Maison Communicante

RAPPORT DE PROJET IMA4 – PROJET N°18

Florian Royer & Jérémy Gondry | Projet IMA4 | 15 avril 2014

Remerciements à M. Boé Alexandre qui nous a conseillé et suivi notre projet. Un grand merci également à M. Flamen Thierry pour ses conseils riches d'enseignements en matière d'électronique et de montage de circuits imprimés. Enfin, merci à Xavier Redon pour son aide lors de quelques débogages qui nous ont fait gagné un temps précieux.

Sommaire

I) Cahier des charges et système global 4

A) Cahier des charges 4

B) système global 4

II) Sous-système IHM Web 6

A) Cahier des charges spécifique 6

B) Présentation de l'ihm 6

1) Page d'accueil 6

2) Programmeur 8

3) Panneau de configuration 8

C) Base de données 10

D) bibliothèque parser.c pour le prog principal 11

III) Sous système Réseaux de capteurs 11

A) Description du réseau : 11

B) Les différents composant de ce réseau : 11

1) Modules capteurs 11

2) Modules chauffage 12

3) La raspberry. 12

C) La communication dans le réseau. 13

D) Communication en API 14

E) Conception de la carte Ultra-basse tension 15

F) Conception de la carte basse tension 17

G) Problèmes rencontrés : 19

1) La programmation de l'atmega. 19

2) Le capteur de température : 21

3) Lecture série 21

4) Problème Carte basse tension 22

Introduction

Dans le cadre actuel des nouvelles technologies, des objets connectés et de l'optimisation des consommations d'énergies, il est pertinent de penser qu'il est possible contrôler nos chauffages à la maison automatiquement. La consommation énergétique d'une maison ou d'une entreprise est souvent un gros problème car on est la plupart du temps en manque de moyens techniques ou de systèmes assez modulaires pouvant s'adapter à tout type d'installation. A terme, ce projet pourrait être joint au projet 43 initié par la direction de Polytech afin de réduire la consommation énergétique des radiateurs de l'école. A titre d'exemple, Polytech chauffe les salles jour et nuit, notre devrait permettre de chauffer les salles qui sont utilisées uniquement à des heures prévues, et ainsi économiser au moins 40% de l'énergie.

Nous allons vous présenter une solution commercialement envisageable mis en place dans le cadre d'un projet en 4ème année à Polytech. Cette solution répond au contexte de développement durable de notre ère. Elle consiste à automatiser les déclenchements de nos radiateurs grâce à des relevés de température et à la présence ou non dans la maison.

Nous allons dans un premier temps présenter le système global d'un point de vue macroscopique. Puis, diviser le système en plusieurs sous parties présentées une à une, la partie réseaux et mise en place du réseau permettant aux différents périphériques de communiquer et la partie centrale qui stock les données récupérées et les traite.

I) Cahier des charges et système global

A) CAHIER DES CHARGES

Plusieurs objectifs finaux sont à considérer :

- On compte principalement réduire la consommation due au chauffage dans une habitation ou infrastructure ;
- Les températures devront être récupérées dans plusieurs pièces à l'aide d'un réseau de capteurs.
- Ces données seront récupérées par un dispositif centralisé communicant en Ethernet.
- Ce dispositif servira à élaborer et transmettre, à distance, les commandes aux dispositifs de chauffage.
- La régulation opérée sur la température, sera modulaire, on pourra étendre la commande à différents types de chauffage (sans modifier fondamentalement le programme embarqué sur le dispositif)
- Elles seront classées selon une structure de donnée adaptée.
- Une Interface Homme Machine sera développée permettant la supervision du système.

B) SYSTEME GLOBAL

Voici un schéma du système global.

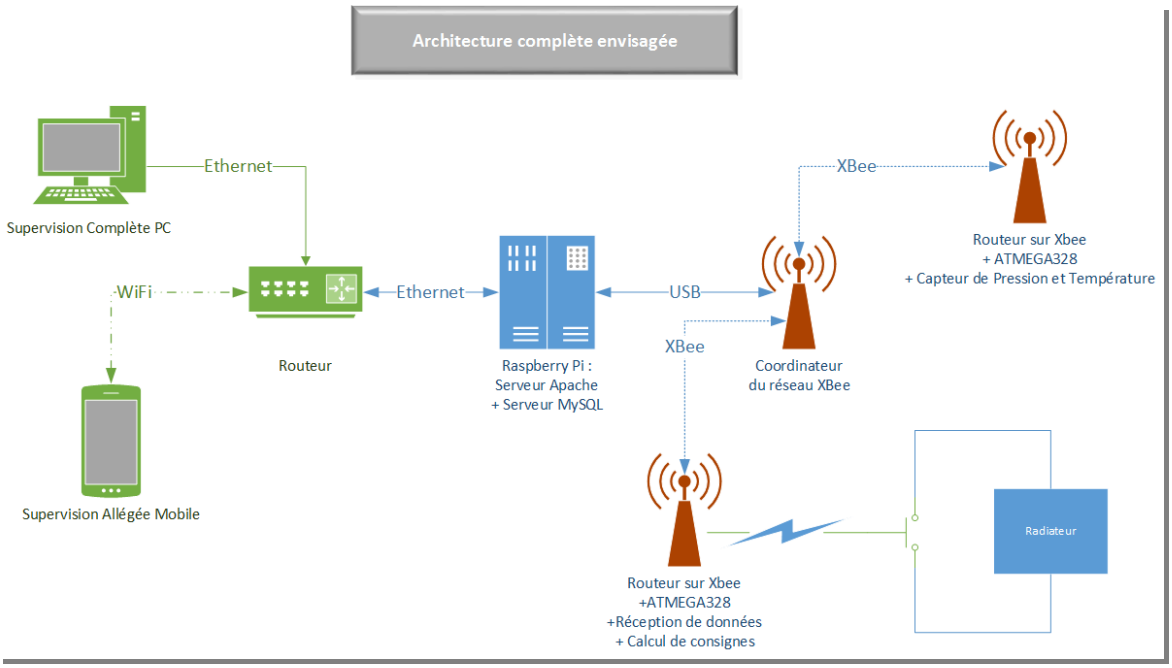


Figure 1 : Système global

En vert, la partie supervision du système grâce à une IHM (interface Homme-Machine) implanté dans la Raspberry Pi. Pourquoi une Raspberry Pi ? Il s'agit d'une petite carte embarquant un système OS linux. Son faible coût pourrait un gros avantage par rapport à une FoxBoard, si le système devait être commercialisé. Concernant l'IHM, celle-ci doit être multiplateforme (Android, iOS, PC, Windows,

linux…). Afin de pallier à ce problème de multiplateforme, nous avons décidé de faire une interface Web. Celle-ci peut même être accessible depuis l'extérieur.

En rouge, la partie réseau de capteur utilisant le protocole ZigBee 801.15. Les modules devront permettre de mesurer une température et actionner le radiateur. De plus, dans un souci de changement des piles ou batteries qui alimentent les modules, ils devront consommer que très peu d'énergie. Enfin, les modules s'installeront dans une maison ou une infrastructure, impossible de prévoir un câblage entre les modules, il faut donc opter pour une architecture réseau Sans-fil. Pourquoi utiliser la technologie Zigbee ?

