



ENSEIRB

---

# Projet HomeSIP

---

2007-2008

Enseignant :	PATRICE KADIONIK
Étudiants :	WILLY AUBRY, MIKEL AZKARATE-ASKASUA, FABIEN MARTEAU, HANÉN SABEUR
date :	7 février 2008

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Le Projet HomeSIP</b>	<b>6</b>
2.1	Présentation du protocole SIP . . . . .	6
2.2	L'ajout du XML . . . . .	7
<b>3</b>	<b>Les capteurs Xbee</b>	<b>8</b>
3.1	A propos de Xbee . . . . .	8
3.1.1	Caractéristiques de transmission dans les réseaux Zigbee . . . . .	8
3.1.2	Applications Zigbee : . . . . .	9
3.1.3	Spécifications techniques du standard IEEE 802.15.4 . . . . .	10
3.2	Programmation du PIC . . . . .	10
3.2.1	Librairie DS1620 . . . . .	11
3.2.2	Librairie de la liaison série . . . . .	12
3.2.3	Librairie Xbee . . . . .	12
3.2.4	Librairie du protocole de communication . . . . .	12
3.2.5	Programme Principal . . . . .	13
3.3	Maître Xbee : Noeud ARM . . . . .	13
3.3.1	Fonctionnalité . . . . .	13
3.3.2	Mise en œuvre . . . . .	14
3.3.3	Compilation croisée . . . . .	15
3.4	Simulateur de capteurs Xbee . . . . .	15
<b>4</b>	<b>Les capteurs iButton</b>	<b>17</b>
4.1	le bus 1-Wire . . . . .	17
4.1.1	Fonctionnement . . . . .	17
4.2	La Bibliothèque de fonctions libiButton . . . . .	18
<b>5</b>	<b>Structure du code</b>	<b>19</b>
5.1	Les répertoires du projets . . . . .	19
5.2	Structure générale du programme . . . . .	19
<b>6</b>	<b>Conclusion</b>	<b>21</b>

# Table des figures

1.1	Le projet HomeSIP . . . . .	4
3.1	communication entre deux modules xbee . . . . .	9
3.2	structure d'un paquet de donnée . . . . .	10
3.3	Diagramme de séquence de la commande SCAN . . . . .	15
3.4	Simulateur de capteur Xbee . . . . .	16
4.1	Des iButtons . . . . .	17
4.2	Schématique du bus 1-wire . . . . .	18

# Chapitre 1

## Introduction

Les avancées technologiques dans le domaine des réseaux multimedia a fait naitre un nouveau type de d'application appelée la domotique.

Le principe de la domotique est de collecter, transmettre et traiter les informations afin de coordonner les actionneurs en réaction à un stimuli donné. La domotique a aussi pour interet qu'elle peut être commandée en local ou à distance.

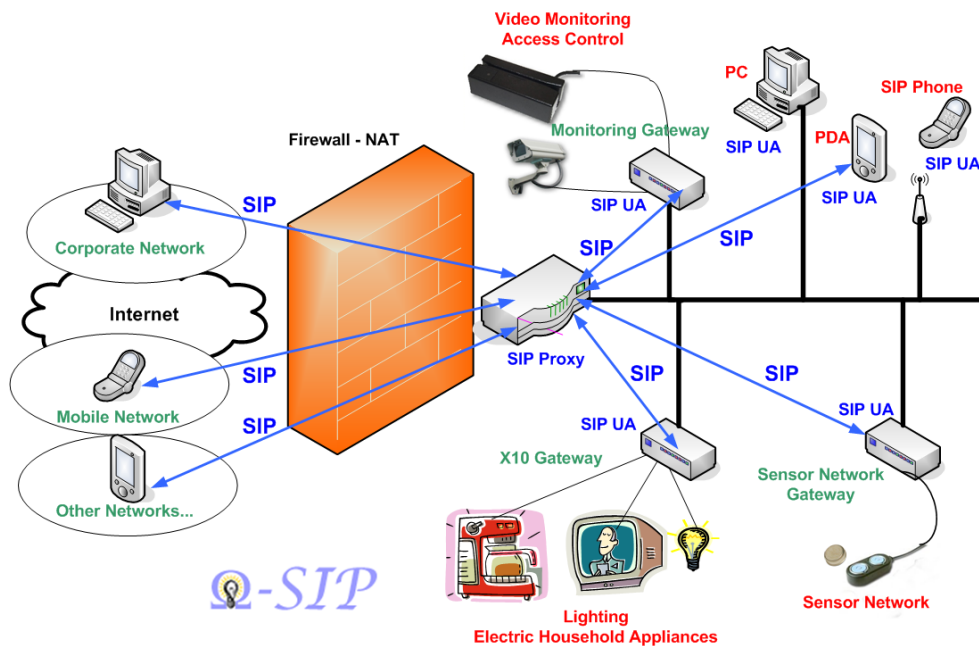


FIG. 1.1 – Le projet HomeSIP

La domotique se retrouve dans différents domaines d'application, que ce soit dans le médical, le militaire, la sécurité publique ou l'automatisation du domicile. Ce nouveau type d'application a besoin d'une nouvelle plateforme pouvant gérer la coordination entre les différents périphériques (capteurs, caméras, etc), appelé infrastructure de communication.

