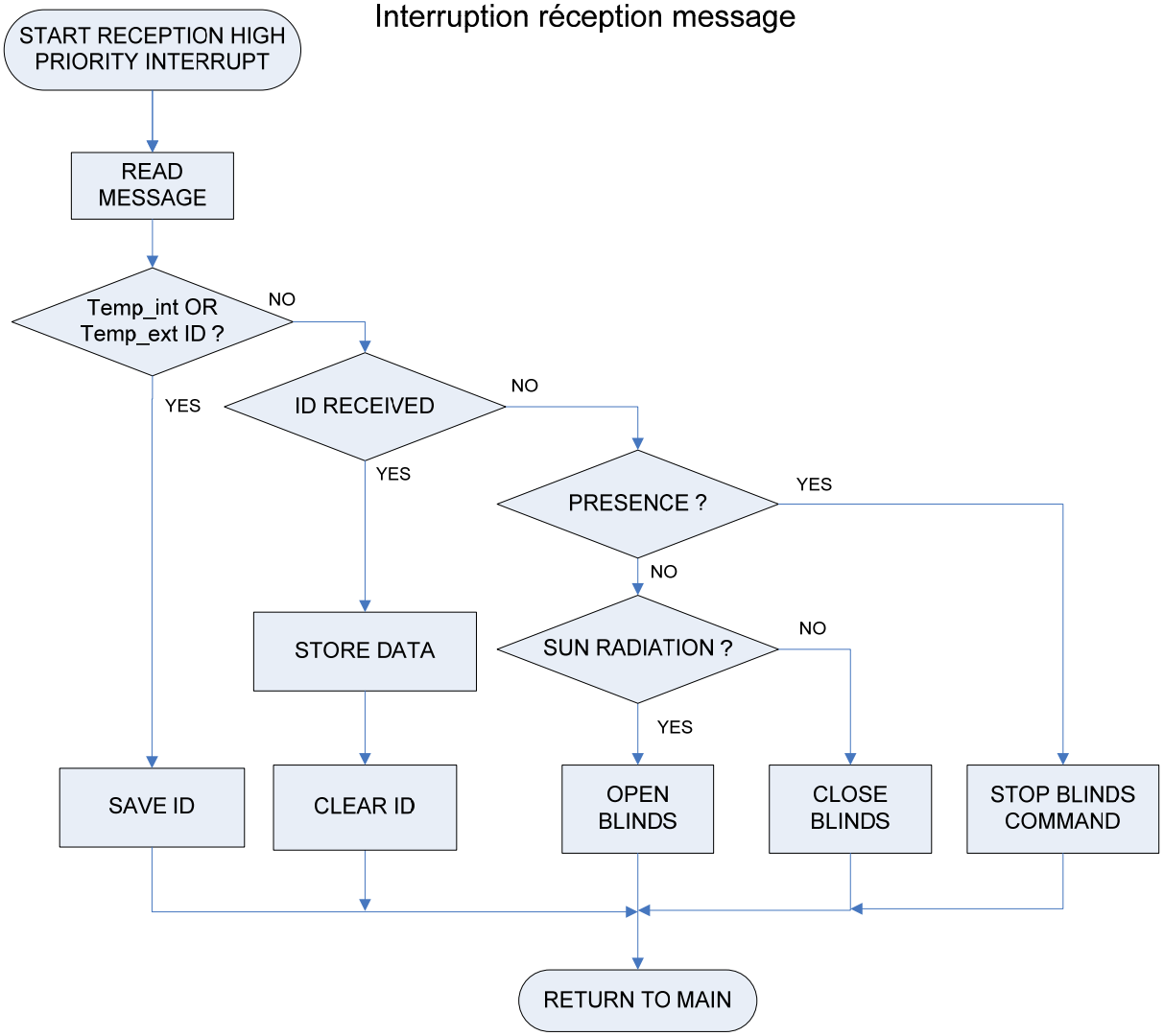
Interruption régulation



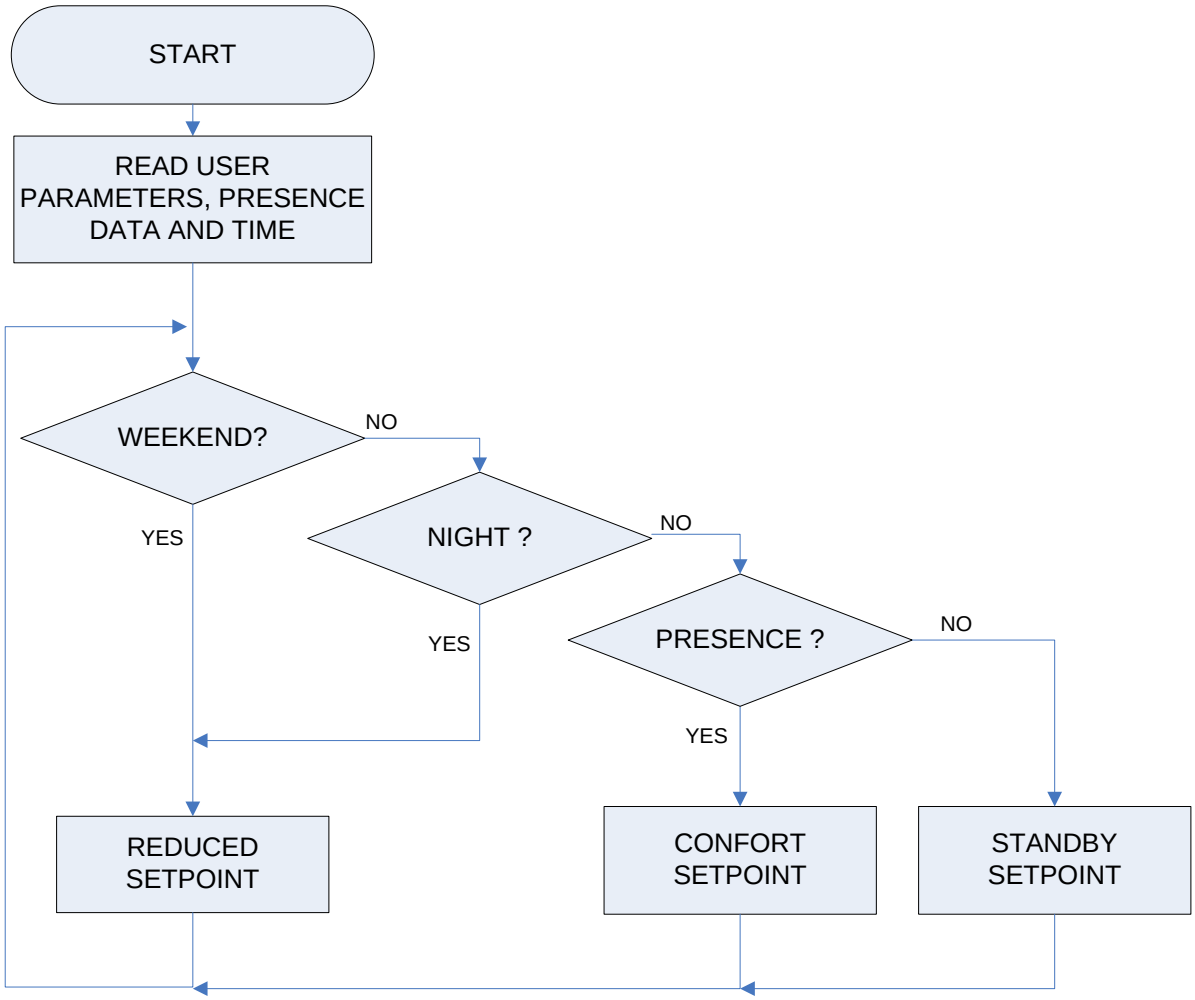
Programme principal



Interruption réception message



Calcul consigne



```
1   /*************************************************************************
2   * FILENAME  :   REGULATEUR.c                                             *
3   * FUNCTION  :   HEATING REGULATION AND BLINDS COMMAND                    *
4   *------------------------------------------------------------------------*
5   * DATE      :   23.11.07                                                 *
6   * AUTHOR    :   BELOMETTI MATTEO                                         *
7   *------------------------------------------------------------------------*
8   * PROCESSOR :   PIC18F4525                                               *
9   * COMPILER  :   HI-TECH PICC18 V                                         *
10  *------------------------------------------------------------------------*
11  * REVISION  :   1.0 november 2007                                        *
12  *************************************************************************/
13
14  #include <pic18.h>
15  #include <stdlib.h>
16  #include <stdio.h>
17  #include <hitech.h>
18  #include <string.h>
19  #include "xlcd.h"
20
21  /*************************************************************************
22   * CONSTANTS DEFINITION
23   *************************************************************************/
24
25  // CONTROL BYTES FOR RTC DS1307 WRITING ------------------------------------
26  #define CLOCK_DS1307_W  0xD0
27  #define CLOCK_DS1307_R  0xD1
28
29  // PORTS DIRECTION ---------------------------------------------------------
30  #define PORTA_DIR 0b11101111
31  #define PORTB_DIR 0b11000110
32  #define PORTC_DIR 0b11111011
33  #define PORTD_DIR 0b00000000
34
35  // I/O DEFINITION ----------------------------------------------------------
36  #define UP_BUTTON       RA0
37  #define DOWN_BUTTON     RA1
38  #define LEFT_BUTTON     RA2
39  #define RIGHT_BUTTON    RA3
40  #define RELAIS_UP       RA4
41  #define RELAIS_DOWN     RB0
42  #define VANNE           RC2
43  #define UART_TX         RA6
44  #define UART_RX         RA7
45
46  /*************************************************************************
47   * STRING TABLES DECLARATION
48   *************************************************************************/
49
50  char men[16];
51  char param[16];
52  char data1[16];
53  char data2[16];
54
55  /*************************************************************************
56   * GLOBAL VARIABLES DECLARATION
57   *************************************************************************/
58
59  // REAL TIME CLOCK INITIALISATION ------------------------------------------
60
61  unsigned char Jour = 1;
62  unsigned char Mois = 10;
63  unsigned char Annee = 7;
64  unsigned char Heure = 10;
65  unsigned char Minute = 44;
66
67
68  // DISPLAY MANAGEMENT ------------------------------------------------------
69
70  static bit up_actual;       // BUTTONS ACTUALS AND OLD VALUES
71  static bit up_old;
72  static bit down_actual;
73  static bit down_old;
74  static bit left_actual;
75  static bit left_old;
```

```
76  static bit right_actual;
77  static bit right_old;
78
79  char cons_norm=22;          // SETPOINTS DEFAULT VALUES
80  char cons_standby=21;
81  char cons_red=18;
82  char start_time=6;
83  char stop_time=16;
84  char menu=0;
85  char updateVis=0;
86  int soleil=0;               // DISPLAY DATA
87  int presence=1;
88
89  // SENSORS DATA ------------------------------------------------------------
90
91  float tempExt=0;
92  float tempInt=0;