Norme	Avantages	Inconvénients
Wi-Fi 802.11	▪ Très répandue, on pourrait ainsi utiliser un routeur ou une box dans une maison comme coordinateur ; ▪ Protocole TCP/IP, on pourrait utiliser un serveur DHCP pour faciliter la mise en place du réseau ; ▪ Sécurisé par cryptage (WEP ou WPA-PSK)	▪ le type ad-hoc et répéteur est compliqué à mettre en place ; ▪ Trame de transmission inadaptée au projet. Le Wi-Fi a été principalement développé pour des transferts à grande vitesse de beaucoup de données, or nous souhaitons juste envoyer 2 ou 3 octets à intervalle de temps assez longue ; ▪ Très gourmande en ressource, l'alimentation par pile est inenvisageable avec cette technologie ; ▪ La portée diffère suivant la norme a/b/g/n/ac mais, elle sera tout de même très limitée par rapport à ce que peut atteindre le Zigbee ; ▪ Très grande probabilité interférences avec d'autres appareils, et risque de saturation du réseau si on utilise la bande des 2.4 GHz (limité à 4 ou 5 appareils sur une box par exemple, trop peu).
Zigbee 802.15	▪ Semblable à une communication série, mais sans-fil ; ▪ Trame très simple, nécessite en moyenne que 10% des lignes de codes Wi-Fi ; ▪ Portée pouvant atteindre 1 km avec un appareils tout les 30m ; ▪ Nous allons utiliser des modules XBee Pro qui ne coûtent que 20€ ; ▪ On peut mettre jusque plus de 65 000 appareils en réseau ! ; ▪ Consomme très peu. Avec les mode de réveil du module, on peut descendre à 1mA ;	▪ Vitesse limitée à 250 kbit/s, mais très suffisant dans le cadre de notre projet ; ▪ La portée entre 2 modules est inférieure à 30 m ;

Figure 2 : Tableau comparatif des 2 technologies Wireless

Nous avons divisé le travail en 2 sous-systèmes. Florian l'IHM et la carte de puissance pour le module de radiateur, Jérémy s'est occupé du sous-système réseau.

II) Sous-système IHM Web

A) CAHIER DES CHARGES SPECIFIQUE

La Raspberry Pi utilisée dans le projet permettra de recevoir une base de données MySQL, une interface Web accessible depuis tablette, smartphone et PC, un programme principal permettant la coordination des données. Concernant l'interface Homme-Machine (IHM), nous voyons le projet comme un futur produit commercial, l'IHM doit donc être simple, facile et intuitive.

La première étape de conception consiste à établir une liste des éléments qui devront être présent sur l'IHM. A savoir :

- Un affichage de la température extérieur (facultatif) ;
- La température relevée par les capteurs par pièce ;
- La modification manuelle du mode de confort thermique dans une pièce (Hors-gel, Arret, Economique et confort) ; // chaque mode correspondra à une température
- Un panneau de configuration permettant de :
    - spécifier la ville pour la météo ;
    - rentrer les identifiants de connexion à la base de donnée ;
    - ajouter un nouveau module au réseau ;
- Un programmeur pour spécifier les modes de confort à différents moments de la journée ;

B) PRESENTATION DE L'IHM

1) Page d'accueil

La page d'accueil contient :

- La prévision météorologique à l'heure actuelle ;
- La température relevée lorsqu'on sélectionne une pièce ;
- La possibilité de modifier temporairement le mode de chauffage (temporairement car c'est le programmeur qui a la priorité)

La page d'accueil est codée en PHP HTML en utilisant un add-on « bootstrap » qui rend l'affichage beaucoup plus « sympathique ». Il a fallu un bon moment avant de pouvoir utiliser correctement cet add-on, car il contient beaucoup de fonctions javascript et classes CSS. De plus, le bootstrap permet de dimensionner l'interface en fonction de l'écran, très utile pour passer d'un écran PC à un écran de smartphone, on retrouve par exemple le « collapse » qui permet d'afficher ou masquer certains éléments de la barre header, mais aussi un « *container-fluid* » et « *row-fluid »* pour avoir un système de grille pour redimensionner.

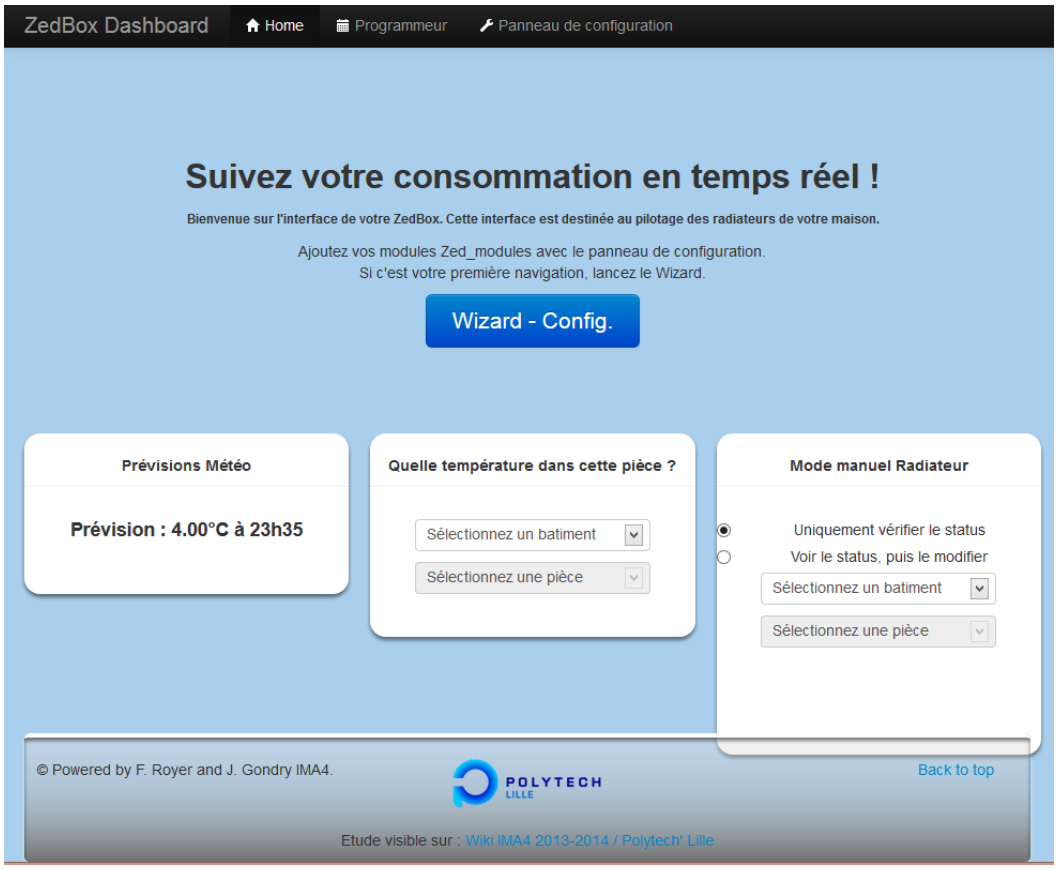


Figure 3 : Vue sur PC ou tablette

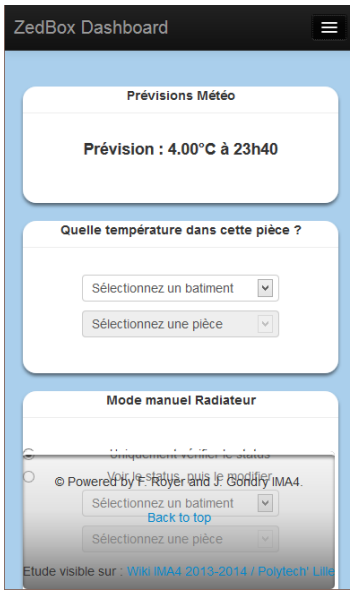


Figure 4 : Vue sur smartphone Le logo en haut à droite permet d'afficher le menu

2) Programmeur

Un programmeur permet de choisir le niveau de confort en fonction de l'heure dans la journée grâce à un code couleur (Hors-gel – Arrêt – Economique (18 °C) et Confort (20°C). Ce programmeur est codé en javascript et agit sur la BdD (voire partie BdD).