Cette infrastructure administre tout les aspects de la communication entre les entités et l'application. Elle doit prendre en considération l'hétérogénéité des composants et les

différentes fonctionnalités réseaux utilisées.

L'infrastructure développée se base sur l'utilisation du protocole SIP (Session Initiation protocole) qui a été implémenté dans les langages de programmation tel que le C, le java et le C#.

Le protocole SIP est un protocole utilisé pour démarer, modifier et terminer des sessions avec un ou plusieurs participants. Il peut être utilisé pour créer des sessions à deux ou plusieurs intervenants tels que la téléphonie via internet, la distribution multimédia et les visio-conférences. Ce protocole peut être facilement transposé à la domotique dans le domaine de la communication machine-machine.

# Chapitre 2

## Le Projet HomeSIP

### 2.1 Présentation du protocole SIP

Le protocole SIP est un protocole situé au niveau de la couche applicative. Ce protocole a été créé afin de développer la téléphonie sur IP. Pourtant le protocole ne se soucie pas de l'information qu'il transporte. Il peut donc être utilisé pour toute application qui a besoin d'initialiser une session. Les clients SIP utilisent le TCP ou l'UDP pour se connecter au serveur SIP. Le protocole SIP se concentre principalement sur l'initialisation d'une session entre deux ou plusieurs utilisateurs.

Dans la domotique, nous pouvons distinguer deux types de transferts de données :

**Transfert Synchrone :** Une donnée est demandée par l'utilisateur (par exemple une demande de température) ou l'utilisateur peut souhaiter de réaliser une action (par exemple allumer le chauffage).

**Transfert Asynchrone :** Une alarme est lancée lorsqu'il y a une anomalie ou un niveau dangereux (un pic de température dans la cuisine, une présence alors que personne ne devrait être dans la pièce)

Afin de réaliser ces deux types de transfert le protocole SIP offre les services suivants :

**MESSAGE :** Permet l'envoi d'un message simple, utilisé pour le transfert synchrone de données.

**SUBSCRIBE/NOTIFY :** Permet de s'abonner (SUBSCRIBE) à une liste afin d'être prévenu (NOTIFY) en cas d'alarme.

Les réseaux de capteurs peuvent aussi être basés sur le protocole SNMP qui possède les mêmes fonctionnalités. Mais, l'utilisation du protocole SIP permet de simplifier les requêtes, de les sécuriser, et de prendre une empreinte mémoire plus faible que le protocole SNMP. Enfin, de nombreux clients ont déjà été développés pour le protocole SIP ce qui n'est pas le cas pour le protocole SNMP.

## 2.2 L'ajout du XML

Le SIP est un protocole mettant en jeu la gestion de session mais ne gère pas les commandes que l'on veut passer. Nous souhaitons dialoguer entre serveur et clients, pour cela nous avons besoin d'établir une norme. Cette norme sera établie en utilisant le langage XML.

Le choix de l'XML a été fait car connu à travers le monde et donc facile à mettre en oeuvre que ce soit pour le créer ou le lire.

Deux commandes ont été implémentées :

- askvalue : prenant en argument l'ID du capteur et retournant la valeur lue sur le capteur, le type de capteur ainsi que la date d'acquisition de la donnée.
- scannetwork : scan le réseau de capteur et retourne la liste de tous les capteurs présents sur le réseau.

Ces deux commandes sont appelées via l'xml de la façon suivante : askvalue : l'appel se fait de la manière suivante :

```
<askvalue>
<id>1234</id>
</askvalue>
```

le retour se fait de la manière suivante :

```
<ansvalue>
<id>1234</id>
<value>12</value>
<type>temperature</type>
<acquisition_date>543</acquisition_date>
<ansvalue>
```

scannetwork : L'appel se fait de la manière suivante :

```
<scannetwork>
</scannetwork>
```

le retour se fait suivant le format :

```
<ansscan>
<capteur>
<id>1234</id>
<type>temperature</type>
</capteur>
<capteur>
...
</capteur>
</ansscan>
```

# Chapitre 3

## Les capteurs Xbee

### 3.1 A propos de Xbee

Le standard 802.15.4 est ratifié depuis Août 2003 par l'IEEE. Il est défini de manière à devenir le standard global des réseaux de surveillance et de contrôle. Ses caractéristiques sont principalement la faible consommation d'énergie et le faible coût. Avec un débit ne dépassant pas 250 Kbps, le Zigbee est moins rapide que le Bluetooth mais il est conçu pour des connexions sur un rayon de portée de 30 mètres avec une consommation minimale d'énergie, de manière à ce que les batteries puissent fonctionner sur une longue période (jusqu'à quelques années suivant l'application). Le nom Zigbee est attribué par «Zigbee Alliance» qui est une association de 50 entreprises travaillant à la mise en œuvre et l'utilisation du standard. Son nom veut dire réseau en zigzag.