93  int presenceCounter=0;
94  int manualMode=0;
95  char tempExt_L=0;
96  char tempExt_H=0;
97  char tempInt_L=0;
98  char tempInt_H=0;
99
100 // BLINDS COMMAND ----------------------------------------------------------
101
102 int storesUp = 0;
103 int storesDown = 0;
104 bit up = 0;                     // BLINDS OPEN FLAG
105 bit down = 0;                   // BLINDS CLOSE FLAG
106 bit dir_up = 0;                 // OPENING BLINDS
107 bit dir_down = 0;               // CLOSING BLINDS
108
109 // MESSAGES RECEPTION MANAGEMENT -------------------------------------------
110
111 int tint=0;
112 int text=0;
113 int tintCount=0;
114 int textCount=0;
115 unsigned char receivedMessage = 0;
116
117 // PID REGULATOR PARAMETERS AND DATA ---------------------------------------
118
119 int count0=0;                   // TIMER 0 COUNTER TO MAKE 5 MINUTES
120 float erreur = 0;               // ERROR
121 float consignePID = 0;          // SETPOINT
122 double u = 0;                  // COMMAND SIGNAL
123 double u_prim = 0;             // VARIABLES FOR ANTIRESET WINDUP
124 float ui = 0;
125 float erreur_old = 0;
126 float erreur_lim = 0;
127 float Kp = 100;                // PID GAINS
128 float Ki = 0.05;
129 float Kd = 100;
130 int duty = 0;                  // DUTY CYCLE OF COMMAND SIGNAL
131
132
133 /*****************************************************************************
134  * FORWARD FUNCTIONS DECLARATION
135  *****************************************************************************/
136
137 void i2cInit(void);                     // I2C BUS FUNCTIONS
138 void i2cWaitForIdle(void);
139 void i2cStart(void);
140 void i2cRepStart(void);
141 void i2cStop(void);
142 void DelayMs(unsigned char cpt);
143 unsigned char i2cRead(unsigned char ack);
144 unsigned char i2cWrite(unsigned char data);
145 void WriteRTC(void) ;                   // RTC FUNCTIONS
146 void ReadRTC(void) ;
147 void PID(void);                         // PID ALGORITHM
148 void affichage_base(void);              // STANDARD DISPLAYED MESSAGE
149 void affichage_menu(void);              // MENU DATA DISPLAYED
150 void consigne(void);                    // SETPOINT CHOICE
151 void low_priority interrupt time(void); // LOW PRIORITY INTERRUPTS
152 void interrupt reception(void);         // HIGH PRIORITY INTERRUPT
```

```
153
154 /*************************************************************************
155  *  FUNCTION: XLCD DELAYS
156  *  GOAL: delays generation for lcd management
157  *  COMMENTS: -
158  *************************************************************************/
159
160 void XLCDDelay15ms (void)
161 {
162     int i;
163     for(i=0;i<20000;i++)
164     {
165     asm("nop");
166     }
167     return;
168 }
169 void XLCDDelay4ms (void)
170 {
171     int i;
172     for(i=0;i<2500;i++)
173     {
174     asm("nop");
175     }
176     return;
177 }
178 void XLCD_Delay500ns(void)
179 {
180     asm("nop");
181     asm("nop");
182     asm("nop");
183     return;
184 }
185 void XLCDDelay(void)
186 {
187     int i;
188     for(i=0;i<1000;i++)
189         {
190         asm("nop");
191         }
192     return;
193 }
194 void DelayMs(unsigned char cpt)
195 {
196     unsigned short t ;
197     while (cpt != 0)
198     {
199         -- cpt ;
200         t = 450 ;
201         while (t !=0)   -- t ;
202     }
203 }
204
205 /*************************************************************************
206  *  FUNCTION: INIT()
207  *  GOAL: This function initializes all needed parameters
208  *  COMMENTS: clock, uart, ports, timer, LCD, i2c, interrupts, pwm
209  *************************************************************************/
210