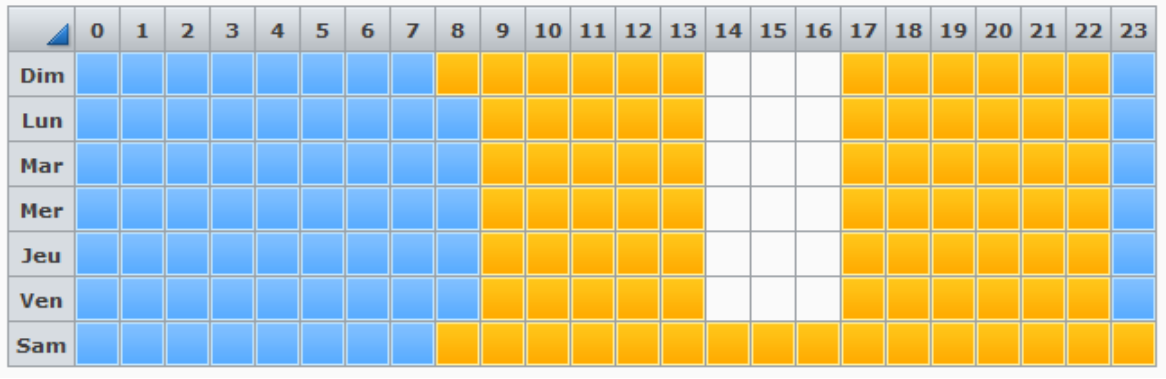


Figure 5 : Programmateur envisagé

3) Panneau de configuration

Ce panneau comporte 3 onglets :

a) Config météo, cet onglet permet de configurer sa localisation météo. Les informations sont récupérées sur le site tameteo.com qui renvoie en caché un identifiant de station météo. Cet identifiant est lors stocké dans un fichier station.conf qui sera utilisé par le programme principal.

Figure 6 : panneau de configuration météo

b) Configuration de la BdD. Cet onglet permet de rentrer les informations qui permettent de se connecter à la base de données. Ces informations sont aussi sauvegardées dans un fichier utilisé aussi par le programme principal.

Figure 7 : Config de la base de données

c) Ajout de modules. Lorsqu'il y a une apparition de nouveaux modulex au sein du réseau ZigBee, le programme principale enregistre les ID des modules dans un fichier. Il devienne visible sur cette page PHP. L'utilisateur rentre alors la pièce et le bâtiment dans lequel sont placés les capteurs ou modules de radiateur.

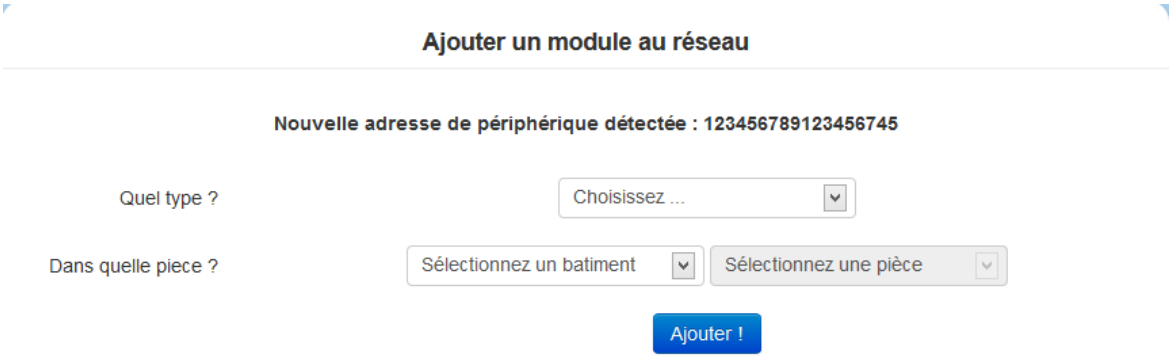


Figure 8 : Paramétrage des nouveaux modules

C) BASE DE DONNEES

Une base de données MySQL est installée sur la Raspberry Pi. C'est le lieu de stockage des multiples informations. Le diagramme UML ci-dessous traduit les différentes tables présentes dans la base.

Figure 9 : UML de la BdD

Attributs de classes ambigües :

- P_mode est le mode qu'a rentré l'utilisateur pour cette pièce (Hors-gel – confort...).
- Net_type correspond au type de module : capteur ou contrôle de radiateur

Nous avons dimensionné la base de données pour qu'elle soit utilisable dans une infrastructure. Il sera en effet possible de brancher une Raspberry dans chaque bâtiment et un coordinateur ZigBee pour chaque bâtiment. Les bâtiments devront cependant être connectés au même réseau ethernet, et il faudra choisir une Raspberry pour qu'elle accueille la base de données. Les ID sont les numéros de

série des ZigBee, sur 8 octets, la probabilité qu'ils soient identiques et donc de 16 puissance 8 (très faible).

D) BIBLIOTHEQUE PARSER.C POUR LE PROG PRINCIPAL

Un programme principal tournera à l'infini sur la Raspberry. Ce programme :

- Lit les infos émises par les modules
- Analyse la trame réseau
- Stock dans la BdD

Nous avons divisé le travail, un binôme s'est chargé du traitement réseau (voire III) pendant que l'autre a développé la bibliothèque parser.c.

Le parser.c contient plusieurs fonctions. Il permet entre autre de récupérer les infos météo et de les placer dans la BdD. Il contient le parser XML pour décomposer le XML météo. (voir code parser.h pour avoir un détail des fonctions).

III) Sous système Réseaux de capteurs

A) DESCRIPTION DU RESEAU :

Souhaitant créer un réseau maillé, notre choix s'est porté sur l'utilisation de Xbee Serie 2. Un réseau Xbee 2 se définit par un channel et un PAN ID (Personal Area Network).

Les Xbee peuvent y jouer 3 types de rôle : Coordinateur, Routeur, et End Device (dispositif en fin de ligne).

Le **Coordinateur** est « le médecin » du réseau, c'est celui qui :

- Attribue les adresses logiques.
- Assaissinit le réseau, lorsqu'un périphérique, il redistribue son adresse logique à un autre périphérique existant et lui attribue une autre adresse lorsqu'il rentre dans le réseau.

Le **Routeur** envoie des informations le concernant et sert de relais pour les End Device qui sont trop éloignés des autres modules pour communiquer avec eux.

Le **End Device,** joue un rôle mineur dans le réseau il envoie ses propres messages router ceux des autres.

B) LES DIFFERENTS COMPOSANT DE CE RESEAU :

Dans ce projet, nous avons différents modules :

1) Modules capteurs

Ces dispositifs sont destinés uniquement à l'envoi de mesures. Ils sont alimentés par batterie donc ils s'endorment périodiquement, pour une question d'autonomie. En s'endormant (dans le mode de sommeil : POWER DOWN, et se réveillant à l'aide d'un Watchdog timer), ils ne peuvent pas assurer le routage des informations ni même recevoir, (ils ont perdu leur adresse logique dans le réseau, les routeurs ne sont pas capable d'acheminer le message jusqu'à ce capteur (cf vérification avec X-CTU, transmit status)). Ils seront donc des **End Devices**.

Composés d' :

- un Atmega328p : il consomme peu, au maximum 2 mA, en comparaison d'un arduino et on consomme des dizaines de microAmpères en veille, avec une fréquence de fonctionnement minimale.
- un Xbee série 2, pour le réseau maillé.
- un régulateur de tension abaisseur à 3,3 v (MAX1684). Un capteur de température numérique (DS18B20) d'une précision de 0,5 degrés, étant donné la grande fluctuation de la grandeur observée (position dans la pièce, courant d'air, ouverture fermeture des portes), sa précision ne doit pas constituer le facteur limitant. C'est la raison pour laquelle notre choix s'est porté sur celui-ci. L'atmega communique avec lui, avec la technologie OneWire, le protocole de communication utilisant un seul fil pour la transmission de donnée. Cette technologie peut être utilisée sur tous les pins digitaux de l'atmega328p, contrairement à l'I2C. Et tout comme l'I2C, il existe un système d'adressage pour communiquer avec différents capteurs.

2) Modules chauffage

Ces derniers attendent des consignes, pour activer ou non un chauffage électrique par exemple. Dans ce cas, le signal de commande correspond au standard fil pilote, et ils nécessitent une puissance importante, ce module doit donc être proche d'une source suffisante, l'alimentation secteur. C'est la raison pour laquelle, étant proche d'une source d'alimentation, ce sont des modules qui ne s'endorment jamais et sont donc des **Routeurs**.

Ce module opère une régulation en hystérésis de la température, à partir de la mesure envoyée par le coordinateur dans la pièce concernée et une consigne, si la température dépasse un certain plafond au-dessus de la consigne, on ne chauffe plus, si la température descend en dessous d'un certain seuil, on chauffe.

Du point de vue automatique, on peut assimilier cette régulation à une boucle fermée.

Ce module chauffage se comporte d'une carte basse tension (la carte du module capteur) reliée à une carte haute tension.

