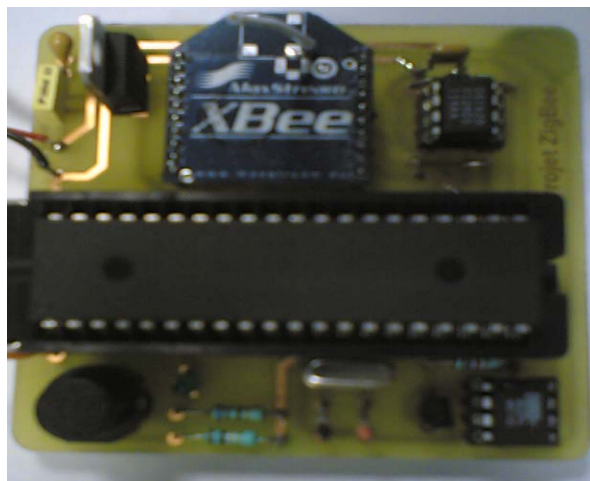


*Cedric BEAUSSE  
Mohamed BOUGUERRA  
Hong Yu GUAN  
El Ayachi MOKTAD*



*Projet avancé en systèmes embarqués  
2006 / 2007*

*Réalisation d'un réseau de capteurs de température  
sans fil basé sur le protocole ZigBee*



## *Sommaire*

<b>Introduction.....</b>	<b>3</b>
<b>1- Le protocole ZigBee.....</b>	<b>3</b>
<b>1-1 Protocoles sans fil.....</b>	<b>3</b>
<b>1-2 Comparaison entre les protocoles sans fil.....</b>	<b>4</b>
<b>2- Le Module « XBee ».....</b>	<b>5</b>
<b>2-1 Caractéristiques des modules XBee.....</b>	<b>5</b>
<b>2-2 Description du module XBee.....</b>	<b>5</b>
<b>2-3 Communication RF via le module XBee.....</b>	<b>7</b>
<b>3- Développement du Hardware.....</b>	<b>9</b>
<b>3-1 Description des fonctions de la carte.....</b>	<b>9</b>
<b>3-2 Réalisation de la carte.....</b>	<b>14</b>
<b>4- Développement du software.....</b>	<b>15</b>
<b>4-1 Librairie du DS1620.....</b>	<b>16</b>
<b>4-2 Librairie XBee.....</b>	<b>18</b>
<b>4-3 Librairie de l'UART.....</b>	<b>19</b>
<b>4-4 Programme principal.....</b>	<b>19</b>
<b>Conclusion.....</b>	<b>20</b>
<b>Bibliographie.....</b>	<b>20</b>
<b>Annexes.....</b>	<b>21</b>

# Introduction

L'objectif de ce projet est de concevoir et réaliser un système numérique à base de microcontrôleur permettant l'acquisition de la température par un capteur numérique et de transmettre la valeur vers un PC par voie RF en utilisant le protocole ZigBee. Ainsi, ce projet comporte deux parties : le développement de la carte permettant d'assurer les fonctions demandées, et le développement du logiciel pour le pilotage de la carte. Dans la suite, nous présenterons le protocole ZigBee ainsi que le module XBee permettant d'implémenter ce protocole. Et enfin, nous décrirons la réalisation du hardware et du software pour ce projet.

## 1- Le protocole ZigBee [1]

Afin de révéler l'intérêt du protocole ZigBee, nous allons présenter en plus de ce protocole quelques protocoles sans fil notamment le protocole Wi-fi et le bluetooth.

### 1-1 Protocoles sans fil

#### ❖ *Protocole Wi-fi*

Le Wi-Fi est une technologie de réseau informatique sans fil mise en place pour fonctionner en réseau interne et, depuis, devenue un moyen d'accès à haut débit à Internet. Il est basé sur la norme IEEE 802.11 (ISO/CEI 8802-11).

En pratique, pour un usage informatique du réseau Wi-Fi, il est nécessaire de disposer au minimum de deux équipements Wi-Fi, par exemple un ordinateur, et un routeur ADSL.

#### ❖ *Protocole bluetooth*

Le Bluetooth est une spécification de l'industrie des télécommunications. Elle utilise une technologie radio courte distance destinée à simplifier les connexions entre les appareils électroniques. Elle a été conçue dans le but de remplacer les câbles entre les ordinateurs et les imprimantes, les scanners, les claviers, les souris, les téléphones portables, les PDAs et les appareils photos numériques.

Le Bluetooth sera capable d'assurer des débits cent fois supérieurs par rapport à la version actuelle passant donc de 1 Mb/s à 100 Mb/s (soit 12,5 Mo/s). Cette technologie - utilisée dans les téléphones mobiles, périphériques informatiques et autres appareils portables comme les assistants personnels (PDA), verra sa vitesse de transmission augmenter dans les années à venir, lui permettant alors d'être utilisée pour les vidéos haute définition et l'échange de fichiers avec son baladeur MP3 par exemple. La nouvelle norme incorporera une nouvelle technologie radio, connue comme l'Ultra wideband ou UWB.

### ❖ *Protocole ZigBee :*

ZigBee est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur le standard IEEE 802.15.4 pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPANs). Ratifiées le 14 décembre 2004, les spécifications de ZigBee 1.0 sont disponibles auprès des membres de la communauté industrielle ZigBee Alliance. Cette technologie a pour but la communication de courte distance telle que le propose déjà la technologie Bluetooth, tout en étant moins chère et plus simple.

La spécification initiale de ZigBee propose un protocole lent dont le rayon d'action est relativement faible, mais dont la fiabilité est assez élevée, le prix de revient faible et la consommation considérablement réduite.

On retrouve donc ce protocole dans des environnements embarqués où la consommation est un critère de sélection. Ainsi, la domotique et les nombreux capteurs qu'elle implémente apprécie particulièrement ce protocole en plein essor et dont la configuration du réseau maillée se fait automatiquement en fonction de l'ajout ou de la suppression de nœuds. On retrouve aussi ZigBee dans les contrôles industriels, les applications médicales, les détecteurs de fumée et d'intrusion.

Les nœuds sont conçus pour fonctionner plusieurs mois (jusqu'à deux ans pour les moins consommant) en autonomie complète grâce à une simple pile alcaline de 1,5V.

## 1-2 Comparaison entre les protocoles sans fil

Protocole	Zigbee	Bluetooth	Wi-Fi
IEEE	802.15.4	802.15.1	802.11a/b/g
Besoins mémoire	4-32 Kb	250 Kb +	1 Mb +
Durée de vie	Années	Jours	Heures
Nombre de nœuds	65 000+	7	32
Vitesse de transfert	250 Kb/s	1 Mb/s	11-54 Mb/s
Portée	100 m	10 m	100 m

**Tableau 1 : Comparaison des protocoles sans fil**

Toutes les caractéristiques du protocole ZigBee sont bien adaptées aux systèmes embarqués. En effet, le protocole ZigBee se distingue des autres protocoles par ses faibles besoins en mémoire, ce qu'est favorable pour son implémentation. De plus, il présente une durée de vie très importante qu'est de l'ordre de plusieurs années, ainsi qu'un très large nombre de nœuds à supporter dans son réseau. Enfin, ce protocole convient parfaitement aux applications nécessitant un faible vitesse de transfert de l'ordre de 250 Kb/s.

## 2- Le Module « XBee » [2]

Les modules « OEM XBee » sont des transceivers radio au standard ZigBee™ / IEEE 802.15.4, qui sont spécialement conçus pour la réalisation de systèmes de communication au sein de réseaux de capteurs sans fil. De petites dimensions, ces modules se distinguent par leur grande simplicité d'utilisation et leur coût très compétitif qui les prédestinent également à de très nombreuses autres applications.