211 void init()
212 {
213     // CLOCK INIT ------------------------------------------------------
214     IRCF2 = 1;              // 2 MHz INTERNAL CLOCK SOURCE
215     IRCF1 = 0;
216     IRCF0 = 1;
217 //  SCS1 = 1;
218
219     // PORTS INIT ------------------------------------------------------
220
221     TRISA = PORTA_DIR;      // PORTS I/O DEFINITION
222     TRISB = PORTB_DIR;
223     TRISC = PORTC_DIR;
224     TRISD = PORTD_DIR;
225     ADCON1 = 255;           // PORTA AS DIGITAL INPUT
226
227     // LCD INIT --------------------------------------------------------
228     XLCDInit();
229     TRISA0=1;               // BUG ON XLCD INIT....
```

```
230
231     // I2C BUS INIT ----------------------------------------------------
232     i2cInit();
233
234     // UART INIT -------------------------------------------------------
235     BRG16 = 0;              // BAUD RATE GENERATION
236     SPBRG = 51;
237     BRGH = 1;
238     SYNC = 0;               // ASYNCHRONOUS MODE
239     SPEN = 1;               // SERIAL PORT ENABLED
240     TXEN = 0;               // TRANSMISSION DISABLED
241     CREN = 1;               // RECEPTION ENABLED
242
243     // INTERRUPTS INIT -------------------------------------------------
244     TXIE=0;                 // UART INTERRUPTS (RX ON,TX OFF)
245     RCIE=1;
246     RCIP=1;                 // RECEPTION: HIGH PRIORITY
247     RCIF=0;
248     TXIF=0;
249     TMR0IE=1;               // TIMER 0 INTERRUPT
250     TMR0IF=0;
251     TMR0IP=0;
252     TMR1IE=1;               // TIMER 1 INTERRUPT
253     TMR1IF=0;
254     TMR1IP=0;
255
256     IPEN=1;                 // GLOBAL INTERRUPTS AND PRIORITIES
257     GIE = 1;
258     PEIE = 1;
259
260     // TIMER 0 INIT ----------------------------------------------------
261     T08BIT=0;               // REGULATION TIMING (5 MINUTES)
262     T0CS=0;
263     PSA=0;
264     T0PS2=1;                // PRESCALER = 128
265     T0PS1=1;
266     T0PS0=0;
267     TMR0H=0xC2;             // 4 SECONDS TIME
268     TMR0L=0xF7;
269
270     // TIMER 1 INIT ----------------------------------------------------
271     T1CON=0b10110000;       // BLINDS PULSE TIMING
272     TMR1L=0xCC;
273     TMR1H=0xCC;
274     TMR1ON=0;
275
276     //PWM MODULE INIT -------------------------------------------------
277     CCP1CON=0b00001100;     // ENABLE PWM MODE
278     CCPR1L=0;               // DUTY CYCLE INIT TO 0%
279     TRISC2=0;               // PWM PORT
280     PR2=24;                 // PWM FREQUENCY DEFINITION (2 kHz)
281     TMR2ON=1;               // TIMER2 ON TO COMPARE WITH PR2
282 }
283
284 /*************************************************************************
285  *  FUNCTION: I2C ROUTINES FOR REAL TIME CLOCK
286  *  GOAL: This functions allows to manage i2c bus
287  *  COMMENTS: I2C bus reading/writing/initializing
288  *************************************************************************/
289 void i2cInit(void)
290 {
291     TRISC3 = 1 ;            // SDA AND SCL PINS AS INPUT
292     TRISC4 = 1 ;
293     SSPCON1 = 0x38 ;        // PIC AS MASTER
294     SSPCON2 = 0x00 ;
295
296     // TRANSMISSION SPEED
297
298     SSPADD = 4 ;            // Fi2c = Fosc/(4*(SSPADD+1)) = 100 kHz
299     CKE = 0 ;               // INPUT LEVELS FOR I2C
300     SMP = 1 ;               // UNABLE SLEW RATE
301     PSPIF = 0 ;             // CLEARSSPIF INTERRUPT FLAG
302     BCLIF = 0 ;             // CLEAR BUS COLLISION FLAG
303
304     DelayMs(10) ;
305 }
306
```

```
307  // WAIT FOR I2C BUS IS IDLE ------------------------------------------------
308  void i2cWaitForIdle(void)
309  {
310      while ((SSPCON2 & 0x1F) | RW) ; // WAIT FOR IDLE AND NOT WRITING