En effet, malgré qu'un tel réseau puisse fonctionner en étoile ou en point à point, la topologie de base est maillée. Ceci permet un trajet multiple entre deux dispositifs de ce réseau. Ceci est important puisqu'en cas de panne d'un nœud du réseau, deux autres nœuds peuvent faire un zigzag à l'intérieur du réseau pour être connectés.

#### 3.1.1 Caractéristiques de transmission dans les réseaux Zigbee

Les terminaux Zigbee sont généralement alimentés en énergie par des piles. La durée de vie de ces piles doit être la plus grande possible ce qui signifie que la consommation d'énergie doit être réduite autant que possible. La meilleure solution pour accomplir cet objectif est d'avoir des dispositifs qui sont en veille s'ils ne sont pas en communication. La distance de transmission entre deux terminaux peut atteindre 20 à 50 mètres.

Zigbee définit les couches réseau, sécurité et application. La couche réseau supporte trois topologies : en étoile, en arbre et maillée. La topologie en étoile donne l'avantage de la consommation minimale d'énergie, alors que la topologie maillée donne l'avantage de la fiabilité de transmission en définissant plusieurs trajets entre deux points quelconques dans le réseau.

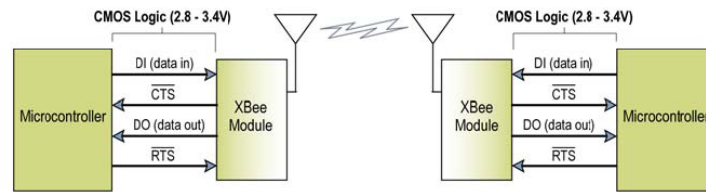


FIG. 3.1 – communication entre deux modules xbee

La topologie la plus fréquemment utilisée est celle en maille. Afin de garantir une consommation minimale de puissance, la spécification du standard distingue deux types de périphériques à savoir ceux à fonctions réduites (RFD) et ceux à fonctions complètes (FFD). Un réseau Zigbee nécessite au minimum un FFD. Ce dernier peut jouer le rôle d'un coordinateur de réseau ou de connexion ou juste un nœud de communication.

Les FFD peuvent communiquer entre eux et avec les RFD pour établir des connexions. Ils sont généralement alimentés par des lignes de courant. Par contre, les RFD sont implémentés avec des ressources mémoire limitées, alimentées par des batteries et sont conçus pour être des nœuds d'émission/réception simples. Ces périphériques peuvent chercher les réseaux disponibles, transférer des données et être en veille hors des communications afin de minimiser la consommation d'énergie.

Les modules Xbee que nous avons utilisés sont associés à des kits qui possèdent une interface RS-232 ou une interface USB. Ainsi ils peuvent être connecté à n'importe quel système (ex :  $\mu C$ , ordinateur) qui possède une interface compatible La figure 3.1 présente un exemple de communication entre deux modules Xbee.

Les entités logiques d'un réseau Zigbee sont essentiellement les coordinateurs, les routeurs et les nœuds terminaux. Un coordinateur initialise les connexions et gère les informations des nœuds du réseau. Un routeur Zigbee participe dans le réseau en routant les messages entre les nœuds. Un nœud terminal peut être un RFD ou un FFD. Dans notre projet on utilise un seul coordinateur et le reste des modules sont des nœuds.

### 3.1.2 Applications Zigbee :

La spécification initiale de Zigbee propose un débit faible avec un rayon d'action relativement limité. Mais, en revanche, sa fiabilité est assez élevée, son prix de revient est faible, et sa consommation est 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.

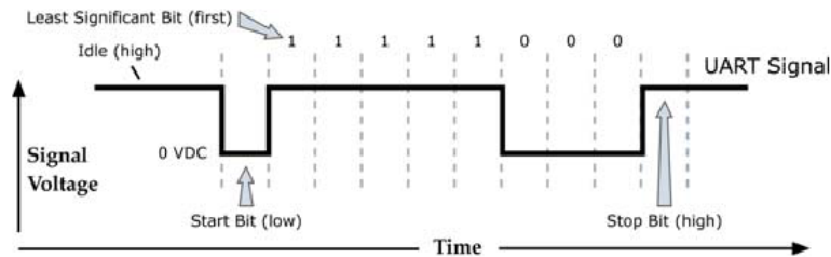


FIG. 3.2 – structure d'un paquet de donnée

Dans une maison, le Zigbee est utilisé pour la gestion de l'énergie et les fonctions de confort avec des liens radio. Les exemples incluent le thermostat, le chauffage, ventilateurs et climatiseurs, le contrôle de fenêtres, etc. Le Zigbee est aussi utilisé pour les systèmes de mesure automatiques (électricité, eau, gaz, etc.).

Parmi les applications les plus utilisées avec Zigbee, les systèmes de contrôle d'éclairage sont une application censée avoir un essor dans le marché de ces réseaux. En effet, Zigbee permet des connexions sans fils entre commutateurs, lampes et les alimentations de puissance. Zigbee peut aussi être employé dans le domaine agricole. En effet, des opérations de mesure de l'humidité, de la température et d'autres indicateurs d'environnements sont réalisées en utilisant les réseaux Zigbee. Enfin, une application intéressante des réseaux Zigbee concerne les systèmes d'alarme, de surveillance, contrôle d'accès, de détection de fuites d'eau, etc.