Les modules XBee opèrent dans la bande ISM des 2.4 GHz, et ils reposent sur une technologie de communication de type DSSS (Direct Sequence Spread Spectrum). Pouvant être utilisés sans aucun paramétrable, les modules disposent également d'une multitude de modes de fonctionnement très facilement accessibles via des commandes AT.

### 2-1 Caractéristiques des modules XBee

Les modules XBee présentent des hautes performances en terme de transmission RF, de consommation électrique et de gestion réseaux.

#### ➤ **Communication RF :**

Les modules XBee présentent une puissance de sortie RF de 10 mW (0dbm). Concernant la portée intérieure, elle peut atteindre au maximum les 30 mètres suivant la nature des obstacles. Pour la portée extérieure, elle est au maximum de 100 mètres en champ libre. Le débit RF est de 250 Kbps alors que le débit à l'interface est au maximum de 115.2 Kbps. Enfin, le récepteur possède une sensibilité de -92 dbm.

#### ➤ **Gestion réseaux :**

Les modules XBee présentent des caractéristiques avancées en matière de gestion de réseaux. En effet, il est possible de configurer les modules sur près de 65.000 adresses différentes ainsi que de disposer de communications point à point, multipoints, broadcast ou encore "peer-to-peer".

#### ➤ **Consommation :**

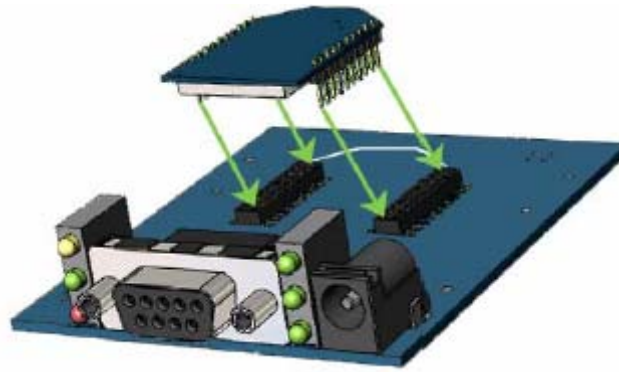
Le module XBee est alimenté sous une tension comprise entre 2.8 et 3.4 V. Pour une alimentation de 3.3 V, la consommation est de 45 mA en émission et elle vaut 50 mA en réception.

### 2-2 Description du module XBee

Les modules Xbee présentent les propriétés physiques suivantes :

- Dimensions : 2.438 cm x 2.761 cm
- Poids : 3g
- Températures de fonctionnement : -40 °C à 85° C

Le module XBee se monte sur une carte à l'aide d'un support permettant de connecter ses 20 broches comme le montre la figure 1. Ainsi aucune soudure n'est nécessaire pour câbler ce composant.



**Figure 1 : Montage d'un module XBee sur une carte avec une interface RS232**

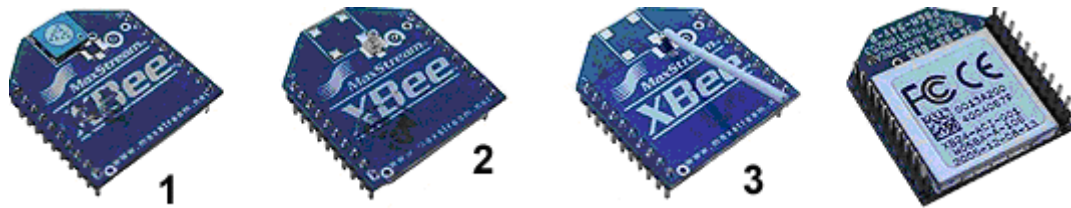
Le tableau suivant décrit les fonctions des 20 broches présentes sur le module XBee. Pour notre application, seulement les 9 broches encadrées sont câblées. En effet, la communication entre modules XBee nécessitent au minimum de câbler les broches d'alimentation (VCC et GND) et les signaux DIN et DOUT pour respectivement les données entrantes et sortantes via le module. Les autres broches câblées servent à configurer le module dans différents modes (réception, émission, veille...)

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / CONFIG	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	RESET	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	[reserved]	-	Do not connect
8	[reserved]	-	Do not connect
9	DIR / SLEEP_RQ* / DIO	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4* / DIO4*	Either	Analog Input 4 or Digital I/O 4
12	CTS / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / SLEEP	Output	Module Status Indicator
14	VREF*	Input	Voltage Reference for A/D Inputs
15	Associate / AD5* / DIO5*	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	RTS* / AD6* / DIO6*	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3* / DIO3*	Either	Analog Input 3 or Digital I/O 3
18	AD2* / DIO2*	Either	Analog Input 2 or Digital I/O 2
19	AD1* / DIO1*	Either	Analog Input 1 or Digital I/O 1
20	AD0* / DIO0*	Either	Analog Input 0 or Digital I/O 0

\* Functions not supported at the time of this release.

**Tableau 2 : Description des broches du module XBee**

**NOTE : Les différentes versions disponibles des modules XBee**



**Figure 2 : Différentes versions du module XBee**

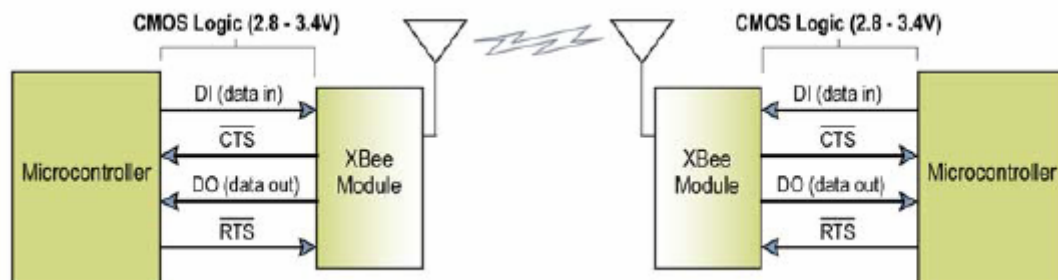
Il existe 3 versions différentes de modules XBee (voir figure 2):

- Version avec antenne Chip intégrée (XBee1)
- Version avec connecteur U.FL (MMCX) pour antenne externe (XBee2).
- Version avec antenne filaire intégrée (XBee3).

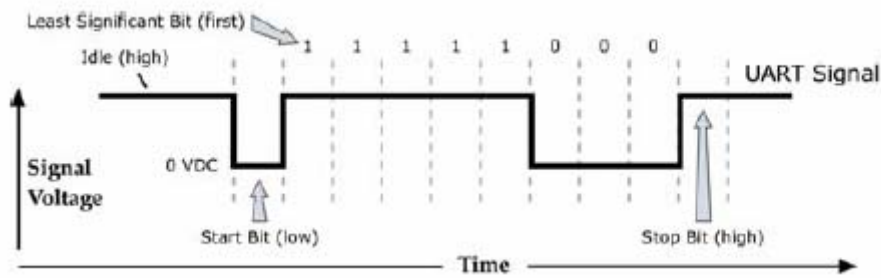
Le modèle "XBee1" est tout indiqué lorsque vous avez des contraintes de dimensions sur votre application. Le modèle "XBee2" est préconisé lorsque le module est enfermé dans un coffret métallique (afin d'éviter l'effet cage de Faraday) ou si vous désirez optimiser les portées avec une antenne appropriée. Le modèle "XBee3" est également préconisé si vous avez ces contraintes de dimensions au sein de votre application (ce dernier offre moins de directivité que le module "XBee1"). A noter enfin que les modules XBee sont également compatibles broches à broches avec les modules XBee-PRO.