Ici, la carte module capteur ne joue pas un rôle de mesure, le capteur proche de la source de chaleur fausserait les mesures. Il réceptionne grâce au Xbee, les consignes et température qu'il transforme en commande. Cette carte n'est donc pas alimentée par pile mais via la carte haute tension. Nous avons choisi notre régulateur MAX1684 parce qu'il supportait des tensions en entrées jusqu'à 14 volts dans le but d'avoir une certaine marge, pour le design de la régulation haute tension, basse tension.

La carte haute tension contient un étage supérieur de conversion de tension abaisseur Haute tension vers Basse Tension, alternatif vers continu (230 v / 3.3V) et un système de commande : marche, arrêt.

3) La raspberry.

C'est le superviseur du réseau de part son Xbee relié en Coordinateur qui recueille toutes les mesures en provenances des modules capteurs, pour les enregistrer dans la base de donnée, et récupère de celle-ci, les consignes utilisateur. Elle envoie ces deux informations aux modules chauffages qui les interprètent pour établir la commande. Elle est reliée à la puce Xbee par le port série à l'aide d'une platine Xbee.

Ci-contre la platine en question.

C) LA COMMUNICATION DANS LE RESEAU.

Le Xbee Serie 2 impose un choix de le flashage du firmware, AT(et donc mode transparent) ou API, avec choix du rôle. Contrairement au Xbee Serie 1 qui peut être utilisé en AT et en API sans reflashage du firmware. Les API étant un moyen de communication plus bas niveau que le mode transparent pour envoyer des messages par exemple en choisissant le destinataire autre que celui figurant dans la configuration du module Xbee.

Pour ce qui est du choix du channel c'est le module qui choisit le plus optimal, grâce à un scan énergétique. Le Xbee Serie 2 coordinateur scanne chaque channel pour choisir celui qui est optimal du point de vue énergétique, les routeurs et les end-devices rejoindront ce coordinateur si ils ont le même PAN ID.

Chaque puce Xbee configurable dispose d'une adresse logique MY sur 2 octets, et d'une adresse physique sur 8 octets.

Configuration Xbee :

SM : Sleep Mode. Lorsque ce registre passe à 1, il permet de pouvoir endormir le Xbee, ce qui fait passer sa consommation de 80-40 mA à moins de 10µA.

AP (escaped) : Nous configurons ce registre à 2, pour signaler qu'il y aura des caractères d'échappement, (à échapper manuellement coté utilisateur), ces caractères sont des délimiteurs de trames, des signaux d'extinction…

Nous ne configurons donc pas les adresses de destinations, puisque cela se fait lors de l'envoi de message.

Deux réglages très intéressants et illustrant certaines qualités du Xbee Serie 2.

SP : Sleep Period. Ce paramètre indique la période d'endormissement du Xbee. (Maximum 28secondes) Dans le cas d'un End Device et sachant que nous commandons nous-mêmes les cycles d'endormissement du Xbee, l'utilité est ailleurs. Dans le cas d'un **Routeur** ce paramètre indique la durée d'endormissement des End Devices (jouant le rôle de fils de ce routeur), et par extension la durée pendant laquelle le routeur doit garder dans son buffer, les paquets à destination des modules endormis.

SN : Number of Sleep Period. Comme son nom l'indique, ce paramètre indique le nombre de période de sommeil, on peut donc aggrandir la durée de conservation des messages.

Cela permettrait de ne pas avoir de communication en sens unique entre les modules capteurs et le coordinateur du réseau, étant donné qu'un Xbee endormis ne se réveille pas sur réception d'un message. Ainsi, il pourrait récupérer à son réveil les messages lui étant destinés. Cela impose toujours l'obligation d'avoir des routeurs toujours en état de marche.

D) COMMUNICATION EN API

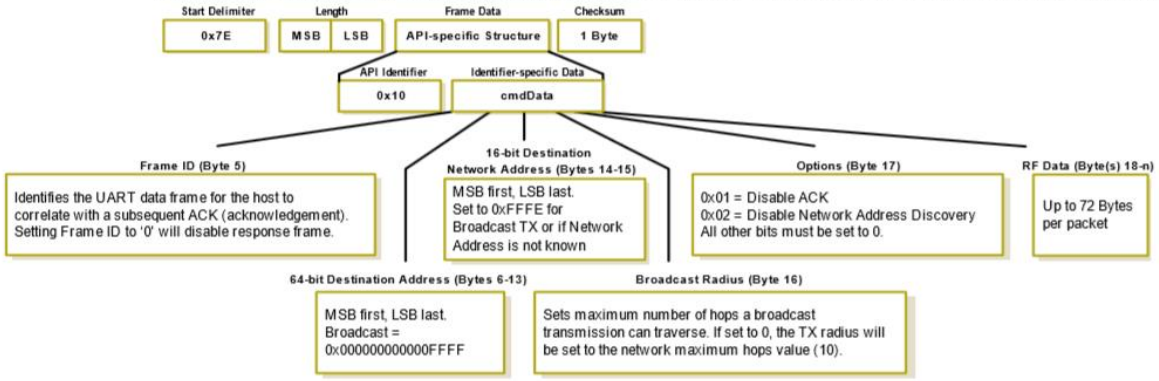
Nous utilisons principalement 2 types de trames API.

La requête de transmission, l'atmega envoie sur la broche RX du Xbee, caractères par caractères les constituants de la trame, il faut échapper les caractères spéciaux sinon le destinataire ne lira pas la trame. (envoyer un 0x7D avant le caractère et envoyer le résultat du OU exclusif entre ce caractère et 0x20). Il faut également calculer l'octet de checksum pour contrôler les éventuelles erreurs de transmission (nous n'en avons jamais eu). Dans cette trame, on précise l'adresse physique du destinataire ou son adresse logique, on précise la taille, les options associés comme l'accusé réception. Ci-dessous l'intégralité de la trame à fabriquer pour envoyer un message.

ZigBee Transmit Request

API Identifier Value: 0x10

A TX Request message will cause the module to send RF Data as an RF Packet.TX Packet Frames



Lorsque le Xbee reçoit un message, il le transmet sous forme de trame sur le port RX de l'atmega.

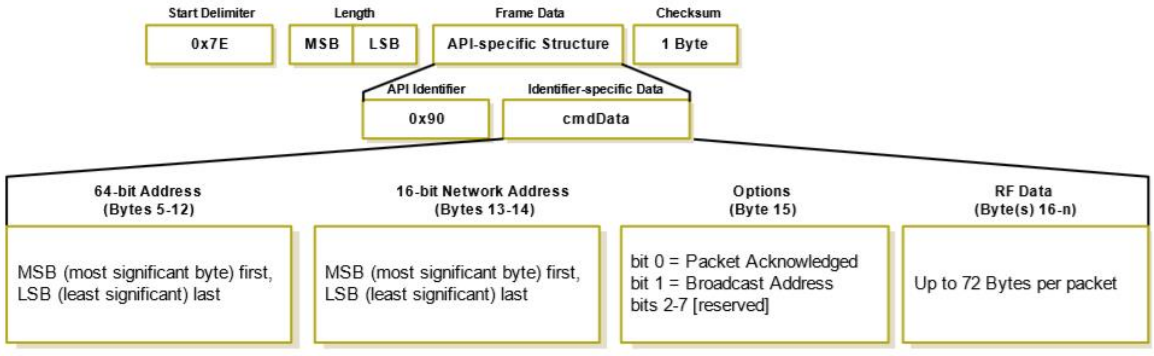
On a donc accès aux adresses logiques et physiques de l'émetteur. Il faut détecter les caractères échappés, précédés d'un 0x7d et faire un OU exclusif entre ce caractère et 0x20.

ZigBee Receive Packet

API Identifier Value: 0x90

When the module receives an RF packet, it is sent out the UART using this message type.

Figure 6-16. RX Packet Frames



E) CONCEPTION DE LA CARTE ULTRA-BASSE TENSION

En ce qui concerne la conception de la carte, nous avons travaillé sur eagle un outil plus léger qu'Altium et gratuit. Nous avons dû réaliser nos propres bibliothèques pour les régulateurs de tensions, le capteur de température. Et nous avons modifié des bibliothèques existantes pour adapter la largeur des pastilles.