### 3.1.3 Spécifications techniques du standard IEEE 802.15.4

Le standard IEEE802.15.4 peut opérer dans deux bandes ISM, la bande des 2.4 GHz utilisable dans le monde entier et les bandes 868 MHz pour l'Europe et 915 MHz pour l'Amérique du nord. Le spectre est étalé par la technique de séquence directe avec des codes de différentes longueurs suivant la bande d'opération. La norme prévoit des débits de données de 250 kbps pour la bande 2.4 GHz et de 20 kbps et 40 kbps pour les deux autres bandes en Europe et en Amérique du nord respectivement. Les modules Xbee que nous avons utilisés lors de ce projet utilisent la bande centrée sur la fréquence 2.4 GHz.

Voyons maintenant les spécifications détaillées du module Xbee.

Chaque paquet de données a la structure représentée figure 3.2. Un start bit pour indiquer le début du paquet transmis. Un champ de données de taille 8 bits. Un stop bit qui indique la fin du paquet. Le signal est au niveau haut quand il n'y a aucun paquet à transmettre.

## 3.2 Programmation du PIC

On s'intéresse dans cette partie à programmer le PIC afin de pouvoir assurer la communication entre le capteur de température le DS1620 et le module Xbee.

Ce dernier fonctionne par défaut en mode transparent c'est-à-dire que tout paquet reçu

à travers le pin DI est envoyé vers la transmission RF et quand le paquet est reçu les données sont transmises à travers le pin DO. Il suffit ainsi d'écrire sur le pin DI pour transmettre des données RF.

Ceci dit, on peut avoir besoin d'envoyer des commandes au module même. Il s'agit du mode API. Dans ce mode toutes les données reçues ou envoyées par le module sont des opérations ou des événements à l'intérieur du module. C'est le cas par exemple des trames de commande AT. Ces paquets servent à changer les paramètres du module comme son adresse ou l'adresse de destination. Dans les deux cas on a besoin de programmer le PIC pour communiquer avec le module Xbee suivant que l'on veut le configurer ou l'utiliser pour communiquer avec le coordinateur. Pour ce faire on écrit les bibliothèques suivantes.

### 3.2.1 Librairie DS1620

Cette librairie est développée afin d'assurer la communication entre le  $\mu$ C PIC et le capteur de température DS1620. Cette librairie présente les fonctions suivantes :

- Ds1620\_write8 : Cette fonction permet l'écriture vers le composant. Elle permet ainsi de le configurer, lui demander d'effectuer une conversion ou d'envoyer la valeur de la température. Il faut ainsi écrire huit bits. On commence par configurer le port C en sortie, puis pour chaque bit, il faut baisser l'horloge(RC4), envoyer la valeur du bit (RC3) puis remettre l'horloge à 1. Finalement on remet le port C en entrée (état par défaut). Par conséquent on a besoin de lire la valeur de température envoyée par le capteur. D'où l'intérêt des deux fonctions.
- ds1620\_read8 et ds1620\_read9 : Par similitude avec la fonction écriture, on baisse l'horloge, on reçoit le bit envoyé par le capteur et finalement on remet l'horloge au niveau haut. On refait les mêmes opérations autant de fois que le nombre de bits du message envoyé par le ds1620 (8 ou 9 bits). Avec ces trois fonctions on peut initialiser, configurer et commander le capteur.
- Ds1620\_init : Cette fonction permet de configurer le ds1620 afin qu'il effectue une seule conversion. Il faut ainsi mettre le reset (RC5) au niveau haut, écrire le caractère 0x0C pour indiquer au capteur qu'il va recevoir une commande suivi de 0x03 qui est l'octet du mode one-shot et on baisse le reset. Finalement on attend 50ms le temps de sauvegarde de cette configuration.
- Ds1620\_start : cette fonction est appelée après chaque rafraîchissement de la température du main. On baisse le reset, on écrit le caractère 0xEE pour indiquer le début de conversion et enfin on remet le reset à zéro.
- Ds1620\_readtemp : cette fonction sert à lire la température envoyée par le capteur. On met le reset à 1, on donne l'ordre d'envoi de la valeur en écrivant 0xAA, on lit la valeur grâce à la fonction ds1620\_read9 et enfin on remet le reset à zéro.
- Ds1620\_counter, ds1620\_slope : Ces deux fonctions permettent d'avoir plus de précision sur la valeur de température convertie. On cherche ainsi à récupérer les deux valeurs stockées dans les deux registres 'counter' et 'slope'. Pour ce faire, on procède de la même manière que pour la fonction ds1620\_readtemp sauf qu'on remplace le caractère 0xAA par 0xA0 pour la fonction ds1620\_counter et 0xA9 pour la fonction ds1620\_slope. Avec ces deux valeurs et la valeur de la température envoyée par le capteur on arrive à faire plus de calcul afin d'avoir une valeur plus précise.