## **2-3 Communication RF via le module XBee**

Le module XBee peut être connecté à n'importe quel circuit présentant une liaison UART notamment les microcontrôleurs comme le montre la figure 3. Le microcontrôleur envoie ses données en via le port DI d'une manière série et asynchrone selon l'ordre illustré par la figure 4. Chaque paquet est constitué d'un bit start de niveau bas, suivi de 8 bits de données avec le bit de poids faible en premier, et enfin un bit stop de niveau haut. La synchronisation des données et la vérification de la parité sont assurées par le module UART.



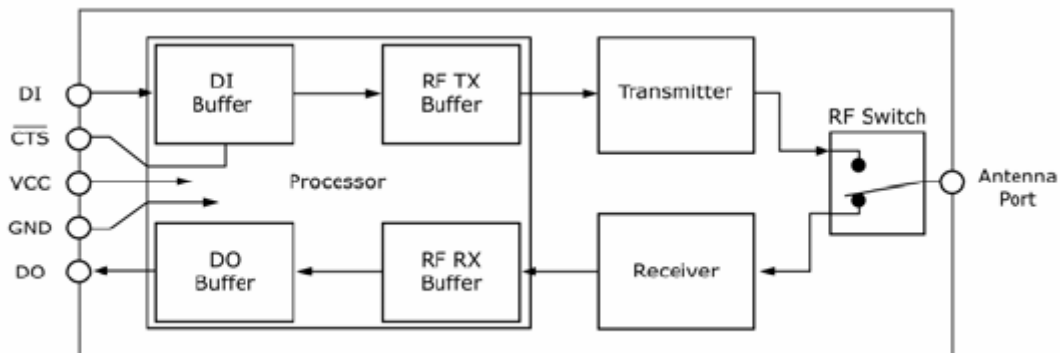
**Figure 3: Système de communication RF utilisant des modules XBee**



**Figure 4 : Exemple de flux de données envoyé via le port série**

La figure 5 présente une vue interne du module XBee. Etant connecté à un circuit présentant une liaison série asynchrone, le module utilise le « buffer DI » pour stocker les données transmises par ce circuit via le port DI. Ce flux de données est contrôlé par le signal /CTS de la manière suivante : lorsque le buffer DI ne dispose que d'un espace libre de 138 bits, /CTS est mis à 1 afin de signaler au circuit d'arrêter l'envoi de données. Il est remis à 0 que lorsque le buffer DI dispose de 276 bits d'espace mémoire libre.

Par ailleurs, le buffer DO est utilisé pour stocker les données envoyées par un autre module XBee par exemple. Lorsque le buffer DO atteint sa capacité maximale, toute donnée envoyée par voie RF est perdue. Ce flux est également contrôlé par le signal /RTS : lorsqu'il est au niveau haut, les données restent stockées dans le buffer DO et ils ne sont transmises via le port DO que lorsqu'il est niveau bas.



**Figure 5 : Architecture interne du module XBee**

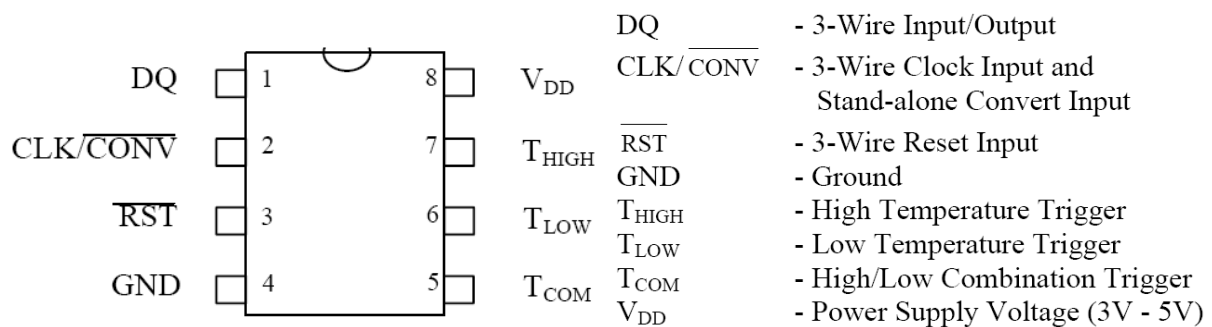
### 3- Réalisation du Hardware

#### 3-1 Description des fonctions de la carte

Voici un descriptif des différentes fonctions de notre projet, dont le schéma complet est illustré par la figure 7.

**FP1 :** Cette fonction réalise le découplage des composants consommant le plus de courant, les condensateurs servent de réservoir lors des appels de courant des composants. Ils sont placés au plus près des composants concernés. C1 est à coté du PIC et C3 à coté du DS1620.

**FP2 :** Cette fonction permet la mesure de la température grâce à un capteur numérique le DS1620. L'avantage de ce composant est qu'il est facilement interfaçable avec un microcontrôleur grâce à une connexion série 3 fils directement intégrée au composant, comme le montre la figure suivante.



**Figure 6 : Le capteur numérique DS1620**

Ce composant est câblé au port C du PIC sur les broches RC3, RC4 et RC5. Il fournit la température avec une précision de 0,5°C, avec des mots de 9 bits en complément à 2 comme le montre le tableau ci-dessous.

TEMP	DIGITAL OUTPUT (Binary)
+125°C	0 11111010
+25°C	0 00110010
+½°C	0 00000001
+0°C	0 00000000
-½°C	1 11111111
-25°C	1 11001110
-55°C	1 10010010

**Tableau 3 : Conversion binaire de la température**

**FP3 :** cette fonction permet de vérifier que le thermomètre fonctionne. La LED est branchée sur la broche RA4 qui est un collecteur ouvert. Pour allumer la LED il faut mettre un '0' sur la broche. Pour permettre l'allumage de la LED il faut mettre en série de celle-ci une résistance pour limiter le courant la traversant :

$$\frac{5V - 2V}{0,005mA} = 600\Omega, \text{ On prend } 680\Omega \text{ comme valeur normalisée.}$$

**FP4 :** Ce bouton poussoir permet de faire un reset de la carte. Au repos la broche est à '1'. Il est câblé sur l'entrée /MCLR du PIC. Un condensateur est placé en parallèle pour éviter les rebonds.

La fréquence de coupure du filtre est de :  $\frac{1}{2 \times \pi \times 1\mu F \times 10k\Omega} = 16Hz$

**FP5 :** Voici le microcontrôleur PIC 16F877 dont voici les caractéristiques :

Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F877
Operating Frequency	DC - 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	8K
Data Memory (bytes)	368
EEPROM Data Memory	256
Interrupts	14
I/O Ports	Ports A,B,C,D,E
Timers	3
Capture/Compare/PWM modules	2
Serial Communications	MSSP, USART
Parallel Communications	PSP
10-bit Analog-to-Digital Module	8 input channels
Instruction Set	35 Instructions

**Tableau 4 : Caractéristiques du PIC16F877**

**FP6 :** Cette fonction réalise l'oscillateur du PIC grâce à un quartz de 20MHz, les valeurs des condensateurs sont tirées de la documentation du PIC. Elle est câblée sur les broches OSC1, OSC2 du PIC.