Pour la carte module capteur, nous avons sur le top de la carte, tous nos composants traversant ainsi, qu'un composant de surface, le régulateur de tension, et c'est sur le bottom que tout est routé. Le plan de masse est sur le top. Il a fallu reprendre la carte à plusieurs reprises pour refaire certaines connexions.

Une résistance de pull-up interne sur le pin d'interruption externe ne fonctionnait pas de manière optimale et ne recevait pas correctement le bon signal d'interruption, envoyé par un bouton poussoir. Nous avons utilisé une pull externe, et en redirigé une voie avec un strap. Nous avons également utilisé des straps pour relier à la masse les compoants dont les pattes étaient trop courtes, ou router des voies pour ne pas avoir à le faire sur le top. (Ce qui est gênant en termes de design pour la carte, les surfaces dénués de cuivre sur le plan de masse sont sujettes à des champs électriques, ce qui peut provoquer du bruit indésirable). Cette exigence a rendu le routage difficile, il a fallu constamment changer l'organisation des composants sur la carte. Visualiser la carte est très important, l'option de routage automatique d'Eagle est très rapidement dépassée.

(voir schematic page suivante).

Ci-dessous le Top, de la carte avec le plan de masse.

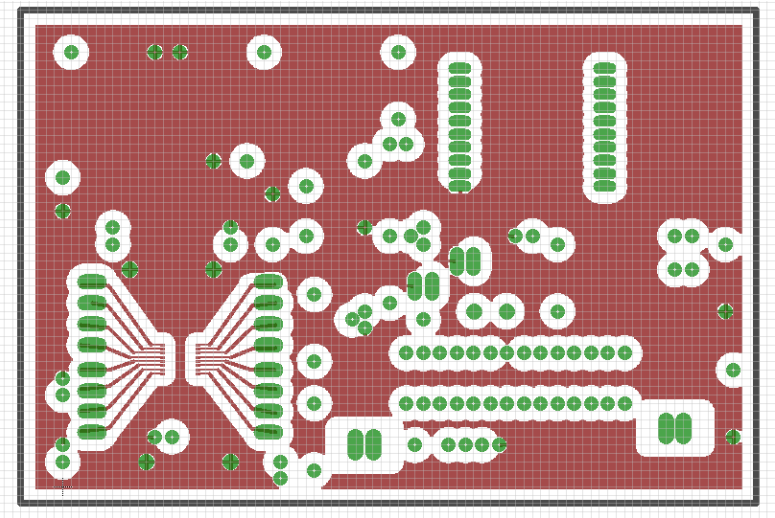


Figure 10 : Top de la carte

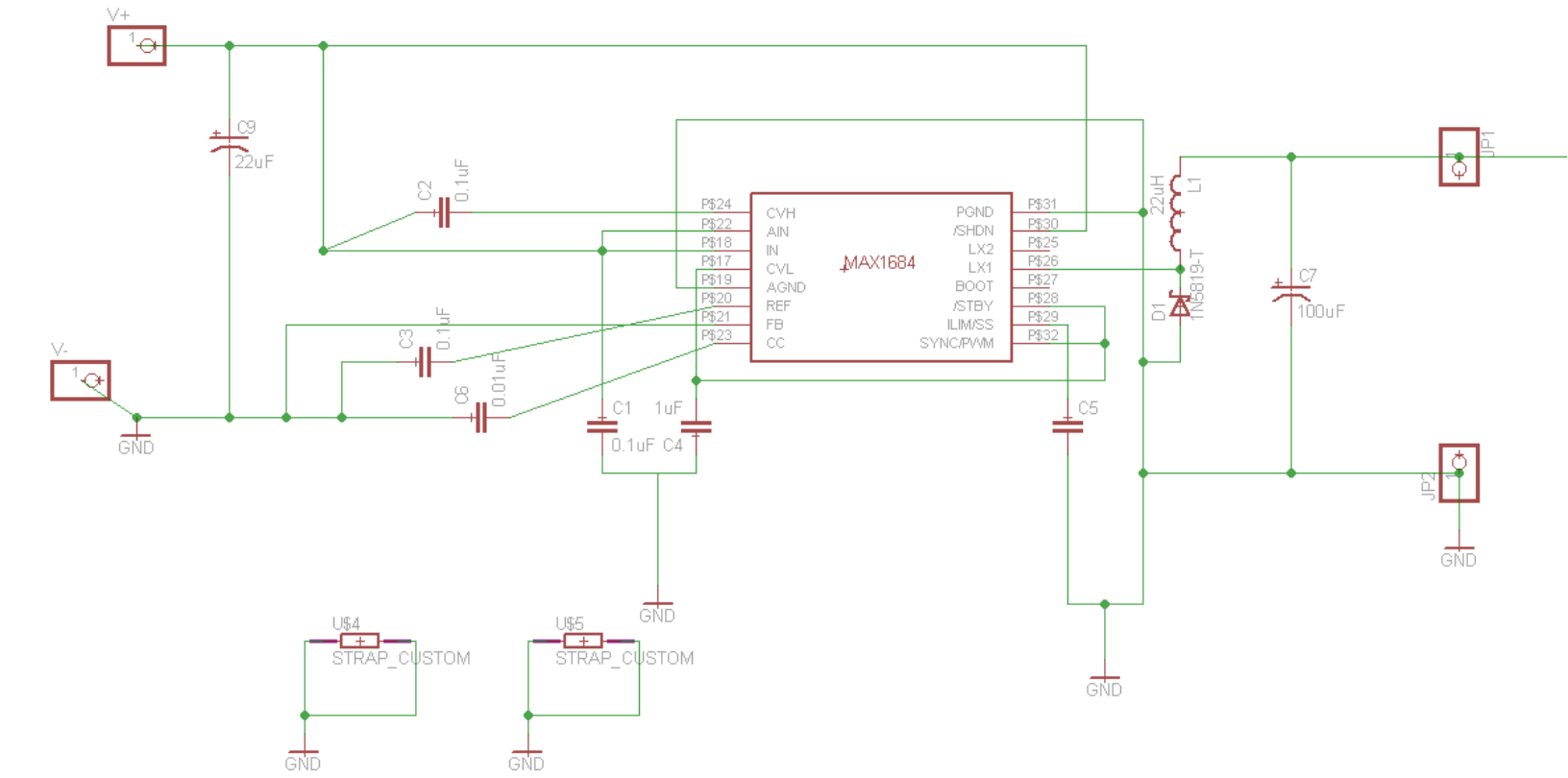


Figure 12: Partie Régulateur

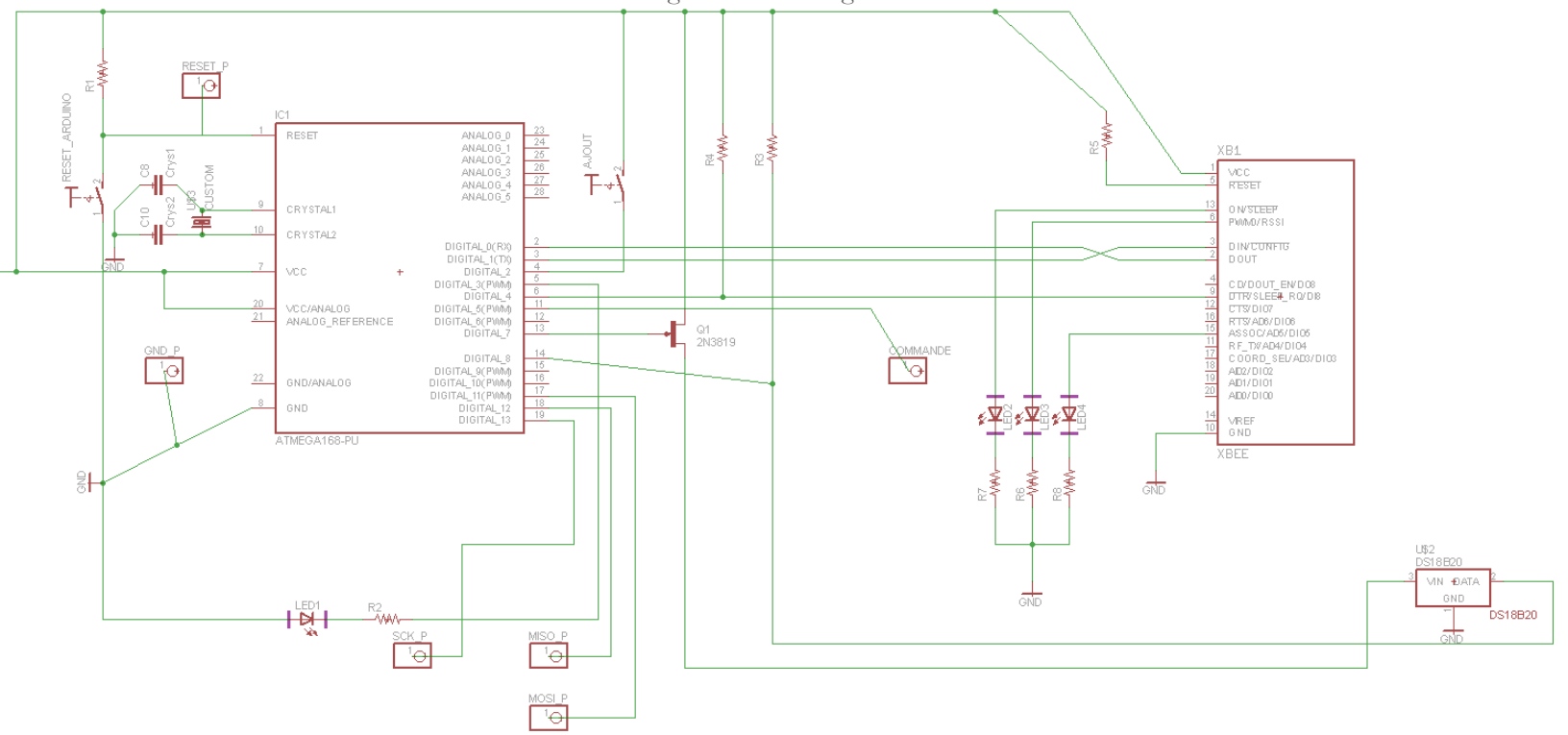


Figure 11 bis : Partie µControleur

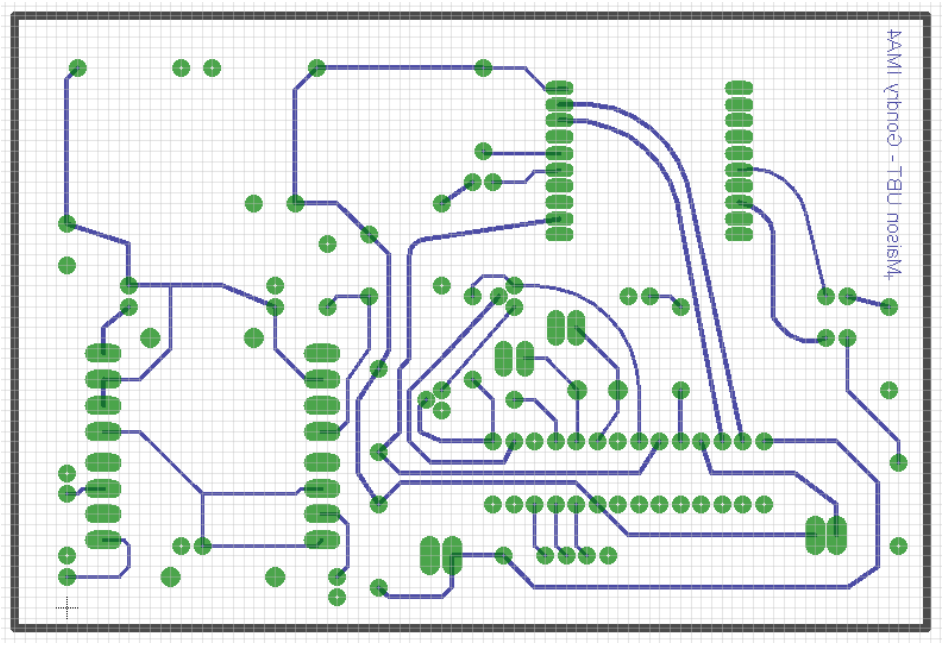


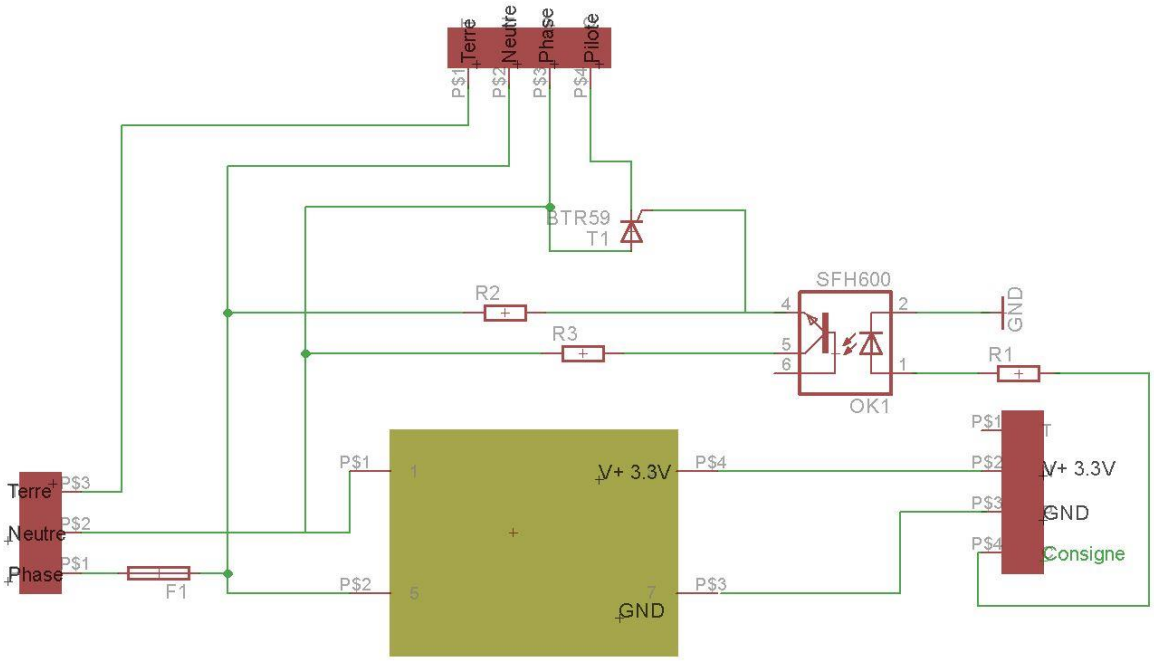
Figure 13 : Bottom de la carte

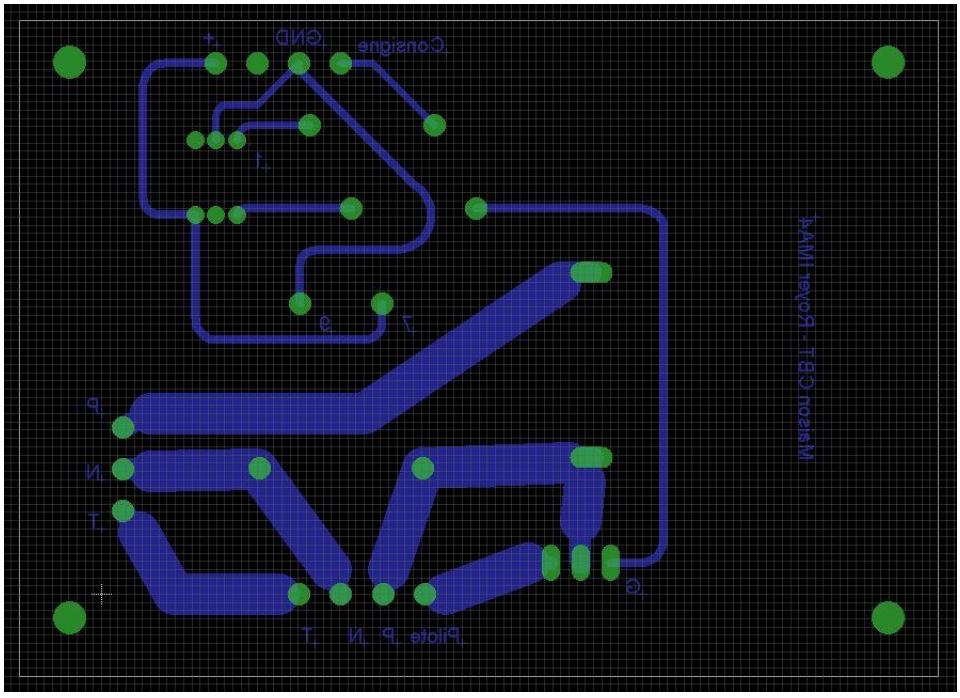
Figure 14 : carte finale

F) CONCEPTION DE LA CARTE BASSE TENSION

La carte basse tension se branche entre la source (prise secteur et le radiateur). Il y a une entrée (Phase, Neutre, Terre, Consigne) et plusieurs sorties (V_{DC} 3.3V , masse, Phase, Neutre, Terre, File Pilote).V_{DC} et la masse permettent d'alimenter la carte ultra basse tension. Les sorties phase, neutre et Terre, se branchent sur le radiateur directement. La consigne est une entrée, il s'agit d'un pin de sortie du µControleur qui définira la marche ou l'arrêt du radiateur.