311  }
312
313  // START I2C BUS -----------------------------------------------------------
314  void i2cStart(void)
315  {
316      i2cWaitForIdle() ;
317      SEN =1 ;                // SEND START
318      while (SEN) ;
319  }
320
321  // RESTART I2C BUS ---------------------------------------------------------
322  void i2cRepStart(void)
323  {
324      i2cWaitForIdle() ;
325      RSEN = 1 ;              // SEND REPEATED START
326      while (RSEN) ;
327  }
328
329  // STOP I2C BUS ------------------------------------------------------------
330  void i2cStop(void)
331  {
332      i2cWaitForIdle() ;
333      PEN = 1;                // SEND STOP
334      while (PEN) ;
335  }
336
337  // READ I2C BUS ------------------------------------------------------------
338  unsigned char i2cRead(unsigned char ack)
339  {
340      unsigned char data ;
341      i2cWaitForIdle() ;
342      RCEN = 1 ;
343      i2cWaitForIdle() ;
344      data = SSPBUF ;
345      i2cWaitForIdle() ;
346      if (ack)    ACKDT = 0 ;         // ACK
347      else        ACKDT = 1 ;         // NO ACK
348      ACKEN = 1 ;                     // SEND ACKNOWLEDGE SEQUENCE
349      return(data) ;
350  }
351
352  // WRITE I2C BUS -----------------------------------------------------------
353  unsigned char i2cWrite(unsigned char data)
354  {
355      i2cWaitForIdle() ;
356      SSPBUF = data ;
357      return(!ACKSTAT) ;              // RETURN 1 IF TRANSMISSION OK
358  }
359
360  /*****************************************************************************
361   *  FUNCTION: REAL TIME CLOCK ROUTINES
362   *  GOAL: This functions allows to read and  write on DS1307 RTC
363   *  COMMENTS: see DS1307 datasheet for more information (registers and...)
364   *****************************************************************************/
365
366  void WriteRTC(void)                 // WRITE DATA INTO RTC
367  {
368      i2cStart() ;
369      i2cWrite(CLOCK_DS1307_W) ;
370      i2cWrite(0x00) ;
371      i2cWrite(0x00) ;                // ENABLE OSCILLATOR, SECONDS=0
372      i2cWrite(((Minute/10) << 4) | Minute%10) ;
373      i2cWrite((((Heure/10) << 4) | Heure%10) & 0x3F) ;
374      i2cWrite(((Jour/10) << 4) | Jour%10) ;
375      i2cWrite(((Mois/10) << 4) | Mois%10) ;
376      i2cWrite(((Annee/10) << 4) | Annee%10) ;
377      i2cWrite(0x00) ;
378      i2cStop() ;
379  }
380
381  void ReadRTC(void)                  // READ DATA INTO RTC
382  {
383      unsigned char temp ;            // TEMPORARY VARIABLE TO READ RTC VALUES
```

```
384
385   rtc_start:
386     i2cStart() ;
387     if (!i2cWrite(CLOCK_DS1307_W))  goto rtc_start ; // WRITE MODE
388     i2cWrite(0x01) ;
389     i2cRepStart() ;
390     if (!i2cWrite(CLOCK_DS1307_R))  goto rtc_start ; // READ MODE
391
392     temp = i2cRead(1) ;
393     Minute = (temp & 0x0F) + (temp >> 4) * 10 ;
394     temp = i2cRead(1) ;
395     Heure = (temp & 0x0F) + (temp >> 4) * 10 ;
396     temp = i2cRead(1) ;
397     Jour = (temp & 0x0F) + (temp >> 4) * 10 ;
398     temp = i2cRead(1) ;
399     Mois = (temp & 0x0F) + (temp >> 4) * 10 ;
400     temp = i2cRead(0) ;
401     Annee = (temp & 0x0F) + (temp >> 4) * 10 ;
402     i2cStop() ;
403 }
404
405 /*************************************************************************
406  *  FUNCTION: PID
407  *  GOAL: This function makes a PID regulation on heating
408  *  COMMENTS: Initialize parameter on the top of the program
409  *************************************************************************/
410
411 void PID(void)
412 {
413     consigne();                                 // SETPOINT CHOICE
414