Osc Type	Crystal Freq	Cap. Range C1	Cap. Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF

**Tableau 5 : Les différents oscillateurs du PIC**

**FP7 :** Cette fonction permet l'adaptation de la tension 5V vers le 3.3V pour l'alimentation du ZigBee. Les valeurs des deux capacités sont données ci-dessous :  
DATASHEET DU REGULATEUR

**FP8 :** Cette fonction permet l'adaptation des niveaux de tension pour la liaison série entre le XBee et le PIC. Pour la mise en œuvre de cette liaison nous utilisons l'UART intégré au PIC et avons donc relié le XBee sur les broches Tx et Rx du PIC. Les tableaux qui suivent présentent les différents niveaux de tension en sortie du XBee et en entrée du PIC ( $V_{IH}$  et  $V_{OH}$ ) :

D041	with Schmitt Trigger buffer	0.8VDD	-	VDD	V	For entire VDD range
D042	MCLR	0.8VDD	-	VDD	V	
D042A	OSC1 (XT, HS and LP)	0.7VDD	-	VDD	V	Note1
D043	OSC1 (in RC mode)	0.9VDD	-	VDD	V	

Symbol	Parameter	Condition	Min	Typical	Max	Units
V <sub>IL</sub>	Input Low Voltage	All Digital Inputs	-	-	0.35 * VCC	V
V <sub>IH</sub>	Input High Voltage	All Digital Inputs	0.7 * VCC	-	-	V
V <sub>OL</sub>	Output Low Voltage	I <sub>OL</sub> = 2 mA, VCC >= 2.7 V	-	-	0.5	V
V <sub>OH</sub>	Output High Voltage	I <sub>OH</sub> = -2 mA, VCC >= 2.7 V	VCC - 0.5	-	-	V
I <sub>IN</sub>	Input Leakage Current	V <sub>IN</sub> = VCC or GND, all inputs, per pin	-	0.025	1	µA
I <sub>OZ</sub>	High Impedance Leakage Current	V <sub>IN</sub> = VCC or GND, all I/O High-Z, per pin	-	0.025	1	µA
TX	Transmit Current	VCC = 3.3 V	-	45 (XBee)   215 (PRO)	-	mA
RX	Receive Current	VCC = 3.3 V	-	50 (XBee)   55 (PRO)	-	mA
PWR-DWN	Power-down Current	SM parameter = 1	-	< 10	-	µA

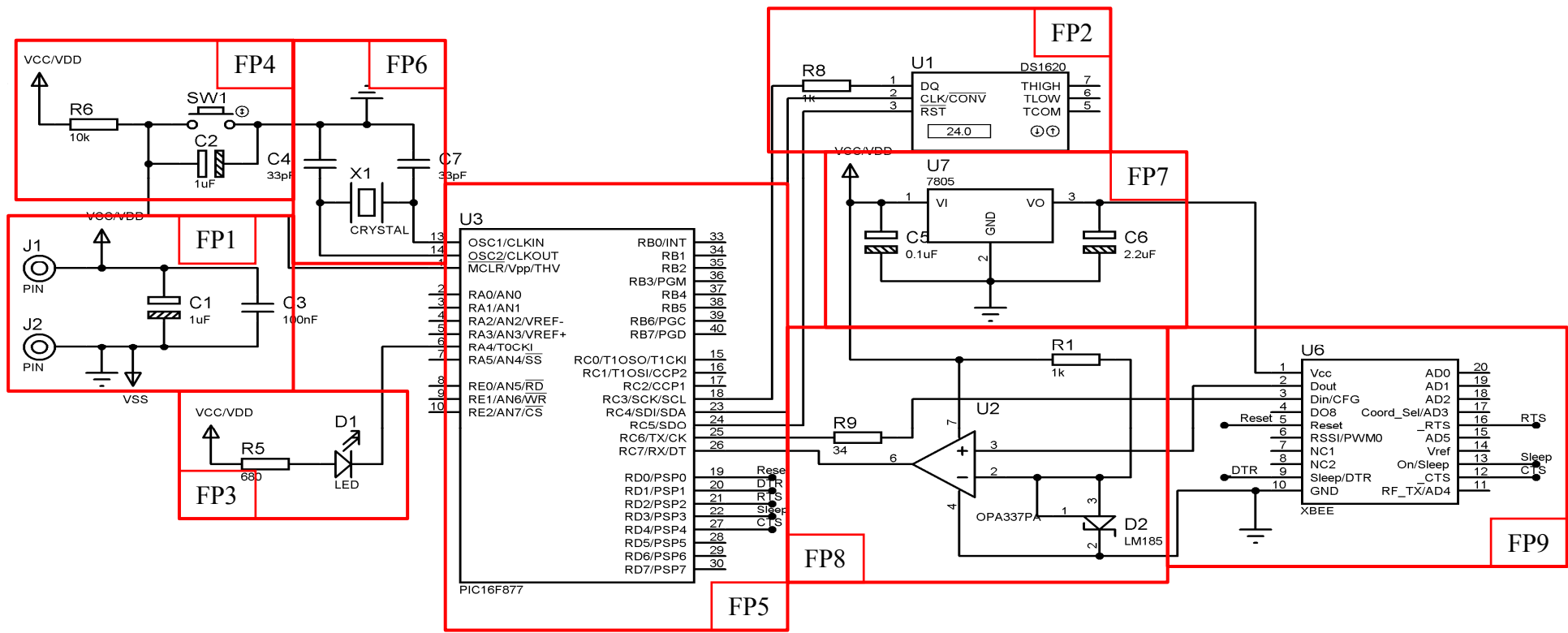
**Tableau 6 : Caractéristiques électriques du PIC et du XBee**

Le XBee fournit une tension de 2,7V minimum et évidemment 3,3V au maximum alors que le PIC a besoin de 4V au minimum pour voir un '1' logique sur son entrée. Pour réaliser cette adaptation nous avons placé un AOp monté en comparateur. Ce dernier compare la tension en sortie du XBee et la compare avec une référence de tension de 2,5V. Dès que le XBee envoie un '1' logique le comparateur fournit du 5V.

En ce qui concerne la réception des données nous avons seulement placé une résistance de 34Ω pour limiter le courant à l'entrée du XBee :

$$\frac{5V - 3,3V}{50mA} = 34\Omega$$

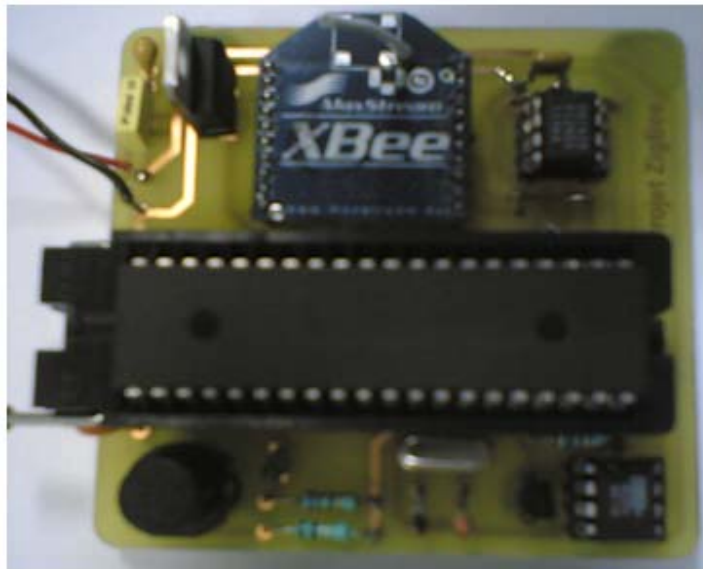
**FP9 :** Cette fonction est réalisée par le XBee de MaxStream. Ce dernier est alimenté par la fonction FP7 et est relié au PIC par la fonction FP8.