### 3.2.2 Librairie de la liaison série

Comme son nom l'indique, cette librairie sert à communiquer avec le module Xbee par la liaison série. Cette librairie contient quatre fonctions :

- `UartInit()` : Elle sert à initialiser la liaison série. On configure le registre SPBRG afin de fixer le baudrate à 2400, puis les deux registres TXTSA et RCSTA afin de choisir une liaison asynchrone.
- `EmitUart(char c)` : On écrit sur le registre TXREG pour pouvoir envoyer un octet du  $\mu$ C au module Xbee. On attend ensuite que l'octet soit envoyé en testant le 5ème bit du registre PIR1 qui doit passer à 1 quand l'envoi est terminé.
- `RecUart()` : Lorsqu'un octet est reçu, le flag RCIF se met à 1. Dans ce cas, il suffit de lire le registre RCREG pour lire l'octet reçu.
- `SendString(const unsigned char *s)` : Cette fonction sert à envoyer une chaîne de caractères. Il suffit d'utiliser la fonction `EmitUart(char c)` pour chaque caractère de la chaîne.

### 3.2.3 Librairie Xbee

Cette librairie utilise les fonctions qui ont été développées dans la librairie de la liaison série. Comme déjà expliqué, le module Xbee peut fonctionner en deux modes : celui par défaut qui est le mode transparent (Tout paquet reçu à travers le pin DI est envoyé vers la transmission RF). Il suffit ainsi d'écrire sur le pin DI pour transmettre des données RF. Le deuxième mode est le mode API. Toutes les données reçues ou envoyées par le module sont des opérations ou des événements à l'intérieur du module. C'est le cas des trames de commande AT :

- `XBeeInit()` : Cette fonction sert à initialiser le module Xbee en initialisant les pins du port D connectées au module.
- `SendByte(char c)` : Elle sert à envoyer un caractère au module Xbee via la liaison série. On utilise ainsi la fonction `EmitUart()` de la librairie série.
- `ReadByte()` : Elle sert à lire un caractère envoyé par le module Xbee vers le PC. Là aussi on utilise la librairie de la liaison série.
- `EnterConfiguration ()` : Cette fonction sert à envoyer trois fois le caractère '+' au module Xbee. En recevant ces trois caractères, le module s'attend à une commande AT. D'où l'intérêt de la fonction qui suit.
- `EnterCommand (char *cmd, char *param)` : Cette fonction sert à envoyer une commande AT. Ainsi on commence par envoyer la chaîne "AT" suivie par la chaîne cmd qui présente la commande à exécuter suivie par le paramètre associé. Par exemple : ATCE 1 qui sert à configurer le module en tant que coordinateur.
- `TestOK ()` : Cette fonction teste si le caractère émis à bien était transmis en testant le sixième bit du registre PIR1 qui doit être à zéro.

### 3.2.4 Librairie du protocole de communication

Cette librairie sert à définir le protocole de communication entre les modules Xbee. On commence ainsi par définir une structure "msg" qui est composée de deux caractères. Le

premier est le “id” qui est l’identifiant du message échangé. Le deuxième est “len” qui définit la longueur du message. Une chaîne de caractères fait aussi partie de cette structure. Elle est associée à la commande ou la réponse envoyée. Le tableau qui suit résume l’ensemble des messages envoyés.

ID	LENGTH	REQUETE/REPONSE
REQTYPE	1	vide
REQLECTURE	1	vide
REQCHANGE	4	vide
RPSTYPE	1-5	TYPESENSORPIC
RPSLECTURE	5	TEMPERATURE
RPSCHANGE	2	0/1

- SendMsg (msg \* trame) : Cette fonction sert à envoyer un message au coordinateur. Ce message est une réponse à la requête envoyée par ce dernier. Il peut le renseigner sur son type( température, présence ou humidité) ou lui envoyer la valeur de la température mesurée par le capteur DS1620.
- ReceiveMsg (msg \* trame) : Cette fonction sert à recevoir la requête envoyée par le coordinateur. Cette requête peut être une demande du type de capteur associé au module Xbee, une demande de la valeur de température mesurée ou une commande AT comme par exemple la commande “ATCN” qui sert à scanner le réseau. On reçoit la chaîne envoyée par le coordinateur qui se termine par le caractère ‘Z’.
- TraiteMsg (msg \* trame) : Cette fonction sert à traiter les messages reçus par le module Xbee. Suivant l’identifiant du message, on envoie la bonne réponse.(voir les deux fonctions précédentes)

### 3.2.5 Programme Principal

On commence tout d’abord par appeler les fonctions d’initialisations de toutes les bibliothèques afin d’initialiser notre système. Ensuite on attend que le module Xbee reçoive une requête de la part du coordinateur. Si c’est le cas on traite le message reçu avec la fonction TraiteMsg() et on lui envoie la réponse avec la fonction SendMsg().

## 3.3 Maître Xbee : Noeud ARM

### 3.3.1 Fonctionnalité

Le responsable de la gestion du réseau Xbee est le noeud ARM, qui prend les commandes SIP et peut exécuter les commandes suivantes :

- Xbee\_Scan : Remplissage d’une liste avec les IDs des capteurs Xbee disponibles.
- Xbee\_DemandeType : Demande du type de capteur (température, présence, etc.) pour un ID donné.