415     erreur = consignePID - tempInt;             // ERROR CALCULATION
416     u_prim=ui+(Kp+Ki+Kd)*erreur-Kd*erreur_old;  // PID ALGORITHM
417
418     if(u_prim<0)                                // COMMAND SIGNAL SATURATION
419         u=0;
420     else if (u_prim>100)
421         u=100;
422     else
423         u=u_prim;
424
425     duty=(int)u;                                // PWM DUTY CYCLE (0-100%)
426     CCPR1L= duty>>2;
427     CCP1CON &= 0b11001111;
428     CCP1CON|=((duty & 0x03))<<4;                // 2 LSB OF DUTY CYCLE VALUE
429
430     erreur_lim=erreur-(u_prim-u)/(Kp+Ki+Kd);    // INTEGRATOR LIMIT.(ANTIRESET WINDUP)
431     ui=ui+Ki*erreur_lim;
432     erreur_old=erreur;                          // STORE OLD ERROR VALUE
433     updateVis=1;                                // DISPLAY NEW VALUES
434 }
435
436 /*************************************************************************
437  *  FUNCTION:   BLINDS
438  *  GOAL:       This function manage blinds command
439  *  COMMENTS:   Up/down on auto mode; stop in manual mode.
440  *              Mode defined by motion sensor
441  *************************************************************************/
442
443 void blinds()
444 {
445     if(manualMode==0)               // AUTOMATIC MODE
446     {
447         if(storesUp==1 && up==0)    // OPEN BLINDS IF NOT OPENED
448         {
449             dir_up=1;
450             storesUp=0;
451             TMR1L=0xCC;
452             TMR1H=0xCC;
453             TMR1ON=1;               // START TEMPORIZATION
454             RELAIS_DOWN=0;          // COMMAND RELAIS
455             RELAIS_UP=1;
456             up=0;
457             down=0;
458         }
459         if(storesDown==1 && down==0)// CLOSE BLINDS IF NOT CLOSED
460         {
```

```
461             dir_down=1;
462             storesDown=0;
463             TMR1L=0xCC;
464             TMR1H=0xCC;
465             TMR1ON=1;               // START TEMPORIZATION
466             RELAIS_UP=0;            // COMMAND RELAIS
467             RELAIS_DOWN=1;
468             down=0;
469             up=0;
470         }
471     }
472     else
473     {
474         TMR1ON=0;                   // STOP BLINDS
475         RELAIS_UP=0;
476         RELAIS_DOWN=0;
477     }
478 }
479
480 /*********************************************************************
481  *  FUNCTION:   Reception interrupt
482  *  GOAL:       This function generates an interrupt on
483  *              UART received message
484  *  COMMENTS:   High priority
485  ********************************************************************/
486
487 void interrupt reception(void)
488 {
489     if(RCIF == 1)                       // MESSAGE IS PENDING
490     {
491         RCIE=0;
492         RB3=1;
493         receivedMessage = RCREG;        // READ RECEIVED MESSAGE
494
495         if(receivedMessage=='T')        // EXTERNAL TEMPERATURE ID
496         {
497             text=1;
498             textCount=1;
499         }
500         if(receivedMessage=='t')        // INTERNAL TEMPERATURE ID
501         {
502             tint=1;
503             tintCount=1;
504         }
505         if(text==1 && receivedMessage!='T')     // EXTERNAL TEMPERATURE
506         {
507             if(textCount==1)
508             {
509                 tempExt_L=receivedMessage;
510                 textCount=2;
511             }
512             if(textCount==2 && receivedMessage!=tempExt_L)
513             {
514                 tempExt_H=receivedMessage;
515                 tempExt=(tempExt_L+(tempExt_H*256))/100.0;
516                 textCount=0;
517                 text=0;
518                 ReadRTC();
519                 updateVis=1;