**Figure 7 : Schéma du montage de la carte**

### 3-2 Réalisation de la carte

Voici la carte finale de notre projet.

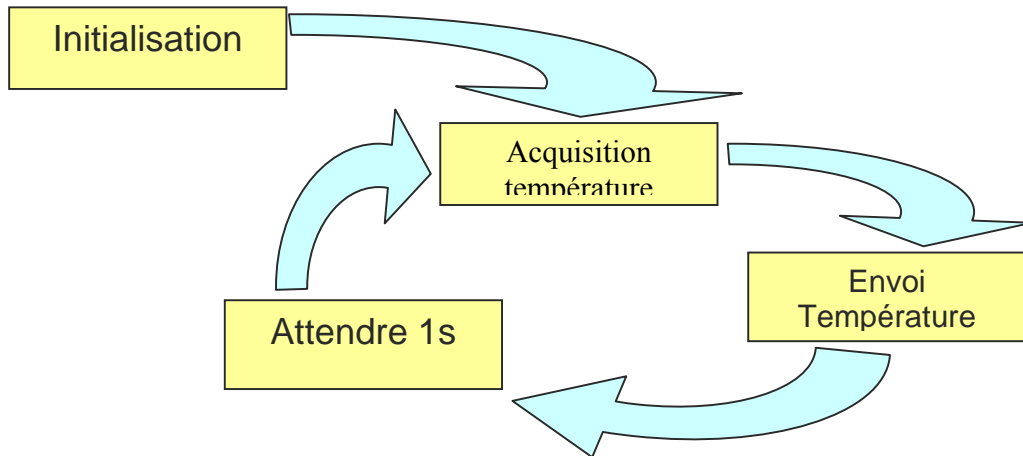


**Figure 8 : Schéma du montage de la carte**

Le schéma de la carte a été réalisé sous ISIS et le routage sous ARES. Le placement des composants et les typons sont fournis en annexe.

## 4- Développement du software

Le système conçu doit permettre l'envoi périodique de la température vers le PC via la liaison ZigBee avec une période d'une seconde. Le diagramme suivant illustre ce fonctionnement :

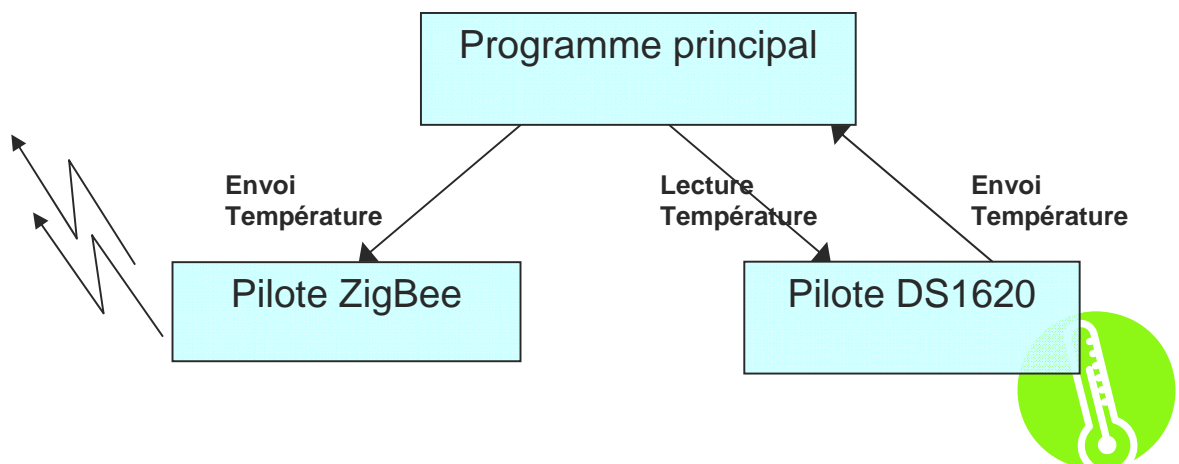


**Figure 9: Déroulement du programme**

Le code sources est organisé en trois parties :

- Le programme principal
- La librairie DS1620
- La librairie Xbee

Comme l'illustre la figure suivante :



**Figure 10: Organisation du code**

Le programme principal communique avec les deux drivers et synchronise les événements.

#### 4-1 Librairie du DS1620

Afin de communiquer avec le thermomètre numérique, il a tout d'abord fallu développer les fonctions d'échanges de données.

La toute première concerne l'écriture vers le composant, *ds1620\_write8*. Elle permet, entre autres, d'envoyer au capteur des paramètres de configuration, mais aussi des ordres de conversion et d'envoi de la température.

Il faut tout d'abord porter le port C du PIC en output (par défaut nous l'avions posé en input).

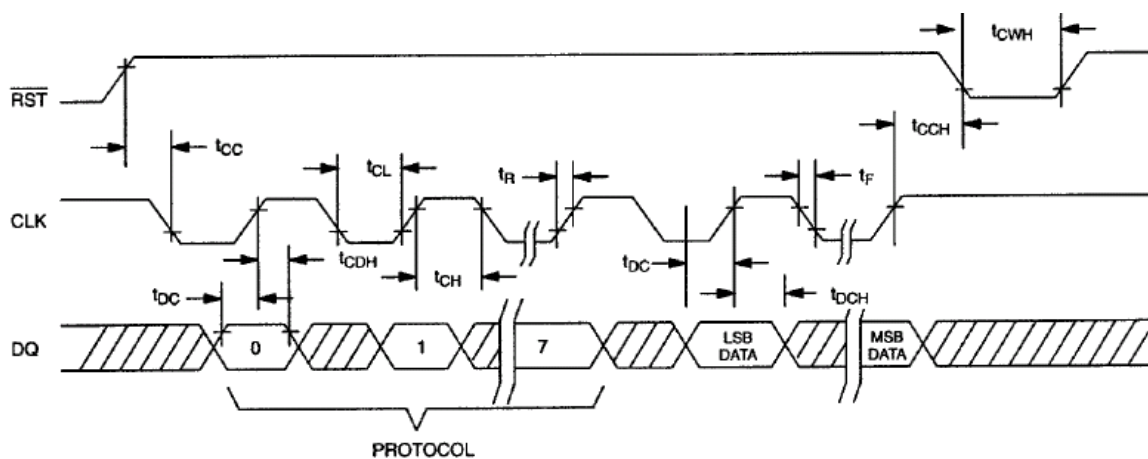
Le reste de la fonction suit le protocole du constructeur, sauf pour le niveau de \*RST, que l'on a préféré changer en dehors de la fonction.

Voici les opérations effectuées pour chaque bit transmis :

- Mise au niveau bas du bit RC4 (CLK)
- Envoi du bit concerné vers le thermomètre
- Mise au niveau haut de CLK.

Aucune instruction de timing supplémentaire n'est nécessaire, ces étapes s'effectuant toutes plus lentement que les différents temps prévus par le constructeur (de l'ordre des dizaines de nanosecondes).

Une fois les 8bits transmis, le port C est remis en input. Voici les chronogrammes du cycle d'écriture :



Il a ensuite fallu créer deux fonctions similaires, *ds1620\_read8* et *ds1620\_read9*, qui gèrent les bits envoyés par le thermomètre au PIC, dans des mots de 8 et 9 bits respectivement.

Une fois encore le \*RST est géré en dehors de la fonction.