Voici le schematic de la carte basse tension :

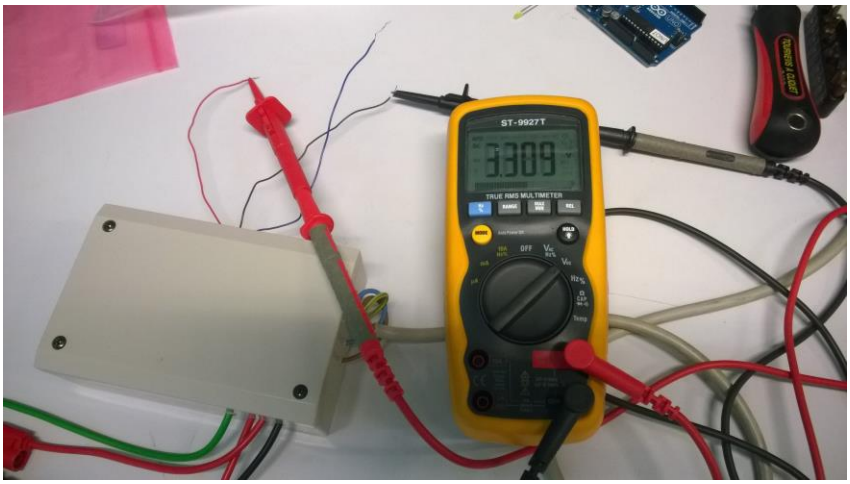




Principe de fonctionnement, la carte reçoit une commande de la part de la carte ultra basse tension, qui agit sur un optocoupleur (une led rayonnant sur un phototriac), permettant la commande d'un triac de puissance. L'optocoupleur permet d'isoler la partie basse tension de la partie haute tension. Le triac de puissance permet de laisser passer une tension pleine alternance (signal sur le fil pilote

alternatif ou nul). D'après la datasheet du radiateur, le signal du fil pilote doit être nul pour que le radiateur soit en marche et alternatif pour qu'il soit en marche.

Test effectué en sortie V_{DC} :



G) PROBLEMES RENCONTRES :

1) La programmation de l'atmega.

Nous voulions, pour optimiser l'autonomie, abaisser la fréquence de fonctionnement de l'atmega328p jusqu'à un minimum de 1MHz, en effet nous utilisons le protocole de communicatiosn OneWire (un seul fil pour la transmission de donnée, pas de fil pour l'horloge) pour communiquer avec notre capteur de température, celui-ci nécessite des timings de communication de l'ordre de la microseconde.

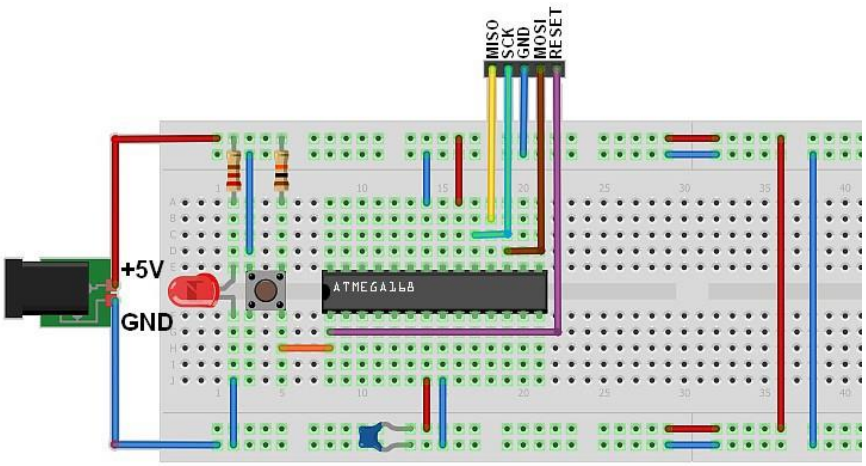
Vers la fin du projet, en essayant de reprogrammer les registres de diviseur de fréquences, nous nous sommes rendu compte que c'était impossible dans le corps d'un programme.

En effet, les registres liés à la source de l'horloge (CKSEL, CLKPR, etc) font partie des « fuses » (fusibles) on ne peut pas écrire sur ces derniers à l'aide d'instructions Arduino/C/C++. Cette opération s'effectue via un programmateur AVR ISP mkll par exemple.

Ci-contre le programmateur en question.

Il utilise un bus de 6 broches pour communiquer avec l'atmega.

Nous avons décidé de réaliser la programmation de l'atmega sur Breadboard. Nous avons procédé au montage ci-dessous :



Ce montage n'a pas donné de résultats probants. Après de multiples tentatives, à l'aide du programmateur, et l'outil AVR-Bur-O-Mat lancé sur le PC. Ce dernier indiquant une lecture des fusibles impossibles. Celui-ci utilisant des drivers généralistes que d'autres utilitaires peuvent utiliser.