520             }
521         }
522         if(tint==1 && receivedMessage!='t' )    // INTERNAL TEMPERATURE
523         {
524             if(tintCount==1)
525             {
526                 tempInt_L=receivedMessage;
527                 tintCount=2;
528             }
529             if(tintCount==2 && receivedMessage!=tempInt_L)
530             {
531                 tempInt_H=receivedMessage;
532                 tempInt=(tempInt_L+(tempInt_H*256))/100.0;
533                 tintCount=0;
534                 tint=0;
535                 ReadRTC();
536                 updateVis=1;
537             }
```

```
538         }
539         if(receivedMessage=='s' && tint==0 && text==0)      // SUN VALUE
540         {
541             storesDown=1;
542             blinds();
543             soleil=0;
544         }
545         if(receivedMessage=='S' && tint==0 && text==0)
546         {
547             if(presence==0) storesUp=1;
548             blinds();
549             soleil=1;
550         }
551         if(receivedMessage=='p'&& tint==0 && text==0)   //NO PRESENCE
552         {
553             presence=0;
554             presenceCounter++;
555             manualMode=0;
556             blinds();
557             presence=0;
558         }
559         if(receivedMessage=='P'&& tint==0 && text==0)   //PRESENCE
560         {
561             presence=1;
562             presenceCounter=0;
563             manualMode=1;
564             DelayMs(50);
565             blinds();
566             presence=1;
567         }
568         RCIE=1;
569         RB3=0;
570     }
571 }
572
573 /*****************************************************************************
574  *  FUNCTION:   time interrupts
575  *  GOAL:       This function generates an interrupt on
576  *              Timer0 and Timer1 flags
577  *  COMMENTS:   Low priority
578  *              Timer0: each 4 seconds (75 times to make 5 minutes)
579  *              Timer1: blinds pulse time
580  *****************************************************************************/
581
582 void low_priority interrupt time(void)
583 {
584     // EACH 5 MINUTES MAKE PID REGULATION ----------------------------------
585     if(TMR0IF==1)
586     {
587         TMR0IF=0;                       // CLEAR FLAG
588         TMR0H=0xC2;                     // RESTART TIMER 0
589         TMR0L=0xF7;
590         count0++;
591         if(count0==75)
592         {
593             PID();
594             count0=0;
595         }
596     }
597
598     // BLINDS TIMING -------------------------------------------------------
599     if(TMR1IF==1)
600     {
601         TMR1ON=0;                       // STOP TIMER 1
602
603         if(dir_down==1)                 // ACK BLINDS CLOSED
604         {
605             down=1;
606             dir_down=0;
607         }
608         if(dir_up==1)                   // ACK BLINDS OPENED
609         {
610             up=1;
611             dir_up=0;
612         }
613         RA4=0;
614         RB0=0;
```

```
615         TMR1IF=0;                       // CLEAR FLAG
616     }
617 }
618 
619 /**************************************************************************
620  *  FUNCTION:   CONSIGNE
621  *  GOAL:       This function calculate normal and reduced setpoints
622  *  COMMENTS:   Setpoint as interface choice
623  **************************************************************************/
624 
625 void consigne()
626 {
627     if( Jour>=6 || Heure<start_time || Heure>=stop_time || ((Heure>=11 && Minute>=30) &&
    (Heure<12 && Minute<30)))   // SETPOINT CALCULATION
628         consignePID=cons_red;                           // REDUCED SETPOINT
629     else