- `Xbee_Lecture` : Demande de la valeur d'un capteur.

### 3.3.2 Mise en œuvre

Pour le développement de cette fonction on a réutilisé la bibliothèque Xbee que on a créé pour la gestion de PIC. Dans cette bibliothèque, on exploite le Firmware des émetteurs Xbee pour la gestion des IDs et le `SCAN` du réseau Xbee car ces fonctions sont déjà prévues dans le Firmware.

Comme on a vu dans la section 3.2, on peut rentrer en mode configuration des transmetteur Xbee et appeler ses commandes internes. Pour pouvoir entrer à ce Firmware interne, on doit écrire un `'+++'` dans la ligne série et attendre un `'OK'` pour assurer qu'on a bien entré. Au noeud ARM on utilise les suivants commandes :

- `'ATDL' + 'ID'` : Changement de l'ID de destination. On l'utilise pour changer l'ID chaque fois que on veut s'adresser à un capteur différent :

```
+++OK
atdl1
OK
atdl
1
```

- `'ATND'` : À la exécution de cette commande le transmetteur Xbee retourne des informations des capteurs Xbee présents. Il y a plusieurs informations sur chaque capteur mais on ne récupère que l'ID (la première information). L'ordre d'affichage est par proximité, donc le transmetteur le plus proche est le premier à être affiché.

```
+++OK
atnd
1
13A200
40086108
28
Y

3
13A200
400418C7
17
xx
```

Les transmetteurs Xbee sont manipulés comme des lignes séries, il suffit donc d'utiliser la librairie précédemment décrites.

Le diagramme de séquence pour la fonction `Xbee_Scan` est visible figure 3.3.

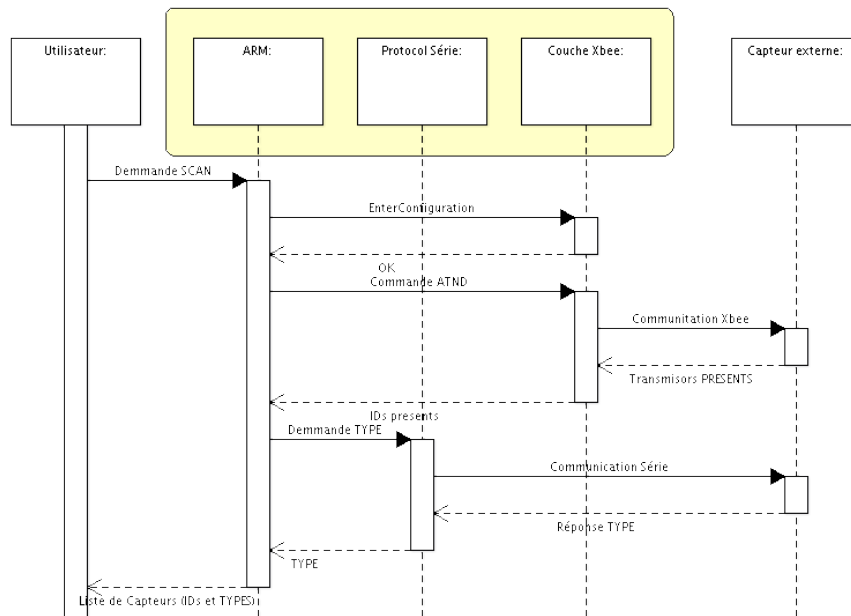


FIG. 3.3 – Diagramme de séquence de la commande SCAN

### 3.3.3 Compilation croisée

Pour finir nous avons fait de la compilation croisée du programme développé et nous l'avons intégré sur la carte ARM.

Nous avons développé une bibliothèque qui contient toutes les fonctions Xbee développés pour la carte ARM (`libxbee.a`). Pour générer la bibliothèque, il suffit de se rendre dans le répertoire `SE/target/libxbee/Xbee/` et de faire un `make`. Le Makefile générera aussi un binaire de test de la bibliothèque `testlibxbee`, qu'il suffit de télécharger sur la cible pour tester.

## 3.4 Simulateur de capteurs Xbee

Pendant le projet on a vu l'intérêt d'avoir plusieurs capteurs au moment de tester et valider le logiciel développé. Pour ça, nous avons pensé à intégrer un transmetteur Xbee sur un PC et créer un logiciel qui fait la simulation de capteurs sur un environnement windows.

Le logiciel avait besoin d'un driver série et d'une interface graphique réalisé sur *Visual Studio .NET* en langage *Visual Basic*.

Pour la lecture périodique de la ligne série nous avons utilisé un timer logiciel qui voit s'il y a une nouvelle demande et envoie le type ou la lecture du capteur. Le code du projet se trouve au fichier `sensors.vb`.