Un test supplémentaire a écarté l'hypothèse d'un montage incorrect, en essayant de reprogrammer directement un arduino sur ses broches ISP, le même problème est apparu. Le problème était ailleurs.

Nous avons décidé d'utiliser AVR Studio qui utilise ses propres drivers pour la programmation ISP avec les produits AVR .

AVR Studio a donné des résultats complémentaires, un message d'erreur ne correspondant pas à ceux trouvé sur la FAQ du site d'AVR indique un problème au niveau de la liaison USB.

Pour proposer rapidement un PCB en vue de la réalisation d'une carte, nous avons donc fait une concession en incluant sur la carte un crystal cadencé à 16MHz (quasiment le même que ceux utilisés sur les Arduino Uno) et les condensateurs associés. Libre à nous de les souder ou non pour la phase finale. L'atmega serait nu mais tournera à plus grande fréquence avec une consommation plus élevé qu'à l'aide d'une clock interne c'est-à-dire qu'il consomme 0,8 mA au lieu des 8µA espérés (seulement lorsqu'il sera hors veille). Nous uploadons le programme sur un Arduino première génération (l'atmega ayant une empreinte P-DIP). Et nous empruntons cet atmega pour la carte. En attendant, nous avons ajouté sur la carte des connecteurs reliés aux broches digitales de l'atmega concernant l'ISP (broches 11, 12, 13), ces connecteurs serviront de également pour l'ajout d'autre capteurs numériques, (de pression par exemple)

A la fin du projet, nous avons trouvé d'autres pistes pour envisager la reprogrammation, au sujet des drivers se pose la question des signatures numériques, nos tests ne ce sont effectués qu'avec des machines à 64 bits.

Solution éventuelle : essayer AVR studio sur une machine 32 bits.

Nous avons également essayé la programmation en utilisant un Arduino Uno en tant que programmateur ISP, en aprennant par la suite que cette possibilité s'offre plutôt aux Arduino du type Duemilanove. Solution éventuelle : se procurer un de ces modèles pour essayer le montage suivant.