630     {
631         if(presenceCounter>5)
632             consignePID=cons_standby;                   // STANDBY SETPOINT
633         else
634             consignePID=cons_norm;                      // CONFORT SETPOINT
635     }
636 }
637 
638 /**************************************************************************
639  *  FUNCTION:   AFFICHAGE_BASE()
640  *  GOAL:       This function displays data in normal mode
641  *  COMMENTS:   -
642  **************************************************************************/
643 
644 void affichage_base()
645 {
646     GIE=0;
647     XLCDClear();
648     sprintf(data1,"I%3.1f E%3.1f u%3.0f",tempInt,tempExt,u);
649     XLCDL1home();
650     XLCDPutRomString(data1);
651     sprintf(data2,"S%dP%d %2d:%2d c%2.0f", soleil, presence, Heure, Minute, consignePID);
652     XLCDL2home();
653     XLCDPutRomString(data2);
654     GIE=1;
655 }
656 
657 /**************************************************************************
658  *  FUNCTION:   AFFICHAGE_MENU()
659  *  GOAL:       This function displays menu data
660  *  COMMENTS:   -
661  **************************************************************************/
662 
663 void affichage_menu()
664 {
665     switch (menu)
666     {
667         case 1:
668         {
669             XLCDClear();
670             XLCDL1home();
671             XLCDPutRomString("Consigne jour");
672             XLCDL2home();
673             sprintf(param,"Actuel: %2d",cons_norm);
674             XLCDPutRomString(param);
675             break;
676         }
677         case 2:
678         {
679             XLCDClear();
680             XLCDL1home();
681             XLCDPutRomString("Consigne reduite");
682             XLCDL2home();
683             sprintf(param,"Actuel: %2d",cons_standby);
684             XLCDPutRomString(param);
685             break;
686         }
687         case 3:
688         {
689             XLCDClear();
690             XLCDL1home();
```

```
691             XLCDPutRomString("Consigne nuit");
692             XLCDL2home();
693             sprintf(param,"Actuel: %2d",cons_red);
694             XLCDPutRomString(param);
695             break;
696         }
697         case 4:
698         {
699             XLCDClear();
700             XLCDL1home();
701             XLCDPutRomString("Demarrage matin");
702             XLCDL2home();
703             sprintf(param,"Actuel: %2d h",start_time);
704             XLCDPutRomString(param);
705             break;
706         }
707         case 5:
708         {
709             XLCDClear();
710             XLCDL1home();
711             XLCDPutRomString("Arret soir");
712             XLCDL2home();
713             sprintf(param,"Actuel: %2d h",stop_time);
714             XLCDPutRomString(param);
715             break;
716         }
717     }
718 }
719
720 /*********************************************************************
721  *  FUNCTION:   MAIN PROGRAM
722  *  GOAL:       This function is the main program: calls initialisation and
723  *              make an infinity loop
724  *  COMMENTS:   -
725  *********************************************************************/
726
727 void main()
728 {
729     init();
730 //  WriteRTC();                         // ONLY FOR FIRST START
731     ReadRTC();                          // READ RTC VALUE
732     blinds();
733     count0=74;
734     TMR0IF=1;                           // MAKE REGULATION INTERRUPT
735     RB3=0;                              // PIC IS READY
736
737     while(1)                            // INFINITY LOOP
738     {
739         up_actual= UP_BUTTON;           // UPDATE BUTTONS STATUS
740         down_actual= DOWN_BUTTON;