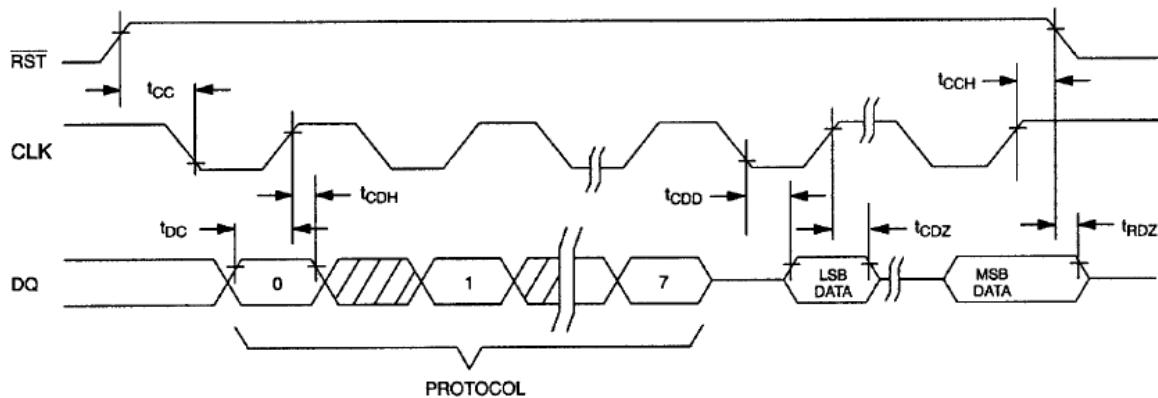
Ici aussi les différents timings du constructeur sont très inférieurs à la durée d'exécution réelle des instructions, ce qui fait que le code n'en tient pas compte.

Il s'agit pour chaque bit de procéder à ces opérations :

- Mise au niveau bas du bit RC4 (CLK)

- Réception du bit concerné vers le thermomètre, qui est alors « concaténé » aux bits déjà réceptionnés.
- Mise au niveau haut de CLK.

Ci-dessous ce trouve les chronogrammes du cycle de lecture :



A l'aide de ces fonctions, il va être possible de gérer le thermomètre. En premier lieu, vient la fonction *ds1620\_init* qui initialise le thermomètre numérique. Pour cela, elle utilise la fonction *ds1620\_write8* pour envoyer les instructions, et c'est elle qui doit gérer le \*RST.

Son appel provoque les opérations :

- \*RST en position haute
- Ecriture du caractère 0x0C : commande Write Config.
- Ecriture de 0x03, qui donne au thermomètre les deux paramètres suivants : utilisation d'un CPU, et fonctionnement en mode one-shot.
- \*RST en position basse pour la fin de l'écriture.

Cette configuration va alors être sauvegardée par le thermomètre dans sa mémoire E<sup>2</sup>. On attend alors 50ms pour gérer le temps d'écriture.

Il ne reste plus ensuite qu'à déclencher le début de la conversion de la température grâce à la fonction *ds1620\_start*.

Cette dernière sera aussi utilisée par le programme principal : en effet, le thermomètre étant en mode one-shot, il procède à une seule conversion puis s'arrête. Ceci est nécessaire pour avoir ensuite accès aux registres SLOPE et COUNTER. Cela veut aussi dire que *ds1620\_start* devra être appelée après chaque rafraîchissement de la température du *main*.

Cette fonction consiste en :

- \*RST en position haute
- Ecriture du caractère 0xEE : commande de début de conversion.
- \*RST en position basse pour la fin de l'écriture.

Il faut ensuite pouvoir lire la température que le thermomètre a converti. Ce sera chose faite avec la fonction *ds1620\_readtemp*:

- \*RST en position haute

- Ecriture du caractère 0xAA : ordre d'envoi de la valeur de la température.
- Réception des 9bits grâce à *ds1620\_read9* et stockage local.
- \*RST en position basse pour la fin de l'écriture.

On a ainsi à disposition la valeur de la température à 0.5°C près. Comme cette précision n'est pas assez grande (le sujet impose une valeur à 0.1°C près), il faut procéder à des calculs supplémentaires qui mettent en jeu les valeurs contenues après conversion dans les registres SLOPE et COUNTER.

Nous avons donc deux fonctions supplémentaires : *ds1620\_counter* et *ds1620\_slope*. Elles sont similaires à *ds1620\_readtemp* excepté pour le caractère envoyé au capteur. Dans le cas de *ds1620\_counter*, le caractère 0xA0 est écrit, contre 0xA9 pour *ds1620\_slope*, qui donne respectivement les ordres de lecture des registres concernés.

On dispose ainsi de toutes les fonctions nécessaires à la gestion du thermomètre numérique dans le programme principal.

## **4-2 Librairie du XBee :**

Afin de communiquer avec le module XBee, il a fallu développer des fonctions qui permettent de le configurer et d'échanger des données avec.

Cette bibliothèque contient quatre fonctions :

**XBeeInit()** : fonctions d'initialisation du module XBee.

Elle initialise les pins CTS=1, DTR=0, SLEEP=0 afin d'activer la communication avec ce module

**SendByte()** : permet d'envoyer un caractère via liaison ZigBee vers le PC.

Elle utilise la fonction EmitUart() de la librairie de l'UART don le fonctionnement est décrit plus loin.

**ReadByte()** : Permet de lire un caractère reçu via la liaison ZigBee du PC

Elle utilise la fonction RecUart() de la librairie de l'UART don le fonctionnement est décrit plus loin.

**SendString()** : d'envoyer une chaine de caractères via liaison ZigBee vers le PC.

Elle utilise la fonction SendByte() pour envoyer les caractère un par un.

### 4-3 Librairie de l'UART :

On peut également inclure dans cette librairie celle de l'UART car il est utilisé uniquement pour piloter le module XBee, elle comporte les fonctions suivantes :

**UartInit()** : elle permet l'initialisation de la liaison série

Elle agit sur les registres SPBRG, TXSTA, RCSTA pour configurer une liaison série asynchrone 9600baud comme spécifié sur la documentation de l'XBee afin d'assurer la compatibilité

**EmitUart()** : permet d'envoyer un octet sur la liaison série

L'envoi de l'octet se fait en l'écrivant dans le registre TXREG, l'envoi alors se fait automatiquement, il suffit ensuite d'attendre que cet octet soit envoyé pour en envoyer un deuxième de la même façon.

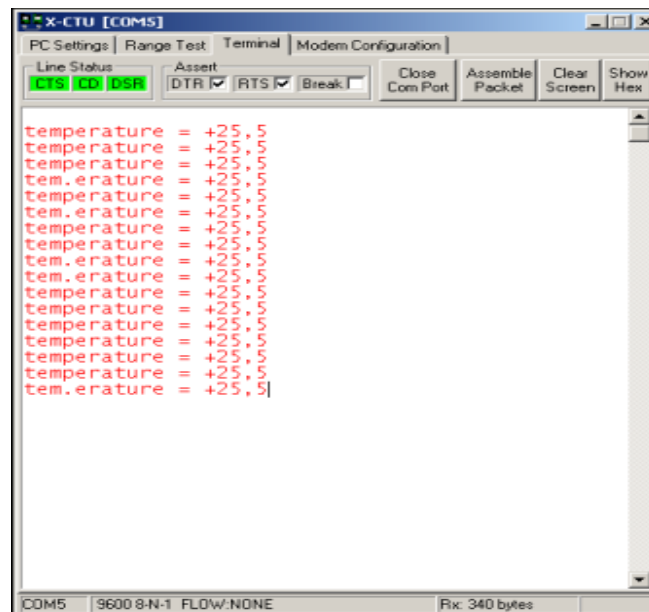
**RecUart()** : permet de lire un octet reçu sur la liaison série

Lorsqu'un octet est reçu il est signalé par le flag RCIF, il suffit donc de lire le contenu du registre RCREG pour récupérer l'octet reçu.