Le résultat est visible sur la figure 3.4 :

À gauche on voit une partie pour la configuration de la communications série. Au centre trois types de capteurs : température, humidité et présence. Et à droite un historique du

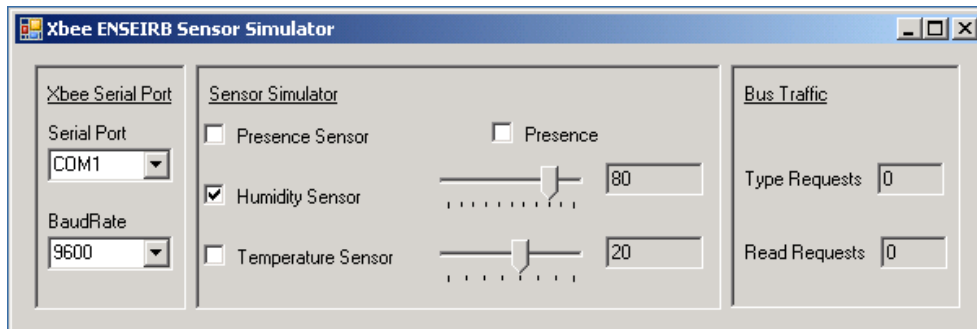


FIG. 3.4 – Simulateur de capteur Xbee

nombre de demandes et leur nature.

## Chapitre 4

# Les capteurs iButton

Les iButton sont des microcontrôleurs spécifique développés par *Dallas Semiconductor*. Ces composants se présentent sous la forme de capsules ressemblant à des piles boutons. Leurs alimentations ainsi que le bus de communication passe par les même deux fils : le bus *1-Wire*.

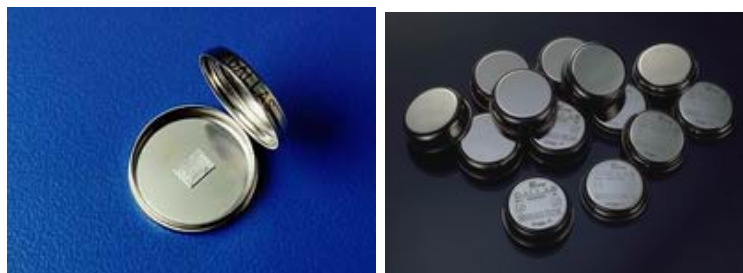


FIG. 4.1 – Des iButtons

Il existe une multitude de iButtons<sup>1</sup> remplissant différentes fonctions comme la mesure de la température, l'identification, des capteurs diverses et variés, l'enregistrement de mesures, ...

Pour ce projet nous avons utilisé deux iButtons capteurs de température *DS1920*.

### 4.1 le bus 1-Wire

Le *1-Wire* est défini comme un bus de communication en 1 fil (plus la masse). Il a été développé par *Dallas Semiconducteur* pour fournir une méthode de connexion simple avec alimentation intégré pour des capteurs.

#### 4.1.1 Fonctionnement

Chaque composant *1-Wire* possède une adresse unique sur 16 bits, ce qui permet de l'identifier de manière certaine. La communication est initié par le maître (généralement

<sup>1</sup>Voir sur le site de MAXIM : <http://www.maxim-ic.com/products/ibutton/>

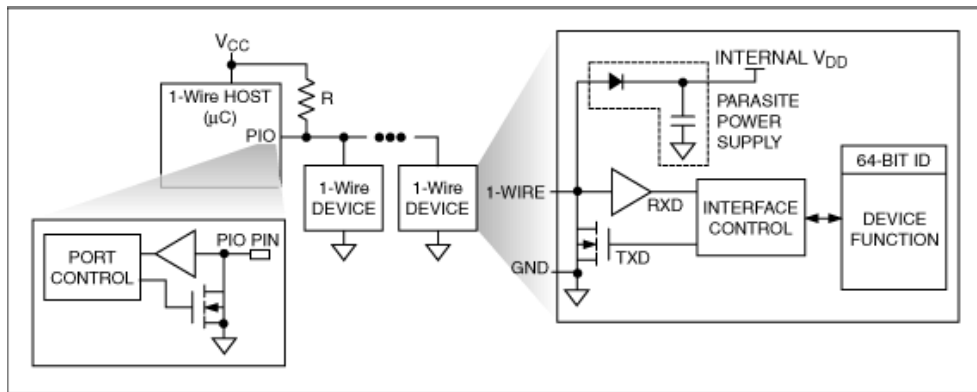


FIG. 4.2 – Schématisation du bus 1-wire

une interface connecté à un ordinateur) en utilisant cet identifiant unique.

Pour alimenter un composant *1-Wire* deux méthodes sont possible :

- Externe : Un troisième fils est utilisé pour alimenter le composant en 5V.
- Parasité : Par cette méthode, le composant va s'alimenter en utilisant la broche de donnée DQ du bus. Grâce à cette méthode, il suffit de brancher deux fils pour faire fonctionner le composant.

Les iButtons utilisent le mode d'alimentation parasité, ce qui leurs permettent de n'utiliser que deux fils de connection.

## 4.2 La Bibliothèque de fonctions libiButton

La bibliothèque libiButton se trouve dans le répertoire `libiButton/` et contient les fonctions permettant de manipuler des capteurs sur iButton.