741         left_actual=LEFT_BUTTON;
742         right_actual=RIGHT_BUTTON;
743
744         if(OERR==1)                     // RX BUFFER ERROR => CLEAR FLAG
745         {
746             CREN=0;
747             CREN=1;
748         }
749
750         if( updateVis==1)               // DATA DISPLAY
751         {
752             affichage_base();
753             updateVis=0;
754         }
755
756         if(up_actual!=up_old && up_actual==0)           // UP BUTTON PRESSED
757         {
758             switch(menu)                                // MODIFY MENU VALUES
759             {
760                 case 1:
761                 {
762                     cons_norm++;
763                     break;
764                 }
765                 case 2:
766                 {
767                     cons_standby++;
```

```
768                     break;
769                 }
770                 case 3:
771                 {
772                     cons_red++;
773                     break;
774                 }
775                 case 4:
776                 {
777                     start_time++;
778 
779                     break;
780                 }
781                 case 5:
782                 {
783                     stop_time++;
784                     break;
785                 }
786             }
787             affichage_menu();
788         }
789         if(down_actual!=down_old && down_actual==0)     // DOWN BUTTON PRESSED
790         {
791             switch(menu)                                // MODIFY MENU VALUES
792             {
793                 case 1:
794                 {
795                     cons_norm--;
796                     break;
797                 }
798                 case 2:
799                 {
800                     cons_standby--;
801                     break;
802                 }
803                 case 3:
804                 {
805                     cons_red--;
806                     break;
807                 }
808                 case 4:
809                 {
810                     start_time--;
811                     break;
812                 }
813                 case 5:
814                 {
815                     stop_time--;
816                     break;
817                 }
818             }
819             affichage_menu();
820         }
821 
822         if(left_actual!=left_old && left_actual==0)     // LEFT BUTTON PRESSED
823         {
824             menu=0;
825             updateVis=1;
826             count0=73;
827             GIE=1;
828         }
829         if(right_actual!=right_old && right_actual==0)  // RIGHT BUTTON PRESSED
830         {
831             if(menu==0)
832                 GIE=0;
833             if(menu==5)
834                 menu=0;
835             menu++;
836             affichage_menu();
837         }
838 
839         up_old=up_actual;                               // STORE OLD BUTTON VALUES
840         down_old=down_actual;
841         left_old=left_actual;
842         right_old=right_actual;
843     }
844 }
```

ANNEXE 7

Contenu du CD

Le CD qui se trouve sur la dernière page de ce rapport contient les datasheets des principaux composants présents sur les cartes, en format PDF.

Il s'agit des composants suivants :

- Microcontrôleur PIC18F1220
- Microcontrôleur PIC 18F4525
- Module Radio ER900TRS-02
- Capteur de température digitale DS18S20
- Capteur de présence Matsushita AMN11111
- Photorésistance SILONEX NSL 4960
- Amplificateur opérationnel LM6482
- Driver relais ULN2803
- Horloge temps réel DS1307
- Relais finder série 32
- Servovanne électrique Danfoss DDC Alpha
- Alimentation switching MAX 756