### 4-4 Programme principal

Le programme principal commence par effectuer toutes les initialisations des différents modules (DS1620, UART et XBee), il lance en suite une conversion de température sur le DS1620, puis une acquisition de température, la valeur de la température est ensuite formaté en chaîne de caractère ASCII pour être envoyé octet par octet vers le PC.

Ces opérations sont répétées toutes les secondes, on obtient donc sur le moniteur ZigBee sur le PC la valeur de la température réactualisée toutes les secondes.



**Figure 11: Résultat de test du programme**

## Conclusion

Les résultats obtenus au terme de ce projet sont satisfaisants. En effet le capteur ainsi réalisé permet la visualisation de la température toute les secondes sur l'ordinateur via une liaison sans fil ZigBee. Cependant des améliorations peuvent être apportées à ce système en rendant la liaison bidirectionnelle et en le munissant d'une reconnaissance de requêtes et de plus de services. Le cruel manque de temps ne nous a pas permis d'apporter ces améliorations.

Ce projet nous a permis de manipuler une jeune technologie très prometteuse qu'est le ZigBee. Cette technologie, facile à mettre en place grâce à des modules comme le XBee de MaxStream que nous avons utilisé se montre très adaptée à des petits systèmes embarqués ne nécessitant pas de haut débit de transfert, tels que les appareils de contrôle et d'acquisition. Ce qui la rend encore plus adaptée à ce type de systèmes est sa faible consommation et la petite taille des modules. Elle est aussi bien adaptée à une utilisation en milieu industriel grâce à sa portée et son bon fonctionnement dans des environnements bruités.

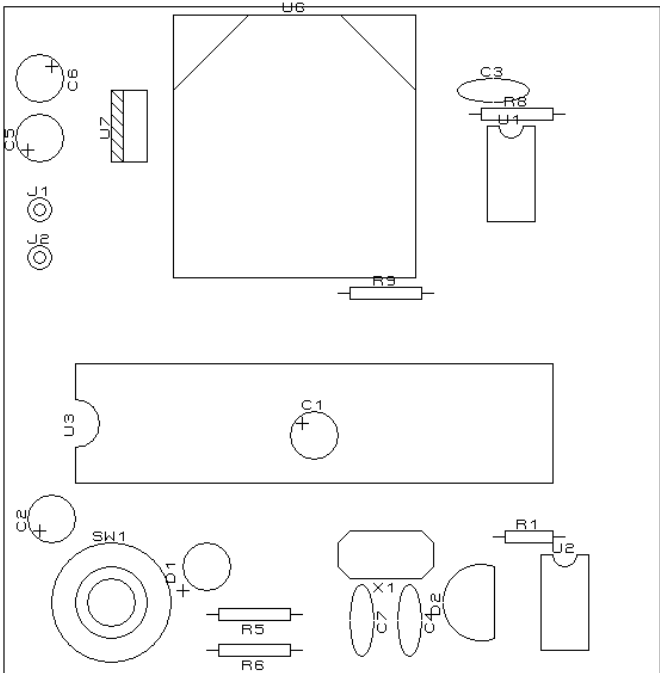
## Bibliographie

[1] <http://fr.wikipedia.org/wiki/Zigbee>

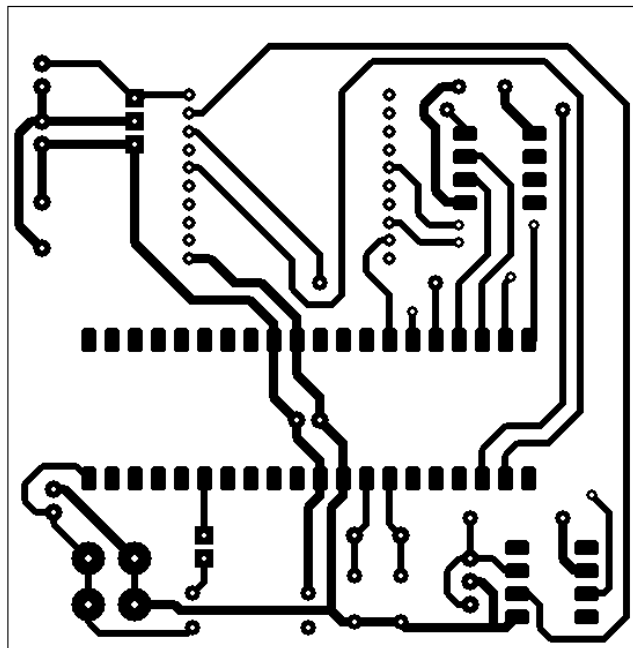
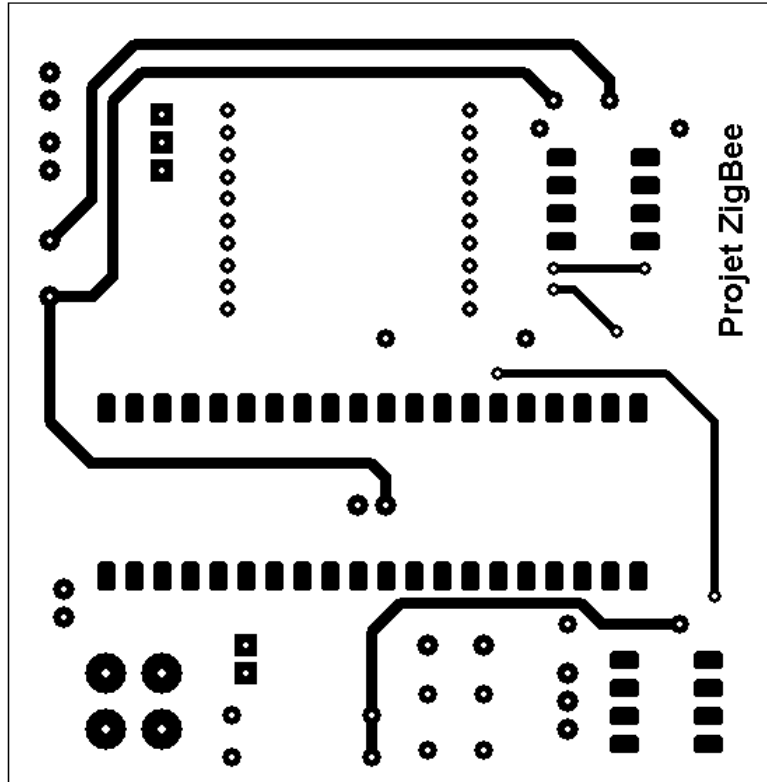
[2] [www.maxstream.net/products/xbee/datasheet\\_XBee\\_OEM\\_RF-Modules.pdf](http://www.maxstream.net/products/xbee/datasheet_XBee_OEM_RF-Modules.pdf) –