Pour l'instant, les deux seules fonctions qui ont été implémentés sont :

- `iButton_scan`, qui permet de scanner le bus pour détecter la présence de capteurs et retourne les capteurs présents sous la forme d'une liste.
- `iButton_temperature`, cette fonction permet de lire la température d'un iButton à partir d'un identifiant passé en paramètre.

Comme nous avons que des capteurs de température iButton DS1920 la seule fonction qui ai été implémenté est la fonction de lecture de température, pour que la bibliothèque soit complète il serait nécessaire de développer les fonctions de configuration des seuils ainsi que la fonction permettant de détecter le dépassement de ces seuils.

Les capteurs DS1920 possède un registre de configuration permettant de définir des seuils de températures haut et bas. Il est donc possible de régler ces seuils pour lever des alarmes via le protocole SIP.

# Chapitre 5

## Structure du code

### 5.1 Les répertoires du projets

La structure du projet est divisée en quatre répertoires :

- **Documents** : Ce répertoire contient de la documentation sur le projet.
- **rapport** : Ce rapport et les sources en  $\text{\LaTeX}$ .
- **host** : Dans ce répertoire se trouve un petit programme de test d’envoi de requêtes simple pour tester le réseau de capteur. Pour le lancer faire :  
`./interface_host sip:guest@linux01 sip:root@arm01$`.
- **target** : c’est dans ce repertoire que se trouve les sources principales du projet. Il est divisé en 6 répertoires :
  - **server** : C’est dans ce répertoire que se trouve le programme principale.
  - **xmlParser** : Le code du parser xml.
  - **iButton** : Code fourni par Dallas<sup>©</sup> pour lire les iButtons.
  - **libcapteurs** : Librairie générique permettant de manipuler les capteurs xbee et ibutton.
  - **libiButton** : Librairie spécifique aux iButtons.
  - **libxbee** : librairie spécifique aux capteurs sur xbee.

Chaque librairie contient un code de test<sup>1</sup>, il suffit de faire un make pour les compiler, l’exécutable est automatiquement copié dans le répertoire `/tftp/` qui permet de transférer le fichier vers la cible pour les tests au moyen de la commande `get`<sup>2</sup>.

### 5.2 Structure générale du programme

L’architecture générale du programme s’articule autour du `serveur`/. La fonction `main()` lance le thread `Sipevent()` qui se charge d’attendre une requête SIP d’un client.

Dès que `Sipevent()` reçoit une requête SIP il utilise la fonction `traiter_message()` qui se charge de traduire les commandes XML au moyen de `xmlParser`. Puis en fonction des

<sup>1</sup>testiButton pour libiButton, et testxbee pour libxbee

<sup>2</sup>Petit script permettant d’éviter d’avoir à retaper la commande longue comme le bras de tftp

commandes passées, elle utilise les fonctions généralistes de `libcapteur.a`. Pour chaque type de bus (Xbee ou iButton) les fonctions de la librairie `libcapteur` utilisent la librairie correspondant au bon bus `libiButton` et `libxbee`.

# Chapitre 6

## Conclusion

Ce projet nous a permis de pointer les différentes problématiques de la fusion de plusieurs projets que sont le développement d'une pile SIP et la création d'un capteurs de température sur protocole ZigBee ; projets qui furent développés par nos prédécesseurs durant leurs projets avancés.

La fusion de ces deux projets a conduit à la séparation de notre équipe en deux groupes. Le premier groupe fut chargé de la conception et de la programmation du protocole ZigBee alors que le second eut la responsabilité de l'implémentation de la grammaire XML pour la communication SIP.

Une telle organisation du groupe de projet à nécessité une parfaite collaboration entre les différents membres de l'équipe. De plus, la présence d'un projet de réalisation d'un client HomeSIP a requis une bonne communication entre les deux équipes.

Nous avons créé les bases d'une plate-forme HomeSIP sur processeur ARM. La communication entre un client et le serveur est effective et permet de lire la température sur différents capteurs de température ZigBee<sup>tm</sup> et iButton<sup>©</sup>.

Dans un futur développement, il serait nécessaire d'implémenter la gestion des alarmes ainsi que de faciliter l'ajout de nouveaux types de capteurs.

# Bibliographie

- [1] Mohamed ABID, Anne CHEBROU, Adrien FAVOT, Sébastien ELLERO, and Stéphane KOEHLER. Mise en œuvre d'une pile sip sur un système embarqué pour le contrôle de capteurs. Technical report, ENSEIRB, 2007.
- [2] Bernard Desgraupes. *LaTeX, Apprentissage, guide de référence*. Vuibert, 2003.
- [3] Mathieu GIL, Yacin CHANTIT, and J-H DUBOURG. Projet sip. Technical report, ENSEIRB, 2006.
- [4] Christopher Hallinan. *Embedded Linux Primer*. Prentice Hall Pearson Education, 2007.
- [5] Ouakil Laurent and Pujolle Guy. *Téléphonie sur IP*. EYROLLES, 2007.
- [6] Mesures. Les liaisons radio, principes de base. *Mesure 800*, 2007.
- [7] P.Kadionik. Homesip project page. <http://www.enseirb.fr/cosynux>, 2008.