Pour flasher le bootloader : relier les pins MISO, MOSI, SCK (ISP) de cette Arduino relié au PC et les connecter à ceux de l'atmega, dans l'IDE Arduino téléverser le programme en choisissant l'option «Téléverser avec un programmateur », (en ayant choisi le type de programmateur « Arduino as

ISP » et le type de carte « Arduino to Breadboard»). Pour uploader un programme utiliser uniquement les ports séries de l'Arduino en retirant le micro-contrôleur de celui-ci. Ce qui impose d'avoir un modèle disposant d'un microcontrôleur P-DIP.

Certains changements au niveau des fusibles sont possibles dans un Makefile. En transposant le programme en C pur, de meilleurs résultats peuvent être obtenus. Cela constiturait une autre solution.

2) Le capteur de température :

Lorsque le dispositif est alimenté de manière parasite (alimenté par le port de donnée), il donne des valeurs de température faussées.

Nous avons pris le problème d'un angle différent, c'est-à-dire en alimentant le capteur sur son port d'alimentation à l'aide d'un transistor JFET (commandé par un pin digital de l'atmega), un certain temps avant de prendre la mesure. Pendant ce temps le capteur consomme 1 mA. Les résultats étaient satisfaisants du point de vue de la mesure, mais la carte perd de l'autonomie puisque la consommation du capteur est bien plus prolongée que celle du Xbee.

3) Lecture série

Au niveau de la lecture sur le port série entre la Raspberry et le Xbee, il était parfois impossible, d'avoir accés au port, par exemple après débogage avec minicom, en effet, dès qu'il y avait déconnexion d'un utilisateur du Xbee, le programme C, Minicom, le Xbee s'éteignait.

Après de multiples recherches, la cause a été trouvé, lorsque un utilitaire se déconnecte du port série (modem), il y a abaissement d'un signal bas niveau, le signal DTR (Data Terminal Ready), coupant la connexion, et seul minicom était à priori capable de redémarrer la connexion avec le port série.

Dans le programme C, en décommentant au fur et mesure le programme d'initialisation du port série, nous avons remarqué que le signal DTR était abaissé au moment d'appliquer la configuration (avec la structure de donnée termios sur linux). Il a fallu dans le programme C, remonter manuellement ce signal.

Dans la fonction de configuration du port série

tcsetattr(fd,TCSANOW,&new); //application de la configuration

//fd : le descripteur de fichier lecture, écriture pour le port série.

//new : contient la configuration du port série (Baud Rate, droits, etc…)

//TSCANOW, flag indiquant que l'application de cette configuration se fait tout de suite

A ce moment, il y a abaissement du signal, le port série va se fermer.

int status;

ioctl(fd,TIOCMGET,&status);

//recupération des signaux bas niveau du port série

status |=TIOCM_DTR; //empêcher le passage du DTR à 0

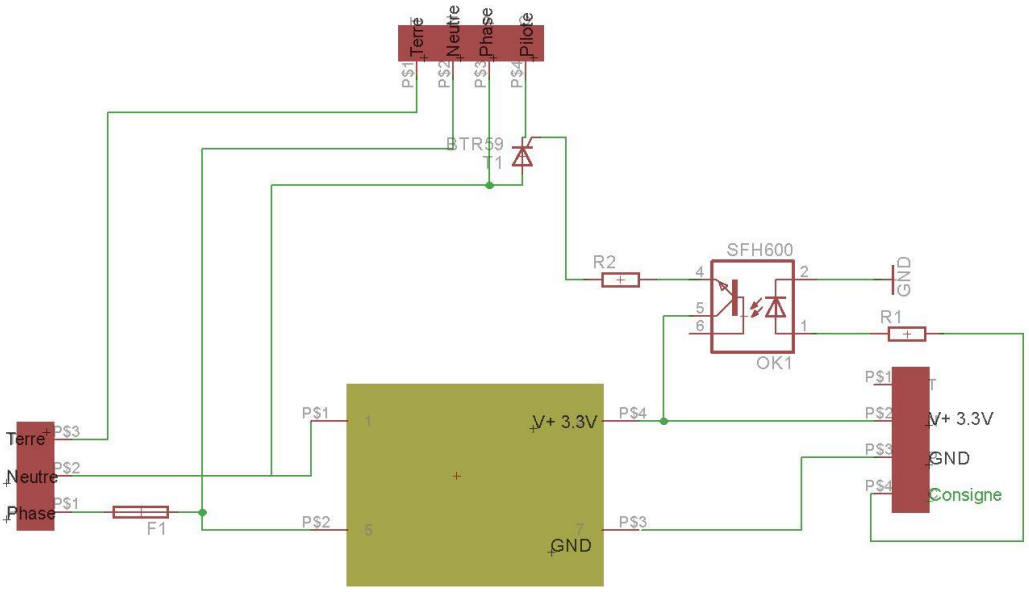
ioctl(fd,TIOCMSET,&status);

//application de ces signaux.

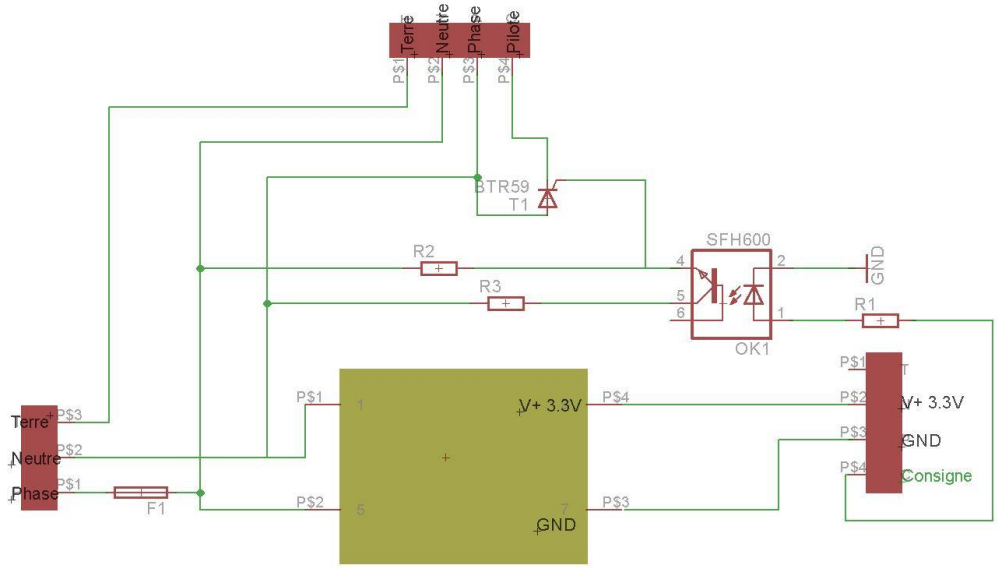
Cette solution présente l'avantage de n'utiliser qu'un seul programme pour utiliser un xbee connecté au port série. Dans un projet de l'année précédente, il fallait utiliser un programme C, pour maintenir la connexion sur le Port Série, et un autre programme pour communiquer avec ce port série

4) Problème Carte basse tension

Il se trouve que nous étions partis à la base sur ce type de conception :



L'alimentation étant un transformateur, la masse est « virtuelle » et ne corrsepond pas au point 0 entre la phase et le neutre. Du coup, la commande sur la gâche du triac de puissance n'a pas de masse à proprement parlé, le courant ne fuite pas. Il faut donc envisager cette correction qui par manque de temps n'a pas pu être appliquée :



Conclusion

Nous avons réussi à proposer une solution 100% autonome, capable d'alimenter une base de données et d'interagir avec. Cette solution s'inscrit clairement dans l'ère du développement durable et des économies d'énergies. Ce projet nous a permis d'aborder le côté logiciel (programmation du microcontrôleur, du programme principal sur la RPI, de l'IHM et BdD) et le côté matériel par la création de 2 cartes (une ultra basse tension et une basse tension : les 2 cartes ne répondant pas aux mêmes normes de sécurité).

Bibliographie

Building Wireless Sensor Networks *by Robert Faludi*

Datasheet des différents composants utilisés.

Idle Blog : http://blog.idleman.fr/raspberry-pi-15-creer-sa-propre-prise-radio-et-autres-peripheriques-pour-6e/