**Annexes :**

**Annexe 1 : Implantation des composants**



## Annexe 2: Typons



### Annexe 3 : Librairie DS1620

```

/*****
/***** BEAUSSE - MOKTAD - BOUGUERRA - GUAN *****/
/***** PROJET AVANCE S5 *****/
/*****
/***** Bibliothèque du thermomètre DS1620 *****/
/*****
void ds1620_write8 (unsigned char data);
void ds1620_start (void);
short ds1620_readtemp (void);
unsigned char ds1620_read8 (void);
short ds1620_read9 (void);
void DelayMs(unsigned char cnt);
void Delays(unsigned char cnt);

/***** Initialisation *****/
void ds1620_init (void)
{
    /* *RST haut */
    RC5=1;
    /* écriture de la config */
    /* mode one-shot */
    ds1620_write8(0x0C);
    ds1620_write8(0x03);
    /* *RST bas */
    RC5 = 0;
    /* attente de l'écriture dans la mémopire E2 */
    DelayMs(50);
    ds1620_start();
}

/***** Début de la conversion *****/
void ds1620_start (void)
{
    /* *RST haut */
    RC5=1;
    /* ordre de conversion de la température */
    ds1620_write8(0xEE);
    /* *RST bas */
    RC5 = 0;
}

/***** Lecture de la température *****/
short ds1620_readtemp (void)
{
    short temp=0;
    /* *RST haut */
    RC5=1;
    /* ordre de lecture du PIC */
    ds1620_write8(0xAA);
    /* le ds1620 envoie la température */
    temp = ds1620_read9();
    /* *RST bas */
}

```

```

RC5=0;
return temp;
}

/***** Lecture d'un mot de 8b du ds1620 *****/
unsigned char ds1620_read8 (void)
{
    unsigned char i;
    unsigned char temp=0;
    for(i=0; i<8; i++)
    {
        /* CLK niveau bas */
        RC4=0;
        /* concaténation bit par bit */
        temp = temp + (RC3 << i);
        /* CLK niveau haut */
        RC4 = 1;
    }
    return temp;
}

/***** Lecture d'un mot de 9b du ds1620 *****/
/***** similaire à la précédente *****/
short ds1620_read9 (void)
{
    unsigned char i;
    short temp=0;
    for(i=0; i<9; i++)
    {
        RC4=0;
        temp = temp + ((unsigned char) RC3 << i);
        RC4 = 1;
    }
    return temp;
}

/***** Ecriture d'un mot de 8b sur le ds1620 *****/
void ds1620_write8 (unsigned char data)
{
    unsigned char i;
    /* Port C en sortie pour l'écriture */
    TRISC = 0x00;
    for(i=0; i<8; i++)
    {
        //NOP;
        /* CLK niveau bas */
        RC4=0;
        /* ecriture bit par bit du caractère */
        RC3 = (( (data>>i) &0x01 ) ==0)?0:1 ;
        /* CLK niveau haut */
        RC4 = 1;
    }
    /* Port C en input */
    TRISC = 0x08;
}

```

```

/*****
/***** Les fonctions suivantes servent à établir *****/
/**** la précision à 0.1°C (contre 0.5°C par défaut)*****/
/*****/

/***** Lecture du registre COUNTER *****/
short ds1620_counter (void)
{
    short temp=0;
    /* *RST haut */
    RC5=1;
    /* ordre de lecture du registre counter */
    ds1620_write8(0xA0);
    /* lecture du résultat */
    temp = ds1620_read9();
    /* *RST bas */
    RC5=0;
    return temp;
}

/***** Lecture du registre SLOPE *****/
short ds1620_slope (void)
{
    short temp=0;
    /* *RST haut */
    RC5=1;
    /* ordre de lecture du registre slope */
    ds1620_write8(0xA9);
    /* lecture du résultat */
    temp = ds1620_read9();
    /* *RST bas */
    RC5=0;
    return temp;
}

/*****/

//*****/
// Temporisation de x ms //pour les test cette temporisation est
//trés approximative
//*****/
void DelayMs(unsigned char cnt)
{
    int compteur_ms;
    compteur_ms=0;
    while(compteur_ms!=cnt)
    {
        compteur_ms++;
    }
}

//*****/
// Temporisation de x s //pour les test cette temporisation est
//trés approximative
//*****/
void Delays(unsigned char cnt)

```

```

{
  int compteur_s;
  compteur_s=0;
  while(compteur_s!=cnt*1000)
  {
    compteur_s++;
  }
}

```

## **Annexe 4 : Bibliothèque UART**

```

/*****
/***** BEAUSSE - MOKTAD - BOUGUERRA - GUAN *****/
/***** PROJET AVANCE S5 *****/
/***** Bibliothèque de l'UART *****/
/*****

char RecUART (void);
void EmitUART(char c);
void InitUART(void);

/* Initialisation de l'UART */

void InitUART(void)
{
  TXSTA = 0x20;
  RCSTA = 0x90;
  SPBRG = 0x06;
}

/* Emission d'un octet sur l'UART */

void EmitUART(char c)
{
  TXREG = c;
}

/* Reception d'un Octet sur l'UART */

char RecUART (void)
{
  return(RCREG);
}

```

## Annexe 5 : Bibliothèque XBee

```
void InitXbee(void)
{
    TRISD=0x18;
    PORTD=0x05;
}

void SendByte(char c)
{
    EmitUART(c);
}

char ReadByte(void)
{
    return(RecUART());
}

void SendString(const unsigned char *s)
{
    int i=0;
    while(s[i]!='\0')
    {
        /* utilisation de lcd_putch */
        SendByte(s[i]);DelayMs(100);
        i++;
    }
}
```

## Annexe 6 : programme principal

```
/*
*****
***** BEAUSSE - MOKTAD - BOUGUERRA - GUAN *****
***** PROJET AVANCE S5 *****
***** programme principal *****
*****
*/

#include <pic.h>
#include "lib1620.c"
#include "beelib.h"

__CONFIG (0x3F72);

//short temperature_ext,temperature_int,slope,counter;

/*
***** Quelques puissances de 10 utiles *****
*/
unsigned short puiss10(unsigned char i)
{
    switch(i)
    {
```

```

    case 0: return 1;
    case 1: return 10;
    case 2: return 100;
    case 3: return 1000;
    case 4: return 10000;
    default: return 1;
}
}

/***** Initialisation des ports du PIC *****/
void InitPort(void)
{
    /* thermomètre LM335 + LED (RA4 en sortie)*/
    TRISA = 0x00;
    /* thermomètre DS1620 */
    TRISC = 0x00;
}

/** à partir d'un nombre, renvoie le chiffre *****/
/**      situé à la position voulue *****/
unsigned char conversion(short temp, unsigned char indice)
{
    short aux = ( temp/puiss10(indice) - (temp/puiss10(indice+1))*10 );
    unsigned char res = (unsigned char) aux+0x30;
    return res;
}

/* envoi de la temperature */
void envoi(short temp)
{
    SendByte('\r');DelayMs(100);
    SendString("temperature = ");

    /* signe négatif si température négative */
    if(temp<0)
    {
        SendByte('-');
        DelayMs(100);
    }
    else
    {
        SendByte('+');
        DelayMs(100);
    }

    /* envoi des chiffres de la température un à un */
    SendByte(conversion(temp,2));DelayMs(100);
    SendByte(conversion(temp,1));DelayMs(100);
    SendByte(',');DelayMs(100);
    SendByte(conversion(temp,0));DelayMs(100);
}

```

```

void main(void)
{
    short temperature;
    char counter,slope;
    InitPort();
    InitUART();
    InitXbee();
    ds1620_init();
    ds1620_start();
    Delays(2);
    while(1)
    {
        PORTA=0x00;
        temperature=100*(ds1620_readtemp(>>1)-25;
        counter=ds1620_counter();
        slope=ds1620_slope();
        /* pour ne pas diviser par zéro */
        if(slope==0) slope=1;
        /* calcul final de la température interne */
        temperature=(temperature + (100*slope - 100*counter)/slope)/10;
        /* on relance la conversion du ds1620 en mode one-shot */
        ds1620_start();
        envoi(temperature);
        PORTA=0xFF;
        Delays(5);
    }
}

```